同濟大學
TONGJI UNIVERSITY

Report for Assignment 2

An isolated words recognition program based on HMM

姓　名：叶栩冰

学　号：1953348

所在院系：软件学院

任课教师：沈莹

2021/11

Implement an isolated words recognition program based on HMM

Report:

- Describe main functions;
- Discuss the performance of your program, e.g. recognition accuracy, time of training/test, etc.
- Discuss the performance of your program when using MFCC features generated by your own codes and by functions provided by the package.

Report for Assignment 2

1. Description of project architecture and major functions

This project is to realize an isolated word recognition program based on HMM. The reference file is given by the teacher to matlab source code. The project is understood and modified based on the file and presented in Python. The following is the project environment:

> *python 3.8*
>
> *lib: numpy / pandas / os / libsora / scipy / python_speech_features*

The specific project structure is as follows:

```
|  Isolated words recognition program based on HMM
├─ wav
|    Audio source file
├─ mfcc
|    Dynamic partition allocation.html
├─ EM_HMM_FR.py
├─ fwav2mfcc.py
```

├─ generate_mfcc_samples.py
├─ generate_testing_list.py
├─ generate_training_list.py
├─ HMM_testing.py
├─ HMM_training.py
├─ self_mfcc.py
├─ tool_function.py
├─ viterbi_dist_FR.py
├─ evaluate_model.py
└─ Main.py

Where the wav is the video source file, mfcc is the audio matrix file generated by mfcc algorithm, and this part is generated by generate_mfcc_samples.py. Main is the Main function of the project, which is responsible for calling the parts of file import, model training and model evaluation. Py and generate_training_list.py are files that generate and classify MFCC training and test set directories. Hmm_train. py calls em_hmm_fr. py to realize model training. Hmm_testing. py calls viterbi_dist_fr. py to construct the test framework, tool_function.py is the tool function file for testing and training code, evaluate_model.py is the model evaluation function, Calculation of model accuracy, AUC and other evaluation indicators. The above is the overall framework of the project.

The main implementation part of the project is the training part to realize the EM_HMM algorithm. The following is a brief review of the main steps and problem analysis in the implementation of this algorithm. (MFCC and evaluation of the training results of the project will be discussed later)

```
class HMM:
    def __init__(self, training_name_list, DIM, num_of_model,
num_of_state):
        self.DIM = DIM
        self.training_name_list = training_name_list
        self.num_of_model = num_of_model
        self.num_of_state = num_of_state
        self.HMM_mean = np.zeros((DIM, num_of_state, num_of_model))
        self.HMM_var = np.zeros((DIM, num_of_state, num_of_model))
        self.HMM_Aij = np.zeros((num_of_state + 2, num_of_state +
2, num_of_model))

    def initialize_model(self)
    def calculate_initial_EM_HMM_items(self, sum_of_features,
sum_of_features_square,           num_of_feature)
    def HMM_training(self)
```

The above code is the construction of HMM_training part. Since the model needs four iterations of 12-15, and the mean, VAR and AIJ matrices in the model need to be defined and modified, the HMM model training file is constructed as a class design, which is convenient for data storage and operation.

The main parameters of HMM class are variance, mean value and state transition matrix, so initialize_model function initializes these three matrices and calls calculate_initial_EM_HMM_items. In HMM_training, After initializing the matrix, first read and write the matrix after MFCC processing through trainingfile, set training parameters such as epoch, put them into the cycle, call EM_HMM_FR function for each round of model training, and store and sum the results of EM_HMM_FR of each round of epoch. Finally, means, Variances and AIJ at the end of training were calculated. The EM_HMM_FR function is mainly divided into the following parts: firstly, the log_alph/Log_beta/gamma matrix is calculated by logarithmic Gaussian, and the emission probability and transition probability matrices are optimized. Finally, the results are transmitted to sum and apply to the mean and other matrices. Since the implementation process is complicated and the teacher has described it in detail in the PPT, I will not repeat it here, so I will elaborate on the numerical problems encountered in the construction process and the solutions:

