Project 3

(2015 Fall)

Course name：J1799d　Instructor：Ming Li　TA：Wenbo Liu

ID number	15213796 15213782	Name	Luting Wang Dajian Li
Tel	+86 15622777988 +86 13570300549	Email	364706267@qq.com dajianl@andrew.cmu.edu
Starting date	**2015.10.07**	Finished date	**2015.10.14**

1、 Project requirement

In this project, we are required to record digits many times and build DTW and HMM-based isolated word recognition systems. Each recording must be isolated, with no more than half a second pf preceding and trailing silence. As you can see in Fig1.

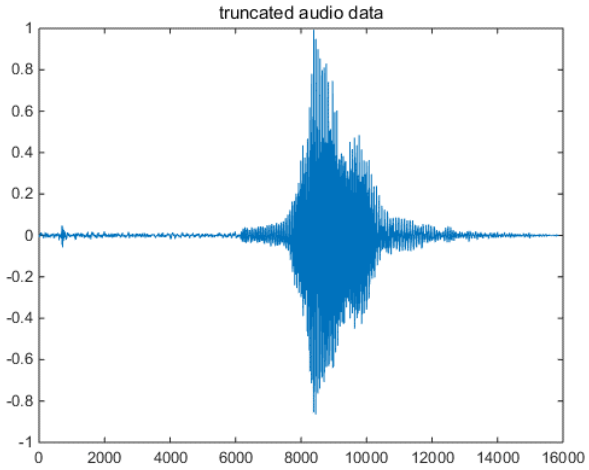

Figure 1

For DTW-based recognition, we first record one instance each of digits zero, one, two, three, four, five, six, seven, eight, nine as templates, and obtain their feature vector sequences. Then we record further five instances each of the same digits (isolated words) as test data. By applying time-synchronous DTW, we compare their feature vectors with the template and recognize them. This can been done with relative pruning or not, and we can plot pruning threshold as a function of recognition accuracy. At last, we collect five more recordings of each of the digits and redo the

above steps with pruning and without pruning, and plot the accuracy with different template sizes.

For HMM-based recognition, we use the segmental K-means procedure to train an HMM for each of the digits. First, we assume each state to have a single Gaussian distribution, and each digit to have 5 states. Then we assume each state to have four Gaussian distributions and 5 states. We train the HMM parameters and recognize the 5 test utterances and get their accuracy.

2、 Design your program

In DTW-based recognition, we just modify the Levenstein distance code (assignment 2) to perform DTW. There are three main differences: one is there is no edge cost, but all the node cost. And node costs measure the dissimilarity between the test input and template frame. The other is that the frame content is a multi-dimensional feature vector, and to compute their dissimilarity, the Euclidean distance is applied. And the last one is that the transition from one node to another only has three possible paths: one horizontal transition and two diagonal transitions. Since every input frame must be matched to some template frame, vertical transition is not allowed. And the frame content is the MFCC cepstra, which can be obtained from the code we write for assignment 1. Since the effect of speaker variations and noise can be considered for each recordings, we should normalize the cepstra to make all the recording to have 0 mean and variance of 1.0. And the mfcc feature should also contain the acceleration and the derivative of velocity, so we compute the delta and double delta value for 39 cepstral values, as shown in Fig2.

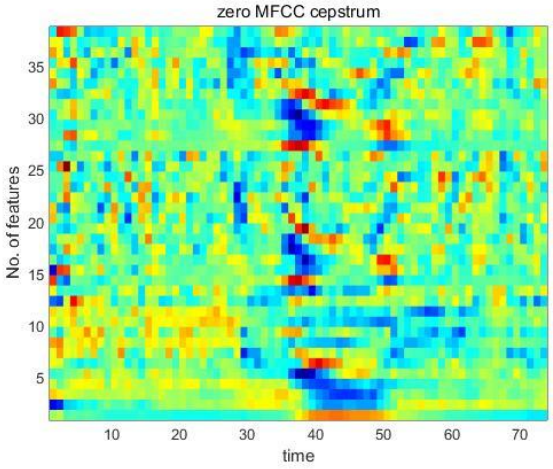

Figure 2

In HMM-based recognition with single Gaussian, we used segmental K-means to train it. We first hard cut the sequences into 5 states. And initialize all the parameters, including means and covariance, transition cost and entry cost. Then we adjust all the training templates' states, and get

new parameters. Then we reestimate parameters from these segmented training sequences until convergence. At last, we get final template of each digit with its mean and convergence, and each's state's transition score and entry score.

