DATS 6103 – Intro to Datamining
Individual Final Report
Group-7
May 3, 2021
Pranay Bhakthula

## 1.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 after training 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 current job for HR researches too.

By model(s) will use the current credentials, demographics, experience data to 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.

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 powerpoint presentation, writing the group report, and creating a demo of the GUI.

## 2.Personal Contribution:

Code:

Pre-processing:

1. dropped 'enrollee_id' column as it was not relevant to the prediction.
2. Filled the missing values with the most repeated term in that column since most of the variables having missing values are categorical data.
3. Label encoding was used on X-variables since most of the variables are categorical data.
4. Target variable was Label encoded as it is a categorical type.
5. Split the data set into train and test dataset.

Eda analysis:

1. Built the Histogram for various variables – the algorithm was designed to plot histogram of any variable selected by the user.

Model Building:

1. Built the 2 models (Gini and entropy) for Decision Tree Model.

**Decision Tree Classifier**

Decision Tree is a Supervised learning technique that can be used for both classification and Regression

problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

**Entropy**: It is defined as a measure of impurity present in the data. The entropy is almost zero when the sample attains homogeneity but is one when it is equally divided. Entropy with the lowest value makes a model better in terms of prediction as it segregates the classes better. Entropy is calculated based on the following formula:

$$Entropy = -p_1 * log_2\, p_1 - \ \ldots - -p_n * log_2\, p_n$$

Here n is the number of classes. Entropy tends to be maximum in the middle with value up to 1 and minimum at the ends with value up to 0.

**Gini**: It is a measure of misclassification and is used when the data contain multi class labels. Gini is similar to entropy but it calculates much quicker than entropy. Algorithms like CART (Classification and Regression Tree) use Gini as an impurity parameter.

Gini is calculated as:

$$Gini = 1 - \sum_i p_i^2$$ where i is the no of classes.

2. 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.

Pyqt5:

1. Worked on Pyqt5 coding part of Decision Tree which includes feature selections, display of results from each models which includes confusion matrix, classification report, accuracy, ROC curve Area under the Curve and Importance of features plot. It was developed to give user the choice of selection of X-variables to build model with and to display the results of the decision tree model. The popup warning of error also exists if none features are selected to build the model.
2. Worked on pyqt5 coding part of histogram which includes feature selections and display of the graph. It was developed to give user the choice of selection of X-variables to display the results of

the Histogram. The popup warning of error also exists if more than one features are selected to plot.

3. Worked on upload section of GUI, which includes the uploading the dataset when path of the dataset is given and display of variables of dataset when it is uploaded. The popup warning of error also exists wrong path is given of the dataset.
4. Worked on error pop up window at various error areas.
5. Helped in merging the different sections of the code.

Presentation:

1. Recorded the Demo video of the working of the GUI of our project.

Group Final Report:

1. Worked on Results of machine learning models, summary and conclusion part of Group Final report.

**3.Personal Contribution in detail:**

Pre-processing:

dropped ‘enrollee_id’ column as it was not relevant to the prediction

```
data.drop(["enrollee_id"], axis=1, inplace=True)
```

Filled the missing values with the most repeated term in that column since most of the variables having missing values are categorical data

```
data = data.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

Label encoding  was used on X-variables since most of the variables are 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
```

Target variable was Label encoded as it is a categorical type

```
y = data.iloc[:, -1]

# label encoding the target
class_le2 = LabelEncoder()
y = class_le2.fit_transform(y)
```

Split the data set into train and test dataset

```
# 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)
```

Eda analysis:

Built the Histogram for various variables – the algorithm was designed to plot histogram of any variable selected by the user

```
self.ax1.clear()
self.current_features.value_counts().plot(kind='bar', ax=self.ax1)
self.ax1.set_title('Histogram of : ' + x_a)
self.ax1.set_xlabel(x_a)
self.ax1.set_ylabel('frequency')
self.fig1.tight_layout()
self.fig1.canvas.draw_idle()
```

Model Building:

Built the 2 models (Gini and entropy) for Decision Tree Model.

Gini model:

```
# specify random forest classifier
self.clf_df_gini = DecisionTreeClassifier(criterion="gini", random_state=100,
max_depth=vmax_depth)

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

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

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

Entropy model:

```
# specify random forest classifier
self.clf_df_entropy = DecisionTreeClassifier(criterion="entropy",
random_state=100, max_depth=vmax_depth)

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

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

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.

Gini model:

```
# 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_df_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)
```

Entropy model:

```
# 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_df_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)
```

PYQT5 part of the code:

The code for main decision Tree class:

```
class DecisionTree(QMainWindow):
    #::-----------------------------------------------------------------------
---------
    # Implementation of Decision Tree 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 base on the
parametes
    #               chosen by the user
    #::-----------------------------------------------------------------------
----------
    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 layout 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('Decision Tree 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.lblMaxDepth = QLabel('Maximun Depth :')
        self.txtMaxDepth = QLineEdit(self)
        self.txtMaxDepth.setText("3")

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

        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.lblMaxDepth, 8, 0)
        self.groupBox1Layout.addWidget(self.txtMaxDepth, 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.groupBox1Layout.addWidget(self.btnDTFigure, 10, 1)

        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):
        # processing the parameters
        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):
        '''
        Decision Tree Classifier
        We pppulate 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
        :return:None
        '''


        vtest_per = float(self.txtPercentTest.text())
        vmax_depth = float(self.txtMaxDepth.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
```