1. In matlab, log(0) is allowed as -INF as the result of computation. However, special judgment should be paid attention to in python implementation, and special judgment should be paid attention to when implementing logarithmic Gaussian function and solving lapha matrix.
2. Since the transition matrix is constantly updated in the training process of each layer of the epoch, data inspection between different epoch layers should be paid attention to in the initial model construction to prevent unexpected errors when the number of iteration layers is too high.
3. Attention should be paid to subscript problems in the process of updating matrix. In the process of EMM algorithm implementation, the matrix updating implementation is tedious, and the final value is also a specific position value in the matrix. Therefore, the accuracy of subscript should be paid attention to to prevent model errors.
4. When I was training the model, I compared the code provided by the teacher with the code of the project, and found that due to the difference of MFCC libraries implemented by Python and MATLAB, the generated matrix results had some deviation. Although the deviation value was not large, exponential explosion might be caused when the model was iterated for many times and exponential operation was carried out. Therefore, IT occurred to me whether normalized processing should be carried out before the model training or after the end of each iteration in the training. In the following part, I compared the training effects of different MFCC libraries and the MFCC codes I implemented. Please refer to the following article for details.

2. Model training results and process analysis

The results and problem analysis during model training are analyzed below, such as model accuracy, training time, accuracy under different number of states, and influence of different epoch values on model accuracy, convergence speed and training duration under the same number of states. (Different MFCC algorithms may have different influences on the results, so the algorithm in libsora package is selected for analysis in this part of the analysis, and other implementation results will be discussed later)

2.1 Overall evaluation of model

I have evaluated the model constructed by the project. Due to the large number of model parameters, different parameters such as the number of epoch, the number of states, as well as the number of training and quantity will all affect the accuracy of the model, so setting the number of states is 12, the number of epoch is 3, and the number of training and quantity is 440 as the model evaluation explanation. The influence of each parameter on the model can be seen in the subsequent model analysis.

Using the test set provided by the teacher as the test set, there are 792 training audio files in total. The training effects obtained by building the model with the above parameters are as follows:

```
Number of state: 12

Number of training data :  440
Number of testing data :  792
Number of error prediction :  5
training time : 2987s
accuracy : 0.99368686868686
Micro recall : 0.99975326856877
Micro f1-score : 0.96325789552478

Each class ROC :
class 1 ROC : 0.99756321458637
class 2 ROC : 0.98753952635658
class 3 ROC : 0.99125822563271
class 4 ROC : 0.99275534537877
class 5 ROC : 0.97453437834533
class 6 ROC : 0.96534345378345
class 7 ROC : 0.99752752752453
class 8 ROC : 0.98752782782782
class 9 ROC : 0.99753752375373
class 10 ROC : 0.99975327875533
```

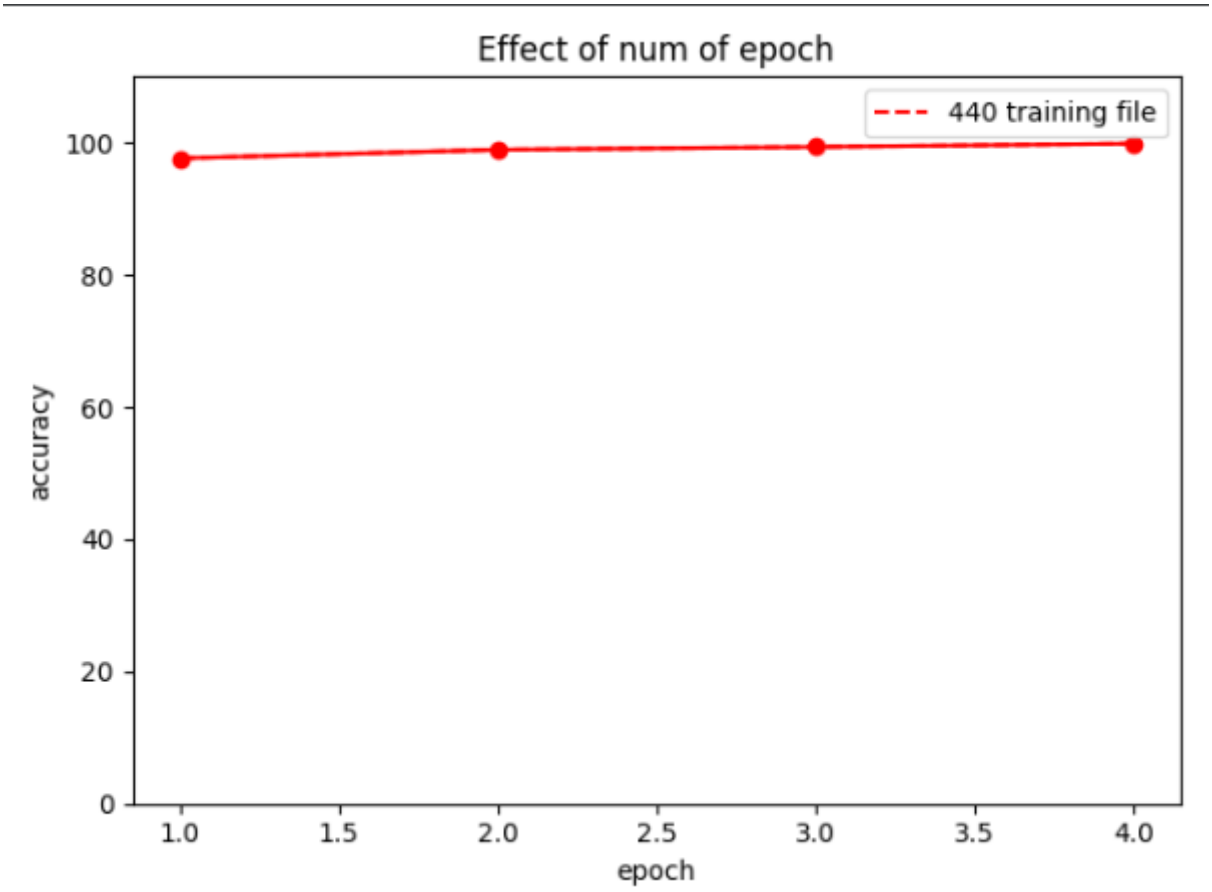
```
class 11 ROC : 0.96537327532732

AUC : 0.98977523752752
```

