GMM

Variational Base

$$q^{\star}(\boldsymbol{\pi}) = \mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha})$$

where,

$$\alpha_k = \alpha_0 + N_k$$
$$\alpha_0 = \sum_{k=1}^{K} \alpha_{0k}$$
$$N_k = \sum_{n=1}^{N} r_{nk}$$

And,

$$\ln q^*(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) = \mathcal{N}\left(\boldsymbol{\mu}_k|\mathbf{m}_k, (\beta_k\boldsymbol{\Lambda}_k)^{-1}\right)\mathcal{W}(\boldsymbol{\Lambda}_k|\mathbf{W}_k, \nu_k)$$

where,

$$\beta_k = \beta_0 + N_k$$
$$\mathbf{m}_k = \frac{1}{\beta_k}\left(\beta_0\mathbf{m}_0 + N_k\bar{\mathbf{x}}_k\right)$$
$$\mathbf{W}_k^{-1} = \mathbf{W}_0^{-1} + N_k\mathbf{S}_k + \frac{\beta_0 N_k}{\beta_0 + N_k}\left(\bar{\mathbf{x}}_k - \mathbf{m}_0\right)\left(\bar{\mathbf{x}}_k - \mathbf{m}_0\right)^{\mathrm{T}}$$
$$\nu_k = \nu_0 + N_k$$

$$\bar{\mathbf{x}}_k = \frac{1}{N_k}\sum_{n=1}^{N} r_{nk}\mathbf{x}_n$$
$$\mathbf{S}_k = \frac{1}{N_k}\sum_{n=1}^{N} r_{nk}\left(\mathbf{x}_n - \bar{\mathbf{x}}_k\right)\left(\mathbf{x}_n - \bar{\mathbf{x}}_k\right)^{\mathrm{T}}$$

In [1]:

```
import matplotlib.cm as cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import math
from scipy.special import digamma, gammaln
from scipy.special import gamma as Gam
```

