(1).

Consider a generative classification model with K-classes defined by prior probabilities $p(C_K) = \pi_K$ and class conditional densities $p(\phi/C_K)$

where ϕ is input feature vector.

$\{\phi_n, t_n\}$ for $n = 1, 2 .. N$

t_n : denotes a binary target vector of dimension K

with components $t_{nj} = \delta_{j,K}$ if input pattern ϕ_n belongs to class K.

$$\log p(\phi, t / C_K) = \log \prod_{n=1}^{N} \prod_{K=1}^{K} \left[p(\phi_n / C_K)\, p(C_K) \right]^{t_{nK}}.$$

$$= \log \prod_{n=1}^{N} \prod_{K=1}^{K} \left[p(\phi_n / C_K)\, \pi_K \right]^{t_{nK}}$$

$$= \sum_{n=1}^{N} \sum_{K=1}^{K} t_{nK} \left(\log p(\phi_n / C_K) + \log(\pi_K) \right)$$

To find Maximum Likelihood, with constraint $\sum_K \pi_K = 1$

$$L(\pi_K) = \ln p(\phi, t / C_K) + \lambda \left(\sum_{K=1}^{K} \pi_K - 1 \right).$$

$$\frac{d}{d\pi_K}(L(\pi_K)) = \frac{d}{d\pi_K}\left[\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log p(\phi_n / c_k) + t_{nk}\log \pi_k\right] + \frac{d}{d\pi_K}\lambda\left(\sum_{k=1}^{K}\pi_k - 1\right)$$

$$= \sum_{n=1}^{N} t_{nk}\cdot\left(\frac{1}{\pi_k}\right) + \lambda$$

We need to get solution $\frac{d}{d\pi_K}(L(\pi_K)) = 0$

$$\sum_{n=1}^{N}\frac{t_{nk}}{\pi_K} + \lambda = 0 \dots\dots \rightarrow ①$$

$$\pi_K = -\frac{1}{\lambda}\cdot\sum_{n=1}^{N} t_{nK}$$

$$\pi_K = -\frac{N_K}{\lambda} \dots\rightarrow ②$$

Multiply ① by π_K

$$\sum_{n=1}^{N}\frac{t_{nk}}{\pi_k}\cdot\pi_K + \lambda\pi_K = 0 \longrightarrow ③$$

for ③, sum it over by K

$$\sum_{k=1}^{K}\sum_{n=1}^{N}\left(\frac{t_{nk}}{\pi_k}\pi_k + \lambda\pi_k\right) = 0$$

$$\sum_{k=1}^{K} \sum_{n=1}^{N} t_{nk} + \lambda = 0$$

$$\lambda = -N \longrightarrow (4)$$

from (2), (4),

$$\boxed{\pi = \frac{N_k}{N}.}$$

②

ⓐ let x_i, $i=1,2..N$

y_i, $i=1,2,..N$

denotes the feature sequence corresponding to the source and output channel

Begins with linear model assumption.

$$y = Ax + b + c$$

$$y \in R^D, \quad x \in R^D.$$

$$A \in R^{D \times D}$$

$$b \in R^{D+1}$$

$$\epsilon \in N(0, \sigma^2 I)$$

$$P(y_n) \sim N(Ax+b, \sigma^2 I)$$

$$= \frac{1}{\sqrt{2\pi} \, |\sigma^2 I|^{1/2}} \exp\left(-\frac{(y_n - (Ax_n+b))^T (y_n - (Ax_n+b))}{2\sigma^2}\right)$$

Assuming data y_n is independent.

$$P(y) = \prod_{i=1}^{N} P(y_i)$$

$$= \prod_{i=1}^{N} \frac{1}{[2\pi \, |\sigma^2 I|^{1/2}]} \exp\left(\frac{-(y_i - (Ax_i+b))^T (y_i - (Ax_i+b))}{2\sigma^2}\right)$$

Now consider,

$$\log P(y) = \sum_{i=1}^{N} -D/2 \log 2\pi - 1/2 \log \sigma^{2D} - 1/2 \frac{(y_i - (Ax_i+b))^T (y_i - (Ax_i+b))}{\sigma^2}$$

$$= -\frac{DN}{2} \log 2\pi - 1/2\, 2ND \log \sigma - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - (Ax_i+b))^T (y_i - (Ax_i+b))$$

Differentiate bothsides with respect to σ^2

$$\frac{d}{d\sigma^2} (\log P(y)) = -\cancel{\frac{DN}{2} \log 2\pi} + \frac{ND}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{i=1}^{N} (y_i - (Ax_i+b))^T (y_i - (Ax_i+b)) = 0$$

$$\sigma^2 = \frac{1}{ND} \sum_{i=1}^{N} (y_i - (Ax_i+b))^T (y_i - (Ax_i+b))$$

differentiate respect to b.

$$\frac{d}{db} (\log P(y)) = -\frac{1}{2\sigma^2} \sum_{i=1}^{N} 2 (y_i - (Ax+b)) (-1) = 0$$

$$\sum_{i=1}^{N} y_i - (Ax_i+b) = Nb$$

$$b = 1/N \sum_{i=1}^{N} (y_i - Ax_i).$$

$$\log P(y) = -\frac{ND}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(Y - AX)^T(Y - AX)^T \rightarrow (3)$$

Where,

$Y_{D \times N}$ – each coloum is $y_i - b$.

$X_{D \times N}$ – each coloum is x_i.

Now differentiate (3) by A

$$\frac{d}{dA}(\log P(y)) = -\frac{\partial}{\partial A}\left(\frac{1}{2\sigma^2}(Y - AX)^T(Y - AX)\right)$$

$$= -\frac{1}{2\sigma^2}X^T(Y - XA)$$

$$X^TY = X^TXA$$

$$(X^TX)A = (X^TY)$$

$$A = (X^TX)^{-1}(X^TY)$$

3. **Implementing GMM** - A set of training and test examples of music and speech are provided.
$http://www.leap.ee.iisc.ac.in/sriram/teaching/MLSP21/assignments/speechMusicData.tar.gz$

Using these examples,

a Generate spectrogram features - Use the log magnitude spectrogram as before with a 64 component magnitude FFT (NFFT). In this case, the spectrogram will have dimension 32 times the number of frames (using 25 ms with a shift of 10 ms).

b Train two GMM models with K-means initialization (for each class) separately each with (i) 2 mixtures with diagonal covariance, (ii) 2 mixtures with full covariance and (iii) 5-mixture components with diagonal/full covariance respectively on this data. Plot the log-likelihood as a function of the EM iteration.

c Classify the test samples using the built classifiers and report the performance in terms of error rate (percentage of mis-classified samples) on the text data.

e Discuss the impact on the performance for different number of mixture components, diagonal versus full covariance ?

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import wave
import matplotlib.image as mpimg
import numpy.linalg as la
import os
import sys
import math
from pathlib import Path
```

In [2]:
```
# It's a function for the reading the audio file.

def readWavFile(filename):
    audio  = wave.open(filename,'r')
    sample_frequecny=audio.getframerate()
    total_samples=audio.getnframes()

    signal = audio.readframes(-1)
    signal = np.frombuffer(signal,'int16')
    signal = np.asarray(signal,dtype='double')
    audio_time=total_samples/sample_frequecny
    #print('sample frequency:',sample_frequecny)
    #print('total samples',total_samples)
    #print('total time of the audio:'+str(audio_time)+" seconds")
    audio.close()
    return signal, sample_frequecny,total_samples

# it's function for the plotting the audio.
def plotAudio(data,title='title'):
    fig, ax = plt.subplots()
    #ax.plot(t, s)
    ax.plot(data)
    ax.set(xlabel='samples', ylabel='Signal', title=title)
    ax.grid()
    fig.savefig(title+".png")
    plt.show()


# it is giving the window of the audio file.
def get_window(audio,window_size,hop_index,hop_length):
    lower=hop_index*hop_length
    upper=lower+window_size
    return audio[lower:upper]

# converting the sample_window to vector
def convert_sample_window_to_vector(window_sample):
    output=np.fft.fft(window_sample,64)
    return np.log(np.abs(output[0:32]))


def get_spectrogram_vector(audio_file_path,window_type,window_size,ho
p_length):

    audio1, sample_frequency, total_samples = readWavFile(audio_file_
path)
    total_hops=math.floor((total_samples-window_size)/hop_length+1)
    window=np.ones(window_size)
    if(window_type=='hamming'):
        window=np.hamming(window_size)
        #print('\n using hamming window')

    total_hops=math.floor((total_samples-window_size)/hop_length+1)
    spectrogram_vector=[]
    for hop_index in range(total_hops):
        sample_window=get_window(audio1, window_size, hop_index, hop_
length)
```

```
        sample_window=np.multiply(sample_window,window)
        feature_vector=convert_sample_window_to_vector(sample_window)
        spectrogram_vector.append(feature_vector)
    spectrogram_vector=np.array(spectrogram_vector)


    return spectrogram_vector

def whitening(X):
    X_mean=np.mean(X,axis=0)
    X_centred=X-X_mean
    #print(np.mean(X_centred))
    L,U=np.linalg.eigh((1/N)*(X_centred.T@X_centred))
    idx = np.argsort(-L)
    L = L[idx]
    U = U[:,idx]
    L_sqrt=np.diag(np.sqrt(1/L))
    transformed_data=X_centred@U
    Y=transformed_data@L_sqrt
    return Y,L,U

def plot_audio_spectrogram(audio_feature_vector,title):
    N=audio_feature_vector.shape[0]
    fig=plt.figure(figsize=(20,20))
    plt.imshow(clean_audio_feature_vector.T)
    plt.title(title)
    plt.xlabel('Hop index')
    plt.ylabel('Frequencies')
```

In [3]:
```
window_type='hamming'
window_size=400
hop_length=160
hop_index=10


music_path='speechMusicData/speech_music_classification/train/music/'
speech_path='speechMusicData/speech_music_classification/train/speec
h/'
```

Loading the training data:

In [4]:
```
import os

music_data=[]
music_labels=[]
speech_data=[]
speech_labels=[]
music_files_list=os.listdir(music_path)

for i in music_files_list:
    individual_file_path=music_path+i
    print(individual_file_path)
    audio_spectorgram_vector=get_spectrogram_vector(individual_file_p
ath,window_type,window_size,hop_length)
    music_data.append(audio_spectorgram_vector)
    music_labels.append(np.ones(shape=(2998,1)))
speech_files_list=os.listdir(speech_path)

for i in speech_files_list:
    individual_file_path=speech_path+i
    print(individual_file_path)
    audio_spectorgram_vector=get_spectrogram_vector(individual_file_p
ath,window_type,window_size,hop_length)
    speech_data.append(audio_spectorgram_vector)
    speech_labels.append(np.zeros(shape=(2998,1)))
```

```
speechMusicData/speech_music_classification/train/music/blues.wav
speechMusicData/speech_music_classification/train/music/glass1.wav
speechMusicData/speech_music_classification/train/music/hendrix.wav
speechMusicData/speech_music_classification/train/music/brahms.wav
speechMusicData/speech_music_classification/train/music/bmarsalis.wav
speechMusicData/speech_music_classification/train/music/loreena.wav
speechMusicData/speech_music_classification/train/music/cure.wav
speechMusicData/speech_music_classification/train/music/georose.wav
speechMusicData/speech_music_classification/train/music/guitar.wav
speechMusicData/speech_music_classification/train/music/ballad.wav
speechMusicData/speech_music_classification/train/music/caravan.wav
speechMusicData/speech_music_classification/train/music/copland.wav
speechMusicData/speech_music_classification/train/music/duke.wav
speechMusicData/speech_music_classification/train/music/bartok.wav
speechMusicData/speech_music_classification/train/music/gismonti.wav
speechMusicData/speech_music_classification/train/music/eguitar.wav
speechMusicData/speech_music_classification/train/music/echoes.wav
speechMusicData/speech_music_classification/train/music/chaka.wav
speechMusicData/speech_music_classification/train/music/corea1.wav
speechMusicData/speech_music_classification/train/music/bigband.wav
speechMusicData/speech_music_classification/train/music/classical1.wa
v
speechMusicData/speech_music_classification/train/music/debussy.wav
speechMusicData/speech_music_classification/train/music/glass.wav
speechMusicData/speech_music_classification/train/music/ipanema.wav
speechMusicData/speech_music_classification/train/music/birdland.wav
speechMusicData/speech_music_classification/train/music/bagpipe.wav
speechMusicData/speech_music_classification/train/music/copland2.wav
speechMusicData/speech_music_classification/train/music/led.wav
speechMusicData/speech_music_classification/train/music/jazz1.wav
speechMusicData/speech_music_classification/train/music/beatles.wav
speechMusicData/speech_music_classification/train/music/deedee.wav
speechMusicData/speech_music_classification/train/music/beat.wav
speechMusicData/speech_music_classification/train/music/gravity.wav
speechMusicData/speech_music_classification/train/music/classical.wav
speechMusicData/speech_music_classification/train/music/deedee1.wav
speechMusicData/speech_music_classification/train/music/corea.wav
speechMusicData/speech_music_classification/train/music/gravity2.wav
speechMusicData/speech_music_classification/train/music/canonaki.wav
speechMusicData/speech_music_classification/train/music/jazz.wav
speechMusicData/speech_music_classification/train/music/classical2.wa
v
speechMusicData/speech_music_classification/train/speech/comedy.wav
speechMusicData/speech_music_classification/train/speech/male.wav
speechMusicData/speech_music_classification/train/speech/my_voice.wav
speechMusicData/speech_music_classification/train/speech/emil.wav
speechMusicData/speech_music_classification/train/speech/conversion.w
av
speechMusicData/speech_music_classification/train/speech/georg.wav
speechMusicData/speech_music_classification/train/speech/china.wav
speechMusicData/speech_music_classification/train/speech/kedar.wav
speechMusicData/speech_music_classification/train/speech/allison.wav
speechMusicData/speech_music_classification/train/speech/nether.wav
speechMusicData/speech_music_classification/train/speech/news1.wav
speechMusicData/speech_music_classification/train/speech/greek.wav
speechMusicData/speech_music_classification/train/speech/diamond.wav
speechMusicData/speech_music_classification/train/speech/dialogue.wav
```

