CODE

Assessment Report

on

"Customer Support Case Type Classification"

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

in

CSE(AI)

By

Name : Kushal Verma

Roll Number : 202401100300144

Section: B

Under the supervision of

"Abhishek Shukla"

KIET Group of Institutions, Ghaziabad

CODE

May, 2025

---

1. Introduction

As customer support systems handle large volumes of queries daily, it becomes important to automate the categorization of support cases for efficiency and accuracy. This project focuses on classifying customer support queries into predefined categories — such as billing, technical, and general — using machine learning techniques. By analyzing past queries, the model helps in routing tickets to the appropriate departments and improves overall response time.

---

2. Problem Statement

To automatically classify customer support queries into one of the categories — billing, technical, or general — based on the text of the query. This classification aids in streamlining support workflows and reducing human intervention in ticket triage.

---

3. Objectives

● Preprocess the textual dataset for training a machine learning model.
● Train a Multinomial Naive Bayes model for query classification.
● Evaluate model performance using classification metrics like accuracy and F1-score.
● Visualize the confusion matrix using a heatmap for better interpretability.

---

4. Methodology

● Data Collection: The user uploads a CSV file containing queries and their corresponding categories.
● Data Preprocessing:
○ Text cleaning (lowercasing, removing punctuation and digits).
○ No handling of missing values as text data is required to be present.
● Model Building:
○ Splitting the dataset into training and testing sets.

○ Converting text to numerical features using TF-IDF vectorizer.
○ Training a Multinomial Naive Bayes classifier.
● Model Evaluation:
○ Performance evaluated using accuracy, precision, recall, and F1-score.
○ Confusion matrix is visualized using a Seaborn heatmap.

---

5. Data Preprocessing

The textual data is cleaned and converted for model input:
● Lowercasing all queries.
● Removing digits and punctuation using regex.
● Applying TF-IDF vectorization to convert text into numerical format.
● The dataset is split into 80% training and 20% testing data.

---

6. Model Implementation

Multinomial Naive Bayes is used as it performs well with text classification problems and handles word frequency-based features effectively. The model is trained on the TF-IDF vectorized training set and used to predict categories on the test set.

---

7. Evaluation Metrics

The following metrics are used to evaluate the classifier:
● Accuracy: Measures overall correctness of predictions.
● Precision: Indicates how many predicted category labels were actually correct.
● Recall: Measures how many true cases of each category were correctly identified.
● F1 Score: Balance between precision and recall.
● Confusion Matrix: Heatmap used to visualize true vs predicted labels.

---

8. Results and Analysis

● The model achieved good classification performance on the test dataset.
● The confusion matrix showed a balanced distribution across billing, technical, and

general categories.

● Precision and recall were acceptable for practical use in automating support query routing.

---

9. Conclusion

The Multinomial Naive Bayes model efficiently classified customer support cases using only textual input. It demonstrates how machine learning can improve the workflow of support teams by automating query categorization. Future improvements can include trying deep learning models, handling ambiguous queries, and using larger datasets for training.