In [2]:

```
class VariationalGaussianMixture(object):

    def __init__(self, n_cluster=10, alpha0=1.):
        self.n_cluster = n_cluster
        self.alpha0 = alpha0

    def init_params(self, X):
        np.random.seed(0)
        self.N, self.D = X.shape
        self.alpha0 = np.ones(self.n_cluster) * self.alpha0
        self.m0 = np.zeros(self.D)
        self.W0 = np.eye(self.D)
        self.nu0 = self.D
        self.beta0 = 1.

        self.Nk = self.N / self.n_cluster + np.zeros(self.n_cluster)
        self.alpha = self.alpha0 + self.Nk
        self.beta = self.beta0 + self.Nk
        indices = np.random.choice(self.N, self.n_cluster, replace=False)
        self.m = X[indices].T
        self.W = np.tile(self.W0, (self.n_cluster, 1, 1)).T
        self.nu = self.nu0 + self.Nk

    def fit(self, X, iter_max=100):
        self.init_params(X)
        record = []
        for i in range(iter_max):
            #params = np.hstack([array.flatten() for array in self.get_params()])
            gamma = self.e_like_step(X)
            L = self.lower_bound(X, gamma)
            self.m_like_step(X, gamma)
            #L = self.lower_bound(X, gamma)
            if i % 100 == 0:
                print("iter: %d, lower bound: %s" % (i, L))
                record.append([i, L])
            #if np.allclose(params, np.hstack([array.ravel() for array in self.get_params()])):
            if i == 0:
                oldL = L
            else:
                if L - oldL < 1e-5:
                    print("converge")
                    break
                else:
                    oldL = L
        else:
            print("parameters may not have converged")
        print("last step: %d, lower bound: %f"%(i, L))
        record.append([i, L])
        return gamma, np.array(record)

    def e_like_step(self, X):
        d = X[:, :, None] - self.m
        gauss = np.exp(
            -0.5 * self.D / self.beta
            -0.5 * self.nu * np.sum(
                np.einsum('ijk,njk->nik', self.W, d) * d,
                axis=1)
```

```
    60
pi = np.exp(digamma(self.alpha) - digamma(self.alpha.sum()))
Lambda = np.exp(digamma(self.nu - np.arange(self.D)[:, None]).sum(axis=0) + self.D * np.log(2) + np.lina
gamma = pi * np.sqrt(Lambda) * gauss
gamma = gamma / np.sum(gamma, axis=1, keepdims=True)
gamma[np.isnan(gamma)] = 1. / self.n_cluster
gamma[gamma<1e-10] = 1e-10
return gamma

def m_like_step(self, X, gamma):
self.Nk = gamma.sum(axis=0)
Xm = X.T.dot(gamma) / self.Nk
d = X[:, :, None] - Xm
S = np.einsum('nik,njk->ijk', d, gamma[:, None, :] * d) / self.Nk
self.alpha = self.alpha0 + self.Nk
self.beta = self.beta0 + self.Nk
self.m = (self.beta0 * self.m0[:, None] + self.Nk * Xm) / self.beta
d = Xm - self.m0[:, None]
self.W = np.linalg.inv(
    np.linalg.inv(self.W0)
    + (self.Nk * S).T
    + (self.beta0 * self.Nk * np.einsum('ik,jk->ijk', d, d) / (self.beta0 + self.Nk)).T).T
self.nu = self.nu0 + self.Nk

def lower_bound(self, X, gamma):
Nk = np.sum(gamma, axis=0)
means = gamma.T @ X / Nk[:, None]
S = np.zeros([self.n_cluster, self.D, self.D])
for k in range(self.n_cluster):
    for n in range(self.N):
        S[k] += gamma[n, k] * np.einsum("i, j -> ij", (X[n] - means[k]), (X[n] - means[k]))
log_det = self.e_log_det()
log_pi = self.e_log_pi(self.alpha)

L1 = 0
for k in range(self.n_cluster):
    L1 += Nk[k]*(log_det[k])-self.D/self.beta[k] - self.nu[k] * np.trace(S[k]@self.W[:, :, k]) - self.nu[k]*(m
L1 = L1/2

L2 = np.sum(log_pi @ gamma.T)

L3 = (self.alpha0[0]-1)*np.sum(log_pi) + self.logC(self.alpha0)

L4 = 0
dammy = 0
for k in range(self.n_cluster):
    L4 += self.D*np.log(self.beta0/(2*math.pi)) + log_det[k] + self.D*self.beta0/self.beta[k] - self.beta0*s
    dammy += self.nu[k]*np.trace(np.linalg.inv(self.W0)@self.W[:, :, k])
L4 = L4/2 + self.n_cluster*np.log(self.logB(self.W0, self.nu0)) + (self.nu0-self.D-1)*np.sum(log_det)/2 -

L5 = np.sum(gamma*np.log(gamma))

L6 = np.sum((self.alpha-1)*log_pi) + self.logC(self.alpha)

L7 = 0
for k in range(self.n_cluster):
    L7 += log_det[k]/2 + self.D*np.log(self.beta[k]/(2*math.pi))/2 - self.D/2 - self.H(self.W[:, :, k], self.nu

L = L1 + L2 + L3 + L4 + L5 + L6 + L7

```

```
    return L

def logC(self, alpha):
    return gammaln(np.sum(alpha)) - np.sum(gammaln(alpha))

def logB(self, W, nu):
    "scaler"
    tmp1 = 0
    for i in range(self.D):
        tmp1 += gammaln((nu-i)/2)
    tmp2 = nu*np.linalg.det(W)/2 + (nu*self.D/2)*np.log(2) + self.D*(self.D-1)*np.log(math.pi)/4 + tmp1
    return tmp2


def H(self, W, nu, log_det):
    "scaler"
    tmp1 = 0
    for i in range(self.D):
        tmp1 += gammaln((nu-i)/2)
    tmp2 = nu*np.linalg.det(W)/2 + (nu*self.D/2)*np.log(2) + self.D*(self.D-1)*np.log(math.pi)/4 + tmp1
    return  tmp2 - (nu-self.D-1)*log_det/2 + nu*self.D/2


def E_log_det(self):
    tmp = np.zeros([self.n_cluster])
    E_log_lam = np.empty([self.N, self.n_cluster])
    for i in range(self.D):
        tmp += digamma((self.nu-i)/2)
    log_lam = tmp + self.D*np.log(2) + np.log([np.linalg.det(w) for w in self.W.T])
    return E_log_lam


def E_log_pi(self, alpha):
    log_pi = np.empty([self.n_cluster])
    for k in range(self.n_cluster):
        log_pi[k] = digamma(alpha[k]) - digamma(np.sum(alpha))
    return log_pi
```