```
speechMusicData/speech_music_classification/train/speech/news2.wav
speechMusicData/speech_music_classification/train/speech/geography.wa
v
speechMusicData/speech_music_classification/train/speech/greek1.wav
speechMusicData/speech_music_classification/train/speech/lena.wav
speechMusicData/speech_music_classification/train/speech/dialogue2.wa
v
speechMusicData/speech_music_classification/train/speech/india.wav
speechMusicData/speech_music_classification/train/speech/fem_rock.wav
speechMusicData/speech_music_classification/train/speech/dialogue1.wa
v
speechMusicData/speech_music_classification/train/speech/austria.wav
speechMusicData/speech_music_classification/train/speech/god.wav
speechMusicData/speech_music_classification/train/speech/geography1.w
av
speechMusicData/speech_music_classification/train/speech/female.wav
speechMusicData/speech_music_classification/train/speech/bathroom1.wa
v
speechMusicData/speech_music_classification/train/speech/jvoice.wav
speechMusicData/speech_music_classification/train/speech/danie.wav
speechMusicData/speech_music_classification/train/speech/amal.wav
speechMusicData/speech_music_classification/train/speech/charles.wav
speechMusicData/speech_music_classification/train/speech/fire.wav
speechMusicData/speech_music_classification/train/speech/acomic.wav
speechMusicData/speech_music_classification/train/speech/chant.wav
speechMusicData/speech_music_classification/train/speech/danie1.wav
speechMusicData/speech_music_classification/train/speech/comedy1.wav
speechMusicData/speech_music_classification/train/speech/acomic2.wav
speechMusicData/speech_music_classification/train/speech/kid.wav
speechMusicData/speech_music_classification/train/speech/jony.wav
speechMusicData/speech_music_classification/train/speech/ellhnika.wav
```

In the following code, we loaded the training and testing data separately

In [5]:
```
# X: vector representation of the music.
# X_labels:     true labels of the features.

music_X=np.vstack(music_data)
music_X_labels=np.vstack(music_labels)

speech_X=np.vstack(speech_data)
speech_X_labels=np.vstack(speech_labels)
```

In [6]:
```
from sklearn.cluster import KMeans

def init_parameters_for_GMM(X,no_components):

    # we got the means initialization from the k-means......
    kmeans = KMeans(n_clusters=no_components, random_state=0).fit(X)
    labels=kmeans.labels_
    inital_means=kmeans.cluster_centers_

    #initalizing the cov matrices
    gmm_cov_matrix=[]
    for index in range(no_components):
        points_i=X[np.where(labels==index)]
        no_points_i=points_i.shape[0]
        centred_data=points_i-inital_means[index,:]

        cov_matrix_i=np.zeros(shape=(32,32),dtype='float64')


        for i in centred_data:
            temp=np.outer(i,i)
            cov_matrix_i=cov_matrix_i+temp

        cov_matrix_i=(1/no_points_i)*cov_matrix_i
        gmm_cov_matrix.append(cov_matrix_i)
    gmm_cov_matrix=np.array(gmm_cov_matrix)

    #init the probabilites
    theta=1/no_components*np.ones(shape=(no_components,1))



    return labels,inital_means,gmm_cov_matrix,theta

# Probability Distribution Function of the multi-variable normal dist
rubution.
def multivariate_normal(X, mean_vector, covariance_matrix):
    D=X.shape[0]
    exponent=0.5*(X-mean_vector).T@np.linalg.inv(covariance_matrix)@(
X-mean_vector)
    denominator=((2*np.pi)**D)*np.linalg.det(covariance_matrix)
    denominator=np.sqrt(denominator)
    value=np.exp(-exponent)/denominator
    return value

# we are using this function for random  sampling the data.
# we used this for loading some part of data for training the model.

def random_sampling(X,X_labels,total_no_samples):
    list_indices=list(range(total_no_samples))
    np.random.shuffle(list_indices)
    return X[list_indices],X_labels[list_indices]
```

In [7]:
```
# we are using the following function for calculating the log-likelih
ood sum.
def get_log_likelihood(X,gmm_means,gmm_cov_matrix,theta):
    log_likelihood=0
    total_no_points=X.shape[0]
    no_components=gmm_means.shape[0]

    #print('total_no_points',total_no_points,'no_components',no_compo
nents)
    for i in range(total_no_points):
        mixed_guassian_sum=0
        for cluster_yi in range(no_components):
            likelihood=multivariate_normal(X[i],gmm_means[cluster_yi
],gmm_cov_matrix[cluster_yi])
            mixed_guassian_sum=mixed_guassian_sum+likelihood*theta[cl
uster_yi]
        log_likelihood=log_likelihood+np.log(mixed_guassian_sum)
    return log_likelihood
```

In [8]:
```
# It returns the updated covariance matrix.
# cov_matrix_type=='DIAG': it reurns the diagonal covariance matirx.
# cov_matrix_type=='FULL': it reurns the FULL covariance matirx.
# cov_matrix_type=='IDENTITY': it reurns the identity matrix as covar
iance matirx.


def get_updated_cov_matrix(X,r,means_updated,cov_matrix_type):
    #print("updating the cov matrix")
    total_no_points=X.shape[0]
    D=X.shape[1]
    no_components=r.shape[1]

    # updating the covariance matrices..................
    gmm_cov_matrix_updated=np.zeros(shape=(no_components,D,D))

    if(cov_matrix_type=='FULL'):
        for cluster_yi in range(no_components):
            temp_cov_matix=np.zeros(shape=(32,32))
            for i in range(total_no_points):
                x_i=X[i]
                temp_cov_matix=temp_cov_matix+r[i,cluster_yi]*np.oute
r(x_i-means_updated[cluster_yi],x_i-means_updated[cluster_yi])
            temp_cov_matix=temp_cov_matix/np.sum(r[:,cluster_yi])
            gmm_cov_matrix_updated[cluster_yi]=temp_cov_matix
    elif(cov_matrix_type=='DIAG'):
        for cluster_yi in range(no_components):
            temp_cov_matix=np.zeros(shape=(32,32))
            for i in range(total_no_points):
                x_i=X[i]
                temp_cov_matix=temp_cov_matix+r[i,cluster_yi]*np.oute
r(x_i-means_updated[cluster_yi],x_i-means_updated[cluster_yi])
            temp_cov_matix=np.diag(np.diag(temp_cov_matix))
            temp_cov_matix=temp_cov_matix/np.sum(r[:,cluster_yi])
            gmm_cov_matrix_updated[cluster_yi]=temp_cov_matix
    elif(cov_matrix_type=='IDENTITY'):
        for cluster_yi in range(no_components):
            gmm_cov_matrix_updated[cluster_yi]=np.eye(D)

    return gmm_cov_matrix_updated
```

In [9]:
```
# It returns the parameters for fitting the given data.
# It fits the data using multi-variable normal distribution.

def Expectation_Maximization_Updated(X,no_components,no_iterations,co
v_matrix_type):

    # we are initilizing the parameters using K-means
    labels,inital_means,gmm_cov_matrix,theta=init_parameters_for_GMM(
X,no_components)

    #print('get_log_likelihood',get_log_likelihood(X,inital_means,gmm
_cov_matrix,theta))

    log_likelihood=[]

    for iteration in range(no_iterations):
        total_no_points=X.shape[0]
        D=X.shape[1]

        # storing the posterior-probabilities...
        r=np.zeros(shape=(total_no_points,no_components),dtype='float
64')

        cluster_yi=1
        for i in range(total_no_points):
            dec_sum=0
            for cluster_yi in range(no_components):
                temp=multivariate_normal(X[i],inital_means[cluster_yi
],gmm_cov_matrix[cluster_yi])
                r[i][cluster_yi]=temp*theta[cluster_yi]
                dec_sum=dec_sum+r[i][cluster_yi]
            r[i,:]=r[i,:]/dec_sum
        #print('Posterior calculation completed-------')

        # M-step updating the mean and Covariance-matirce
s....................

        # updating the probabilities........................

        theta_updated=np.zeros_like(theta)

        theta_updated=(1/total_no_points)*np.sum(r,axis=0)

        #updating the means........................
        means_updated=np.zeros_like(inital_means)

        for cluster_yi in range(no_components):
            #print("X:shape",X.shape)
            #print("r shape",r[:,index].shape)
            means_updated[cluster_yi]=X.T@r[:,cluster_yi]/np.sum(r[:,
cluster_yi])
            #print("Mean:"+str(i),means_updated[index])


        # updating the covariance matrices..................
        gmm_cov_matrix_updated=get_updated_cov_matrix(X,r,means_updat
```

```
ed,cov_matrix_type)

        inital_means=means_updated
        gmm_cov_matrix=gmm_cov_matrix_updated
        theta=theta_updated

        print("iteration: "+str(iteration)+" completed.")
        log_likelihood_value=get_log_likelihood(X,inital_means,gmm_co
v_matrix,theta)
        print('get_log_likelihood',log_likelihood_value)
        log_likelihood.append(log_likelihood_value)
    log_likelihood=np.array(log_likelihood)
    return inital_means,gmm_cov_matrix,theta,log_likelihood
```

In [10]:

```
# It returns the multi-variable normal distribution parameters for th
e speech and music.
# It is using the Expectation-Maximization Algorithm for modeling the
music and speech.

def GMM_Music_Speech_Model(music_data,speech_data,no_components,no_it
erations,cov_matrix_type):
    music_X,music_X_labels=music_data
    speech_X,speech_X_labels=speech_data

    gmm_music_parameters=Expectation_Maximization_Updated(music_X,no_
components,no_iterations,cov_matrix_type)
    gmm_speech_parameters=Expectation_Maximization_Updated(speech_X,n
o_components,no_iterations,cov_matrix_type)

    return gmm_music_parameters,gmm_speech_parameters
```

In [52]:

```
# it is plotting log-likelihood sum of the EM-Algorithm.
# it visualizes the algorithm progress.

def plot_log_likelihood_sum(music_log_likelihood_sum,speech_log_likel
ihood_sum):
    fig=plt.figure(figsize=(10,5))

    plot=fig.add_subplot(1,2,1)
    plt.plot(music_log_likelihood_sum,'r')
    plot.set_title('loglikelihood_music')

    plot=fig.add_subplot(1,2,2)
    plot.set_title('loglikelihood_speech')
    plt.plot(speech_log_likelihood_sum,'g')
```

In [26]:
```
# it loads the test data as spectrogram_vector of audio file and it's label.

def load_test_data(test_data_path):
    test_data=[]
    test_labels=[]
    test_files_list=os.listdir(test_data_path)
    for i in test_files_list:
        individual_file_path=test_data_path+i
        file_type=Path(individual_file_path).parts[-1].split('_')[0]
        audio_spectorgram_vector=get_spectrogram_vector(individual_file_path,window_type,window_size,hop_length)
        test_data.append(audio_spectorgram_vector)
        no_window_samples=audio_spectorgram_vector.shape[0]
        if(file_type=='music'):
            test_labels.append(1)
        elif(file_type=='speech'):
            test_labels.append(0)

    test_data=np.array(test_data)
    test_labels=np.array(test_labels)
    return test_data,test_labels
```

In [27]:
```
# it returns the likelihood of the X.

def get_likelihood(test_X,gmm_parameters):
    gmm_means_01,gmm_cov_matrix_01,theta_01,log_likelihood_01=gmm_parameters
    total_samples=test_X.shape[0]
    no_components=theta_01.shape[0]


    likelihood_table=np.zeros(shape=(total_samples,no_components))

    #print(total_samples,no_components)

    for i in range(total_samples):
        for cluster_yi in range(no_components):
            x_i=test_X[i]
            likelihood_table[i][cluster_yi]=multivariate_normal(x_i,gmm_means_01[cluster_yi],gmm_cov_matrix_01[cluster_yi])

    likelihood_table=likelihood_table@theta_01
    #print(likelihood_table.shape)
    return likelihood_table.reshape(-1,1)
```

Since, the data is very heavy we are traing on the 10000 samples. We are taking 10000 random samples from trainging data.

In [28]:
```
# it is a function for classifing the audio file.

