UC San Diego
Electrical and Computer Engineering
JACOBS SCHOOL OF ENGINEERING

Simple Speech Calculator

SIWEN YAN

TIANYU ZHUANG

YI ZHENG

Abstract

This report introduces a calculator controlled by speech input. A calculator is a device that can do arithmetic operation on several numbers, our simple speech calculator can calculate four arithmetic operations from 1 to 20.

Speech recognition refers to methodologies and technologies that enable the recognition and translation of spoken language into texts by computers and computerized devices such as those categorized as smart technologies and robotics.

This report describes how to develop the theorem to design the speech calculator, how to implement the HMM algorithms to realize the function of the speech calculator, as well as the illustration of source code in detail.

Description

Theorem of the algorithm

The voices recorded are saved as audio files (such as one.wav, two.wav, etc.), and the waveform will be divided into identical frames (25ms) with increment of 10ms and covered by hamming window. Each frame will be analyzed with MFCC function, which will give out 39-dimension vector as the feature extraction process. The allocation of 39-dimensional features is shown below:

- MFCC:
  - 12 MFCC (mel frequency cepstral coefficients)
  - 1 energy feature
  - 12 delta MFCC features
  - 12 double-delta MFCC features
  - 1 delta energy feature
  - 1 double-delta energy feature
- Total 39-dimensional features

Then 39-dimension mfcc of single frame is imported into 8-bit vector quantization, thus these vectors are our observations and we may have 256 different observations. One HMM model represents one word. One word is composed of some phonemes, namely, multiple states with one state illustrating one phoneme. From state to state, we have transition probabilities, which will form a transition probability matrix. At the same time, there's a probability of observation for each state, that's the emission probability. The emission probability is given by Gaussian mixture model, which contains s set of parameters like number of cluster, mean and variance. And there's also a probability for the start state. These are the parameters we need to get from the training part for further test. It goes as the following figure 1.

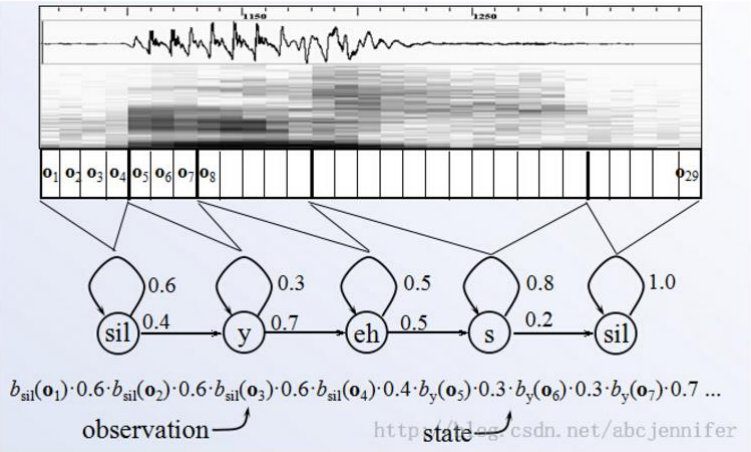
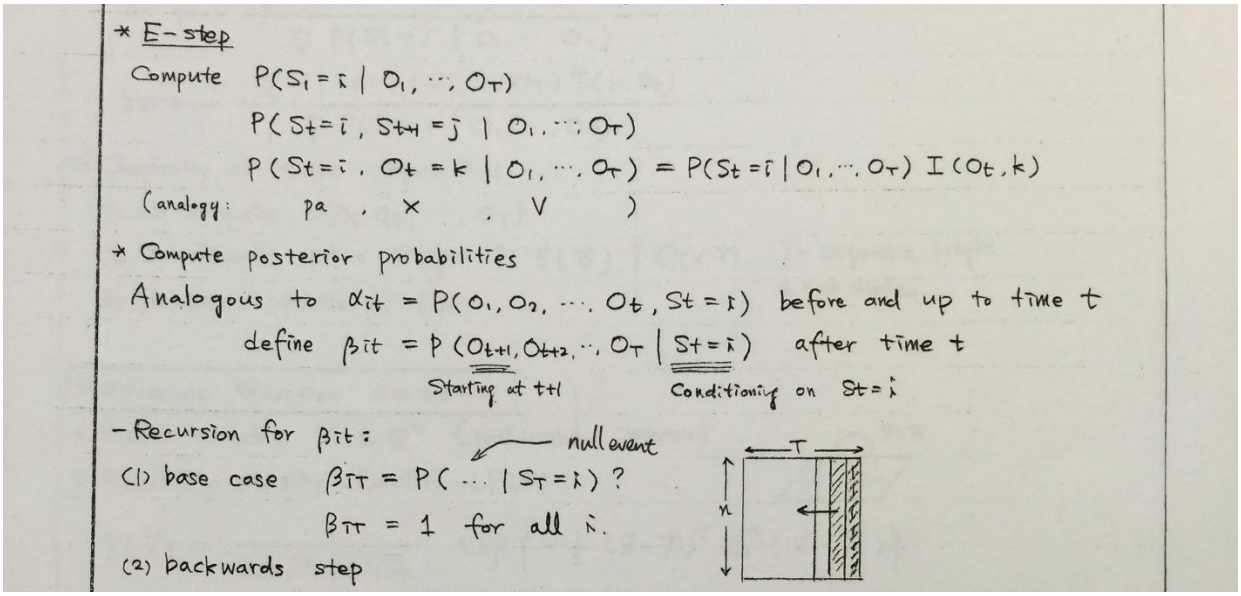


