Project : Kmeans Clustering Implementation

Student Name : Wenjie Duan

In [1]:
```
from kmeans import *

from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import SpectralClustering
from sklearn.datasets import make_blobs
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from collections import Counter
```
executed in 666ms, finished 17:21:09 2020-02-03

Blob Clustering

- From the first graph with 2 subplots, we can see that kmeans++ method give better output than centroids=None method.

- From the second graph, we can see that kmeans++ method can really make sparse initial centroids. That's where it works better than normal method.

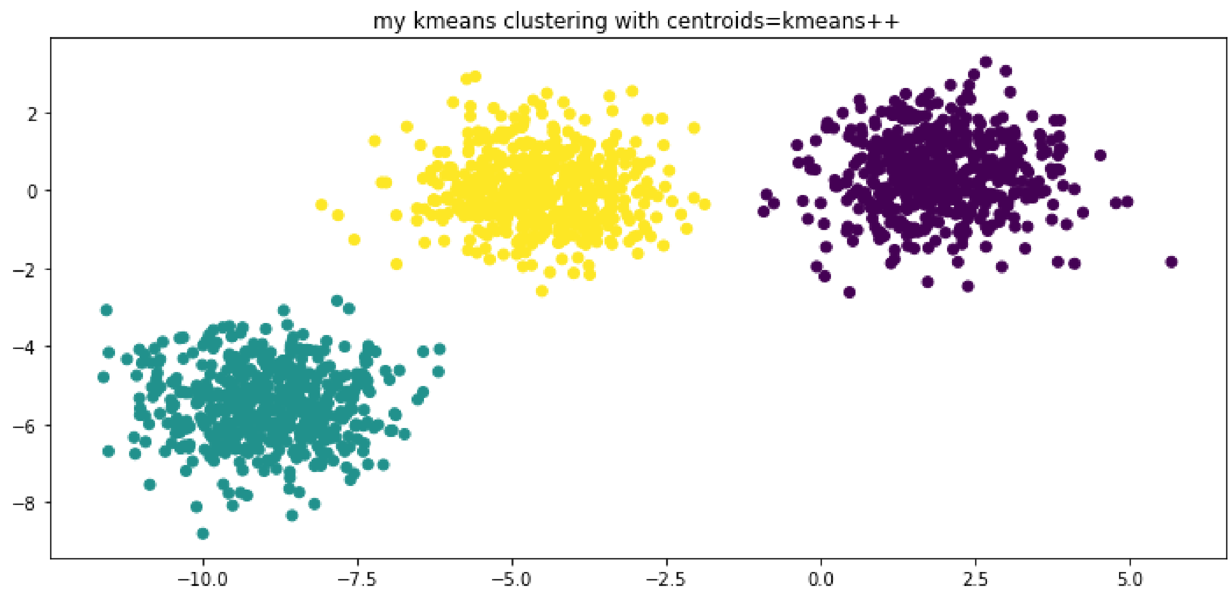
In [2]:
```
# kmeans++ has sparse initial centroids
k = 3
X, y = make_blobs(n_samples=1500, random_state=170)
```
executed in 3ms, finished 17:21:09 2020-02-03

