Part 1

Create 2D data from 3 gaussians

In [1]:

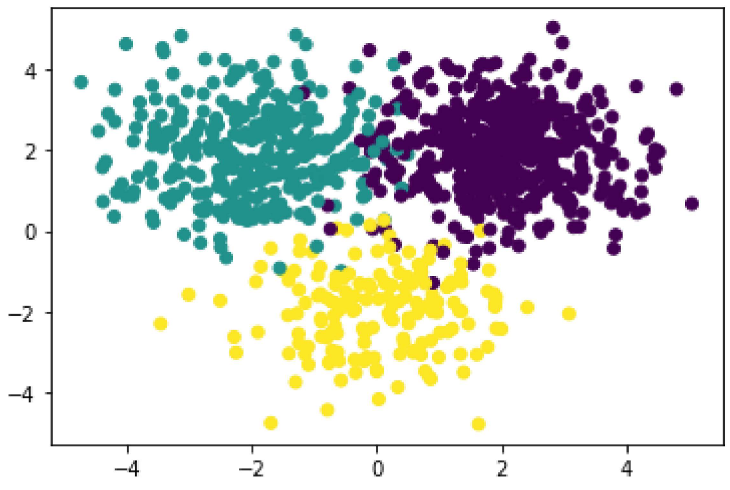
```
import numpy as np
import matplotlib.pyplot as plt
from math import dist
from scipy import stats
from sklearn import metrics
```

In [2]:

```
# constants
mu0 = (2, 2)
mu1 = (-2, 2)
mu2 = (0, -2)
mu = (mu0, mu1, mu2)
v0 = ((1.1, 0), (0, 1.1))
v1, v2 = v0, v0
v = (v0, v1, v2)
a = (1/2, 1/3, 1/6)

# generate data
data = []
true_labels = []
for i in range(1000):
    g = np.random.choice([0, 1, 2], p=a)
    true_labels.append(g)
    data.append(np.random.multivariate_normal(mu[g], v[g]))
data = np.array(data)

# plot data
x, y = zip(*data)
plt.scatter(x, y, c=true_labels)
```

Out[2]:

```
<matplotlib.collections.PathCollection at 0x21c5aa7c760>
```



K means

In [3]:

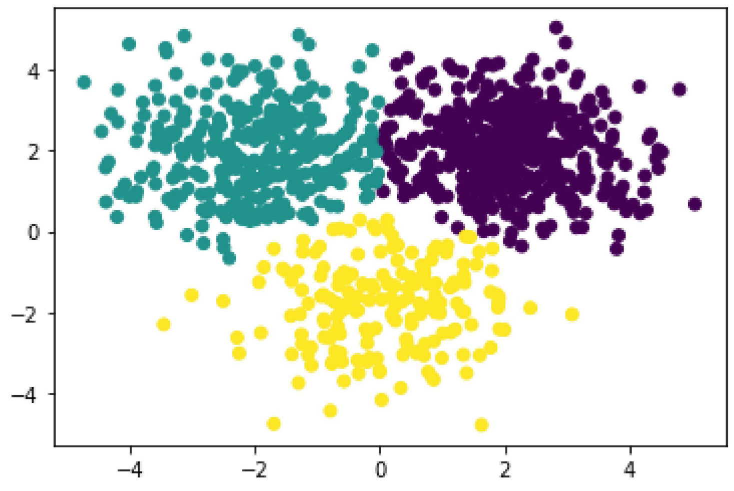
```
mu = [(0, 1), (0, 0), (0, -1)]  # to start with
k_labels = np.zeros(1000)
success_rate = []
for i in range(10):
    for j in range(1000):
        k_labels[j] = np.argmin([dist(data[j], mu[i]) for i in range(3)])
    mu = [np.sum(data[k_labels==i], axis=0)/len(data[k_labels==i]) for i in range(3)]
    success_rate.append(metrics.adjusted_rand_score(true_labels, k_labels))
```

In [4]:

```
plt.scatter(x, y, c=k_labels)
```

Out[4]:

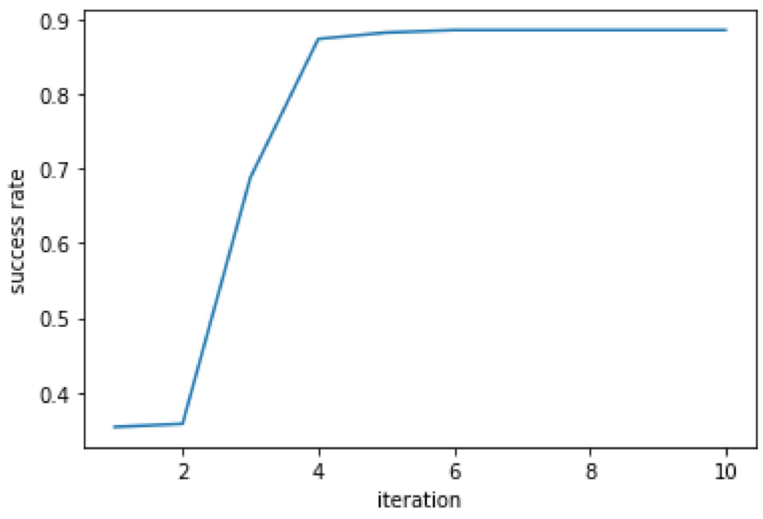
```
<matplotlib.collections.PathCollection at 0x21c5736e2e0>
```



In [5]:

```
print('Success rate:', success_rate[9])
print('Centroids:', mu)
plt.plot(range(1,11), success_rate)
plt.xlabel('iteration')
plt.ylabel('success rate')
plt.show()
```

```
Success rate: 0.8859085885347433
Centroids: [array([2.06709039, 2.04872675]), array([-1.9939682 ,  2.0048439
4]), array([ 0.06492663, -1.81216153])]
```



EM - MOG

In [6]:

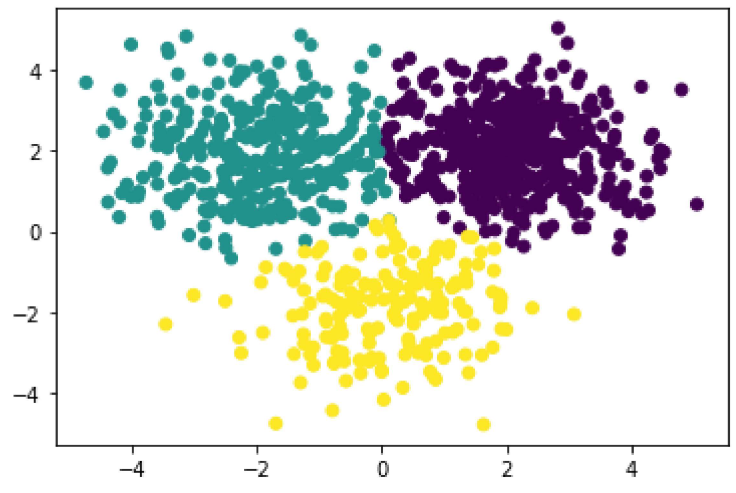
```
# random start values
mu = [(4, 4), (-4, 4), (0, -4)]
v = [ ((1, 0), (0, 1)), ((1, 0), (0, 1)), ((1, 0), (0, 1)) ]
a = [1/3, 1/3, 1/3]

w_ti = np.zeros((1000,3))
em_labels = np.zeros(1000)
success_rate = []

for i in range(25):

    # get w_ti
    for j in range(1000):
        w_ti[j] = [a[i]*stats.multivariate_normal.pdf(data[j], mu[i], v[i]) for i in range(
        row_sum = sum(w_ti[j])
        w_ti[j] = [w_ti[j,i] / row_sum for i in range(3)]

    # update parameters
    a = 1/1000 * sum(w_ti)
    for i in range(3):
        mu[i] = [0,0]
        for t in range(1000):
            mu[i] += w_ti[t,i] * data[t]
        mu[i] /= sum(w_ti)[i]

    for i in range(3):
        v[i] = [[0,0],[0,0]]
        for t in range(1000):
            v[i] += w_ti[t,i] * ((data[t]-mu[i])[:, np.newaxis] * (data[t]-mu[i]))
        v[i] /= sum(w_ti)[i]

    em_labels = np.array([np.argmax([w_ti[t,0], w_ti[t,1], w_ti[t,2]]) for t in range(1000)
    success_rate.append(sum(em_labels == true_labels)/len(data))
```

