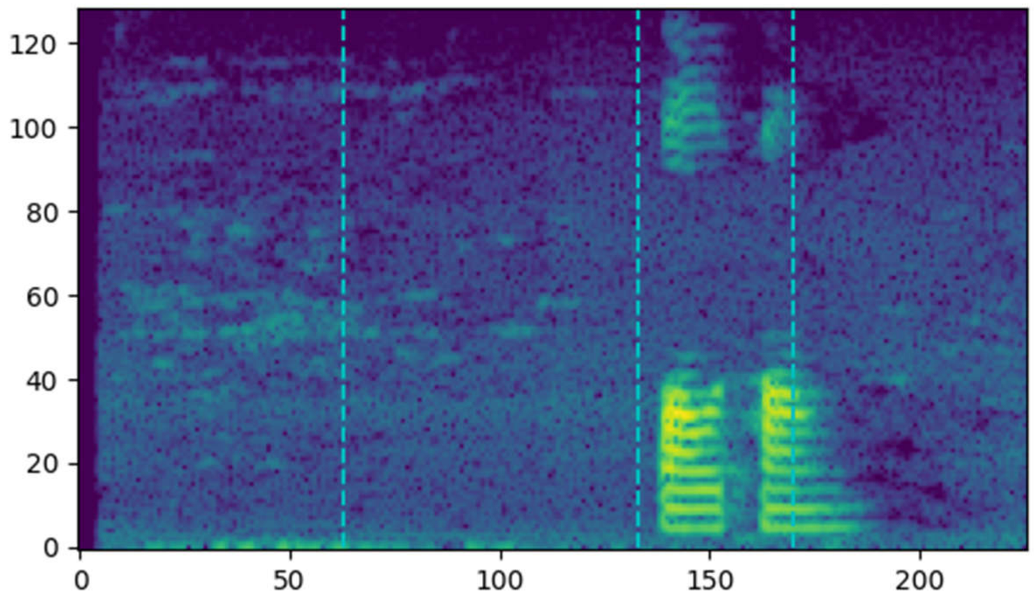
基于GMM-HMM的孤立词识别

GMM-HMM based Isolated Word Recognition

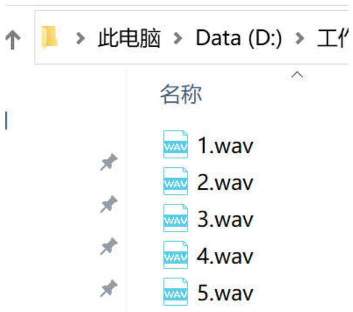


于泓
鲁东大学
信息与电气工程学院
2021.6.28

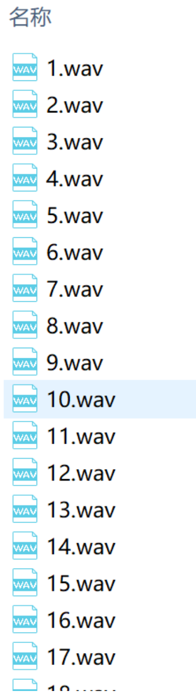
数据库简介

训练数据
包括14个孤立词

每条孤立词
5条语音



测试语音98条
每词 7条



样例

杭州

煲汤

爆炒

任务实现过程

训练部分

(1) 特征提取 MFCC

(2) 为每个孤立词训练一个GMMHMM

共14个

测试部分

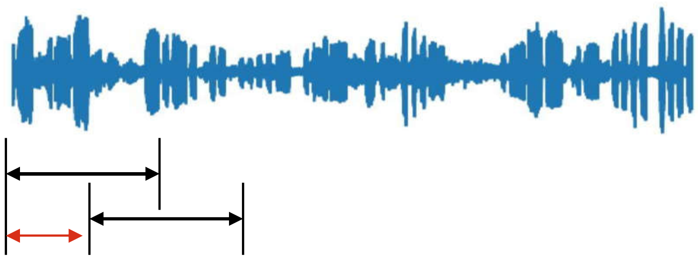
(3) 测试语音特征提取

(4) 分别送入14个GMMHMM模型计算概率值

根据最大值实现分类

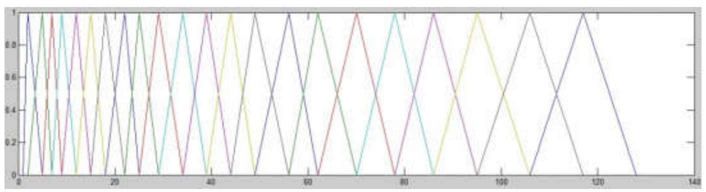
数据准备:

