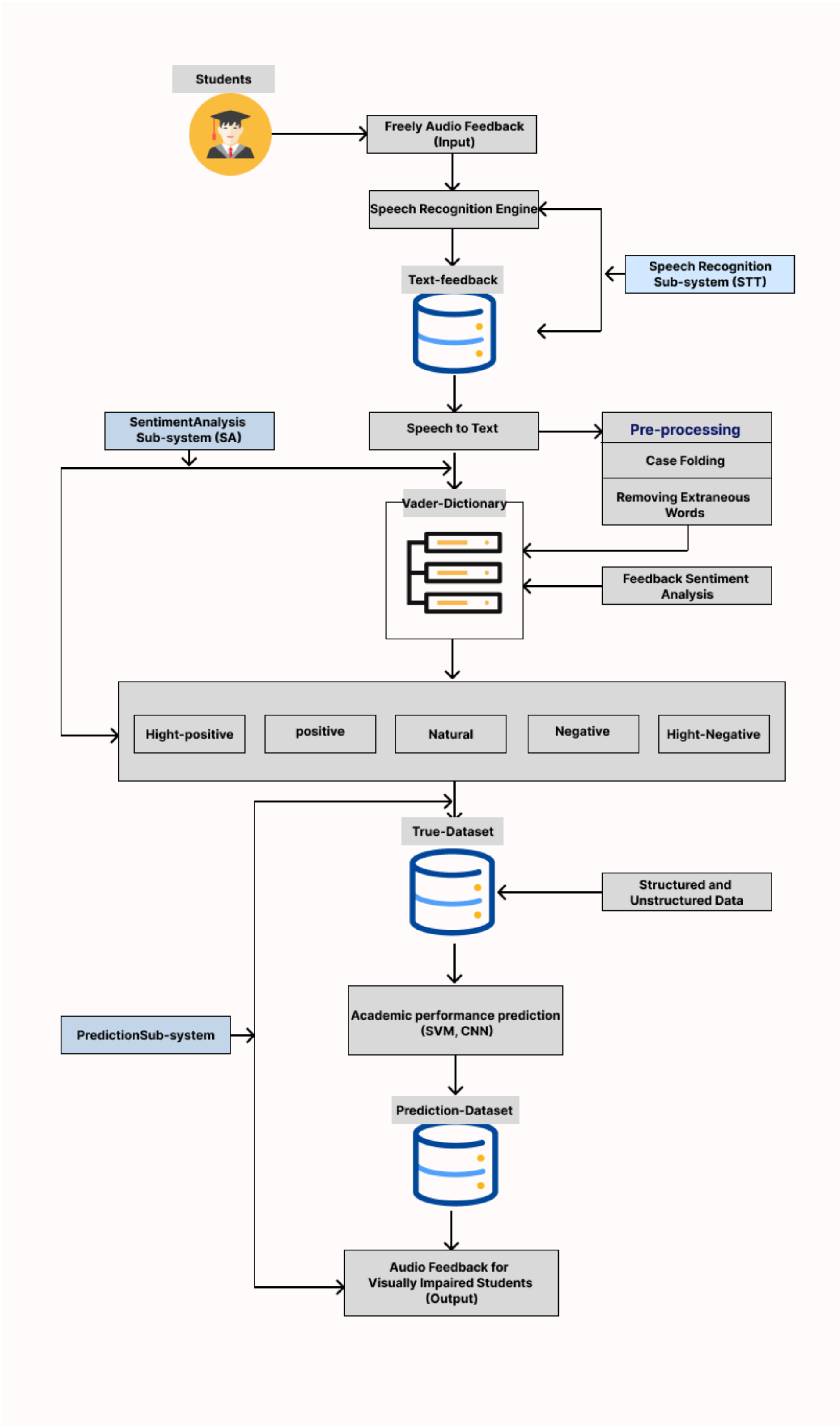
Automated Sentiment Analysis of Visually Impaired Students' Audio Feedback in Virtual Learning Environments



Proposal system diagram

Automated Sentiment Analysis of Visually Impaired Students' Audio Feedback in Virtual Learning Environments

Abstract

This document outlines the implementation and analysis of an intelligent system designed to evaluate the sentimental responses of visually impaired students. The system integrates speech recognition, sentiment analysis, and predictive modeling to assess academic performance.

