EE603A- Project Report

1st Saksham Mehra
190742

2nd Abhishek Yadav
190044

***Abstract*—This report explains the complete process and steps we have followed in the Audio Tagging and event detection task assigned for the project. We begin with audio data pre-processing techniques followed by the various machine learning models we have deployed. The methods discussed in this report are Linear ML Models, CNN , RNN , Neural Networks using back-propagation, GMM and K-means clustering. We have also discussed some post-processing techniques for converting model results to desired form.**

I. Introduction

As per the project statement we were supposed to apply our machine learning model on an unknown audio clip spectrogram of 10 seconds. We had to detect and report the presence of speech and/or music in the given clip along with the time stamps of their occurrence.

II. Pre-Processing

A. Data Collection

All the raw data was prepared and collected by the group members.For speech , we recorded audio samples of different people in .mp3 format and converted that to .wav format. Both male and female voices were sampled at various speaking pace. For music, we collected data from the internet, of varied genres and musical instruments. We also recorded silence (absence of both speech and music) with some background noises . For handling unknown data white noise , we added White Gaussian and Uniform Gaussian noise to some samples of each kind. Generation and addition of noise was done using Audacity software.

B. Data Annotation and labelling

Pure data of a single class was sufficient to train models which do not require temporal data. These include **GMM, CNN , K-Means** ,etc. In this kind of data , one sample contained only a single class. However we needed impure samples(data from all three classes) to capture time evolution in the test set. We wrote a **python script** to take samples from different classes randomly and merge them together to a 10 second audio file using numpy. Data labels and time stamps were stored in two different numpy arrays(one for each). We also tried generating this kind of impure data manually my merging audio files using Audacity software and annotating them manually(time stamps and class labels), but this consumed a lot of time.

C. Feature Extraction

We primarily used MFCC features for training and testing our models. Since the data was given in the form of a log power spectrum, we had to do some mathematical operations to obtain the **MFCC** features.We first converted the decibel power spectrum to magnitude power spectrum using ***librosa.db_to_power()*** . Then we converted this power spectrum to a **Mel-Spectrogram** using ***librosa.feature.melspectrogram()*** followed by its conversion to a **Decibel Mel spectrogram** using ***librosa.power_to_db()*** . This can finally be used to generate required number of MFCC features using **librosa.feature.mfcc()**.

The choice of number of features is a hyperparameter and depends on the type of model involved. We also do not want to make our model over complicated by using too many features. The problem will be handled later on case to case basis. Usually in all models we have ignored the first feature , since it will be drastically different even for same classes due to scaling issues(given decibel power spectrograms are scaled by an unknown value, more precisely the maximum value of the spectrogram). Other features are not affected the scaling problems.
Capturing temporal information is an important aspect of audio event detection. For this purpose at several places we have also used the **first and the second order differences of the MFCC features** to study their evolution over time. This is similar to the concept of velocity and acceleration.

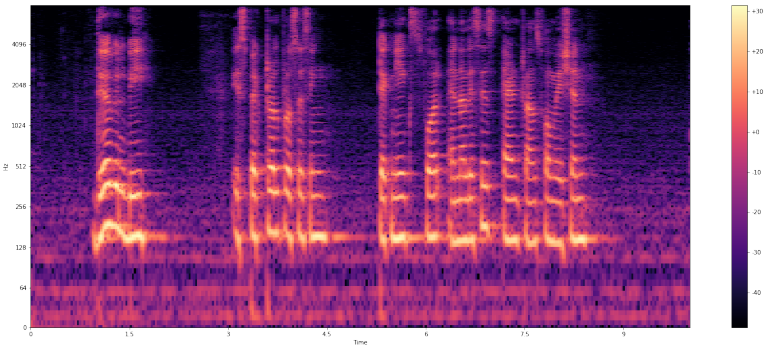

Fig. 1. An example mel-spectrogram from validation set which we use to extract **MFCC** features.

III. ML Models

We will now look at the different machine learning models we have used for audio event detection and tagging task. We

have also discussed their accuracy on validation and test sets and made a comparison of their performance towards the end.

A. Linear Model - Logistic Regression

Logistic Regression is one of the simplest ML model for classification. The prediction in this model is given by $\hat{y}$= **f(y)** where :

$$y = \sum_{i=1}^{n} (w_i * x_i) \tag{1}$$

$$f(x) = 1/(1 + e^{-x}) \tag{2}$$

f(x) is the sigmoid function.We use Binary Cross Entropy for modelling the loss function. **Gradient Descent** algorithm is used for training our model. The final weight update equation is obtained as follows :

$$W = W - lr * X^{T}.(\hat{Y} - Y) \tag{3}$$

Here $\hat{Y}$ is the predicted labels matrix, **Y** is the actual label matrix , **W** is the weight matrix ,**X** is the feature matrix and lr is the learning rate.

1) Model Hyperparameters :

- lr = 0.2
- Number of gradient descent iterations = 100
- MFCC features - 20 MFCC features were calculated. Along with that, we used 20 first order differences and same number of second order differences to capture temporal data. Hence total number of features used were **60**.
- Test Data = 20%
- Validation Data = Given

2) Training procedure : We extracted total 60 features from each sample and concatenated frames of all samples along the row. Hence each row represented a frame/data and each column(60) represented a feature. Since we used pure samples for this model, labelling was not an issue. We trained two different models - One for classifying audio vs silence. Other one for classifying music vs speech. For testing we first classified each frame of the test set spectrogram as audio or silence. Then after some postprocessing(discussed later) , we extracted the audio frames and using our second model, classified them as speech or music using some aggregation methods(discussed later). The predictions were stored in a .csv file.

B. Neural Networks using Back propagation

This was the first non-linear model we used. To train the model , backpropagation algorithm was coded from scratch. We mainly used the following formulas, for training our simple neural network using backpropagation.

$$\Delta^{l}_{ij} = \Delta^{l}_{ij} + a_j^{l} \delta_i^{l+1} \tag{4}$$

$$D^{l}_{ij} = \Delta_{ij}^{l}/m \tag{5}$$

Here **l** represents the layer number, $\delta^{\mathbf{l}}$ represents the back-propagated errors of the l^{th} layer , $\mathbf{a^l}$ represents the calculated elements of layer l using current weights and Δ^l is an intermediate matrix keeping track of the weight correction. The final weight update equation is given by :

$$\Theta^{l}_{ij} = \Theta_{ij}^{l} - lr * \Theta^{l}_{ij} \tag{6}$$

Θ_{ij}^{l} stores the weight connecting element i of layer l with element j of layer l+1. Hence it represents the weight matrix of our model. lr is the learning rate. The code can be referred to for further details.

1) Model Hyperparameters:

- lr = 20. We used a high learning rate to reduce training time since the model was a complicated non linear one.
- Number of gradient descent iterations = 100
- MFCC features - 5 MFCC features were calculated. Note that to reduce total number of training parameters and training time we have used less features.
- Model Architecture :
  Layer 1 - 5 input features
  Layer 2- 6 elements . **Sigmoid** activation
  Layer 3- 6 elements . **Sigmoid** activation
  Layer 4 - Output layer with 2 elements. **Softmax** activation
- Test Data = 20%
- Validation Data = Given

2) Training procedure : The training procedure was similar to Linear model.We used total 6 features from each sample and concatenated frames of all samples along the row. Hence each row represented a frame/data and each column(6) represented a feature . Pure samples for this model, labelling was not an issue. Here too we trained two different models - One for classifying audio vs silence. Other one for classifying music vs speech. For testing we first classified each frame of the test set spectrogram as audio or silence. Since output neuron had only one element our model directly calculated the probability of a frame belonging to a particular class or not.Then after some postprocessing(discussed later) , we extracted the audio frames and using our second model, classified them as speech or music using some aggregation methods(discussed later). The predictions were stored in a .csv file.

C. Convolutional neural network

This was a non-linear model made by applying the concept of image convolution to audio. We all know that spectrogram is one feature of audio data which helps us visualize the sample and infer its properties.The spectrogram may be assumed to be a gray scale image with the value of frequency bins as gray-scale image values. Hence feeding these spectrograms into a convolutional network can give us their true labels. However here the problem at hand was different. The spectrograms we given were impure in nature. In other words they contained samples from all three classes and hence did not have one true label. So we modified the convolutional method to cater our needs.We treated each frame of given spectrogram as an image and trained our model on these image cum frames!

1) Model Hyperparameters :

- MFCC features- 20 MFCC features were calculated. Further, we used 20 first order differences and same number of second order differences to capture temporal data. Hence total number of features used were **60**.
- Model architecture :
  1st Conv Layer - 32 filters of size 3×3 with a stride of 1. Padding = **same**, Activation = **relu**. This was followed by a **Max-Pool** layer with size 2×2 and a stride of 2. Padding = **same**
  2nd Conv Layer- Same architecture as layer 1.
  3rd Conv layer- Same as Layer 1 except the size of filters used in convolutional operation was 2×2.
  This was followed by 4 conventional neural network layers of size **256, 192, 128, 64** with **relu** activation. The output layer contained **2** elements with **softmax** activation.
- We used a **dropout of 0.3** in all neural network layers to prevent **overfitting**.
- Categorical Cross Entropy loss with **Adam** optimizer
- Test Data = 20%
- Validation Data = Given

2) Training procedure: We concatenated all frames of each sample together along the rows , with MFCC features as columns.As discussed earlier, training was done considering each frame as an image. **Categorical Cross Entropy** was used as loss function. We used **Tensorflow** library to implement our model. Two different models were used for audio vs silence and music vs speech. Rest everything was similar to previous models.

D. Recurrent Neural Networks

This is an ideal model for capturing temporal information in the audio samples. We used **LSTM (Long short term memory)** layers for training the model. Further as mentioned in the beginning, impure class audio samples were used to train the model. In other words training clips consisted of data from all three classes- speech , music and silence. To get predictions from all frames of n unknown samples, we exploited the concept of **Time Distributed layers** in our architecture.

1) Model Hyperparameters :

- MFCC features- 20 MFCC features were calculated. Further, we used 20 first order differences and same number of second order differences to capture temporal data. Hence total number of features used were **60**.
- Model architecture :
  2 LSTM layers - 64 elements with **relu** activation which returns outputs from each frame of a sample
  This was followed by **2 time distributed** neural network layers of size **64,32** with **relu** activation. The output layer was a **Time Distributed** layer with 2 elements and **softmax** activation.
- We used a **dropout of 0.3** in all neural network layers to prevent **overfitting**.
- Categorical Cross Entropy loss with **Adam** optimizer
- Test Data = 20%
- Validation Data = Given

2) Training Procedure: Instead of training with frames, here we trained our model with 10 seconds audio clip containing samples from all three classes.The frames are modelled in the form of a time series and fed into the LSTM network. Since we have used time distributed layers, the predicted label is given for every frame. For prediction we feed the complete 10 second audio spectrum features of the test data into our model, which gives us a frame wise prediction of the underlying class(speech/music or audio/silence).Here too we are using two different models. one for each task.The postprocessing remains same.

E. Gaussian Mixture Models(GMM)

This is an unsupervised learning model used for classification. Roughly speaking , here the data can be modelled as a probabilistic distribution of the pair of two or more Gaussians.The probability of an element is mathematically formulated as :

$$p(x) = \sum_k p(z_k) N(X; \mu_k, \sigma_k); z_k \epsilon \{0,1\}; \sum_k z_{\mathrm{k}} = 1 \quad (7)$$

Here z_k denotes the probability of selecting a particular **Gaussian**.Using the **Expectation Maximization Algorithm**, the parameters can be estimated as :

$$\mu_{\mathrm{k'}} = \sum_n \gamma_{\mathrm{nk'}} s_{\mathrm{n}} / \sum_n \gamma_{\mathrm{nk'}} \quad (8)$$

$$\Sigma_{k'} = \sum_n \gamma_{\mathrm{nk'}} (s_{\mathrm{n}} - \mu_{\mathrm{k'}})(s_{\mathrm{n}} - \mu_{\mathrm{k'}})^{\mathrm{T}} / N_{\mathrm{k'}} \quad (9)$$

$$\pi_{\mathrm{k'}} = \sum_n \gamma_{\mathrm{nk'}} / N \quad (10)$$

Here γ is the responsibility matrix as defined in the lecture, N is the total number of samples ans s_n is the n^{th} sample.$\mu_{k'}$ and $\Sigma_{k'}$ are the parameters of the k' cluster and $\pi_{k'}$ is the probability of selecting the k' cluster.

1) Model Hyperparameters:

- MFCC features- Owing to the large number of clacula-tions involved, we used only 2 MFCC features .
- **Two Gaussian clusters** in each model
- Number of training iterations = **100**
- Test Data = Given

2) Training procedure: We collected 2 features from all frames of all samples and concatenated them along the rows. Hence frames were along the rows and features(2) along the columns. These frames were used to train two different models- speech vs music and audio vs silence. Each model was a combination of two **Gaussian** models. While testing, we predict which cluster a particular frame will belong to. Rest post-processing features will remain same as previous models.

F. K-Means Clustering

This is another unsupervised learning model we use to predict data. This is different from **GMM** in the sense, here we do hard clustering rather than assigning soft responsibilities like the former. We use an iterative approach for training model parameters using the following equations :

$$\mu_{k'} = \frac{\sum_n r_{\text{nk'}} x_n}{\sum_n r_{\text{nk'}}} \tag{11}$$

$$[r_{nk} = \begin{cases} 1, & \text{if } argmin_{k'} ||x_n - \mu_{k'}|| \\ 0, & \text{otherwise} \end{cases}] \tag{12}$$

Here $\mu_{k'}$ represents the mean of the k' cluster.

1) Model Hyperparameters:

- MFCC features- Owing to the large number of clacula-tions involved, we used only 2 MFCC features.
- **Two clusters** in each model
- Number of training iterations = **100**
- Test Data = Given

2) Training procedure: The training procedure is identical to the **GMM** model except that the underlying algorithm and mathematical equations are changed. These are given above. The predict function identifies which cluster a particular frame belongs to in each model(speech vs music and audio vs silence). The postprocessing techniques remain same.

IV. Post processing techniques

The techniques discussed here are common for all the models discussed above. Suppose we have the framewise predictions of the unknown audio sample. Since models are not 100% accurate, we might encounter some wrongly classified audio frames among silence or vice versa. This will heavily disturb the time stamps of audio data.

To solve this issue we used an approach similar to what is called **Erosion/Dilation** in the literature. However our method was original and gave good results after some hyperparameter tuning. We took a window of 16 frames($\approx 0.5 seconds$) and counted the total number of audio and silence frames. If the number of silence frames were greater than 10, two cases arise :

- Case 1 : If the previous window was classified as silence, change the end time of current silence period to time of last frame of this window. Jump to the next window with hop size 1.
- Case 2 : If the previous window was classified as audio, find the first silence frame of the current window and set its time to mark the end of audio and beginning of silence.Set the time of last frame to mark the end of new silence period. The next window is 16 elements from the first silence frame found out.

Similar procedure can be repeated if number of silence frames are found to be less than 10. Here too , two cases arise which can be handled just like mentioned above. However remember replacing silence with audio. The value of 10 frames is an hyperparameter. We observed that our models classified silence frames correctly in most cases. Hence we chose to label a window as silence only if they had a **2/3** rd majority .

This procedure gives us the start and end times of all audio events in the clip. Using our second model we identify them as speech or music and publish the results.

V. Evaluation Metrics

We evaluated our models using confusion matrix on audio tagging task. For finding accuracy of our predicted time stamps, we stored the results in a .csv file and used the script provided to us , which gave us the model's performance. The evaluation results are discussed below in terms of model complexity and performance.

VI. Model comparisons

Model Name	Test data performance	Validation Data Performance	Time to Train
Linear Model	75%	Moderate	30 min
Neural Network(BP)	83%	Moderate	45 min
CNN	97%	Very Good	40 min
RNN	72%	Poor	25 min
GMM	81%	Moderate	35 min
K-Means	73%	Moderate	30 min

- Test Data performance indicates how well the model performs on data separated from the training sets(model accuracy using **Confusion Matrix**). Note that this accuracy is only on frame classification.
- Validation performance is on data given to us as validation set. In indicates how accurately time labels are predicted, according to following legend:
  Poor - error of +- 5 sec.
  Moderate - error of +- 1 sec.
  Very good - error of +- 3 sec.

Conclusion : Hence we conclude that the **Convolutional Neural Network** model performs best.

References

[1] A. Temko, C. Nadeu, and J. Biel, "Acoustic event detection: SVM-based system and evaluation setup in CLEAR'07," in Springer-Verlag, Berlin, 2008.

[2] E. A. P. H. Soumitro Chakrabarty, "Broadband doa estimation using convolutional neural networks trained with noise signals," in arXiv:1705.00919, 2017.

[3] G. Parascandolo, H. Huttunen, and T. Virtanen, "Recurrent neural networks for polyphonic sound event detection in real life recordings," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016.

[4] E. C¸ akır, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen, "Convolutional recurrent neural networks for polyphonic sound event detection," in IEEE/ACM TASLP, volume 25, issue 6, 2017.

[5] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[6] A. Mesaros, T. Heittola, and T. Virtanen, "Metrics for polyphonic sound event detection," in Applied Sciences, 2016