University of New Haven

Classifying Song Genres from Audio Data

Machine Learning DSCI 6003

Team Members

Ashish Agrawal

Bikram Chand

Chandramohan Chandankumar

April 29, 2024

Table of Contents

Abstract

In this project, we tackled the challenge of classifying song genres using machine learning techniques. Despite previous attempts to use machine learning approaches for genre prediction, there's still room for improvement. We experimented with the different machine learning to identify 10 different music genre, utilizing the GTZAN dataset available on Kaggle. The findings from this research is highlty anticipated to contribute to the field of music genre classification, potentially enhancing the functionality for applications like music recommendation systems.

Introduction

In the digital age of music industry, the ability to automatically classify song genres from audio data is very crucial for various applications, including music recommendations applications and other media organization. This project focuses on developing a machine learning-based classification system to accurately predict the genre labels of songs.

This report outlines into the development of classification methods aimed at identifying distinct music genres from audio features. It begins with an overview of the dataset utilized and the preprocessing techniques applied. Additionally, the report dives into the description of various models employed, including Random Forest Classifier (RFC), K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), and Gaussian Naive Bayes (GaussianNB). Each model's classification accuracy is thoroughly evaluated, followed by an in-depth error analysis. Finally, the report concludes with recommendations for future research.

Methods

Gathering Data

To gather data for this report, we obtained the dataset from GTZAN available on Kaggle. This dataset comprises 1000 audio tracks, each lasting 30 seconds. It encompasses a diverse range of music genres, with 10 genres represented, and each genre (blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, rock) is represented by 100 tracks. The audio files are all encoded at a sampling rate of 22050 Hz, ensuring high-quality representation, and are in monophonic 16-bit audio format (.au). This dataset offers a comprehensive collection of music samples, enabling a thorough exploration of genre classification methods based on audio features.

Data Preprocessing and Feature Extraction

Prior to model training, the dataset underwent preprocessing steps to extract relevant features for classification. This involved utilizing the Librosa library to extract features such as Mel-frequency cepstral coefficients (MFCCs), chroma features, and Mel spectrograms from the audio files. These features serve as the basis for training the classification models, capturing essential characteristics of the audio signals relevant to genre classification.

Once the features were extracted, the labels associated with each audio track were encoded using LabelEncoder from scikit-learn, converting categorical genre labels into numerical values for model training. After that the the dataset was partitioned into an 80% training set and a 20% testing set. This division allowed for comprehensive model training on a majority of the data while reserving a small portion for the evaluation of model performance.

Model Development

Algorithm’s Used

- Random Forest Classifier (RFC)
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Naive Bayes

Random Forest Classifier (RFC)

The Random Forest Classifier (RFC) is like a team of decision trees working together to make predictions. Each decision tree is like a small rulebook that helps decide the outcome based on certain features. RFC combines the predictions from all these trees to come up with the final answer. It's really good at handling lots of different kinds of data and is known for being accurate and reliable.

```
"Random Forest": RandomForestClassifier(n estimators=100. random state=42).
```

Our Random Forest Classifier, instantiated with 100 decision trees, was employed to harness the ensemble learning approach. This model, facilitated by the RandomForestClassifier class, leveraged its ability to mitigate overfitting and accommodate high-dimensional data.

Support Vector Machine (SVM)

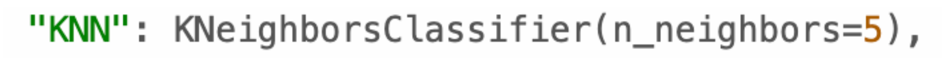
An SVM model is like drawing lines or boundaries to separate different groups of things in space that has many dimensions. The model keeps adjusting these lines to make sure it's as accurate as possible. The main aim of SVM is to find the best line that separates the different groups as much as possible, so there's as much space as possible between them.

```
"SVM": SVC(kernel='linear', random_state=42)
```

Our classification included the Support Vector Machine algorithm, configured with a linear kernel, as implemented by the SVC class. SVM's capability to construct optimal hyperplanes in higher-dimensional spaces made it a compelling choice for our genre classification endeavor.

K-Nearest Neighbors (KNN)

The K-nearest neighbors (KNN) algorithm works by looking at the features of new data points and finding the most similar data points from the training set. It then assigns a value to the new data point based on how closely it matches these similar points. Essentially, KNN finds the nearest neighbors in the training data and uses their values to predict the value of the new data point.

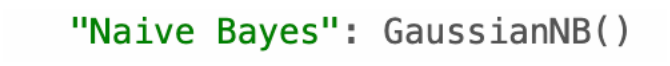


```
"KNN": KNeighborsClassifier(n_neighbors=5),
```

The K-Nearest Neighbors algorithm, instantiated with 5 neighbors, offered a non-parametric approach to classification. With the KNeighborsClassifier class, KNN demonstrated its simplicity and effectiveness in handling multi-class classification tasks.

Naive Bayes

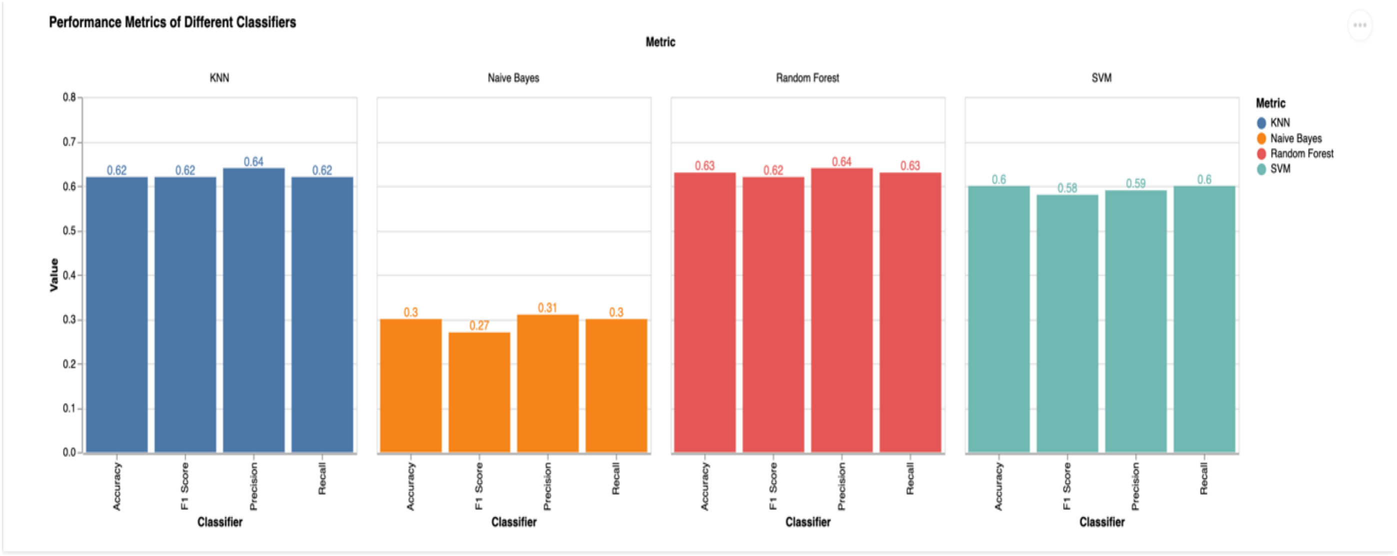
In simple terms, the Naive Bayes algorithm is like a smart guesser. It looks at the features of different things and tries to guess their category based on how often those features show up together in the data. It assumes that each feature contributes independently to the outcome, which is why it's called "naive."



```
    "Naive Bayes": GaussianNB()
```

Completing our ensemble, the Naive Bayes classifier, represented by the GaussianNB class, provided a probabilistic approach to classification.