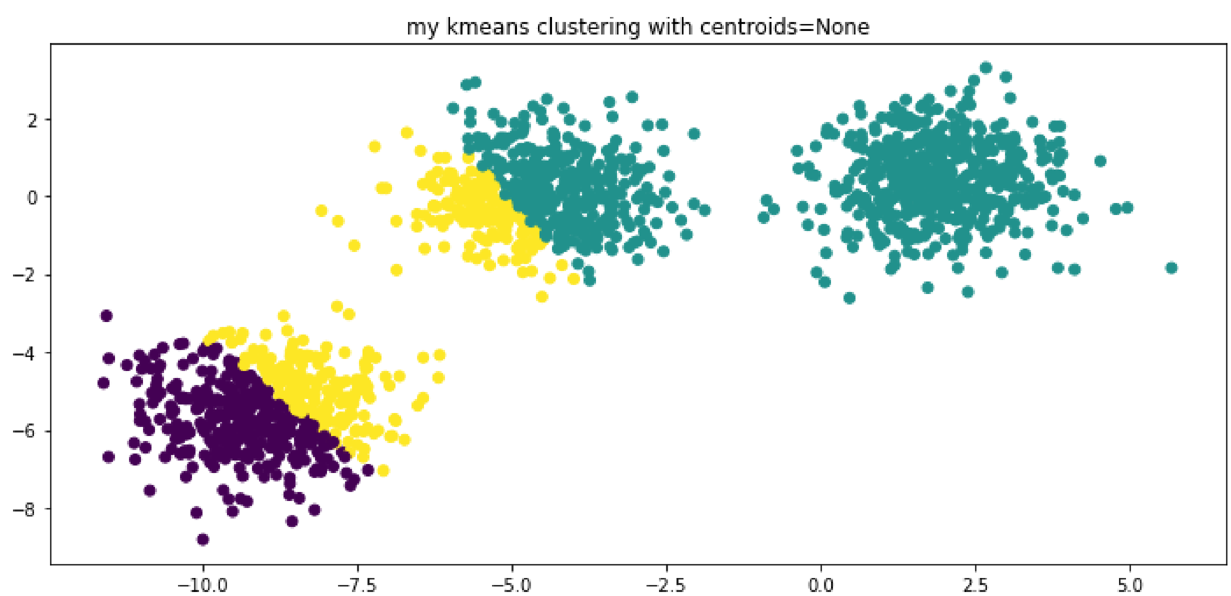
In [3]:
```python
# my prediction
results = []
for c in ['kmeans++',None]:
    centroids, clusters = kmeans(X=X,k=k,centroids=c)
    y_pred = np.zeros([len(X),1])
    for cluster_number in range(k):
        y_pred[clusters[cluster_number]] = cluster_number
    y_pred = y_pred.reshape(1,-1)[0]
    results.append(y_pred)

plt.figure(figsize=(12, 12))
plt.subplot(211)
plt.scatter(X[:, 0], X[:, 1], c= results[0])
plt.title('my kmeans clustering with centroids=kmeans++')
plt.subplot(212)
plt.scatter(X[:, 0], X[:, 1], c= results[1])
plt.title('my kmeans clustering with centroids=None')
```

executed in 632ms, finished 17:21:10 2020-02-03

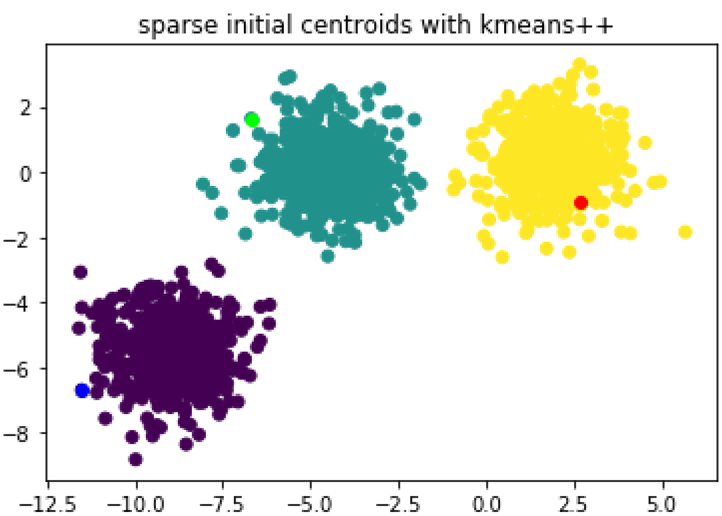
Out[3]:
```
Text(0.5, 1.0, 'my kmeans clustering with centroids=None')
```