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

        # 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]

        # label encoding the target
        class_le2 = LabelEncoder()
        y = class_le2.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_df_gini = DecisionTreeClassifier(criterion="gini",
random_state=100, max_depth=vmax_depth)

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

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


        # predicton on test using all features
        y_pred_gini = self.clf_df_gini.predict(X_test)
        y_pred_score_gini = self.clf_df_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_df_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_df_entropy = DecisionTreeClassifier(criterion="entropy",
random_state=100, max_depth=vmax_depth)

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

        # predicton on test using all features
        y_pred_entropy = self.clf_df_entropy.predict(X_test)
        y_pred_score_entropy = self.clf_df_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_df_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)

        #::-------------------------------------
        ##  Graph2 :
        ##  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()

    def view_tree(self):
        '''
        Executes the graphviz to create a tree view of the information
         then it presents the graphic in a pdf format using webbrowser
        :return:None
        '''

        # produces pdf results of gini model
        dot_data1 = export_graphviz(self.clf_df_gini, filled=True,
rounded=True, class_names=class_names,
```

```
feature_names=self.current_features.columns, out_file=None)

        graph = graph_from_dot_data(dot_data1)
        graph.write_pdf("decision_tree_gini.pdf")
        webbrowser.open_new(r'decision_tree_gini.pdf')

        # produces pdf results of entropy model
        dot_data2 = export_graphviz(self.clf_df_entropy, filled=True,
rounded=True, class_names=class_names,

feature_names=self.current_features.columns, out_file=None)

        graph = graph_from_dot_data(dot_data2)
        graph.write_pdf("decision_tree_entropy.pdf")
        webbrowser.open_new(r'decision_tree_entropy.pdf')

class Plotter(FigureCanvas):
    #::-------------------------------------------------------------
    # creates a figure on the canvas
    # later on this element will be used to draw a ROC curve
    #::-------------------------------------------------------------
    def __init__(self, parent=None, width=7, height=6, dpi=100):
        self.fig = Figure(figsize=(width, height), dpi=dpi)

        FigureCanvas.__init__(self, self.fig)
        self.setParent(parent)

        FigureCanvas.setSizePolicy(self,
                                   QSizePolicy.Expanding,
                                   QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)

    def plot(self):
        self.ax = self.figure.add_subplot(111)
#-------------------------------------------------------------
# Class to plot Importance of features
#-------------------------------------------------------------
class Imp_Main(QMainWindow):
    #::---------------------------------
    # Creates a canvas containing the plot for the ROC curve
    # ;;---------------------------------
    def __init__(self, parent=None):
        super(Imp_Main, self).__init__(parent)

        self.left = 100
        self.top = 100
        self.Title = 'Importance Bar plot'
        self.width = 1100
        self.height = 850
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.Title)
        self.setStyleSheet(font_size_window)

        self.setGeometry(self.left, self.top, self.width, self.height)

        self.imp = Plotter(self, width=11, height=8.5)
#-------------------------------------------------------------
# Class to plot ROC curves
#-------------------------------------------------------------
```

```
class ROC_Main(QMainWindow):
    #::--------------------------------
    # Creates a canvas containing the plot for the ROC curve
    # ;;--------------------------------
    def __init__(self, parent=None):
        super(ROC_Main, self).__init__(parent)

        self.left = 250
        self.top = 250
        self.Title = 'ROC curve'
        self.width = 700
        self.height = 600
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.Title)
        self.setStyleSheet(font_size_window)

        self.setGeometry(self.left, self.top, self.width, self.height)

        self.roc = Plotter(self, width=7, height=6)
```

pyqt5 coding part of histogram which includes feature selections and display of the graph:

```
#---------------------------------
# Class to display the histogram window
#---------------------------------
class Histogram_plots(QMainWindow):
    send_fig = pyqtSignal(str)

    def __init__(self):
        super(Histogram_plots, self).__init__()
        self.Title = "Histograms"
        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 One of the Features')
        self.groupBox1Layout = QGridLayout()
        self.groupBox1.setLayout(self.groupBox1Layout)

        self.feature = []

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

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

        self.btnExecute = QPushButton("Plot")

        self.btnExecute.clicked.connect(self.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.feature[12], 6, 0)
        self.groupBox1Layout.addWidget(self.btnExecute, 7, 1)

        self.fig1, self.ax1 = plt.subplots()
        self.axes = [self.ax1]
        self.canvas1 = FigureCanvas(self.fig1)

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

        self.canvas1.updateGeometry()

