hmm_viterbi

May 1, 2019

[1]:

```
import numpy as np
import os
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
import librosa as ls
```

[2]:

```
zero_mfcc = []
for file in os.listdir('digits_speech/zero/'):
    if file.endswith(".wav"):
        path = os.path.join('digits_speech/zero/',file)
        signal,sr = ls.load(path ,sr=None,  duration=0.21)
        mfccs = ls.feature.mfcc(y=signal, sr=sr, n_mfcc=13, hop_length=int(0.
 ↪015*sr), n_fft=int(0.025*sr))
        zero_mfcc.append(mfccs.T)

seven_mfcc = []
for file in os.listdir('digits_speech/seven/'):
    if file.endswith(".wav"):
        path = os.path.join('digits_speech/seven/',file)
        signal,sr = ls.load(path ,sr=None,  duration=0.21)
        mfccs = ls.feature.mfcc(y=signal, sr=sr, n_mfcc=13, hop_length=int(0.
 ↪015*sr), n_fft=int(0.025*sr))
#          print(mfccs.T.shape)
        seven_mfcc.append(mfccs.T)

zero_mfcc = np.array(zero_mfcc)
seven_mfcc = np.array(seven_mfcc)
print(zero_mfcc.shape, seven_mfcc.shape)
```

```
/home/snehith/.local/lib/python3.5/site-packages/librosa/filters.py:284:
UserWarning: Empty filters detected in mel frequency basis. Some channels will
produce empty responses. Try increasing your sampling rate (and fmax) or
reducing n_mels.
  warnings.warn('Empty filters detected in mel frequency basis. '
```

```
(200, 15, 13) (200, 15, 13)
```

[3]:

```
temp = zero_mfcc - np.mean(zero_mfcc, axis = 0)
input1 = temp/np.std(zero_mfcc, axis = 0)

temp = seven_mfcc - np.mean(seven_mfcc, axis = 0)
input2 = temp/np.std(seven_mfcc, axis = 0)

in1_train,in1_test = train_test_split(input1, test_size=0.2)
in2_train,in2_test = train_test_split(input2, test_size=0.2)
print(in1_test.shape, in2_test.shape)
```

```
(40, 15, 13) (40, 15, 13)
```

[4]:

```
n_states = 5
m_gmm = 3
vect_len = 13
d = in1_train.shape[2]

phi = np.ones(n_states)/n_states
print(phi)

A = np.ones((n_states, n_states))/(n_states)
print(A)

w = np.random.uniform(size = (n_states,m_gmm))
w = np.transpose(np.transpose(w)/np.sum(w, axis = 1))
print(w)

mu = np.random.rand(n_states, m_gmm, vect_len)

co_var = [np.eye(vect_len, vect_len) for _ in range(n_states*m_gmm)]
co_var = np.array(co_var).reshape(n_states, m_gmm, vect_len, vect_len)
print(co_var.shape)
# print(co_var[4,2,:,:])
```

```
[0.2 0.2 0.2 0.2 0.2]
[[0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]]
[[0.33966772 0.60330084 0.05703145]
 [0.41159951 0.27824687 0.31015362]
 [0.40519277 0.24820557 0.34660166]
 [0.35770708 0.25810899 0.38418393]
 [0.45678863 0.30307822 0.24013315]]
(5, 3, 13, 13)
```

[5]:
```
def pdf(x, state):
    wt = w[state]
    mean = mu[state]
    var = co_var[state]

    pdf = 0

    for i in range(m_gmm):
        a = (np.sqrt((np.linalg.det(var[i]) * (2*np.pi)**len(x))))
        b = np.exp((-np.matmul(np.matmul(np.transpose(x-mean[i]) , np.
 ↪matrix(var[i]).I ), (x-mean[i]))/2))
        pdf = pdf + float(b/a)
    return pdf
```

[6]:
```
def viterbi(x):
    alpha = np.zeros((x.shape[0], n_states))
    shi = np.zeros((x.shape[0], n_states))

    for j in range(alpha.shape[1]):
        alpha[0][j] = phi[j] * pdf(x[0],j)
        shi[0][j] = 0

    for i in range(1,alpha.shape[0]):
        for j in range(alpha.shape[1]):
            alpha[i][j] = np.max(A[:,j].reshape(-1,) * alpha[i-1].reshape(-1,))␣
 ↪* pdf(x[i], j)
            shi[i][j] = np.argmax(A[:,j].reshape(-1,) * alpha[i-1].reshape(-1,))

    path = []
    p = np.max(alpha[-1])
    q = np.argmax(alpha[-1])
    path.append(q)

    for i in range(alpha.shape[0]-1,0,-1):
        q = shi[i][int(q)]
        path.append(q)

    path.reverse()
    path = np.array(path).astype(int)
    return alpha,path
```

[7]:
```
alp, path = viterbi(in1_train[0])
print('Best possible path by viterbi algorithm for 5 states')
print('Path :',  path+1)
```

```
Best possible path by viterbi algorithm for 5 states
Path : [5 3 3 5 5 3 3 3 3 4 3 3 3 2 2]
```