Boosted-GMM Algorithm

Zheqi Wu
wzq2017@ucla.edu
UID: 705054782

Abstract

Boosting is an efficient method to improve the classification performance. Recent theoretical work has shown that the boosting technique can be viewed as a gradient descent search for a good fit in function space. Several authors have applied such viewpoint to solve the density estimation problems. In this paper, we introduce the Boosted-GMM algorithm, which embeds the EM algorithm in a boosting framework and which can be used to reliably and accurately estimate the class-conditional probabilistic distributions in any pattern recognition problems based on a training data set. We do the simulation to prove that the algorithm is effective and reliable. Then we apply the Boosted-GMM algorithm to handwritten digits data set and our experiments show it perform better than EM-GMM algorithm and increase 8% of accuracy of clustering.

Keywords: boosted, GMM, hand-written pattern recognition

1 Introduction

1.1 Hand-Written Pattern Recognition

Automatic recognition of hand-written characters has been hot recent years due to its many potential applications such as automatic sorting of postal articles and for intelligent man/machine interface. The emphasis has mostly been on developing a writer-independent hand-written character recognition system. This requirement translates into dealing with large data sets with very wide variability within a given characters manifestations as well as close similarities between various renderings of different characters. To achieve good recognition performance, e.g. over 95%[1] accuracy on all hand-written characters is a daunting task still not achieved by commercial character recognition systems.

1.2 Gaussian Mixture Model

Gaussian mixture models (GMMs) is one of the most popular and effective tools for hand-written pattern recognition problems. Typically, GMMs are used to model the class-conditional distributions of the features and their parameters are estimated by the expectation maximization (EM) algorithm based on input data set. Then, classification is performed to minimize the classification error w.r.t. the estimated class-conditional distributions. We call this method the EM-GMM algorithm.

One known problem of maximum likelihood estimation techniques such as the EM algorithm is that the estimate is not globally optimal. Most often, the EM algorithm gets stuck in the local maximum of the data log likelihood function. This problem can become very severe when bad initializations of the model parameters are given. Since the accuracy of model estimation is the most important, or even deciding, factor for generative model-based classification, how to reliably and precisely estimate the class-conditional GMMs based on a training data set becomes a central problem of hand-written recognition.

1.3 Boosting

Boosting[2] is one of the most important developments in classification methodology. It can reduce the variance of the unstable classifiers and improve the classification performance[3]. Some theoretical work suggests that the effectiveness of this method can be attributed to its tendency to produce large margin classifiers. Mason et al[4] generalized this margin-based idea and derived boosting algorithms as gradient descent algorithms. They proved that the weights in every iteration of the boosting algorithms correspond to the gradient of some margin cost function at current fit. In a recent paper, Saharon et al[5] showed that the gradient-based boosting methodology can be applied to density estimation problems and proposed a general boosting density estimation framework. They also illustrated the potential of their framework by experiments with boosting Bayesian networks to learn density models.

2 Boosted GMM Algorithm

The GMM model assume that the form of the probability density function is a linear combination of a finite number of Gaussian distribution

$$p(\mathbf{y}|\theta,\lambda) = \sum_{m=1}^{K} \lambda_m \mathcal{N}(\mathbf{y}|\mu_m, \Sigma_m) \tag{1}$$

where λ_m is the proportion of the population from the m^{th} component

$$\mathcal{N}(\mathbf{y}|\mu_m, \Sigma_m) = \frac{\exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu_m})^{\mathrm{T}}\boldsymbol{\Sigma_m}^{-1}(\mathbf{y}-\boldsymbol{\mu_m})\right)}{\sqrt{(2\pi)^n|\boldsymbol{\Sigma_m}|}}$$

and $\sum_m^K \lambda_m = 1$.

Given a training data set $\mathbf{Y} = \{\mathbf{y_i}\}_{\mathbf{i=1}}^{\mathbf{N}}$ and a probability density function $p(\mathbf{y})$ to be estimated, the data log likelikhood is given by

$$L(p) = \sum_{i=1}^{N} \log p(\mathbf{y_i}) \tag{2}$$

2.1 EM-GMM

The next step is to estimate the model parameters that maximizes the data log likelihood. In practice, this problem can be solved by maximum likelihood estimation (MLE) techniques such as expectation maximization (EM) algorithm.

