**Methods for extracting data and modelling**

**Extracting target emotion labels**

The emotion labels of those sentence audios are stored in different txt files, for easier processing in later steps, and to avoid repeated work, we extracted those labels from text using regular expression and stored the file names of all sentence audios together with their emotion labels in a data-frame, which was then written to a csv file.

**Extracting input audio data**

Since all sentence audios are in wav format, we had to transform them into data structure that allows easier feature extraction and data analysis. To achieve this, we used a python package named ‘librosa’, which provides function for retrieving audio information (by loading audio files as floating-point time series), as well as a wide range of building blocks necessary to pre-process audio, extract various features and visualize those audio features.

In our dataset, a speech was sampled at a rate of 16,000 audio data points per second, and we used this native sampling rate when loading each audio file. After that, because some audio can be quite long, we firstly trimmed any leading or trailing silence by setting the threshold as 20 decibels (dB). Then, we performed noise reduction to the trimmed waveform, so that there won’t be noises degrading the performance of our speech emotion recognition models.

After these pre-processing procedures, the waveform can actually be used directly as input to our MLP and CNN models as long as any shorter ones are padded to the same length. However, we found that the size of data would be so large, which makes the training of NNs too long while did not provide extra benefit on prediction performance. So, we decided to use extracted features as input instead of the waveform. We tried two types of features as well as the combination of them, which are Mel spectrogram and Mel-frequency cepstral coefficients (MFCC) (with the number of MFCCs to return set to be 40 to get more components). For a given waveform, both of Mel spectrogram and MFCCs are originally extracted as matrices, we took means of those 2-d arrays to transform them into 1-d arrays, which is a standard treatment in audio analysis. To avoid padding, when taking means, we firstly transposed those arrays, which did not reduce performance of our models while providing a fixed size of input (128 for Mel spectrogram, 40 for MFCC, and 168 for combined; for 1-d CNN, they’ll then be expanded into 2-d input with size of 1 X 40or128or168, respectively). This trick also allowed our models to have less weights to learn, thus reduced running time requirement.

**Experimental Setup**

Before modelling, since there’re some categories of emotions not suitable for the speech emotion recognition problems we’re trying to solve. We needed to filter out or combine some of them before splitting our data into training, validation, and testing.

For binary classification, we combined happy and neutral into calm, while combined the other emotions including angry, sad, frustration and fear into uncalm.

For Multi-class classification, we kept only angry, sad, neutral, happy and excited emotions, and combined happy and excited, so four categories in total.

**Modelling**

We explored two types of Neural Networks.

For baseline model, we used the MLPClassifier in Scikit-learn package, which provides convenient interface for implementing a wide variety of fully connected feed-forward MLP models with different architectures and parameters. Here’s a summary of what we experimented with.

|  |  |
| --- | --- |
| Number of hidden layers | 1 to 4 |
| Sizes of hidden layer | 10, 20, 30, 50, 100 (combinations of them) |
| Hidden layer activation function | logistic sigmoid function,  hyperbolic tan function (tanh),  and rectified linear unit function (ReLU) |
| Solver for weight optimization | stochastic gradient descent (SGD),  Adam,  and lbfgs (a quasi-Newton method) |
| Alpha (for L2 regularization term in the log-loss function) | 0.01 to 0.3 |
| Learning rate | 0.001 or adaptive (only for SGD) |
| Minibatch size for stochastic optimizers | 16, 64 |
| Epsilon (for numerical stability in adam) | 0.0001 to 0.1 |

A more advanced model we tried was 1-d CNN, which is mostly used in time-series data while also used on audio or other signal data. Unlike MLP, which views each input unit independently and does not capture correlation in input, 1-d CNN is able to extract extra feature from the input array by utilizing kernel that slides along one dimension.

In the implementation, we built the 1-d CNN from scratch using Pytorch. Here’s a summary of structure and summary of parameters of our final model.

|  |  |
| --- | --- |
| Input | Mel spectrogram + MFCC features |
| Activation function for inner layers | ReLU |
| Optimizer | SGD |
| Momentum | 0.9 |
| Learning rate | 0.0001 |
| Weight decay | 0.00001 |
| Minibatch size | 4 |
| Loss function | Cross Entropy Loss |

Structure: 需要将这个变成图

|  |  |
| --- | --- |
| Input layer | Number of channels: 1  Input size: 1 X 168 |
| Convolution layer 1 | Number of channels: 256  Kernel size: 5  Padding: 2  Stride: 1 |
| Max Pooling layer (for subsampling) | Kernel size: 2 |
| Convolution layer 2 | Number of channels: 128  Kernel size: 5  Padding: 2  Stride: 1 |
| Dropout | Dropout probability 0.1 |
| Max Pooling layer | Same as above |
| Convolution layer 3 | Number of channels: 128  Kernel size: 5  Padding: 2  Stride: 1 |
| Max Pooling layer | Same as above |
| Convolution layer 4 | Number of channels: 128  Kernel size: 5  Padding: 2  Stride: 1 |
| Convolution layer 5 | Number of channels: 128  Kernel size: 5  Padding: 2  Stride: 1 |
| Convolution layer 6 | Number of channels: 128  Kernel size: 5  Padding: 2  Stride: 1 |
| Flatten |  |
| Fully connected layer | Input size: 2688  Output size: 4 |
| Output | Softmax |