import joblib  
import numpy as np  
  
X\_reduced = joblib.load("../artifacts/X\_reduced.pkl") # shape (31420, 150)  
Y = joblib.load("../artifacts/Y.pkl") # shape (31420, 3694)  
  
print("Loaded reduced features and labels")  
print(f"X\_reduced shape: {X\_reduced.shape}")  
print(f"Y shape: {Y.shape}")

**Description**: This code loads the reduced feature matrix X\_reduced and the multi-label targets Y from serialized .pkl files using joblib. It prints their shapes to confirm successful loading.

from lightgbm import LGBMClassifier  
from sklearn.multioutput import MultiOutputClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report, hamming\_loss, accuracy\_score  
  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_reduced, Y, test\_size=0.2, random\_state=42)  
  
model = MultiOutputClassifier(LGBMClassifier(n\_estimators=50, random\_state=42))  
model.fit(X\_train, Y\_train)  
  
Y\_pred = model.predict(X\_test)  
  
print("✨ Initial Results")  
print(classification\_report(Y\_test, Y\_pred, zero\_division=0))  
print("Hamming Loss:", hamming\_loss(Y\_test, Y\_pred))  
print("Subset Accuracy:", accuracy\_score(Y\_test, Y\_pred))

**Description**: This is a quick sanity check using a LightGBM classifier wrapped in MultiOutputClassifier for multi-label classification. The data is split and the model is trained, followed by predictions and evaluation using several classification metrics.

import pandas as pd  
  
label\_counts = pd.Series(Y.sum(axis=0))  
print(label\_counts)  
  
feature\_counts = pd.Series(X\_reduced.sum(axis=0))  
print(feature\_counts)

**Description**: This block calculates the number of positive (non-zero) instances for each label and feature, giving a sense of label distribution and sparsity in the dataset.

import numpy as np  
  
min\_label\_count = 50  
label\_mask = np.array(label\_counts >= min\_label\_count)  
  
Y\_filtered = Y[:, label\_mask]  
  
print("Labels after filtering:", Y\_filtered.shape[1])

**Description**: Filters out rare labels with fewer than 50 positive samples. This helps reduce noise and computational load in subsequent steps.

row\_mask = Y\_filtered.sum(axis=1) > 0  
  
X\_filtered = X\_reduced[row\_mask]  
Y\_filtered = Y\_filtered[row\_mask]  
  
print("Filtered X shape:", X\_filtered.shape)  
print("Filtered Y shape:", Y\_filtered.shape)

**Description**: Removes samples that have no remaining positive labels after label filtering, ensuring each sample has at least one label.

from sklearn.model\_selection import train\_test\_split  
  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(  
 X\_filtered, Y\_filtered, test\_size=0.2, random\_state=42  
)  
  
print("Training samples:", X\_train.shape[0])  
print("Testing samples:", X\_test.shape[0])

**Description**: Splits the filtered data into training and testing sets using an 80-20 split for model evaluation.

from sklearn.linear\_model import LogisticRegression  
from sklearn.multiclass import OneVsRestClassifier  
from sklearn.metrics import f1\_score, classification\_report  
  
model = OneVsRestClassifier(LogisticRegression(solver='liblinear'))  
model.fit(X\_train, Y\_train)  
  
Y\_pred = model.predict(X\_test)  
  
print("F1 Score (micro):", f1\_score(Y\_test, Y\_pred, average='micro'))  
print("F1 Score (macro):", f1\_score(Y\_test, Y\_pred, average='macro'))  
print("\nClassification report:")  
print(classification\_report(Y\_test, Y\_pred))

**Description**: Trains a baseline logistic regression model using One-vs-Rest strategy and evaluates it using F1 scores and a detailed classification report.

from sklearn.ensemble import RandomForestClassifier  
from sklearn.multiclass import OneVsRestClassifier  
from sklearn.metrics import f1\_score  
  
rf\_model = OneVsRestClassifier(RandomForestClassifier(n\_estimators=100, n\_jobs=-1))  
rf\_model.fit(X\_train, Y\_train)  
  
Y\_pred\_rf = rf\_model.predict(X\_test)  
  
print("Random Forest F1 Score (micro):", f1\_score(Y\_test, Y\_pred\_rf, average='micro'))  
print("Random Forest F1 Score (macro):", f1\_score(Y\_test, Y\_pred\_rf, average='macro'))

**Description**: Implements a One-vs-Rest classifier using Random Forests. Useful for capturing complex non-linear patterns in the data. Evaluated with F1 metrics.

