Speech Data Classification Project Report

Introduction

This project aims to classify speech data using an attention-based LSTM model. The dataset consists of audio files stored in Google Drive, and the preprocessing steps include extracting MFCC features, padding sequences, and encoding labels. The model architecture is an LSTM with Bahdanau Attention, trained using categorical cross-entropy loss and evaluated using accuracy.

Dataset Pre-Processing

1. Installing Necessary Libraries

The following libraries are required for the project:

```
!pip install librosa sklearn tensorflow
```

2. Mounting Google Drive

Mount Google Drive to access the dataset:

```
from google.colab import drive
drive.mount('/content/drive')
```

3. Counting Audio Files

To count the number of audio files in each subfolder:

```
import os

def count_audio_files(folder_path):
    count_dict = {}
    for subdir, dirs, files in os.walk(folder_path):
        count = sum(1 for file in files if file.endswith('.mp3'))
        if count > 0:
            folder_name = os.path.basename(subdir)
            count_dict[folder_name] = count
    return count_dict

akan_folder_path = '/content/drive/MyDrive/akan'
```

```
audio_file_counts = count_audio_files(akan_folder_path)
print('Audio file counts per folder:', audio_file_counts)
```

4. Visualizing Audio Files

To visualize an audio file's waveform, spectrogram, MFCC, zero crossings, and spectral centroid:

```
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt

def visualize_audio(file_path):
    audio, sr = librosa.load(file_path, sr=None)
    print(f'Sample rate: {sr}, Audio duration: {len(audio)/sr} seconds')

    plt.figure(figsize=(12, 4))
    librosa.display.waveshow(audio, sr=sr)
    plt.title('Audio Waveform')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.show()

first_folder = sorted(audio_file_counts.keys())[0]
first_file = next((f for f in os.listdir(os.path.join(akan_folder_path, first_folder)) if
f.endswith('.mp3')), None)
if first_file:
    file_path = os.path.join(akan_folder_path, first_folder, first_file)
    visualize_audio(file_path)
```

5. Feature Extraction

Extracting MFCC features from the audio files:

```
def load_and_extract_features(directory, sr=22050, n_mfcc=13, n_fft=2048,
hop_length=512):
    features, labels = [], []
    for subdir, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith('.mp3'):
                file_path = os.path.join(subdir, file)
                audio, _ = librosa.load(file_path, sr=sr)
                mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc, n_fft=n_fft,
hop_length=hop_length)
                features.append(mfcc.T)
            labels.append(os.path.basename(subdir))
    return features, labels

dataset_directory = '/content/drive/MyDrive/akan'
features, labels = load_and_extract_features(dataset_directory)
```

6. Padding Features and Encoding Labels

Padding MFCC sequences to a fixed length and encoding labels:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder

def pad_features(features):
    return pad_sequences(features, padding='post', dtype='float32', maxlen=500)

def encode_labels(labels):
    encoder = LabelEncoder()
    return encoder.fit_transform(labels)

features_padded = pad_features(features)
labels_encoded = encode_labels(labels)
```

7. Splitting Data

Splitting the data into training, validation, and test sets:

```
from sklearn.model_selection import train_test_split

def split_data(features, labels, test_size=0.2):
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=test_size,
random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=test_size,
random_state=42)
    return X_train, X_val, X_test, y_train, y_val, y_test

X_train, X_val, X_test, y_train, y_val, y_test = split_data(features_padded, labels_encoded)
```

Model Development

1. Bahdanau Attention Layer

Implementing the Bahdanau Attention mechanism:

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Input, LSTM
from tensorflow.keras.models import Model

class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = Dense(units)
        self.W2 = Dense(units)
        self.V = Dense(1)

    def call(self, query, values):
        query_with_time_axis = tf.expand_dims(query, 1)
        score = self.V(tf.nn.tanh(self.W1(query_with_time_axis) + self.W2(values)))
        attention_weights = tf.nn.softmax(score, axis=1)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attention_weights
```

2. Model Architecture

Building the attention-based LSTM model:

```
def build_attention_model(input_shape, units, num_classes):
    inputs = Input(shape=input_shape)
    lstm_out, _, _ = LSTM(units, return_sequences=True, return_state=True)(inputs)
    attention_layer = BahdanauAttention(units)
    context_vector, attention_weights = attention_layer(lstm_out[:, -1, :], lstm_out)
    output = Dense(units, activation='relu')(context_vector)
    output = Dense(num_classes, activation='softmax')(output)
    model = Model(inputs=inputs, outputs=output)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

model = build_attention_model((None, 13), 128, num_classes=5)
model.summary()
```

Model Training