MFCC特征提取:



帧移
Step-size

帧长
(win_length)



[N/2 +1, n_mel]

Mel 滤波器组

(1) 分帧 预加重 加窗

(2) FFT N点 [n_frame, N/2+1]

(3) Mel 滤波器组（在频域进行局部平均）n_mel

(4) log

(5) DCT （取前n个系数）n-MFCC

(6) 系数提升，提高后面几个系数的数值 lifter

(7) 信号正则 (X-mean)/std

(8) 添加差分项

[D,T]

```
import librosa

def extract_MFCC(wav_file):
    # 读取音频数据
    y,sr = librosa.load(wav_file,sr=8000)
    # 提取特征
    fea = librosa.feature.mfcc(y,sr,n_mfcc=12,n_mels=24,n_fft = 256, win_length=256,hop_length=80,lifter=12)
    # 进行正则化
    mean = np.mean(fea,axis=1,keepdims=True)
    std = np.std(fea,axis =1,keepdims=True)
    fea = (fea-mean)/std
    # 添加1阶差分
    fea_d = librosa.feature.delta(fea)
    fea = np.concatenate([fea.T, fea_d.T],axis=1)

    return fea
```

GMM-HMM 初始化

n_state =4

n_mix =3

Pi = [1,0,0,0]

A = [0.5, 0.5, 0, 0
0, 0.5, 0.5, 0
0, 0 , 0. 5, 0.5
0, 0, 0, 1]

单向转移

```
def init_para_hmm(collect_fea,N_state,N_mix):

    # 初始 一定从 state 0 开始
    pi = np.zeros(N_state)
    pi[0] = 1

    #  当前状态 转移概率0.5 下一状态 转移概率0.5
    #  进入最后一个状态后不再跳出
    A = np.zeros([N_state,N_state])
    for i in range(N_state-1):
        A[i,i] = 0.5
        A[i,i+1] = 0.5
    A[-1,-1] =1

    feas = collect_fea
    len_feas = []
    for fea in feas:
        len_feas.append(np.shape(fea)[0])

    _,D = np.shape(feas[0])
    hmm_means = np.zeros([N_state,N_mix,D])
    hmm_sigmas= np.zeros([N_state,N_mix,D])
    hmm_ws = np.zeros([N_state,N_mix])
```

GMM 初始化

1、 将特征平分成 4份
2、 在每份内，利用kmeans
聚成3类
3、计算每类的mu、sigma
4、统计每类的样本数 计算 w

```
    for s in range(N_state):

        sub_fea_collect = []
        # 初始化时 先为每个状态平均分配特征
        for fea,T in zip(feas,len_feas):

            T_s = int(T/N_state)*s
            T_e = (int(T/N_state))*(s+1)

            sub_fea_collect.append(fea[T_s:T_e])
        ws, mus, sigmas = gen_para_GMM(sub_fea_collect,N_mix)
        hmm_means[s]=mus
        hmm_sigmas[s]=sigmas
        hmm_ws[s] = ws

    return pi,A,hmm_means,hmm_sigmas,hmm_ws
```

```python
from sklearn.cluster import KMeans

def run_kmeans(dataset,K, m = 20):
    labs = KMeans(n_clusters=K, random_state=9).fit_predict(dataset)
    return labs

def gen_para_GMM(fea_collect,N_mix):

    # 首先对特征进行kmeans 聚类
    feas = np.concatenate(fea_collect,axis=0)
    N,D = np.shape(feas)
    # print("sub_fea_shape",feas.shape)
    # 初始化聚类中心
    labs = run_kmeans(feas,N_mix, m = 20)
    mus = np.zeros([N_mix,D])
    sigmas = np.zeros([N_mix,D])
    ws = np.zeros(N_mix)
    for m in range(N_mix):
        index = np.where(labs == m)[0]
        # print("----index---------",index)
        sub_feas = feas[index]
        mu = np.mean(sub_feas,axis=0)
        sigma = np.var(sub_feas,axis=0)
        sigma = sigma+0.0001
        mus[m] = mu
        sigmas[m] = sigma

        # print("------N  D-------",N,np.shape(index)[0])
        ws[m] = np.shape(index)[0]/N
    ws = (ws+0.01)/np.sum(ws+0.01)
    return ws,mus,sigmas
```