Figure 1

The training process includes training the parameters of codebook (called code vector and partition area) and hidden Markov model. First implementing k means method to define centroids, then using LBG algorithm iteratively update the parameters until the code book size reaches 256, ending the iteration when the derivation is small enough. Another training process involved is setting up model of hidden Markov model based on given observations, which is the feature vectors of the training samples, with each model illustrates one word. The training samples come from folder with same name. Here we use Expectation Maximization (EM) algorithm and Baum-Welch Algorithm to re-estimate (iterative update and improvement) of HMM parameters. We initialize the parameters to appropriate values then recursively calculate the forward and backward variable α and β to get the posterior probabilities we need in Maximization step. Then in Maximization step, we update our parameters with the posterior probabilities we calculated to maximize the likelihood given the observations. The theoretical calculation is shown below:



$$\beta_{Tt} = P(O_{t+1}, \cdots, O_T \mid S_t = i)$$

$$= \sum_{j=1}^{n} P(O_{t+1}, \cdots, O_T, S_{t+1} = j \mid S_t = i) \quad \text{marginalization.}$$

$$= \sum_{j=1}^{n} P(O_{t+1}, \cdots, O_T \mid S_{t+1} = j, S_t = i) P(S_{t+1} = j \mid S_t = i) \quad \text{product rule}$$

conditional independence →

$$= \sum_{j=1}^{n} P(O_{t+1} \mid S_{t+1} = j) P(O_{t+2}, \cdots O_T \mid S_{t+1} = j, O_{t+1}) P(S_{t+1} = j \mid S_t = i) \quad \text{product rule}$$

$$= \sum_{j=1}^{n} b_j(O_{t+1}) \beta_{j,t+1} a_{ij}$$

Computing α & β matrix ⟷ "forward-backward" algorithm (a.k.a Baum-Welch)

- Posterior probabilities in E-step:

$$P(S_t = i, S_{t+1} = j \mid O_1, \cdots O_T) = \frac{P(S_t = i, S_{t+1} = j, O_1, \cdots, O_T)}{P(O_1, \cdots, O_T)}$$

$$= \frac{P(O_1, \cdots, O_t, S_t = i) P(S_{t+1} = j \mid S_t = i, O_1, \cdots O_t) P(O_{t+1} \mid S_{t+1} = j, S_t = i, O_1, \cdots, O_t) \times P(O_{t+2}, \cdots O_T \mid S_{t+1} = j, S_t = i, O_{t+1})}{P(O_1, O_2, \cdots, O_T)}$$

$$= \frac{\alpha_{it} a_{ij} b_j(O_{t+1}) \beta_{j,t+1}}{\sum_{i=1}^{n} \alpha_{iT}}$$

Other posteriors in E-step

$$P(S_t = i \mid O_1, \cdots, O_T) = \sum_{j=1}^{n} P(S_t = i, S_{t+1} = j \mid O_1, \cdots, O_T)$$

$P(S_1 = i \mid O_1, \cdots O_T)$ special case with $t = 1$.