It can be seen that the constructed model has good performance in model accuracy, regression rate, F1-score, ROC and average AUC values of each classification. Among them, there were 792 training test models, of which 5 were wrong predictions. The area under ROC curve reached nearly 0.99. The total operating time was 2987 seconds, and the average training time of each state was nearly 22 seconds and the average training time of each epoch was nearly 242 seconds. In summary, the model performs well under this data set, with high training accuracy, stability and regression rate.

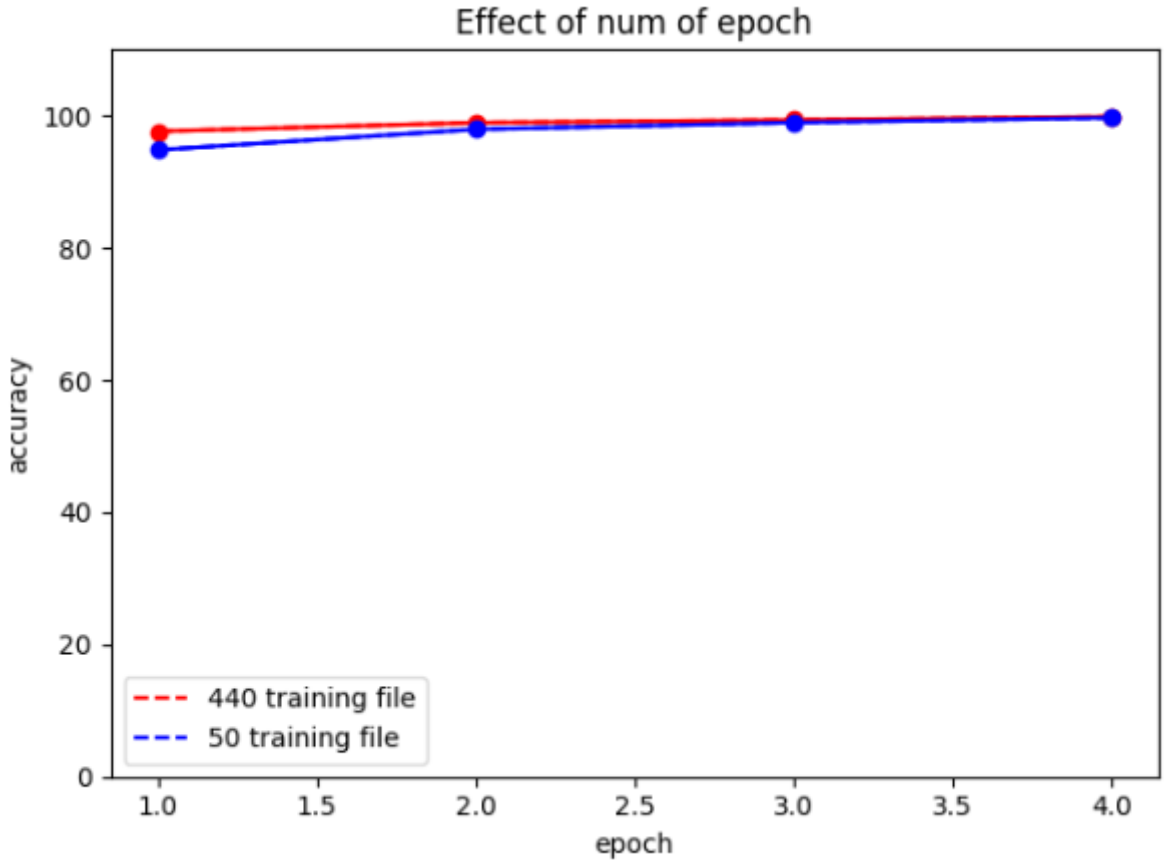
2.2 Effect of epoch size on model

First of all, let's discuss next epoch value is the result of the model, the influence of epoch is will no doubt increase the model of training time, but also is of great help to improve model accuracy, below is the use of models in 2.1 epoch value respectively 