Speech Emotion Detector

Introduction:

In the realm of speech emotion detection, advanced techniques leveraging audio feature extraction and machine learning have proven instrumental in discerning emotional states from spoken words. This project employs the powerful combination of the librosa library for audio feature extraction and the scikit-learn library for implementing a Multi-Layer Perceptron (MLP) classifier. Librosa facilitates the extraction of essential features such as Mel-frequency cepstral coefficients (MFCCs), chroma, and mel spectrograms, while scikit-learn's MLPClassifier forms the backbone of our emotion detection model. The dataset, sourced from Kaggle, encompasses a diverse array of emotional expressions in speech, allowing for a robust training and evaluation process. Leveraging Kaggle's efficiency, the dataset offers a comprehensive collection of audio samples, ensuring the model's adaptability to a wide spectrum of emotions commonly encountered in real-world scenarios. Through this amalgamation of methodologies and datasets, we aim to develop a proficient speech emotion detection system capable of accurately discerning emotional nuances in spoken language.

Methods & Libraries Used:

Methods & Libraries Used:

- **Audio Feature Extraction with Librosa:**

The project leverages the powerful Librosa library for audio feature extraction. Librosa allows for the computation of essential audio features such as Mel-frequency cepstral coefficients (MFCCs), chroma, and mel spectrograms. These features serve as crucial inputs for training the emotion detection model.

- **Machine Learning with scikit-learn:**

The scikit-learn library plays a pivotal role in implementing the Multi-Layer Perceptron (MLP) classifier. MLPClassifier, a part of scikit-learn, is utilized for training the model on the extracted audio features. This machine learning approach enables the system to learn complex patterns in the audio data, contributing to accurate emotion prediction.

- **Dataset Sourcing from Kaggle:**

The dataset chosen for this project is obtained from Kaggle. Kaggle's efficiency in providing diverse and well-curated datasets is a key factor in selecting it as the data source. The dataset encompasses a broad range of emotional expressions in speech, ensuring the model is trained on a comprehensive set of examples. This diversity enhances the model's adaptability to recognize a wide spectrum of emotions encountered in real-world scenarios.

By combining advanced audio feature extraction techniques, machine learning methodologies, and a rich dataset from Kaggle, this project aims to develop an efficient and accurate speech emotion detection system capable of discerning nuanced emotional states in spoken language.

Data Flow:

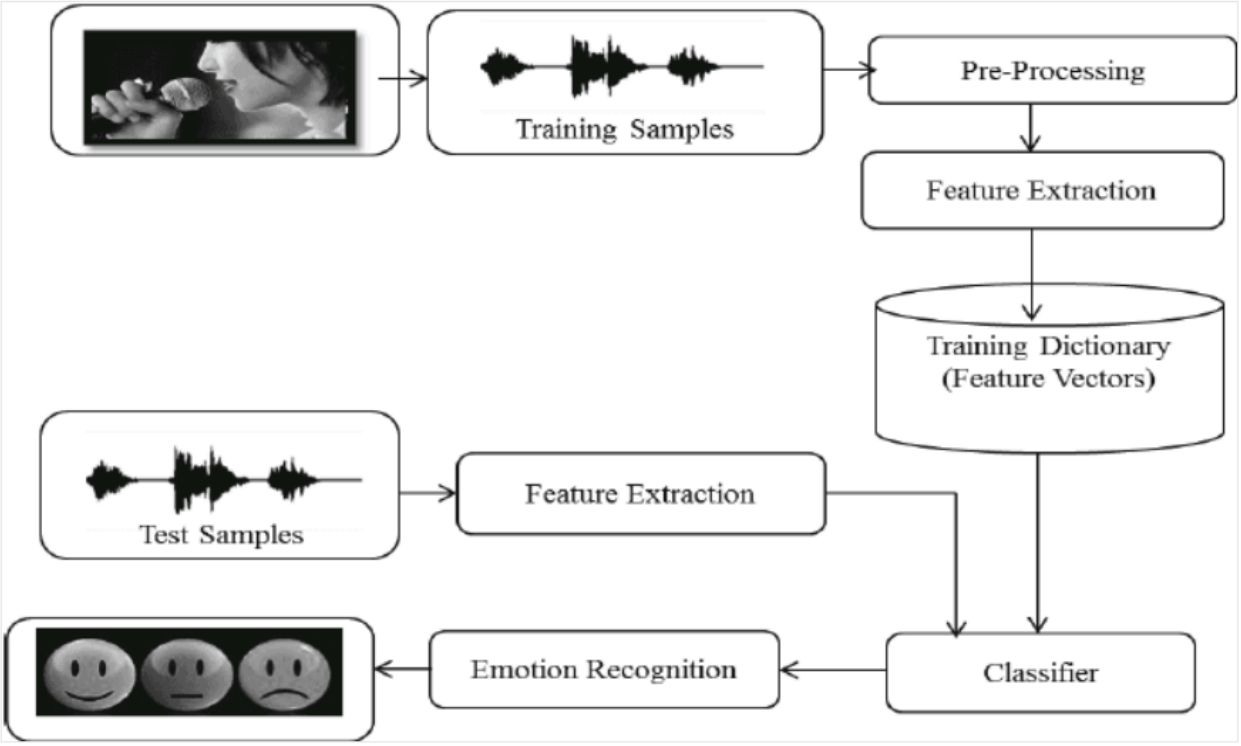


Image from researchgate.net

Code Walkthrough:

Audio Feature Extraction Function

```python
def extract_feature(file_name, mfcc, chroma, mel):
    with soundfile.SoundFile(file_name) as sound_file:
        X = sound_file.read(dtype="float32")
        sample_rate = sound_file.samplerate
        if chroma:
            stft = np.abs(librosa.stft(X))
        result = np.array([])
        if mfcc:
            mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
            result = np.hstack((result, mfccs))
        if chroma:
            chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
            result = np.hstack((result, chroma))
        if mel:
            mel = np.mean(librosa.feature.melspectrogram(y=X, sr=sample_rate).T, axis=0)
            result = np.hstack((result, mel))
    return result
```

This function takes the file name of an audio clip along with flags for enabling/disabling MFCC, Chroma, and Mel features. It extracts these features from the audio file using the librosa library.

Output:

```
(576, 192)
Features extracted: 180
Accuracy: 73.44%
```

Emotion Mapping

```python
emotions = {
    '01': 'neutral',
    '02': 'calm',
    '03': 'happy',
    '04': 'sad',
    '05': 'angry',
    '06': 'fearful',
    '07': 'disgust',
    '08': 'surprised'
}
observed_emotions = ['calm', 'happy', 'fearful', 'disgust']
```

Defines a mapping between numeric emotion codes and human-readable emotions. Only specific emotions (observed_emotions) will be considered in the analysis.

Data Loading and Splitting

```
def load_data(test_size=0.2):
    x, y = [], []
    for file in glob.glob("E:\\Speech Emotion Detector\\Dataset\\Actor_*\\*.wav"):
        file_name = os.path.basename(file)
        emotion = emotions[file_name.split("-")[2]]
        if emotion not in observed_emotions:
            continue
        feature = extract_feature(file, mfcc=True, chroma=True, mel=True)
        x.append(feature)
        y.append(emotion)
    return train_test_split( *arrays: np.array(x), y, test_size=test_size, random_state=9)

x_train, x_test, y_train, y_test = load_data(test_size=0.25)
print((x_train.shape[0], x_test.shape[0]))
print(f'Features extracted: {x_train.shape[1]}')
```

Loads audio data, extracts features using the extract_feature function, and splits it into training and testing sets.

Multi-Layer Perceptron (MLP) Model Initialization and Training

```
model = MLPClassifier(alpha=0.01, batch_size=256, epsilon=1e-08, hidden_layer_sizes=(300,), learning_rate='adaptive', max_iter=500)
# DataFlair - Train the model
model.fit(x_train, y_train)
# DataFlair - Predict for the test set
y_pred = model.predict(x_test)
# DataFlair - Calculate the accuracy of our model
accuracy = accuracy_score(y_true=y_test, y_pred=y_pred)
```

Initializes an MLP model with specified hyperparameters and trains it using the training data.

Model Evaluation and Output

```
model.fit(x_train, y_train)
# DataFlair - Predict for the test set
y_pred = model.predict(x_test)
# DataFlair - Calculate the accuracy of our model
accuracy = accuracy_score(y_true=y_test, y_pred=y_pred)

# DataFlair - Print the accuracy
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

Confusion Matrix and Classification Report

```
# DataFlair - Create confusion matrix
conf_mat = confusion_matrix(y_true=y_test, y_pred=y_pred)