整体训练流程

特征提取
统计时长

```
if __name__ == "__main__":

    models = []
    train_path = "train"
    for i in range(1,15):

        # 进入孤立词i所在的文件夹
        wav_path = os.path.join(train_path,str(i))
        collect_fea = []
        len_feas = []
        dirs = os.listdir(wav_path)
        for file in dirs:
            # 找到 .wav 文件并提取特征
            if file.split(".")[-1]=="wav":
                wav_file = os.path.join(wav_path,file)
                fea = extract_MFCC(wav_file)
                collect_fea.append(fea)
                len_feas.append(np.shape(fea)[0])
```

```
        # 获取模型参数初始化
        N_state = 4
        N_mix = 3
        pi,A,hmm_means,hmm_sigmas,hmm_ws=init_para_hmm(collect_fea,N_state,N_mix)

        train_GMMHMM = GMMHMM(n_components=N_state,
                              n_mix=N_mix,
                              covariance_type='diag',
                              n_iter =90,
                              tol =1e-5,
                              verbose = False,
                              init_params ="",
                              params ="tmcw",
                              min_covar = 0.0001
                              )
        train_GMMHMM.startprob_ = pi
        train_GMMHMM.transmat_ = A

        train_GMMHMM.weights_ = hmm_ws
        train_GMMHMM.means_ =  hmm_means
        train_GMMHMM.covars_ = hmm_sigmas

        print("train GMM-HMM",i)
        train_GMMHMM.fit(np.concatenate(collect_fea,axis=0),np.array(len_feas))

        models.append(train_GMMHMM)

    np.save("models_hmmlearn.npy",models)
```

测试部分

```
# 测试部分
test_dir ="test"

models = np.load("models_hmmlearn.npy",allow_pickle=True)
count = 0
count2 =0
for i in range(98):
    # 读取wav文件
    wav_file = os.path.join(test_dir,str(i+1)+".wav")
    fea = extract_MFCC(wav_file)

    lab_true = int(i//7)+1
    scores = []
    scores2 = []
    for m in range(1,15):
        model = models[m-1]
        score,_ = model.decode(fea)
        scores.append(score)
        score2= model.score(fea)
        scores2.append(score2)

    det_lab = np.argmax(scores)+1
    det_lab2 = np.argmax(scores2)+1
    if det_lab == lab_true:
        count = count+1

    if det_lab2 == lab_true:
        count2 = count2+1

    print("true lab  %d det lab1 %d  det lab2 %d "%(lab_true,det_lab,det_lab2))
print("decode  %.2f   "%(count*100/98))
```

获取真实标签

在每一个模型上进行测试
（两种打分方案）

根据分值进行识别

```
PS D:\工作相关\我设计的课程\python与人工智能课程设计\HMM\GMMHMM_speech> python .\isolated_word_recognition_hmmlearn.py
train GMM-HMM 1
train GMM-HMM 2
train GMM-HMM 3
train GMM-HMM 4
train GMM-HMM 5
train GMM-HMM 6
train GMM-HMM 7
train GMM-HMM 8
train GMM-HMM 9
train GMM-HMM 10
train GMM-HMM 11
train GMM-HMM 12
train GMM-HMM 13
train GMM-HMM 14
true lab  1 det lab1 1  det lab2 1
true lab  1 det lab1 2  det lab2 2
true lab  1 det lab1 2  det lab2 2
true lab  1 det lab1 2  det lab2 2
true lab  1 det lab1 2  det lab2 2
true lab  1 det lab1 2  det lab2 2
true lab  1 det lab1 2  det lab2 2
true lab  2 det lab1 2  det lab2 2
true lab  2 det lab1 2  det lab2 2
true lab  2 det lab1 2  det lab2 2
true lab  2 det lab1 2  det lab2 2
true lab  2 det lab1 2  det lab2 2
true lab  2 det lab1 2  det lab2 2
```

```
true lab  13 det lab1 13  det lab2 13
true lab  13 det lab1 13  det lab2 13
true lab  13 det lab1 13  det lab2 13
true lab  13 det lab1 13  det lab2 13
true lab  14 det lab1 14  det lab2 14
true lab  14 det lab1 14  det lab2 14
true lab  14 det lab1 14  det lab2 14
true lab  14 det lab1 14  det lab2 14
true lab  14 det lab1 7  det lab2 7
true lab  14 det lab1 14  det lab2 14
true lab  14 det lab1 7  det lab2 7
decode  58.16
```