def music_speech_classifier(audio_feature_vector,gmm_music_parameters
,gmm_speech_parameters):

    total_samples=audio_feature_vector.shape[0]
    music_likelihood=get_likelihood(audio_feature_vector,gmm_music_pa
rameters)
    speech_likelihood=get_likelihood(audio_feature_vector,gmm_speech_
parameters)

    temp=np.hstack([speech_likelihood,music_likelihood])

    calculated_labels=np.argmax(temp, axis=1)

    music_file_prob=np.where(calculated_labels==1)[0].shape[0]
    speech_file_prob=np.where(calculated_labels==0)[0].shape[0]

    if(music_file_prob>speech_file_prob):
        return 1
    else:
        return 0
```

In [33]:
```
from pathlib import Path
test_data_path='speechMusicData/speech_music_classification/test/'

# It is function for calculating testing accuracy.
def test_accuracy(test_data_path,gmm_music_parameters_01,gmm_speech_p
arameters_01):
    test_data,test_labels=load_test_data(test_data_path)
    total_samples=test_data.shape[0]
    correct=0
    for i in range(test_data.shape[0]):
        #print(test_data[i].shape)
        result=music_speech_classifier(test_data[i],gmm_music_paramet
ers_01,gmm_speech_parameters_01)
        if(result==test_labels[i]):
            correct=correct+1
    return (correct/total_samples)*100
```

In [11]:
```
random_music_data=random_sampling(music_X,music_X_labels,10000)
random_speech_data=random_sampling(speech_X,speech_X_labels,10000)

#gmm_music_parameters_01=Expectation_Maximization_Updated(random_musi
c_X,no_components=2,no_iterations=30,cov_matrix_type='FULL')
#gmm_speech_parameters_01=Expectation_Maximization_Updated(random_spe
ech_X,no_components=2,no_iterations=30,cov_matrix_type='FULL')
```

In [ ]:

2-GMM FULL-COVARIANCE MATRIX:

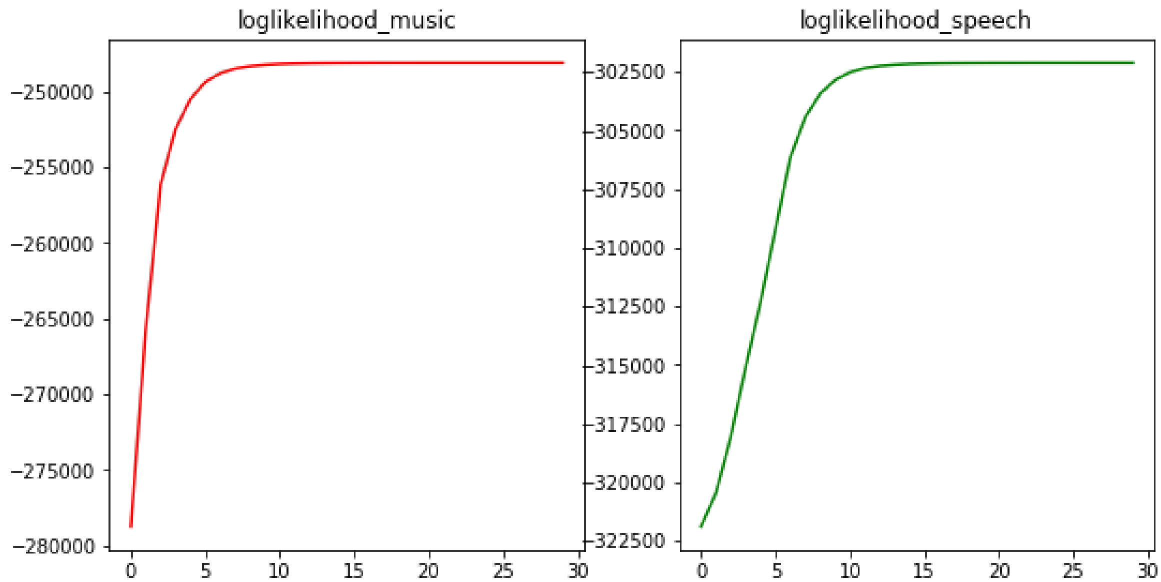
In [49]:
```
gmm_music_parameters_01,gmm_speech_parameters_01=GMM_Music_Speech_Mod
el(random_music_data,

random_speech_data,2,30,'FULL')
```

```
iteration: 0 completed.
get_log_likelihood -278732.17293927865
iteration: 1 completed.
get_log_likelihood -265725.9096942995
iteration: 2 completed.
get_log_likelihood -256133.93254310443
iteration: 3 completed.
get_log_likelihood -252462.64997190257
iteration: 4 completed.
get_log_likelihood -250478.23290849908
iteration: 5 completed.
get_log_likelihood -249348.20826217535
iteration: 6 completed.
get_log_likelihood -248769.44193708623
iteration: 7 completed.
get_log_likelihood -248461.26995460235
iteration: 8 completed.
get_log_likelihood -248304.13950146016
iteration: 9 completed.
get_log_likelihood -248214.6810905347
iteration: 10 completed.
get_log_likelihood -248158.78922889603
iteration: 11 completed.
get_log_likelihood -248126.92575016577
iteration: 12 completed.
get_log_likelihood -248109.214785677
iteration: 13 completed.
get_log_likelihood -248098.81927648935
iteration: 14 completed.
get_log_likelihood -248092.36410768345
iteration: 15 completed.
get_log_likelihood -248088.33201457426
iteration: 16 completed.
get_log_likelihood -248085.87276841013
iteration: 17 completed.
get_log_likelihood -248084.3917501392
iteration: 18 completed.
get_log_likelihood -248083.49866408613
iteration: 19 completed.
get_log_likelihood -248082.95679759272
iteration: 20 completed.
get_log_likelihood -248082.6257189205
iteration: 21 completed.
get_log_likelihood -248082.42206448346
iteration: 22 completed.
get_log_likelihood -248082.29602166382
iteration: 23 completed.
get_log_likelihood -248082.21758722488
iteration: 24 completed.
get_log_likelihood -248082.16854691913
iteration: 25 completed.
get_log_likelihood -248082.13776097036
iteration: 26 completed.
get_log_likelihood -248082.1183692359
iteration: 27 completed.
get_log_likelihood -248082.10612063936
iteration: 28 completed.
```

```
get_log_likelihood -248082.09836644348
iteration: 29 completed.
get_log_likelihood -248082.0934485424
iteration: 0 completed.
get_log_likelihood -321894.0405165091
iteration: 1 completed.
get_log_likelihood -320464.8092149584
iteration: 2 completed.
get_log_likelihood -318070.7949739581
iteration: 3 completed.
get_log_likelihood -315109.4371584911
iteration: 4 completed.
get_log_likelihood -312319.71929922974
iteration: 5 completed.
get_log_likelihood -309213.60859645664
iteration: 6 completed.
get_log_likelihood -306148.6875654243
iteration: 7 completed.
get_log_likelihood -304433.9861532711
iteration: 8 completed.
get_log_likelihood -303430.7811943281
iteration: 9 completed.
get_log_likelihood -302847.0837786264
iteration: 10 completed.
get_log_likelihood -302516.8114180201
iteration: 11 completed.
get_log_likelihood -302339.2385209815
iteration: 12 completed.
get_log_likelihood -302249.97481449967
iteration: 13 completed.
get_log_likelihood -302197.92457554647
iteration: 14 completed.
get_log_likelihood -302164.65737600054
iteration: 15 completed.
get_log_likelihood -302144.89088774583
iteration: 16 completed.
get_log_likelihood -302133.8191825564
iteration: 17 completed.
get_log_likelihood -302127.2909057474
iteration: 18 completed.
get_log_likelihood -302123.4069605828
iteration: 19 completed.
get_log_likelihood -302121.01926028304
iteration: 20 completed.
get_log_likelihood -302119.4823871254
iteration: 21 completed.
get_log_likelihood -302118.45902350167
iteration: 22 completed.
get_log_likelihood -302117.7620202922
iteration: 23 completed.
get_log_likelihood -302117.2875919748
iteration: 24 completed.
get_log_likelihood -302116.9727877618
iteration: 25 completed.
get_log_likelihood -302116.7700317732
iteration: 26 completed.
get_log_likelihood -302116.64132359077
```

```
iteration: 27 completed.
get_log_likelihood -302116.5595015711
iteration: 28 completed.
get_log_likelihood -302116.5070703436
iteration: 29 completed.
get_log_likelihood -302116.4732091126
```

In [53]:
```
plot_log_likelihood_sum(gmm_music_parameters_01[3],gmm_speech_paramet
ers_01[3])
```



ACCURACY TESTING

In [51]:
```
test_accuracy_01=test_accuracy(test_data_path,gmm_music_parameters_01
,gmm_speech_parameters_01)
print('test_accuracy',test_accuracy_01)
```

```
test_accuracy 87.5
```

2-GMM DIAGONAL COVARIANCE MATRIX:

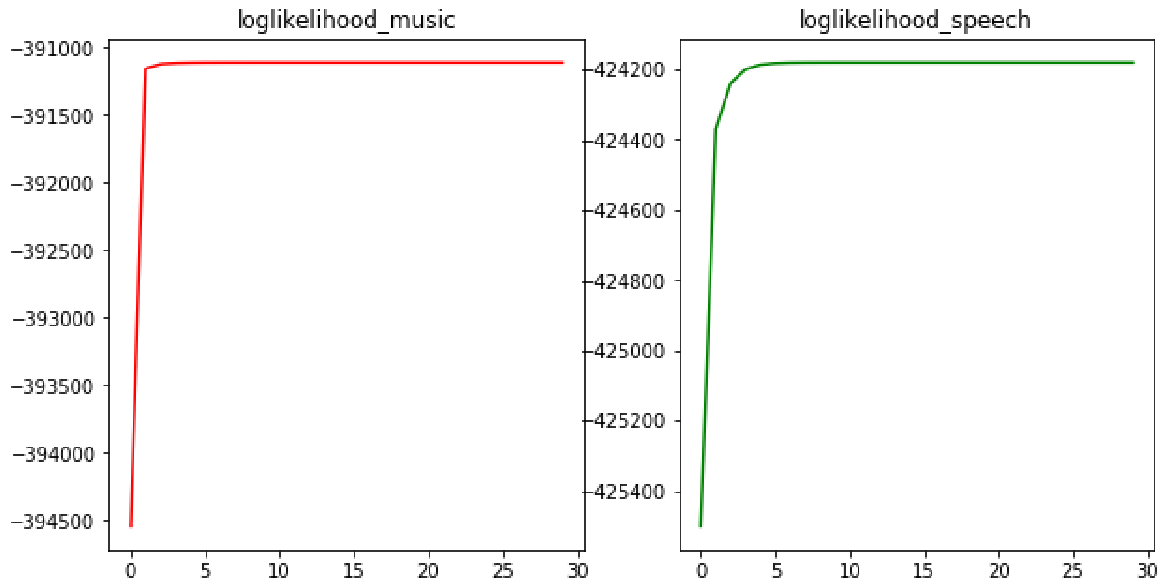
In [36]:
```
gmm_music_parameters_02,gmm_speech_parameters_02=GMM_Music_Speech_Mod
el(random_music_data,

random_speech_data,2,30,'DIAG')
```

```
iteration: 0 completed.
get_log_likelihood -394543.28009817615
iteration: 1 completed.
get_log_likelihood -391157.195646989
iteration: 2 completed.
get_log_likelihood -391122.3243252123
iteration: 3 completed.
get_log_likelihood -391115.2625771932
iteration: 4 completed.
get_log_likelihood -391112.6216839998
iteration: 5 completed.
get_log_likelihood -391111.5907472876
iteration: 6 completed.
get_log_likelihood -391111.1865351656
iteration: 7 completed.
get_log_likelihood -391111.02772524365
iteration: 8 completed.
get_log_likelihood -391110.96523753676
iteration: 9 completed.
get_log_likelihood -391110.94062440866
iteration: 10 completed.
get_log_likelihood -391110.93092278787
iteration: 11 completed.
get_log_likelihood -391110.92709699133
iteration: 12 completed.
get_log_likelihood -391110.92558786256
iteration: 13 completed.
get_log_likelihood -391110.92499245744
iteration: 14 completed.
get_log_likelihood -391110.92475751945
iteration: 15 completed.
get_log_likelihood -391110.9246648094
iteration: 16 completed.
get_log_likelihood -391110.9246282232
iteration: 17 completed.
get_log_likelihood -391110.9246137848
iteration: 18 completed.
get_log_likelihood -391110.92460808635
iteration: 19 completed.
get_log_likelihood -391110.924605839
iteration: 20 completed.
get_log_likelihood -391110.9246049504
iteration: 21 completed.
get_log_likelihood -391110.92460459966
iteration: 22 completed.
get_log_likelihood -391110.9246044634
iteration: 23 completed.
get_log_likelihood -391110.92460440856
iteration: 24 completed.
get_log_likelihood -391110.92460438714
iteration: 25 completed.
get_log_likelihood -391110.9246043774
iteration: 26 completed.
get_log_likelihood -391110.9246043747
iteration: 27 completed.
get_log_likelihood -391110.9246043732
iteration: 28 completed.
```

```
get_log_likelihood -391110.92460437276
iteration: 29 completed.
get_log_likelihood -391110.9246043749
iteration: 0 completed.
get_log_likelihood -425499.9064096202
iteration: 1 completed.
get_log_likelihood -424369.3632476791
iteration: 2 completed.
get_log_likelihood -424239.7561282704
iteration: 3 completed.
get_log_likelihood -424200.2011540571
iteration: 4 completed.
get_log_likelihood -424187.2452067859
iteration: 5 completed.
get_log_likelihood -424182.9945665313
iteration: 6 completed.
get_log_likelihood -424181.5871196866
iteration: 7 completed.
get_log_likelihood -424181.11569189845
iteration: 8 completed.
get_log_likelihood -424180.95642799325
iteration: 9 completed.
get_log_likelihood -424180.90232425963
iteration: 10 completed.
get_log_likelihood -424180.88388189184
iteration: 11 completed.
get_log_likelihood -424180.8775825604
iteration: 12 completed.
get_log_likelihood -424180.8754282925
iteration: 13 completed.
get_log_likelihood -424180.87469104247
iteration: 14 completed.
get_log_likelihood -424180.8744386278
iteration: 15 completed.
get_log_likelihood -424180.8743521875
iteration: 16 completed.
get_log_likelihood -424180.8743225817
iteration: 17 completed.
get_log_likelihood -424180.8743124365
iteration: 18 completed.
get_log_likelihood -424180.87430896563
iteration: 19 completed.
get_log_likelihood -424180.8743077742
iteration: 20 completed.
get_log_likelihood -424180.8743073677
iteration: 21 completed.
get_log_likelihood -424180.8743072274
iteration: 22 completed.
get_log_likelihood -424180.8743071816
iteration: 23 completed.
get_log_likelihood -424180.87430716277
iteration: 24 completed.
get_log_likelihood -424180.874307158
iteration: 25 completed.
get_log_likelihood -424180.87430715485
iteration: 26 completed.
get_log_likelihood -424180.87430715584
```

```
iteration: 27 completed.
get_log_likelihood -424180.87430715474
iteration: 28 completed.
get_log_likelihood -424180.8743071526
iteration: 29 completed.
get_log_likelihood -424180.874307154
```

In [54]:
```
plot_log_likelihood_sum(gmm_music_parameters_02[3],gmm_speech_parameters_02[3])
```



In [38]:
```
test_accuracy_02=test_accuracy(test_data_path,gmm_music_parameters_02,gmm_speech_parameters_02)
print('test_accuracy',test_accuracy_02)
```

```
test_accuracy 68.75
```

5-GMM MODEL WITH FULL COVARIANCE MATRIX

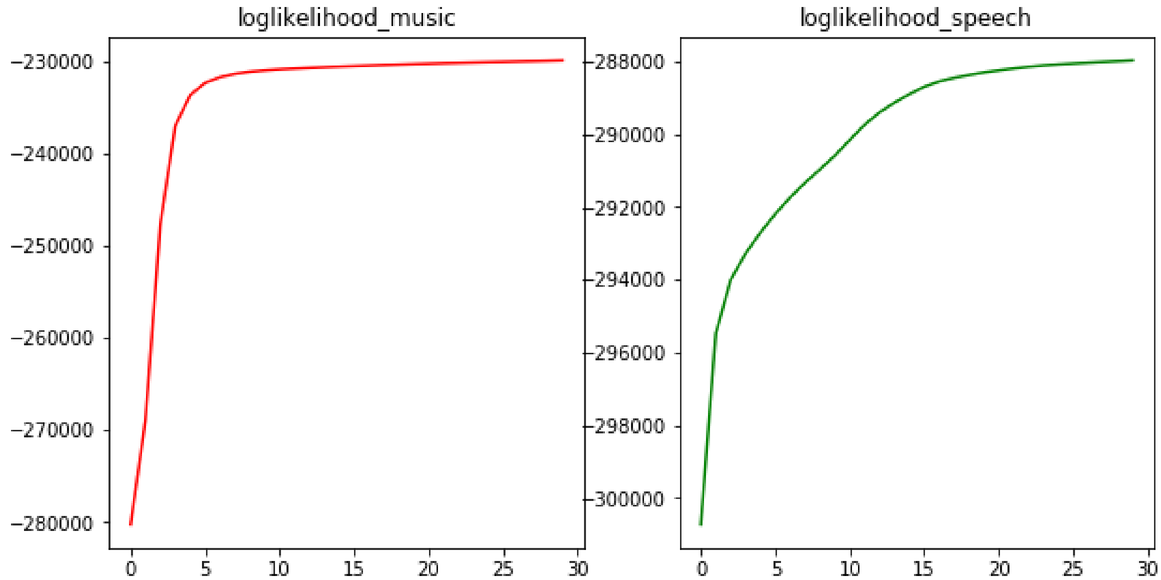
In [39]:
```
gmm_music_parameters_03,gmm_speech_parameters_03=GMM_Music_Speech_Mod
el(random_music_data,

random_speech_data,5,30,'FULL')
```

```
iteration: 0 completed.
get_log_likelihood -280228.339939392
iteration: 1 completed.
get_log_likelihood -268883.1306210619
iteration: 2 completed.
get_log_likelihood -247700.9107252477
iteration: 3 completed.
get_log_likelihood -237048.9438019702
iteration: 4 completed.
get_log_likelihood -233728.76287519903
iteration: 5 completed.
get_log_likelihood -232399.26122934092
iteration: 6 completed.
get_log_likelihood -231785.4871349807
iteration: 7 completed.
get_log_likelihood -231424.10538899162
iteration: 8 completed.
get_log_likelihood -231199.89209329846
iteration: 9 completed.
get_log_likelihood -231048.36999371366
iteration: 10 completed.
get_log_likelihood -230937.3584408024
iteration: 11 completed.
get_log_likelihood -230851.51605106442
iteration: 12 completed.
get_log_likelihood -230775.422145057
iteration: 13 completed.
get_log_likelihood -230705.2987528361
iteration: 14 completed.
get_log_likelihood -230641.53720040552
iteration: 15 completed.
get_log_likelihood -230583.50800167583
iteration: 16 completed.
get_log_likelihood -230528.74492473074
iteration: 17 completed.
get_log_likelihood -230475.24171494326
iteration: 18 completed.
get_log_likelihood -230422.99620957323
iteration: 19 completed.
get_log_likelihood -230372.45599070148
iteration: 20 completed.
get_log_likelihood -230323.32292351167
iteration: 21 completed.
get_log_likelihood -230278.3850524837
iteration: 22 completed.
get_log_likelihood -230237.11654378986
iteration: 23 completed.
get_log_likelihood -230198.62969345896
iteration: 24 completed.
get_log_likelihood -230161.57275121706
iteration: 25 completed.
get_log_likelihood -230124.8650961687
iteration: 26 completed.
get_log_likelihood -230088.01513436876
iteration: 27 completed.
get_log_likelihood -230050.03372245483
iteration: 28 completed.
```

```
get_log_likelihood -230009.4590484542
iteration: 29 completed.
get_log_likelihood -229964.23934893686
iteration: 0 completed.
get_log_likelihood -300723.5372103198
iteration: 1 completed.
get_log_likelihood -295484.4357649119
iteration: 2 completed.
get_log_likelihood -294008.7245946615
iteration: 3 completed.
get_log_likelihood -293282.170025845
iteration: 4 completed.
get_log_likelihood -292703.25522424886
iteration: 5 completed.
get_log_likelihood -292190.1680225236
iteration: 6 completed.
get_log_likelihood -291733.1276617039
iteration: 7 completed.
get_log_likelihood -291333.1968325455
iteration: 8 completed.
get_log_likelihood -290966.0968676232
iteration: 9 completed.
get_log_likelihood -290586.662211468
iteration: 10 completed.
get_log_likelihood -290156.0986730427
iteration: 11 completed.
get_log_likelihood -289733.7072579169
iteration: 12 completed.
get_log_likelihood -289398.4616202991
iteration: 13 completed.
get_log_likelihood -289130.7043776092
iteration: 14 completed.
get_log_likelihood -288897.7401261486
iteration: 15 completed.
get_log_likelihood -288694.3308865663
iteration: 16 completed.
get_log_likelihood -288548.06836821634
iteration: 17 completed.
get_log_likelihood -288449.19131920196
iteration: 18 completed.
get_log_likelihood -288370.1827507823
iteration: 19 completed.
get_log_likelihood -288300.9755174801
iteration: 20 completed.
get_log_likelihood -288243.6087957016
iteration: 21 completed.
get_log_likelihood -288190.4514838985
iteration: 22 completed.
get_log_likelihood -288145.44814577606
iteration: 23 completed.
get_log_likelihood -288108.62086637435
iteration: 24 completed.
get_log_likelihood -288080.5735371088
iteration: 25 completed.
get_log_likelihood -288056.03333092417
iteration: 26 completed.
get_log_likelihood -288033.25621303293
```

```
iteration: 27 completed.
get_log_likelihood -288011.81118398806
iteration: 28 completed.
get_log_likelihood -287990.7177625386
iteration: 29 completed.
get_log_likelihood -287968.23929961637
```

In [55]:
```
plot_log_likelihood_sum(gmm_music_parameters_03[3],gmm_speech_paramet
ers_03[3])
```



In [41]:
```
test_accuracy_03=test_accuracy(test_data_path,gmm_music_parameters_03
,gmm_speech_parameters_03)
print('test_accuracy',test_accuracy_03)
```

```
test_accuracy 89.58333333333334
```

5-GMM MODEL WITH DIAGONAL COVARIANCE MATRIX

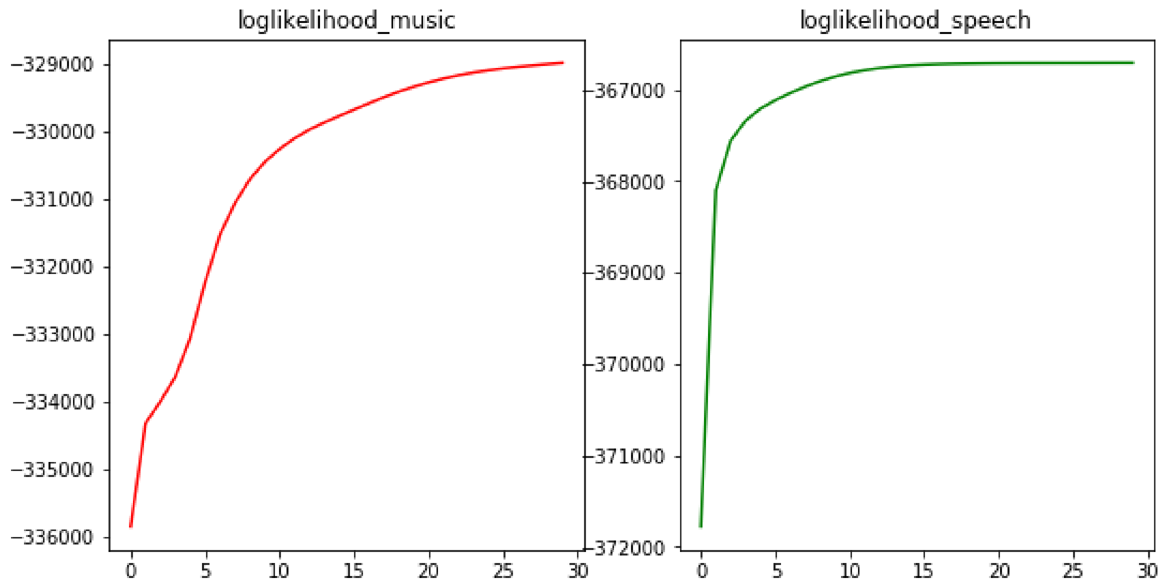
In [42]:
```
gmm_music_parameters_04,gmm_speech_parameters_04=GMM_Music_Speech_Model(random_music_data,

random_speech_data,5,30,'DIAG')
```

```
iteration: 0 completed.
get_log_likelihood -335855.1496583496
iteration: 1 completed.
get_log_likelihood -334325.6980620374
iteration: 2 completed.
get_log_likelihood -334001.7695476347
iteration: 3 completed.
get_log_likelihood -333637.2369822151
iteration: 4 completed.
get_log_likelihood -333060.7666442543
iteration: 5 completed.
get_log_likelihood -332234.639489045
iteration: 6 completed.
get_log_likelihood -331523.9105197213
iteration: 7 completed.
get_log_likelihood -331058.8203930257
iteration: 8 completed.
get_log_likelihood -330705.52171136416
iteration: 9 completed.
get_log_likelihood -330448.980461379
iteration: 10 completed.
get_log_likelihood -330255.3962262162
iteration: 11 completed.
get_log_likelihood -330096.269026693
iteration: 12 completed.
get_log_likelihood -329968.1432353451
iteration: 13 completed.
get_log_likelihood -329862.08838316955
iteration: 14 completed.
get_log_likelihood -329765.5329169255
iteration: 15 completed.
get_log_likelihood -329670.85403922235
iteration: 16 completed.
get_log_likelihood -329576.9251938532
iteration: 17 completed.
get_log_likelihood -329486.93442475464
iteration: 18 completed.
get_log_likelihood -329403.96892453986
iteration: 19 completed.
get_log_likelihood -329331.1091564101
iteration: 20 completed.
get_log_likelihood -329266.89965469507
iteration: 21 completed.
get_log_likelihood -329210.00784133584
iteration: 22 completed.
get_log_likelihood -329161.205849038
iteration: 23 completed.
get_log_likelihood -329120.490026755
iteration: 24 completed.
get_log_likelihood -329086.918678408
iteration: 25 completed.
get_log_likelihood -329059.15188455663
iteration: 26 completed.
get_log_likelihood -329035.59483267687
iteration: 27 completed.
get_log_likelihood -329014.70810713747
iteration: 28 completed.
```

```
get_log_likelihood -328995.19006511156
iteration: 29 completed.
get_log_likelihood -328975.96842448553
iteration: 0 completed.
get_log_likelihood -371776.72275558935
iteration: 1 completed.
get_log_likelihood -368105.71711339103
iteration: 2 completed.
get_log_likelihood -367562.6223557257
iteration: 3 completed.
get_log_likelihood -367344.0556851853
iteration: 4 completed.
get_log_likelihood -367210.21566235996
iteration: 5 completed.
get_log_likelihood -367117.787790671
iteration: 6 completed.
get_log_likelihood -367039.52876396774
iteration: 7 completed.
get_log_likelihood -366969.90856020316
iteration: 8 completed.
get_log_likelihood -366910.21433504455
iteration: 9 completed.
get_log_likelihood -366859.74083049194
iteration: 10 completed.
get_log_likelihood -366819.5800894544
iteration: 11 completed.
get_log_likelihood -366788.9773709582
iteration: 12 completed.
get_log_likelihood -366766.18922611064
iteration: 13 completed.
get_log_likelihood -366749.3404806363
iteration: 14 completed.
get_log_likelihood -366737.2610834871
iteration: 15 completed.
get_log_likelihood -366728.9947231629
iteration: 16 completed.
get_log_likelihood -366723.4020301401
iteration: 17 completed.
get_log_likelihood -366719.48973036715
iteration: 18 completed.
get_log_likelihood -366716.59746606974
iteration: 19 completed.
get_log_likelihood -366714.5162754095
iteration: 20 completed.
get_log_likelihood -366713.1555969128
iteration: 21 completed.
get_log_likelihood -366712.25705156557
iteration: 22 completed.
get_log_likelihood -366711.64799892716
iteration: 23 completed.
get_log_likelihood -366711.22722857073
iteration: 24 completed.
get_log_likelihood -366710.92265275976
iteration: 25 completed.
get_log_likelihood -366710.6638026054
iteration: 26 completed.
get_log_likelihood -366710.3217390009
```

```
iteration: 27 completed.
get_log_likelihood -366709.6897698352
iteration: 28 completed.
get_log_likelihood -366709.1231233748
iteration: 29 completed.
get_log_likelihood -366708.9187535744
```

In [56]:
```
plot_log_likelihood_sum(gmm_music_parameters_04[3],gmm_speech_paramet
ers_04[3])
```



In [48]:
```
test_accuracy_04=test_accuracy(test_data_path,gmm_music_parameters_04
,gmm_speech_parameters_04)
print('test_accuracy',test_accuracy_04)
```

```
test_accuracy 70.83333333333334
```

2-GMM with FULL-Covariance matrix test_accuracy 87.5

2-GMM with Diagonal-Covariance matrix test_accuracy 68.75

5-GMM with FULL-Covariance matrix test_accuracy 89.58333333333334

5-GMM with Diagonal-Covariance matrix test_accuracy 70.83333333333334

We observed the following thing from the above experiment.

The accuaracy of the 5-GMM is more than 2-GMM model.

The accuracy of the 2-GMM with full-covariance matrix is more than 2-GMM with Diagonal covariance matrix.

The accuracy of the 5-GMM with full-covariance matrix is more than 5-GMM with Diagonal covariance matrix.

Because, GMM-model with more guassians can model data in a better manner.

GMM-with full-covariance matrix is more powerful than Daiagonal covariance matrix.

In the diagonal covariance matrix, we are modeling features as independent.

In the full-covariance matrix, we are considering correlation between individual features.

That's why, GMM-model with full-covariance matrix is more powerful than GMM-with diagonal covariance matrix.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```python
# It returns all list of individual words of the document text.
def get_individual_words(document_text):
    full_text=document_text
    full_text=full_text.strip()
    full_text=full_text.replace("\n", ",")
    full_text=full_text.replace("\t", ",")
    full_text=full_text.replace(":", ",")
    full_text=full_text.replace(";", ",")
    full_text=full_text.replace(" ", ",")
    full_text=full_text.replace(" ", ",")
    full_text=full_text.replace("-", ",")
    full_text=full_text.replace("*", ",")
    full_text=full_text.replace("&", ",")
    full_text=full_text.replace("+", ",")
    full_text=full_text.replace("$", ",")
    full_text=full_text.replace("/", ",")
    full_text=full_text.replace("%", ",")

    full_text=full_text.lower()
    full_words=full_text.split(',')

    return full_words;

# it returns the documents and it's labels from corpus
def get_documents_labels(file_path):

    file1 = open(file_path, 'r')
    Lines = file1.readlines()

    documents=[]
    labels=[]
    for line in Lines:
        temp=line.split('\t')
        #print(temp)
        documents.append(temp[0])
        if(temp[1]=='0\n'):
            labels.append(0)
        elif(temp[1]=='1\n'):
            labels.append(1)
    return documents,labels

# It returns list of all the words of the documents with repeatation.
def get_all_words_of_documents(documents):
    full_text=''

    for i in documents:
        full_text=full_text+" "+i

    full_words=get_individual_words(full_text)

    for i in range(len(full_words)):
        full_words[i]=full_words[i].lower()


    full_words_updated=[]
    for individual_word in full_words:
```

```
        if(len(individual_word)!=0 and individual_word!=' '):
            full_words_updated.append(individual_word)
    return full_words_updated;
```

In [3]:
```
# It returns the term frequnecy of the term_t in the document_text
def tf(term_t,document_text):
    term_t=term_t.lower()
    full_words=get_individual_words(document_text)
    #print(full_words)
    term_t_count=0
    if(term_t in full_words):
        for i in full_words:
           if(i==term_t):
               term_t_count=term_t_count+1
    #print(term_t_count,len(full_words))
    return term_t_count/len(full_words)



import math

# it returns the tf-idf of the term_t of the document_text.
def tf_idf_single_term(term_t,document_text):
    term_t=term_t.lower()
    index=term_dictionary[term_t]
    term_freq=tf(term_t,document_text)
    document_freq=Document_Frequency[index]
    #print(term_freq,document_freq)
    if(document_freq>0):
        return term_freq*math.log(total_words_N/document_freq)
    else:
        return term_freq*math.log(total_words_N/1+document_freq)


# it returns the tf-idf of the document_text.
# it will be used to represent the document as a feature vector.
def tf_idf(document_text):
    feature_vector=np.zeros(total_unique_words_V,)
    for i in range(len(dictionary)):
        feature_vector[i]=tf_idf_single_term(dictionary[i],document_t
ext)
    return feature_vector
```

Corpus reading

In [4]:
```
file_path='movie_reviews/movieReviews1000.txt'

# reading a documents and labels from the corpus (text file)
documents,labels=get_documents_labels(file_path)

# full words of all the documents with repeatation.
full_words=get_all_words_of_documents(documents)

dictionary = list(set(full_words))
dictionary.sort()
```

In [5]:
```
total_unique_words_V=len(dictionary)
total_words_N=len(full_words)
total_documents_D=len(documents)
```

Corpus Details:

In [6]:
```
print('total no of documents:',total_documents_D)
print('total no of words in the documents(N):',total_words_N)
print('total no of unique words(V):',total_unique_words_V)
```

```
total no of documents: 1000
total no of words in the documents(N): 14436
total no of unique words(V): 3139
```

making a term dictionary

In [7]:
```
#making a term dictionary

term_dictionary={}
for i in range(len(dictionary)):
    term_dictionary[dictionary[i]]=i
```

print term-frequency of the slow word in the following sentence.

In [8]:
```
#print term-frequency of the slow word in the following sentence.

print(documents[0])
print("term-frequency of word slow:",tf("slow",documents[0]))
```

```
A very very very slow-moving aimless movie about a distressed driftin
g young man
term-frequency of word slow: 0.07142857142857142
```

Term Frequency Calculation Table:

Term-Frequency(t,d): $\frac{Totalnooftimestoccuredindocumentt}{Totalnoofwordsinthedocumentt}$

In [9]:
```
Term_Frequecy=np.zeros((total_unique_words_V,total_documents_D),dtype
='float64')

for i in range(len(dictionary)):
    for j in range(len(documents)):
        Term_Frequecy[i][j]=tf(dictionary[i],documents[j])

print("Term_Frequecy calculation was completed---------------------")
```

```
Term_Frequecy calculation was completed---------------------
```

Document Frequecny calculation:

$Document - Frequency(t)$=*total no of documents containg t*

In [10]:
```
#document frequecny calculations
# execution is only once for code..............


Document_Frequency=np.zeros(total_unique_words_V)

for i in range(len(dictionary)):
    temp=Term_Frequecy[i,:]
    document_indices=np.where(temp>0)
    document_indices=np.array(document_indices)
    Document_Frequency[i]=document_indices.shape[1]

print("Document Frequency calculation was completed------------------
---")
```

```
Document Frequency calculation was completed---------------------
```

tf-idf of document calculation:

In [11]:
```
document_id=np.random.randint(1,1000)
document_vector=tf_idf(documents[document_id])

print('Document text:\n',documents[document_id])
print('tf-idf array:\n',document_vector )

print("Following elements of document vectors are greater than zero:
\n")
indices=np.where(document_vector>0)
print("index-----tf-idf")
for i  in indices[0]:
    print(i,"-----",document_vector[i])
```

```
Document text:
 Think of the film being like a dream
tf-idf array:
 [0. 0. 0. ... 0. 0. 0.]
Following elements of document vectors are greater than zero:

index-----tf-idf
44 ----- 0.4689350322977786
284 ----- 0.8865717144968348
811 ----- 1.1971850457203348
1045 ----- 0.5691999806145503
1616 ----- 0.7241613414805522
1897 ----- 0.48132907727193525
2712 ----- 0.41763884694487186
2732 ----- 0.816619741004907
```

Representing the Corpus into feature vector:

We are representing the corpus as a feature vector.

make a feature vector for every document and form a array with these feature vectors.

In [12]:
```
document_vector=[]

for i in documents:
    document_vector.append(tf_idf(i))

document_vector=np.array(document_vector)
document_vector.shape
```

Out[12]:
```
(1000, 3139)
```

Since, the number of the documents is:1000 and number of the unique words is: 3139.

Corpus will be represented as a array of : 1000×3139 .