In HMM-based recognition with 4 Gaussians, we also used segmental K-means to train it. First, we randomly choose the 4 training vectors as means, and set the four covariance as 1, and four weight as 1/4. After we apply GMM to train the parameters, it won't work. Then we use K-means to initial the parameters, and after applying GMM, we succeed to get the final template, with each state of 4 Gaussian distribution and transition cost and entry cost.

3、 Program implementation and testing

In DTW-based recognition, we perform it with pruning and without pruning. First we record one instance each of digits zero, one, two, three, four, five, six, seven, eight, nine in lab as templates, and then recognize further five instances each of the same digit using time-synchronous DTW without pruning. The recognition accuracy is almost 100%. Then we redo the steps using relative pruning. We try different threshold and compute its recognition accuracy. For each threshold, we try 10 different utterance of the digits.

T	1	5	10	30	50	80
accuracy	10%	50%	70%	90%	90%	100%

Since when T is 10, the accuracy is 90%, and if we further increase T, the increase of accuracy is not obvious. So we pick 30 as our threshold. And we further record another set of four recordings of each of the digits as our template. And change the number of templates of words from 1 to 5 to calculate recognition accuracy. We found the as the number of template increase, the accuracy increase slightly.

In HMM-based recognition with single Gaussian and multiple Gaussian, we implement two class: SegmentalKmeans and SegmentalKmeansGMM. The headers are as follows:

```
#ifndef SegmentalKmeans_hpp
#define SegmentalKmeans_hpp
#ifndef MFCC_Feature
#define MFCC_Feature std::vector<float>
```

```
#define MEBS 1e-100
#define PI 3.14
#endif
#include <vector>
#include <stdio.h>
#include <math.h>
struct HMMModel{
	std::vector<float> * entry_cost;
	std::vector<std::vector<float>> * transition_cost;
	std::vector<MFCC_Feature> * miu;
	std::vector<std::vector<float>> * cov;
};
class SegmentalKmeans{
private:
	int K; // K states
	int FEATURE_LENGTH; // # of MFCC features
	int MAX_ITERATIONS; // maximum # of iteration
	float MIN_DELTA; // convergence threshold

	std::vector<float> * entry_cost; // entry costs
	std::vector<std::vector<float>> * transition_cost; // transition costs
	std::vector<MFCC_Feature> * miu; // centers: K by FEATURE_LENGTH Matrix
	std::vector<std::vector<float>> * cov; // covariances: K by FEATURE_LENGTH Matrix
	MFCC_Feature & addBToA(MFCC_Feature & A, const MFCC_Feature & B); // to add vector B to vector A
	MFCC_Feature & divideAByB(MFCC_Feature & A, long B); // to divide vector A by B
	void estimateCovariances(std::vector<std::vector<MFCC_Feature>> & container); // to estimate covariance
public:
	// Gaussian probability distance function
	float distanceMFCC(const MFCC_Feature & v1, int state);
	// construction
	SegmentalKmeans();
```

```
        // set parameters of the model
        void setParameters(int K, int FEATURE_LENGTH);
        // set maximum # of iteration
        void setMaxIteration(int max_iteration);
        // set convergence threshold
        void setMinDelta(int min_delta);
        // commit single Gaussian HMM
        void commit(std::vector<std::vector<MFCC_Feature>>);
        // destruction
        ~SegmentalKmeans();
        // get the model
        HMMModel getModel();
        // transition cost
        float getCost(int s1, int s2);
        // get # of states
        int getNumberOfStates();
};
#endif /* SegmentalKmeans_hpp */
////////////////////////////////////////////////////////////////////////////////////////////////////
#ifndef SegmentalKmeansGMM_hpp
#define SegmentalKmeansGMM_hpp
#ifndef MFCC_Feature
#define MFCC_Feature std::vector<float>