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

        self.groupBoxG1Layout.addWidget(self.canvas1)

        self.layout.addWidget(self.groupBox1, 0, 0)
        self.layout.addWidget(self.groupBoxG1, 0, 1)

        self.setCentralWidget(self.main_widget)
        self.resize(1200, 900)
        self.show()

    def Message(self):
        QMessageBox.about(self, "Warning", " You can't exceed more than 1
feature")

    def update(self):
        self.current_features = pd.DataFrame([])
        x_a = ''
        work = 0
        for i in range(13):
            if self.feature[i].isChecked():
                if len(self.current_features) > 1:
                    self.Message()
                    work = 1
                    break

                elif len(self.current_features) == 0:
                    self.current_features = data[features_list_hist[i]]
                    x_a = features_list_hist[i]
                    work=0

        if work == 0:
            self.ax1.clear()
            self.current_features.value_counts().plot(kind='bar', ax=self.ax1)
            self.ax1.set_title('Histogram of : ' + x_a)
            self.ax1.set_xlabel(x_a)
            self.ax1.set_ylabel('frequency')
```

```
            self.fig1.tight_layout()
            self.fig1.canvas.draw_idle()
```

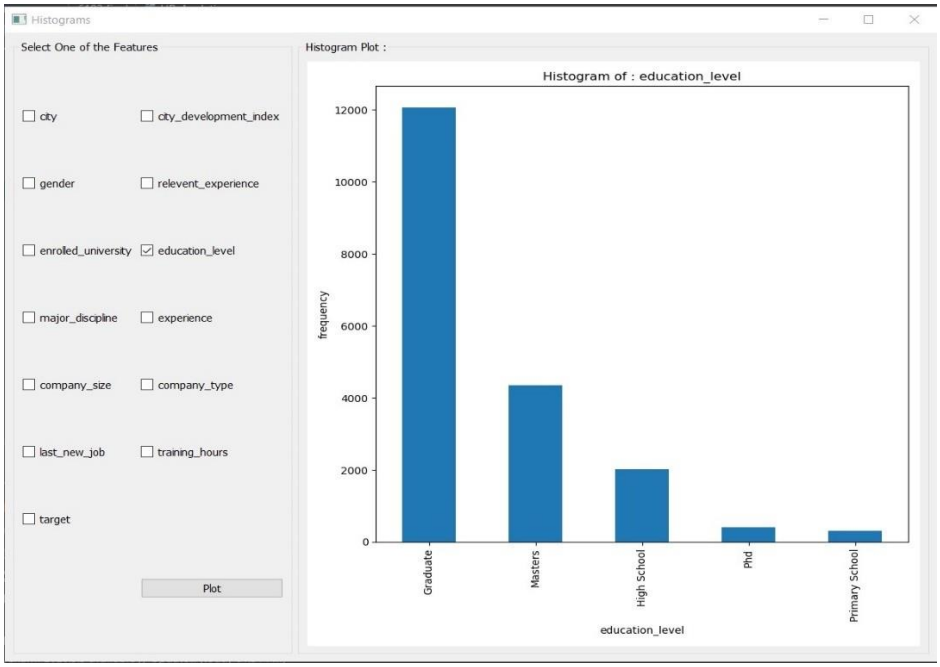
Upload section of GUI, which includes the uploading the dataset when path of the dataset is given and display of variables of dataset when it is uploaded. The popup warning of error also exists wrong path is given of the dataset:

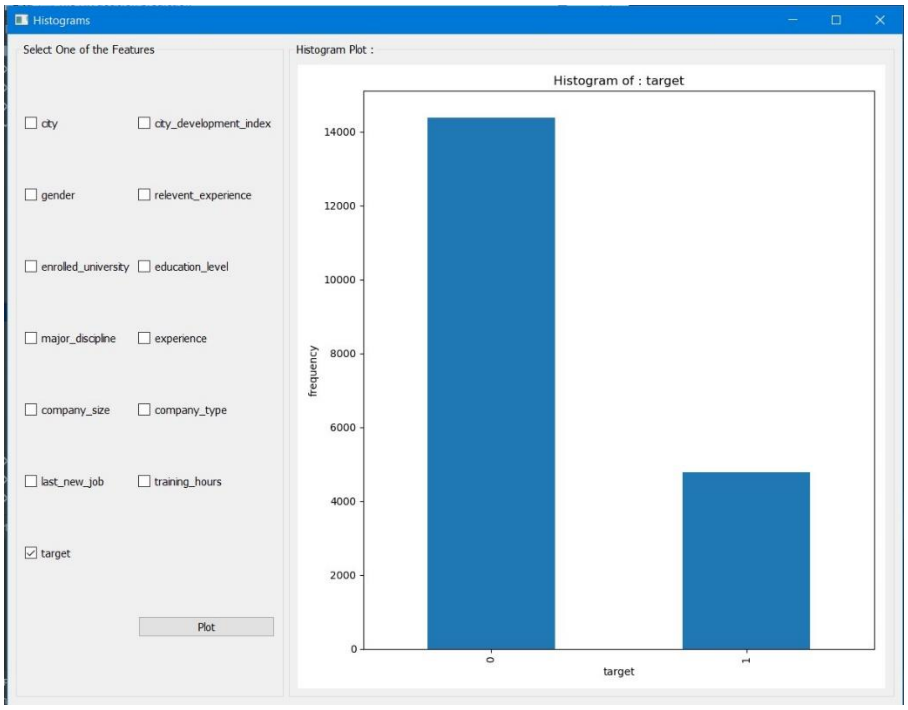
```
class Data_find(QMainWindow):

    send_fig = pyqtSignal(str)

    def __init__(self):
        super(Data_find, self).__init__()
        self.Title = "Select Dataset"
        self.initUi()

    def initUi(self):
        self.setWindowTitle(self.Title)
        self.setStyleSheet(font_size_window)

        self.main_widget = QWidget(self)

        self.layout = QVBoxLayout(self.main_widget)