$$P(S_t = i, Q_t = k \mid O_1, \cdots, O_T) = I(O_t, k) P(S_t = i \mid O_1, O_2, \cdots, O_T) = I(O_t, k) \sum_{j=1}^{n} P(S_t = i, S_{t+1} = j \mid O_1, O_2, \cdots, O_T)$$

marginalization

* M-step updates

$$\pi_i \leftarrow P(S_1 = i \mid O_1, \cdots, O_T)$$

$$a_{ij} \leftarrow \frac{\sum_t P(S_t = i, S_{t+1} = j \mid O_1, \cdots, O_T)}{\sum_t P(S_t = i \mid O_1, \cdots O_T)}$$

$$b_{ik} \leftarrow \frac{\sum_t P(S_t = i \mid O_1, \cdots, O_T) I(k, O_t)}{\sum_t P(S_t = i \mid O_1, \cdots, O_T)}$$

For test part, we need to extract the feature vectors from waveform, same as what we do in training part, afterwards put the 39 dimensional features into vector quantization function to find the most likely model, according to Viterbi algorithm to detect which model gives the highest likelihood probability, then output the sequence as the result (see figure 2). If the word detected is among “plus”, “minus”, “times” or “divide”, it will distinguish the spoken word as the operator, from where we can process calculation combined with first input number and second input number.

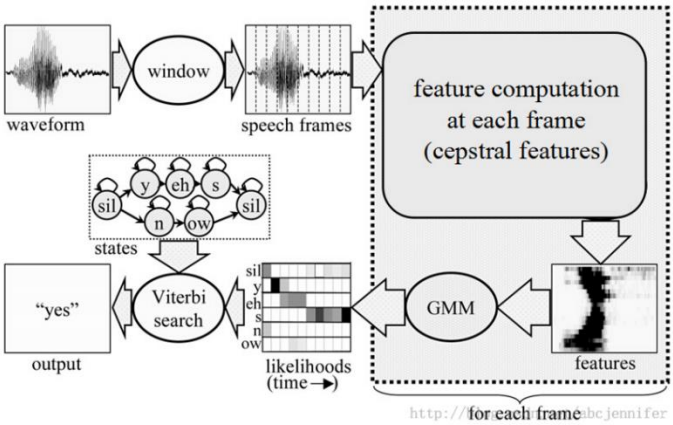


Figure 2

How to use it

The implementation of our source codes go as following. Opening the command or terminal window, using command "java name.swyan.speechcalculator.SpeechCalculator" to open the graphic user interface as figure 3.

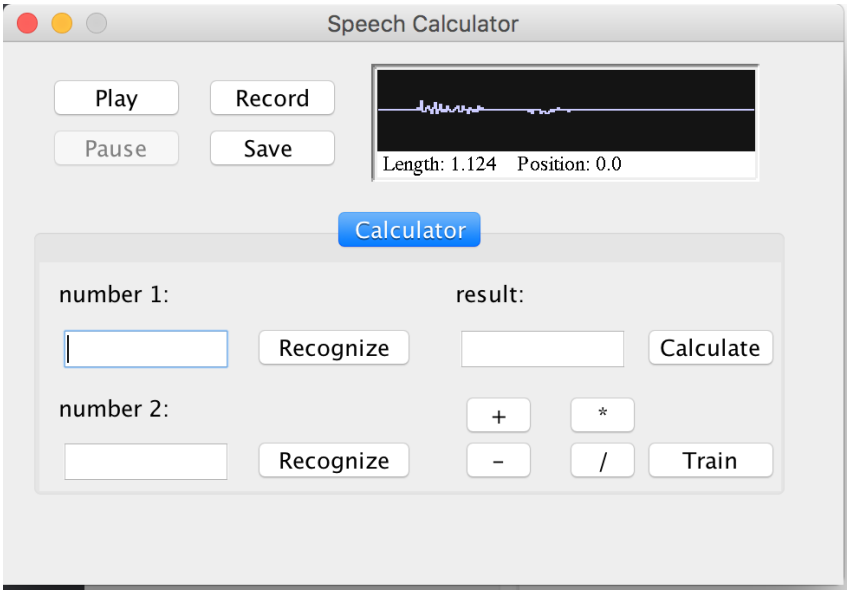


Figure 3

The interface contains two parts. The one up part is the wav player. It can record, play and save wav files. We use it to record our audio samples as well as the speech input. The other down part is the main part of our calculator.