Let $Z_i \sim \mathcal{M}(1, \lambda)$, then we want to estimate the parameters $\hat{\theta} = (\hat{\lambda}, \hat{\mu}_m, \hat{\Sigma}_m)$.

E-step:

$$W_{im}^{(t)} = \frac{\lambda_m^{(t)} p(y_i|u_m^{(t)}, \Sigma_m)}{\sum_{j=1}^{K} \lambda_j^{(t)} p(y_i|u_j^{(t)}, \Sigma_j)}$$

M-step:

$$u_m^{(t+1)} = \frac{\sum_i W_{im}^{(t)} y_i}{W_{\cdot m}^{(t)}}$$

$$\Sigma_m^{(t+1)} = \frac{\sum_i W_{im}^{(t)} y_i y_i^T}{W_{\cdot m}^{(t)}} - \mu_m^{(t+1)} (\mu_m^{(t+1)})^T$$

Table 1: The Boosted-GMM algorithm

Algorithm The Boosted-GMM algorithm

```
1. Input: $\mathbf{Y} = \{\mathbf{y_i}\}_{\mathbf{i=1}}^{\mathbf{N}}$, cluster K and iteration T
2. Initialize $W_0(\mathbf{y}_i) = 1/N$, probability density function given original data $\mathrm{G}_0$
3. For $t = 1, ..., T$ or until $L(G_t) \leq L(G_{t-1})$:
   (a) Sample $Y_t$ from $Y$ according to $W_t$ and do GMM estimation on the sampled data
       set to get $g_t'$
   (b) For each example $\mathbf{y_i}$ from original data set Y, If $y_i \notin Y_t$, $g_t(y_i) = 0$,
       else $g_t(y_i) =$ mean$(g_t'(Y_t == y_i))$
   (c) $G_t = (1 - \eta_t)G_{t-1} + \eta_t g_t$ where $\eta = argmin_\eta - L(G_t)$
   (d) Update $W_t(\mathbf{y_i}) = \frac{1}{G_{t-1}(\mathbf{y_i})}$ with normalization
4. Output: Final density estimate $G_T$
```

Predict Z_i:

$$p(Z_{im} = 1 | y_i, \hat{\theta}, \hat{\lambda}) = \frac{p(y_i|\hat{\theta}_m)\hat{\lambda}_m}{\sum_{j=1}^{K} p(y_i|\hat{\theta}_j)\hat{\lambda}_j}$$

Let $m^* = \hat{W}_{im}$, then predict $Z_{im}^* = 1$.

2.2 Boosted-GMM

However, the EM algorithm will converge to local optimal points which means that other local optimums may, or may not, exist with better objective values. Typically, we are careful about the initial value of parameters, especially when there is insufficient training data.

In this paper, we apply GMM algorithm with boosting methodology so that we can reduce the change of getting trapped in a local optimum during EM process.

In detail, instead of directly optimizing Equation 2 as in the EM algorithm, we start with an initial estimate G_0 which is predicted posterior probability of each component given the original data from GMM, and iteratively add to this estimate a small component g_t (also a probability distribution from GMM given a new bootstrap sample) at roumd t. That is

$$G_t = (1 - \eta_t)G_{t-1} + \eta_t g_t \tag{3}$$

where $\eta \in (0, 1)$, and the new bootstrap sample at each round is weighted by $W_t = \frac{1}{G_{t-1}}$. This meets our intuition of boosting that more focus on the examples with low probabilities under the previous estimate.

More precisely, For each example $\mathbf{y_i}$ we have initial estimate $G_{0\mathbf{1}\times\mathbf{K}}(\mathbf{y_i})$, we do the weighted bootstrap sampling to get a new data set $\mathbf{Y_1}$ and get a the probability distribution function from GMM $g'_{1\mathbf{N}\times\mathbf{K}}$. If $y_i \notin Y_1$, $g_1(y_i) = 0$, else $g_1(y_i) =$ mean$(g_1'(Y_1 == y_i))$.