%pip install xgboost  
  
from xgboost import XGBClassifier  
from sklearn.multiclass import OneVsRestClassifier  
  
xgb\_model = OneVsRestClassifier(XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', n\_jobs=-1))  
xgb\_model.fit(X\_train, Y\_train)  
  
Y\_pred\_xgb = xgb\_model.predict(X\_test)  
  
print("XGBoost F1 Score (micro):", f1\_score(Y\_test, Y\_pred\_xgb, average='micro'))  
print("XGBoost F1 Score (macro):", f1\_score(Y\_test, Y\_pred\_xgb, average='macro'))

**Description**: Trains an XGBoost model using One-vs-Rest scheme, which is efficient and powerful for high-dimensional multi-label tasks. Reports both micro and macro F1 scores.

from sklearn.metrics import classification\_report  
  
print("Classification Report for XGBoost:")  
print(classification\_report(Y\_test, Y\_pred\_xgb))

**Description**: Provides a detailed classification report for the XGBoost model, showing per-label precision, recall, and F1-score.

import joblib  
  
joblib.dump(xgb\_model, 'xgb\_multilabel\_model.pkl')

**Description**: Saves the trained XGBoost model to a local .pkl file for future use.

import os  
import joblib  
  
save\_path = "../saved\_model"  
os.makedirs(save\_path, exist\_ok=True)  
  
model\_path = os.path.join(save\_path, "xgb\_multilabel\_model.pkl")  
  
joblib.dump(xgb\_model, model\_path)  
  
print(f"Model saved successfully to: {model\_path}")

**Description**: Creates a directory and saves the XGBoost model there, ensuring a structured storage setup for production or further experimentation.

import streamlit as st  
import joblib  
import numpy as np  
  
# Load the trained model and preprocessed data  
model = joblib.load("saved\_model/xgb\_multilabel\_model.pkl")  
X\_reduced = joblib.load("artifacts/X\_reduced.pkl") # For feature format reference  
Y = joblib.load("artifacts/Y.pkl") # For label names  
  
# Get number of features  
num\_features = X\_reduced.shape[1]  
num\_labels = Y.shape[1]  
  
# Label mapping (can be improved if you have actual label names)  
label\_names = [f"Label {i}" for i in range(Y.shape[1])]  
  
# Streamlit UI  
st.title("🔍 Multi-label Classification App")  
st.write("Enter feature values to predict associated labels.")  
  
# Feature input  
input\_values = []  
with st.form("prediction\_form"):  
 for i in range(num\_features):  
 val = st.number\_input(f"Feature {i + 1}", value=0.0, key=f"feat\_{i}")  
 input\_values.append(val)  
  
 submitted = st.form\_submit\_button("Predict Labels")  
  
if submitted:  
 X\_input = np.array(input\_values).reshape(1, -1)  
  
 # Make prediction  
 prediction = model.predict(X\_input)  
  
 # Display results  
 st.subheader("✅ Predicted Labels:")  
 predicted\_labels = [label\_names[i] for i in range(len(prediction[0])) if prediction[0][i] == 1]  
  
 if predicted\_labels:  
 for label in predicted\_labels:  
 st.write(f"✔️ {label}")  
 else:  
 st.warning("No labels predicted for this input.")

**Description**: This is the app.py file used to deploy the trained XGBoost model in a Streamlit web application. It allows users to input features and get predicted multi-label outputs, facilitating real-time model inference through a user-friendly interface.