Spoken Digit Classification Report

Jincy Baby

Introduction

The dataset, we are dealing with, consist of recordings of 6 different speakers who pronounce each digit from 0 to 9 for 50 times. The .wav files are saved at 8 kHz and with a name of the form {digit}_{speakername} _{attemptnumber}.wav. Two folders namely Training and Testing each includes the files with MFCCs and labels of the training and testing data respectively. The training and validation data are split in the ration 80:20.

Description

Audio classification is the process of analyzing audio recordings and has various applications in AI ranging from music-genre identification to virtual assistants. Mel- Frequency Cepstral Coefficients or MFCC is one of the main audio features that may be considered in Machine Learning processes involving audio files. MFCC are the coefficients that make up Mel-frequency cepstrum and it describes the overall shape of a spectral envelope.

We use MFCC as a feature to classify the data using different models and obtain optimal weights for classification of validation data. In the first part we start with visualisation of data and the features. We then proceeds to the preprocessing stage where we standardise the data inputs and clean the labels for further use. KFold cross validation is used to find the regularisation parameter and optimal weights.

Analysis

Visualisation of data samples

The library pandas is imported and used to print out the MFCC values and the label so that we can view what is included in the imported file. There are a total of 900 coefficients for each file and its paired with the label.

5	6	7	8	9	...	891	892	893	894	895	896	897	898	899	label
7	-519.283020	-523.684753	-520.529175	-478.866180	...	6.576453	9.443312	6.769020	3.878198	1.093867	1.110493	-0.417849	-4.568942	-6.523507	0
3	-353.766663	-350.711029	-361.763733	-375.541840	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
6	-356.141510	-352.765839	-349.870911	-362.951904	...	2.335657	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
8	-332.169006	-334.234650	-338.372986	-337.583008	...	-8.136976	-4.376896	-1.517604	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
5	-335.528809	-332.170227	-333.991577	-353.383728	...	-7.899344	-2.855740	-1.100652	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0

Figure 1

The mean and variance of the MFCCs are computed as shown in figure 2. Here, the variance is quite high and it means that the points are vastly spread out from the mean, and from one another. It is possible to scale the MFCC such that the mean remains close to 0 and for

smaller variance as thus the random variations in model remain less. Thus, we standardise the MFCC value in the preprocessing stage.

```
: print (audio_mnist_training_mfccs.mean(axis=1))
  print (audio_mnist_training_mfccs.var(axis=1))

[-13.88185235  -8.26082565  -8.87601785 ...  -6.70594501  -6.00281653
  -6.08288715]
[7343.17252373 5031.53311258 5531.37301877 ... 8798.3727984  7528.74341
066
 7200.93304515]
```

Figure 2

The librosa library is used to the amplitude envelope of a waveform. Sampling rate of audio time series is computed to plot the Figure 3.

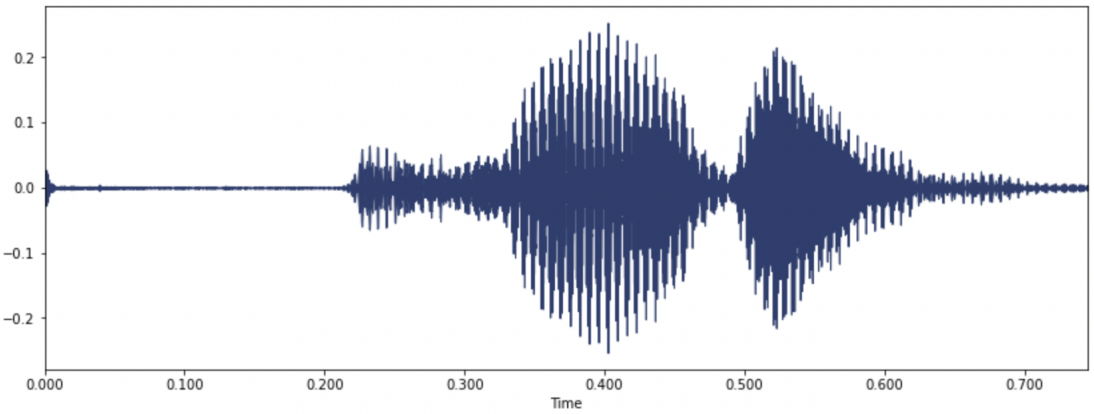

Figure 3

The MFCCs of training dataset is imported and the spectral diagram is stored in the folders 0 to 9 so that the file names starting with 0 is stored in the folder 0, 1 is stored in folder 1 and so on. This can be used as the training data for CNN classification model if need to be. However, we are not using the model in this project and simply use it to randomly display the spectral diagrams for the MFCCs representing each of the digit recordings. The code for saving the image is commented out as the images are already saved in the img_data folder after the first execution.

