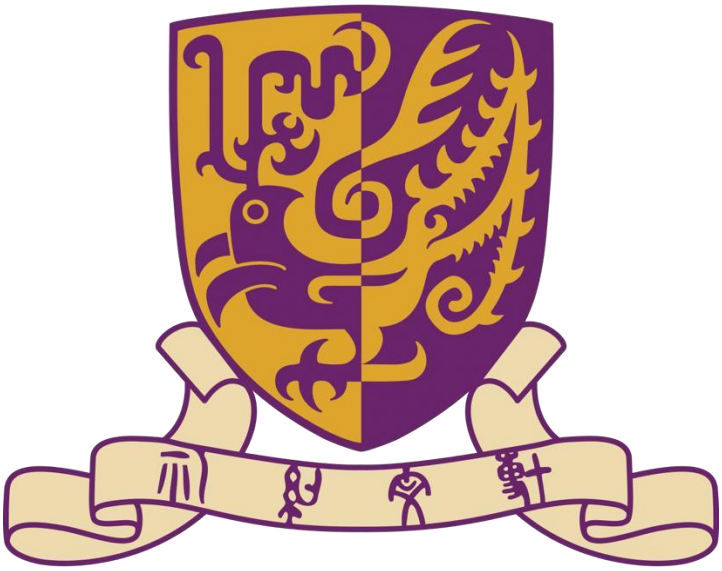

The Chinese University of Hong Kong, Shenzhen

Report for Classification of Cat Vocalizations by HMM and EM Algorithm

Author: Student Number:

Zhen Tong 120090694

December 25th, 2021

Contents

Abstract

The problem is classifying cat vocalization by machine learning. Cats' meow was recorded when they were being brushed, fed, and isolated. Mel-Frequency Cepstral Coefficients (MFCC) is used to pre-process data. The hidden Markov model (HMM) estimates the meow process. The EM algorithm is used to find the optimized transition model and emission model in HMM. Finally, the classifier estimates the similarity of the test data and three different training models. The result of the best classification is 68.78%. The EM algorithm convergence speed during training was recorded and explained. This project also gives the reason for the best hidden variable kind.

1 Introduction

Can we understand cats? The project is originally from Kaggle. To better understand cats' communication by machine learning, this project aims to classify the cat vocalization audio into three categories. The training data from Kaggle includes cats' vocalization records of brushing, feeding, and being isolated. We used Mel-Frequency Cepstral Coefficients (MFCC) to extract sound information. We assumed that the information could be explained by the hidden Markov model (HMM). The way cat mew can be described by the transition probability of audio states along the time sequence. Finally, the three sets of probability matrices $\lambda(A, B, \pi)$ will be the standard for the classification. The method in this project can also be used in other animal audio classifications.

2 Method

2.1 MFCC

We used Mel Frequency Cepstral Coefficient (MFCC) to extract features from a cat vocalization audio in the data processing part. The MFCC uses the MEL-scale to transfer the simple frequency to frequency animals are sensitive to. MEL scale is based on the way humans distinguish sound frequencies. It is essential to use MFCC to process sounds in voice identification if the project is done with neural network training.

Short-Time Fourier Transform (STFT)

The audio is a time-domain signal, which is impossible to see the rule of frequency change. It can be transformed into the frequency domain by Fourier transform. Short-Time Fourier Transform can get the amplitude and frequency information in a short period.