According to the Taylor's theorem, the new data log likelihood

$$L(G_t) = L((1 - \eta_t)G_{t-1} + \eta_t g_t) \tag{4}$$

Then given the g_t, we can seek the η that minimize the negative log likelihood $-L(G_t)$.

Finally, once the final GMM G is estimated, we assign each data object to the Gaussian component in G under which its probability is the highest. The detailed description of our algorithm is shown in Table 1.

3 Simulation

We firstly test our boosted-GMM algorithm on simulation data set and compare the performance with EM-GMM algorithm, K-means and k-medoids algorithm.

3.1 Toy Dataset

Let cluster $K = 3$, we draw $n = 160$ examples with 2 features from each multivariate normal distribution

$$\begin{pmatrix} Y_{m1} \\ Y_{m2} \end{pmatrix} \sim \mathcal{N}_2 \left(\begin{pmatrix} \mu_{m1} \\ \mu_{m2} \end{pmatrix}, \begin{pmatrix} \sigma_{m1}^2 & 0 \\ 0 & \sigma_{m2}^2 \end{pmatrix} \right)$$

where $\mu = [[0, 0.5], [1, 1], [2, 0.5]]$ and $\Sigma = [[0.1, 0.1], [0.25, 0.25], [0.15, 0.15]]$ with label $\in \{0, 1, 2\}$.

3.2 Metrics to Evaluate Algorithms

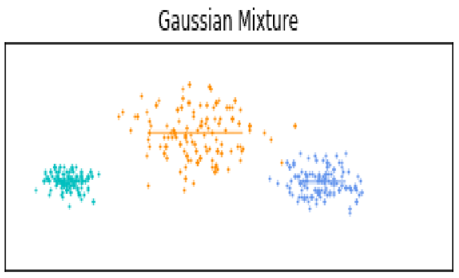
We use Accuracy[6] to measure the performance of different clustering algorithms. Since all the algorithms compared in this paper are unsupervised machine learning algorithms, i.e., label $\{1, 2\}$ is equal to label $\{2, 1\}$, their previous actual label does not matter, it only matters the clusters themselves.

So we divide the data set by their actual label, and set the mode of their predicted label as their actual label in order to calculate the accuracy. We ran such experiments 100 times so that we have the accuracy mean and accuracy standard deviation representing the effectiveness and reliability of the different unsupervised algorithms.

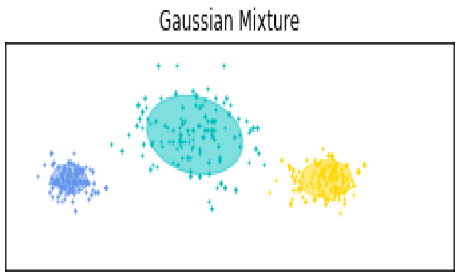
3.3 Experiment Result

Since we only have two features in the toy data set, the classification result for one time can be shown in 2-D space in Figure 1 and Figure 2.

Our experiment results demonstrate that both boosted-GMM and EM-GMM can 100% sucessfully devide these points into three clusters during 100 repeated process while K-means and K-medoids would sometimes be confused by outliers. In general, the boosted-GMM we come up with are highly reliable and effective, but we are not sure if it is better than EM-GMM algorithm, so it is worthwhile to be tested in the real world data set.