Introduction

The growing need to adapt educational tools for visually impaired students has led to the development of innovative solutions that can process and analyze emotional feedback through audio inputs. This project aims to leverage advanced machine learning algorithms to predict student success and provide actionable insights into educational strategies.

System Overview

The system is composed of three main sub-systems: Speech Recognition, Sentiment Analysis, and Prediction. Each sub-system plays a pivotal role in the processing and analysis pipeline, from audio data input to predictive outputs on student performance.

Methodology

Detailed descriptions of the functionality and interplay between these sub-systems are provided, focusing on the technical implementations and the underlying algorithms employed.

Implementation Details

- Data source

Our study involved 100 impaired vision students from various universities around Egypt and was carried out for nine weeks. The course "Computer Skills Course in the English Language" was started through Microsoft Teams. All participants were communicated to, and forms of consent were signed before the course, whereby a choice of withdrawal/non-participation without penalties was given. The process of data collection was extensive and carefully integrated into the Microsoft Teams platform to ensure comprehensiveness and no loss of accuracy.

Throughout the course, multidimensional data were collected, including structured academic performance indicators and unstructured sentimental feedback. Both these kinds of data, when integrated, provided the opportunity to have an overall view of the progress and sentimental responses of the student throughout the course. Our data collection was multi-faceted and integrated into the Microsoft Teams platform. and we was gathering data as following:

• *Digital Tracking on Microsoft Teams (Structured Data):* We fully used the inbuilt functionalities of Microsoft Teams in the process of participating in tracking student participation. The same included checking attendance in each session, checking the submission rate of homework, and analyzing involvement in classroom discussions. In this regard, Microsoft Teams provided us with electronic logs and participation reports that were exported for analysis to quantify student engagement. The first evaluation was done by the end of the fourth week, and the second one by the end of the ninth week.

• *Collection of Audio Feedback (Unstructured Data):* This work illustrates the study's responsiveness, which is useful to help provide a useful means for identifying the needs of visual impairment in the section of the program where major assessments are needed. The first evaluation was done by the end of the fourth week, and the second one by the end of the ninth week, after audio feedback had been collected. The students were to audio-record themselves for every lesson while speaking, self-assessing, and reflecting on the learning content, then submit their recordings on the Microsoft Teams platform. The approach did not help in collecting valuable qualitative data but also provided in-depth insights into the experiences and understanding of the students.

My system content of three sub-systems is as follows:

The first sub-system: speech recognition (STT)

Libraries Used:

- speech_recognition: for converting speech into text.

- pydub: for manipulating audio files, especially useful for splitting the audio into chunks and handling different audio formats.

- librosa: for audio normalization and processing.

Python Code for Speech Recognition Component

```
import speech_recognition as sr
from pydub import AudioSegment
from pydub.silence import split_on_silence
import librosa
def normalize_audio(audio_path):
    # Load audio file with librosa
    y, sr = librosa.load(audio_path, sr=None)
    # Normalize the audio to -20 dBFS
    y_norm = librosa.util.normalize(y, norm=float(librosa.db_to_amplitude(-20)))
    # Save the normalized audio back to a file
    librosa.output.write_wav(audio_path, y_norm, sr)
def transcribe_audio(audio_path):
    # Initialize the recognizer and load the audio file
    recognizer = sr.Recognizer()
    audio = AudioSegment.from_file(audio_path)
    # Split the audio file where silence is 0.5 seconds or more and dBFS difference is -30
    chunks = split_on_silence(audio, min_silence_len=500, silence_thresh=-30)
    # Process each chunk of audio
    complete_text = ''
    for chunk in chunks:
        with sr.AudioFile(chunk.export(format="wav")) as source:
            audio_data = recognizer.record(source)
            # Try recognizing the speech in the chunk
            try:
                text = recognizer.recognize_google(audio_data)
                complete_text += text + " "
            except sr.UnknownValueError:
                print("Could not understand audio")
            except sr.RequestError as e:
                print(f"Request Error from Google Speech Recognition service; {e}")
```

```
    return complete_text
# Normalize the audio file before processing

normalize_audio("path_to_audio_file.wav")

# Transcribe the normalized audio

transcript = transcribe_audio("path_to_audio_file.wav")

print(transcript)
```