# DataFlair - Create classification report
class_report = classification_report(y_true=y_test, y_pred=y_pred)

# DataFlair - Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=observed_emotions, yticklabels=observed_emotions)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# DataFlair - Display classification report
print("Classification Report:\n", class_report)
```

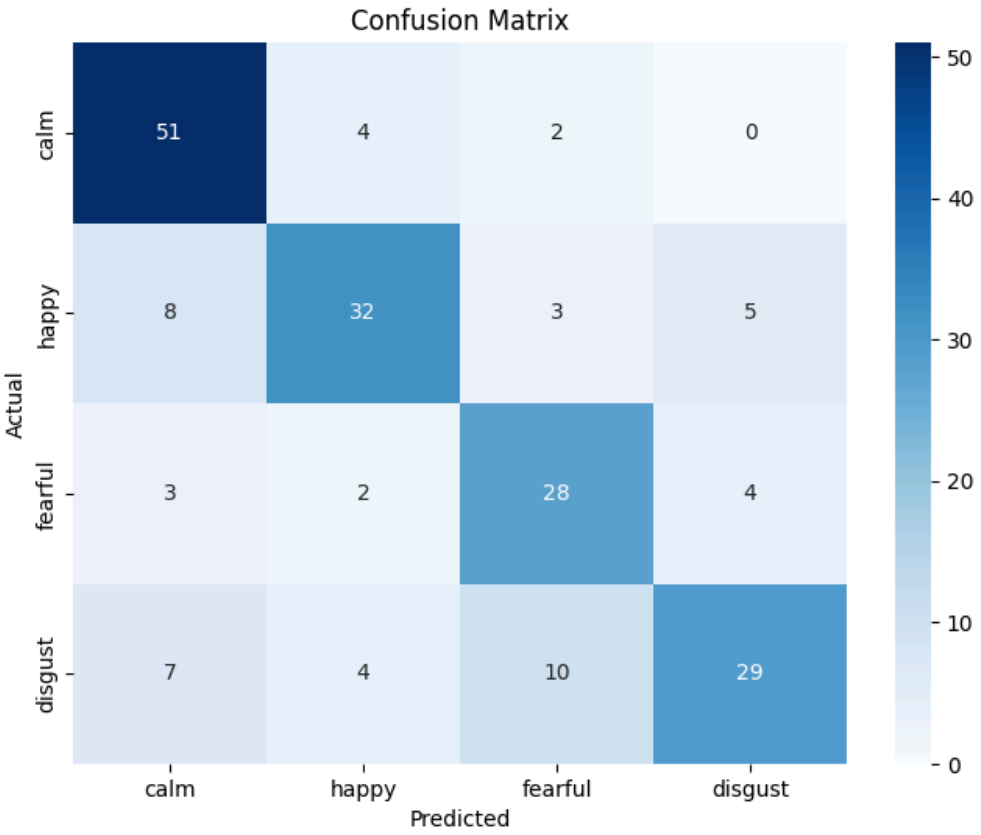
Generates a confusion matrix and classification report to evaluate the model's performance.

Distribution Bar Charts for Actual and Predicted Emotions

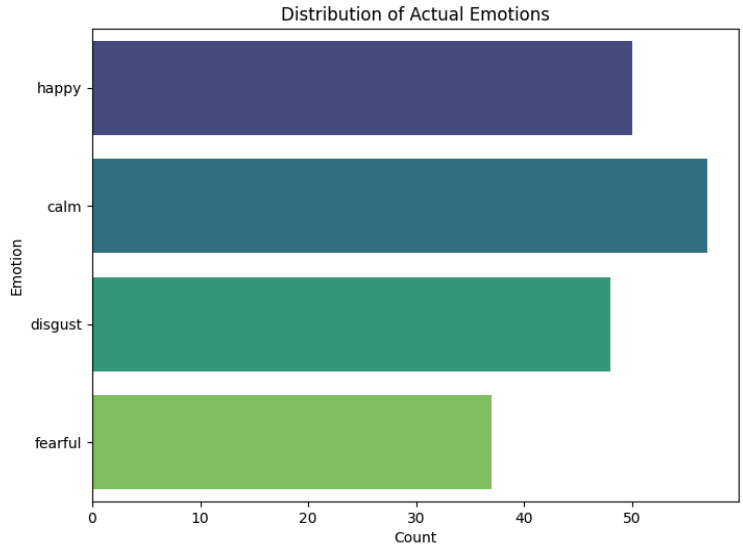
```
# DataFlair - Plot bar chart for the distribution of actual emotions
# DataFlair - Plot bar chart for the distribution of actual emotions
plt.figure(figsize=(8, 6))
sns.countplot(y_test, palette='viridis')
plt.title('Distribution of Actual Emotions')
plt.xlabel('Count')
plt.ylabel('Emotion')
plt.show()

# DataFlair - Plot bar chart for the distribution of predicted emotions
plt.figure(figsize=(8, 6))
sns.countplot(y_pred, palette='viridis')
plt.title('Distribution of Predicted Emotions')
plt.xlabel('Count')
plt.ylabel('Emotion')
plt.show()
```

Graph Analysis:

Confusion Matrix:



Distribution of Actual Emotions:



Distributions of Predictive Emotions: