Speech Emotion Classification with K-Nearest Neighbors, Support Vector Machines, and Convolutional Neural Networks

---

Team JARL
Jasmine Lee, Arielle Sanford, Robert Heeter, Lindsey Russ

ELEC 378: Machine Learning: Concepts & Techniques
Rice University

Submitted 2 May 2023

Contents

Data Exploration

The data consists of a set of 1,125 training audio files and 315 testing audio files for the Kaggle competition, each about 3-4 seconds long and stored with the *.wav* format. The audio samples are of an American male and female voice saying either the phrase “dogs are sitting by the door” or “kids are talking by the door” in a similar rhythm. The training data is split into 8 different labeled emotions classes corresponding to the relative tone of the speaker in each clip: *angry*, *calm*, *disgust*, *fearful*, *happy*, *neutral*, *sad*, and *surprised*. The audio clips are all sampled at 22050 Hz.

We experimented with many different preprocessing algorithms (as described more below) and classification models for this problem, including logistic regression, *k*-nearest neighbors (KNN), support vector machines (SVM), and convolutional neural networks (CNN), the latter three of which we explore more in this report given that logistic regression did not appear to yield good results for us during initial testing.

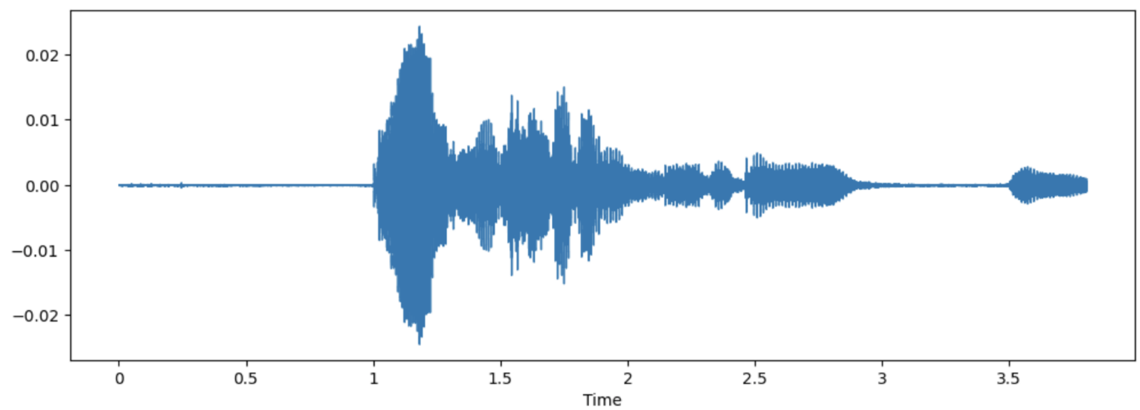
Feature Extraction

The primary feature we decided to extract (or calculate) from the audio files was the mel frequency cepstral coefficients, or MFCCs, as cepstral features are commonly used in industry applications for speech analysis and have been well-researched. The mel-frequency cepstrum is a representation of the power spectrum of a sound, which is directly related to the biology and dynamics of the vocal tract structure during speech.

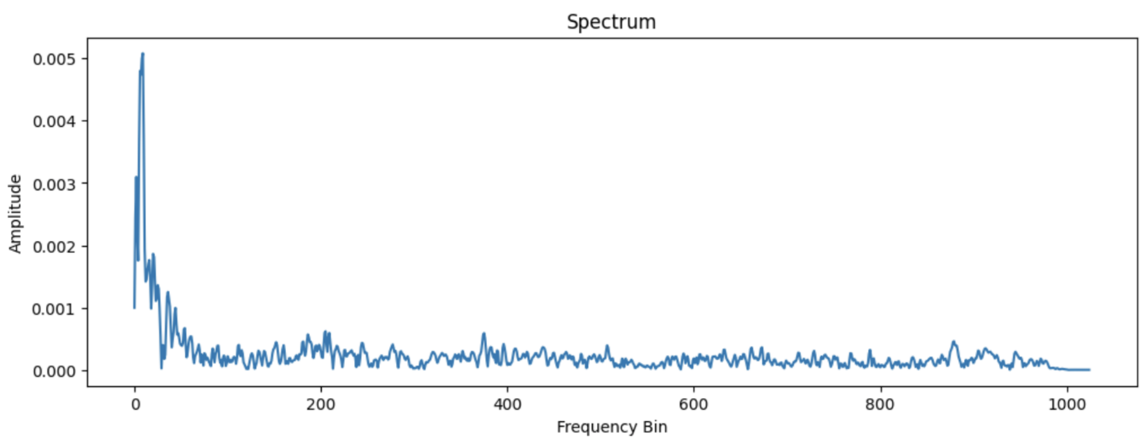
MFCCs are generated through a multistep process. First the audio is sampled at a given frequency, and the Fourier transform is taken from a windowed excerpt of the signal. The powers of this spectrum are mapped on the Mel Scale, which is a scale of perpetual pitch, having a logarithmic mapping from the frequency scale. Then the log is taken of the powers at each Mel frequency. This list of log powers is converted into a signal, and a discrete cosine transform is taken of that signal. The resulting spectrum’s amplitudes are the MFCCs. This complicated process can be greatly simplified using the python package librosa, from which you can access the MFCCs of a given audio signal via librosa.feature.mfcc.

` We also experimented with other possible features which we thought may be useful. One option was taking the spectrogram of the audio file and converting the image into a rgb array. However we found that this did not allow us to accurately predict the emotions using any of the various methods that we tried, so we moved on from this idea.

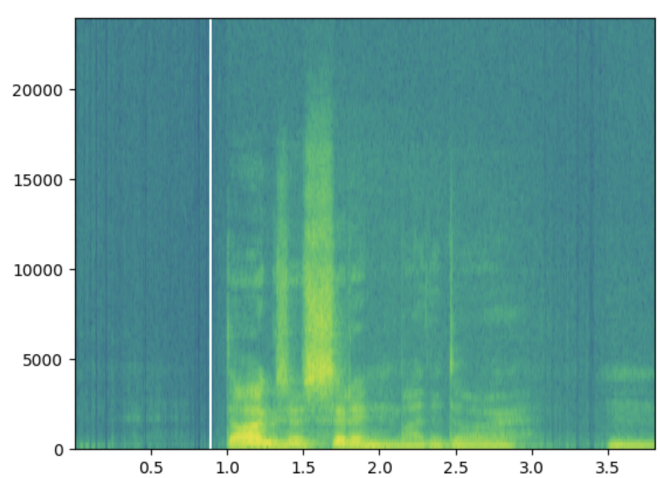
Another feature considered was the spectral centroid. However, since the vector we can access through this demonstrates the spectral centroid of the signal over time, it differs in size between audio files of different lengths. Therefore, we would have to either pad or remove data from each of the vectors, and in either case it makes our data less accurate and therefore more difficult to train on. Other features of audio signals include the “energy” or amplitude variation of the signal, spectral roll-off, spectral flux, spectral entropy, chroma vector/deviation, and pitch.



Audio signal waveform (amplitude vs. time) of calm111.wav.



Audio FFT (amplitude vs. frequency) of calm111.wav.



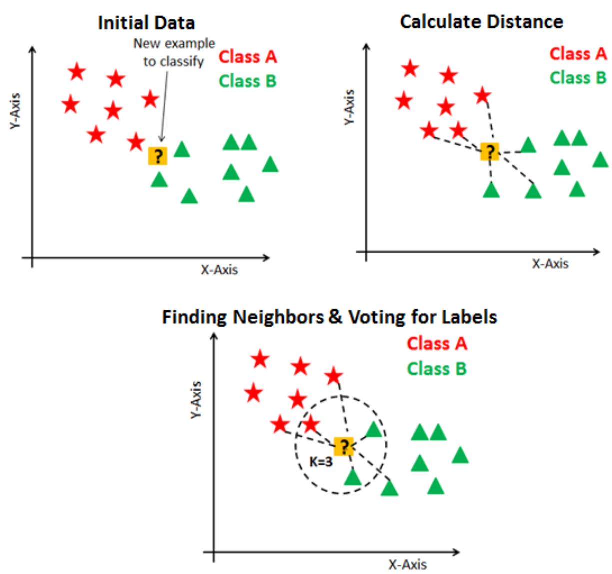
Audio spectrogram of calm111.wav.

Model Selection

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) gave an accuracy of 51.063. This accuracy is good but the worst of the three submitted models. This result can be attributed to several factors, both positive and negative. First, KNN performs well on small datasets, like ours. This is due to the fact that it stores the entire dataset, making no assumptions about the underlying distribution. It uses all available training data to make predictions based on the closest neighbors to the test sample, illustrated in the figure below. Additionally, it handles multi-class classification problems well. KNN doesn't work well with large datasets but thankfully we don't need to worry about this.

As for its weaknesses, KNN is sensitive to the choice of distance metric and K-value. We tried many different K-values to reach an optimum. We stayed with the default distance metric, the 2-norm, but perhaps more experimentation with the distance metric would yield a better accuracy. Also, KNNs accuracy can be affected by imbalanced datasets, where one class has many more samples than the others.



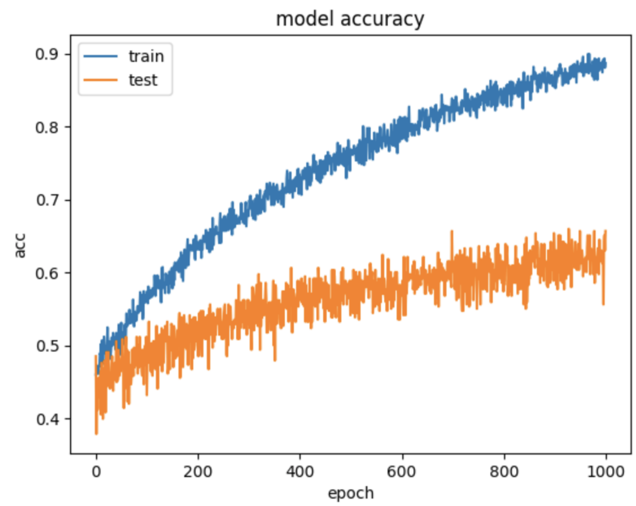
Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) gave an accuracy of 64.893% when submitted to Kaggle. The results from the model on our training data had an accuracy of ~90% which led us to believe that it would be our best model. However, the Kaggle results indicate that it may have been overfitted on the training data.

Despite this, we were still able to achieve a relatively high level of accuracy, and we believe that there are a few reasons. First, CNNs are capable of automatically learning and extracting useful features from raw data via convolution. Convolution is a mathematical operation that involves sliding a filter or kernel over the input data, performing element-wise multiplication between the filter and the input data, and then summing the results to produce a single output value. By repeating this process with different filters at different locations in the input data, CNNs are able to identify relevant patterns and features that are indicative of the emotion expressed in the speech. Additionally, CNNs can be regularized to prevent overfitting, which is particularly important with small datasets like ours. We chose to do regularization via dropout layers, which work by “dropping out” some of the neurons during each training iteration. This prevents overfitting by forcing the network to learn more robust and generalizable features. Perhaps more experimentation regularization layers could have led to a greater accuracy. Lastly, CNNs are attuned to recognizing complex patterns among the training data, which makes it particularly well suited for speech classification tasks. This is because speech signals are complex and contain a large amount of information that is distributed across time and frequency domains. In order to accurately classify speech signals, it is necessary to capture and analyze this information at multiple levels of abstraction.

For our specific convolutional neural network model, the feature extraction and preprocessing steps were kept nearly identical and we were able to optimize the performance and build the model using a few different steps. First, we built a neural network model on the training data using Keras, a high-level neural network API with a TensorFlow backend. We selected a Sequential model so that we could have 2 convolutional layers followed by a fully connected layer and a softmax output layer. These layers were important for avoiding overfitting and improving the generalization of the model. The convolutional layers of the CNNs learn to detect simple features in the lower layers and gradually build up more complex features in the higher layers. This hierarchical representation can be useful in capturing the underlying structure of the data and learning discriminative features for classification. Additionally, we selected CNN for processing MFCCs because it was able to automatically extract relevant and logical features from audio signals, share parameters, and have relatively stable reactions to scaling and variances in pitch and other features.

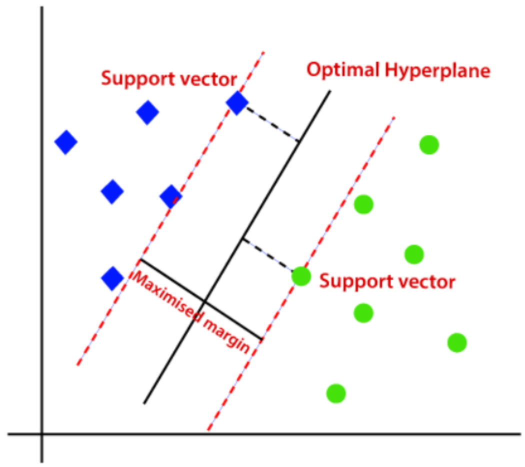
However CNN has some notable drawbacks. First CNN does not work well with small datasets because it uses so many convolutional layers. Our dataset is not particularly large so our CNN algorithm was prone to overfitting. The figure below demonstrates the gap between the overfitted training data and the testing data as the number of epochs go up. Also CNNs, like the other models used, can be computationally expensive with large datasets or deep architectures. While our dataset is not large, it is a relatively deep architecture as many transformations are required to extract more abstract features. It took longer to run than the other models but we still only needed to wait a few seconds. Additionally the model is composed of several layers all with their own parameters. The success of the model is highly dependent on these parameters, and further research was required to determine the optimal parameters for speech classification problems.



Support Vector Machines (SVM)

Support Vector Machines (SVMs) gave an accuracy of 77.659%. This is our most accurate model. Its success can be attributed to several factors. First, SVMs are effective in handling high-dimensional feature spaces, such as that generated by the use of MFCCs. Next, SVMs are particularly useful for binary classification problems, such as distinguishing between positive and negative emotions, and can be extended to handle multi-class classification by using one-vs-rest strategies. Lastly, SVMs are less prone to overfitting than the other algorithms used, because they seek to maximize the margins between the optimal hyperplane and the support vectors, pictured below.

SVMs, like all models, have their drawbacks. One of its weaknesses is that SVMs success is highly dependent on the choice of kernel function and the input parameters. We spent much time trying to figure out an optimal kernel function and tuning the parameters. This tuning caused our accuracy to ultimately increase 15%, but it's possible further tuning would give even more improvement.



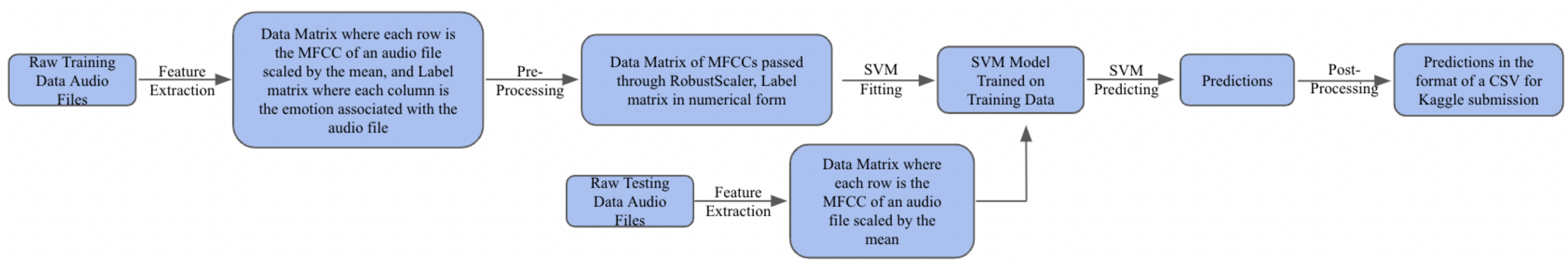
Complete Pipeline

The highest accuracy kaggle submission was achieved with an SVM model trained on the MFCCs of the audio files, with an accuracy of .77659. The first step in this process was feature extraction. We extracted the MFCCs from each file, creating our training data matrix. Each row of this matrix was the MFCC of one of the audio files, resulting in a 1125x34 matrix, as we found ~38 to be the optimal n_MFCC to use in our SVM training. We also created our label matrix, which was a 1x1125 matrix such that the emotion in the ith column of our label matrix was associated with the same audio file as the MFCC in the ith row of our data matrix.

We then pre-processed our data matrix using RobustScaler, from sklearn's preprocessing library, which removes the mean and scales the data according to the quantile range, so that the data scaling isn't affected by outliers. We also converted our label matrix into a numerical matrix, mapping each of the emotions to a different value, 0-7.

Once this was completed our data was all in the correct format for SVM fitting. We used sklearn.svm.SVC to train our model with a regularization parameter, C, of ~20 which we found to produce the most accurate results. This handles multi-class classification with a one-vs-one scheme, which splits the multi-class classification into a binary problem for each pair of classes. Once this model was fitted to our training data, we could predict the labels for our testing data, but first we needed to convert our testing data into the correct format. We created our testing data matrix in the same way we created our training data matrix: by extracting the MFCC from each audio file, resulting in a 315x34 matrix. From this we were able to get a matrix of emotion predictions.

Post-processing was required to get this into the correct format for submission. We first needed to convert the numerical values into strings containing the name of the predicted emotion. After that, we created a CSV file with one row containing the filename and the other containing the predicted emotion associated with that file, as well as a header. This CSV could then be submitted to Kaggle to get our accuracy score.


Raw Training Data Audio Files
Feature Extraction
Data Matrix where each row is the MFCC of an audio file scaled by the mean, and Label matrix where each column is the emotion associated with the audio file
Pre-Processing
Data Matrix of MFCCs passed through RobustScaler, Label matrix in numerical form
SVM Fitting
SVM Model Trained on Training Data
SVM Predicting
Predictions
Post-Processing
Predictions in the format of a CSV for Kaggle submission
Raw Testing Data Audio Files
Feature Extraction
Data Matrix where each row is the MFCC of an audio file scaled by the mean

Conclusions

Altogether, our most accurate final pipeline uses a support vector machine model (from sklearn) with tuned parameters n_MFCC (number of MFCCs to calculate per audio file) of 38 and regularization parameter C of 20. Relative to the accuracy rankings of the other teams in the course, our 77.7% accuracy for the testing data appears to be very good (it is the highest accuracy in the course). We believe that MFCCs were a good feature for distinguishing between emotions since it is built upon the mel scale, which provides a more accurate, albeit nonlinear, representation of the frequency bands in human speech, rather than the linearly-spaced frequency bands in generic audio. This inherent nonlinear transformation of the audio signal better emphasizes or weighs the slight changes in pitch, loudness, and quality of human voice which are essential for predicting emotion.

The limited size of the dataset appears to have limited the accuracy of our convolutional neural network model, while K-nearest neighbors may be sensitive to other irrelevant features in the data, such as the gender of the speaker and the relative volume between clips with the same emotion. From extensive testing and cross validation with the training dataset, the support vector machine decision boundary seems to reliably apply to the testing dataset based upon our Kaggle accuracy scores.

Looking back, we think that it would have been better to explore more of the audio processing libraries available on the internet and the numerous features available for building machine learning models from auditory data. During the first few weeks of the project, we tested various classification models without having invested time in the preprocessing and feature extraction steps, resulting in poor accuracy results. This project has also taught us the importance of good code commenting, documentation, and record-keeping, especially in the context of machine learning, since many of the preprocessing steps and the classification models required precise tuning. Finally, through the writing of this report, we have also realized the importance of taking the time to understand and rationalize our decisions for designing our pipeline, beyond simply plugging data matrices into pre-built functions.

References

Conceptual & Code Sources

1. ELEC 378 Lecture Notes
2. https://towardsdatascience.com/how-i-understood-what-features-to-consider-while-training-audio-files-eedfb6e9002b
3. https://gist.github.com/stevemclaugh/80f192130852353ad53e6d8b6b275983
4. https://zenodo.org/record/1188976#.ZFHfaC-B3T9
5. https://github.com/yfliao/Emotion-Classification-Ravdess/blob/master/EmotionsRecognition.ipynb

6. https://github.com/sarufi-io/AUDIO-ANALYSIS-WITH-LIBROSA/blob/main/Audio%20Analysis%20with%20Librosa.ipynb
7. https://blog.neurotech.africa/audio-analysis-with-librosa/
8. https://devopedia.org/audio-feature-extraction
9. https://www.projectpro.io/article/speech-emotion-recognition-project-using-machine-learning/573
10. https://www.tensorflow.org/tutorials/keras/overfit_and_underfit
11. https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d
12. https://medium.com/@gryangalario/image-classification-using-logistic-regression-on-the-american-sign-language-mnist-9c6522242ddf
13. https://www.youtube.com/watch?v=4ZZrP68yXCI
14. https://jovian.ml/aakashns/03-logistic-regression
15. https://analyticsindiamag.com/how-to-use-logistic-regression-for-image-classification/
16. https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5
17. https://www.analyticsvidhya.com/blog/2022/03/implementing-audio-classification-project-using-deep-learning/
18. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8898841/
19. https://monkeylearn.com/blog/classification-algorithms/

Librosa Sources

20. https://librosa.org/doc/main/generated/librosa.load.html
21. https://librosa.org/doc/main/generated/librosa.feature.rms.html#librosa.feature.rms
22. https://librosa.org/doc/main/generated/librosa.feature.mfcc.html
23.

scikit-learn Sources

24. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
25. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html
26. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
27. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
28. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
29. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Tensorflow & Keras Sources

30. https://keras.io/api/models/
31. https://keras.io/api/layers/
32. https://keras.io/api/optimizers/
33. https://www.tensorflow.org/api_docs/python/tf/keras/losses

Other Dependencies

34. https://pytorch.org/audio/stable/torchaudio.html
35. https://pytorch.org/audio/main/generated/torchaudio.transforms.Spectrogram.html
36. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.specgram.html

ELEC378-PROJECT-FINAL

May 2, 2023

1 ELEC 378 FINAL PROJECT: SPEECH EMOTION CLASSIFICATION

- Team JARL
- Jasmine Lee, Arielle Sanford, Robert Heeter, Lindsey Russ
- ELEC 378: Machine Learning: Concepts & Techniques
- Rice University
- Submitted 2 May 2023

[ ]:
```
import numpy as np
import os
import matplotlib.pyplot as plt

import librosa
```

1.1 Get training dataset and calculate MFCCs

[ ]:
```
directory = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/')
data = np.empty((1125, 2), dtype=object)

emotion_to_id = {
    "angry" : 0,
    "calm" : 1,
    "disgust" : 2,
    "fearful" : 3,
    "happy" : 4,
    "neutral" : 5,
    "sad" : 6,
    "surprised" : 7
}

i = 0
for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
```

```
    if os.path.isfile(f):
        emotion = filename[:len(filename)-7]
        data[i][0] = "/"+filename
        data[i][1] = int(emotion_to_id[emotion])
        i += 1

def make_mfcc(file, n_mfcc):
    sig, sr = librosa.load(file)
    sig_mfcc = librosa.feature.mfcc(y=sig, sr=sr, n_mfcc=n_mfcc, S=None,␣
 ↪htk=True)
    sig_mfcc_avg = np.mean(sig_mfcc, axis=1)

    return sig_mfcc_avg

n_mfcc = 38

X = np.empty((len(data), n_mfcc), dtype=float)
y = np.empty((len(data)), dtype=int)

for i in range(len(data)):
    file = directory + data[i][0]
    X[i] = make_mfcc(file, n_mfcc=n_mfcc)
    y[i] = data[i][1]

X_train = X
y_train = y
```

1.2 Get testing dataset and calculate MFCCs (FOR KAGGLE)

[ ]:
```
# directory = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/test/test/')
# data = np.empty((315, 2), dtype=object)

# emotion_to_id = {
#     "angry" : 0,
#     "calm" : 1,
#     "disgust" : 2,
#     "fearful" : 3,
#     "happy" : 4,
#     "neutral" : 5,
#     "sad" : 6,
#     "surprised" : 7
# }

# i = 0
```

```
# for filename in os.listdir(directory):
#     f = os.path.join(directory, filename)

#     if os.path.isfile(f):
#         data[i][0] = "/"+filename
#         i += 1

# def make_mfcc(file, n_mfcc):
#     sig, sr = librosa.load(file)
#     sig_mfcc = librosa.feature.mfcc(y=sig, sr=sr, n_mfcc=n_mfcc, S=None,␣
↪htk=True)
#     sig_mfcc_avg = np.mean(sig_mfcc, axis=1)

#     return sig_mfcc_avg

# n_mfcc = 3

# X = np.empty((len(data), n_mfcc), dtype=float)

# for i in range(len(data)):
#     file = directory + data[i][0]
#     X[i] = make_mfcc(file, n_mfcc=n_mfcc)

# X_test = X
```

1.3 Split train/test data (NOT FOR KAGGLE)

[ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,␣
↪test_size=0.2, random_state=42)
```

1.4 Support vector machine (SVM)

[ ]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

clf = make_pipeline(RobustScaler(), SVC(C=20, tol=0.001))
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"n_mfcc: {n_mfcc}, c: {c}, acc: {accuracy*100}%")
```

1.5 Multilayer perceptron (MLP)

```
[ ]: from sklearn.neural_network import MLPClassifier

mlp_params = {'activation': 'relu',
              'solver': 'lbfgs',
              'hidden_layer_sizes': 1283,
              'alpha': 0.3849485717707319,
              'batch_size': 163,
              'learning_rate': 'constant',
              'max_iter':1000}

clf = make_pipeline(RobustScaler(), MLPClassifier(**mlp_params))
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"{accuracy*100}%")
```

1.6 Convolutional neural network (CNN)

```
[ ]: import tensorflow as tf
import keras
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D,␣
↪BatchNormalization
from keras.models import Model
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import confusion_matrix

model = Sequential()

# model2.add(layers.Conv2D(64, (4, 4), activation='relu',␣
↪kernel_regularizer=regularizers.l2(l=0.01)))
# model2.add(layers.MaxPooling2D((2, 2)))# Hidden Layer 2
# model2.add(layers.Conv2D(128, (3, 3), activation='relu',␣
↪kernel_regularizer=regularizers.l2(l=0.01)))
# model2.add(layers.MaxPooling2D((2,2)))
```

```
# model3.add(layers.Conv2D(32, (3, 3), activation='relu',␣
↪kernel_initializer='he_normal', input_shape=(96, 96, 3)))

# random.seed(123) # Establish Consistency in resultsmodel4 = Sequential() #␣
↪Instantiate the 4th Modelmodel4.add(layers.Conv2D(32, (3, 3),␣
↪activation=’relu’, input_shape=(96, 96, 3)))
# model4.add(layers.MaxPooling2D((2, 2)))
# model4.add(Dropout(0.4))
# model4.add(layers.Conv2D(64, (4, 4), activation='relu'))
# model4.add(layers.MaxPooling2D((2, 2)))
# model4.add(Dropout(0.4)) # Flattening- Convert 2D matrix to a 1D vector
# model4.add(layers.Flatten())
# model4.add(layers.Dense(512, activation = 'relu'))
# model4.add(Dropout(0.2))
# model4.add(layers.Dense(1, activation='sigmoid'))

# model5.add(layers.Conv2D(32, (3, 3), activation='relu',␣
↪kernel_constraint=unit_norm(), input_shape=(96, 96, 3)))


# model.add(Conv1D(128, 16, padding='same', input_shape=(40,1)))
# model.add(Activation('relu'))

# model.add(Conv1D(128, 16, padding='same', input_shape=(40,1)))
# model.add(BatchNormalization())
# model.add(Activation('relu'))
# # model.add(Dropout(0.4))

# model.add(MaxPooling1D(pool_size=8))

# model.add(Conv1D(128, 16, padding='same', input_shape=(40,1)))
# model.add(Activation('relu'))

# model.add(Conv1D(128, 16, padding='same', input_shape=(40,1)))
# model.add(Activation('relu'))

# model.add(Conv1D(128, 16, padding='same', input_shape=(40,1)))
# model.add(Activation('relu'))

# # model.add(Conv1D(128, 16, padding='same', input_shape=(40,1)))
# # model.add(BatchNormalization())
# # model.add(Activation('relu'))

# # model.add(MaxPooling1D(pool_size=5))

# # model.add(Conv1D(128, 5, padding='same', input_shape=(40,1)))
# # model.add(Activation('relu'))
```

```
# # model.add(Conv1D(128, 5, padding='same', input_shape=(40,1)))
# # model.add(Activation('relu'))

# # model.add(Dropout(0.2))
# model.add(Flatten())
# # model.add(Dense(10, kernel_regularizer='l2', bias_regularizer='l2'))
# model.add(Dense(8))
# # model.add(Activation('softmax'))

# opt = keras.optimizers.RMSprop(learning_rate=0.0001, rho=0.9, epsilon=None,␣
↪decay=0.0)


# model.add(Conv1D(128, 5, padding='same', input_shape=(40,1)))
# model.add(Activation('relu'))
# model.add(Dropout(0.1))
# model.add(MaxPooling1D(pool_size=(8)))
# model.add(Conv1D(128, 5, padding='same',))
# model.add(Activation('relu'))
# model.add(Dropout(0.1))
# model.add(Flatten())
# model.add(Dense(10))
# model.add(Activation('softmax'))
# opt = keras.optimizers.RMSprop(learning_rate=0.0005, rho=0.9, epsilon=None,␣
↪decay=0.0)

model.add(Conv1D(128, 5,padding='same', input_shape=(34,1)))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(128, 5,padding='same',))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(10))
model.add(Activation('softmax'))
opt = keras.optimizers.RMSprop(lr=0.00005, rho=0.9, epsilon=None, decay=0.0)
```

[ ]:
```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

[ ]:
```
X_train_cnn = np.expand_dims(X_train, axis=2)
X_test_cnn = np.expand_dims(X_test, axis=2)
```

```
cnnhistory = model.fit(X_train_cnn, y_train, batch_size=4, epochs=60,␣
↪validation_data=(X_test_cnn, y_test))
```

[ ]:
```
plt.plot(cnnhistory.history['loss'])
plt.plot(cnnhistory.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

[ ]:
```
plt.plot(cnnhistory.history['accuracy'])
plt.plot(cnnhistory.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

[ ]:
```
loss, accuracy = model.evaluate(X_test_cnn, y_test)
print(f"{accuracy*100}%")

y_pred = model.predict(X_test_cnn)
print(np.shape(y_pred))
y_pred = np.argmax(y_pred, axis=1)
print(y_pred)

accuracy = accuracy_score(y_test, y_pred)
print(f"{accuracy*100}%")
```

1.7 Logistic regression

[ ]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.linear_model import LogisticRegression

clf = make_pipeline(RobustScaler(), LogisticRegression(tol=0.00001,␣
↪max_iter=10000))
clf.fit(X_train, y_train)
```

1.8 k-nearest neighbors

[ ]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.neighbors import KNeighborsClassifier
```

```
clf = make_pipeline(RobustScaler(), KNeighborsClassifier(n_neighbors = 8))
clf.fit(X_train, y_train)
```

1.9 Other models

[ ]:
```
import pandas as pd
import numpy as np
import os
import random
import sys
import glob
import librosa
import librosa.display
import matplotlib.pyplot as plt
# import seaborn as sns

# from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
# from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.utils.multiclass import unique_labels
from sklearn.neural_network import MLPClassifier
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.ensemble import RandomForestClassifier, VotingClassifier

# import lightgbm as lgb
# import xgboost as xgb
# import optuna
# from tqdm import tqdm

# import warnings
# warnings.filterwarnings('ignore')
```

[ ]:
```
def extract_feature(file_name):
    """Function Extracts Features from WAV file"""
    X, sample_rate = librosa.load(file_name)
    stft=np.abs(librosa.stft(X))
    result=np.array([])
    mfccs=np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)
    result=np.hstack((result, mfccs))
    chroma=np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
    result=np.hstack((result, chroma))
    mel=np.mean(librosa.feature.melspectrogram(y=X, sr=sample_rate).T,axis=0)
    result=np.hstack((result, mel))
    return result
```

[ ]:
```
emotions = {
    "angry" : 0,
    "calm" : 1,
    "disgust" : 2,
    "fearful" : 3,
    "happy" : 4,
    "neutral" : 5,
    "sad" : 6,
    "surprised" : 7
}
```

[ ]:
```
def load_data(test_size=0.2):
    x,y=[],[]

    directory = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/')


    for filename in os.listdir(directory):
        file = os.path.join(directory, filename)

        emotion=emotions[filename[:len(filename)-7]]
        feature=extract_feature(file)
        x.append(feature)
        y.append(emotion)

    return train_test_split(np.array(x), y, test_size=test_size, random_state=9)
```

[ ]:
```
X_train, X_test, y_train, y_test = load_data()
print((X_train.shape[0], X_test.shape[0]))

# np.set_printoptions(threshold=np.inf)
print(np.shape(X_test))
print(X_train[0])

print(f'Features extracted: {X_train.shape[1]}')
```

[ ]:
```
# scaler = StandardScaler()
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.transform(X_test)
```

[ ]:
```
mlp_params = {'activation': 'relu',
              'solver': 'lbfgs',
              'hidden_layer_sizes': 1283,
              'alpha': 0.3849485717707319,
              'batch_size': 163,
              'learning_rate': 'constant',
```

```
                'max_iter':1000}
```

[ ]:
```
# clf_model = MLPClassifier(**mlp_params)
# clf_model.fit(X_train, y_train)
# y_pred = clf_model.predict(X_test)

from sklearn import preprocessing

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.metrics import accuracy_score


scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)

scaler = StandardScaler().fit(X_test)
X_test = scaler.transform(X_test)

clf = MLPClassifier(**mlp_params)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)


# y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(y_pred)
print(f"{accuracy*100}%")
```

[ ]:
```
v4_params = {'estimators':[('mlp', models['mlp']),
                           ('xgb', models['xgb'])],
            'voting':'soft'}

from sklearn.ensemble import VotingClassifier

scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)

scaler = StandardScaler().fit(X_test)
X_test = scaler.transform(X_test)

clf = make_pipeline(RobustScaler(), VotingClassifier(**v4_params))
clf.fit(X_train, y_train)
y_pred = clf_model.predict(X_test)


# y_pred = clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(y_pred)
print(f"{accuracy*100}%")
```

1.10 Exporting predictions (FOR KAGGLE)

[ ]:
```
id_to_emotion = dict((v, k) for k, v in emotion_to_id.items())
y_pred = [id_to_emotion[x] for x in y_pred]

print(np.shape(y_pred))
print(y_pred)

import pandas as pd
names = [x[1:len(x)-4] for x in data[:,0]]

df = pd.DataFrame(list(zip(names, y_pred)), columns=['filename', 'label'])
df.to_csv("y_kaggle_svm16.csv", index=False)
```

ELEC378-PROJECT-V3

May 2, 2023

1 ELEC 378 FINAL PROJECT more stuffs: SPEECH EMOTION CLASSIFICATION TESTING & STUFF C:

- Team JARL
- Jasmine Lee, Arielle Sanford, Robert Heeter, Lindsey Russ
- ELEC 378: Machine Learning: Concepts & Techniques
- Rice University

- Submitted 2 May 2023

1.1 Playing with waveforms, FFTs, and spectrograms

[ ]:
```
import numpy as np
import os
import matplotlib.pyplot as plt

import scipy.io.wavfile as wavfile
```

[ ]:
```
directory = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/calm111.
 ↪wav')

Fs, aud = wavfile.read(directory)
# select left channel only
# aud = aud[:,0]
# trim the first 125 seconds
print(Fs)
first = aud[:int(Fs*4)]
print(np.shape(first))
# np.set_printoptions(threshold=np.inf)
```

[ ]:
```
spectrum, freqs, t, im = plt.specgram(first, Fs=Fs)
# print(np.shape(powerSpectrum))
# print(powerSpectrum)
print(np.shape(spectrum))
print(np.shape(freqs))
```

```
print(np.shape(t))

print(np.max(spectrum))

plt.show()
plt.imshow(spectrum, cmap = 'gray')
plt.colorbar()
plt.show()
```

[ ]:
```
import librosa as lb
import os
import IPython.display as ipd
```

[ ]:
```
directory = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/calm111.
 ↪wav')

ipd.Audio(directory)
```

[ ]:
```
data, sampling_rate = lb.load(directory)
```

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
lb.display.waveshow(data, sr=sampling_rate)
plt.show()
```

[ ]:
```
n_fft = 2048
plt.figure(figsize=(12, 4))
ft = np.abs(lb.stft(data[:n_fft], hop_length = n_fft+1))
plt.plot(ft);
plt.title('Spectrum');
plt.xlabel('Frequency Bin');
plt.ylabel('Amplitude');
```

[ ]:
```
n_fft = 2048

plt.figure()

fig, axs = plt.subplots(8, 1, figsize=(20,20))

# fig.figsize([10,10])

# plt.xlim([0,1024])
# plt.ylim([0,0.5])
```

```python
# axs[0]

# plt.title('Spectrum');
# plt.xlabel('Frequency Bin');
# plt.ylabel('Amplitude');


ft_data = np.empty([1200,1025], dtype=float)


# angry = np.empty([0,1025], dtype=float)
# calm = np.empty([0,1025], dtype=float)
# disgust = np.empty([0,1025], dtype=float)
# fearful = np.empty([0,1025], dtype=float)
# happy = np.empty([0,1025], dtype=float)
# neutral = np.empty([0,1025], dtype=float)
# sad = np.empty([0,1025], dtype=float)
# surprised = np.empty([0,1025], dtype=float)

directory = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/')
i = 0

for filename in os.listdir(directory):
    f = os.path.join(directory, filename)

#     if os.path.isfile(f):

    emotion = filename[:len(filename)-7]
#         trial_data[count][0] = "/"+emotion
#         trial_data[count][1] = emotion
#         count += 1

#         plt.title(emotion)

    data, sampling_rate = lb.load(f)
    ft = np.abs(lb.stft(data[:n_fft], hop_length = n_fft+1))

#     print(np.shape(ft))
#     print(np.shape(ft_data[i:i+1,:]))
    ft_data[i:i+1,:] = ft.T


#     print(np.shape(ft_data))
    ft_data = ft_data.reshape((8,150,1025))

    ft_avgs = np.mean(ft_data, axis=1)
```

```
#     if emotion == 'angry':
#         i = 0
#         axs[i].set_title(emotion)
#         ft_data = np.stack((ft_data, ft), axis=2

#     elif emotion == 'calm':
#         i = 1
#         axs[i].set_title(emotion)
#         calm = np.vstack((calm, ft))

#     elif emotion == 'disgust':
#         i = 2
#         axs[i].set_title(emotion)
#         disgust = np.vstack((disgust, ft))

#     elif emotion == 'fearful':
#         i = 3
#         axs[i].set_title(emotion)
#         fearful = np.vstack((fearful, ft))

#     elif emotion == 'happy':
#         i = 4
#         axs[i].set_title(emotion)
#         happy = np.vstack((happy, ft))

#     elif emotion == 'neutral':
#         i = 5
#         axs[i].set_title(emotion)
#         neutral = np.vstack((neutral, ft))

#     elif emotion == 'sad':
#         i = 6
#         axs[i].set_title(emotion)
#         sad = np.vstack((sad, ft))

#     elif emotion == 'surprised':
#         i = 7
#         axs[i].set_title(emotion)
#         surprised = np.vstack((surprised, ft))

    i += 1

for i in range(8):
#     if i == 0:
#         axs[i].set_title('angry')
```

```
#     if i == 1:


    axs[i].plot(ft_avgs[i,:])
    axs[i].set_xlim(0, 1024)
    axs[i].set_ylim(0, 0.00005)
    axs[i].set_xlabel('Frequency Bin')
    axs[i].set_ylabel('Amplitude')

plt.show()
```

[ ]:
```
a = np.arange(64).reshape((8,8))
print(a)
print(a.reshape(2,4,8))
```

[ ]:
```

```

[ ]:
```
wav_pathname = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/sad103.
 ↪wav')

y, sr = librosa.load(wav_pathname)

S = librosa.feature.melspectrogram(y, sr, n_mels=128)
log_S = librosa.logamplitude(S, ref_power=np.max)
plt.figure(figsize=(12,4))

librosa.display.specshow(log_S, sr=sr, x_axis='time', y_axis='mel')
plt.title('mel power spectrogram')
plt.colorbar(format='%+02.0f dB')
plt.tight_layout()

# y, sr = lb.load(directory)
# print(np.shape(y))
# y = lb.util.fix_length(y, size=90000)
# # print(np.shape(y))
# mfcc = lb.feature.mfcc(y=y, sr=22050, hop_length = 1024, n_mfcc=20, htk=True)
# print(np.shape(mfcc))
# plt.figure(figsize=(20,20))
# plt.imshow(mfcc, cmap='hot', interpolation='nearest')
# plt.show()
```