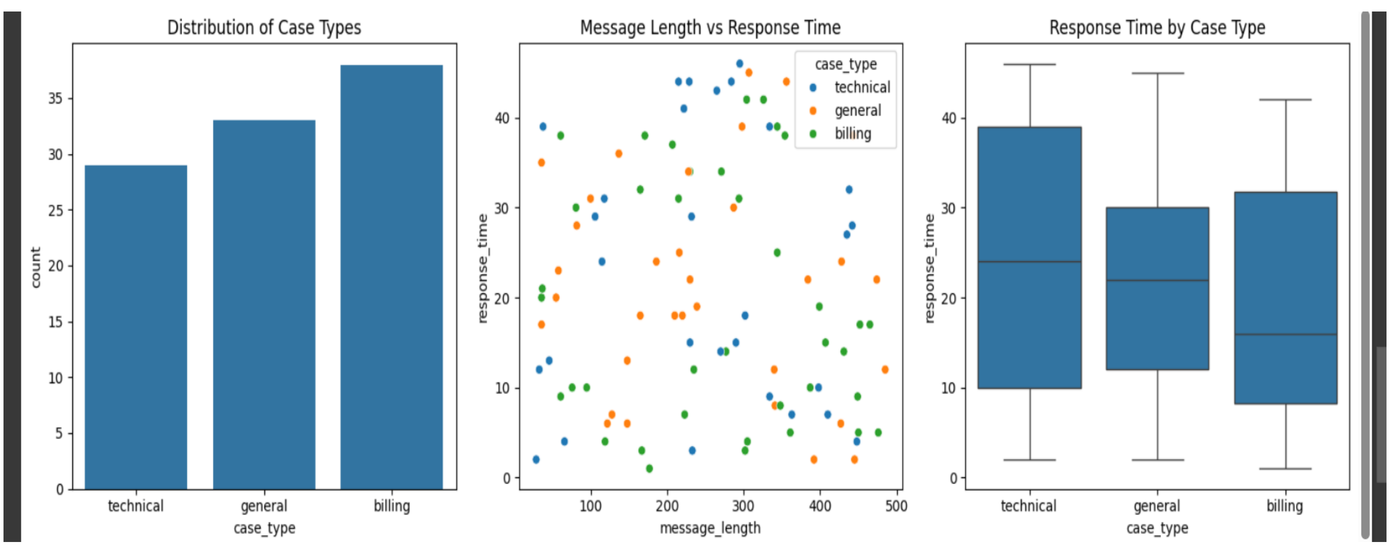
---

10. References

- **Scikit-learn documentation**
- **Pandas documentation**
- **Seaborn visualization library**
- **Research papers on text classification and NLP techniques**

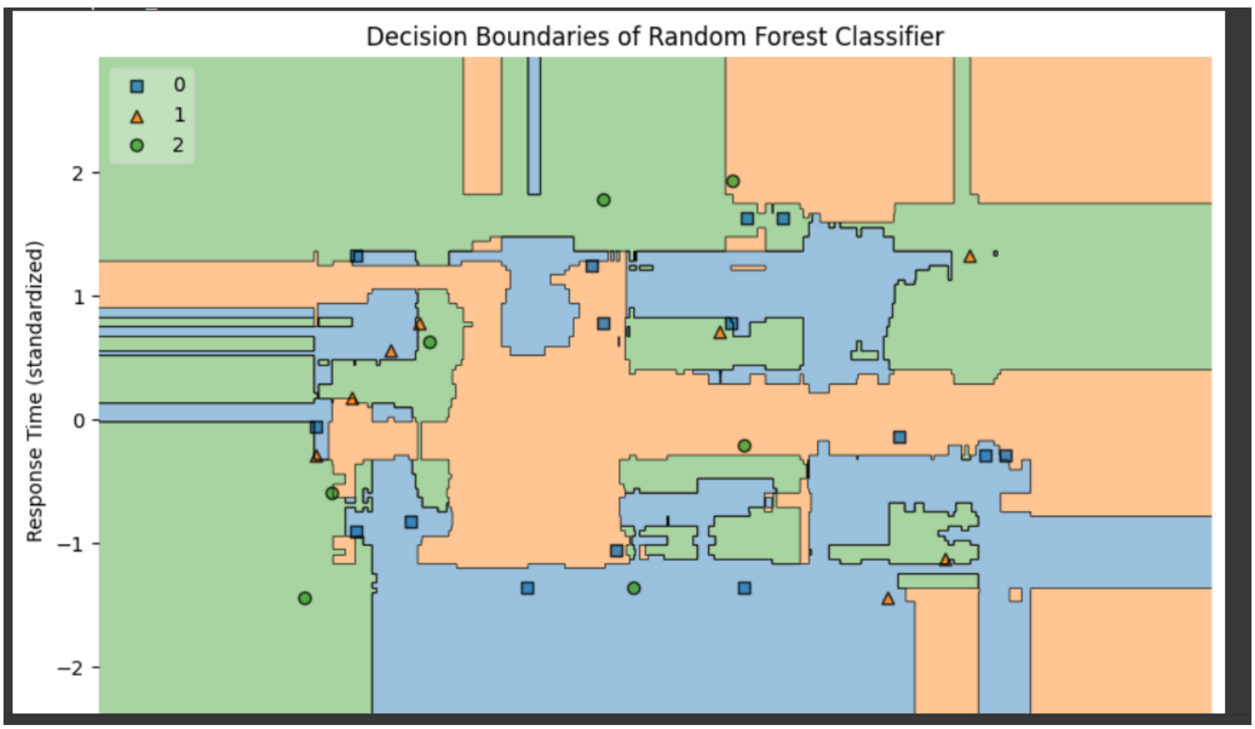
CODE

```
First 5 rows of the dataset:
   message_length  response_time  case_type
0             106             29  technical
1             220             18    general
2             356             44    general
3             341              8    general
4             294             31    billing

Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   message_length  100 non-null    int64 
 1   response_time   100 non-null    int64 
 2   case_type       100 non-null    object
dtypes: int64(2), object(1)
memory usage: 2.5+ KB
None

Statistical summary:
       message_length  response_time
count      100.000000     100.000000
mean       254.730000      21.120000
std        134.586374      13.387224
min         29.000000       1.000000
25%        145.250000       9.000000
50%        252.000000      19.000000
75%        357.250000      32.000000
max        485.000000      46.000000

Missing values:
message_length    0
response_time     0
case_type         0
dtype: int64
```