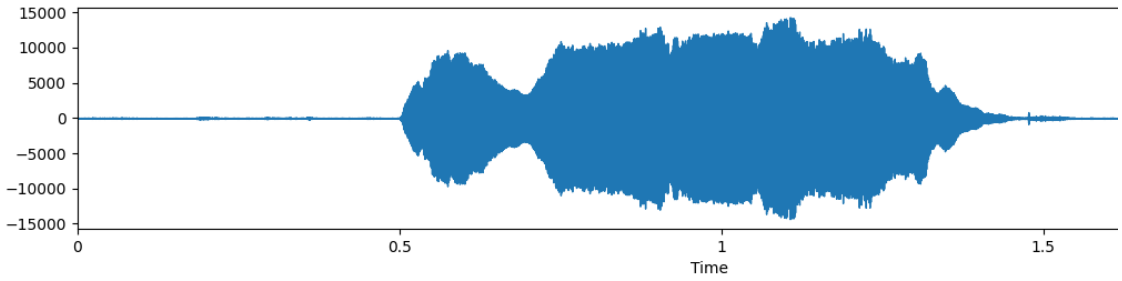

Figure 1 Amplitude vs. Time graph for a cat mew

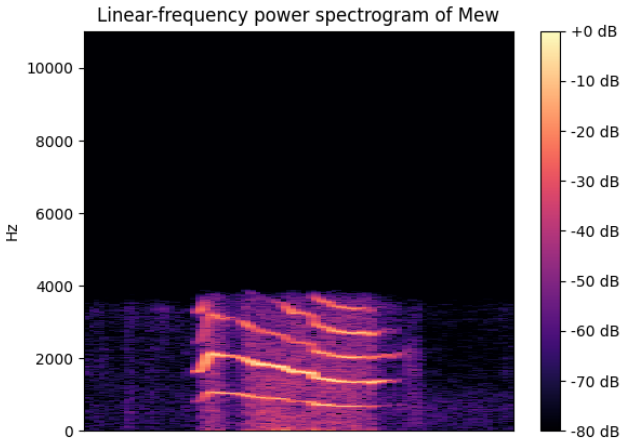

Figure 2 Linear-frequency power spectrogram by STFT

Mel scale

The Mel scale is proposed to modify hearing sensitivity to different frequencies. Specifically, the human ear is more sensitive to low-frequency changes than the high-frequency ones. Therefore, a transformation of the frequency (f) is performed to obtain the Mel scale:

$$\mathrm{M(f)} = 1125\ln(1 + \frac{f}{700}) \quad (1)$$

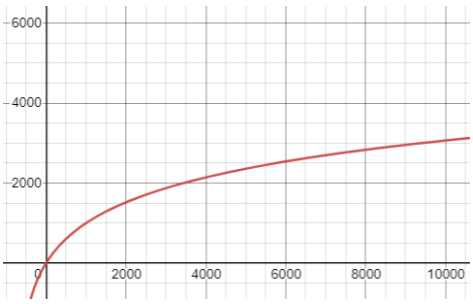

Figure 3Mel scale function (M (f) vs. f)

The whole process of MFCC will return a 13 dimension matrix, with column numbers related to the time length of each audio.

2.2 The hidden Markov model (HMM)

The human cannot understand the cats because the implicit information is behind the sound, similar to the assumption of the hidden Markov model. HMM is used to process time sequence data. The HMM assumes that the hidden variables are only related to one hidden variable before it. Each observed variable is only associated with a hidden variable before it. The basic structure of a class of models is shown as follows, $z_n$is the hidden variable, and x_n is the observed variable. In this project, the x_n is the MFCC vector we observed.

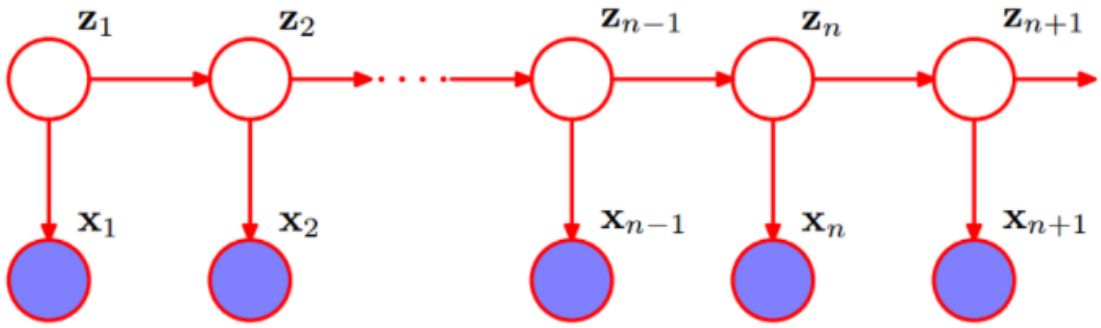

Figure 4 The hidden Markov model

In this problem, we must use the MFCC method because the HMM assumption is ideal, which requires reliable observation attributes along time. The target of the HMM is to find the transition rule for the hidden variable z_n. The transition matrix of two adjacent hidden states indicates how cats organize the implicit information.

The Transition Probability Matrix

First, z_n is the hidden vector at time t. Suppose there are K kinds of hidden states. In that case, z_n can be represented by a K dimensional variable, with each dimension corresponding to a state. Each dimension can only take two values of 0 or 1. Only one dimension has a value of 1. The conditional probability of $\mathrm{P}(z_n = q_j | z_{in-1} = q_i)$ can be represented as:

$$\mathrm{P}(z_n | z_{n-1}) = \prod_{i=1}^{K} \prod_{j=1}^{K} A_{ij} \tag{2}$$

$$\sum_{j=1}^{K} A_{ij=1}, 0 \le A_{ij} \le 1 \tag{3}$$

The Initial Probability Vector

Since the first hidden variable z_1, in the beginning, has no parent node, its distribution can be represented by a probability vector π, where the element π_k represents the probability of z_1 taking the number k state

$$\mathrm{P}(z_1 | \Pi) = \prod_{i=1}^{K} \pi_k \tag{4}$$

$$\sum_{k=1}^{K} \pi_k = 1 \tag{5}$$

The Emission Probability Matrix

The emission probability can be expressed as $P(z_n|z_{n-1}, \theta)$. The parameter of the model is denoted asθ. Since z_n is a discrete variable, when x_n is known, $P(z_n|z_{n-1}, \theta)$ is a K dimension vector, whose k element $P(z_n|z_{n-1}, \theta)_k$represents the conditional probability of $z_{nk} = 1$. So the emission probability can be written as:

$$P(x_n|z_n, \theta) = \prod_{i=1}^{K} P(x_n|\theta_n)^{z_{nk}} \tag{6}$$

In this project, I used normal distribution as an emission probability. With the initial probability model, transfer probability model, and emission probability model, the joint distribution of all variables can be expressed as:

$$P(X|Z, \lambda) = P(z_1|\pi) \left[\prod_{n=2}^{N} P(z_n|z_{n-1}, A)\right]\left[\prod_{n=2}^{N} P(x_n|z_n, \theta)\right] \tag{7}$$