        # Add a group to upload dataset
        self.groupBox1 = QGroupBox('Upload the dataset')
        self.groupBox1Layout = QGridLayout()
        self.groupBox1.setLayout(self.groupBox1Layout)

        self.lblPath = QLabel('Paste your datset path :')
        self.lblPath.adjustSize()

        self.txtPath = QLineEdit(self)
        self.txtPath.setText("HR_Analytics.csv")

        self.btnUpload = QPushButton("Upload")
        self.btnUpload.clicked.connect(self.update)

        self.groupBox1Layout.addWidget(self.lblPath, 0, 0)
        self.groupBox1Layout.addWidget(self.txtPath, 0, 1)
        self.groupBox1Layout.addWidget(self.btnUpload, 1, 0)

        self.groupBox2 = QGroupBox('List of Features')
        self.groupBox2Layout = QVBoxLayout()
        self.groupBox2.setLayout(self.groupBox2Layout)

        self.lblResults1 = QLabel('Features :')
        self.lblResults1.adjustSize()
        self.txtResults1 = QPlainTextEdit()

        self.groupBox2Layout.addWidget(self.lblResults1)
        self.groupBox2Layout.addWidget(self.txtResults1)

        self.layout.addWidget(self.groupBox1)
        self.layout.addWidget(self.groupBox2)

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

```
    def Message(self):
        QMessageBox.about(self, "Warning", "given path does not exist")

    def update(self):

        path1 = self.txtPath.text()
        if os.path.isfile(path1):
            global data
            global features_list
            global class_names
            global features_list_hist
            data = pd.read_csv(path1)

            data.drop(["enrollee_id"], axis=1, inplace=True)

            data = data.apply(lambda x: x.fillna(x.value_counts().index[0]))

            features_list = data.iloc[:, :-1].columns
            features_list_hist = data.columns
            class_names = ['nojob change', 'job change']
            self.list1 = 'These are the list of features in the dataset :'
            for name in data.columns:
                self.list1 += '\n' + name + '\n'

            self.txtResults1.appendPlainText(self.list1)
            global upload1
            upload1=1
        else:
            self.Message()
```

## 4.RESULTS:

**Histogram**: The image describes the pictorial representation of histogram and the various features provided on the left grid and after pushing the plot button the histogram gets displayed on the canvas.




The above histogram of target variable shows us that there are more observations of 0's than 1's in the 'target' variable. This may lead to false negatives while building of models.

## MODEL BUILDING –

## FIRST MODEL:

First, we select all the X-variables and build two decision tree models, using criterion ‘gini’ and ‘entropy’ seperatley. We calculate the accuracy, classification report and area under the curve.

We find the confusion matrix and plot it as a heatmap, plot the ROC curve, plot the importance of features, for both the models.

We obtain following results:

Decision Tree Classifier: The image displays the decision tree dashboard and the user can manually change the percentage for test and also the maximum depth, The features can be selected as per the user’s choice and execute the decision tree.The plot roc button displays the ROC graph and view tree button provides two graphs one for entropy and gini in pdf format.