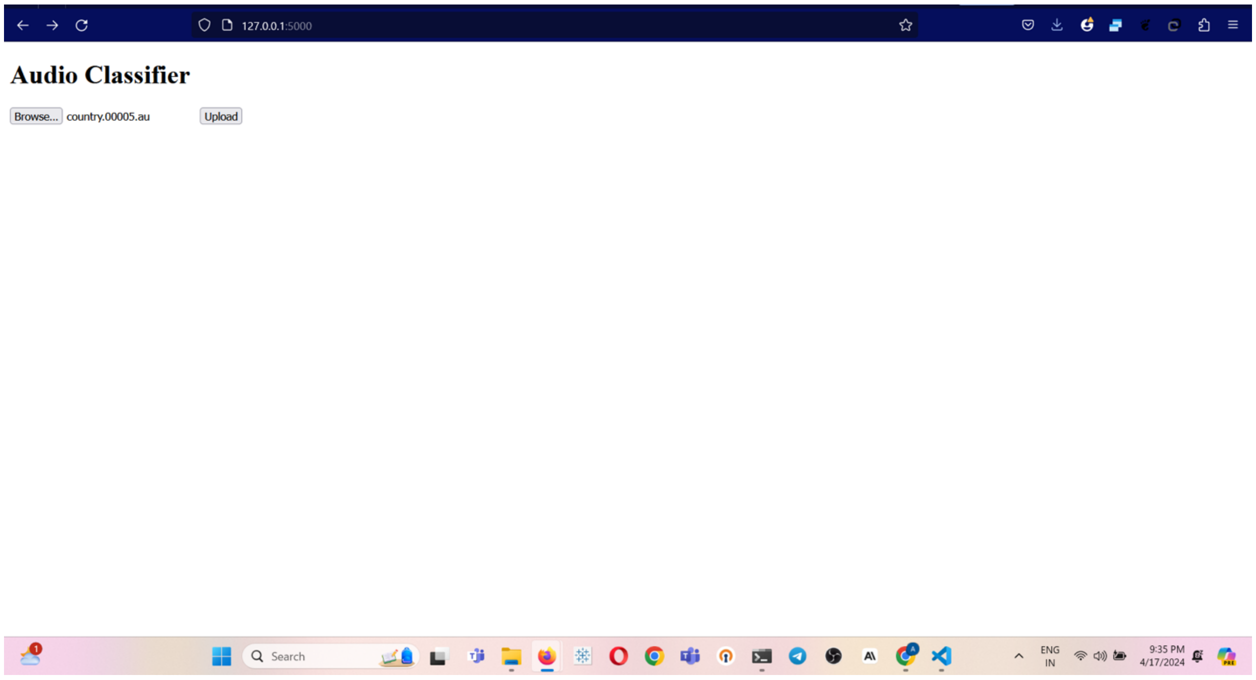
Results and Discussion

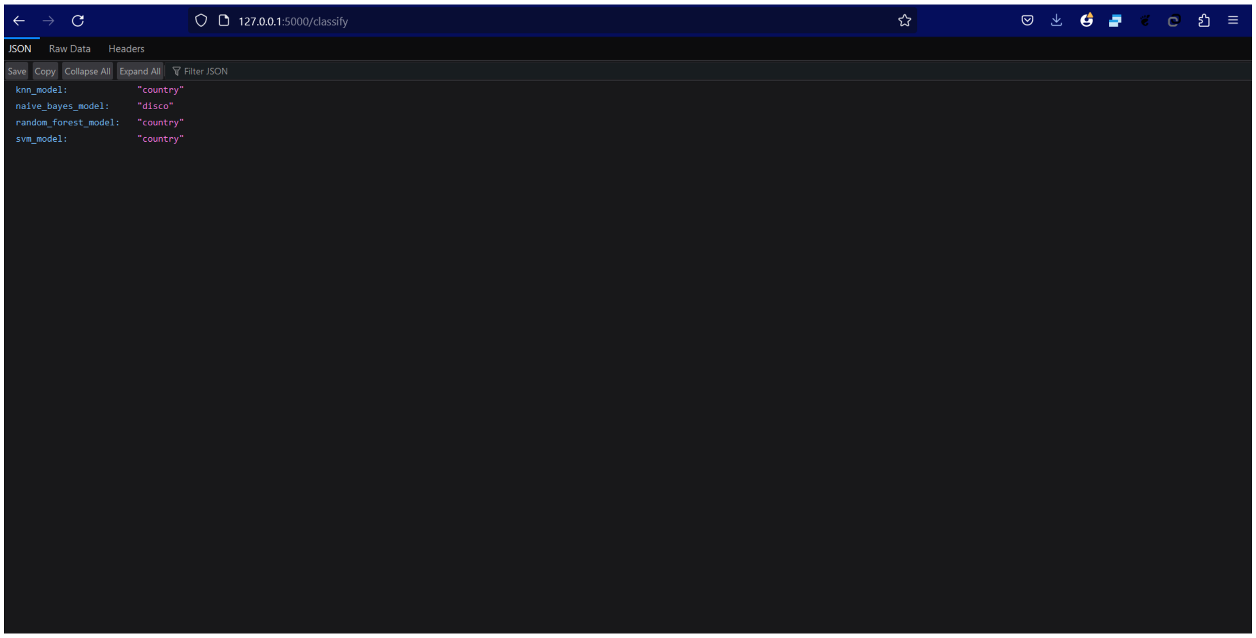


We trained various four different learning models to classify audio song genres and observed differing performance among them. The k-Nearest Neighbors (kNN) model achieved an accuracy of 62%, with corresponding F1 score, precision, and recall values of 0.62, 0.64, and 0.62 respectively. Naive Bayes exhibited lower performance with an accuracy of 30%, and F1 score, precision, and recall values of 0.27, 0.31, and 0.30 respectively. Random Forest achieved an accuracy of 63% with similar F1 score, precision, and recall values of 0.62, 0.64, and 0.63 respectively. Lastly, the Support Vector Machine (SVM) model yielded an accuracy of 60%, and corresponding F1 score, precision, and recall values of 0.58, 0.59, and 0.60 respectively. Among the models tested, the Random Forest model emerged as the most effective, demonstrating an accuracy of 63% and consistent performance across F1 score, precision, and recall metrics.

Product Demonstration

Our product features a simple user interface (UI) designed for ease of use. Users can easily upload audio files in any format they prefer. Once uploaded, our powerful models quickly analyze each file, predicting its genre. This streamlined process ensures that users can effortlessly discover the genre of their audio files without any hassle





```
knn_model:            "country"
naive_bayes_model:    "disco"
random_forest_model:  "country"
svm_model:            "country"
```

Conclusion

We implemented a variety of machine learning techniques to classify music genres using GTZAN dataset. The highest test accuracy is 63%, achieved by Random Forest Classifier. Feature were strandidized with Standard Scaler to reduce model complexity and overfitting, and improved test accuracies were observe. We also found that when we pick the right features using different methods and train the model with those important features, the model can performs much better.