Preprocessing the data samples

The data inputs, MFCCs, are standardised and stored in aud_data_inputs. The labels are resized and saved as int data types so that we can use it further in the program to fit the models. Evaluating the mean and variance of the standardised MFCCs shows a low value of mean and variance.

```
print (aud_data_inputs.mean(axis=1))
print (aud_data_inputs.var(axis=1))
```

```
[-1.32145813 -0.42012707 -0.48319505 ...  0.08510476  0.06004648
  0.03982926]
[7.17679823 0.71614593 0.88559615 ... 0.42549882 0.41092291 0.40575815]
```

Figure 4

Evaluation of optimal weights

We have calculated optimal weights with multinomial lasso logistic regression and then with multinomial logistic regression. KFold Cross Validation using ridge regression was done to get optimal values for regularisation parameter and weights. Then, training data was split into training and validation data to fit the model and find the classification accuracy of validation data with the obtained optimal weight.

Evaluation of optimal weights using multinomial lasso logistic regression

In the multinomial lasso logistic regression, we define the functions soft_thresholding, lasso_logistic_regression_cost_function and proximal_gradient_descent. soft_thresholding takes the two arguments argument, threshold and performs the soft-thresholding operation to each component of the array and is defined as:

$$soft_\tau(x) = \begin{cases} x - sgn(x).\tau, & |x| \geq \tau \\ 0, & |x| < \tau \end{cases}$$

Whereas, lasso_logistic_regression_cost_function inputs data matrix, weight matrix, and the one hot vector encoding of labels and outputs MSE+$\alpha||w||_1$, where MSE stands for Mean-squared-Error(multinomial logistic regression loss), α represents the regularisation parameter and w, the weights. The step size is calculated using the formula $\tau = \frac{3.9s}{||\Phi(X)||^2}$.

While using multinomial lasso logistic regression, we got a classification accuracy rate of 98.917% on the training data. The number of iteration was set as 6000 and the objective obtained at 6000th iteration was approximately 0.12398. The table below shows the classification accuracy corresponding to the number of iterations.

Iteration	objective	accuracy
100	1.3327717526421339	82.33333333333334
500	0.5400010343897699	94.08333333333333
1000	0.3474078428762476	96.04166666666667
2000	0.22870354265591536	97.25
3000	0.18116169242066182	98.0
4000	0.15427354156840994	98.41666666666666
5000	0.13661032496614336	98.625
6000	0.12398349417089416	98.91666666666666

The accuracy of the model increases as the number of iteration increases. But we need to make sure to stop the number of iterations such that we can avoid overfitting the data. We can test it by checking the optimal weight with the validation data that is done later in this project.

K-Fold Cross Validation using ridge regression

In this part, we define the functions KFold_split, KFold_cross_validation, ridge_regression, ridge_regression_data, prediction_function and prediction_error. ridge_regression_data computes the ridge regression data matrix and ridge_regression returns the solution $\hat{W}$ of the normal equation $\Phi^T(X)\Phi(X) + \alpha I)\hat{W} = \Phi^T(X)Y$ where $\Phi(X)$ is the mathematical representation of data matrix, Y is that of the data outputs and α is the regularisation, while $\hat{W}$ is stands for coefficients of the ridge regression.

KFold_split takes two arguments data_size, K and outputs a random split of integer indexes [0, 1, . . . , data_size] into K almost equal chunks. KFold_cross_validation takes 5 arguments data_matrix, data_outputs, K, model_evaluation, error_evaluation. where K is positive integer number representing the number of chunks used, model_evaluation is the lambda function that takes data_matrix and data_outputs, and evaluates optimal weights for the model.The function error_evaluation is a lambda-function that takes data_matrix, data_outputs, and weights to evaluate a validation error.

```
An optimal value of regularisation parameter is 19.0.
For this value of     regularisation parameter one gets optimal weights of the form
[ 4.47165174e-02 -7.26559746e-02  1.21592485e-01  5.91211874e-02
  2.26439122e-02 -8.87588877e-02 -5.96668789e-02  1.26478226e-02
  1.13316137e-01  2.75470127e-03  2.05929621e-02 -3.56379644e-02
 -2.51665717e-01 -1.70250361e-02  6.70780660e-02  2.31042395e-02
 -2.84133184e-01 -1.33422973e-01  2.96441632e-01 -7.26680457e-02
 -1.17866785e-01 -1.05292632e-01  8.44672727e-02  1.29354971e-01
  1.11036822e-01  8.43700863e-02  2.41050926e-01  1.03092548e-01
  5.04914445e-02 -3.25655602e-01 -1.00050500e-01 -6.63243572e-04
  3.07234953e-01  2.75714747e-01  1.58702539e-01 -1.58832678e-02
 -1.13987940e-01  4.85685974e-02 -7.99362051e-02 -5.86836897e-02
  8.38885774e-02 -1.70755111e-01 -4.72012717e-01 -7.17642543e-02
  2.57816525e-01 -9.56857032e-02  2.89473676e-01  1.39196770e-01
  1.62226351e-01  1.60319606e-01 -1.34458235e-01  8.25600506e-03
  7.32313161e-02  1.11301203e-01  2.04358418e-01 -4.40692820e-02
  2.29758466e-01 -4.49357464e-02 -6.42514239e-02  9.25710509e-02
 -1.18067139e-01 -4.21025017e-01  1.37016316e-01  2.78515767e-01
  1.15603896e-02  4.59290887e-02 -1.11412392e-02 -7.00218194e-02
```