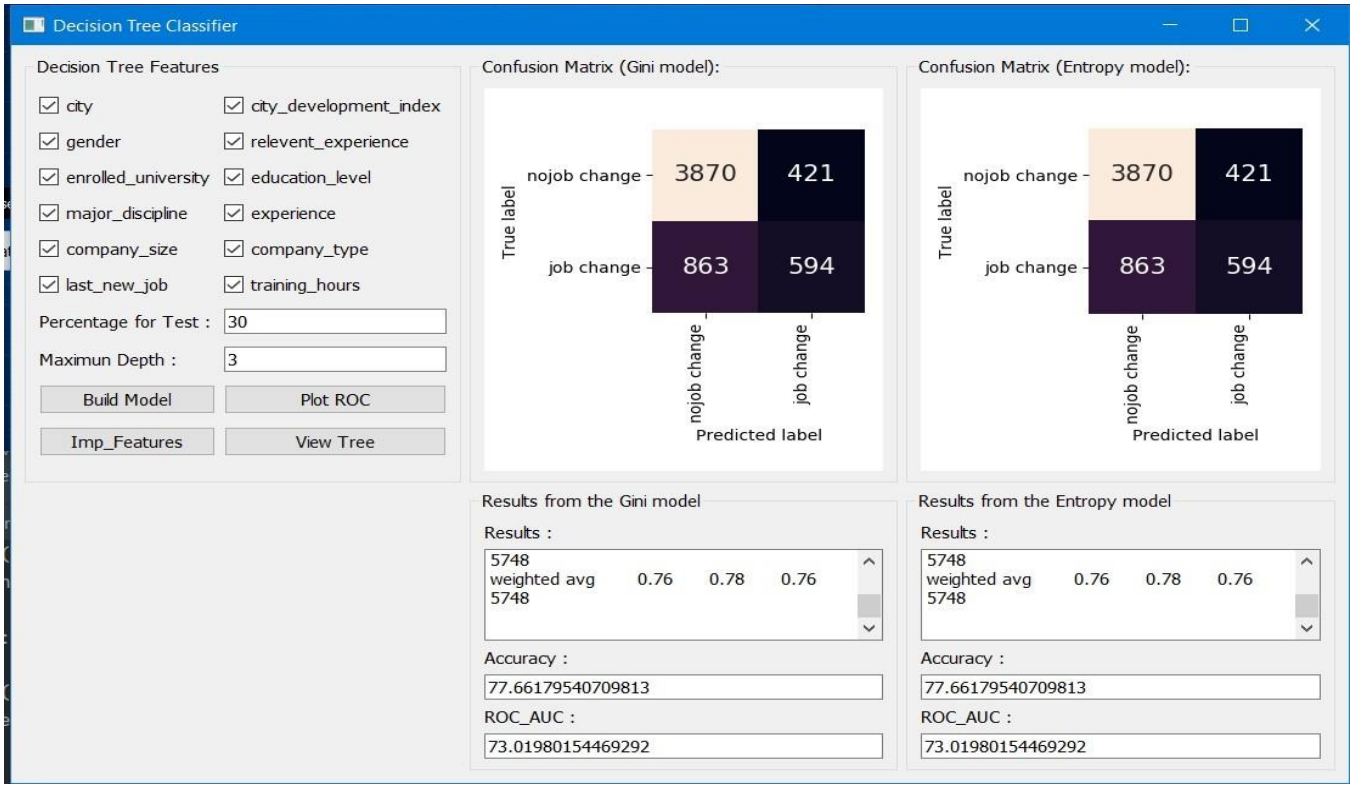



Fig.1.Decision Tree Classifier

ROC_AUC Curve of gini and entropy model (Decision Tree): ROC is a probability curve and AUCrepresent the degree or measure of separability. The green colored represents the roc curve and blue represents the auc. The roc_auc value of decision tree models is 0.73.

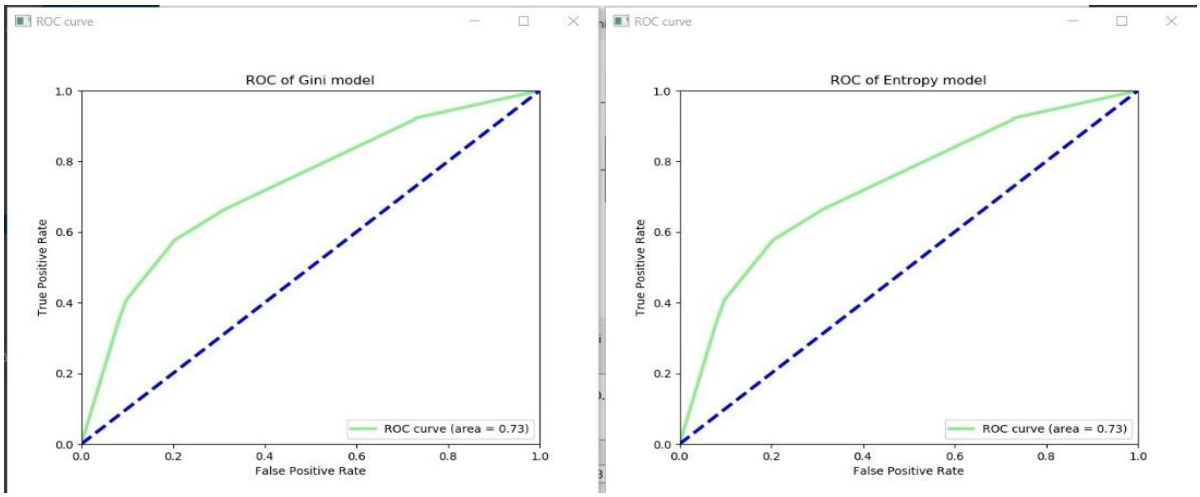
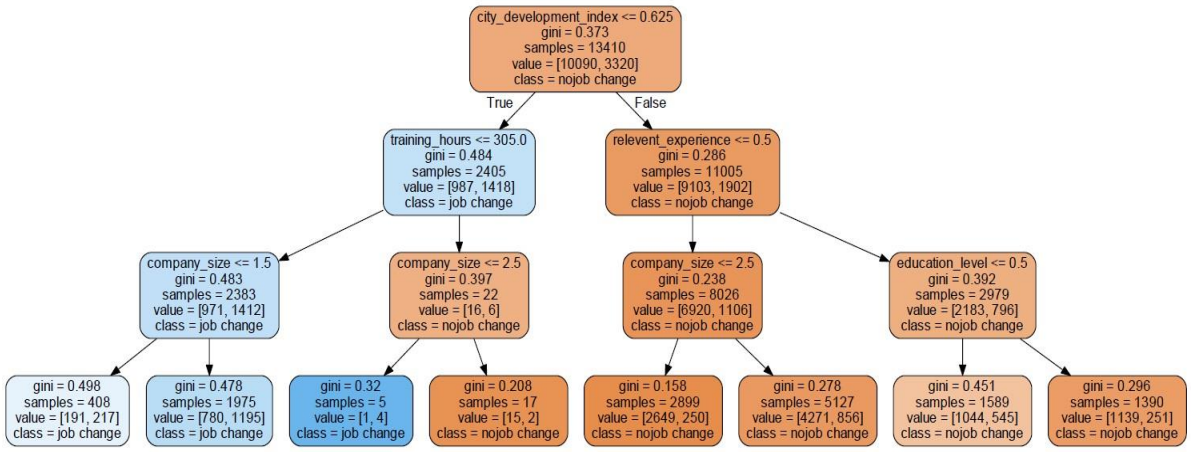