In [4]:
```
# kmeans++ initialization gives sparse initial centroids
centroids = init_centroids(X=X,k=k) # init sparse centroids, not randomly i
plt.scatter(X[:, 0], X[:, 1], c= y)
plt.scatter(centroids[0,0],centroids[0,1],c='red')
plt.scatter(centroids[1,0],centroids[1,1],c='blue')
plt.scatter(centroids[2,0],centroids[2,1],c='lime')
plt.title('sparse initial centroids with kmeans++ ')
```
executed in 260ms, finished 17:21:10 2020-02-03

Out[4]:
```
Text(0.5, 1.0, 'sparse initial centroids with kmeans++ ')
```



Cancer or Not : Binary Prediction

- In this kind of case, kmeans clustering can be used for classification, normally with good accuracy.

- Here, it's important to link our cluster number with its correct group, we should flip_over predictions when necessary.

In [5]:
```
cancer = load_breast_cancer()
X = cancer.data
y = cancer.target
print('{:.2%} observations has y_true=1'.format(y[y==1].shape[0]/y.shape[0]
# y=0 cancer, y=1 benign
```

executed in 11ms, finished 17:21:10 2020-02-03

```
62.74% observations has y_true=1
```

- How to identify which cluster centroids should be associated with which true labels.

- **METHOD 1**:
  - The Basic idea, is to identify each cluster's size.
  - Sort values in y by its counts, then associate clusters in y_pred with labels by finding its rank in value_counts.
  - To be more specific, in this case, the bigger cluster would be assigned lable=1.
  - Pros: can deal with multi_classification problems.
  - Cons: when clusters' size are similar, it may fail to find its correct label.

In [6]:
```python
def sort_by_counts(y):
    # return a list of tuples, like [(1, 357), (0, 212)]
    return sorted(Counter(y).items(),key=lambda x:x[1],reverse=True)

def flip_over_if_necessary(y,y_pred):
    sorted_y = sort_by_counts(y)
    sorted_y_pred = sort_by_counts(y_pred.reshape(1,-1)[0])
    # similar distribution, same order in counts
    if [item for item,count in sorted_y] == [item for item,count in sorted_
        return y_pred
    else:
        y_pred_new = y_pred
        for i in range(len(sorted_y)):
            y_pred_new[clusters[int(sorted_y_pred[i][0])]] = sorted_y[i][0]
        return y_pred_new

for i in range(1,11):
    print('Test Number {} '.format(i))
    k = 2
    centroids, clusters = kmeans(X, k)

    y_pred = np.zeros([len(X), 1])
    for cluster_number in range(k):
        y_pred[clusters[cluster_number]] = cluster_number

    accuracy_before = accuracy_score(y, y_pred.reshape(1, -1)[0])
    print('Accuracy before flip_over : \n {:.4f}'.format(accuracy_before))

    # flip_over according to the value_counts distribution
    y_pred = flip_over_if_necessary(y,y_pred)

    accuracy = accuracy_score(y, y_pred.reshape(1, -1)[0])
    print('Accuracy : \n {:.4f}'.format(accuracy))
    if accuracy<0.5:
        print('Failed !!! Because predicted clusters have similar sizes !!!
    print('Confusion Matrix : \n {} \n'.format(confusion_matrix(y, y_pred))
```

