A blue and white logo

Description automatically generated

Individual Assessment Coversheet

To be attached to the front of the assessment.

**Campus : MIDRAND \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Faculty: ITMLA2-44**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Group: 1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Lecturer’s Name: LEHONG \_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Student Full Name: NDIVHO MASIYA \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Student Number: DV.2023.C6T1D0**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |
| --- | --- | --- |
| **Indicate** | **Yes** | **No** |
| Plagiarism report attached |  |  |

**Declaration:**

I declare that this assessment is my own original work except for source material explicitly acknowledged. I also declare that this assessment or any other of my original work related to it has not been previously, or is not being simultaneously, submitted for this or any other course. I also acknowledge that I am aware of the Institution’s policy and regulations on honesty in academic work as set out in the Eduvos Conditions of Enrolment, and of the disciplinary guidelines applicable to breaches of such policy and regulations.

|  |  |
| --- | --- |
| **Signature N Masiya** | **Date 20/10/2024** |

**Lecturer’s Comments:**

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| **Marks Awarded:** |  | **%** |
|  |  |  |
| **Signature** | **Date** | |

Eduvos (Pty) Ltd. (formerly Pearson Institute of Higher Education) is registered with the Department of Higher Education and Training as a private higher education institution u

# QUESTION 1

* 1. To be able to detect bots based on Twitter user data, I will have to build a classification model to predict the "Bot Label" (binary: 0 for human, 1 for bot). I would remove irrelevant or redundant features that don't provide information for classification, such as user ID and username.

I would create additional features like ratio of retweets to follower count, time since the account was created, number of hashtags in each tweet. Making use pf the random forest classifier due to its robustness and ability to handle a mix of numerical and categorical data. Random Forests are effective with minimal parameter tuning and offer feature importance, which can be useful in understanding which attributes are most indicative of bot behavior. I would classify the model with accuracy, precision, recall, and F1-score. Since bot detection is a classification problem, these metrics help me assess the model's performance comprehensively. ROC curve and AUC Score will help measure the models ability to distinguish between bots and non-bots.

1.2. # Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score, roc\_auc\_score, roc\_curve

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

data = pd.read\_csv("twitter\_bots.csv")

# 1. Data Cleaning

# Drop irrelevant columns

data = data.drop(columns=['User ID', 'Username', 'Location'])

# Convert 'Verified' to a binary numerical feature (0 or 1)

data['Verified'] = data['Verified'].astype(int)

# Convert 'Created At' to datetime and calculate account age in days

data['Created At'] = pd.to\_datetime(data['Created At'], errors='coerce')

data['Account Age'] = (pd.Timestamp.now() - data['Created At']).dt.days

# Fill missing values in 'Account Age' with the median age

data['Account Age'].fillna(data['Account Age'].median(), inplace=True)

# Calculate Hashtag Count

data['Hashtag Count'] = data['Hashtags'].apply(lambda x: len(str(x).split(',')) if pd.notnull(x) else 0)

# Drop columns that won't be used for training

data = data.drop(columns=['Created At', 'Hashtags', 'Tweet'])

# Separate features and target

X = data.drop(columns=['Bot Label'])

y = data['Bot Label']

# 2. Feature Standardization

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# 3. Splitting data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# 4. Model Training

classifier = RandomForestClassifier(random\_state=42)

classifier.fit(X\_train, y\_train)

# 5. Model Evaluation

# Predictions and probabilities

y\_pred = classifier.predict(X\_test)

y\_pred\_proba = classifier.predict\_proba(X\_test)[:, 1]

# Print classification report and confusion matrix

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

# Accuracy Score

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")

# ROC Curve and AUC Score

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

# Plot ROC Curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, label=f"Random Forest (AUC = {roc\_auc:.2f})")

plt.plot([0, 1], [0, 1], 'k--')

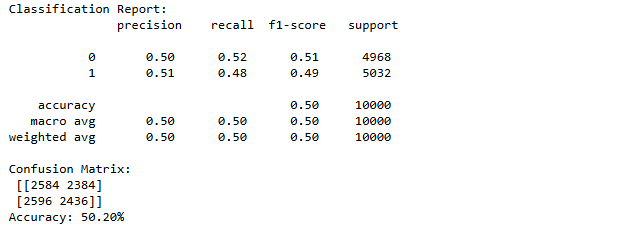
plt.xlabel("False Positive Rate")

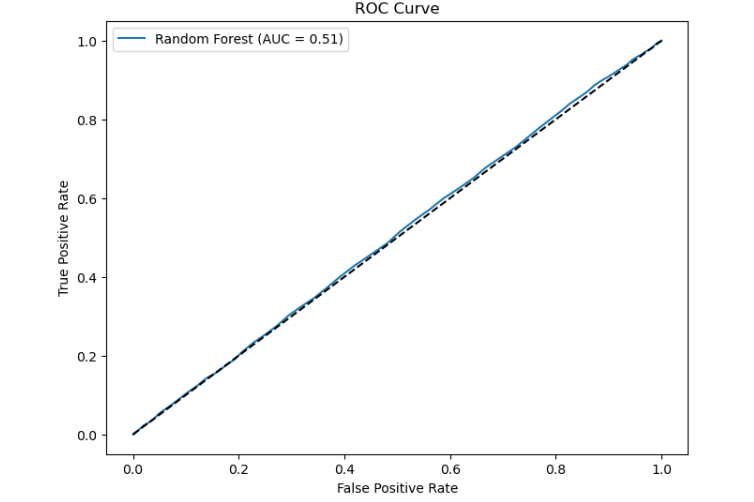
plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend()

plt.show()