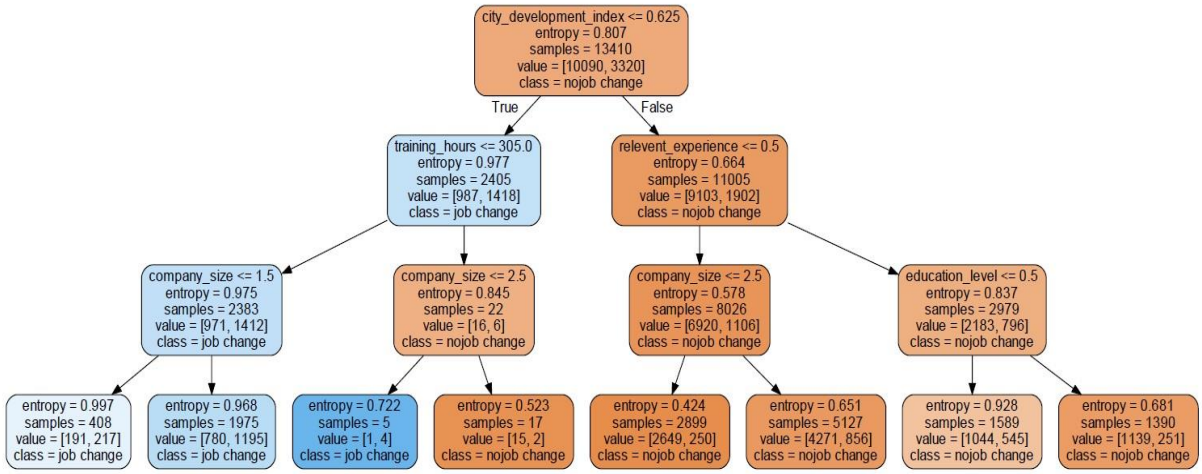

Fig.7. ROC_AUC Curve of gini and entropy model (Decision Tree)

Gini model Decision Tree:




This tree graph shows the split of branches for the gini model of the decision tree model. The maximum depth of the above model is 3.

Entropy model Decision Tree:




This tree graph shows the split of branches for the entropy model of the decision tree model. The maximum depth of the above model is 3.

Feature Importance Graph (Decision Tree): The feature importance graph in decision tree displays the most important features which needs to be considered while building a model. The below figure shows that only 5 features have significant importance to the model.

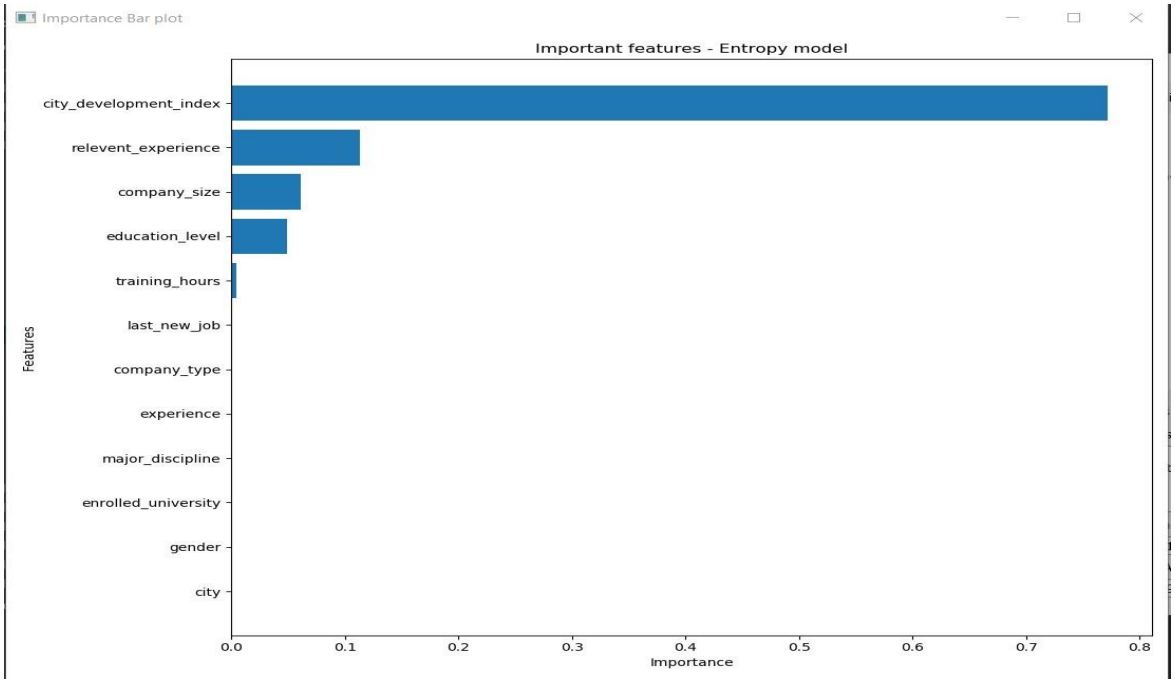



Fig.6.Feature Importance Graph (Decision Tree)

The features with highest importance are – city_development_index, relevant_expericence, company_size, education_level and training_hours. So, these features are most correlated to the target variable in  this model, other features have no importance at all in this model.

