KMeans

November 13, 2020

k-means

219

beccohov.a@yandex.ru, GitHub link

0.1

[1]:
```
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from IPython.display import Video
from sklearn import datasets
import plotly.express as px
import plotly.offline as pyo
from matplotlib.animation import ArtistAnimation
import plotly.graph_objects as go
```

0.2 , ,

0.3

[2]:
```
class KMeans:

    def __init__(self, k, metric = np.linalg.norm, max_iterations = 10,␣
↪precision = 0.01):
        #                          .                                    -
        # 1)              (                        precision)
        # 2)                                 (                      )
        # 3)                       -
        self.k = k
        self.metric = metric
        self.max_iterations = max_iterations
        self.groups = None
        self.precision = precision
        self.error = np.inf
```

```
        self.fitted = False

    def closest_center(self, points):
        #                        ,
        #        ,
        distances = self.metric(points - self.centroids[:,np.newaxis], axis = 2)
        mins = np.argmin(distances, axis = 0)
        self.error = np.min(distances,axis = 1).sum() / points.shape[0]
        return mins

    def move_clusters(self, points, closest, centers):
        #                   "       "

        return np.array([points[closest==k].mean(axis=0) for k in range(self.
↪k)])


    def fit_transform(self, X, initial_clusters = None):
        #         ,
        #
        #                              ,             ,
        #              ,           "    "     ,
        #

        clusters = []
        if not initial_clusters:
            mean = np.mean(X, axis = 0)
            std = np.std(X, axis = 0)
            clusters = np.random.randn(self.k,X.shape[1])*std + mean
            self.centroids = clusters
        else :
            self.k = initial_clusters.shape[0]
            self.centroisds = initial_clusters
        #               "    ",
        #                 "       "
        last_error = self.error

        for i in range(self.max_iterations):
            closest = self.closest_center(X)
            clusters = self.move_clusters(X, closest, clusters)
            self.centroids = clusters
            self.iterations_required = i
            if np.abs(last_error -  self.error) < self.precision :
                break #                           ,
        self.closest = closest #
        self.fitted = True
```

```
        return clusters
    def init_only(self, X, initial_clusters = None):
        #                          -
        #                         ,                 ,
        #                       API
        clusters = []

        if not initial_clusters:
            mean = np.mean(X, axis = 0)
            std = np.std(X, axis = 0)
            clusters = np.random.randn(self.k,X.shape[1])*std + mean
            self.centroids = clusters
        else :
            self.k = initial_clusters.shape[0]
            self.centroisds = initial_clusters

        self.fitted = True

    def make_k_steps(self, X, k_steps):
        #                                    .
        #     k_steps
        #                                 init_only(),
        #                                  part_train
        assert self.fitted == True, "Algorithm is unfitted yet!\n"
        last_error = self.error
        clusters = self.centroids
        closest = self.closest_center(X)
        for i in range(k_steps):
            closest = self.closest_center(X)
            clusters = self.move_clusters(X, closest, clusters)
            self.centroids = clusters
            self.iterations_required = i
        self.centroids = clusters
        self.closest = closest #
        return clusters

    def test_this(self, X, steps_to, initial_clusters = None):
        if not self.fitted:
            self.init_only(X, initial_clusters)
        self.make_k_steps(X, steps_to)
```

