import librosa  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error  
import pandas as pd  
import os  
import glob  
from reportlab.lib.pagesizes import letter  
from reportlab.pdfgen import canvas  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score  
from sklearn.metrics import roc\_curve, roc\_auc\_score  
import matplotlib.pyplot as plt  
from sklearn.utils.class\_weight import compute\_class\_weight  
import matplotlib.pyplot as plt  
import seaborn as sns  
from scipy.stats import shapiro  
  
from holoviews.ipython import display  
  
import pandas as pd  
import matplotlib.pyplot as plt  
import pandas as pd  
import os  
import librosa  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score, f1\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score, f1\_score, roc\_curve, auc, precision\_recall\_curve  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
  
import numpy as np  
import matplotlib.pyplot as plt  
import scipy.stats as stats  
from statsmodels.graphics.gofplots import qqplot  
from scipy.stats import shapiro  
  
import numpy as np  
import pandas as pd  
from scipy.cluster.hierarchy import dendrogram, linkage  
from matplotlib import pyplot as plt  
  
import numpy as np  
import pandas as pd  
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster  
  
import numpy as np  
import pandas as pd  
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster  
from matplotlib import pyplot as plt  
  
import numpy as np  
import pandas as pd  
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster  
from matplotlib import pyplot as plt  
import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
from sklearn.preprocessing import StandardScaler  
import os  
import librosa  
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
from sklearn.preprocessing import StandardScaler  
  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
from sklearn.preprocessing import StandardScaler  
  
from sklearn.preprocessing import StandardScaler  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from time import time  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
from sklearn.preprocessing import StandardScaler  
from sklearn import datasets, tree  
from sklearn.model\_selection import GridSearchCV  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
import pandas as pd  
import numpy as np  
import os  
import librosa  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
  
# Importing necessary libraries  
import numpy as np  
import pandas as pd  
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, BaggingClassifier  
from sklearn.model\_selection import GridSearchCV, train\_test\_split  
from sklearn.feature\_selection import SelectFromModel  
  
from sklearn.model\_selection import GridSearchCV, train\_test\_split  
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, GradientBoostingClassifier  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import Pipeline  
  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
  
  
#DATA  
  
def print\_to\_pdf(data, plots, pdf\_file):  
 # Create a canvas  
 c = canvas.Canvas(pdf\_file, pagesize=letter)  
  
 # Set font  
 c.setFont("Helvetica", 12)  
  
 # Write data to PDF  
 y = 750 # Initial y-coordinate  
 for lines in data: # Iterate over lines in data  
 for line in lines: # Iterate over lines in each list  
 c.drawString(50, y, line)  
 y -= 15 # Move to next line  
  
 # Draw plots  
 for plot in plots:  
 # Save each plot individually  
 plot.savefig(pdf\_file, format='pdf')  
 c.showPage() # Add a new page for each plot  
  
 # Save the PDF  
 c.save()  
  
  
  
path = r'C:\Users\חנא 2\Desktop\MSc statistics\2023-2024\statistical learning\csv'  
csv\_files = glob.glob(os.path.join(path, "\*.csv"))  
  
# Creating an empty list to store DataFrames from all files  
all\_dfs = []  
  
# Define the expected column names  
expected\_columns = ['Transcriptions',  
 'Transcriptions generated',  
 'Time utterance begins',  
 'Time utterance ends',  
 'contains stutter',  
 'interjection stutter',  
 'no interjection stutter',  
 'stutter types']  
  
  
# Loop over CSV files  
for f in csv\_files:  
 try:  
 # Read the CSV file with the expected column names  
 df = pd.read\_csv(f, encoding='utf-8', names=expected\_columns)  
 except UnicodeDecodeError:  
 try:  
 df = pd.read\_csv(f, encoding='latin1', names=expected\_columns)  
 except Exception as e:  
 print(f"Error reading file {f}: {e}")  
 continue  
  
 # Append DataFrame to the list  
 all\_dfs.append(df)  
  
# Concatenate all DataFrames into one large DataFrame  
big\_df = pd.concat(all\_dfs, ignore\_index=True)  
  
# Rename columns  
 df.columns = ['Transcriptions',  
 'Transcriptions generated',  
 'Time utterance begins',  
 'Time utterance ends',  
 'contains stutter',  
 'interjection stutter',  
 'no interjection stutter',  
 'stutter types']  
  