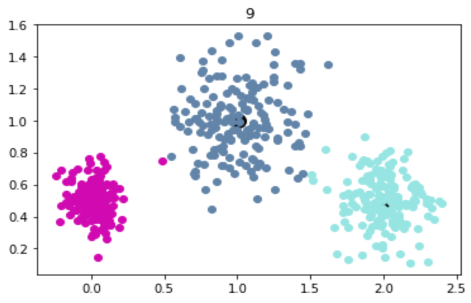


(a) GMM clustering

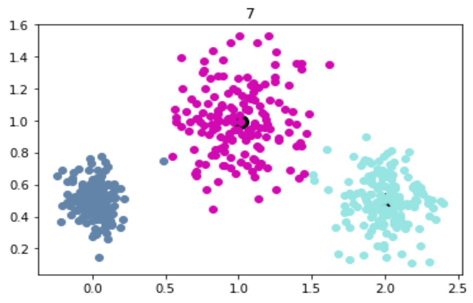


(b) Boosted GMM clustering

Figure 1: GMM and Boosted-GMM clustering



(a) K-means clustering



(b) K-medoids clustering

Figure 2: K-means and K-medoids clustering

Table 2: Simulation Results

Algorithm	Accuracy mean	Accuracy Std
K-means	0.9625	0.0106
K-medoids	0.9896	0.0098
GMM	**1.0000**	0.0000
Boosted-GMM	**1.0000**	0.0000

4 Real data application

We then apply the Boosted-GMM algorithm into pattern recognition. We will introduce a classic hand-written data set with 10 clusters and then apply all algorithms we mentioned above to compare the results.

4.1 Data Set Description

The MNIST database of handwritten digits, has a set of 70,000 examples, with 784 features and labels $\in [0, 1, ..., 9]$. The digits have been size-normalized and centered in a fixed-size image.

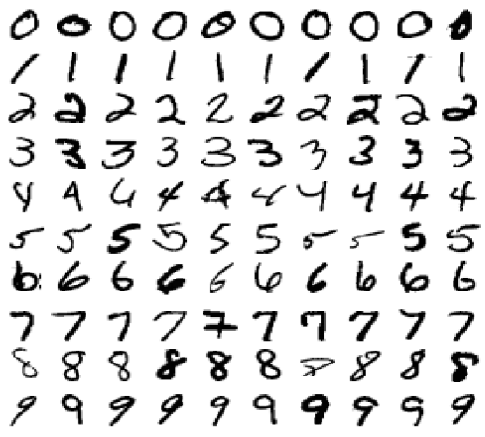


Figure 3: MNIST data

Table 3: MNIST dataset Results

Algorithm	Accuracy mean	Accuracy Std
K-means	0.3478	0.0580
K-medoids	0.3958	0.0703
GMM	0.6332	0.0713
Boosted-GMM	0.7121	0.0141

4.2 Data Preprocessing

Instead of conducting a complete data process, which would be very time consuming, we ran experiments on the transformed data set which contains 70% information of original data set. The Principal component analysis (PCA) is an effective method to carry out this procedure. Also, we randomly choose 20% data as our input. Now our new data set is consist of 14000 examples, with 26 features and belongs to 10 different clusters.

4.3 Experiment Result

Same as the simulation experiment, we ran each experiment 100 times to get the accuracy mean and accuracy standard deviation, which can be shown in Table 3. Then our experiments show the boosted-GMM algorithm performs better than EM-GMM algorithm and increase 8% of accuracy of clustering with higher stability.

Figure 4 shows the average overall hand-written pattern recognition rates of the Boosted-GMM algorithm compared with the recognition rates with the average overall hand-written recognition rate of the EM-GMM algorithm. Our experiments indicate that the recognition rates are effectively and significantly boosted by the Boosted-GMM algorithm as compared to the EM-GMM algorithm due to the fact that boosting can lead to more accurate estimates of the class-conditional GMMs.