$X = \{x_1, x_2, \dots, x_N\}$ represents all observed variables. $Z = \{z_1, z_2, \dots, z_N\}$ represents all hidden variables $\lambda = \{\Pi, A, \theta)$

2.3 EM Algorithm

We were not told how many implicit states a cat can express in this project. As a result, we assume there are k kinds of hidden states. The parameter learning of HMM adopts the maximum likelihood method. In the absence of a hidden state, the maximum likelihood method is general and reasonable. Due to hidden variables, it is impossible to obtain the analytical solution of the parameters directly, so it is necessary to adopt EM (Expectation-Maximization) algorithm and gradually iterate until convergence to obtain the model parameters. Fortunately, the EM algorithm converges quickly for looking for the most reasonable argument in this project. The convergence was achieved after ten to twenty rounds of iterations.

E step

In the E step, the posterior distribution of the hidden variable is$P(X|Z, \lambda)$, whose parameter λ is assumed as known. The expectation is a function of the old parameters λ and new parameter $\bar{\lambda}$

$$L(\lambda, \bar{\lambda}) = \sum_{Z} P(Z|X, \lambda) \ln P(X, Z|\lambda) \tag{8}$$

At M step, I calculated the maximum expected value of the logarithmic likelihood $L(\lambda, \bar{\lambda})$ of the whole data under the posterior distribution. The logarithm operation can conveniently change the multiplication into the accumulation.

$$\ln P(X, Z|\lambda) = P(z_1|\pi)\left[\sum_{n=2}^{N} P(z_n|z_{n-1}, A)\right]\left[\sum_{n=2}^{N} P(x_n|z_n, \theta)\right] \tag{9}$$

$$= \sum_{k=1}^{K} z_{1k}\ln(\pi_k) + \sum_{n=1}^{N}\sum_{j=1}^{K}\sum_{i=1}^{K} z_{n-1,i}z_{n,j}\ln(A_{i,j}) + \sum_{n=1}^{N}\sum_{i=1}^{K} z_{n,k}\ln(P(x_n|\theta_k)$$

To better introduce the following formula, two notations $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$, which respectively represent the posterior distribution of z_n and the joint posterior distribution of $z_n$and z_{n-1}

$$\gamma(z_n) = P(z_n|X, \theta) \tag{10}$$

$$\xi(z_{n-1}, z_n) = P(z_{n-1}, z_n|X, \theta) \tag{11}$$

M Step

Substitute the three parts on the right of Equation (10) into Equation (9)

The first part is:

$$\sum_{Z} P(Z|X, \lambda)\sum_{k=1}^{K} z_{1k}\ln(\pi_k) = \sum_{k=1}^{K}\gamma(z_{1k})\ln(\pi_k) \tag{12}$$

The second part is:

$$\sum_{Z} P(Z|X, \lambda)\sum_{n=1}^{N}\sum_{j=1}^{K}\sum_{i=1}^{K} z_{n-1,i}z_{n,j}\ln(A_{i,j}) = \sum_{n=1}^{N}\sum_{j=1}^{K}\sum_{i=1}^{K}\xi(z_{n-1,i}, z_{n,j})\ln(A_{i,j}) \tag{13}$$

The third part is:

$$\sum_{Z} P(Z|X, \lambda)\sum_{n=1}^{N}\sum_{i=1}^{K} z_{n,k}\ln(P(x_n|\theta_k) = \sum_{n=1}^{N}\sum_{i=1}^{K}\gamma(z_{n,k})\ln(P(x_n|\theta_k) \tag{14}$$

Sum up and get:

$$L(\lambda, \bar{\lambda}) = \sum_{k=1}^{K}\gamma(z_{1k})\ln(\pi_k) + \sum_{n=1}^{N}\sum_{j=1}^{K}\sum_{i=1}^{K}\xi(z_{n-1,i}, z_{n,j})\ln(A_{i,j}) + \sum_{n=1}^{N}\sum_{i=1}^{K}\gamma(z_{n,k})\ln(P(x_n|\theta_k) \tag{15}$$