Since there are only 5 features which have high importance in this model and the rest have none, it is better to drop all the remaining variables and build new decision tree models.

## SECOND MODEL:

We build two(gini and entropy) new decision tree models with X-variables as  city_development_index, relevant_expericence, company_size, education_level and training_hours and drop the remaining variables. We have chosen these 5 variables from the importance of features plot from the first model.

These are the results that we obtain:

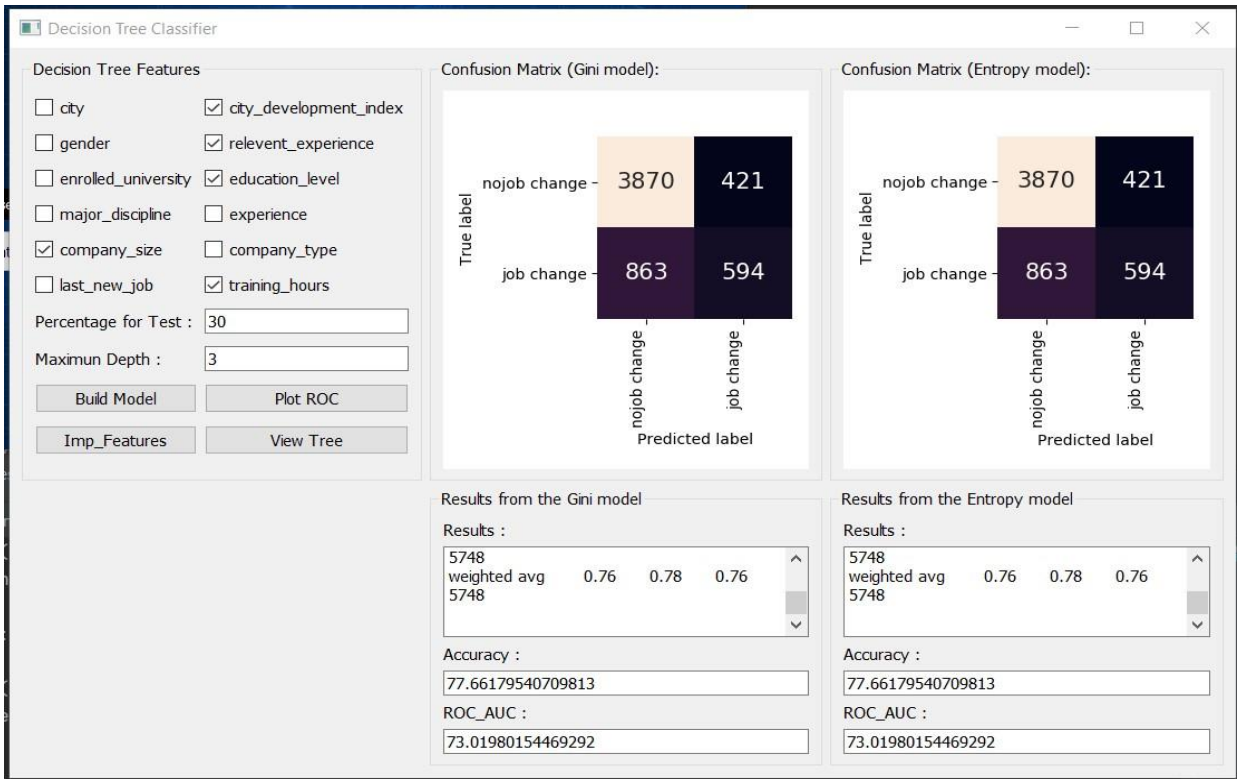



Fig.7.Decision Tree Classifier

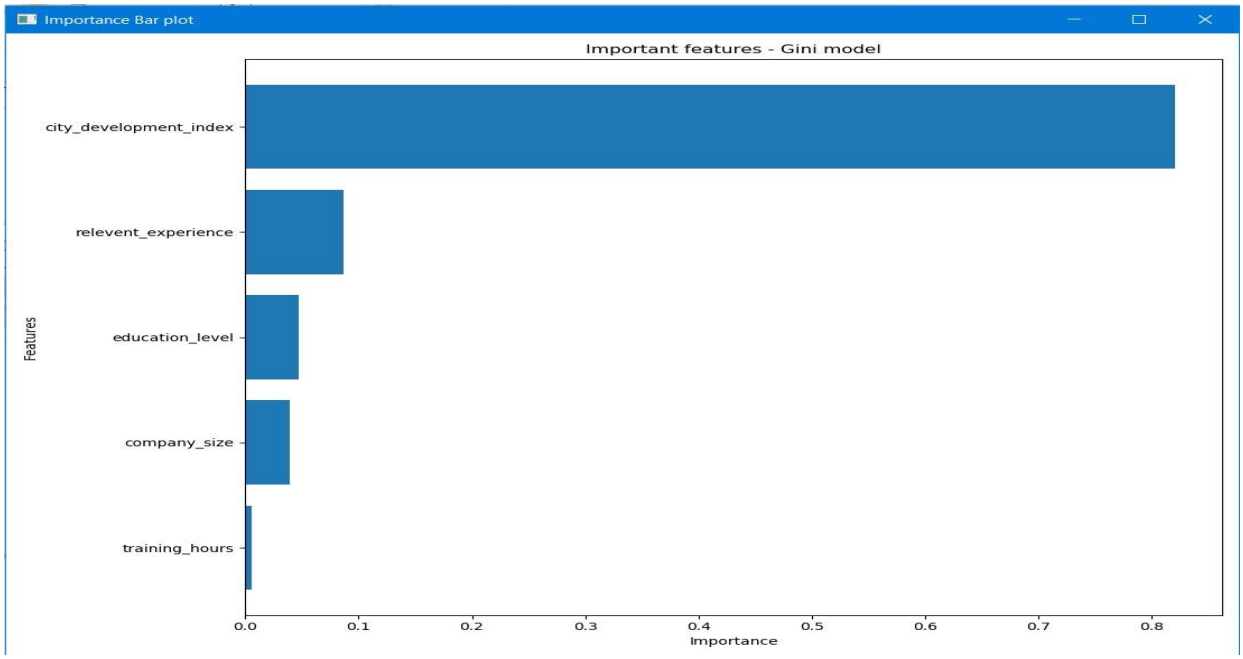



Fig.8.Feature Importance Graph (Decision Tree)

From the main DT figure we can observe that the results have not changes even after we dropped the features of less importance.

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

The in depth details of results of the First and Second model are given in the summary part of the report.

## 5.SUMMARY and CONCLUSION:

## SUMMARY:

## FIRST MODEL:

For gini model:

1. Accuracy of the gini model, Accuracy = 77.66%.
2. From the classification report of gini model:
    1. F1 score for 0's = 0.86 and f1 score for 1's = 0.48
    2. Precision for 0's = 0.82 and precision for 1's = 0.59
    3. Recall for 0's = 0.90 and recall for 1's = 0.41
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. The features with highest importance are – city_development_index, relevant_expericence, company_size, education_level and training_hours. So, these features are most correlated to the target variable in this model, other features have no importance at all in this model.

Accuracy is 77.66% which is acceptable but we can not 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 more 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 gini model, Accuracy = 77.66%.
2. From the classification report of gini model:
    1. F1 score for 0's = 0.86 and f1 score for 1's = 0.48
    2. Precision for 0's = 0.82 and precision for 1's = 0.59
    3. Recall for 0's = 0.90 and recall for 1's = 0.41
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. The features with highest importance are – city_development_index, relevant_expericence, company_size, education_level and training_hours. So, these features are most correlated to the target variable in this model, other features have no importance at all in this model.

Accuracy is 77.66% which is acceptable but we can not 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 more 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.