  
#EDA  
  
# Basic Data Inspection  
print("Data Types:")  
print(big\_df.dtypes)  
  
# Handling missing values (if necessary)  
# For demonstration, we'll fill missing values with a placeholder  
big\_df.fillna("Unknown", inplace=True)  
  
# Descriptive Statistics  
print("\nSummary of 'contains stutter':")  
print(big\_df['contains stutter'].value\_counts())  
  
# Data Visualization  
# Histogram of stutter occurrences  
  
plt.figure(figsize=(8, 6))  
sns.countplot(data=big\_df, x='contains stutter')  
plt.title('Distribution of Stutter Occurrences')  
plt.xlabel('Contains Stutter')  
plt.ylabel('Count')  
plt.show()  
  
  
  
# Distribution of utterance times  
plt.figure(figsize=(12, 6))  
plt.hist(big\_df['Time utterance ends'] - big\_df['Time utterance begins'], bins=50, alpha=0.75)  
plt.title('Distribution of Utterance Duration')  
plt.xlabel('Duration (seconds)')  
plt.ylabel('Frequency')  
plt.show()  
  
# Correlation heatmap for numerical columns  
plt.figure(figsize=(10, 8))  
numerical\_cols = big\_df.select\_dtypes(include=['float64', 'int64']).columns  
corr\_matrix = big\_df[numerical\_cols].corr()  
sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')  
plt.title('Correlation Heatmap')  
plt.show()  
  
  
  
  
  
# Scatter plot of utterance start vs end times  
plt.figure(figsize=(10, 6))  
plt.scatter(big\_df['Time utterance begins'], big\_df['Time utterance ends'], alpha=0.5)  
plt.title('Scatter Plot of Utterance Start vs. End Times')  
plt.xlabel('Time Utterance Begins')  
plt.ylabel('conta')  
plt.show()  
  
  
  
  
  
  
  
  
  
  
  
  
#csv MODEL  
  
utterance\_length = big\_df['Time utterance ends'] - big\_df['Time utterance begins']  
  
# Filter data for 'contains stutter' column where value is True or False  
contains\_stutter\_true = big\_df.loc[big\_df['contains stutter'] == True, 'Time utterance ends'] - big\_df.loc[big\_df['contains stutter'] == True, 'Time utterance begins']  
contains\_stutter\_false = big\_df.loc[big\_df['contains stutter'] == False, 'Time utterance ends'] - big\_df.loc[big\_df['contains stutter'] == False, 'Time utterance begins']  
  
# Create Box Plot  
plt.figure(figsize=(10, 6))  
plt.boxplot([contains\_stutter\_false, contains\_stutter\_true], labels=['False', 'True'])  
plt.title('Relationship between Utterance Length and "Contains Stutter"')  
plt.xlabel('Contains Stutter')  
plt.ylabel('Utterance Length')  
plt.grid(True)  
plt.tight\_layout()  
plt.show()  
  
  
  
# Assuming 'big\_df' contains all your data  
X = big\_df[['Time utterance ends', 'Time utterance begins']] # Add your features here  
  
# Map non-integer values in 'contains stutter' column to integers  
y = big\_df['contains stutter'].map({'TRUE': 1, 'FALSE': 0})  
  
# Drop rows with missing values in 'contains stutter' column  
y.dropna(inplace=True)  
  
# Ensure X and y have the same number of samples  
X = X.iloc[:len(y), :]  
y = y.iloc[:len(X)]  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Random Forest classifier  
classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
# Train the classifier  
classifier.fit(X\_train, y\_train)  
  
# Predictions  
y\_pred = classifier.predict(X\_test)  
  
# Evaluate the model  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Accuracy:", accuracy)  
  
  
# Calculate confusion matrix  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
print("Confusion Matrix:")  
print(conf\_matrix)  
  
#True Positives (TP) = 46  
#False Positives (FP) = 1  
#True Negatives (TN) = 0  
#False Negatives (FN) = 2  
  