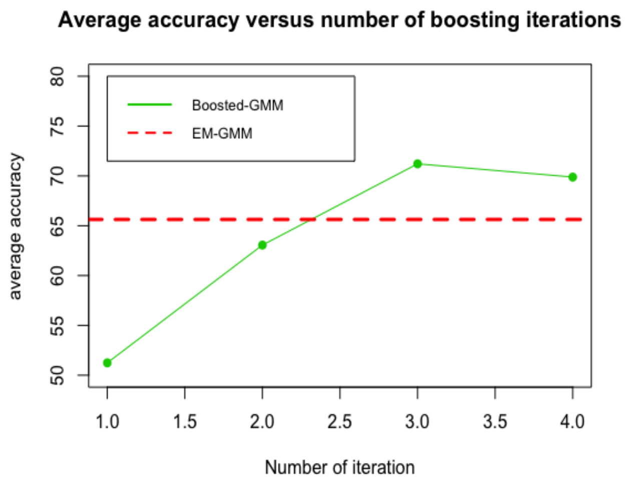


Figure 4: Average accuracy VS number of boosting iterations

5 Next Step and Conclusion

Actually, applying PCA to do the dimension reduction from 784 to 26 will result in losing a huge amount of detailed information so that the average accuracy of all four algorithms are not so satisfying. Also, we have not discuss about the choice of the number of mixtures, K, which is also a fundamental issue of GMM algorithm. For all the experiments above, we assume that we know the exact number of clusters, which may not true for other cases. H. Tang et al[7] shows that F-J algorithm will help to estimate the hyper parameter K. In addition, we may choose the Bayesian Information Criterion (BIC) [8] to solve the problem. BIC is a model selection criterion derived from the Laplace approximation [8]. The BIC value of a GMM can be defined as follows :

$$BIC(G|X) = \log P(X|\hat{G}) - \frac{d}{2} \log N \tag{5}$$

The first term of (5) is the log-likelihood term, where $\hat{G}$ represent the GMM with the ML parameter configuration. The second term is the model complexity penalty term where d represents the number of parameters in G and N is the size of the dataset. BIC selects the best GMM corresponding to the largest BIC value by trading of these two terms.

In this paper, we introduce the Boosted-GMM algorithm, which embeds the EM algorithm in a boosting framework and which can be used to reliably and accurately estimate the class-conditional probabilistic distributions in any pattern recognition problems based on a training data set. We do the simulation to prove that the algorithm We apply the Boosted-GMM algorithm to handwritten digits data set and our experiments show that the recognition rates are effectively and significantly boosted by the Boosted-GMM algorithm as compared to the EM-GMM algorithm due to the fact that boosting can lead to more accurate estimates of the class-conditional GMMs.

References

[1] W. L. Goh, D. P. Mital and H. A. Babri, ”An artificial neural network approach to handwriting recognition,” Knowledge-Based Intelligent Electronic Systems, 1997. KES ’97. Proceedings., 1997 First International Conference on, Adelaide, SA, 1997, pp. 132-136 vol.1.

[2] Y. Freund, R. Schapire. Experiments With a New Boosting Algorithm. Proc. of the 13th ICML. (1996)

[3] L. Brieman. Arcing Classifiers. The Annals of Statistics, 26(3): 801-849. (1998)

[4] L. Mason, J. Baxter, P. Bartlett, M. Frean. Boosting Algorithms as Gradient De- scent. Proc. NIPS99. (1999)

[5] S. Rosset, E. Segal. Boosting Density Estimation. Proc. NIPS02. (2002)

[6] L. Friedrich. Bagged Clustering. Working papers SFB Adaptive Information Sys- tems and Modeling in Economics and Management Science, no.51. (1999)

[7] H. Tang, S. M. Chu, M. Hasegawa-Johnson and T. S. Huang, ”Emotion recognition from speech VIA boosted Gaussian mixture models,” 2009 IEEE International Conference on Multimedia and Expo, New York, NY, 2009, pp. 294-297.

[8] G. Schwarz. Estimating the dimension of a model. Annuals of Statistics, 6. (1978)

Code

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
 Stat 201C
 Author: Zheqi Wu
 Date : 06/10/2018
 Description: boosting GMM
"""

import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.cross_validation import train_test_split
import random
from scipy import linalg
import itertools
from sklearn.mixture import GaussianMixture
import matplotlib as mpl
from scipy.stats import multivariate_normal
import pandas as pd
from scipy import stats


def plot_results(X, Y_, means, covariances, index, title):
    splot = plt.subplot(2, 1, 1 + index)
    for i, (mean, covar, color) in enumerate(zip(
            means, covariances, color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # as the DP will not use every component it has access to
        # unless it needs it, we shouldn't plot the redundant
        # components.
        if not np.any(Y_ == i):
            continue
        plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], .8, color=color)