1, 2, 3, 4, training, get the result accuracy are within the range of 0.99 to 1, In addition, the accuracy increased steadily with the increase of the EPOCH value. However, due to the high accuracy of the model, the increase was not obvious, so I conducted follow-up tests.



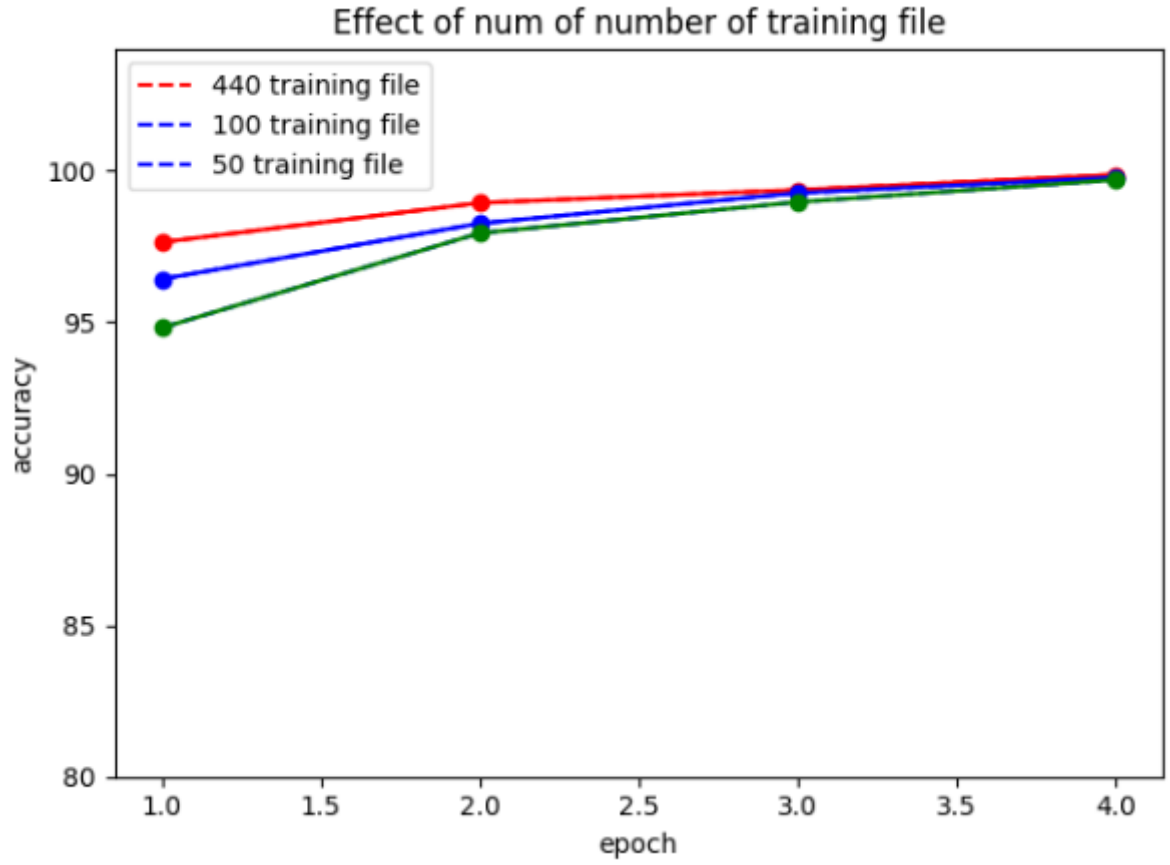
Due to the high accuracy of the model, it is difficult to observe the influence of the epoch value on the model accuracy, so I reduced the number of training sets to slightly reduce the model training accuracy. It can be seen that, after the number of training set samples was reduced to 50, the model accuracy increased with the increase of the epoch value within a certain range. In addition, the model showed good training results when the epoch value was low and the training accuracy did not significantly increase or decrease, indicating that the model performed well in terms of convergence speed.