Future Work

1. To further improve the accuracy, more music data and genre needs to be added to train our model.
2. To add other musically relevant features for better classification results.
3. Build real application with more more intuitive interface.
4. Eperiment with other neural network or deep learning models to examine if it could offer further improvements in classification accuracy and robustness.

Github Link

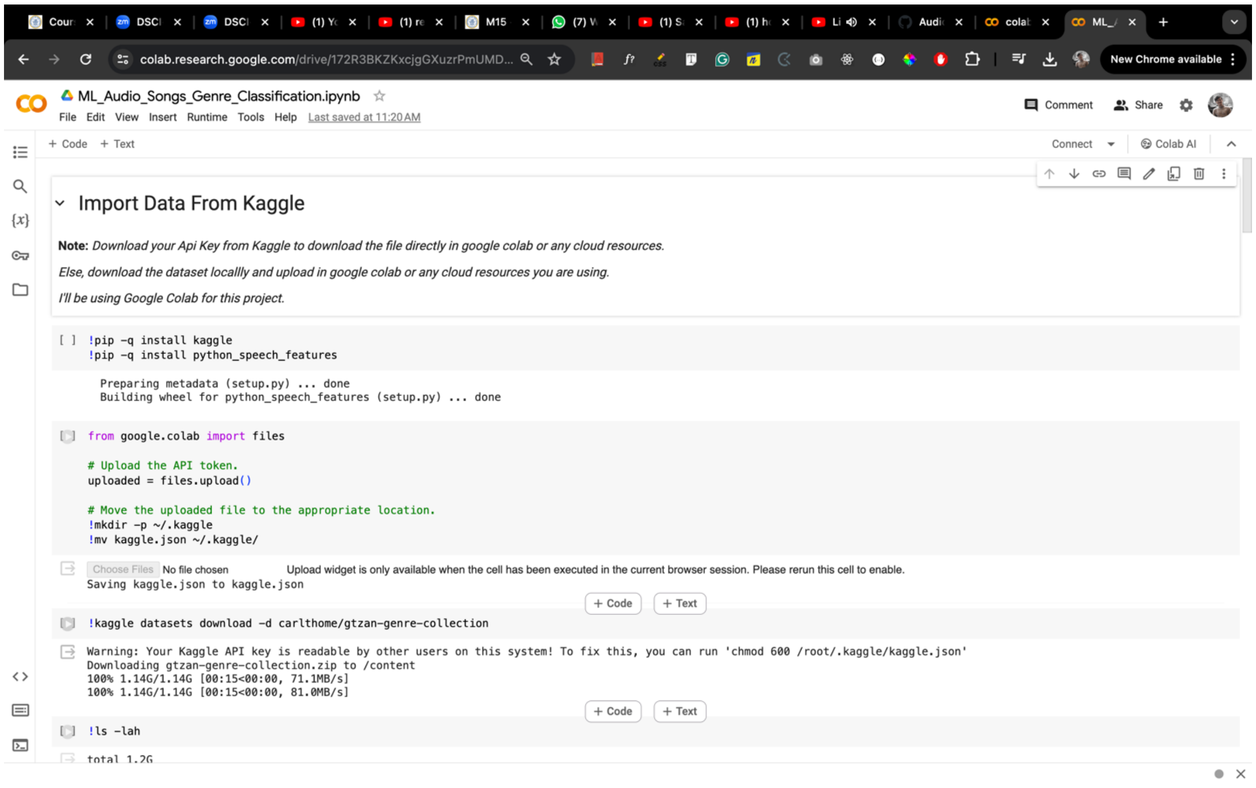
https://github.com/CRLannister/Audio_Songs_Genre_Classification_ML_6003_FP

References

- *What is the K-nearest neighbors algorithm?* (2024) *IBM*. Available at: https://www.ibm.com/topics/knn (Accessed: 28 April 2024).
- GeeksforGeeks (2023) *Support Vector Machine (SVM) algorithm*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/support-vector-machine-algorithm/ (Accessed: 28 April 2024).

Appendix

Code Screenshots


colab.research.google.com/drive/172R3BKZKxcjgGXuzrPmUMD...
ML_Audio_Songs_Genre_Classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:20 AM
Comment Share
+ Code + Text
Connect
Colab AI
New Chrome available

Import Data From Kaggle

Note: *Download your Api Key from Kaggle to download the file directly in google colab or any cloud resources.*

Else, download the dataset locallly and upload in google colab or any cloud resources you are using.

I'll be using Google Colab for this project.

```
!pip -q install kaggle
!pip -q install python_speech_features
```

```
  Preparing metadata (setup.py) ... done
  Building wheel for python_speech_features (setup.py) ... done
```

```
from google.colab import files

# Upload the API token.
uploaded = files.upload()

# Move the uploaded file to the appropriate location.
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
```

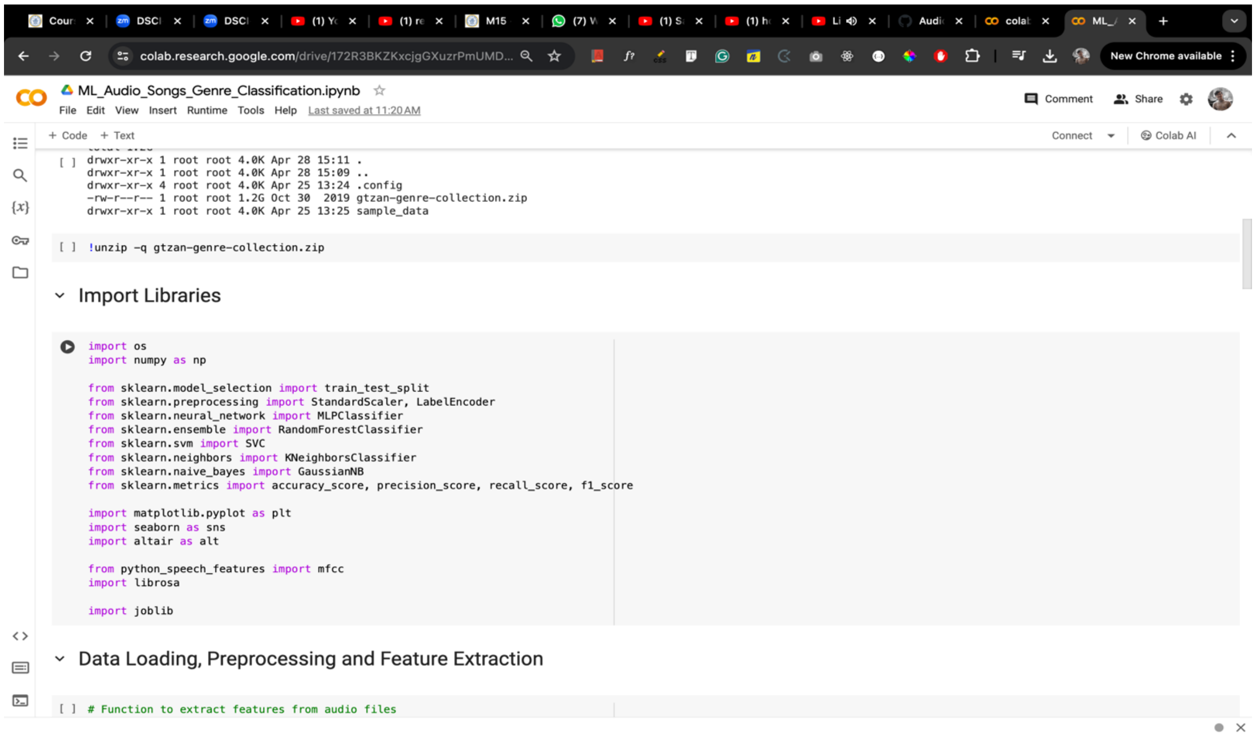
```
Choose Files No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
```

```
!kaggle datasets download -d carlthome/gtzan-genre-collection
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading gtzan-genre-collection.zip to /content
100% 1.14G/1.14G [00:15<00:00, 71.1MB/s]
100% 1.14G/1.14G [00:15<00:00, 81.0MB/s]
```

```
!ls -lah
```

```
total 1.2G
```