executed in 262ms, finished 17:21:10 2020-02-03

```
Test Number 1
Accuracy before flip_over :
 0.8225
Accuracy :
 0.1775
Failed !!! Because predicted clusters have similar sizes !!!
Confusion Matrix :
 [[  6 206]
 [262  95]]

Test Number 2
Accuracy before flip_over :
 0.1880
Accuracy :
 0.8120
Confusion Matrix :
 [[105 107]
 [  0 357]]
```

```
Test Number 3
Accuracy before flip_over :
 0.8576
Accuracy :
 0.8576
Confusion Matrix :
 [[132  80]
 [  1 356]]

Test Number 4
Accuracy before flip_over :
 0.7786
Accuracy :
 0.7786
Confusion Matrix :
 [[ 86 126]
 [  0 357]]

Test Number 5
Accuracy before flip_over :
 0.1916
Accuracy :
 0.8084
Confusion Matrix :
 [[103 109]
 [  0 357]]

Test Number 6
Accuracy before flip_over :
 0.8946
Accuracy :
 0.8946
Confusion Matrix :
 [[157  55]
 [  5 352]]

Test Number 7
Accuracy before flip_over :
 0.1125
Accuracy :
 0.8875
Confusion Matrix :
 [[195  17]
 [ 47 310]]

Test Number 8
Accuracy before flip_over :
 0.8998
Accuracy :
 0.8998
Confusion Matrix :
 [[162  50]
 [  7 350]]

Test Number 9
Accuracy before flip_over :
 0.1424
```

```
Accuracy :
 0.8576
Confusion Matrix :
 [[132  80]
 [  1 356]]

Test Number 10
Accuracy before flip_over :
 0.8875
Accuracy :
 0.8875
Confusion Matrix :
 [[186  26]
 [ 38 319]]
```

- **METHOD 2**:
  - The basic idea, is to investigate the performace of clustering.
  - To be more specific, in this case, if accuracy<0.5, we flip_over the prediction, and recalculate the metrics.
  - In this case, method 2 works better than method 1, because 62.74% observations has y_true=1. When the clusters' size don't differ that much, method 1 would fail to implement flip_over, even though it's really necessary.

In [7]:
```
accuracies_kmeans = []

for i in range(1, 6):
    print('Test Number {} '.format(i))
    k = 2
    centroids, clusters = kmeans(X, k)

    y_pred = np.zeros([len(X), 1])
    for cluster_number in range(k):
        y_pred[clusters[cluster_number]] = cluster_number

    accuracy_before = accuracy_score(y, y_pred.reshape(1, -1)[0])
    if accuracy_before<0.5: # simple-judge
        print('Accuracy before flip_over : \n {:.4f}'.format(accuracy_befor
        y_pred = np.where(y_pred==1,0,1) # flip it

    accuracy = accuracy_score(y, y_pred.reshape(1, -1)[0])
    accuracies_kmeans.append(accuracy)
    print('Accuracy : \n {:.4f}'.format(accuracy))
    print('Confusion Matrix : \n {} \n'.format(confusion_matrix(y, y_pred))

print('In Average, Accuracy_kmeans = {:.4f}'.format(np.mean(accuracies_kmea
```

executed in 131ms, finished 17:21:11 2020-02-03

```
Test Number 1
Accuracy :
 0.8752
Confusion Matrix :
 [[144  68]
 [  3 354]]

Test Number 2
Accuracy :
 0.8120
Confusion Matrix :
 [[105 107]
 [  0 357]]

Test Number 3
Accuracy before flip_over :
 0.1916
Accuracy :
 0.8084
Confusion Matrix :
 [[103 109]
 [  0 357]]

Test Number 4
Accuracy before flip_over :
 0.1265
Accuracy :
 0.8735
Confusion Matrix :
 [[142  70]
 [  2 355]]

Test Number 5
```

```
Accuracy before flip_over :
 0.1459
Accuracy :
 0.8541
Confusion Matrix :
 [[130  82]
 [  1 356]]

In Average, Accuracy_kmeans = 0.8446
```

Image Compression

- Greyscale : set the points that has similar grey_value to a single (centroid) value.

In [8]:
```
image_path = './north-africa-1940s-grey.png'
img = np.array(Image.open(image_path))
# the greyscale value given by img[i][j]
```
executed in 15ms, finished 17:21:11 2020-02-03

In [9]:
```
display(Image.fromarray(img))
```
executed in 41ms, finished 17:21:11 2020-02-03

In [10]:
```
X = img.reshape(-1,1)
# Number of cluster k : how many grey_values are chosen.
k = 4
centroids, clusters = kmeans(X,k,centroids='kmeans++')
# get the four centroids, many points grey_value are around these centroids
centroids = centroids.astype(np.uint8)

y_pred = np.zeros([len(X), 1])
# convert all the grey_values to the 4 centroids
for cluster_number in range(k):
    y_pred[clusters[cluster_number]] = centroids[cluster_number]
```
executed in 33.9s, finished 17:21:45 2020-02-03

In [11]:
```
y_pred = y_pred.reshape(img.shape)
grey_img = y_pred.astype(np.uint8)
display(Image.fromarray(grey_img))
```
executed in 14ms, finished 17:21:45 2020-02-03

- Color : set the points that have similar color to a single (centroid) color.