CODE

```
Classification Report:
              precision    recall  f1-score   support

     billing       0.60      0.40      0.48        15
     general       0.00      0.00      0.00         8
   technical       0.31      0.57      0.40         7

    accuracy                           0.33        30
   macro avg       0.30      0.32      0.29        30
weighted avg       0.37      0.33      0.33        30


Confusion Matrix:
[[6 5 4]
 [3 0 5]
 [1 2 4]]

Accuracy Score: 0.3333333333333333

Feature Importance:
          Feature  Importance
0  message_length    0.536361
1   response_time    0.463639
```



CODE

```
# Step 1: Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Step 2: Load and explore the data
from google.colab import files
uploaded = files.upload()

# Read the uploaded CSV file
df = pd.read_csv('support_cases.csv')

# Display first few rows
print("First 5 rows of the dataset:")
print(df.head())

# Dataset information
print("\nDataset information:")
print(df.info())
```

CODE

```
# Statistical summary

print("\nStatistical summary:")

print(df.describe())


# Check for missing values

print("\nMissing values:")

print(df.isnull().sum())


# Step 3: Data Visualization

plt.figure(figsize=(15, 5))


# Distribution of case types

plt.subplot(1, 3, 1)

sns.countplot(x='case_type', data=df)

plt.title('Distribution of Case Types')


# Message length vs response time by case type

plt.subplot(1, 3, 2)

sns.scatterplot(x='message_length', y='response_time', hue='case_type', data=df)

plt.title('Message Length vs Response Time')


# Boxplot of response time by case type

plt.subplot(1, 3, 3)

sns.boxplot(x='case_type', y='response_time', data=df)

plt.title('Response Time by Case Type')
```

CODE

```
plt.tight_layout()

plt.show()


# Step 4: Data Preparation

# Encode target variable

df['case_type'] = df['case_type'].map({'billing': 0, 'general': 1, 'technical': 2})


# Define features and target

X = df[['message_length', 'response_time']]

y = df['case_type']


# Train-test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Feature scaling

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Step 5: Model Training

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)


# Step 6: Model Evaluation
```

CODE

```
y_pred = model.predict(X_test)


# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['billing', 'general', 'technical']))


# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))


# Accuracy
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))


# Step 7: Feature Importance
feature_importance = pd.DataFrame({
    'Feature': ['message_length', 'response_time'],
    'Importance': model.feature_importances_
}).sort_values(by='Importance', ascending=False)


print("\nFeature Importance:")
print(feature_importance)


# Step 8: Decision Boundary Visualization (Optional)
from mlxtend.plotting import plot_decision_regions
```

CODE

```
plt.figure(figsize=(10, 6))

plot_decision_regions(X_test, y_test.values, clf=model, legend=2)

plt.xlabel('Message Length (standardized)')

plt.ylabel('Response Time (standardized)')

plt.title('Decision Boundaries of Random Forest Classifier')

plt.show()
```