$\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ are the variables required in E step, which are constant in the M step. Then let $L(\lambda, \bar{\lambda})$ take the maximum of the parameter. It can be obtained by using the Lagrange method:

$$\pi_k = \frac{\gamma(z_{1k})}{\sum_{k=1}^{K}\gamma(z_{1k})} \tag{16}$$

$$A_{i,j} = \frac{\sum_{n=1}^{N}\xi(z_{n-1,j}, z_{nk})}{\sum_{k=1}^{K}\sum_{n=2}^{N}\xi(z_{n-1,j}, z_{nk})} \tag{17}$$

The optimization of the parameter θ depends on the specific form of the emission probability model. In this project, the emission probability is gaussian distribution. We have

$$\mathrm{P}(x_n|\theta_k) = N(x_n|\mu_k, \Sigma_k) \tag{18}$$

We take the derivative of the parameter and set the derivative to be 0 to obtain:

$$\mu_k = \frac{\sum_{n=1}^{N}\gamma(z_{\mathrm{n}k})\, x_n}{\sum_{n=1}^{N}\gamma(z_{\mathrm{n}k})} \tag{19}$$

$$\Sigma_k = \frac{\sum_{n=1}^{N}\gamma(z_{\mathrm{n}k})\,(x_n-\mu_k)(x_n-\mu_k)^T}{\sum_{n=1}^{N}\gamma(z_{\mathrm{n}k})} \tag{20}$$

In this project μ_k, Σ_k is initiated as independent and identically distributed as the initial value.

3 Process

3.1 Feature Extraction by MFCC

In this project, I used python 3 for implementation. First, we transform the audio into MFCC form as a $13 \times n$ matrix.

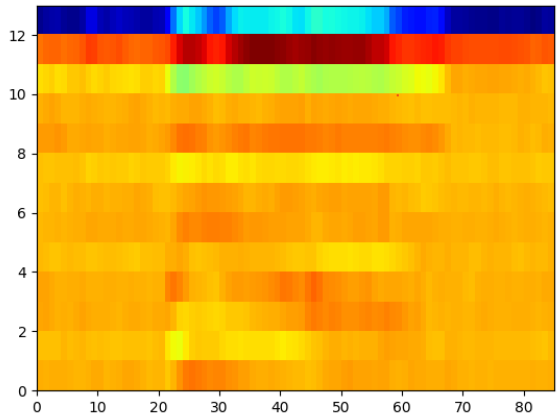

Figure 5 Heat map of MFCC 13 dimension matrix

In Figure 4, the x-axis is the time, and the y axis is 13 dimensions of the MFCC matrix. We can observe that each dimension changed continuously from left to right since the color changed continuously. Because the data is continuous, we need to unify it before using it.

Then, all the audio files can be first transformed into an MFCC matrix. We combine all the matrices that cats mew in the same situation into a $13 \times N$ matrix, which can be understood as cats keep on mewing one after another for quite a long time. Continuous MFCC variable values need to be converted to uniform scale discrete variables.

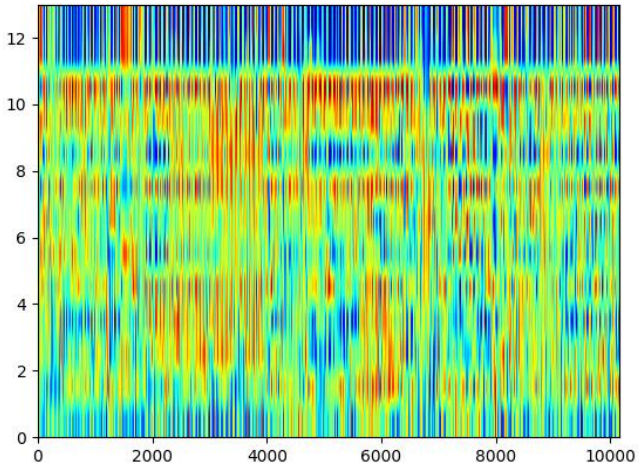

Figure 6 Combined MFCC of different cats in the same situation

3.2 EM Algorithm Structure

The goal of the E step is to evaluate $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$. We build α here by dynamic programming, where the element $\{\alpha_{i,j}\} \in R^{K\times N}$ created the forward and backward probability matrices α and β. The Matrix β is a K × N matrix where $\beta_{i,j}$ represents the backward probability.

$$\gamma(z_n) = P(z_n|X, \theta) = \alpha \cdot \beta$$

The goal of the M step is to calculate the initial probability vector π, the transition probability matrix A the coefficient μ, Σ in Gaussian distribution as emission probability. $\gamma(z_n)$ and $\xi(z_n, z_{n-1})$ are served as constant. During the E and M steps iteration, the key coefficients π, A, μ, and Σ will converge in 10 to 20 time iterations.

```
EM_teration(Data, K , iter_number){
  # Setting up total number of clusters which will be fixed
  # Initialization: Build a state transition matrix with uniform probability
    Set the implicit variable number as K
    Count = 1
    while(Count < iter_number){
        past transition matrix = transition matirx
        Count ++
        # Calculate log likelihood.
        Past log-likelihood = log-likelihood.
        Find log-likelihood by E-Step.
        Set A, π, μ and Σ  in M-Step
}
```