colab.research.google.com/drive/172R3BKZKxcjgGXuzrPmUMD...
ML_Audio_Songs_Genre_Classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:20 AM
Comment Share
+ Code + Text
Connect
Colab AI
New Chrome available

```
drwxr-xr-x 1 root root 4.0K Apr 28 15:11 .
drwxr-xr-x 1 root root 4.0K Apr 28 15:09 ..
drwxr-xr-x 4 root root 4.0K Apr 25 13:24 .config
-rw-r--r-- 1 root root 1.2G Oct 30  2019 gtzan-genre-collection.zip
drwxr-xr-x 1 root root 4.0K Apr 25 13:25 sample_data
```

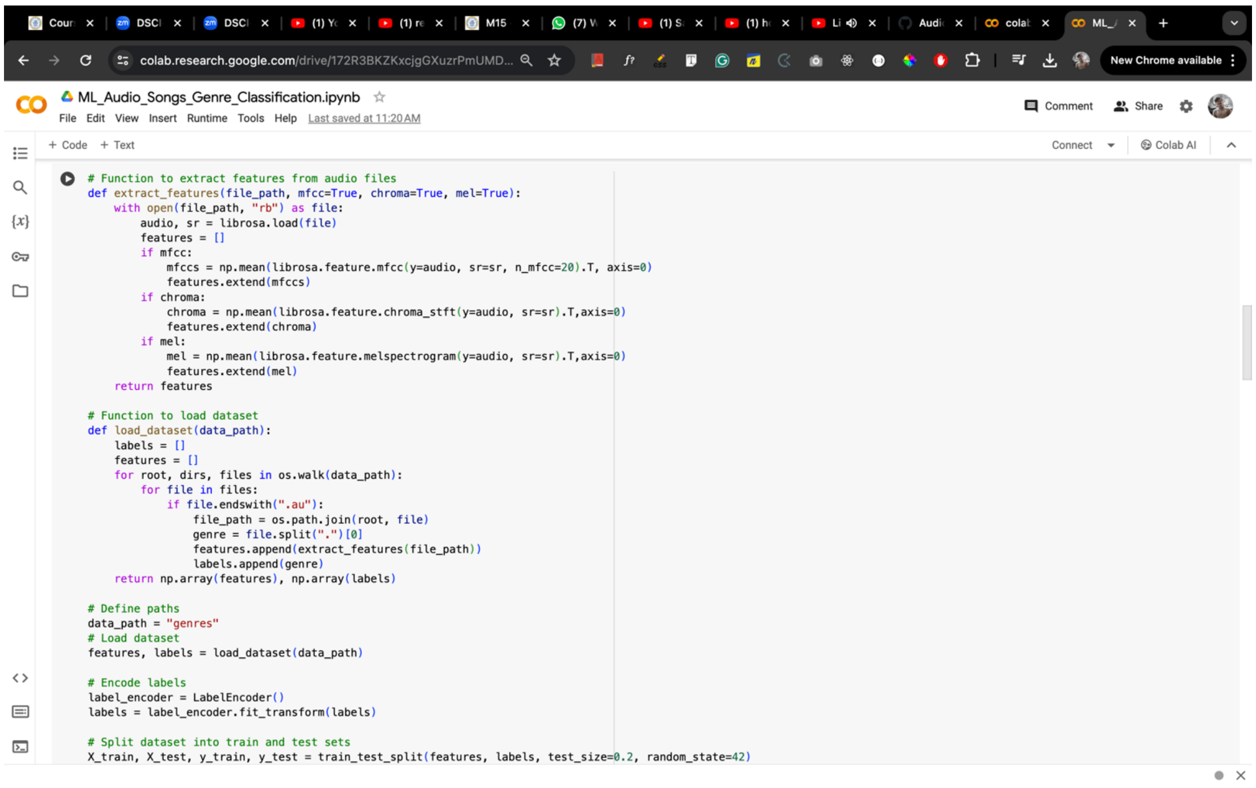
```
!unzip -q gtzan-genre-collection.zip
```

Import Libraries

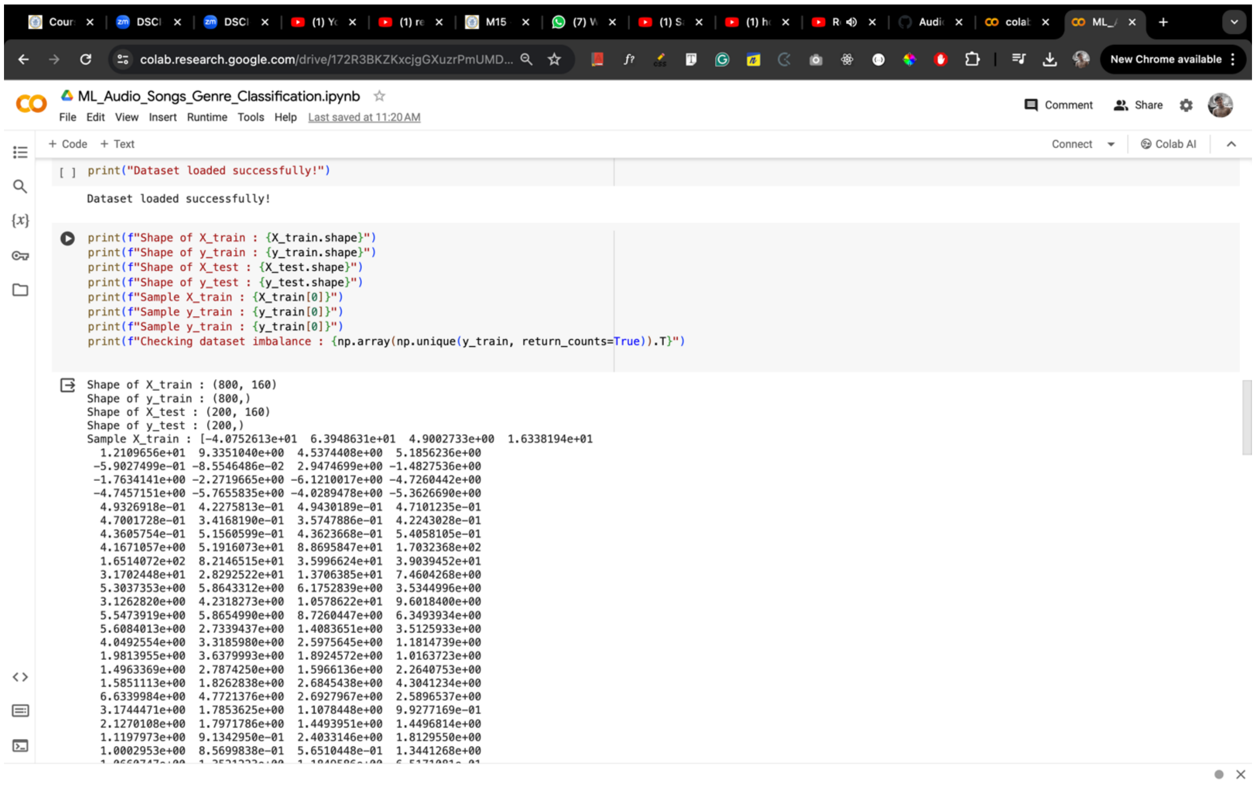
```
import os
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt

from python_speech_features import mfcc
import librosa

import joblib
```