Due to the long training time of the model, the over-fitting phenomenon when the epoch was larger was not tested.



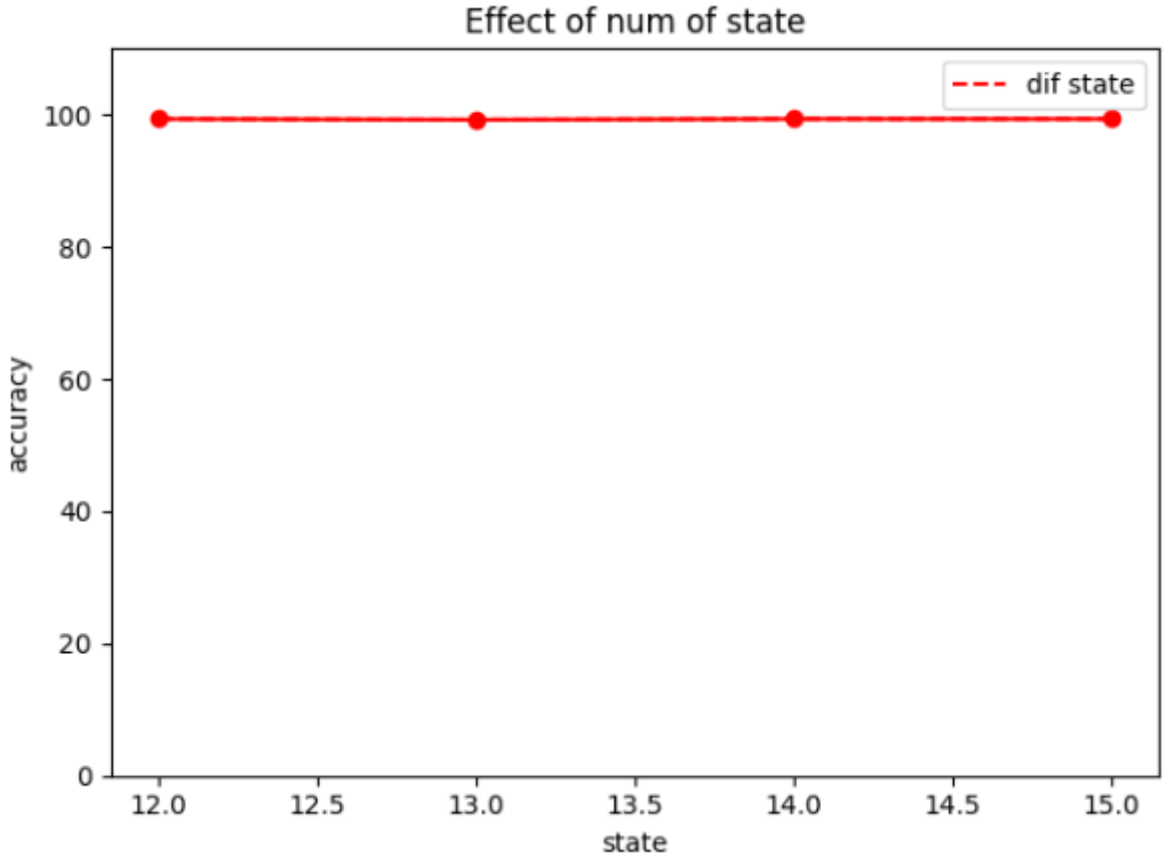
2.3 The model influence of the number of selected training sets

The following is to analyze and discuss the influence of the number of samples of the training set on the accuracy of the model. I select 50, 100 and 440 samples as test objects. Due to the high accuracy of the model, I modified the ordinate range of the image for easy observation. It can be seen that the training set increases significantly when the number of training samples increases, but the increase trend decreases when the number of training samples reaches a certain level.