Since there are only 5 features which have high importance in this model and the rest have none, it is better to drop all the remaining variables and build new decision tree models

**SECOND MODEL:**

So, we build two new decision tree models with X-variables as city_development_index, relevant_expericence, company_size, education_level and training_hours and drop the remaining variables.

From the above figures we can make following observations:

For both gini and entropy models:

1. Accuracy of the gini model, Accuracy = 77.66%.
2. From the classification report of gini model:
    1. F1 score for 0's = 0.86 and f1 score for 1's = 0.48
    2. Precision for 0's = 0.82 and precision for 1's = 0.59
    3. Recall for 0's = 0.90 and recall for 1's = 0.41
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.

We can observe that the results have not changes even after we dropped the features of less importance.

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

## CONCLUSION:

Decision Tree model is the best model of the three different classifiers because it has the highest accuracy, f1-score and precision. It has the best AUC of the three classifiers as well. Also we have built the model with only 5 features which have highest importance of features, so it is more efficient of the three classifiers in the sense of having more accuracy with only 5 dimensions used.

The dataset has more number of 0's than 1's in the dataset, they are not equally proportional so there is always a chance of more false negative values which leads to recall rate and precision of 1's to very low. So, we cannot be sure of the result when we predict 1's but we will be confident of our prediction of 0's as it has high precision and recall rate.

So, in future we should select a dataset that has more equally proportional distribution of target variable.

In conclusion we can say that we can use Decision Tree model to predict whether an employee is going to change to new job or not with an accuracy of 77.66%

## 6.PERCENT OF CODE:

**0%** of code is used from the internet. Search for syntaxes were done to apply some function but was never copied from internet. All the code was developed by coding individually which we learnt from Prof. Amir Jafari.

## 7.REFERENCES:


- https://numpy.org/doc/
- https://matplotlib.org/stable/users/index.html
- https://matplotlibguide.readthedocs.io/en/latest/
- https://sklearn.org/user_guide.html
- https://www.w3schools.com/