The button "record" is used for recording voice file both in training and testing procedure. When designing to add more voice to train the model, click "save" button and save the file in the corresponding folder, i.e. if you want to train "two", simply click the "record" button, pronouncing "two" and click "record" again (after the first click, the word on the button will change to "stop"), the voice capture

procedure ends, you can click “play” to listen again, and click “save” if no noise. Then finding the folder “two” (figure 4),put it in. Afterwards, clicking “train” button to obtain the new HMM model.

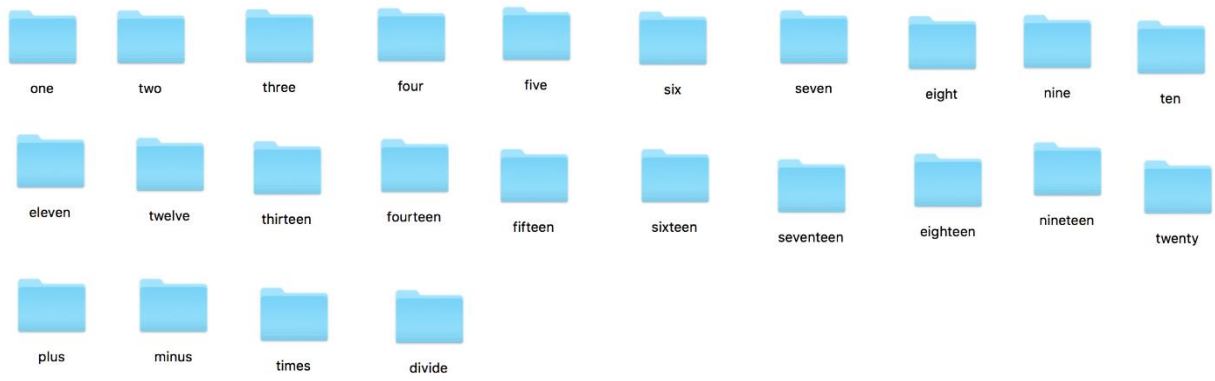


Figure 4

Testing: if you want to test if your spoken word matches the actual word well or not, click “record” button, record first vocabulary, from 1 to 20, same process as training, the difference is this time we will click the first “recognize” button, it will tell you the result, then speak the second vocabulary and verify the result. Now you have the option calculate the result by button on the panel, called option 1 or input the operator by voice, called option 2. For option 1, simply click the operator button on the panel, the dialog box below the results will show the numerical operation result. For option 2, input the voice record one of four operators, “plus”, “minus”, “times” (not “multiply”), “divide”, click the “calculate” button, the results will be demonstrated same place as option 1.

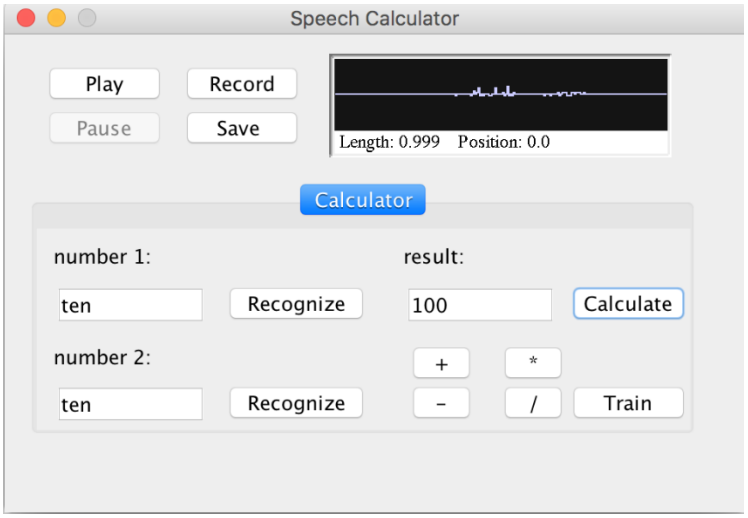


Figure 5

Important java source code description

Typical MFCC Features

We need to do feature extraction on the audio file, that is to say, extract the MFCC feature vectors from the training sets. Here we have 12 mfcc, 1energy feature, 12 delta MFCC features, 12 double delta MFCC features, 1 delta energy feature, 1 double delta energy feature.