# QUESTION 2

2.1. # Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Load the dataset

file\_path = 'Prek\_HR\_data.csv' # Update with the path to your dataset if necessary

data = pd.read\_csv(file\_path)

# Select the specified features and target

features = data[['nqf', 'age', 'exp\_level']] # Features: NQF level, Age, Experience Level

target = data['q\_score'] # Target: Quality Score

# Split data into training and validation sets (85% training, 15% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.15, random\_state=42)

# Display the sizes of the training and test sets

print("Training set size:", X\_train.shape[0])

print("Test set size:", X\_test.shape[0])

# Show the first few rows of the training set as a check

print("Training Set (Features):\n", X\_train.head())

print("Training Set (Target):\n", y\_train.head())



2.2. # Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

# Load the dataset

file\_path = 'Prek\_HR\_data.csv'

data = pd.read\_csv(file\_path)

# Select features and target

features = data[['nqf', 'age', 'exp\_level']] # Features: NQF level, Age, Experience Level

target = data['q\_score'] # Target: Quality Score

# Split data into training and validation sets (85% training, 15% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.15, random\_state=42)

# Initialize and train the Decision Tree Classifier

classifier = DecisionTreeClassifier(random\_state=42)

classifier.fit(X\_train, y\_train)

# Calculate accuracy on training and test sets

train\_accuracy = accuracy\_score(y\_train, classifier.predict(X\_train))

test\_accuracy = accuracy\_score(y\_test, classifier.predict(X\_test))

print(f'Training Accuracy: {train\_accuracy \* 100:.2f}%')

print(f'Test Accuracy: {test\_accuracy \* 100:.2f}%')

# Plot the decision tree using matplotlib

plt.figure(figsize=(15, 10))

plot\_tree(

classifier,

feature\_names=['nqf', 'age', 'exp\_level'],

class\_names=['Not Qualified', 'Qualified'],

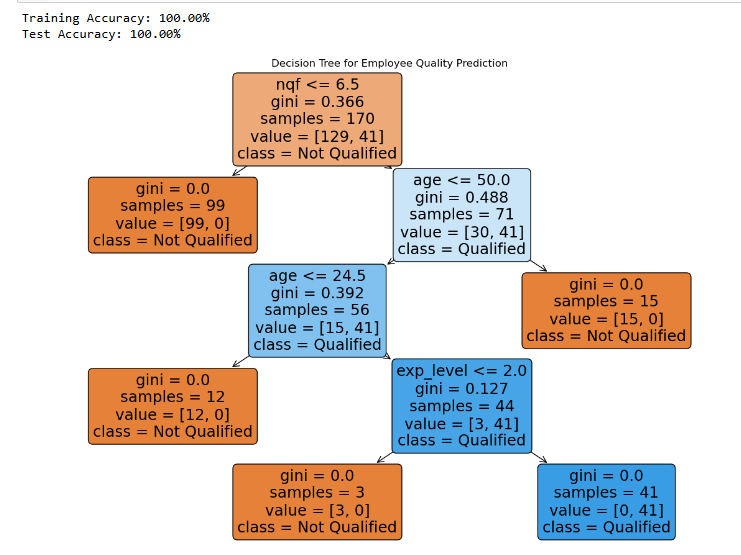
filled=True,

rounded=True

)

plt.title("Decision Tree for Employee Quality Prediction")

plt.show()



2.3. import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Load the dataset

file\_path = 'Prek\_HR\_data.csv'

data = pd.read\_csv(file\_path)

# Check the columns in the DataFrame

print("Columns in the DataFrame:")

print(data.columns.tolist()) # Print as a list for easier readability

# Strip whitespace from column names

data.columns = data.columns.str.strip()

# Selecting features and target

features = data[['age', 'nqf', 'exp\_level']]

# Use the correct column name for quality score

target = data['q\_score'] # Updated to match the actual column name

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.15, random\_state=42)

# Initialize and train the decision tree classifier

classifier = DecisionTreeClassifier(random\_state=42)

classifier.fit(X\_train, y\_train)

# Define the candidate profiles to predict their quality scores

candidates = pd.DataFrame({

'age': [23, 30, 60, 45, 36],

'nqf': [7, 9, 8, 10, 6],

'exp\_level': [1, 3, 3, 8, 10]

})

# Ensure that the columns in candidates match the training feature names

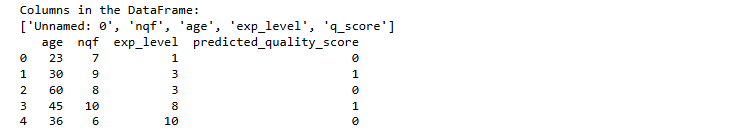
candidates = candidates[features.columns]