准确率
不高

可能原因:

在进行前向、后向计算时

序列越长，后面的部分概率越低，导致区分性不强

对 GMM-HMM模型进行重写

(1) 求高斯pdf时，将full 高斯改为diag高斯

$$\begin{bmatrix} d_1 & & \\ & d_2 & \\ & & d_3 \end{bmatrix}$$

$$N(\mathrm{x};\mathrm{m},\Sigma)=\frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(\mathrm{x}-\mathrm{m})\Sigma^{-1}(\mathrm{x}-\mathrm{m})^T}$$

对sigma进行限制

```
def getPdf(x,mu,sigma):

    D = np.shape(x)[0]
    # 防止sigma 过小
    sigma[sigma<0.0001]=0.0001

    # 计算行列式的值,元素连乘
    covar_det = np.prod(sigma)

    # 计算pdf
    c = 1.0 / ( (2.0*np.pi)**(float(D/2.0)) * (covar_det)**(0.5))
    temp = np.dot( (x-mu)*(x-mu) , 1.0/sigma)
    pdfval = c* np.exp(-0.5*temp)

    return pdfval
```

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}\begin{bmatrix} d_1^{-1} & & \\ & d_2^{-1} & \\ & & d_3^{-1} \end{bmatrix}\begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix}=x_1x_4d_1^{-1}+x_2x_5d_2^{-1}+x_3x_6d_3^{-1}$$

（2） 为了提高计算速度，提前计算
观测样本属于**每个状态**中的**每个高斯成分**的概率

```
def compute_B_map(datas,model):
    # 计算 B_map
    T,D = np.shape(datas)
    N_mix = np.shape(model["S"][0]["ws"])[0]
    N_state = len(model["S"])

    B_map_mix = np.zeros([T,N_state,N_mix])
    B_map = np.zeros([T,N_state])

    for t in range(T):
    # 样本在状态上的概率
        for s in range(N_state):
            # o 在状态 s 的每个 mixture上的概率
            for m in range(N_mix):
                mu =model["S"][s]["mus"][m]
                sigma = model["S"][s]["sigmas"][m]
                w = model["S"][s]["ws"][m]
                B_map_mix[t,s,m] = w*getPdf(datas[t],mu,sigma)

            # 计算 o 在 每个状态 s 上的概率
            B_map[t,s] = np.sum(B_map_mix[t,s,:])
            if B_map[t,s] ==0:
                B_map[t,s] = np.finfo(np.float64).eps

    return B_map, B_map_mix
```

防止为0

(3) 计算 前向、后向概率时加入正则项，防止概率数值过低

```
# 计算GMM-HMM参数更新时所需要的的参数
def getparam(model,observations,B_map,B_map_mix):
    o = observations
    N_samples = np.shape(o)[0]
    N_state = np.shape(model["pi"])[0]
    N_mix = len(model["S"][0]["ws"])

    # 计算前向概率
    # alpha 初始化
    alpha = np.zeros([N_samples,N_state])
    c     = np.zeros(N_samples)  # 正则项

    # 计算第0个样本属于第i个状态的概率
    alpha[0] = model["pi"]*B_map[0]
    c[0] = 1/np.sum(alpha[0])
    alpha[0] = alpha[0]*c[0]

    # 计算其他时刻的样本属第i个状态的概率
    for t in range(1,N_samples):
        s_current = np.dot(alpha[t-1],model["A"])
        # alpha[t] = s_current*model["B"](model,o[t])
        alpha[t] = s_current*B_map[t]
        alpha[t] = alpha[t]
        c[t] =1.0/(np.sum(alpha[t]))

        alpha[t] = alpha[t]*c[t]
```