In [14]:
```
def higher_dimension_PCA(X):
    # Data Centering
    X_mean=np.mean(X,axis=0)
    X_centered=X-X_mean

    #checking the mean of the centered Data.
    #print(np.mean(X_centered,axis=0))
    #print(np.sum(np.mean(X_centered,axis=0)))

    # we are calculating the eigen values of the X@XT for calculating
the eigen values of the XT@X

    temp_outer_product=(1/N)*(X_centered@X_centered.T)
    temp_eigen_values,temp_eigen_vectors=np.linalg.eigh(temp_outer_pr
oduct)

    idx = np.argsort(-temp_eigen_values)
    temp_eigen_values = temp_eigen_values[idx]
    temp_eigen_vectors = temp_eigen_vectors[:,idx]


    #print(temp_eigen_values.shape)
    #print(temp_eigen_vectors.shape)


    # For calculating the eigenvalues of the XT@X

    #print(temp_eigen_values)
    norms_of_eigen_vectors=np.sqrt(N*temp_eigen_values)
    eigen_vectors=X_centered.T@temp_eigen_vectors

    for i in range(eigen_vectors.shape[1]):
        eigen_vectors[:,i]=eigen_vectors[:,i]/norms_of_eigen_vectors[
i]

    print('Eigen vectors shape',eigen_vectors.shape)

    return eigen_vectors
```

In [13]:
```
def data_centering(X):
    X_mean=np.mean(X,axis=0)
    X_centered=X-X_mean
    return X_centered
```

In [14]:
```
from sklearn.decomposition import PCA

K=10
document_vector_centered=data_centering(document_vector)
print('Data_shape',document_vector_centered.shape)
pca = PCA(n_components = K)
projected_data = pca.fit_transform(document_vector_centered)
print('Projected data shape:',projected_data.shape)
```

```
Data_shape (1000, 3139)
Projected data shape: (1000, 10)
```

In [15]:
```
from sklearn.cluster import KMeans




def multivariate_normal(X, mean_vector, covariance_matrix):
    D=X.shape[0]
    exponent=0.5*(X-mean_vector).T@np.linalg.inv(covariance_matrix)@(
X-mean_vector)
    denominator=((2*np.pi)**D)*np.linalg.det(covariance_matrix)
    denominator=np.sqrt(denominator)
    value=np.exp(-exponent)/denominator
    return value

def random_sampling(X,X_labels,total_no_samples):
    list_indices=list(range(total_no_samples))
    np.random.shuffle(list_indices)
    return X[list_indices],X_labels[list_indices]

def get_log_likelihood(X,gmm_means,gmm_cov_matrix,theta):
    log_likelihood=0
    total_no_points=X.shape[0]
    no_components=gmm_means.shape[0]

    #print('total_no_points',total_no_points,'no_components',no_compo
nents)
    for i in range(total_no_points):
        mixed_guassian_sum=0
        for cluster_yi in range(no_components):
            likelihood=multivariate_normal(X[i],gmm_means[cluster_yi
],gmm_cov_matrix[cluster_yi])
            mixed_guassian_sum=mixed_guassian_sum+likelihood*theta[cl
uster_yi]
        log_likelihood=log_likelihood+np.log(mixed_guassian_sum)
    return log_likelihood


def get_updated_cov_matrix(X,r,means_updated,cov_matrix_type):
    #print("updating the cov matrix")
    total_no_points=X.shape[0]
    D=X.shape[1]
    no_components=r.shape[1]

    # updating the covariance matrices..................
    gmm_cov_matrix_updated=np.zeros(shape=(no_components,D,D))

    if(cov_matrix_type=='FULL'):
        for cluster_yi in range(no_components):
            temp_cov_matix=np.zeros(shape=(D,D))
            for i in range(total_no_points):
                x_i=X[i]
                temp_cov_matix=temp_cov_matix+r[i,cluster_yi]*np.oute
r(x_i-means_updated[cluster_yi],x_i-means_updated[cluster_yi])
            temp_cov_matix=temp_cov_matix/np.sum(r[:,cluster_yi])
            gmm_cov_matrix_updated[cluster_yi]=temp_cov_matix
    elif(cov_matrix_type=='DIAG'):
```

```
        for cluster_yi in range(no_components):
            temp_cov_matix=np.zeros(shape=(D,D))
            for i in range(total_no_points):
                x_i=X[i]
                temp_cov_matix=temp_cov_matix+r[i,cluster_yi]*np.oute
r(x_i-means_updated[cluster_yi],x_i-means_updated[cluster_yi])
            temp_cov_matix=np.diag(np.diag(temp_cov_matix))
            temp_cov_matix=temp_cov_matix/np.sum(r[:,cluster_yi])
            gmm_cov_matrix_updated[cluster_yi]=temp_cov_matix
    elif(cov_matrix_type=='IDENTITY'):
        for cluster_yi in range(no_components):
            gmm_cov_matrix_updated[cluster_yi]=np.eye(D)

    return gmm_cov_matrix_updated
```

In [16]:
```
def init_parameters_for_GMM(X,no_components):

    new_X = np.array_split(projected_data, 2)
    mean_vector = [np.mean(x, axis=0) for x in new_X]
    covariance_matrixes = [np.cov(x.T) for x in new_X]

    mean_vector=np.array(mean_vector)
    covariance_matrixes=np.array(covariance_matrixes)
    #init the probabilites
    theta=1/no_components*np.ones(shape=(no_components,1))

    return mean_vector,covariance_matrixes,theta

def Expectation_Maximization_Updated(X,no_components,no_iterations,co
v_matrix_type):

    # we are initilizing the parameters using K-means
    inital_means,gmm_cov_matrix,theta=init_parameters_for_GMM(X,no_co
mponents)

    print('get_log_likelihood',get_log_likelihood(X,inital_means,gmm_
cov_matrix,theta))

    log_likelihood=[]

    for iteration in range(no_iterations):
        total_no_points=X.shape[0]
        D=X.shape[1]

        # storing the posterior-probabilities...
        r=np.zeros(shape=(total_no_points,no_components),dtype='float
64')

        cluster_yi=1
        for i in range(total_no_points):
            dec_sum=0
            for cluster_yi in range(no_components):
                temp=multivariate_normal(X[i],inital_means[cluster_yi
],gmm_cov_matrix[cluster_yi])
                r[i][cluster_yi]=temp*theta[cluster_yi]
                dec_sum=dec_sum+r[i][cluster_yi]
            r[i,:]=r[i,:]/dec_sum
        print('Posterior calculation completed-------')

        # M-step updating the mean and Covariance-matirce
s...................

        # updating the probabilities........................

        theta_updated=np.zeros_like(theta)

        theta_updated=(1/total_no_points)*np.sum(r,axis=0)

        #updating the means........................
        means_updated=np.zeros_like(inital_means)
```

```
        for cluster_yi in range(no_components):
            #print("X:shape",X.shape)
            #print("r shape",r[:,index].shape)
            means_updated[cluster_yi]=X.T@r[:,cluster_yi]/np.sum(r[:,
cluster_yi])
            #print("Mean:"+str(i),means_updated[index])


        # updating the covariance matrices..................
        gmm_cov_matrix_updated=get_updated_cov_matrix(X,r,means_updat
ed,cov_matrix_type)

        inital_means=means_updated
        gmm_cov_matrix=gmm_cov_matrix_updated
        theta=theta_updated


        print("iteration: "+str(iteration)+" completed.")
        log_likelihood_value=get_log_likelihood(X,inital_means,gmm_co
v_matrix,theta)
        print('get_log_likelihood',log_likelihood_value)
        log_likelihood.append(log_likelihood_value)

    log_likelihood=np.array(log_likelihood)
    return inital_means,gmm_cov_matrix,theta,log_likelihood
```

In [17]:

```
def GMM_Sentiment_Model(positive_data,negative_data,no_components,no_
iterations,cov_matrix_type):

    gmm_positive_parameters=Expectation_Maximization_Updated(positive
_data,no_components,no_iterations,cov_matrix_type)
    gmm_negative_parameters=Expectation_Maximization_Updated(negative
_data,no_components,no_iterations,cov_matrix_type)

    return gmm_positive_parameters,gmm_negative_parameters
```

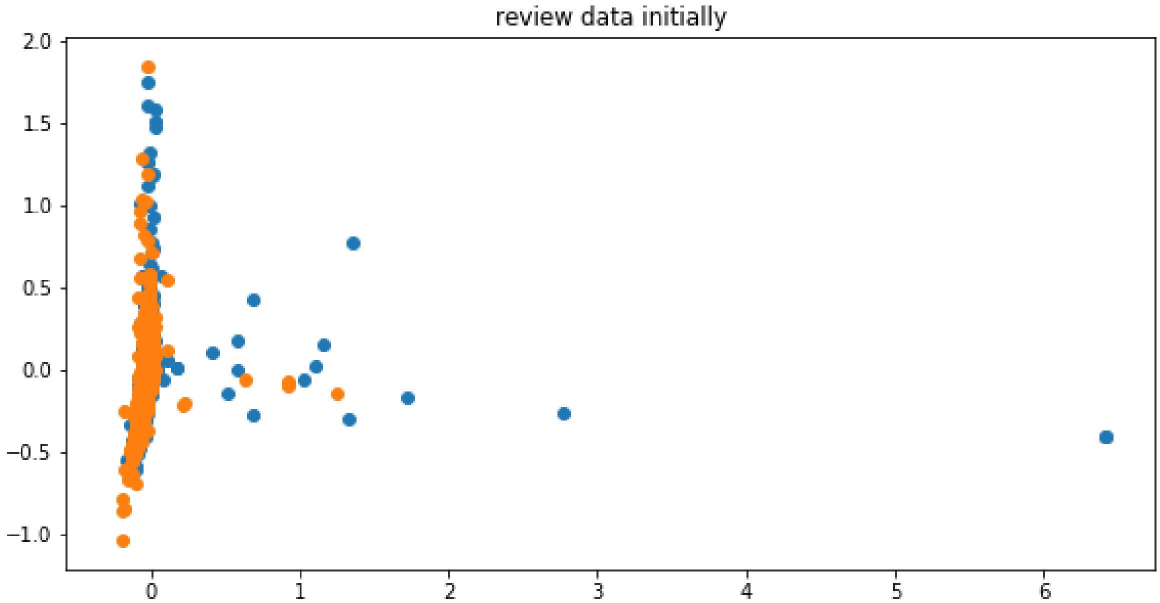
In [18]:

```
# separating the postive data and negative data.

labels=np.array(labels)
positive_indices=np.where(labels==1)
negative_indices=np.where(labels==0)
positive_data=projected_data[positive_indices]
negative_data=projected_data[negative_indices]
no_postive_samples=positive_data.shape[0]
no_negative_samples=negative_data.shape[0]

postive_labels=np.ones(shape=(no_postive_samples,1))
negative_labels=np.zeros(shape=(no_negative_samples,1))
```

In [19]:
```
def plot_log_likelihood_sum(music_log_likelihood_sum,speech_log_likel
ihood_sum):
    fig=plt.figure(figsize=(10,10))

    plot=fig.add_subplot(1,2,1)

    plt.plot(music_log_likelihood_sum,'r')
    plot.set_title('loglikelihood_positive')

    plot=fig.add_subplot(1,2,2)
    plot.set_title('loglikelihood_negative')
    plt.plot(speech_log_likelihood_sum,'g')
```

In [23]:
```
document_vector_centered=data_centering(document_vector)

pca = PCA(n_components = 2)
two_dim_data = pca.fit_transform(document_vector_centered)

plt.figure(figsize=(10,5))
plt.scatter(two_dim_data[:,0][positive_indices],two_dim_data[:,1][pos
itive_indices])
plt.scatter(two_dim_data[:,0][negative_indices],two_dim_data[:,1][neg
ative_indices])
plt.title('review data initially')
```

Out[23]: Text(0.5, 1.0, 'review data initially')



In [24]:
```
def get_likelihood(test_X,gmm_parameters):
    gmm_means_01,gmm_cov_matrix_01,theta_01,log_likelihood_01=gmm_par
ameters
    total_samples=test_X.shape[0]
    no_components=theta_01.shape[0]


    likelihood_table=np.zeros(shape=(total_samples,no_components))

    #print(total_samples,no_components)

    for i in range(total_samples):
        for cluster_yi in range(no_components):
            x_i=test_X[i]
            likelihood_table[i][cluster_yi]=multivariate_normal(x_i,g
mm_means_01[cluster_yi],gmm_cov_matrix_01[cluster_yi])

    likelihood_table=likelihood_table@theta_01
    #print(likelihood_table.shape)
    return likelihood_table.reshape(-1,1)



def sentiment_classifier(projected_data,gmm_positive_parameters,gmm_n
egative_parameters):

    positve_likelihood=get_likelihood(projected_data,gmm_positive_par
ameters)
    negative_likelihood=get_likelihood(projected_data,gmm_negative_pa
rameters)

    temp=np.hstack([negative_likelihood,positve_likelihood])

    calculated_labels=np.argmax(temp, axis=1)

    return calculated_labels
```

In [25]:
```
def Expectation_Maximization(X,no_components,no_iterations,cov_matrix
_type):

    parameters=[]
    # we are initilizing the parameters using K-means
    inital_means,gmm_cov_matrix,theta=init_parameters_for_GMM(X,no_co
mponents)

    print('get_log_likelihood',get_log_likelihood(X,inital_means,gmm_
cov_matrix,theta))

    log_likelihood=[]

    for iteration in range(no_iterations):
        total_no_points=X.shape[0]
        D=X.shape[1]

        # storing the posterior-probabilities...
        r=np.zeros(shape=(total_no_points,no_components),dtype='float
64')

        cluster_yi=1
        for i in range(total_no_points):
            dec_sum=0
            for cluster_yi in range(no_components):
                temp=multivariate_normal(X[i],inital_means[cluster_yi
],gmm_cov_matrix[cluster_yi])
                r[i][cluster_yi]=temp*theta[cluster_yi]
                dec_sum=dec_sum+r[i][cluster_yi]
            r[i,:]=r[i,:]/dec_sum
        print('Posterior calculation completed-------')

        # M-step updating the mean and Covariance-matirce
s....................

        # updating the probabilities........................

        theta_updated=np.zeros_like(theta)

        theta_updated=(1/total_no_points)*np.sum(r,axis=0)

        #updating the means........................
        means_updated=np.zeros_like(inital_means)

        for cluster_yi in range(no_components):
            #print("X:shape",X.shape)
            #print("r shape",r[:,index].shape)
            means_updated[cluster_yi]=X.T@r[:,cluster_yi]/np.sum(r[:,
cluster_yi])
            #print("Mean:"+str(i),means_updated[index])


        # updating the covariance matrices..................
        gmm_cov_matrix_updated=get_updated_cov_matrix(X,r,means_updat
ed,cov_matrix_type)
```

```
        inital_means=means_updated
        gmm_cov_matrix=gmm_cov_matrix_updated
        theta=theta_updated


        print("iteration: "+str(iteration)+" completed.")
        log_likelihood_value=get_log_likelihood(X,inital_means,gmm_co
v_matrix,theta)
        print('get_log_likelihood',log_likelihood_value)
        log_likelihood.append(log_likelihood_value)
        parameters.append([inital_means,gmm_cov_matrix,theta,log_like
lihood_value])

    log_likelihood=np.array(log_likelihood)
    return parameters
```