[ ]:
```
directory = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/')

for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
```

```
    if os.path.isfile(f):
        name = filename[:len(filename)-7]
        test[count][0] = "/"+filename
        test[count][1] = name
        count += 1

df = pd.DataFrame(test)
df.rename(columns={0: "relative_path", 1: "classID"}, inplace=True)
filename = df['relative_path'].to_list()
val_ds = SoundDS(df, directory)
```

[ ]:

[ ]:

[ ]:

[ ]:
```
import IPython.display as ipd
import numpy as np
import os
import librosa
import librosa.display
import matplotlib.pyplot as plt
```

[ ]:
```
# getting training data

data = np.empty((1125, 2), dtype=object)

count = 0

directory = os.path.join(os.
 ↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/')

for filename in os.listdir(directory):
    f = os.path.join(directory, filename)

    if os.path.isfile(f):
        name = filename[:len(filename)-7]
        data[count][0] = "/"+filename
        data[count][1] = name
        count += 1
```

[ ]:
```
# getting test data

data = np.empty((315, 2), dtype=object)
```

```
count = 0

directory = os.path.join(os.
↪getcwd(),'elec-378-sp2023-speech-emotion-classification/data/data/')

for filename in os.listdir(directory):
    f = os.path.join(directory, filename)

    if os.path.isfile(f):
        name = filename[:len(filename)-7]
        data[count][0] = "/"+filename
        data[count][1] = name
        count += 1
```

[ ]:

```

```

[ ]:

```
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.neighbors import KNeighborsClassifier

#training
clf = make_pipeline(RobustScaler(), KNeighborsClassifier(n_neighbors = 5))
# change this to X, y instead of X_train_split, y_train_split if doing Kaggle
clf.fit(X_train, y_train)
```

[ ]:

```
from sklearn.metrics import accuracy_score
# Predict on testing data
#Change this to Xval if doing Kaggle
y_pred = clf.predict(test_data_matrix)

print(y_pred)
# # Calculate accuracy
# accuracy = accuracy_score(y_test, y_pred)
# print("Accuracy:", accuracy)
```

[ ]:

```
#Convert to kaggle upload
number_to_label = {v: k for k, v in label_to_number.items()}
# Map each number back to its corresponding label
label = [number_to_label[number] for number in y_pred]
print(names)
```

```
[ ]: import os
import pandas as pd
os.chdir('/content/drive/MyDrive/2022-2023 Semester 2/Elec 378 Final Project')
print(os.getcwd())
df = pd.DataFrame(list(zip(names, label)), columns=['filename', 'label'])
df.to_csv("y_kaggle1.csv", index=False)
```

1.2 Earlier testing with a preprocessing library and KNN

```
[ ]: from torch.utils.data import random_split
from torch.utils.data import DataLoader, Dataset, random_split
import torchaudio

import math
import random
import torch
import torchaudio
from torchaudio import transforms
from IPython.display import Audio
import pandas as pd
import numpy as np
import os
```

```
[ ]: class AudioUtil():
  # ----------------------------
  # Load an audio file. Return the signal as a tensor and the sample rate
  # ----------------------------
    def open(audio_file):
        sig, sr = torchaudio.load(audio_file)
        return (sig, sr)

    def rechannel(aud, new_channel):
        sig, sr = aud
        if (sig.shape[0] == new_channel):
            # Nothing to do
            return aud

        if (new_channel == 1):
            # Convert from stereo to mono by selecting only the first channel
            resig = sig[:1, :]
        else:
            # Convert from mono to stereo by duplicating the first channel
            resig = torch.cat([sig, sig])

        return ((resig, sr))

    def resample(aud, newsr):
```

```python
        sig, sr = aud

        if (sr == newsr):
            # Nothing to do
            return aud

        num_channels = sig.shape[0]
        # Resample first channel
        resig = torchaudio.transforms.Resample(sr, newsr)(sig[:1, :])
        if (num_channels > 1):
            # Resample the second channel and merge both channels
            retwo = torchaudio.transforms.Resample(sr, newsr)(sig[1:, :])
            resig = torch.cat([resig, retwo])

        return ((resig, newsr))

    def pad_trunc(aud, max_ms):
        sig, sr = aud
        num_rows, sig_len = sig.shape
        max_len = sr//1000 * max_ms

        if (sig_len > max_len):
            # Truncate the signal to the given length
            sig = sig[:, :max_len]

        elif (sig_len < max_len):
            # Length of padding to add at the beginning and end of the signal
            pad_begin_len = random.randint(0, max_len - sig_len)
            pad_end_len = max_len - sig_len - pad_begin_len

            # Pad with 0s
            pad_begin = torch.zeros((num_rows, pad_begin_len))
            pad_end = torch.zeros((num_rows, pad_end_len))

            sig = torch.cat((pad_begin, sig, pad_end), 1)

        return (sig, sr)

    def time_shift(aud, shift_limit):
        sig, sr = aud
        _, sig_len = sig.shape
        shift_amt = int(random.random() * shift_limit * sig_len)
        return (sig.roll(shift_amt), sr)

    def spectro_gram(aud, n_mels=64, n_fft=1024, hop_len=None):
        sig, sr = aud
        top_db = 80
```

```
        # spec has shape [channel, n_mels, time], where channel is mono, stereo␣
↪etc
        spec = transforms.MelSpectrogram(
            sr, n_fft=n_fft, hop_length=hop_len, n_mels=n_mels)(sig)

        # Convert to decibels
        spec = transforms.AmplitudeToDB(top_db=top_db)(spec)
        return (spec)

    def spectro_augment(spec, max_mask_pct=0.1, n_freq_masks=1, n_time_masks=1):
        _, n_mels, n_steps = spec.shape
        mask_value = spec.mean()
        aug_spec = spec

        freq_mask_param = max_mask_pct * n_mels
        for _ in range(n_freq_masks):
            aug_spec = transforms.FrequencyMasking(
                freq_mask_param)(aug_spec, mask_value)

        time_mask_param = max_mask_pct * n_steps
        for _ in range(n_time_masks):
            aug_spec = transforms.TimeMasking(
                time_mask_param)(aug_spec, mask_value)

        return aug_spec
```