Pseudocode for Speech Recognition System

```
Define normalize_audio(audio_path):
    Load audio file with librosa
    Normalize the audio to uniform loudness
    Save the normalized audio file
Define transcribe_audio(audio_path):
    Initialize speech recognizer
    Load and split the audio file into chunks based on silence
    For each chunk:
        Convert chunk to audio format readable by speech recognizer
        Adjust recognizer settings to handle ambient noise
        Try to recognize speech in the chunk
        If speech recognized:
            Append recognized text to complete text
        Else:
            Handle errors and continue
    Return the complete transcript
Procedure to process audio feedback:
    Normalize the audio to ensure uniform audio levels
    Transcribe the normalized audio to text
    Output the transcribed text
Main:
    Call normalize_audio with path to the audio file
    Call transcribe_audio with path to the normalized audio file
    Print the transcription results
# This pseudocode sets up the complete flow from audio normalization to speech-to-text transcription.
```

Parameters and Configuration

- Audio Source Settings: WAV format, with a typical sampling rate of 16000 Hz for speech recognition.

- Ambient Noise Adjustment: First few seconds of the audio are used to calibrate the recognizer's sensitivity to background noise.

- Timeout and Confidence Thresholds: A timeout might be set to avoid hanging during processing, and a confidence threshold could be set to ignore uncertain recognitions, though these aren't explicitly shown in the Python script above.

Notes:

- Ensure that the audio files are accessible at the specified path in your Python script.

- Adjust the silence threshold and minimum silence length in `split_on_silence` according to the actual ambient noise levels in your audio files for optimal results.

Second sub-system: sentiment analysis (SA):

Libraries Used:

- nltk: Natural Language Toolkit, used here for text preprocessing like stopword removal and tokenization.

- vaderSentiment: For sentiment analysis, particularly suited for social media texts and short phrases.

Python Code for Sentiment Analysis Component

```
import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import WordNetLemmatizer

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

import string

# Ensure necessary NLTK resources are downloaded

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

def preprocess_feedback(feedback):

    # Convert to lowercase

    feedback = feedback.lower()

    # Remove punctuation
```

```
    feedback = feedback.translate(str.maketrans('', '', string.punctuation))

    # Tokenize feedback

    words = word_tokenize(feedback)

    # Remove stopwords

    words = [word for word in words if word not in stopwords.words('english')]

    # Lemmatize the words

    lemmatizer = WordNetLemmatizer()

    words = [lemmatizer.lemmatize(word) for word in words]

    return ' '.join(words)

def analyze_sentiment(feedback):

    # Preprocess the feedback

    clean_feedback = preprocess_feedback(feedback)

    # Initialize VADER sentiment analyzer

    analyzer = SentimentIntensityAnalyzer()

    # Get sentiment scores

    sentiment_scores = analyzer.polarity_scores(clean_feedback)

    return sentiment_scores

# Example usage

feedback_text = "The course was extremely helpful and the instructor was very encouraging!"

processed_feedback = preprocess_feedback(feedback_text)

sentiment_results = analyze_sentiment(feedback_text)

print("Processed Feedback:", processed_feedback)

print("Sentiment Scores:", sentiment_results)
```

Pseudocode for Sentiment Analysis System

Automated Sentiment Analysis of Visually Impaired Students' Audio Feedback in Virtual Learning Environments

Function preprocess_feedback(feedback):

Convert feedback to lowercase

Remove all punctuation from feedback

Tokenize feedback into words

Remove stopwords from tokenized words

Lemmatize each word to its base form

Combine lemmatized words back into a single string

Return the processed feedback

Function analyze_sentiment(feedback):

Preprocess the feedback

Initialize the sentiment analyzer (VADER)

Analyze the preprocessed feedback for sentiment scores

Return sentiment scores

Main Procedure:

Input: Raw feedback text

Call preprocess_feedback with raw feedback

Call analyze_sentiment with preprocessed feedback

Print processed feedback and sentiment scores

Parameters and Configuration

- Text Normalization: Lowercase conversion and punctuation removal to standardize the text.

- Tokenization: Splitting text into individual words to process each word for stop words and lemmatization.

- Stopword Removal: Filtering out common words that don't contribute to sentiment analysis.

- Lemmatization: Converting words to their base form to reduce the complexity of the analysis.

- VADER Configuration: Using default settings which are generally well-tuned for social media texts.

Third Sud-system: prediction:

We'll integrate sentiment analysis data with structured academic performance data from Microsoft Teams to train two machine learning models: a Support Vector Machine (SVM) and a Convolutional Neural Network (CNN). These models will predict whether students pass or fail based on the integrated dataset.