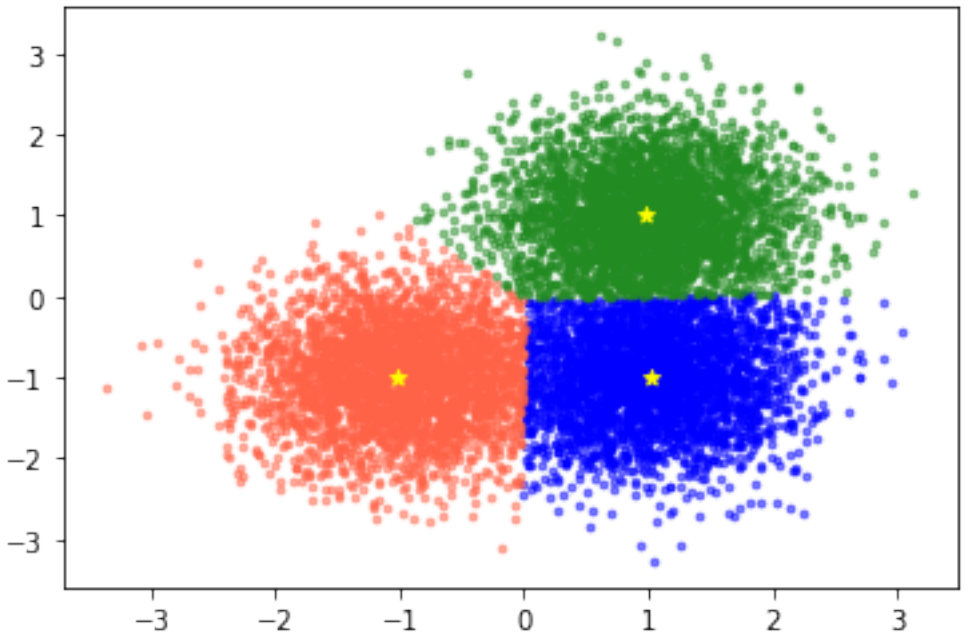
[3]:
```
#
#
colors_set = [
    'blue', 'tomato', 'forestgreen', 'darkorange', 'olivedrab',
    'navy', 'teal', 'darkviolet', 'dodgerblue', 'crimsone'
    ]
```

, ,

[4]:
```
centers = [[1, 1], [-1, -1], [1, -1]]
X, _ = datasets.make_blobs(n_samples=10000, centers=centers, cluster_std=0.6)
```

0.3.1

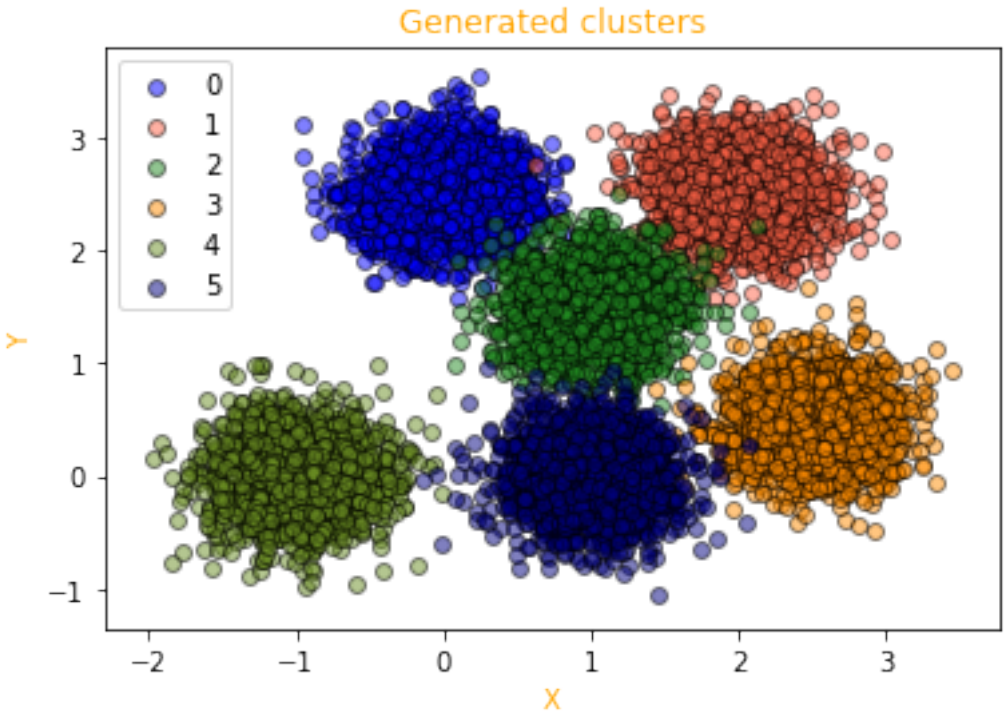
[5]:
```
km = KMeans(3)
km.fit_transform(X)
for i in range(3):
    plt.scatter(X[km.closest == i][:, 0], X[km.closest == i][:, 1], alpha = 0.
 ↪5, s= 7,color = colors_set[i], marker = 'o')
plt.scatter(km.centroids[:,0], km.centroids[:,1],color = 'yellow', marker = '*'␣
 ↪)
pass #                          "   "
```