In [3]:

```
X = np.loadtxt("x.csv", delimiter=",")
vb = VariationalGaussianMixture(6, 0.001)
gamma, record = vb.fit(X, 1000)
```

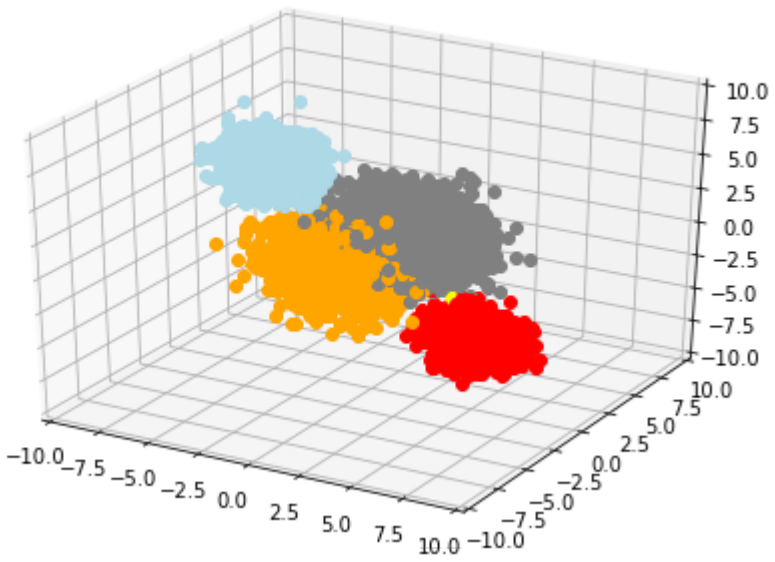
```
/Users/daigofujiwara/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6
4: RuntimeWarning: invalid value encountered in true_divide

iter: 0, lower bound: -278626163.9272527
iter: 100, lower bound: -148474.3425807393
iter: 200, lower bound: -148251.25441444753
iter: 300, lower bound: -148046.52474518432
iter: 400, lower bound: -147810.8235231234
iter: 500, lower bound: -147471.19879409083
converge
last step: 507, lower bound: -147459.101514
```

In [4]:

```
np.savetxt("z.csv", gamma, delimiter=",")
with open("params.dat", "w") as f:
    f.write("alpha:\n")
    for k in range(vb.n_cluster):
        f.write("cluster %d: %s\n" % (k, vb.alpha[k]))
    f.write("m:\n")
    for k in range(vb.n_cluster):
        f.write("cluster %d: %s\n" % (k, vb.m[:, k]))
    f.write("W:\n")
    for k in range(vb.n_cluster):
        f.write("cluster %d:\n%s\n" % (k, vb.W[:, :, k]))
    f.write("nu\n")
    for k in range(vb.n_cluster):
        f.write("cluster %d: %s\n" % (k, vb.nu[k]))
    f.write("beta:\n")
    for k in range(vb.n_cluster):
        f.write("cluster %d: %s\n" % (k, vb.beta[k]))
with open("vblikelihood.txt", "w") as f:
    f.write("iter\tlower bound\n")
    for i in range(len(record)):
        f.write("%d\t%f\n" % (record[i, 0], record[i, 1]))

# plot
labels = np.argmax(gamma, axis=1)
colors = ["red", "lightblue", "lightgreen", "orange", "yellow", "gray"]
label_color = [colors[int(label)] for label in labels]
fig = plt.figure()
ax = Axes3D(fig)
for i in range(X.shape[0]):
    ax.plot([X[i, 0]], [X[i, 1]], [X[i, 2]], "o", color=label_color[i])
    ax.set_xlim(-10, 10)
    ax.set_ylim(-10, 10)
    ax.set_zlim(-10, 10)
plt.savefig("vb.png")
plt.show()
plt.close()
```



In [ ]:

```
1
```