Data Loading, Preprocessing and Feature Extraction

```
# Function to extract features from audio files
```


ML_Audio_Songs_Genre_Classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:20 AM
Comment Share
+ Code + Text
Connect Colab AI

```python
# Function to extract features from audio files
def extract_features(file_path, mfcc=True, chroma=True, mel=True):
    with open(file_path, "rb") as file:
        audio, sr = librosa.load(file)
        features = []
        if mfcc:
            mfccs = np.mean(librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=20).T, axis=0)
            features.extend(mfccs)
        if chroma:
            chroma = np.mean(librosa.feature.chroma_stft(y=audio, sr=sr).T,axis=0)
            features.extend(chroma)
        if mel:
            mel = np.mean(librosa.feature.melspectrogram(y=audio, sr=sr).T,axis=0)
            features.extend(mel)
    return features

# Function to load dataset
def load_dataset(data_path):
    labels = []
    features = []
    for root, dirs, files in os.walk(data_path):
        for file in files:
            if file.endswith(".au"):
                file_path = os.path.join(root, file)
                genre = file.split(".")[0]
                features.append(extract_features(file_path))
                labels.append(genre)
    return np.array(features), np.array(labels)

# Define paths
data_path = "genres"
# Load dataset
features, labels = load_dataset(data_path)

# Encode labels
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)

# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)
```

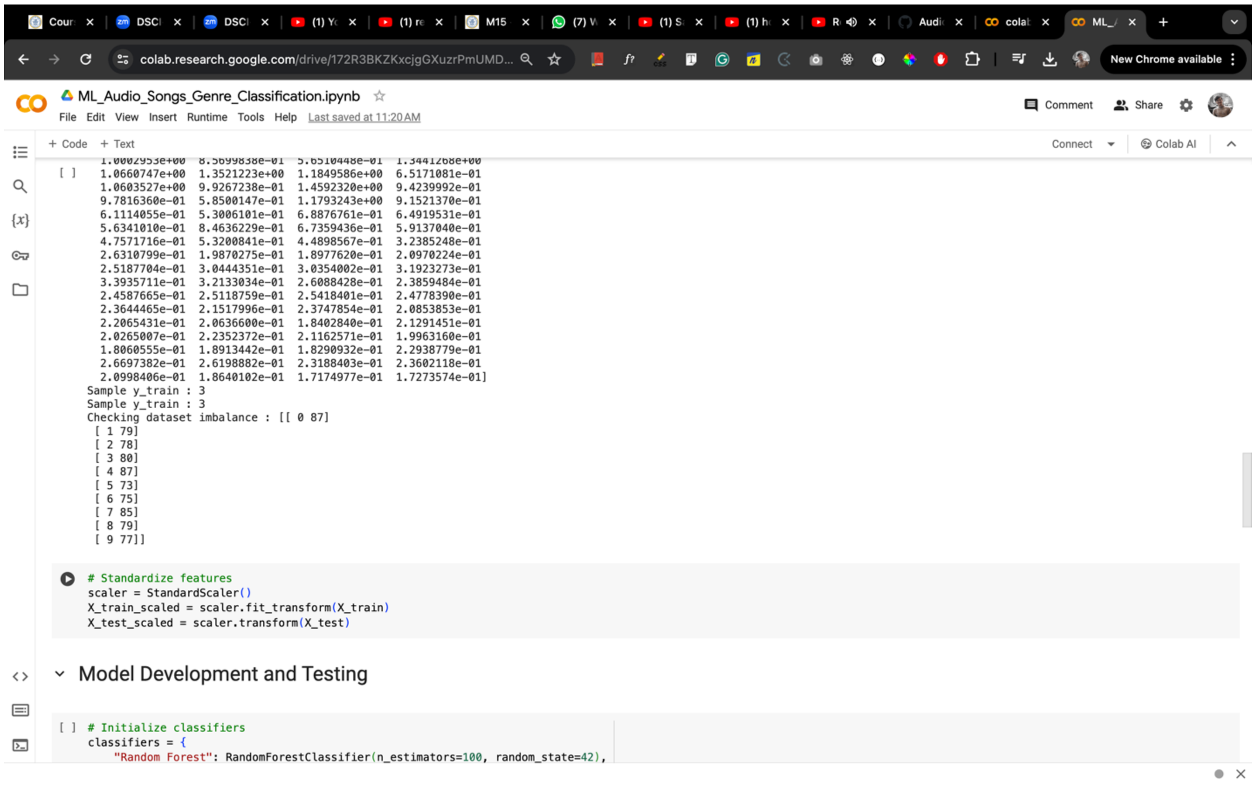

ML_Audio_Songs_Genre_Classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:20 AM
Comment Share
+ Code + Text
Connect Colab AI

```python
print("Dataset loaded successfully!")
```

```
Dataset loaded successfully!
```

```python
print(f"Shape of X_train : {X_train.shape}")
print(f"Shape of y_train : {y_train.shape}")
print(f"Shape of X_test : {X_test.shape}")
print(f"Shape of y_test : {y_test.shape}")
print(f"Sample X_train : {X_train[0]}")
print(f"Sample y_train : {y_train[0]}")
print(f"Sample y_train : {y_train[0]}")
print(f"Checking dataset imbalance : {np.array(np.unique(y_train, return_counts=True)).T}")
```

```
Shape of X_train : (800, 160)
Shape of y_train : (800,)
Shape of X_test : (200, 160)
Shape of y_test : (200,)
Sample X_train : [-4.0752613e+01  6.3948631e+01  4.9002733e+00  1.6338194e+01
  1.2109656e+01  9.3351040e+00  4.5374408e+00  5.1856236e+00
 -5.9027499e-01 -8.5546486e-02  2.9474699e+00 -1.4827536e+00
 -1.7634141e+00 -2.2719665e+00 -6.1210017e+00 -4.7260442e+00
 -4.7457151e+00 -5.7655835e+00 -4.0289478e+00 -5.3626690e+00
  4.9326918e-01  4.2275813e-01  4.9430189e-01  4.7101235e-01
  4.7001728e-01  3.4168190e-01  3.5747886e-01  4.2243028e-01
  4.3605754e-01  5.1560599e-01  4.3623668e-01  5.4058105e-01
  4.1671057e+00  5.1916073e+01  8.8695847e+01  1.7032368e+02
  1.6514072e+02  8.2146515e+01  3.5996624e+01  3.9039452e+01
  3.1702448e+01  2.8292522e+01  1.3706385e+01  7.4604268e+00
  5.3037353e+00  5.8643312e+00  6.1752839e+00  3.5344996e+00
  3.1262820e+00  4.2318273e+00  1.0578622e+01  9.6018400e+00
  5.5473919e+00  5.8654990e+00  8.7260447e+00  6.3493934e+00
  5.6084013e+00  2.7339437e+00  1.4083651e+00  3.5125933e+00
  4.0492554e+00  3.3185980e+00  2.5975645e+00  1.1814739e+00
  1.9813955e+00  3.6379993e+00  1.8924572e+00  1.0163723e+00
  1.4963369e+00  2.7874250e+00  1.5966136e+00  2.2640753e+00
  1.5851113e+00  1.8262838e+00  2.6845438e+00  4.3041234e+00
  6.6339984e+00  4.7721376e+00  2.6927967e+00  2.5896537e+00
  3.1744471e+00  1.7853625e+00  1.1078448e+00  9.9277169e-01
  2.1270108e+00  1.7971786e+00  1.4493951e+00  1.4496814e+00
  1.1197973e+00  9.1342950e-01  2.4033146e+00  1.8129550e+00
  1.0002953e+00  8.5699838e-01  5.6510448e-01  1.3441268e+00
```