        # Plot an ellipse to show the Gaussian component
        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi  # convert to degrees
        ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle, color=color)
        ell.set_clip_box(splot.bbox)
        ell.set_alpha(0.5)
        splot.add_artist(ell)

    plt.xlim(-0.5, 3.)
    plt.ylim(-0.5, 2.)
    plt.xticks(())
    plt.yticks(())
    plt.title(title)

random.seed(111)
color_iter = itertools.cycle(['gold','navy','darkorange', 'c', 'cornflowerblue'])
""""

""""


def prob_fun(g_mean,g_cov,X_train,k):
    P=np.reshape(np.zeros(X_train.shape[0]),(X_train.shape[0],1))
    for i in range(0,g_mean.shape[0]):
        p=multivariate_normal.pdf(X_train, mean=g_mean[i],cov=g_cov[i])
        P=np.c_[P,p]

    return (P[:,1:(k+1)])

def min_eta(G0,g,k):
    eta_=np.arange(0.01,1.01,0.01)
    i=0
    nelog_l=[]
    #object_f=np.zeros((eta_.shape[0],1))
    #g=prob_fun(gmm.means_, gmm.covariances_,gmm.predict(X_train),k)
    for e in eta_:
        l=np.sum(-np.log(np.multiply(e,g)+np.multiply((1-e),G0)))
        nelog_l.append(l)
        i=i+1

    t=np.where(nelog_l==min(nelog_l))
    eta_best=eta_[t[0]]
    return eta_best

def boot_weight(W, X_train):
    W_=[]

    for i in range(0,len(W)):
        a=W[i]
        if a>1e-10:
            W_.append(a)
        else:
            W_.append(0)
    W_=W_/np.sum(W_)
    X_index=np.random.choice(X_train.shape[0], X_train.shape[0], p=W_)
    X=X_train[X_index]

    return X,X_index

def tran_g(g,X_,X_train):
    g1=np.reshape(np.zeros(X_train.shape[0]*k),(X_train.shape[0],k))
    for i in range(0,X_train.shape[0]):
        if np.isin(X_[i],X_train):
            index=np.where(X_==X_train[i])[0][0]
            g1[index]=g[i]

    return g1

#k=10
def BoostGMM(X_train,Y_train,k):
    p=X_train.shape[1]
    #G0=np.reshape(np.zeros(X_train.shape[0]*k),(X_train.shape[0],k))
    W=np.ones(X_train.shape[0])
    gmm=GaussianMixture(n_components=k, covariance_type=’full’,random_state=1234).fit(X_train)
    G0=gmm.predict_proba(X_train)
    #G0=prob_fun(gmm.means_, gmm.covariances_,X_train,k)

    plot_results(X_train, gmm.predict(X_train), gmm.means_, gmm.covariances_, 0,’Gaussian Mixt
    W=W/sum(W)
    item=0

    while(item<100):
        X_,X_index=boot_weight(W,X_train)
        gmm=GaussianMixture(n_components=k, covariance_type=’spherical’,random_state=1234).fit
        #plot_results(X_, gmm.predict(X_), gmm.means_, gmm.covariances_, 0,’Gaussian Mixture’)

        #g1=prob_fun(gmm.means_, gmm.covariances_,X_,k)
        g1=gmm.predict_proba(X_)
        #g=g1[X_index]

        g=np.reshape(np.zeros(X_train.shape[0]*k),(X_train.shape[0],k))
        i=0
        count=np.zeros(X_train.shape[0])

        for i in range(0,len(X_index)):
            index=X_index[i]
            count[index]+=1
            g[index]=g[index]+g1[i]
            i+=1
        for i in range(0,g.shape[0]):
            if g[i][0]!=0:
                g[i]=g[i]*1/count[i]

        eta_=min_eta(G0,g,k)
        print(eta_)
        G1=np.multiply(eta_,g)+np.multiply((1-eta_),G0)
```

