**Revolutionizing customer support with an intelligent chatbot for automated assistance**

# #Source code for chatbot intent classification

# #Program:

# import pandas as pd

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

# from sklearn.feature\_extraction.text import TfidfVectorizer

# from sklearn.ensemble import RandomForestClassifier

# from sklearn.preprocessing import LabelEncoder

# import gradio as gr

# # === Load and Prepare Dataset ===

# df = pd.read\_csv("Training data.csv")

# X = df['instruction']

# y = df['intent']

# # Encode labels

# le = LabelEncoder()

# y\_encoded = le.fit\_transform(y)

# # Train/Test split (not required for deployment, but good for model selection)

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

# # TF-IDF Vectorization

# tfidf = TfidfVectorizer()

# X\_train\_vec = tfidf.fit\_transform(X\_train)

# # Train Random Forest model

# rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

# rf\_model.fit(X\_train\_vec, y\_train)

# # === Prediction Function for Gradio ===

# def predict\_intent(user\_input):

# input\_vector = tfidf.transform([user\_input])

# prediction = rf\_model.predict(input\_vector)[0]

# intent\_name = le.inverse\_transform([prediction])[0]

# return f"Predicted Intent: {intent\_name}"

# # === Gradio Interface ===

# interface = gr.Interface(

# fn=predict\_intent,

# inputs="text",

# outputs="text",

# title="Intent Classifier (Customer Support Chatbot)",

# description="Enter a customer query to predict its intent from the training data."

# )

# interface.launch(share=True)

# #output:

# 

# #Program for measuring the model performance:

# import pandas as pd

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

# from sklearn.feature\_extraction.text import TfidfVectorizer

# from sklearn.ensemble import RandomForestClassifier

# from sklearn.preprocessing import LabelEncoder

# from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

# # === Load and Prepare Data ===

# df = pd.read\_csv("Training data.csv")

# X = df['instruction']

# y = df['intent']  # or use 'category' if preferred

# # Encode target labels

# le = LabelEncoder()

# y\_encoded = le.fit\_transform(y)

# # Train/Test Split

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

# # TF-IDF Vectorization

# tfidf = TfidfVectorizer()

# X\_train\_vec = tfidf.fit\_transform(X\_train)

# X\_test\_vec = tfidf.transform(X\_test)

# # Train Random Forest Classifier

# model = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

# # Make Predictions

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

# # === Evaluation Metrics ===

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

# precision = precision\_score(y\_test, y\_pred, average='weighted', zero\_division=0)

# recall = recall\_score(y\_test, y\_pred, average='weighted', zero\_division=0)

# f1 = f1\_score(y\_test, y\_pred, average='weighted', zero\_division=0)

# print("=== Evaluation Metrics ===")

# print(f"Accuracy : {accuracy:.4f}")

# print(f"Precision: {precision:.4f}")

# print(f"Recall   : {recall:.4f}")

# print(f"F1-score : {f1:.4f}")

# # Detailed breakdown per class

# print("\n=== Classification Report ===")

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

# #Output:

# 