#precision:  
#(true positive)/ (true positives + false positives)=  
precision = 46/(46+1)  
print(precision)  
  
#recall(sensitivity):  
#(true positive)/ (true positives + false negatives)=  
recall = 46/(46+2)  
print(recall)  
  
# Calculate F1-score  
f1 = 2 \* (precision \* recall) / (precision + recall)  
print("F1-score:", f1)  
  
  
  
  
#audio  
  
  
  
# Paths  
path = r'C:\Users\חנא 2\Desktop\MSc statistics\2023-2024\statistical learning\csv'  
path\_audio = r'C:\Users\חנא 2\Desktop\MSc statistics\2023-2024\statistical learning\audio'  
  
# Expected columns in CSV files  
col = ['Transcriptions',  
 'Transcriptions generated',  
 'Time utterance begins',  
 'Time utterance ends',  
 'contains stutter',  
 'interjection stutter',  
 'no interjection stutter',  
 'stutter types']  
  
# Initialize DataFrame for metadata and list for audio data  
meta\_data = pd.DataFrame(columns=col)  
audio\_data = []  
mfcc\_features = []  
labels = []  
  
# Process CSV and audio files  
for i in os.listdir(path):  
 try:  
 df\_temp = pd.read\_csv(os.path.join(path, i), header=None, encoding='latin1')  
 df\_temp.columns = col  
 filename = i[:-3] + 'wav'  
  
 if os.path.isfile(os.path.join(path\_audio, filename)):  
 try:  
 audio, sample\_rate = librosa.load(os.path.join(path\_audio, filename), sr=None)  
 audio\_data.append(audio)  
  
 # Extract MFCCs  
 mfcc = librosa.feature.mfcc(y=audio, sr=sample\_rate, n\_mfcc=13)  
 mfcc\_mean = np.mean(mfcc.T, axis=0) # Use the mean of MFCCs as features  
 mfcc\_features.append(mfcc\_mean)  
  
 # Extract the label for 'contains stutter'  
 label = df\_temp['contains stutter'].values[0]  
 labels.append(1 if label else 0)  
  
 except Exception as err:  
 print(f'Error loading audio file: {filename}')  
 print(err)  
 continue  
 else:  
 print(f'Audio file does not exist: {filename}')  
 continue  
  
 # Append df\_temp to meta\_data using concat  
 meta\_data = pd.concat([meta\_data, df\_temp], ignore\_index=True)  
  
 except Exception as err:  
 print(f'Problem with file: {i}')  
 print(err)  
 continue  
  
# Plot audio waveforms  
for idx, audio in enumerate(mfcc):  
 plt.figure(figsize=(10, 4))  
 plt.plot(audio)  
 plt.title(f'Waveform for Audio {idx}')  
 plt.xlabel('Sample')  
 plt.ylabel('Amplitude')  
 plt.show()  
  
# Plot histograms for audio data  
for idx, audio in enumerate(mfcc):  
 plt.figure(figsize=(8, 6))  
 plt.hist(audio, bins=100, color='blue', alpha=0.7)  
 plt.title(f'Histogram for Audio {idx}')  
 plt.xlabel('Amplitude')  
 plt.ylabel('Frequency')  
 plt.show()  
  
  
# Convert lists to numpy arrays  
mfcc\_features = np.array(mfcc\_features)  
labels = np.array(labels)  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(mfcc\_features, labels, test\_size=0.2, random\_state=42)  
  
# Random Forest classifier  
classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
# Train the classifier  
classifier.fit(X\_train, y\_train)  
  
# Predictions  
y\_pred = classifier.predict(X\_test)  
  
# Evaluate the model  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred, zero\_division=0)  
recall = recall\_score(y\_test, y\_pred, zero\_division=0)  
f1 = f1\_score(y\_test, y\_pred, zero\_division=0)  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
  
print("Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1-score:", f1)  
print("Confusion Matrix:")  
print(conf\_matrix)  
  
  
  
  
  
#Distribution of all columns of data by clasters:  
# Paths  
path = r'C:\Users\חנא 2\Desktop\MSc statistics\2023-2024\statistical learning\csv'  
path\_audio = r'C:\Users\חנא 2\Desktop\MSc statistics\2023-2024\statistical learning\audio'  
  