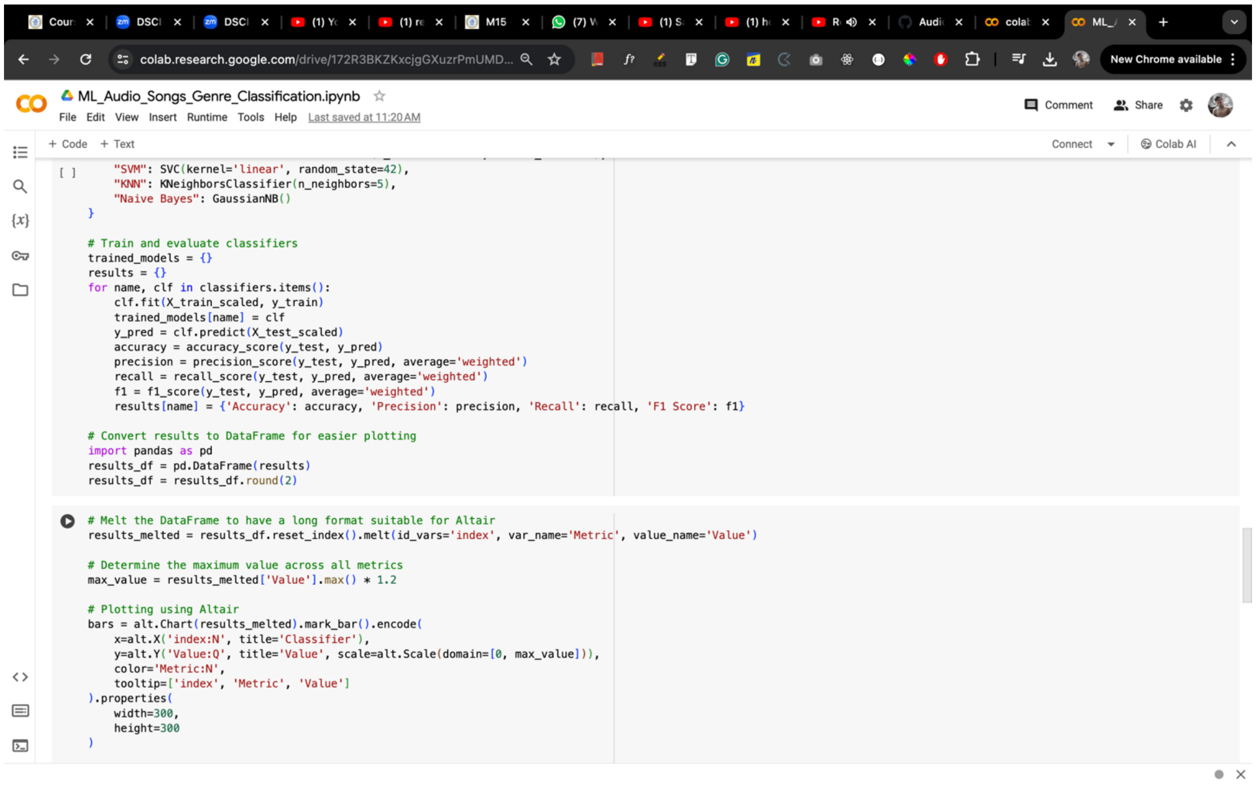

ML_Audio_Songs_Genre_Classification.ipynb

```
 1.0002953e+00  8.5699838e-01  5.6510448e-01  1.3441268e+00
 1.0660747e+00  1.3521223e+00  1.1849586e+00  6.5171081e-01
 1.0603527e+00  9.9267238e-01  1.4592320e+00  9.4239992e-01
 9.7816360e-01  5.8500147e-01  1.1793243e+00  9.1521370e-01
 6.1114055e-01  5.3006101e-01  6.8876761e-01  6.4919531e-01
 5.6341010e-01  8.4636229e-01  6.7359436e-01  5.9137040e-01
 4.7571716e-01  5.3200841e-01  4.4898567e-01  3.2385248e-01
 2.6310799e-01  1.9870275e-01  1.8977620e-01  2.0970224e-01
 2.5187704e-01  3.0444351e-01  3.0354002e-01  3.1923273e-01
 3.3935711e-01  3.2133034e-01  2.6088428e-01  2.3859484e-01
 2.4587665e-01  2.5118759e-01  2.5418401e-01  2.4778390e-01
 2.3644465e-01  2.1517996e-01  2.3747854e-01  2.0853853e-01
 2.2065431e-01  2.0636600e-01  1.8402840e-01  2.1291451e-01
 2.0265007e-01  2.2352372e-01  2.1162571e-01  1.9963160e-01
 1.8060555e-01  1.8913442e-01  1.8290932e-01  2.2938779e-01
 2.6697382e-01  2.6198882e-01  2.3188403e-01  2.3602118e-01
 2.0998406e-01  1.8640102e-01  1.7174977e-01  1.7273574e-01]
Sample y_train : 3
Sample y_train : 3
Checking dataset imbalance : [[ 0 87]
 [ 1 79]
 [ 2 78]
 [ 3 80]
 [ 4 87]
 [ 5 73]
 [ 6 75]
 [ 7 85]
 [ 8 79]
 [ 9 77]]
```

```python
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

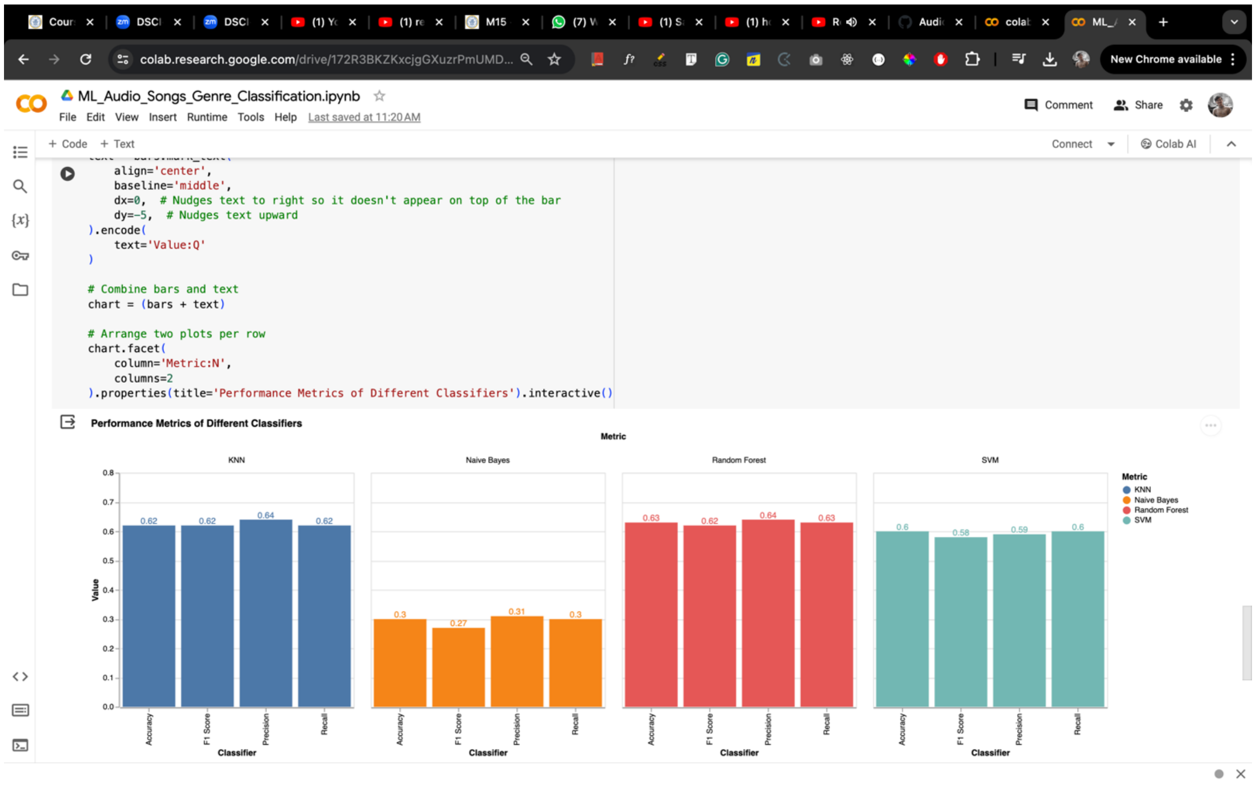
Model Development and Testing