前向概率

$$\alpha_{t+1}(i)=\left(\sum_{j=1}^{N}\alpha_t(j)a_{ji}\right)b_{o_{t+1}i}$$

原始的

$$\alpha_{t+1}(i)=\mathrm{c}_{t+1}(i)\left(\sum_{j=1}^{N}\alpha_t(j)a_{ji}\right)b_{o_{t+1}i}$$

(修正的)

$$\mathrm{c}_t(i)=\frac{1}{\sum_{i=1}^{N}a_t(i)}$$

修正后，保证
t时刻，在每个状态上的
前向概率加和为1

```python
# 计算后向概率
beta = np.zeros([N_samples,N_state])

# 反向初始值
beta[-1] = c[-1]

for t in range(N_samples-2,-1,-1):
    # 由t+1时刻的beta以及t+1时刻的观测值计算
    # t+1时刻的状态值
    # s_next = beta[t+1]*model["B"](model,o[t+1])
    s_next = beta[t+1]*B_map[t+1]
    beta[t] = np.dot(s_next,model["A"].T)
    beta[t] = beta[t]*c[t]
```

$$\beta_{t-1}(i) = \sum_{j=1}^{N} \beta_t(j) b_{o_t j} a_{ij}$$ 原始的

$$\beta_{t-1}(i) = \mathrm{c}_{t-1}(i) \sum_{j=1}^{N} \beta_t(j) b_{o_t j} a_{ij}$$ 修正后

同一正则项

```python
# 计算状态间的转移概率 xi
xi = np.zeros([N_samples-1,N_state,N_state])

for t in range(N_samples-1):
    denom = np.sum(alpha[t]*beta[t,:])
    temp = np.zeros([N_state,N_state])

    t_alpha = np.tile( np.expand_dims(alpha[t,:],axis=1),(1,N_state))
    t_beta = np.tile(beta[t+1,:],(N_state,1))
    # t_b = np.tile(model["B"](model,o[t+1]),(N_stats,1))
    t_b = np.tile(B_map[t+1],(N_state,1))
    temp = t_alpha*model["A"]*t_beta*t_b
    temp = temp/(denom+np.finfo(np.float64).eps)

    xi[t]=c[t]*temp
```

$$\xi_{tij} = P(s_t = i, s_{t+1} = j \mid \mathbf{O}, \lambda) = \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j) b_{o_{t+1} j}}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} \beta_{t+1}(j) b_{o_{t+1} j}}$$

原始 $$= \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j) b_{o_{t+1} j}}{\sum_{i=1}^{N} \alpha_t(i) \beta_t(i)}$$

修正 $$\xi_{tij} = \mathrm{c}_t(i) \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j) b_{o_{t+1} j}}{\sum_{i=1}^{N} \alpha_t(i) \beta_t(i)}$$

```
# 计算每个样本在每个状态的每个mix上的概率
gamma_mix = np.zeros([N_samples,N_state,N_mix])

for t in range(N_samples):
    # 样本在状态上的概率
    pab = alpha[t]*beta[t]  # S

    sum_pab = np.sum(pab)
    if sum_pab==0:
        sum_pab = np.finfo(np.float64).eps

    for s in range(N_state):
        prob = B_map_mix[t,s]    # M
        sum_prob = np.sum(prob)

        if sum_prob ==0:
            sum_prob = np.finfo(np.float64).eps

        temp = pab[s]/sum_pab # 1
        prob = prob/sum_prob# M
        gamma_mix[t,s,:] = temp*prob #M

return c,alpha,beta,xi,gamma_mix
```

t时刻样本属于每个状态的概率

t 时刻样本属于状态s的每个成分的概率

正则保证和为1 （修改）

```python
def update_A(A,collect_xi):
    N_state = np.shape(A)[-1]
    new_A = A.copy()

    collect_xi = np.concatenate(collect_xi,axis=0)
    sum_xi = np.sum(collect_xi,axis=0)

    for i in range(N_state):
        for j in range(N_state):
            # 只对A[i,j]>0的部分参数进行更新
            if A[i,j]>0:
                nom = sum_xi[i,j]
                denom = np.sum(sum_xi[i])
                new_A[i,j] = nom/(denom+np.finfo(np.float64).eps)

    return new_A
```