3.3 Classification

The key difference between the two Markov state models is the transition probability model. We calculate the difference between the two transition probability matrices and sum up the absolute value. The smaller the difference sum is, the similar the two matrices are.

$$D = \sum_{i=1}^{K}\sum_{j=1}^{K} d_{ij}, (d_{ij} \in A_{test} - A_{train}) \quad (21)$$

4 Result and Data Analysis

The classification output showed that the correctness of the classification is from. The MFCC and HMM can describe the feature of cat mew well.

```
================================
No.23>>Isolate
A Diff_B =  5.453359254507073
A Diff_F =  5.088461667656334
A Diff_I =  3.6450326444304477
classify result>> Isolate
output>>True
================================
No.24>>Feed
A Diff_B =  1.9080267829447777
A Diff_F =  1.8578693114259184
A Diff_I =  2.6889136584699056
classify result>> Feed
output>>True
================================
No.25>> Food
A Diff_B =  2.2893631314075797
A Diff_F =  1.8109969393863792
A Diff_I =  1.9373002494026363
classify result>> Isolate
output>>False
```

Figure 7 part of the classification outcome

4.1 Convergence Speed

We first divided the data set into a small set as a developing set and a bigger one as the training set. We record the logarithm likelihood.

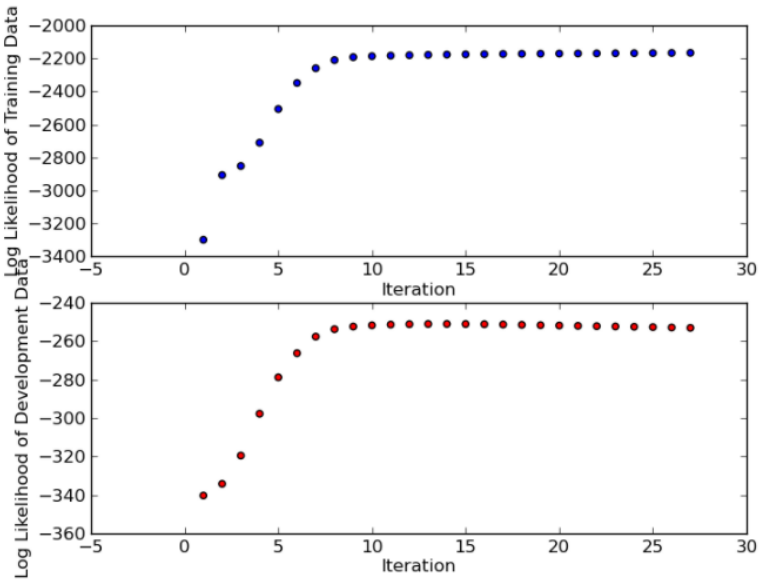

Figure 8 The logarithm likelihood convergence graph of Training Data

We observed that the convergence speed of the big set and the small set is similar because, according to the *Element of Statistical Learning*, the EM convergence speed is related to highly related to the setting value of the initial vector and transition matrix. However, the convergence speed of a single iteration process will not cause a big difference. Because the core of the EM algorithm is an optimization problem that each time of M step will move towards the optimal local solution, the iteration number is how far the initialized value is from the local variable. That is why the training number will not affect the iteration number.

4.2 The Implicit Variable Number

Because the implicit variable is unknown, the number of them is set at the iteration's beginning. We tried different numbers from 5 to 200.

(1) From the graph, the correctness reaches its peak when the number of implicit variables is 35, with correctness of 68.78%.

(2) The graph can show the quality of HMM. On the one hand, when the number of the implicit variable is small, the hidden states of the model cannot be well described, which leads to low correctness. On the other hand, when the implicit variable is too much, the hidden model will be described much too specific, which leads to overfitting.

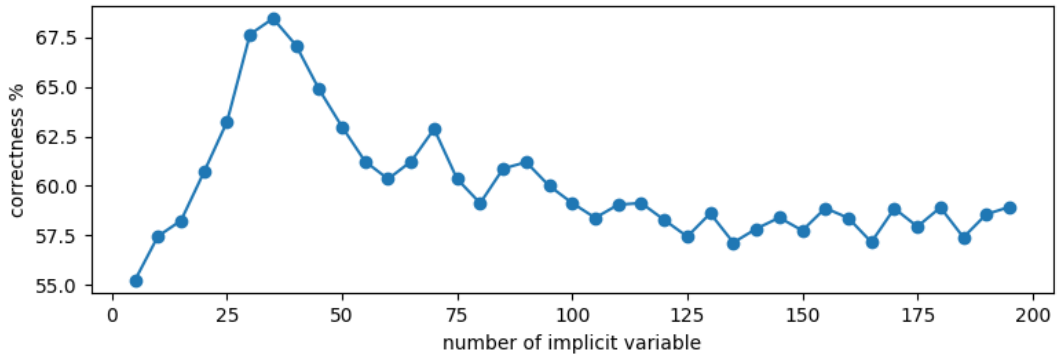

Figure 9 correctness vs. the number of the implicit variable

(3) When the number of implicit variables exceeds 80, the accuracy oscillates around 57% and slowly decreases. Because increasing the number of implicit variables to more than 200 requires a large amount of time. We estimate the accuracy will decrease by the tendency.