```python
# Initialize classifiers
classifiers = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
```

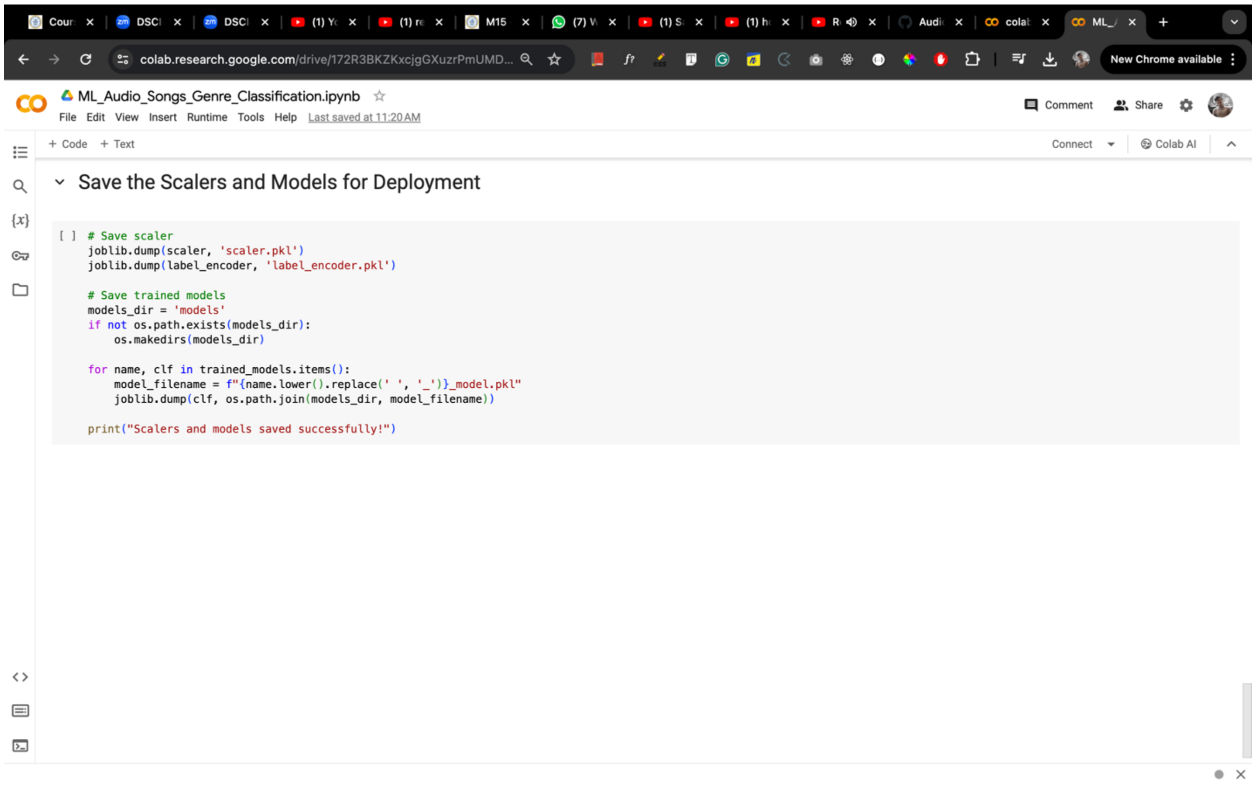

ML_Audio_Songs_Genre_Classification.ipynb

```python
    "SVM": SVC(kernel='linear', random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Naive Bayes": GaussianNB()
}

# Train and evaluate classifiers
trained_models = {}
results = {}
for name, clf in classifiers.items():
    clf.fit(X_train_scaled, y_train)
    trained_models[name] = clf
    y_pred = clf.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    results[name] = {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}

# Convert results to DataFrame for easier plotting
import pandas as pd
results_df = pd.DataFrame(results)
results_df = results_df.round(2)
```

```python
# Melt the DataFrame to have a long format suitable for Altair
results_melted = results_df.reset_index().melt(id_vars='index', var_name='Metric', value_name='Value')