The first: SVM prediction:

We integrate sentiment analysis data with structured academic performance data from various educational tools to train a Support Vector Machine (SVM). This model predicts whether students pass or fail based on an integrated dataset that includes both structured academic metrics and sentiment analysis results.

Step 1: Load and Process Structured Data

This step involves loading structured data, which might come from educational platforms like Microsoft Teams or other Learning Management Systems (LMS). The structured data includes metrics like:

- Homework Grade: Numerical scores representing student performance on homework.

- Homework Clicks: Count of interactions or clicks recorded in the homework module.

- Attendance: Records showing whether students attended sessions.

- Discussion Participation: Measures student engagement in course forums or discussion boards.

Step 2: Process Sentiment Data:

If available, this step involves processing sentiment analysis results which provide a sentiment score for each student based on their textual feedback. This data helps in understanding the emotional and psychological state of the students, which can be a significant factor in their overall performance.

Step 3: Combine Datasets

Here, both the structured academic metrics and the sentiment scores are merged to form a comprehensive dataset. This integration allows the model to utilize both numerical performance metrics and unstructured sentiment data.

Step 4: Data Preprocessing

Before training the SVM model, the combined dataset needs to be preprocessed:

- Categorical Data Encoding: Features like 'Sentiment Type' and 'Attendance' are converted from categorical to numerical formats using label encoding to make them suitable for the model.

- Feature Aggregation: For each student (identified by 'Student ID'), aggregate multiple records by calculating the mean of numerical features and the mode of categorical features.

Step 5: Model Training

The preprocessed data is then used to train an SVM model. This involves:

- Feature Selection: Selecting relevant features for the model, including academic metrics and, if available, sentiment scores.

- Model Training: Training the SVM model on the dataset to predict the 'Pass/Fail' outcome for each student.

Libraries and Tools

- Scikit-learn: Used for building the SVM model, preprocessing data, and for splitting the dataset into training and testing sets.

- Pandas: Utilized for data manipulation and merging datasets.

- NumPy: Employed for numerical operations, especially for handling arrays and matrices during data preprocessing.

- Joblib/Pickle: For saving the model state and ensuring that the predictions can be reproduced later.

Python Code

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.svm import SVC

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report

import joblib

# Load your datasets

data = pd.read_csv('your_dataset.csv')  # Replace with your actual filename

# If there is textual feedback, process it (assuming sentiment analysis has been done)

if 'Feedback Text' in data.columns:

    data['Feedback Text'] = data['Feedback Text'].astype(str)
```

```
# Aggregate data by 'Student ID'
def get_most_frequent(series):
    return series.value_counts().idxmax()
# Numerical features: Calculate the mean
numerical_agg = data.groupby('Student ID')[['Homework Grade', 'Homework Click', 'Discussion']].mean()
# Categorical features: Calculate the most frequent value (mode)
categorical_agg = data.groupby('Student ID')[['Sentiment Type', 'Attendance']].agg(get_most_frequent)
# Combine aggregated data
aggregated_data = pd.merge(numerical_agg, categorical_agg, on='Student ID')
# Preprocess the Data
label_encoder = LabelEncoder()
for column in ['Sentiment Type', 'Attendance']:
    aggregated_data[column] = label_encoder.fit_transform(aggregated_data[column])
# Prepare Features and Labels
X = aggregated_data[['Homework Grade', 'Homework Click', 'Discussion', 'Sentiment Type', 'Attendance']].values
y = np.array([0 if int(student_id.split('_')[-1]) % 2 == 0 else 1 for student_id in aggregated_data.index])
# Train the Model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = SVC(kernel='linear', C=1)
model.fit(X_train, y_train)
# Save the model
joblib.dump(model, 'svm_model.pkl')
# Predict and Evaluate
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

Automated Sentiment Analysis of Visually Impaired Students' Audio Feedback in Virtual Learning Environments

Pseudocode

```
Procedure LoadAndProcessData:
    Load the dataset from a CSV file
    If 'Feedback Text' exists in data columns:
        Convert 'Feedback Text' to string for each record
    Aggregate numerical data by 'Student ID' using mean
    Aggregate categorical data by 'Student ID' using mode
    Merge all aggregated data into one DataFrame

Procedure PreprocessData:
    Initialize a LabelEncoder
    For each categorical column in the dataset:
        Encode the column using LabelEncoder
    Prepare the feature matrix X and label vector y
    Return X and y

Procedure TrainSVM:
    Split the dataset into training and testing parts
    Initialize and train an SVM model with a linear kernel
    Save the trained model using joblib
    Return the trained model

Procedure EvaluateModel:
    Predict labels for the test set using the trained SVM model
    Print the classification report for the predictions

Main:
```