[ ]:
```

```

[11]:
```
centers = [[0, 2.5], [2, 2.5], [1, 1.5], [2.5,0.5], [-1,0], [1,0]]
X, y = datasets.make_blobs(n_samples=10000, centers=centers, cluster_std=0.3)

for i in range(len(centers)):
```

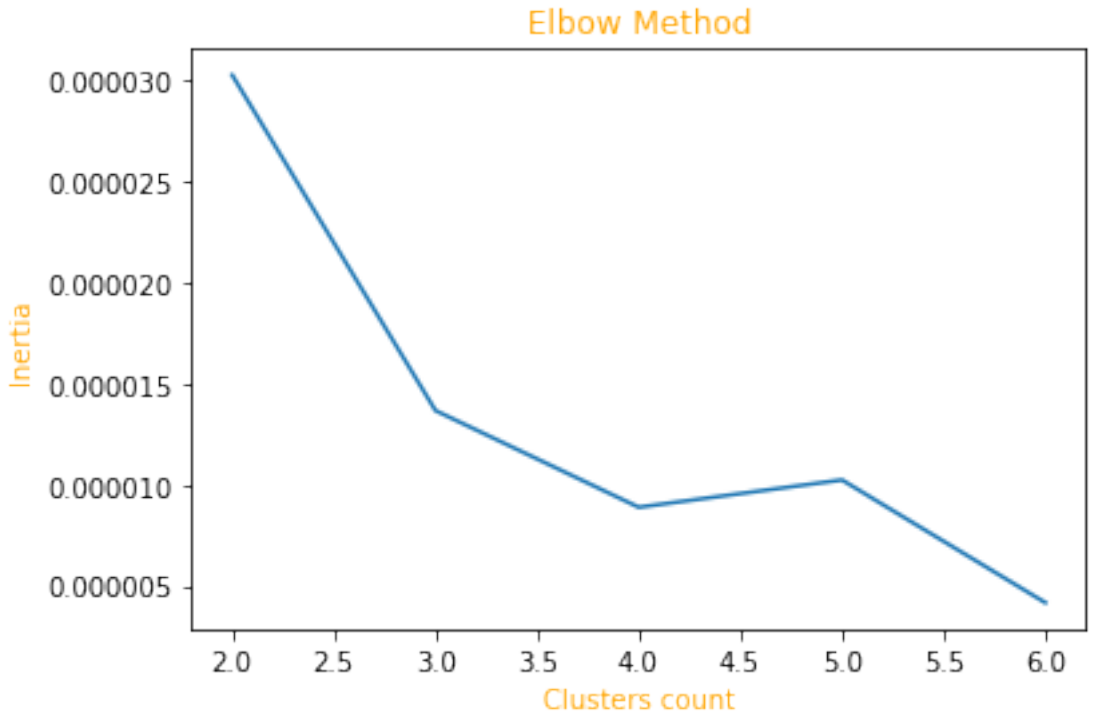
```
    plt.scatter(X[y == i][:,0], X[y == i][:,1],color = colors_set[i], alpha = 0.
↪5, marker = 'o', edgecolors = 'black')

plt.title("Generated clusters", color = 'orange')
plt.xlabel("X", color = 'orange')
plt.ylabel("Y", color = 'orange')
plt.legend([str(i) for i in range(len(centers))])
pass
```



“ ”

[21]:
```
clusters_range  = 6 #

errors = [] # "    "

for i in range(2, clusters_range + 1):
    ex = KMeans(i)
    ex.fit_transform(X)
    errors.append(ex.error)

plt.plot(range(2, clusters_range + 1), errors)
plt.xlabel("Clusters count", color = 'orange')
plt.ylabel("Inertia", color = 'orange')
```

```
plt.title("Elbow Method", color = 'orange')
pass #                          "   "
```