#define MEBS 1e-100
#define PI 3.14
#endif

#include <vector>
#include <stdio.h>
#include <math.h>
```

```
struct HMMModel_GMM{
    std::vector<float> * entry_cost;
    std::vector<std::vector<float>> * transition_cost;
    std::vector<std::vector<MFCC_Feature>> * miu;
    std::vector<std::vector<std::vector<float>>> * cov;
    std::vector<std::vector<float>> * alpha;
};
class SegmentalKmeansGMM{
private:
    int K; // K clusters
    int FEATURE_LENGTH; // # of MFCC features
    int KERNEL_NUMBER; // # of GMM kernel
    int MAX_ITERATIONS; // maximum # of iteration
    float MIN_DELTA; // convergence threshold
    std::vector<float> * entry_cost; // entry costs
    std::vector<std::vector<float>> * transition_cost; // transition costs
    std::vector<std::vector<MFCC_Feature>> * miu; // centers: K * KERNEL_NUMBER * FEATURE_LENGTH Matrix
    std::vector<std::vector<std::vector<float>>> * cov; // covariances: K * KERNEL_NUMBER * FEATURE_LENGTH
Matrix
    std::vector<std::vector<float>> * alpha; // P(y | theta)
    float EuclideanDistance(const MFCC_Feature & A, const MFCC_Feature & B);
    MFCC_Feature & addBToA(MFCC_Feature & A, const MFCC_Feature & B); // to add vector B to vector A
    MFCC_Feature & divideAByB(MFCC_Feature & A, long B); // to divide vector A by B
    // compute the probability of a frame xi s.t a given Gaussian distribution difined by miul and covl
    double GaussianProbability(const MFCC_Feature & xi, MFCC_Feature & miul, std::vector<float> & covl);
    // use EM to estimate GMM model, return the maximum Loss
    float estimateGMM(std::vector<std::vector<MFCC_Feature>> & container);
    // initialize miu and cov (Kmeans)
    void initializeGMMParameters(std::vector<std::vector<MFCC_Feature>> & container);
    void clear();
public:
```

```
    // distance function - GMM distance
    double distanceMFCC(const MFCC_Feature & v1, int state);
    // construction
    SegmentalKmeansGMM();
    // set parameters for HMM
    void setParameters(int K, int FEATURE_LENGTH, int KERNEL_NUMBER);

    // set maximum # of iteration
    void setMaxIteration(int max_iteration);
    // set convergence threshold
    void setMinDelta(int min_delta);
    // train HMM model
    void commit(std::vector<std::vector<MFCC_Feature>>);
    // destruction
    ~SegmentalKmeansGMM();
    HMMModel_GMM getModel();
    // transition cost
    float getCost(int s1, int s2);
    // get # of states
    int getNumberOfStates();
};

#endif /* SegmentalKmeansGMM_hpp */
```

4、 Experimental results and discussion

In HMM-based recognition with single Gaussian, the accuracy could be up to 90%, except that the digit 9 always be wrongly recognized as digit 5. But other digits could be correctly expected. And we also think it is unreasonable to segment all training sequence to 5 states. Since different digit has different phoneme, we should adjust the state number for different digit. We find that the digit one, two, eight, 4 states is enough. And in our implementation, we create a function

to freely choose the state for each digit:

```
void setParameters(int K, int FEATURE_LENGTH);
```

And for multiple Gaussian, the accuracy is lower, only 70%. The main reason is that the template for digit 5, 8, 9 is wrong. It may because the possibility between one mfcc feature is far away from other four Gaussian distribution kernels. The possibility of this feature to be classified into all the features are low, which would be regarding as zero when in the format double. And this can cause computation problem.

In future work, we are going to increase the number of digit recordings as our template, and try different Gaussian numbers, and set different state number for different digit, to see whether the accuracy could be increased.