DCT.java returns the discrete cosine transform value, since there are only real coefficients, this file performs as inverse Fourier Transform

Delta.java uses the linear regression method to calculate the delta

Energy.java returns the energy feature for MFCC

FeatureVector.java stores all the coefficients of MFCC features, which includes mfcc, delta mfcc, double delta mfcc, energy, delta energy, double delta energy

FFT.java return the real and imaginary part of DFT when input the speech signal

MFCC.java returns the 12 mel frequency cepstral coefficients

Voice format

FormatControlConf.java acts as a voice normalization device, where we can obtain encoding, rate, sampling size in bits, big endian, and channels.

JSoundCapture.java records audio in different formats and then playback the recorded audio. This program can store the voice file, such as .wav. And the voice can be loaded when we want to listen again to make sure the exact vocabulary is recorded.

PreProcess.java performs the function of preparation steps before calculating MFCC feature vectors. Steps include pre emphasis, frame blocking (calculation the numbers of frames, along with how many numbers in one frame). In the meantime, implementing hamming window in each frame.

WaveData.java deals with the transformation between wav files and amplitude float data.

Extract.java is used to get mfcc features from input wav files. It can employ the algorithms under the mfcc folder to calculate the 39-dimensional mfcc coefficients from each frame of the input wav files.

Calculator

Calculator.java introduces the operation according to the two numeral inputs with one operator input, determining the operation based on the classifier results.

Vector quantization

Points.java stores coordinates in k-dimensional space. It will be implemented by codebook together with centroid. It creates k-dimensional coordinate array. It is used for create vector for gather points in cell and centroid.

Centroid.java can allocate the points within the acceptable range of centroid to the cells. And update the centroid by taking average of all points in the cell.

CodeBookDictionary.java creates client class for codebook, which includes centroid array and the value of dimension of the codebook.

Codebook.java provides codebook for Vector Quantization component, it implements centroid and points class to realize the function of input speech signals together with set of centroid & indices output. First we need enough points to make sure we can use codebook to train. And it includes the function to show the parameters of codebook.

getDistance calculate the distance between points with centroid; groupToc cluster the points to centroid according to distance, there should be no empty cell; closePoint move the point from centroid 1 to centroid 2 if the distance is less; closestCentroidToCentroid finds the closest Centroid to a specific Centroid; closestCentroidToPoint finds the closest Centroid to a specific Points; quantize function quantizes the input array of points in k-dimensional space; split increase number of centroids by multiple of 2; initialize creates a Codebook using LBG algorithm which includes K-means, Iteration 1: repeat splitting step and K-means until required number of codewords is reached, inside the iteration 1, there is iteration 2, which perform K-means algorithm, update calculates the average of points in cell to define the new centroid.

Here we implement the LBG algorithm and k means method to iteratively calculate the code vector and partition areas to obtain optimal geographical division. LBG algorithm introduces the splitting theorem, which divided 1 code vectors to 2 code vectors, 2 code vectors to 4 code vectors and so on. To make the classification more specific, the default code book size is 256. Vector quantization method train a Codebook with given training points and Codebook.

Hidden Markov Model

HMMModel.java serves as a client class for HiddenMarkov.java, including the states, observation, transition, and symbols.

HiddenMarkov.java implements Viterbi algorithm to find the most possible sequence to for observation results.

getProbability returns the probability calculated from the testing sequence; computeAlpha, computeBeta calculate forward variable alpha and backward variable beta separately; setNumObSeq set the number of training sequences; setTrainSeq set a training sequence for re-estimation step; train determine when the iteration will terminate; reestimate implements Baum-Welch Algorithm - Re-estimate (iterative udpate and improvement) of HMM parameters; HiddenMarkov used to create a left-to-right model with multiple observation sequences for training; randomProb generates random probabilities for transition, output probabilities for initialization.

Silence removal

EndPointDetection.java performs as a voice capture device, which will remove the silence of recorded voice file. Here we use the means and standard deviation to detect the silence period, when the one-dimensional Mahalanobis distance is less than the threshold, it will be regarded as the silence period.

Database.java serves as an interface for model constructing.