对参数A 更新

只更新**非0**部分

```python
def update_GMM_in_States(model,train_datas,collect_gamma_mix):

    # 数据进行拼接
    train_datas = np.concatenate(train_datas,axis=0)
    collect_gamma_mix= np.concatenate(collect_gamma_mix,axis=0)

    # 获取数据的长度
    T,D = np.shape(train_datas)

    # 获取状态数N_state和每个状态中混合成分数N_mix
    N_state = len(model["S"])
    N_mix = np.shape(model["S"][0]["ws"])[0]
```

```python
    for s in range(N_state):
        nommean = np.zeros(D);
        nomvar  = np.zeros(D);
        for m in range(N_mix):
            # 每个样本属于状态s第m个成分的概率(权重)
            weight = collect_gamma_mix[:,s,m]
            weight = np.expand_dims(weight,axis=1)

            # 加权的均值
            nom_mean = np.sum(train_datas*weight, axis=0)

            # 加权的方差
            mu = np.expand_dims(model["S"][s]["mus"][m],axis=0)
            nom_var  = np.sum((train_datas - mu)*(train_datas - mu) *weight ,axis=0 )

            # 权重求和
            denom = np.sum(weight)
            if denom ==0:
                denom = np.finfo(np.float64).eps

            # 取平均 获得新的mu
            model["S"][s]["mus"][m] = nom_mean/denom

            # 取平均的 获得新的sigma
            sigma = nom_var/denom
            model["S"][s]["sigmas"][m] = sigma

            # 取平均 获得新的w
            nom_w = np.sum(weight)
            denom_w = np.sum(collect_gamma_mix[:,s,:]+np.finfo(np.float64).eps)
            model["S"][s]["ws"][m] = nom_w/denom_w

    return model
```

```python
# 训练数据是一个numpy的列表
def train_step_GMM_HMM(train_datas,model,collect_B_map,collect_B_map_mix):

    collect_gamma_mix = []
    collect_xi = []
    for data,B_map,B_map_mix in zip(train_datas,collect_B_map,collect_B_map_mix):
        c,alpha,beta,xi,gamma_mix=getparam(model,data,B_map,B_map_mix)
        collect_gamma_mix.append(gamma_mix)
        collect_xi.append(xi)

    new_A = update_A(model["A"],collect_xi)
    model = update_GMM_in_States(model,train_datas,collect_gamma_mix)
    model["A"] = new_A

    return model
```

GMM-HMM
一步更新
不更新pi

模型收敛评判：由于前向概率计算加入了正则项，所以采用**Viterbi**译码的概率来进行收敛判断

为避免序列过长，用log加替代乘

```python
# 维特比译码  为了避免数据过长的问题
# 这里用log 替代乘法
def decoder(model,observations,B_map):
    o = observations
    N_samples = np.shape(o)[0]
    N_state = np.shape(model["pi"])[0]

    pi = model['pi']
    log_pi = np.zeros(N_state)
    for i in range(N_state):
        if pi[i] == 0:
            log_pi[i] = -np.inf
        else:
            log_pi[i] = np.log(pi[i])

    A = model["A"]
    log_A = np.zeros([N_state,N_state])
    for i in range(N_state):
        for j in range(N_state):
            if A[i,j]==0:
                log_A[i,j] = -np.inf
            else:
                log_A[i,j] = np.log(A[i,j])
```

```python
    # 记录了从t-1 到 t时刻，状态i
    # 最可能从哪个状态（假设为j）转移来的
    psi = np.zeros([N_samples,N_state])

    # 从t-1 到 t 时刻状态 状态j到状态i的最大的转移概率
    delta = np.zeros([N_samples,N_state])

    # 初始化
    # delta[0] = model["pi"]*model["B"](model,o[0])
    delta[0] = log_pi + np.log(B_map[0])
    psi[0]=0

    # 递推填充 delta 与 psi
    for t in range(1,N_samples):
        for i in range(N_state):
            states_prev2current = delta[t-1] + log_A[:,i]
            delta[t][i] = np.max(states_prev2current)
            psi[t][i] = np.argmax(states_prev2current)

        # delta[t] = delta[t]*model["B"](model,o[t])
        delta[t] = delta[t]+ np.log(B_map[t])
    # 反向回溯寻找最佳路径
    path = np.zeros(N_samples)
    path[-1] = np.argmax(delta[-1])
    prob_max = np.max(delta[-1])

    for t in range(N_samples-2,-1,-1):
        path[t] = psi[t+1][int(path[t+1])]

    return prob_max, path
```

```
def compute_prob_viterbi(model,datas,collect_B_map):

    result = 0

    for o,B_map in zip(datas,collect_B_map):
        prob_max,_ = decoder(model,o,B_map)
        result = result+prob_max
    return result
```