Call LoadAndProcessData to load and preprocess the data

X, y <- Call PreprocessData to encode data and prepare features and labels

model <- Call TrainSVM to train the SVM model using X and y

Call EvaluateModel to test and evaluate the trained model

Code Explanation

1. Data Loading and Preprocessing: The script loads the dataset and checks for textual feedback, processing it as needed. It aggregates the numerical and categorical data by student, using the mean for numerical features and mode for categorical features.

2. Data Encoding: Categorical features are encoded to numerical values using `LabelEncoder`, making them suitable for model training.

3. Model Training and Evaluation: The script trains an SVM model using the preprocessed data, then evaluates its performance using a classification report, which provides metrics like precision, recall, and F1-score.

4. Model Saving: The trained model is saved using `joblib` for future use, ensuring that the training process does not need to be repeated.

The second: CNN prediction:

We integrate sentiment analysis data with structured academic performance data from various educational platforms to train a Convolutional Neural Network (CNN). This model predicts whether students pass or fail based on an integrated dataset that includes both structured academic metrics and unstructured sentiment analysis results.

Step 1: Load and Process Structured Data

This step involves loading structured data from educational platforms like Microsoft Teams. The structured data typically includes metrics such as:

- Homework Grades: Numerical scores representing student performance on homework assignments.

- Homework Clicks: Number of interactions or clicks in the homework module, indicating engagement.

- Attendance: Records indicating whether students attended sessions, marked typically as present or absent.

- Discussion Participation: Measures of how actively students participate in course forums or discussion boards.

Step 2: Process Sentiment Data

This step involves processing sentiment analysis results, which provide a sentiment score for each student based on their textual feedback. This helps gauge the emotional and psychological state of the students, which can significantly impact their academic performance.

Step 3: Combine Datasets

Here, both the structured academic metrics and the sentiment scores (if available) are merged to create a comprehensive dataset. This combination allows the model to leverage both numerical performance metrics and unstructured sentiment data.

Step 4: Data Preprocessing

Before training the CNN model, the combined dataset undergoes several preprocessing steps:

- Categorical Data Encoding: Features like 'Sentiment Type' and 'Attendance' are converted from categorical to numerical formats using label encoding. This makes them suitable for model input.

- Feature Aggregation: For each student (identified by 'Student ID'), aggregate multiple records by calculating the mean of numerical features and the mode of categorical features.

- Text Data Processing: If textual feedback is available, it's transformed into numerical data using TF-IDF (Term Frequency-Inverse Document Frequency), which helps in understanding the importance of words in the text.

Step 5: Model Training

The preprocessed data is used to train a CNN model. The steps include:

- Feature Selection: Selecting relevant features for the model, which include academic metrics and, if available, sentiment scores.

- Model Training: Training the CNN model on the dataset to predict the 'Pass/Fail' outcome for each student based on their aggregated and encoded features.

Libraries and Tools

- TensorFlow/Keras: For building and training the CNN model.

- Scikit-learn: Used for SVM model building, data preprocessing, and for splitting the dataset into training and testing subsets.

- Pandas: Utilized for data manipulation and merging datasets.

- NumPy: Employed for numerical operations, especially for handling arrays and matrices during data preprocessing.

- TfidfVectorizer: For converting text data into numerical vectors based on TF-IDF statistics.

Automated Sentiment Analysis of Visually Impaired Students' Audio Feedback in Virtual Learning Environments

Python Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, TfidfVectorizer
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, GlobalMaxPooling1D, Flatten
from tensorflow.keras.utils import to_categorical
from google.colab import files

# Step 1: Upload and Load the dataset
uploaded = files.upload()
filename = list(uploaded.keys())[0]
data = pd.read_csv(filename)

# Display the structure of the dataset
print(data.head())
print("\nUnique values in 'Attendance' column:", data['Attendance'].unique())

# Step 2: Define the function to get the most frequent value
def get_most_frequent(series):
    return series.value_counts().idxmax()

# Step 3: Aggregate data by 'Student ID'
numerical_agg = data.groupby('Student ID')[['Homework Grade', 'Homework Click', 'Discussion']].mean()
```

