Name\*: Ankit Sharma

Roll No.: 202401100300049

Branch\*: CSEAI A

Institute\*: KIET GROUP OF INSTITUTIONS

Topic\*: Customer Support Case Type Classification

Introduction

Customer support centers often handle a wide range of queries related to billing, technical issues, or general information. Manually sorting through these cases is time-consuming and inefficient. The objective of this project is to develop a machine learning model that automatically classifies customer support cases into one of three categories: \*Billing, \*\*Technical, or \*\*General\* queries, using features like message length and response time.

Methodology

1. The dataset containing support case metadata such as message\_length, response\_time, and case\_type was loaded using Pandas.

2. Label encoding was applied to convert the categorical case\_type into numerical format.

3. Features (message\_length and response\_time) and labels were prepared.

4. The data was split into training and testing sets.

5. A \*Random Forest Classifier\* was trained on the training set.

6. The model was evaluated using a classification report.

7. A prediction function was implemented to classify new data based on message length and response time.

8. Visualization using bar plots and scatter plots was performed to understand the distribution and relationship in the data.

Code

python

# STEP 1: Install dependencies (if needed)

!pip install pandas scikit-learn

# STEP 2: Import libraries

import pandas as pd

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

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

# STEP 3: Load the dataset

df = pd.read\_csv('/content/support\_cases.csv')

# STEP 4: Encode the labels

label\_encoder = LabelEncoder()

df['case\_type\_encoded'] = label\_encoder.fit\_transform(df['case\_type'])

# STEP 5: Prepare features and labels

X = df[['message\_length', 'response\_time']]

y = df['case\_type\_encoded']

# STEP 6: Split the dataset

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

# STEP 7: Train the model

model = RandomForestClassifier(random\_state=42)

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

# STEP 8: Evaluate the model

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

print(classification\_report(y\_test, y\_pred, target\_names=label\_encoder.classes\_))

# STEP 9: Prediction function

def predict\_case\_type(message\_length, response\_time):

input\_data = pd.DataFrame([[message\_length, response\_time]], columns=['message\_length', 'response\_time'])

pred\_encoded = model.predict(input\_data)[0]

return label\_encoder.inverse\_transform([pred\_encoded])[0]

# STEP 10: User input and prediction

message\_length = int(input("Enter message length: "))

response\_time = int(input("Enter response time: "))

predicted\_case\_type = predict\_case\_type(message\_length, response\_time)

print(f"Predicted case type: {predicted\_case\_type}")

Output/Result

A screenshot of the output showing prediction and model evaluation was attached as evidence of successful execution.

References/Credits

- Dataset: Provided during the project

- Libraries: scikit-learn, pandas, seaborn, matplotlib

- Visualization code:

python

# STEP 1: Install visualization libraries

!pip install matplotlib seaborn

# STEP 2: Import visualization libraries

import matplotlib.pyplot as plt

import seaborn as sns

# STEP 3: Load dataset

df = pd.read\_csv('/content/support\_cases.csv')

# STEP 4: Bar plot for case type counts

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

sns.countplot(data=df, x='case\_type', palette='Set2')

plt.title('Count of Case Types')

plt.xlabel('Case Type')

plt.ylabel('Count')

plt.show()

# STEP 5: Scatter plot

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

sns.scatterplot(data=df, x='message\_length', y='response\_time', hue='case\_type', palette='Set2', style='case\_type', s=100)

plt.title('Message Length vs. Response Time by Case Type')

plt.xlabel('Message Length')

plt.ylabel('Response Time')

plt.legend(title='Case Type')

plt.show()