4.3 Error Analysis and Limitation

First, because the HMM model assumes that the hidden state is only focused on one previous state, the Markov property may cause information loss. Second, in the MFCC matrix unify step, all the vectors are converted into discrete values from 0 to 1 as ten intervals. It can be divided into 100 or 1000 intervals, indicating that the observed states can be viewed as more variable. However, the time complexity will grow, and due to device limitations, we cannot divide it into too many intervals. Another limitation is that the classification process cannot identify whether the input audio is cat vocalization.

The classification was not set as an "others" category. Those limitations can be fixed in the future.

5 Conclusion

In this project, cat vocalization is well estimated into three categories. The MFCC can reveal the frequency and time information, so we used MFCC for data pre-processing. After combining all training MFCC audio into one big matrix, we chose HMM for modeling because the information is hidden behind the sound. We set the implicit state number at the beginning and tried different numbers from 5 to 200 in the EM algorithm. Then, we compare the training transition matrix and the testing data transition matrix to decide which category the audio fits in.

We also found that the convergence speed is almost the same for the large and small training set with some initial values. Finally, we got the result that the best classification correctness is 68.78%, whose implicit variable number is 35. When the hidden variable is large, the accuracy decreases slowly.

6 Reference

6.1 Python 3 Implementation

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.  import matplotlib.mlab as mlab
4.  import csv
5.  import os
6.
7.
8.  def normPDF(x, mu, sigma):
9.
10.     size = len(x)
11.     if size == len(mu) and (size, size) == sigma.shape:
12.         det = np.linalg.det(sigma)
13.         if det == 0:
14.             #raise NameError("The covariance matrix can't be singular")
15.             return 1
16.
17.         try:
18.             norm_const = 1.0 / \
19.                 (np.math.pow((2*np.pi), float(size)/2) * np.math.pow(det, 1.0/
    2))
20.             x_mu = np.matrix(x - mu).T
21.             inv_ = np.linalg.inv(sigma)
22.             result = np.math.pow(np.math.e, -0.5 * (x_mu.T * inv_ * x_mu))
23.             #print(norm_const * result)
24.             return norm_const * result
25.         except:
26.             return 0
27.     else:
28.         raise NameError("The dimensions of the input don't match")
29.         return -1
30.
31.
```

```
32. def initForwardBackward(X, K, d, N):
33.     # Initialize the state transition matrix A. A (- R:KxK matrix
34.     # K is the implicit state number
35.     # element A_{jk} = p(Z_n = k | Z_{n-1} = j)
36.     # Therefore, the matrix will be row-wise normalized. IOW, Sum(Row) = 1
37.     # State transition probability is time independent.
38.     A = np.ones((K, K))
39.     A = A/np.sum(A, 1)[None].T
40.
41.     # Initialize the marginal probability for the first hidden variable
42.     # PI (-  R:Kx1
43.     PI = np.ones((K, 1))/K
44.
45.     # Initialize Emission Probability. We assume Gaussian distribution
46.     # for emission. So we just need to keep the mean and covariance. These
47.     # parameters are different for different states.
48.     # Mu is dxK where kth column represents mu for kth state
49.
50.     MU = np.random.rand(d, K)
51.     SIGMA = [np.eye(d) for i in range(K)]  # np.eye(d) 是 d 乘 d 对脚为 1 的矩阵
52.
53.     return A, PI, MU, SIGMA
54.
55.
56. def buildAlpha(X, PI, A, MU, SIGMA):
57.     # We build up Alpha here using dynamic programming. Alpha (- R:KxN matrix