# Predict the quality score for each candidate

predictions = classifier.predict(candidates)

# Display the results

candidates['predicted\_quality\_score'] = predictions

print(candidates)

2.4. import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

# Load the dataset

file\_path = 'Prek\_HR\_data.csv'

data = pd.read\_csv(file\_path)

# Select features and target

features = data[['nqf', 'age', 'exp\_level']]

target = data['q\_score']

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.15, random\_state=42)

# Initialize and train the classifier

classifier = DecisionTreeClassifier(random\_state=42)

classifier.fit(X\_train, y\_train)

# Define candidate profiles with matching column order

candidates = pd.DataFrame({

'nqf': [7, 9, 8, 10, 6],

'age': [23, 30, 60, 45, 36],

'exp\_level': [1, 3, 3, 8, 10]

})

# Predict quality score for each candidate

candidates['predicted\_quality\_score'] = classifier.predict(candidates)

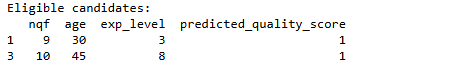
# Filter only eligible candidates (assuming 1 means "Qualified")

eligible\_candidates = candidates[candidates['predicted\_quality\_score'] == 1]

# Display eligible candidates

print("Eligible candidates:")

print(eligible\_candidates)



# Question 3

import numpy as np

import matplotlib.pyplot as plt

# Data Points: (Number of Items Purchased, Number of Store Visits)

data = np.array([

[2, 10], # Customer 1 (C1)

[2, 5], # Customer 2 (C2)

[8, 4], # Customer 3 (C3)

[5, 8], # Customer 4 (C4)

[7, 5], # Customer 5 (C5)

[6, 4], # Customer 6 (C6)

[1, 2], # Customer 7 (C7)

[4, 9] # Customer 8 (C8)

])

# Initial Cluster Centers

centroids = np.array([

[2, 10], # Cluster 1 Center: Customer 1 (C1)

[5, 8], # Cluster 2 Center: Customer 4 (C4)

[1, 2] # Cluster 3 Center: Customer 7 (C7)

])

# Function to calculate Euclidean distance

def euclidean\_distance(point, centroid):

return np.sqrt(np.sum((point - centroid) \*\* 2))

# K-Means Algorithm

num\_clusters = centroids.shape[0]

num\_epochs = 3

# Plotting function

def plot\_clusters(data, centroids, labels, epoch):

plt.figure(figsize=(8, 6))

colors = ['r', 'g', 'b']

for i in range(num\_clusters):

plt.scatter(data[labels == i][:, 0], data[labels == i][:, 1],

color=colors[i], label=f'Cluster {i + 1}')

plt.scatter(centroids[:, 0], centroids[:, 1],

color='k', marker='X', s=200, label='Centroids')

plt.title(f'K-Means Clustering (Epoch {epoch + 1})')

plt.xlabel('Number of Items Purchased')

plt.ylabel('Number of Store Visits')

plt.legend()

plt.grid()

plt.show()

# Main Loop

for epoch in range(num\_epochs):

# Step 1: Assign clusters based on closest centroid

labels = np.zeros(data.shape[0])

for i in range(data.shape[0]):

distances = [euclidean\_distance(data[i], centroids[j]) for j in range(num\_clusters)]

labels[i] = np.argmin(distances)

# Step 2: Calculate new centroids

new\_centroids = np.array([data[labels == i].mean(axis=0) for i in range(num\_clusters)])

# Update centroids

centroids = new\_centroids

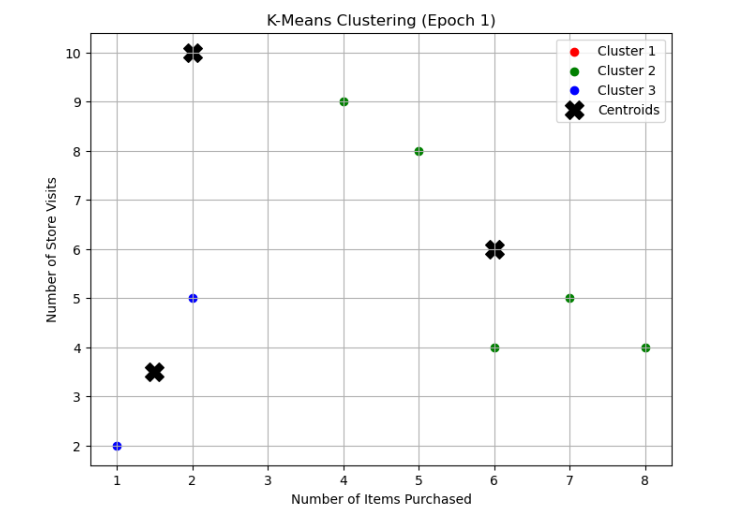
# Plotting the clusters and centroids

plot\_clusters(data, centroids, labels, epoch)

# Final clustering

print("Final cluster labels after 3 epochs:", labels)

print("Final centroids:", centroids)



A graph with numbers and symbols

Description automatically generated