```
            LG1=np.sum(-np.log(np.multiply(eta_,g)+np.multiply((1-eta_),G0)))
            LG0=np.sum(-np.log(G0+0.000001))
            print(LG1)
            print(LG0)
            #if LG1<LG0:
             #     break
            #Z=np.sum(1/np.sum(G1,axis=1))
            W=1/(1/np.sum(G1,axis=1))
            W=W/np.sum(W)
            ##update
            G0=G1
            print(G0)
            LG0=LG1
            print(G1)
            item+=1

    return G0

'''
## simulation
n=200
k=3
mu = [[0,0.5], [1,1], [2,0.5]]
sigma = [[0.1,0.1], [0.25,0.25], [0.15,0.15]]
X1=[]
X2=[]
for i in range(0,k):
    for j in range(0,n):
     x1 = np.random.normal(mu[i][0], sigma[i][0])
     x2 = np.random.normal(mu[i][1], sigma[i][1])
     X1.append(x1)
     X2.append(x2)
label=np.r_[np.ones(n)*0,np.ones(n)*1,np.ones(n)*2]
X = np.c_[X1,X2,label]

X_=X[np.random.choice(X.shape[0],X.shape[0])]
X_train=X_[0:X_.shape[0],0:2]
Y_train=X_[0:X_.shape[0],2]
#X_train, X_test= train_test_split(X_, test_size=0.2)



g=BoostGMM(X_train,Y_train,k=3)
#g=G0
label_result=[]
for i in range(0,X_train.shape[0]):
    label=np.argmax(g[i])
    label_result.append(label)


#label_result=gmm.predict(X_train)
result=np.c_[label_result,Y_train]
df = pd.DataFrame(data=result)
df=df.sort_values(by=[0,1], ascending=True, na_position='first').values
accu=0
for i in range(0,k):
    C=df[df[:,0]==i,0:2]
    print(stats.mode(C[:,1])[0])
    accu=accu+np.sum(C[:,1]==stats.mode(C[:,1])[0])

accu=float(accu)/len(label_result)
'''
color = np.array(['gold','navy','darkorange', 'c', 'cornflowerblue'])
label_result=np.array(label_result)
for i in range(0,3):
    plt.scatter(X_train[label_result == i,0], X_train[label_result==i,1], .8, color=color[i])

#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 12 15:30:23 2018

@author: wuzheqi
"""

from sklearn import datasets
import sys
sys.path.append("/Users/cliccuser/Downloads")
from bgmm import *

'''
iris = datasets.load_iris()
X_train = iris.data[:, :4]  # we only take the first two features.
Y_train = iris.target

'''
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_mldata
import matplotlib
from sklearn.decomposition import PCA


mnist = fetch_mldata('MNIST original')
X = mnist["data"]
y = mnist["target"]

pca = PCA(n_components = 0.7, svd_solver = 'full')
pca.fit(X)
pca_result = pca.transform(X)
X=pca_result

X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.8)



def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = matplotlib.cm.binary, **options)
    plt.axis("off")

plt.figure(figsize=(9,9))
example_images = np.r_[X[:12000:600], X[13000:30600:600], X[30600:60000:590]]
plot_digits(example_images, images_per_row=10)
save_fig("more_digits_plot")
plt.show()


#X_train, X_test, y_train, y_test = train_test_split(X, y)

def accu_(label_result,Y_train,k=3):
    result=np.c_[label_result ,Y_train]
    df = pd.DataFrame(data=result)
    df=df.sort_values(by=[0,1], ascending=True, na_position='first').values
    accu=0
    for i in range(0,k):
        C=df[df[:,0]==i,0:2]
        print(stats.mode(C[:,1])[0])
        accu=accu+np.sum(C[:,1]==stats.mode(C[:,1])[0])

    accu=float(accu)/len(label_result)
    return accu

## B_GMM
g=BoostGMM(X_train ,Y_train ,k=10)
#g=G0
label_result=[]
for i in range(0,X_train.shape[0]):
    label=np.argmax(g[i])
    label_result.append(label)
accu_BGMM=accu_(label_result ,Y_train)

#label_result=gmm.predict(X_train)



## GMM
k=10
gmm=GaussianMixture(n_components=k, covariance_type='full',random_state=1234).fit(X_train)
plot_results(X_train, gmm.predict(X_train), gmm.means_, gmm.covariances_, 0,'Gaussian Mixture'
a=gmm.predict(X_train)
accu_GMM=accu_(gmm.predict(X_train),Y_train)
```