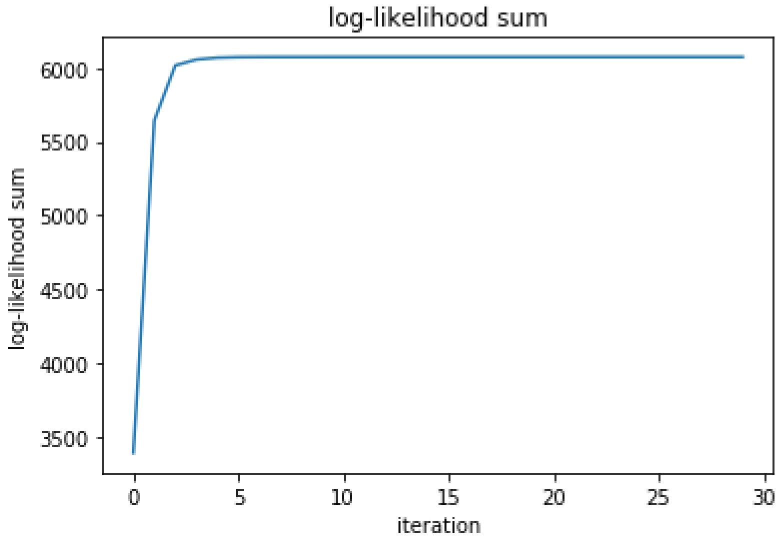
In [26]:
```
gmm_parameters=Expectation_Maximization_Updated(projected_data,2,30,
'DIAG')
```

```
get_log_likelihood [2250.0977149]
Posterior calculation completed-------
iteration: 0 completed.
get_log_likelihood 3389.8023837943183
Posterior calculation completed-------
iteration: 1 completed.
get_log_likelihood 5643.8703110326205
Posterior calculation completed-------
iteration: 2 completed.
get_log_likelihood 6017.2470776596365
Posterior calculation completed-------
iteration: 3 completed.
get_log_likelihood 6057.169215798919
Posterior calculation completed-------
iteration: 4 completed.
get_log_likelihood 6069.979716248172
Posterior calculation completed-------
iteration: 5 completed.
get_log_likelihood 6073.384517108467
Posterior calculation completed-------
iteration: 6 completed.
get_log_likelihood 6074.230681890864
Posterior calculation completed-------
iteration: 7 completed.
get_log_likelihood 6074.46413282525
Posterior calculation completed-------
iteration: 8 completed.
get_log_likelihood 6074.529973733465
Posterior calculation completed-------
iteration: 9 completed.
get_log_likelihood 6074.54852172932
Posterior calculation completed-------
iteration: 10 completed.
get_log_likelihood 6074.553730037406
Posterior calculation completed-------
iteration: 11 completed.
get_log_likelihood 6074.555189278704
Posterior calculation completed-------
iteration: 12 completed.
get_log_likelihood 6074.5555975944835
Posterior calculation completed-------
iteration: 13 completed.
get_log_likelihood 6074.555711765475
Posterior calculation completed-------
iteration: 14 completed.
get_log_likelihood 6074.555743677002
Posterior calculation completed-------
iteration: 15 completed.
get_log_likelihood 6074.555752594637
Posterior calculation completed-------
iteration: 16 completed.
get_log_likelihood 6074.555755086367
Posterior calculation completed-------
iteration: 17 completed.
get_log_likelihood 6074.555755782556
Posterior calculation completed-------
iteration: 18 completed.
```

```
get_log_likelihood 6074.555755977078
Posterior calculation completed-------
iteration: 19 completed.
get_log_likelihood 6074.555756031415
Posterior calculation completed-------
iteration: 20 completed.
get_log_likelihood 6074.5557560466
Posterior calculation completed-------
iteration: 21 completed.
get_log_likelihood 6074.555756050851
Posterior calculation completed-------
iteration: 22 completed.
get_log_likelihood 6074.555756052026
Posterior calculation completed-------
iteration: 23 completed.
get_log_likelihood 6074.555756052365
Posterior calculation completed-------
iteration: 24 completed.
get_log_likelihood 6074.555756052455
Posterior calculation completed-------
iteration: 25 completed.
get_log_likelihood 6074.555756052476
Posterior calculation completed-------
iteration: 26 completed.
get_log_likelihood 6074.555756052472
Posterior calculation completed-------
iteration: 27 completed.
get_log_likelihood 6074.555756052484
Posterior calculation completed-------
iteration: 28 completed.
get_log_likelihood 6074.5557560524885
Posterior calculation completed-------
iteration: 29 completed.
get_log_likelihood 6074.555756052485
```

In [32]:
```
plt.plot(gmm_parameters[3])
plt.title('log-likelihood sum')
plt.xlabel('iteration')
plt.ylabel('log-likelihood sum')
```

Out[32]: Text(0, 0.5, 'log-likelihood sum')



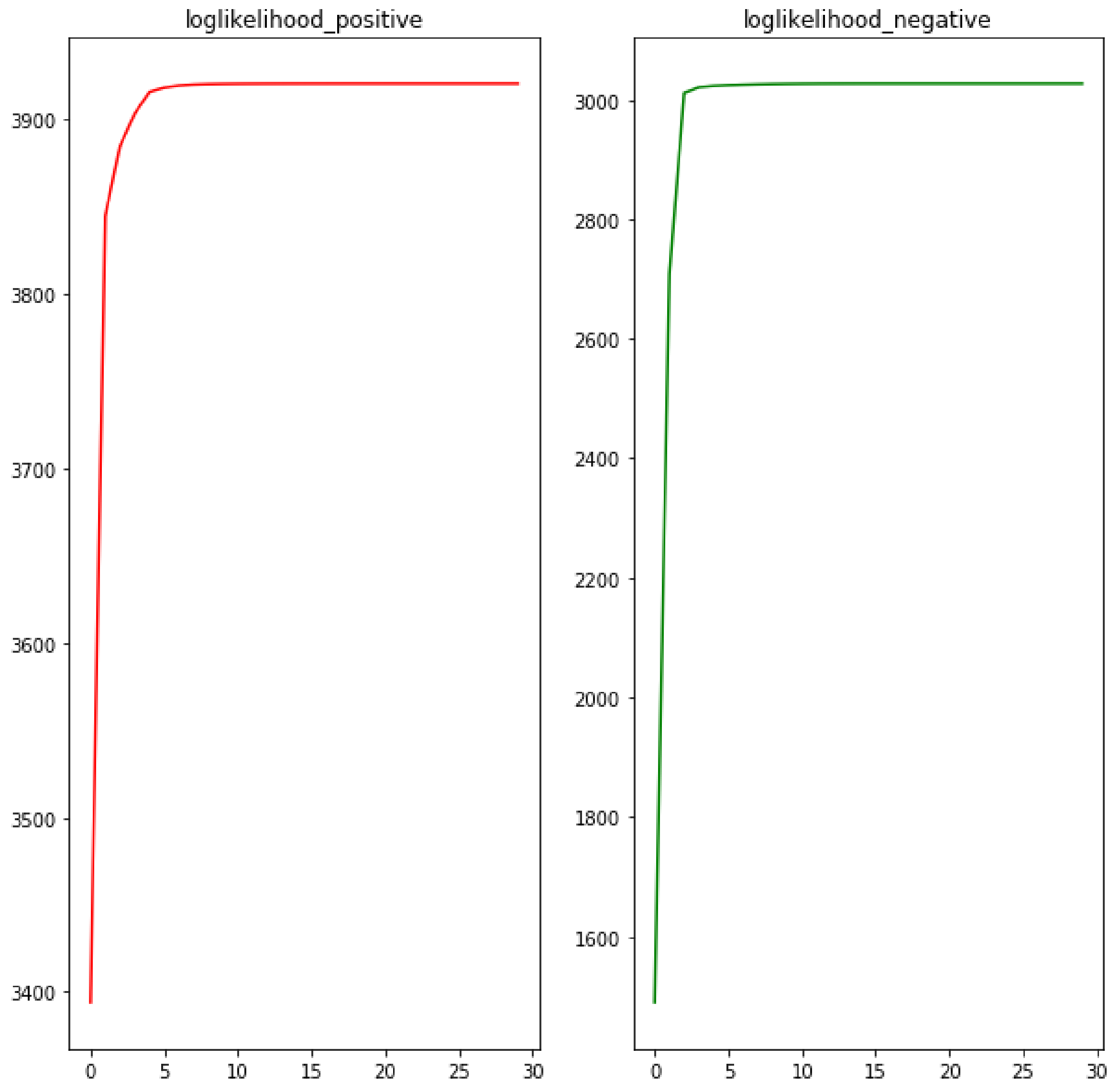
In [33]:
```
gmm_positive_parameters_01,gmm_negative_parameters_01=GMM_Sentiment_Model(positive_data,negative_data,2,30,'DIAG')
```

```
get_log_likelihood [1557.28628426]
Posterior calculation completed-------
iteration: 0 completed.
get_log_likelihood 3393.882436231508
Posterior calculation completed-------
iteration: 1 completed.
get_log_likelihood 3844.741825032239
Posterior calculation completed-------
iteration: 2 completed.
get_log_likelihood 3884.620599194271
Posterior calculation completed-------
iteration: 3 completed.
get_log_likelihood 3903.344319972882
Posterior calculation completed-------
iteration: 4 completed.
get_log_likelihood 3915.662273682217
Posterior calculation completed-------
iteration: 5 completed.
get_log_likelihood 3918.2166648349234
Posterior calculation completed-------
iteration: 6 completed.
get_log_likelihood 3919.4454993632257
Posterior calculation completed-------
iteration: 7 completed.
get_log_likelihood 3920.033299463052
Posterior calculation completed-------
iteration: 8 completed.
get_log_likelihood 3920.317633196484
Posterior calculation completed-------
iteration: 9 completed.
get_log_likelihood 3920.461117627058
Posterior calculation completed-------
iteration: 10 completed.
get_log_likelihood 3920.5381889476885
Posterior calculation completed-------
iteration: 11 completed.
get_log_likelihood 3920.5825029896946
Posterior calculation completed-------
iteration: 12 completed.
get_log_likelihood 3920.6095911044486
Posterior calculation completed-------
iteration: 13 completed.
get_log_likelihood 3920.6269644476197
Posterior calculation completed-------
iteration: 14 completed.
get_log_likelihood 3920.638500770851
Posterior calculation completed-------
iteration: 15 completed.
get_log_likelihood 3920.6463502386355
Posterior calculation completed-------
iteration: 16 completed.
get_log_likelihood 3920.651784921499
Posterior calculation completed-------
iteration: 17 completed.
get_log_likelihood 3920.6555969369633
Posterior calculation completed-------
iteration: 18 completed.
```