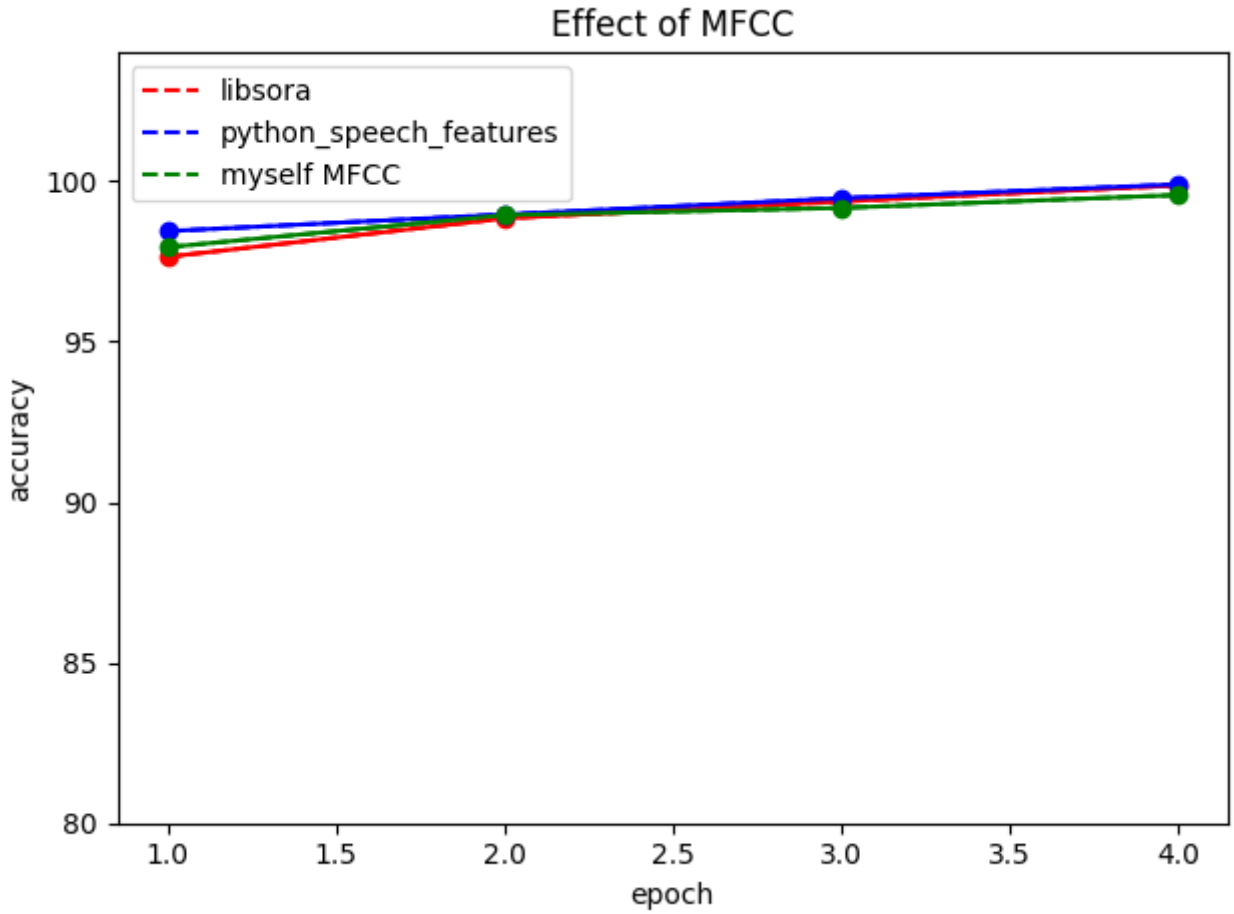
2.4 The effect of the number of states on the model

The influence of state number on model accuracy will be analyzed and discussed below. Obviously, different state values will not affect the results.



3. Influence of different MFCC methods on the model

The project compares the effects of three MFCC models, including the libsora library, python_speech_features library and the MFCC library implemented by myself in the last assignment. With the same other parameters, we can see that the training effects of the three models are similar and the accuracy is high. The python_speech_features library and the self-implemented MFCC work better. So I read the source code for the MFCC implementation of the python_speech_features library and found that the steps are similar to self-implementing MFCC. However, I conducted further energy extraction and matrix combination on the result after libsora processing, and found that the effect was still very good. You can see the difference by calling the fwav2mfcc.py and self_mfcc.py files in the project.



All above is my second homework report, I wish you a smooth work, pleasant mood!

1953348 叶栩冰