In [7]:

```
plt.scatter(x, y, c=em_labels)
```

Out[7]:

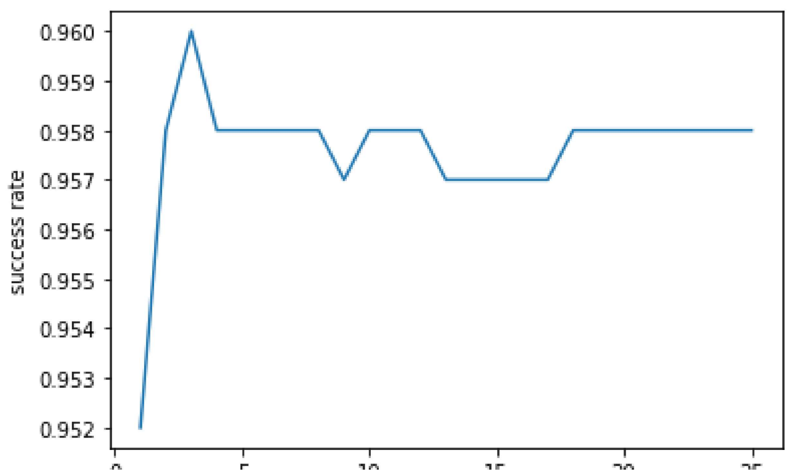
```
<matplotlib.collections.PathCollection at 0x21c5bbd3520>
```



In [8]:

```
print('success rate:', success_rate[24])
print('mean:', mu)
print('var:', v)
print('a:', a)
plt.plot(range(1,26), success_rate)
plt.xlabel('iteration')
plt.ylabel('success rate')
plt.show()
```

```
var: [array([[ 0.93217867, -0.10165358],
       [-0.10165358,  0.91626243]]), array([[ 1.30754704, -0.04280758],
       [-0.04280758,  1.25381673]]), array([[1.20254623, 0.16523675],
       [0.16523675, 1.17781715]])]
a: [0.48264863 0.34317713 0.17417424]
```



EM perform better than K means

Part 2

In [9]:

```
from scipy.io import loadmat
import numpy as np
from PIL import Image
```

In [10]:

```
# get data to np array
data_dict = loadmat('mnist_all.mat')
test_l = ['test' + str(i) for i in range(10)]
test_d = tuple([data_dict[l] for l in test_l])
data2 = np.concatenate(test_d, axis=0)
true_labels = []
for i in range(10):
    true_labels += [i] * len(data_dict['test' + str(i)])
```

In [11]:

```
mu = [data_dict['test'+str(i)][0] for i in range(10)]  # to start with
k_labels = np.zeros(len(data2))
cost = []
for k in range(10):
    for j in range(len(data2)):
        k_labels[j] = np.argmin([dist(data2[j], mu[i]) for i in range(10)])
    mu = [np.sum(data2[k_labels==i], axis=0)/len(data2[k_labels==i]) for i in range(10)]
    cost.append(np.sum([(data2[i]-mu[int(k_labels[i])])**2 for i in range(len(data2))]))
```

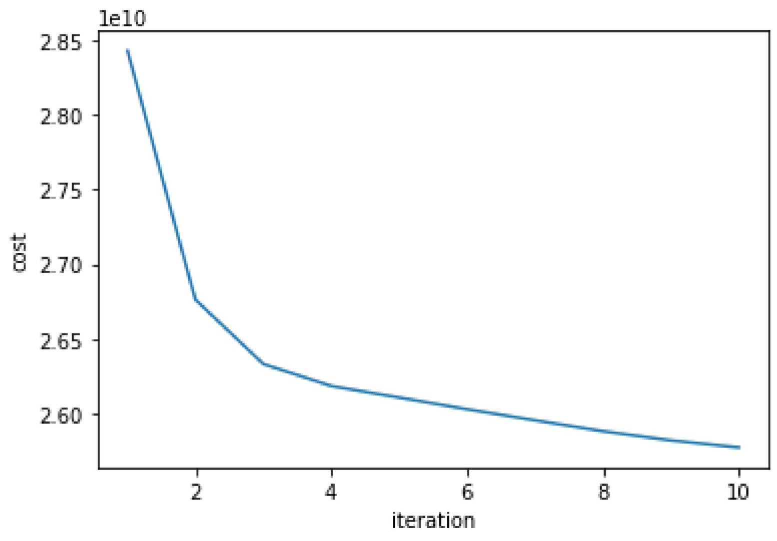
In [12]:

```
print('success rate:', metrics.adjusted_rand_score(true_labels, k_labels))
```

```
success rate: 0.33578356096064205
```