58.     # where the element ALPHA_{ij} represents the forward probability
59.     # for jth timestep (j = 1...N) and ith state. The columns of ALPHA are
60.     # normalized for preventing underflow problem as discussed in secion
61.     # 13.2.4 in Bishop's PRML book. So,sum(column) = 1
62.     # c_t is the normalizing costant
63.     N = np.size(X, 1)
64.     K = np.size(PI, 0)
65.     Alpha = np.zeros((K, N))  # Alpha (- R:KxN matrix
66.     c = np.zeros(N)
67.
68.     # Base case: build the first column of ALPHA
69.     for i in range(K):
70.         try:
71.             Alpha[i, 0] = PI[i]*normPDF(X[:, 0], MU[:, i], SIGMA[i])
72.         except:
73.             Alpha[i, 0] = 0
74.     c[0] = np.sum(Alpha[:, 0])
75.     Alpha[:, 0] = Alpha[:, 0]/c[0]
76.
77.     # Build up the subsequent columns
78.     for t in range(1, N):
79.         for i in range(K):
80.             for j in range(K):
81.                 Alpha[i, t] += Alpha[j, t-
    1]*A[j, i]  # sum part of recursion
82.             # product with emission prob
83.             Alpha[i, t] *= normPDF(X[:, t], MU[:, i], SIGMA[i])
84.         c[t] = np.sum(Alpha[:, t])
85.         Alpha[:, t] = Alpha[:, t]/c[t]   # for scaling factors
86.     return Alpha, c
87.
88.
89. def buildBeta(X, c, PI, A, MU, SIGMA):
90.     # Beta is KxN matrix where Beta_{ij} represents the backward probability
91.     # for jth timestamp and ith state. Columns of Beta are normalized using
92.     # the element of vector c.
93.
94.     N = np.size(X, 1)
95.     K = np.size(PI, 0)
```

```
    Beta = np.zeros((K, N))

    # Base case: build the last column of Beta
    for i in range(K):
        Beta[i, N-1] = 1.

     # Build up the matrix backwards
     for t in range(N-2, -1, -1):
         for i in range(K):
             for j in range(K):
                 Beta[i, t] += Beta[j, t+1]*A[i, j] * \
                     normPDF(X[:, t+1], MU[:, j], SIGMA[j])
         Beta[:, t] /= c[t+1]
     return Beta


def Estep(trainSet, PI, A, MU, SIGMA):
    # The goal of E step is to evaluate Gamma(Z_{n}) and Xi(Z_{n-1},Z_{n})
    # First, create the forward and backward probability matrices
    Alpha, c = buildAlpha(trainSet, PI, A, MU, SIGMA)
    Beta = buildBeta(trainSet, c, PI, A, MU, SIGMA)

    # Dimension of Gamma is equal to Alpha and Beta where nth column represe
nts
    # posterior density of nth latent variable. Each row represents a state

    # value of all the latent variables. IOW, (i,j)th element represents
    # p(Z_j = i | X,MU,SIGMA)
    Gamma = Alpha*Beta

    # Xi is a KxKx(N-1) matrix (N is the length of data seq)
    # Xi(:,:,t) = Xi(Z_{t-1},Z_{t})
    N = np.size(trainSet, 1)
    K = np.size(PI, 0)
    Xi = np.zeros((K, K, N))
    for t in range(1, N):
        Xi[:, :, t] = (1/c[t])*Alpha[:, t-
1][None].T.dot(Beta[:, t][None])*A
        # Now columnwise multiply the emission prob
        for col in range(K):

            Xi[:, col, t] *= normPDF(trainSet[:, t], MU[:, col], SIGMA[col])

    return Gamma, Xi, c


def Mstep(X, Gamma, Xi):
    # Goal of M step is to calculate PI, A, MU, and SIGMA while training
    # Gamma and Xi as constant
    K = np.size(Gamma, 0)
    d = np.size(X, 0)

    PI = (Gamma[:, 0]/np.sum(Gamma[:, 0]))[None].T
    tempSum = np.sum(Xi[:, :, 1:], axis=2)
    A = tempSum/np.sum(tempSum, axis=1)[None].T

    MU = np.zeros((d, K))
    GamSUM = np.sum(Gamma, axis=1)[None].T
    SIGMA = []
    for k in range(K):
        MU[:, k] = np.sum(Gamma[k, :]*X, axis=1)/GamSUM[k]
        X_MU = X - MU[:, k][None].T
        SIGMA.append(X_MU.dot(((X_MU*(Gamma[k, :][None])).T))/GamSUM[k])
    return PI, A, MU, SIGMA
```

```
158.
159. def iteration(allData, K = 10, iter_number = 20, PI=0, A=0, MU=0, SIGMA=0):