In [12]:
```
image_path_2 = './parrt-vancouver.jpg'
img_2 = np.array(Image.open(image_path_2))
# img_2[i][j] = arr(red,gree,blue), a color
display(Image.fromarray(img_2))
```
executed in 76ms, finished 17:21:45 2020-02-03

In [13]:
```
X = img_2.reshape(-1,1,3) # 3 for red, greeb, blue
k = 32

centroids, clusters = kmeans(X,k,centroids= 'kmeans++')
centroids = centroids.astype(np.uint8)
y_pred = np.zeros([len(X), 3])
# convert all the colors to the 32 colors
for cluster_number in range(k):
    y_pred[clusters[cluster_number]] = centroids[cluster_number]
```
executed in 13m 47s, finished 17:35:31 2020-02-03

In [14]:
```
y_pred = y_pred.reshape(img_2.shape[0],img_2.shape[1],3)
clor_img = y_pred.astype(np.uint8)
display(Image.fromarray(clor_img))
```
executed in 23ms, finished 17:35:31 2020-02-03

Advanced: Spectral Clustering

- Spectral_Clustering has higher average accuracy than Kmeans_Clustering in the cancer_or_not classification problem.

In [15]:
```
cancer = load_breast_cancer()
X = cancer.data
y = cancer.target
```
executed in 10ms, finished 17:35:31 2020-02-03

In [16]:
```python
accuracies_spectral = []

for i in range(1,6): # different random_state in rf for each i
    print('Test Number {}'.format(i))
    rf= RandomForestClassifier() # different random_state
    rf.fit(X,y)
    S = similarity_matrix(X,rf)
    cluster = SpectralClustering(n_clusters=2, affinity='precomputed')
    y_pred = cluster.fit_predict(S) # pass similarity matrix not X
    if accuracy_score(y,y_pred)<0.5: # might needs to filp_over
        y_pred = np.where(y_pred==1,0,1)
    accuracy = accuracy_score(y, y_pred)
    accuracies_spectral.append(accuracy)
    print('Accuracy : \n {:.4f}'.format(accuracy))
    print('Confusion Matrix : \n {} \n'.format(confusion_matrix(y, y_pred))

print('In Average, Accuracy_spectral = {:.4f}'.format(np.mean(accuracies_sp
```

executed in 7.52s, finished 17:35:39 2020-02-03

```
Test Number 1
Accuracy :
 0.9754
Confusion Matrix :
 [[204   8]
 [  6 351]]

Test Number 2
Accuracy :
 0.9719
Confusion Matrix :
 [[203   9]
 [  7 350]]

Test Number 3
Accuracy :
 0.9736
Confusion Matrix :
 [[203   9]
 [  6 351]]

Test Number 4
Accuracy :
 0.9754
Confusion Matrix :
 [[204   8]
 [  6 351]]

Test Number 5
Accuracy :
 0.9754
Confusion Matrix :
 [[204   8]
 [  6 351]]

In Average, Accuracy_spectral = 0.9743
```

In [17]:
```
print('In Average, Accuracy_kmeans = {:.4f}'.format(np.mean(accuracies_kmea
print('In Average, Accuracy_spectral = {:.4f}'.format(np.mean(accuracies_sp
```
executed in 3ms, finished 17:35:39 2020-02-03

```
In Average, Accuracy_kmeans = 0.8446
In Average, Accuracy_spectral = 0.9743
```