In [14]:

```
plt.plot(range(1,11), cost)
plt.xlabel('iteration')
plt.ylabel('cost')
plt.show()
```



Show centroids

In [15]:

```
image = Image.fromarray(mu[3].reshape(28,28))
image.show()
```

Try PCA / T-SNE and then KMeans

In [16]:

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
```

In [17]:

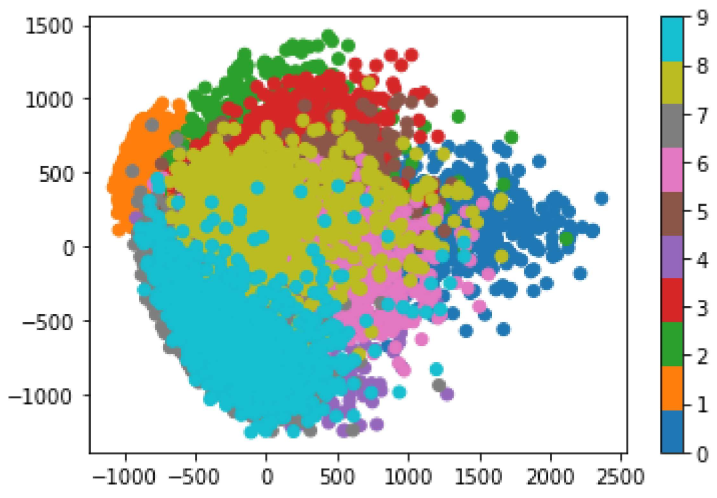
```
pca = PCA(n_components=2)
pca.fit(data2)
df1 = pca.transform(data2)
tsne = TSNE(n_components=2)
df2 = tsne.fit_transform(data2)
```

In [18]:

```
plt.scatter(df1[:,0], df1[:,1], c=true_labels, cmap='tab10')
plt.colorbar()
```

Out[18]:

```
<matplotlib.colorbar.Colorbar at 0x21c5e1c1460>
```

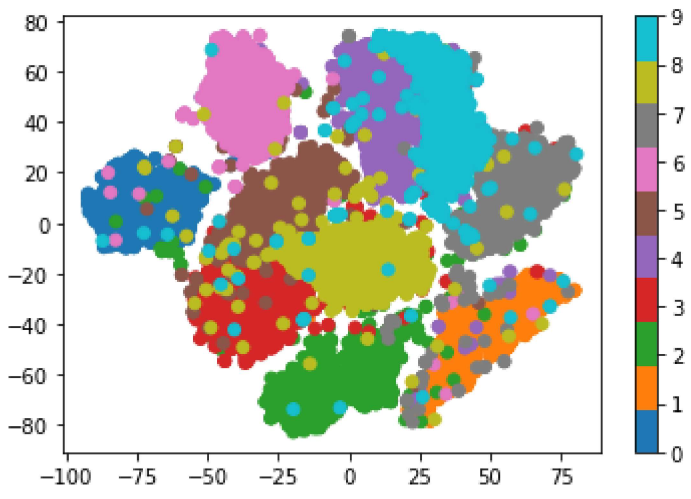


In [19]:

```
plt.scatter(df2[:,0], df2[:,1], c=true_labels, cmap='tab10')
plt.colorbar()
```

Out[19]:

```
<matplotlib.colorbar.Colorbar at 0x21c646cd130>
```



In [20]:

```
kmeans1 = KMeans(n_clusters=10, random_state=0).fit(data2)
kmeans2 = KMeans(n_clusters=10, random_state=0).fit(df2)
print('original:', metrics.adjusted_rand_score(true_labels, kmeans1.labels_))
print('after tsne:', metrics.adjusted_rand_score(true_labels, kmeans2.labels_))
```

```
original: 0.38099011781379183
after tsne: 0.720604748685594
```