# Determine the maximum value across all metrics
max_value = results_melted['Value'].max() * 1.2

# Plotting using Altair
bars = alt.Chart(results_melted).mark_bar().encode(
    x=alt.X('index:N', title='Classifier'),
    y=alt.Y('Value:Q', title='Value', scale=alt.Scale(domain=[0, max_value])),
    color='Metric:N',
    tooltip=['index', 'Metric', 'Value']
).properties(
    width=300,
    height=300
)
```


colab.research.google.com/drive/172R3BKZKxcjgGXuzrPmUMD...
ML_Audio_Songs_Genre_Classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:20 AM
Comment Share
+ Code + Text
Connect Colab AI

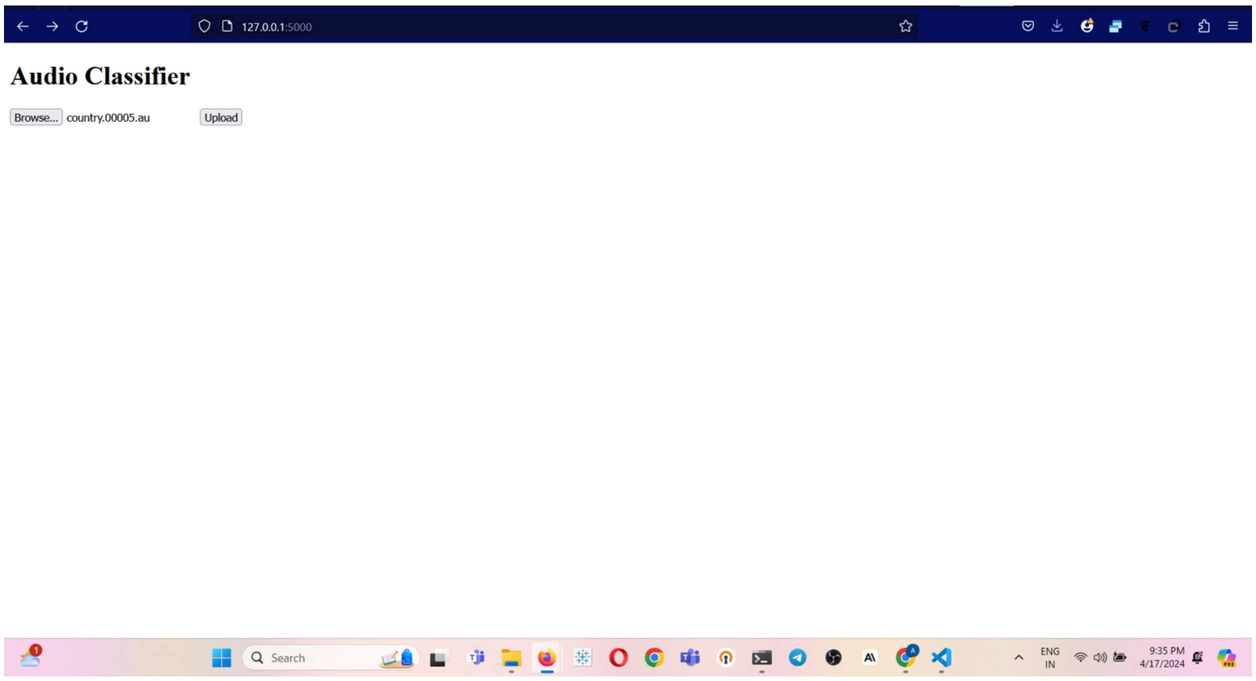
```
text = bars.mark_text(
    align='center',
    baseline='middle',
    dx=0,  # Nudges text to right so it doesn't appear on top of the bar
    dy=-5,  # Nudges text upward
).encode(
    text='Value:Q'
)

# Combine bars and text
chart = (bars + text)

# Arrange two plots per row
chart.facet(
    column='Metric:N',
    columns=2
).properties(title='Performance Metrics of Different Classifiers').interactive()
```


colab.research.google.com/drive/172R3BKZKxcjgGXuzrPmUMD...
ML_Audio_Songs_Genre_Classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:20 AM
Comment Share
+ Code + Text
Connect Colab AI

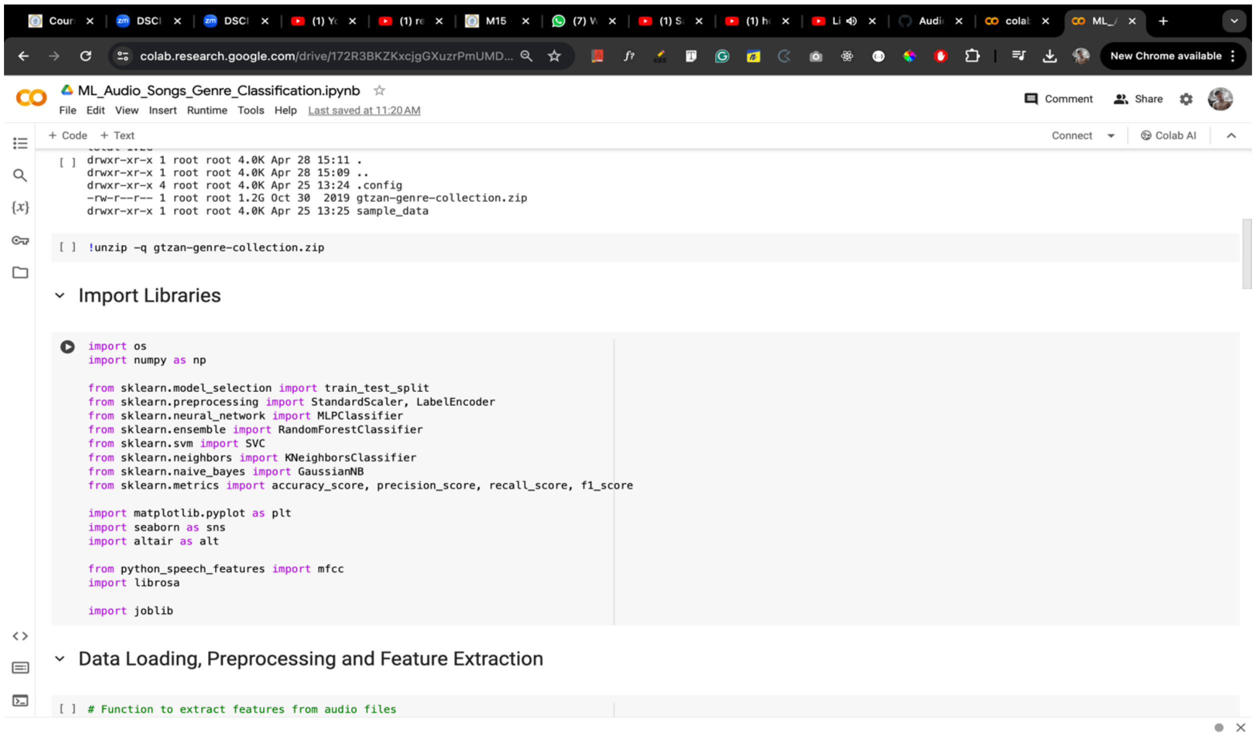
Save the Scalers and Models for Deployment

```
[ ] # Save scaler
    joblib.dump(scaler, 'scaler.pkl')
    joblib.dump(label_encoder, 'label_encoder.pkl')

    # Save trained models
    models_dir = 'models'
    if not os.path.exists(models_dir):
        os.makedirs(models_dir)

    for name, clf in trained_models.items():
        model_filename = f"{name.lower().replace(' ', '_')}_model.pkl"
        joblib.dump(clf, os.path.join(models_dir, model_filename))

    print("Scalers and models saved successfully!")
```

Product Screenshots



Audio Classifier


colab.research.google.com/drive/172R3BKZKxcjgGXuzrPmUMD...
ML_Audio_Songs_Genre_Classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:20 AM
Comment Share
+ Code + Text
Connect
Colab AI

```
drwxr-xr-x 1 root root 4.0K Apr 28 15:11 .
drwxr-xr-x 1 root root 4.0K Apr 28 15:09 ..
drwxr-xr-x 4 root root 4.0K Apr 25 13:24 .config
-rw-r--r-- 1 root root 1.2G Oct 30  2019 gtzan-genre-collection.zip
drwxr-xr-x 1 root root 4.0K Apr 25 13:25 sample_data
```

```
!unzip -q gtzan-genre-collection.zip
```

Import Libraries

```
import os
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt

from python_speech_features import mfcc
import librosa

import joblib
```

Data Loading, Preprocessing and Feature Extraction

```
# Function to extract features from audio files
```