计算一组数据的Viterbi概率和

GMM 训练

```
def train_GMM_HMM(train_datas,model,n_iteration):

    probs = np.zeros(n_iteration+1)
    collect_B_map = []
    collect_B_map_mix = []
    for datas in train_datas:
        B_map,B_map_mix = compute_B_map(datas,model)
        collect_B_map.append(B_map)
        collect_B_map_mix.append(B_map_mix)

    prob_old = compute_prob_viterbi(model,train_datas,collect_B_map)
    probs[0] = prob_old
    print("Prob_first",prob_old)
```

计算初始概率

```
for i in range(n_iteration):

    # 一步训练获取一个新的模型
    model_old = model.copy()
    model=train_step_GMM_HMM(train_datas,model,collect_B_map,collect_B_map_mix)

    # 重新计算map_B
    collect_B_map = []
    collect_B_map_mix = []
    for datas in train_datas:
        B_map,B_map_mix = compute_B_map(datas,model)
        collect_B_map.append(B_map)
        collect_B_map_mix.append(B_map_mix)

    prob_new = compute_prob_viterbi(model,train_datas,collect_B_map)
    probs[i+1] = prob_new

    print("it %d prob %f"%(i,prob_new))

    if i>2:
        if np.abs((probs[i+1]-probs[i])/probs[i+1])< 5e-4:
            break

    if np.isnan(prob_new):
        model = model_old
        break

return model
```

概率增长率过低
停止迭代

在新的 GMM-HMM代码下进行孤立词的识别

```
def init_hmm(collect_fea,N_state,N_mix):

    model_GMM_hmm = dict()

    # 初始 一定从 state 0 开始
    pi = np.zeros(N_state)
    pi[0] = 1
    model_GMM_hmm["pi"] = pi

    #  当前状态 转移概率0.5 下一状态 转移概率0.5
    #  进入最后一个状态后不再跳出
    A = np.zeros([N_state,N_state])
    for i in range(N_state-1):
        A[i,i] = 0.5
        A[i,i+1] = 0.5
    A[-1,-1] =1
    model_GMM_hmm["A"] = A

    feas = collect_fea
    len_feas = []
    for fea in feas:
        len_feas.append(np.shape(fea)[0])

    states = []
```

```
import librosa
import numpy as np
import os
from GMM_hmm import train_GMM_HMM,compute_B_map,decoder
from sklearn.cluster import KMeans
```

```
for s in range(N_state):
    print("STATE ---------------",s)
    sub_fea_collect = []
    # 初始化时 先为每个状态平均分配特征
    for fea,T in zip(feas,len_feas):

        T_s = int(T/N_state)*s
        T_e = (int(T/N_state))*(s+1)

        sub_fea_collect.append(fea[T_s:T_e])
    ws, mus, sigmas = gen_para_GMM(sub_fea_collect,N_mix)
    gmm = creat_GMM(mus,sigmas,ws)
    states.append(gmm)

model_GMM_hmm["S"] = states

return model_GMM_hmm
```

```
def creat_GMM(mus,sigmas,ws):
    gmm = dict()
    gmm['mus'] = mus
    gmm['sigmas'] = sigmas
    gmm['ws'] = ws
    return gmm
```