160.
161.     (m, n) = np.shape(allData)
162.
163.     # Separating out dev and train set
164.     devSet = allData[np.math.ceil(m*0.9):, 0:].T
165.     trainSet = allData[:np.math.floor(m*0.9), 0:].T
166.
167.     # Setting up total number of clusters which will be fixed
168.     # Initialization: Build a state transition matrix with uniform probabili
    ty
169.     if type(PI) == int:
170.         A, PI, MU, SIGMA = initForwardBackward(trainSet, K, n, m)
171.
172.     # Temporary variables. X, Y mesh for plotting
173.     iter = 0
174.     prevll = -999999
175.     count = 0
176.
177.     ll_dif_ls= []
178.     while(True):
179.         past_A = A.copy()
180.         #print("Count>>", count)
181.         count += 1
182.         iter = iter + 1
183.         # E-Step
184.         Gamma, Xi, c = Estep(trainSet, PI, A, MU, SIGMA)
185.
186.         # M-Step
187.         PI, A, MU, SIGMA = Mstep(trainSet, Gamma, Xi)
188.
189.         # Calculate log likelihood. We use the c vector for log likelihood b
    ecause
190.         # it already gives us p(X_1^N)
191.         ll_train = np.sum(np.log(c))
192.         Gamma_dev, Xi_dev, c_dev = Estep(devSet, PI, A, MU, SIGMA)
193.         ll_dev = np.sum(np.log(c_dev))
194.
195.         if(iter > iter_number or abs(ll_train - prevll) < 0.005):
196.             break
197.         print(abs(ll_train - prevll))
198.         ll_dif_ls.append(ll_train - prevll)
199.         prevll = ll_train
200.
201.     print("length",len(ll_dif_ls),count)
202.     print("PI>>", PI.shape)
203.     print("A>>", A.shape)
204.     print("MU>>", MU.shape)
205.     print("SIGMA>>", len(SIGMA),SIGMA[0].shape)
206.     return [PI, A, MU, SIGMA]
207.
208.
209. def unifyData(data):
210.     for i in range(data.shape[0]):
211.         minVal = min(data[i])
212.         maxVal = max(data[i])
213.         domain = (maxVal-minVal)
214.         for j in range(data.shape[1]):
215.             absolute = (data[i][j]-minVal) / domain
216.             data[i][j] = absolute//0.1*0.1  # From 0.0 to 0.9
217.         # Deliberately split the state into 10 levels, which can be changed
    to 100 or 1000
218.     return data
219.
```

```
220. def getData(scr):
221.     f = open(scr, 'r', encoding="utf-8")
222.     reader = csv.reader(f)
223.
224.     Ls = [row for row in reader]
225.     Ls = np.array(Ls)
226.     # It's the raw data with the first row
227.     print(len(Ls))
228.     pure_data = Ls[1:, :].astype(np.float32)
229.
230.     return pure_data
231.
232. def combineMFCCmatrices(read_path):
233.     read_path = r"D:\cat_meow_clasification\jdr\train\small_B"
234.
235.     files = os.listdir(read_path)
236.     combinedM = np.zeros((13, 0))
237.     for file_name in files:
238.         # 读取单个文件内容
239.         file_data = getData(read_path+"\\"+file_name)
240.         unified_data = unifyData(file_data)
241.         combinedM = np.concatenate((combinedM, unified_data), axis=1)
242.     combinedM = combinedM[:, 1:]
243.     return combinedM
244.
245. # main()
246. data_B = combineMFCCmatrices(r"D:\cat_meow_clasification\jdr\train\small_B")
247. data_F = combineMFCCmatrices(r"D:\cat_meow_clasification\jdr\train\small_F")
248. data_I = combineMFCCmatrices(r"D:\cat_meow_clasification\jdr\train\small_I")
249. data_I = getData("I_ANI01_MC_FN_SIM01_101.csv")
250. data_I = unifyData(data_F)
251. testData = getData("Deal_B_BRA01_MC_MN_SIM01_102.csv")
252. testData = unifyData(testData)
253. print("getB")
254. B_coeff = iteration(data_B,4)
255. F_coeff = iteration(data_F,4)
256. I_coeff = iteration(data_I,4)
257. t_coeff= iteration(testData,10,20)
258.
259. def classify(t_coeff,B_coeff,F_coeff,I_coeff):
260.     dif_B = cal_diff(t_coeff,B_coeff)
261.     dif_F = cal_diff(t_coeff,F_coeff)
262.     dif_I = cal_diff(t_coeff,I_coeff)
263.     small_idx = [dif_B,dif_F,dif_I].index(min([dif_B,dif_F,dif_I]))
264.     print(small_idx)
265.     return small_idx
266.
267. def cal_diff(t_coeff, cmp_coeff):
268.     t_A = t_coeff[1]
269.     cmp_A = cmp_coeff[1]
270.     A_diff = 0
271.     for i in range(t_A.shape[0]):
272.         for j in range(t_A.shape[1]):
273.             A_diff += abs(t_A[i][j]-cmp_A[i][j])
274.     print("A Diff = ",A_diff)
275.     return A_diff
276.
277. def test_correctness(B_coeff,F_coeff,I_coeff):
278.     read_path = r"D:\cat_meow_clasification\jdr\test\Test_B"
279.     files = os.listdir(read_path)
280.
281.     correct = 0
282.     sum= 0
```

```
283.     for file_name in files:
284.         # 读取单个文件内容
285.         sum+=1
286.         testData = getData(read_path+"\\"+file_name)
287.         testData = unifyData(testData)
288.         t_coeff= iteration(testData,4)
289.         if classify(t_coeff,B_coeff,F_coeff,I_coeff) == 0:
290.             correct += 1
291.     print("Correctness>>", correct/sum)
292. 
293. test_correctness(B_coeff,F_coeff,I_coeff)
```