[ ]:
```
class SoundDS(Dataset):
    def __init__(self, df, data_path):
        self.df = df
        self.data_path = str(data_path)
        self.duration = 4000
        self.sr = 44100
        self.channel = 2
        self.shift_pct = 0.4

    # ----------------------------
    # Number of items in dataset
    # ----------------------------
    def __len__(self):
        return len(self.df)

    # ----------------------------
    # Get i'th item in dataset
    # ----------------------------
    def __getitem__(self, idx):
```

```
        # Absolute file path of the audio file - concatenate the audio␣
↪directory with
        # the relative path
        audio_file = self.data_path + self.df.loc[idx, 'relative_path']
        # Get the Class ID
        class_id = self.df.loc[idx, 'classID']

        aud = AudioUtil.open(audio_file)
        # Some sounds have a higher sample rate, or fewer channels compared to␣
↪the
        # majority. So make all sounds have the same number of channels and same
        # sample rate. Unless the sample rate is the same, the pad_trunc will␣
↪still
        # result in arrays of different lengths, even though the sound duration␣
↪is
        # the same.
        reaud = AudioUtil.resample(aud, self.sr)
        rechan = AudioUtil.rechannel(reaud, self.channel)

        dur_aud = AudioUtil.pad_trunc(rechan, self.duration)
        shift_aud = AudioUtil.time_shift(dur_aud, self.shift_pct)
        sgram = AudioUtil.spectro_gram(
            shift_aud, n_mels=64, n_fft=1024, hop_len=None)
        aug_sgram = AudioUtil.spectro_augment(
            sgram, max_mask_pct=0.1, n_freq_masks=2, n_time_masks=2)

        return aug_sgram, class_id
```

[ ]:
```
#Spectral Centroid (just an experiment on another feature of audio)

import sklearn
import librosa
import matplotlib.pyplot as plt
def spectralCentroid(wavfile):
    x, sr = librosa.load(wavfile)
    spectral_centroids = librosa.feature.spectral_centroid(y=x)
    return spectral_centroids.flatten().shape

data = spectralCentroid(wavfile = "/content/drive/MyDrive/2022-2023 Semester 2/
↪Elec 378 Final Project/data/angry000.wav")
directory = "/content/drive/MyDrive/2022-2023 Semester 2/Elec 378 Final Project/
↪data"

print(data)
```

[ ]:
```python
# A different way to get the spectrogram (also an experiment, don't need to run)
def graph_spectrogram(wav_file):
    name = wav_file.split('/')[-1]
    name = name[:len(name)-7]
    rate, data = wavfile.read(wav_file)
    if(data.ndim > 1):
        data = data[:, 0]
    powerSpectrum, frequenciesFound, time, image = plt.specgram()
    r = plt.gcf().canvas.get_renderer()
    pic, x, y, trans = image.make_image(r)
    return pic
```

[ ]:
```python
#getting training data
data = np.empty((1125, 2), dtype=object) #or 1127?
count = 0

directory = "/content/drive/MyDrive/2022-2023 Semester 2/Elec 378 Final Project/
↪data"
for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    # checking if it is a file

    if os.path.isfile(f):
        name = filename[:len(filename)-7]
        data[count][0] = "/"+filename
        data[count][1] = name
        count += 1

df = pd.DataFrame(data)
df.rename(columns={0: "relative_path", 1: "classID"}, inplace=True)
train_ds = SoundDS(df, directory)
```

[ ]:
```python
#Only use this if you want to test attempts, this is not what you use for Kaggle
# Random split of 80:20 between training and validation
num_items = len(train_ds)
num_train = round(num_items * 0.8)
num_val = num_items - num_train
train_ds, val_ds = random_split(train_ds, [num_train, num_val])

##Training data
X = []
y = []

for i in range(num_train):
    sample = train_ds[i]
    X.append(sample[0])
    y.append(sample[1])
```

```
X_train_split = np.concatenate([spectrogram.flatten().reshape(1, -1) for␣
↪spectrogram in X])

label_to_number = {}
for label in set(y):
    label_to_number[label] = len(label_to_number)

# Map each label to its corresponding number
y_train_split = [label_to_number[label] for label in y]

##Testing data
Xval = []
yval = []
for i in range(len(val_ds)):
    sample = val_ds[i]
    Xval.append(sample[0])
    yval.append(sample[1])

X_test_split = np.concatenate([spectrogram.flatten().reshape(1, -1) for␣
↪spectrogram in Xval])
y_test_split = [label_to_number[label] for label in yval]
print(X_train_split)
print(y_train_split)
print(X_test_split)
print(y_test_split)
```

[ ]:
```
#getting validation (testing) data
test = np.empty((315, 2), dtype=object)
count = 0
directory = "/content/drive/MyDrive/2022-2023 Semester 2/Elec 378 Final Project/
↪test"
for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    # checking if it is a file

    if os.path.isfile(f):
        name = filename[:len(filename)-7]
        test[count][0] = "/"+filename
        test[count][1] = name
        count += 1

df = pd.DataFrame(test)
df.rename(columns={0: "relative_path", 1: "classID"}, inplace=True)
filename = df['relative_path'].to_list()
val_ds = SoundDS(df, directory)
```

[ ]:
```
##Training data (USE THIS FOR KAGGLE)
X = []
y = []

for i in range(len(train_ds)):
    sample = train_ds[i]
    X.append(sample[0])
    y.append(sample[1])

X_train = np.concatenate([spectrogram.flatten().reshape(1, -1) for spectrogram␣
 ↪in X])

label_to_number = {}
for label in set(y):
    label_to_number[label] = len(label_to_number)

# Map each label to its corresponding number
y_train = [label_to_number[label] for label in y]
```

[ ]:
```
##Testing data (USE THIS FOR KAGGLE)
Xval = []
# yval = []
for i in range(len(val_ds)):
    sample = val_ds[i]
    Xval.append(sample[0])
    #yval.append(sample[1])

X_test = np.concatenate([spectrogram.flatten().reshape(1, -1) for spectrogram␣
 ↪in Xval])
```

[ ]:
```
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.neighbors import KNeighborsClassifier

#training
clf = make_pipeline(RobustScaler(), KNeighborsClassifier(n_neighbors = 5))
# change this to X, y instead of X_train_split, y_train_split if doing Kaggle
clf.fit(X_train_split, y_train_split)
```

[ ]:
```
from sklearn.metrics import accuracy_score
# Predict on testing data
#Change this to Xval if doing Kaggle
y_pred = clf.predict(X_test_split)

# Calculate accuracy
accuracy = accuracy_score(y_test_split, y_pred)
```

```
print("Accuracy:", accuracy)
```

[ ]:
```
#Convert to kaggle upload
number_to_label = {v: k for k, v in label_to_number.items()}
# Map each number back to its corresponding label
label = [number_to_label[number] for number in y_pred]
clean_filename = [file[1:-4] for file in filename]
```

[ ]:
```
import os
os.chdir('/content/drive/MyDrive/elec378 final proj')
print(os.getcwd())
df = pd.DataFrame(list(zip(clean_filename, label)), columns=['filename',␣
↪'label'])
df.to_csv("y_kaggle.csv", index=False)
```