```
categorical_agg = data.groupby('Student ID')[['Sentiment Type', 'Attendance']].agg(get_most_frequent)


# Combine the aggregated data
if 'Feedback Text' in data.columns:
    text_agg = data.groupby('Student ID')['Feedback Text'].apply(' '.join).reset_index()
    aggregated_data = pd.merge(pd.merge(numerical_agg, categorical_agg, on='Student ID'), text_agg, on='Student ID')
else:
    aggregated_data = pd.merge(numerical_agg, categorical_agg, on='Student ID')


# Step 4: Preprocess the Data
label_encoder = LabelEncoder()
for column in ['Sentiment Type', 'Attendance']:
    aggregated_data[column] = label_encoder.fit_transform(aggregated_data[column])


if 'Feedback Text' in aggregated_data.columns:
    tfidf = TfidfVectorizer(max_features=500)
    text_features = tfidf.fit_transform(aggregated_data['Feedback Text']).toarray()
    X_full = np.hstack([aggregated_data.drop(['Feedback Text'], axis=1).values, text_features])
else:
    X_full = aggregated_data.values


X_full = np.expand_dims(X_full, axis=2)
y = to_categorical([0 if int(sid.split('_')[-1]) % 2 == 0 else 1 for sid in aggregated_data.index])


# Step 5: Train and Evaluate CNN Model
X_train, X_test, y_train, y_test = train_test_split(X_full, y, test_size=0.2, random_state=42)
```

```
model = Sequential([
    Conv1D(64, 3, activation='relu', input_shape=(X_train.shape[1], 1)),
    GlobalMaxPooling1D(),
    Dense(32, activation='relu'),
    Dense(2, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=10)

# Predict and evaluate
y_pred = np.argmax(model.predict(X_test), axis=1)
y_true = np.argmax(y_test, axis=1)
print("CNN Accuracy:", accuracy_score(y_true, y_pred))

# Step 6: Predict for the entire dataset
final_predictions = np.argmax(model.predict(X_full), axis=1)
aggregated_data['Pass/Fail'] = ['Pass' if pred == 0 else 'Fail' for pred in final_predictions]

# Merge and Output the final dataset
final_data = pd.merge(data, aggregated_data[['Pass/Fail']], left_on='Student ID', right_index=True, how='left')
output_filename = 'CNN_Aggregated_Results_with_Pass_Fail.csv'
final_data.to_csv(output_filename)

# Download the result file
files.download(output_filename)
```

pseudocode

Automated Sentiment Analysis of Visually Impaired Students' Audio Feedback in Virtual Learning Environments

```
Procedure Process_and_Predict_with_CNN
  Begin
    // Import the necessary libraries
    Import pandas as pd
    Import numpy as np
    Import LabelEncoder, StandardScaler from sklearn.preprocessing
    Import train_test_split from sklearn.model_selection
    Import accuracy_score from sklearn.metrics
    Import TfidfVectorizer from sklearn.feature_extraction.text
    Import Sequential from tensorflow.keras.models
    Import Dense, Conv1D, GlobalMaxPooling1D, Flatten from tensorflow.keras.layers
    Import to_categorical from tensorflow.keras.utils
    Import files from google.colab

    // Step 1: Upload and Load the dataset
    Display "Please upload your dataset"
    uploaded <- files.upload()
    filename <- Get the first key from the uploaded dictionary
    data <- Load CSV file into DataFrame from filename

    // Display the structure of the dataset
    Print the first few rows of the data
    Print "Unique values in 'Attendance' column:", unique values in data['Attendance']