# Expected columns in CSV files  
col = ['Transcriptions', 'Transcriptions generated', 'Time utterance begins', 'Time utterance ends', 'contains stutter', 'interjection stutter', 'no interjection stutter', 'stutter types']  
  
# Initialize DataFrame for metadata and list for audio data  
meta\_data = pd.DataFrame(columns=col)  
audio\_data = []  
mfcc\_features = []  
labels = []  
  
# Process CSV and audio files  
for i in os.listdir(path):  
 try:  
 df\_temp = pd.read\_csv(os.path.join(path, i), header=None, encoding='latin1')  
 df\_temp.columns = col  
 filename = i[:-3] + 'wav'  
  
 if os.path.isfile(os.path.join(path\_audio, filename)):  
 try:  
 audio, sample\_rate = librosa.load(os.path.join(path\_audio, filename), sr=None)  
 audio\_data.append(audio)  
  
 # Extract MFCCs  
 mfcc = librosa.feature.mfcc(y=audio, sr=sample\_rate, n\_mfcc=13)  
 mfcc\_mean = np.mean(mfcc.T, axis=0) # Use the mean of MFCCs as features  
 mfcc\_features.append(mfcc\_mean)  
  
 # Extract the label for 'contains stutter'  
 label = df\_temp['contains stutter'].values[0]  
 labels.append(1 if label else 0)  
  
 except Exception as err:  
 print(f'Error loading audio file: {filename}')  
 print(err)  
 continue  
 else:  
 print(f'Audio file does not exist: {filename}')  
 continue  
  
 # Append df\_temp to meta\_data using concat  
 meta\_data = pd.concat([meta\_data, df\_temp], ignore\_index=True)  
  
 except Exception as err:  
 print(f'Problem with file: {i}')  
 print(err)  
 continue  
  
# Standardize 'contains stutter' values  
big\_df['contains stutter'] = big\_df['contains stutter'].astype(str).str.lower()  
  
# Map 'true' and 'false' to 1 and 0  
big\_df['contains stutter'] = big\_df['contains stutter'].map({'true': 1, 'false': 0})  
  
# Check the results  
print(big\_df['contains stutter'].value\_counts())  
  
# Calculate utterance length  
big\_df['utterance\_length'] = big\_df['Time utterance ends'] - big\_df['Time utterance begins']  
  
# Select relevant columns for clustering  
relevant\_df = big\_df[['utterance\_length', 'contains stutter']].copy()  
  
# Replace non-finite values with NaN  
relevant\_df.replace([np.inf, -np.inf], np.nan, inplace=True)  
  
# Fill NaN values with the mean of the column  
relevant\_df['utterance\_length'].fillna(relevant\_df['utterance\_length'].mean(), inplace=True)  
relevant\_df['contains stutter'].fillna(relevant\_df['contains stutter'].mean(), inplace=True)  
  
# Ensure the DataFrame is not empty  
if relevant\_df.empty:  
 print("DataFrame is empty after removing non-finite values. Here's the original data for inspection:")  
 print(big\_df.head())  
 raise ValueError("DataFrame is empty after removing non-finite values. Check your data cleaning steps.")  
  
# Ensure no non-finite values remain  
assert np.all(np.isfinite(relevant\_df.values)), "Data contains non-finite values."  
  
# Perform hierarchical clustering  
linked = linkage(relevant\_df, method='ward')  
  
# Plotting the dendrogram (for visualization purpose)  
plt.figure(figsize=(10, 7))  
dendrogram(linked, orientation='top', distance\_sort='descending', show\_leaf\_counts=True)  
plt.title('Hierarchical Clustering Dendrogram')  
plt.xlabel('Sample index')  
plt.ylabel('Distance')  
plt.show()  
  
# Assign cluster labels (choosing 3 clusters as per your requirement)  
cluster\_labels = fcluster(linked, t=3, criterion='maxclust')  
  
# Add cluster labels to the original DataFrame  
big\_df['Cluster'] = np.nan  
big\_df.loc[relevant\_df.index, 'Cluster'] = cluster\_labels  
  
# Filter for the first three clusters  
first\_three\_clusters = big\_df[big\_df['Cluster'].isin([1, 2, 3])]  
  
# Define custom aggregation functions for the summary table  
agg\_funcs = {  
 'utterance\_length': 'count',  
 'contains stutter': 'sum',  
 'stutter types': 'count'  
}  
  
# Create a summary table showing the count of each column in the clusters  
summary\_table = first\_three\_clusters.groupby('Cluster').agg(agg\_funcs)  
  
# Display the summary table  
print("Summary of utterance\_length, contains stutter, and stutter types by clusters:")  
print(summary\_table)  
  
# Optionally, save the summary table to a CSV file  
summary\_table.to\_csv('cluster\_summary\_table.csv')  
  
# Plot the summary data for visualization  
fig, axes = plt.subplots(3, 1, figsize=(10, 15))  
  
# Plot utterance\_length  
axes[0].bar(summary\_table.index, summary\_table['utterance\_length'], color='skyblue')  
axes[0].set\_title('Utterance Length by Cluster')  
axes[0].set\_xlabel('Cluster')  
axes[0].set\_ylabel('Count')  
axes[0].set\_xticks(summary\_table.index)  
  
# Plot contains stutter  
axes[1].bar(summary\_table.index, summary\_table['contains stutter'], color='salmon')  
axes[1].set\_title('Contains Stutter by Cluster')  
axes[1].set\_xlabel('Cluster')  
axes[1].set\_ylabel('Count')  
axes[1].set\_xticks(summary\_table.index)  
  
# Plot stutter types  
axes[2].bar(summary\_table.index, summary\_table['stutter types'], color='lightgreen')  
axes[2].set\_title('Stutter Types by Cluster')  
axes[2].set\_xlabel('Cluster')  
axes[2].set\_ylabel('Count')  
axes[2].set\_xticks(summary\_table.index)  
  
plt.tight\_layout()  
plt.show()  
  
  
  
  
  
#model on cluster 2.0:  
  
# Filter data for Cluster 2 and create a deep copy  
cluster\_2\_data = big\_df[big\_df['Cluster'] == 2.0].copy(deep=True)  
  
# Prepare features and target  
X = cluster\_2\_data[['Time utterance ends', 'Time utterance begins']].copy()  
y = cluster\_2\_data['contains stutter'].copy()  
  
# Ensure no missing values  
X.fillna(X.mean(), inplace=True)  
y.fillna(y.mean(), inplace=True)  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Initialize Random Forest classifier  
classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
# Train the classifier  
classifier.fit(X\_train, y\_train)  
  
# Predictions  
y\_pred = classifier.predict(X\_test)  
  
# Evaluate the model  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred, zero\_division=0)  
recall = recall\_score(y\_test, y\_pred, zero\_division=0)  
f1 = f1\_score(y\_test, y\_pred, zero\_division=0)  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
  
print("Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1-score:", f1)  
print("Confusion Matrix:")  
print(conf\_matrix)  
  
  
  
  
  
  
#after overfit:  
# Check distribution of target variable in all clusters  
print(big\_df.groupby('Cluster')['contains stutter'].value\_counts())  
  
# Choose clusters that have both classes (0 and 1) and combine them  
selected\_clusters = [1, 3] # Example: choose clusters 1 and 3 if they have both classes  
combined\_data = big\_df[big\_df['Cluster'].isin(selected\_clusters)].copy(deep=True)  
  
# Prepare features and target  
X = combined\_data[['Time utterance ends', 'Time utterance begins']].copy()  
y = combined\_data['contains stutter'].copy()  
  
# Ensure no missing values  
X.fillna(X.mean(), inplace=True)  
y.fillna(y.mean(), inplace=True)  
  
# Convert target variable to categorical if necessary  
y = y.astype(int)  
  
# Ensure target variable has the expected values  
print(y.value\_counts())  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Apply SMOTE to handle class imbalance in the training set  
smote = SMOTE(random\_state=42)  
X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train, y\_train)  
  
# Initialize Logistic Regression with regularization  
lr = LogisticRegression(random\_state=42, max\_iter=1000)  
  
# Define parameter grid for hyperparameter tuning  
param\_grid = {  
 'C': [0.1, 1.0, 10.0],  
 'penalty': ['l1', 'l2'],  
 'solver': ['liblinear']  
}  
  
# Perform grid search with cross-validation  
grid\_search = GridSearchCV(estimator=lr, param\_grid=param\_grid, cv=5, n\_jobs=-1, verbose=2)  
grid\_search.fit(X\_train\_resampled, y\_train\_resampled)  
  
# Get the best parameters and best estimator  
best\_params = grid\_search.best\_params\_  
best\_classifier = grid\_search.best\_estimator\_  
  
# Perform cross-validation on the training set  
cv\_scores = cross\_val\_score(best\_classifier, X\_train\_resampled, y\_train\_resampled, cv=5)  
  
# Train the best estimator on the entire training set  
best\_classifier.fit(X\_train\_resampled, y\_train\_resampled)  
  
# Predictions  
y\_pred = best\_classifier.predict(X\_test)  
  
# Evaluate the model  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred, zero\_division=0)  
recall = recall\_score(y\_test, y\_pred, zero\_division=0)  
f1 = f1\_score(y\_test, y\_pred, zero\_division=0)  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
  
print("Best Parameters:", best\_params)  
print("Cross-Validation Scores:", cv\_scores)  
print("Mean CV Accuracy:", cv\_scores.mean())  
print("Accuracy:", accuracy)  
print("Precision:", precision)  
print("Recall:", recall)  
print("F1-score:", f1)  
print("Confusion Matrix:")  
print(conf\_matrix)  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
from pptx import Presentation  
from pptx.util import Inches  
  
# Create a presentation object  
prs = Presentation()  
  
# Slide 1: Title Slide  
slide\_layout = prs.slide\_layouts[0]  
slide = prs.slides.add\_slide(slide\_layout)  
title = slide.shapes.title  
subtitle = slide.placeholders[1]  
title.text = "Statistical Learning Project: Classification of Stuttered Speech"  
subtitle.text = "Due: June 15, 2024\nMoran Hasson - 318177813"  
  
# Slide 2: Summary of Findings  
slide\_layout = prs.slide\_layouts[1]  
slide = prs.slides.add\_slide(slide\_layout)  
title = slide.shapes.title  
title.text = "Summary of Findings"  
content = slide.placeholders[1]  
content.text = (  
 "- Achieved 99.34% accuracy using Random Forest classifier\n"  
 "- MFCCs (Mel-frequency cepstral coefficients) were used as primary features\n"  
 "- Successfully balanced classes using SMOTE technique"  
)  
  
# Slide 3: Feature Engineering and Model Selection  
slide = prs.slides.add\_slide(slide\_layout)  
title = slide.shapes.title  
title.text = "Feature Engineering and Model Selection"  
content = slide.placeholders[1]  
content.text = (  
 "- Extracted MFCCs for audio feature representation\n"  
 "- Hierarchical clustering identified 3 clusters; Cluster 2.0 chosen for modeling\n"  
 "- Random Forest classifier chosen for its effectiveness with the given data"  
)  
  
# Slide 4: Addressing Overfitting  
slide = prs.slides.add\_slide(slide\_layout)  
title = slide.shapes.title  
title.text = "Addressing Overfitting"  
content = slide.placeholders[1]  
content.text = (  
 "- Used SMOTE to handle class imbalance\n"  
 "- Hyperparameter tuning with GridSearchCV\n"  
 "- Evaluated model with cross-validation, achieving improved metrics"  
)  
  
# Slide 5: Results and Future Work  
slide = prs.slides.add\_slide(slide\_layout)  
title = slide.shapes.title  
title.text = "Results and Future Work"  
content = slide.placeholders[1]  
content.text = (  
 "- Precision: 0.25, Recall: 1.0\n"  
 "- Model struggles with identifying stutter cases among non-stutter instances\n"  
 "- Future work: Increase dataset size, explore advanced techniques for better precision"  
)  
  
# Save the presentation  
presentation\_path = "/mnt/data/Statistical\_Learning\_Project\_Presentation.pptx"  
prs.save(presentation\_path)  
  
presentation\_path