#######**Feature Selection Code**########

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

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, accuracy\_score

from sklearn.metrics import confusion\_matrix, multilabel\_confusion\_matrix

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Initialize Metrics

Accuracy, Sensitivity, Specificity, Precision, Fscore = [], [], [], [], []

Featurecount = 25

fold = 1

# Convert Multilabel to Single Class Labels (if needed)

if Y.shape[1] > 1:

    Y\_multiclass = np.argmax(Y, axis=1)  # Converts one-hot encoding to class labels

else:

    Y\_multiclass = Y.ravel()  # If already categorical, just flatten

# K-fold Cross-validation

kFold = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

# Initialize Gene Names

GeneName = metadata2.columns[1:-1].tolist()  # Exclude 'Gene Symbol' and 'Class'

# Ensure GeneName aligns with X

if len(GeneName) != X.shape[1]:

    GeneName = GeneName[:X.shape[1]] + [f"Feature\_{i}" for i in range(len(GeneName), X.shape[1])]

original\_GeneName = list(GeneName)

for train\_index, test\_index in kFold.split(X, Y\_multiclass):

    X\_train, X\_test = X[train\_index], X[test\_index]

    Y\_train, Y\_test = Y\_multiclass[train\_index], Y\_multiclass[test\_index]

    # Convert Y\_train and Y\_test to 1D for SVM

    Y\_train = np.array(Y\_train).reshape(-1)

    Y\_test = np.array(Y\_test).reshape(-1)

    # Train SVM

    model = SVC(kernel="linear", C=1.0, probability=True, random\_state=42)

    model.fit(X\_train, Y\_train)

    # Feature Importance (SVM uses coef\_)

    if hasattr(model, "coef\_"):

        importances = np.abs(model.coef\_).mean(axis=0)

        feature\_scores = pd.Series(importances, index=original\_GeneName).nlargest(Featurecount).to\_frame(name="Importance")

    else:

        feature\_scores = pd.DataFrame(index=original\_GeneName, columns=["Importance"])

    print(f"Feature Importance (Fold {fold}):\n", feature\_scores)

    # Visualization of Feature Importance

    plt.figure(figsize=(10, 5))

    plt.bar(feature\_scores.index, feature\_scores["Importance"])

    plt.xticks(rotation="vertical")

    plt.title(f"Fold-{fold} Feature Importance")

    plt.xlabel("Gene Name")

    plt.ylabel("Feature Importance")

    plt.show()

    # Select Top Features

    top\_features = feature\_scores.index.tolist()

    top\_feature\_indices = [original\_GeneName.index(name) for name in top\_features]

    select\_X\_train, select\_X\_test = X\_train[:, top\_feature\_indices], X\_test[:, top\_feature\_indices]

    # Retrain SVM with Selected Features

    model.fit(select\_X\_train, Y\_train)

    prediction = model.predict(select\_X\_test)

    # Compute Confusion Matrix for Multilabel Classification

    cm = multilabel\_confusion\_matrix(Y\_test, prediction)

    # Compute Metrics

    for i in range(cm.shape[0]):  # Iterate through labels

        TP = cm[i, 1, 1]

        TN = cm[i, 0, 0]

        FP = cm[i, 0, 1]

        FN = cm[i, 1, 0]

        ACC = (TP + TN) / (TP + TN + FP + FN + 1e-9)  # Avoid division by zero

        SEN = TP / (TP + FN + 1e-9)

        SPE = TN / (TN + FP + 1e-9)

        PRE = TP / (TP + FP + 1e-9)

        FS = 2 \* PRE \* SEN / (PRE + SEN + 1e-9)

        Accuracy.append(np.around(ACC, 3))

        Sensitivity.append(np.around(SEN, 3))

        Specificity.append(np.around(SPE, 3))

        Precision.append(np.around(PRE, 3))

        Fscore.append(np.around(FS, 3))

        print(f"Fold {fold}, Label {i + 1} Metrics:")

        print(f"Accuracy    : {ACC}")

        print(f"Sensitivity : {SEN}")

        print(f"Specificity : {SPE}")

        print(f"Precision   : {PRE}")

        print(f"F-score     : {FS}")

    fold += 1

print(" Final Gene Names:", GeneName)