    // Step 2: Define the function to get the most frequent value
    Define Function get_most_frequent(series)
      Begin
        counts <- Get value counts of the series
```

If counts is not empty Then

Return the most frequent element in counts

Else

Return NaN

End

// Step 3: Aggregate data by 'Student ID'

numerical_agg <- Group data by 'Student ID' and calculate mean of ['Homework Grade', 'Homework Click', 'Discussion']

categorical_agg <- Group data by 'Student ID' and apply get_most_frequent on ['Sentiment Type', 'Attendance']

If 'Feedback Text' exists in data columns Then

text_agg <- Group data by 'Student ID' and concatenate 'Feedback Text' into a single string

aggregated_data <- Merge numerical_agg, categorical_agg, and text_agg on 'Student ID'

Else

aggregated_data <- Merge numerical_agg and categorical_agg on 'Student ID'

// Step 4: Preprocess the Data

Initialize label_encoder as LabelEncoder

For each column in ['Sentiment Type', 'Attendance'] Do

Encode aggregated_data[column] using label_encoder

If 'Feedback Text' exists in aggregated_data columns Then

Initialize tfidf as TfidfVectorizer with max_features set to 500

text_features <- Transform aggregated_data['Feedback Text'] into numerical data using tfidf

X_full <- Combine aggregated_data (excluding 'Feedback Text') values and text_features horizontally

Else

X_full <- Convert aggregated_data values to a numpy array

X_full <- Expand dimensions of X_full by adding a new last axis

y <- Convert array of [0 if student_id is even else 1 for each student_id in aggregated_data] to categorical data

// Step 5: Train and Evaluate CNN Model

Split X_full and y into X_train, X_test, y_train, y_test with test size 0.2 and random state 42

// Define and train CNN for Structured Data Only

model <- Create a new Sequential model

Add Conv1D layer with 64 filters, kernel size 3, activation 'relu', and input shape (X_train.shape[1], 1) to model

Add GlobalMaxPooling1D layer to model

Add Dense layer with 32 units and activation 'relu' to model

Add Dense layer with 2 units and activation 'softmax' to model

Compile model with optimizer 'adam', loss 'categorical_crossentropy', and metrics ['accuracy']

Train model on X_train and y_train with epochs 10 and batch size 10

// Predict and evaluate

y_pred <- Get argmax of model.predict(X_test) along axis 1

y_true <- Get argmax of y_test along axis 1

Print "CNN Accuracy:", accuracy_score(y_true, y_pred)

// Step 6: Predict for the entire dataset

final_predictions <- Get argmax of model.predict(X_full) along axis 1

aggregated_data['Pass/Fail'] <- ['Pass' if pred is 0 else 'Fail' for each pred in final_predictions]

```
// Merge Predictions with the Original Data
final_data <- Merge data with aggregated_data['Pass/Fail'] on 'Student ID'
output_filename <- 'CNN_Aggregated_Results_with_Pass_Fail.csv'
Save final_data to CSV file named output_filename


// Download the result file
Attempt
    Download file named output_filename
Catch any errors and display "Error downloading the file"


End
```

Pseudocode Explanation

1. Library Imports: The necessary Python libraries are imported, including data manipulation, machine learning preprocessing, model training, and utilities for working with files in Google Colab.

2. Data Upload and Loading:

   - The user is prompted to upload a dataset.

   - The uploaded dataset is read into a DataFrame `data`.

   - The structure of `data` is displayed to understand its columns and initial rows.

   - The unique values in the 'Attendance' column are printed to understand its data distribution.

3. Function Definition - `get_most_frequent`:

   - A helper function `get_most_frequent` is defined to determine the most frequent (mode) value in a series, which is particularly useful for categorical data.

4. Data Aggregation:

   - Numerical features like 'Homework Grade', 'Homework Click', and 'Discussion' are aggregated by the mean for each 'Student ID'.

   - Categorical features like 'Sentiment Type' and 'Attendance' are aggregated by the mode for each 'Student ID'.

   - If 'Feedback Text' is available, it is concatenated into a single string for each 'Student ID'.

   - The aggregated data is stored in `aggregated_data`.

5. Data Preprocessing:

   - Categorical variables are encoded using `LabelEncoder`.

   - If 'Feedback Text' is part of the data, it is transformed into numerical features using `TfidfVectorizer`.

   - The structured and unstructured data are combined into `X_full`.

   - The dimensions of `X_full` are expanded to fit the input requirements of `Conv1D`.

   - Labels `y` are prepared based on the parity of the numeric part of 'Student ID' and converted to a categorical format.

6. Model Training and Evaluation:

   - The dataset is split into training and testing subsets.

   - A CNN model is defined with layers including `Conv1D`, `GlobalMaxPooling1D`, and `Dense`.

   - The model is compiled and trained.

   - Predictions are made on the test set, and accuracy is printed.

7. Predict and Merge Results:

   - The CNN model is used to predict outcomes for the entire dataset.

   - Predictions are converted to 'Pass' or 'Fail' and added to `aggregated_data`.

   - These predictions are merged back with the original `data` to provide a comprehensive view.

8. Output and Save:

   - The final dataset with predictions is saved to a CSV file.

   - An attempt is made to download this file, with any errors in downloading reported to the user.