Figure 5

Here, We use Kfold Cross validation using the ridge regression algorithm and implement the grid search, which search for a minimum value of the function on a given grid points, to find the hyperparameter along with the optimal weights. Analysing the optimal weights, it stays around 10^{-1} to 10^{-2} values mostly.

Evaluation of optimal weights using multinomial lasso logistic regression and testing with testing data

Multinomial logistic regression is implemented to find the optimal weight and we import the testing data to check the classification accuracy of the model. The accuracy of both are tabulated and given in the below table.

Iterations	accuracy for training data	accuracy for validation data
100	89.9167	88.0
500	96.625	93.5
1000	97.667	93.667
2000	98.5833	94.33
4000	99.29167	95.0
5000	99.375	95.33
6000	99.5	95.33
7000	99.67	95.17

As we can see in the above table, the accuracy rate of the training data seems to increase in 7000th iteration when compared to 6000th. However, the accuracy of the validation data seems to decrease in the last one compared to the 6000th iteration. Hence, we set the number of iterations as 6000 and the corresponding optimal weight is taken as the best weight matrix for the model.

Splitting training data to fit the model

As the optimal weight obtained from the multinomial logistic regression is utilised to set the best weight matrix, the project further checks the accuracy of multinomial logistic regression by splitting the training data again into training and testing data. data_split funcion is introduced to split the data randomly. The function randomly choose a row index considering the validation ratio and split the data into two. With number of iterations set as 2000, the classification accuracy of the split training data stayed around 98% and that of the test data stayed around 97%.

```
Iteration completed after 2000/2000, objective = 0.11221520741639854.

[294]: X_audio_recovered_labels = multinomial_prediction_function(audio_split_d
       audio_classification_accuracy = classification_accuracy(Y_train, X_audio
       print("The classification accuracy for the training dataset is {p} %.".f

       The classification accuracy for the training dataset is 98.697916666666
       66 %.

[295]: test_data_inputs, test_row_of_means, test_row_of_stds = standardise(X_te
       test_data_matrix = linear_regression_data(test_data_inputs)
       test_accuracy_rate = classification_accuracy(multinomial_prediction_func
                                                    (test_data_matrix, X_train_optimal_weigh
       print("The classification accuracy for the training dataset is {p} %.".f

       The classification accuracy for the training dataset is 97.083333333333
       33 %.
```

Figure 6

Conclusion

The project considers MFCC as the main feature to develop a classification model. Standardising the data inputs is of atmost importance as high variance rate in MFCC may result in high random variations in the model. To avoid overfitting check the model with the validation data and Kfold cross validation can be done to check if the obtained optimal weight is apt to be considered.

There are several other features that can be considered including Spectral centroid, Spectral Rolloff, Spectral Bandwidth, Zero-Crossing Rate, Chroma feature etc. Spectral Centroid determines the frequency at which the energy of a spectrum is centered upon. Spectral Rolloff measure the shape of the signal. Spectral Bandwidth is the width of the band of light at one-half the peak maximum. Zero-Crossing Rate measure the smoothness of a signal. Chroma feature indicates how much energy of each pitch class, is present in the signal.If we are dealing with spectral images, CNN may be used to develop a model as CNN works best with images.

References

[1] *Rupali G Shintri, Analysis of MFCC & Multitaper MFCC Feature Extraction for Speaker Verification*, International Journal of Computer Science and Information Technologies, 2015

[2] *Michael Bowles, Machine Learning in Python*, Wiley, 2015

[3] *Jonathan Adams, Python Machine Learning from Scratch* , AI Sciences LLC, 2016

[4] *https://towardsdatascience.com/how-i-understood-what-features-to-consider-while-training-audio-files-eedfb6e9002b*

[5] *http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/*