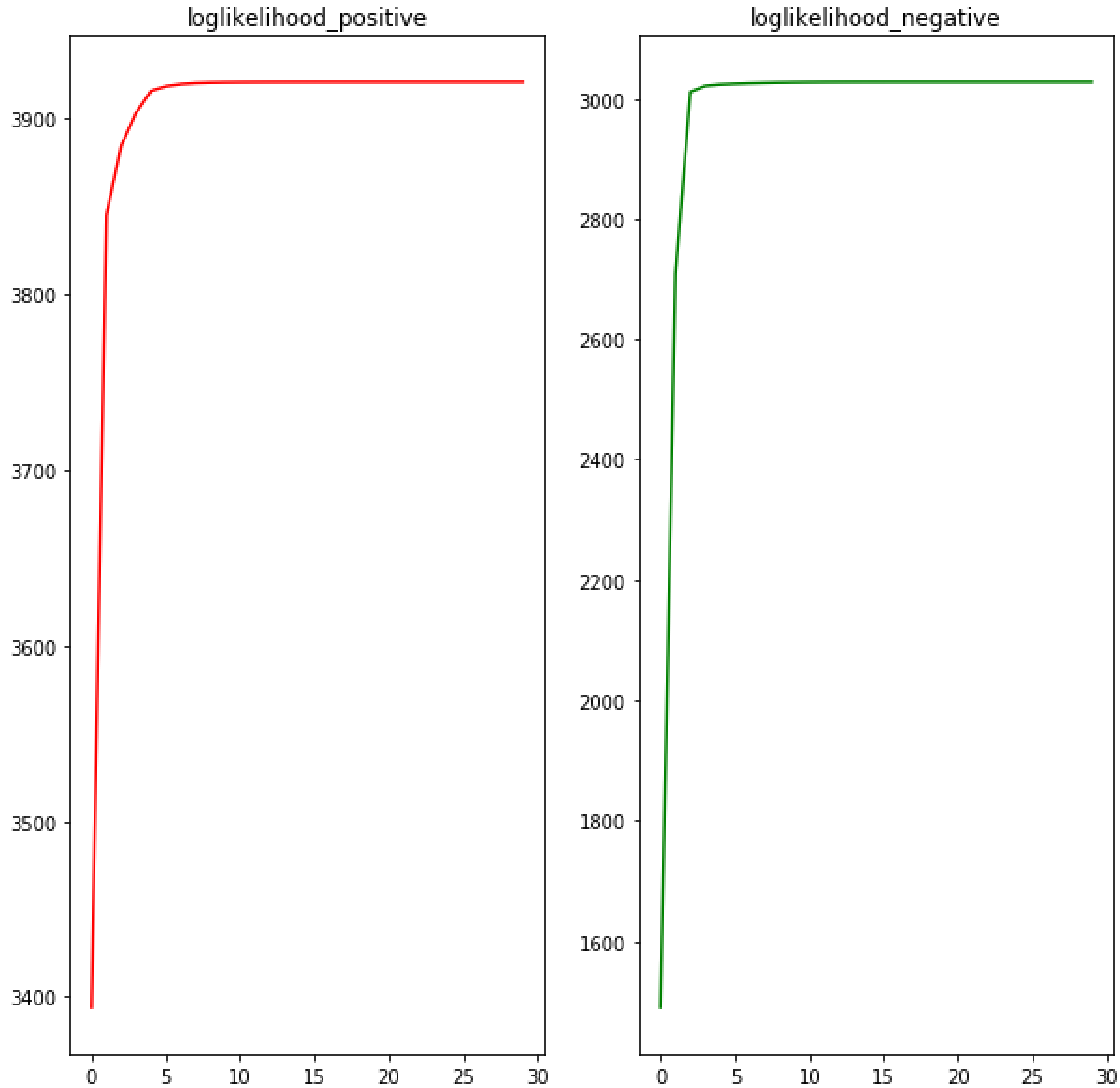
```
get_log_likelihood 3920.65829826168
Posterior calculation completed-------
iteration: 19 completed.
get_log_likelihood 3920.660228674943
Posterior calculation completed-------
iteration: 20 completed.
get_log_likelihood 3920.6616180465176
Posterior calculation completed-------
iteration: 21 completed.
get_log_likelihood 3920.6626241842955
Posterior calculation completed-------
iteration: 22 completed.
get_log_likelihood 3920.6633567117306
Posterior calculation completed-------
iteration: 23 completed.
get_log_likelihood 3920.663892541864
Posterior calculation completed-------
iteration: 24 completed.
get_log_likelihood 3920.664286105863
Posterior calculation completed-------
iteration: 25 completed.
get_log_likelihood 3920.6645762193175
Posterior calculation completed-------
iteration: 26 completed.
get_log_likelihood 3920.6647907504434
Posterior calculation completed-------
iteration: 27 completed.
get_log_likelihood 3920.6649498289007
Posterior calculation completed-------
iteration: 28 completed.
get_log_likelihood 3920.6650680731823
Posterior calculation completed-------
iteration: 29 completed.
get_log_likelihood 3920.6651561504914
get_log_likelihood [692.81143064]
Posterior calculation completed-------
iteration: 0 completed.
get_log_likelihood 1489.8525678062088
Posterior calculation completed-------
iteration: 1 completed.
get_log_likelihood 2708.202407019515
Posterior calculation completed-------
iteration: 2 completed.
get_log_likelihood 3010.665692763451
Posterior calculation completed-------
iteration: 3 completed.
get_log_likelihood 3020.4504494910416
Posterior calculation completed-------
iteration: 4 completed.
get_log_likelihood 3022.793182785805
Posterior calculation completed-------
iteration: 5 completed.
get_log_likelihood 3023.904317380234
Posterior calculation completed-------
iteration: 6 completed.
get_log_likelihood 3024.7565574064465
Posterior calculation completed-------
```

```
iteration: 7 completed.
get_log_likelihood 3025.4410354671227
Posterior calculation completed-------
iteration: 8 completed.
get_log_likelihood 3025.9532415125177
Posterior calculation completed-------
iteration: 9 completed.
get_log_likelihood 3026.3105034692594
Posterior calculation completed-------
iteration: 10 completed.
get_log_likelihood 3026.545635306921
Posterior calculation completed-------
iteration: 11 completed.
get_log_likelihood 3026.6914774830366
Posterior calculation completed-------
iteration: 12 completed.
get_log_likelihood 3026.7765611219734
Posterior calculation completed-------
iteration: 13 completed.
get_log_likelihood 3026.823484035149
Posterior calculation completed-------
iteration: 14 completed.
get_log_likelihood 3026.8481988135695
Posterior calculation completed-------
iteration: 15 completed.
get_log_likelihood 3026.860773183584
Posterior calculation completed-------
iteration: 16 completed.
get_log_likelihood 3026.8670138288567
Posterior calculation completed-------
iteration: 17 completed.
get_log_likelihood 3026.8700577583195
Posterior calculation completed-------
iteration: 18 completed.
get_log_likelihood 3026.871524753856
Posterior calculation completed-------
iteration: 19 completed.
get_log_likelihood 3026.8722259381743
Posterior calculation completed-------
iteration: 20 completed.
get_log_likelihood 3026.872559181101
Posterior calculation completed-------
iteration: 21 completed.
get_log_likelihood 3026.8727169359486
Posterior calculation completed-------
iteration: 22 completed.
get_log_likelihood 3026.8727914136803
Posterior calculation completed-------
iteration: 23 completed.
get_log_likelihood 3026.8728265096265
Posterior calculation completed-------
iteration: 24 completed.
get_log_likelihood 3026.872843026456
Posterior calculation completed-------
iteration: 25 completed.
get_log_likelihood 3026.872850792674
Posterior calculation completed-------
```

```
iteration: 26 completed.
get_log_likelihood 3026.8728544421097
Posterior calculation completed-------
iteration: 27 completed.
get_log_likelihood 3026.8728561563057
Posterior calculation completed-------
iteration: 28 completed.
get_log_likelihood 3026.8728569612476
Posterior calculation completed-------
iteration: 29 completed.
get_log_likelihood 3026.872857339158
```



In [34]:
```
plot_log_likelihood_sum(gmm_positive_parameters_01[3],gmm_negative_pa
rameters_01[3])
```



In [36]:
```
gmm_positive_parameters=Expectation_Maximization(positive_data,2,30,
'DIAG')
gmm_negative_parameters=Expectation_Maximization(positive_data,2,30,
'DIAG')
```

```
get_log_likelihood [1557.28628426]
Posterior calculation completed-------
iteration: 0 completed.
get_log_likelihood 3393.882436231508
Posterior calculation completed-------
iteration: 1 completed.
get_log_likelihood 3844.741825032239
Posterior calculation completed-------
iteration: 2 completed.
get_log_likelihood 3884.620599194271
Posterior calculation completed-------
iteration: 3 completed.
get_log_likelihood 3903.344319972882
Posterior calculation completed-------
iteration: 4 completed.
get_log_likelihood 3915.662273682217
Posterior calculation completed-------
iteration: 5 completed.
get_log_likelihood 3918.2166648349234
Posterior calculation completed-------
iteration: 6 completed.
get_log_likelihood 3919.4454993632257
Posterior calculation completed-------
iteration: 7 completed.
get_log_likelihood 3920.033299463052
Posterior calculation completed-------
iteration: 8 completed.
get_log_likelihood 3920.317633196484
Posterior calculation completed-------
iteration: 9 completed.
get_log_likelihood 3920.461117627058
Posterior calculation completed-------
iteration: 10 completed.
get_log_likelihood 3920.5381889476885
Posterior calculation completed-------
iteration: 11 completed.
get_log_likelihood 3920.5825029896946
Posterior calculation completed-------
iteration: 12 completed.
get_log_likelihood 3920.6095911044486
Posterior calculation completed-------
iteration: 13 completed.
get_log_likelihood 3920.6269644476197
Posterior calculation completed-------
iteration: 14 completed.
get_log_likelihood 3920.638500770851
Posterior calculation completed-------
iteration: 15 completed.
get_log_likelihood 3920.6463502386355
Posterior calculation completed-------
iteration: 16 completed.
get_log_likelihood 3920.651784921499
Posterior calculation completed-------
iteration: 17 completed.
get_log_likelihood 3920.6555969369633
Posterior calculation completed-------
iteration: 18 completed.
```

```
get_log_likelihood 3920.65829826168
Posterior calculation completed-------
iteration: 19 completed.
get_log_likelihood 3920.660228674943
Posterior calculation completed-------
iteration: 20 completed.
get_log_likelihood 3920.6616180465176
Posterior calculation completed-------
iteration: 21 completed.
get_log_likelihood 3920.6626241842955
Posterior calculation completed-------
iteration: 22 completed.
get_log_likelihood 3920.6633567117306
Posterior calculation completed-------
iteration: 23 completed.
get_log_likelihood 3920.663892541864
Posterior calculation completed-------
iteration: 24 completed.
get_log_likelihood 3920.664286105863
Posterior calculation completed-------
iteration: 25 completed.
get_log_likelihood 3920.6645762193175
Posterior calculation completed-------
iteration: 26 completed.
get_log_likelihood 3920.6647907504434
Posterior calculation completed-------
iteration: 27 completed.
get_log_likelihood 3920.6649498289007
Posterior calculation completed-------
iteration: 28 completed.
get_log_likelihood 3920.6650680731823
Posterior calculation completed-------
iteration: 29 completed.
get_log_likelihood 3920.6651561504914
get_log_likelihood [1557.28628426]
Posterior calculation completed-------
iteration: 0 completed.
get_log_likelihood 3393.882436231508
Posterior calculation completed-------
iteration: 1 completed.
get_log_likelihood 3844.741825032239
Posterior calculation completed-------
iteration: 2 completed.
get_log_likelihood 3884.620599194271
Posterior calculation completed-------
iteration: 3 completed.
get_log_likelihood 3903.344319972882
Posterior calculation completed-------
iteration: 4 completed.
get_log_likelihood 3915.662273682217
Posterior calculation completed-------
iteration: 5 completed.
get_log_likelihood 3918.2166648349234
Posterior calculation completed-------
iteration: 6 completed.
get_log_likelihood 3919.4454993632257
Posterior calculation completed-------
```

```
iteration: 7 completed.
get_log_likelihood 3920.033299463052
Posterior calculation completed-------
iteration: 8 completed.
get_log_likelihood 3920.317633196484
Posterior calculation completed-------
iteration: 9 completed.
get_log_likelihood 3920.461117627058
Posterior calculation completed-------
iteration: 10 completed.
get_log_likelihood 3920.5381889476885
Posterior calculation completed-------
iteration: 11 completed.
get_log_likelihood 3920.5825029896946
Posterior calculation completed-------
iteration: 12 completed.
get_log_likelihood 3920.6095911044486
Posterior calculation completed-------
iteration: 13 completed.
get_log_likelihood 3920.6269644476197
Posterior calculation completed-------
iteration: 14 completed.
get_log_likelihood 3920.638500770851
Posterior calculation completed-------
iteration: 15 completed.
get_log_likelihood 3920.6463502386355
Posterior calculation completed-------
iteration: 16 completed.
get_log_likelihood 3920.651784921499
Posterior calculation completed-------
iteration: 17 completed.
get_log_likelihood 3920.6555969369633
Posterior calculation completed-------
iteration: 18 completed.
get_log_likelihood 3920.65829826168
Posterior calculation completed-------
iteration: 19 completed.
get_log_likelihood 3920.660228674943
Posterior calculation completed-------
iteration: 20 completed.
get_log_likelihood 3920.6616180465176
Posterior calculation completed-------
iteration: 21 completed.
get_log_likelihood 3920.6626241842955
Posterior calculation completed-------
iteration: 22 completed.
get_log_likelihood 3920.6633567117306
Posterior calculation completed-------
iteration: 23 completed.
get_log_likelihood 3920.663892541864
Posterior calculation completed-------
iteration: 24 completed.
get_log_likelihood 3920.664286105863
Posterior calculation completed-------
iteration: 25 completed.
get_log_likelihood 3920.6645762193175
Posterior calculation completed-------
```

```
iteration: 26 completed.
get_log_likelihood 3920.6647907504434
Posterior calculation completed-------
iteration: 27 completed.
get_log_likelihood 3920.6649498289007
Posterior calculation completed-------
iteration: 28 completed.
get_log_likelihood 3920.6650680731823
Posterior calculation completed-------
iteration: 29 completed.
get_log_likelihood 3920.6651561504914
```

In [38]:
```
index=5
sentiment_classifier(projected_data,gmm_positive_parameters[index],gmm_negative_parameters[index])
```

Out[38]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```