训练部分

```python
if __name__ == "__main__":

    models = []
    train_path = "train"
    for i in range(1,15):

        # 进入孤立词i所在的文件夹
        wav_path = os.path.join(train_path,str(i))
        collect_fea = []
        len_feas = []
        dirs = os.listdir(wav_path)
        for file in dirs:
            # 找到 .wav 文件并提取特征
            if file.split(".")[-1]=="wav":
                wav_file = os.path.join(wav_path,file)
                fea = extract_MFCC(wav_file)
                collect_fea.append(fea)

        # 模型参数初始化
        GMM_HMM_1 = init_hmm(collect_fea,N_state=4,N_mix=3)
        print("train model %d"%i)
        model = train_GMM_HMM(collect_fea,GMM_HMM_1,40)
        models.append(model)

    np.save("models.npy",models)
```

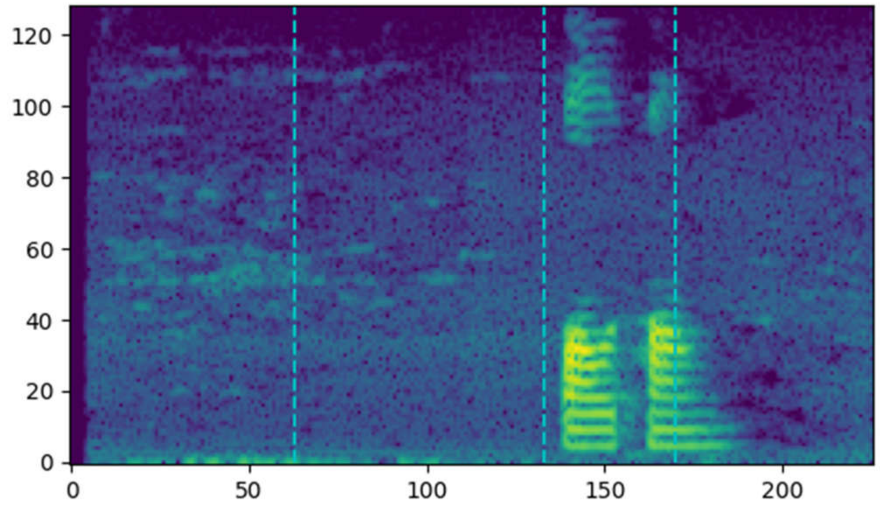
测试部分

```python
# 测试部分
test_dir ="test"
count = 0
models = np.load("models.npy",allow_pickle=True)

for i in range(98):
    # 读取wav文件
    wav_file = os.path.join(test_dir,str(i+1)+".wav")
    fea = extract_MFCC(wav_file)

    # 获取测试语音的标签，每条孤立词有7条测试
    lab_true = int(i//7)+1

    # 在每个模型上测试
    scores = []
    for m in range(14):
        model = models[m]
        B_map,_=compute_B_map(fea,model)
        prob_max,_ = decoder(model,fea,B_map)
        scores.append(prob_max)

    lab_det = np.argmax(scores)+1
    if lab_det == lab_true:
        count = count+1

    print("true lab  %d det lab %d"%(lab_true,lab_det))
print("decode  %.2f   "%(count*100/98))
```

解码测试:

```python
''' viterb 译码测试'''
# 加载模型
models = np.load("models.npy",allow_pickle=True)
model = models[1]
test_dir ="test"

# 读取测试 wav 并提取特征
# wav_file = os.path.join(test_dir,str(2)+".wav")
wav_file = "train\\2\\1.wav"
fea = extract_MFCC(wav_file)

# 进行viterbi译码
B_map,_=compute_B_map(fea,model)
prob_max,states = decoder(model,fea,B_map)
print(states)

# 读取音频文件并计算频谱
y,sr = librosa.load(wav_file,sr=8000)
S = librosa.stft(y,n_fft=256, hop_length=80, win_length=256)
S = np.abs(S)
Spec = librosa.amplitude_to_db(S,ref=np.max)

# 绘制谱图
fig, ax = plt.subplots()
ax.imshow(Spec,origin='lower')

# 找到状态变化的位置并画线
for i in range(1,len(states)):
    if states[i] != states[i-1]:
        plt.vlines(i-1, 0, 128, colors = "c", linestyles = "dashed")

plt.show()
```

```python
import librosa
import numpy as np
import os
from GMM_hmm import compute_B_map,decoder
from isolated_word_recognition import extract_MFCC
import matplotlib.pyplot as plt
```



获取频谱图

显示

找到状态变化的位置
画线显示