, ,

0.3.2 , ( )

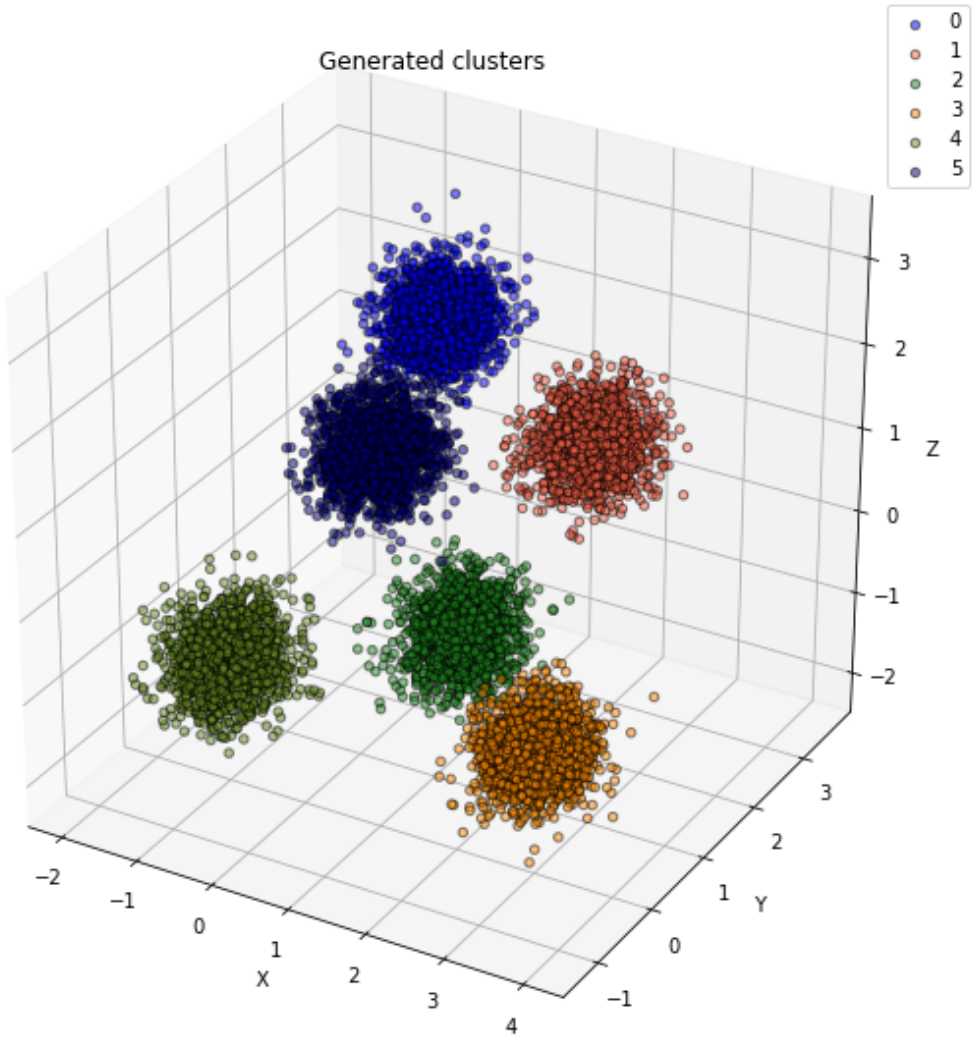
[22]:
```
fig, ax=plt.subplots()
container = [] #


clusters_count = 6 #
test_kmeans = KMeans(clusters_count) #
iter_count = 10 #

for i in range(iter_count):
    test_kmeans.test_this(X,1)
    state = [

            ax.scatter(X[test_kmeans.closest == k,0], X[test_kmeans.closest ==␣
↪k,1], marker = 'o', color = colors_set[k], alpha = 0.3, edgecolors = 'black')
```

```
            for k in range(clusters_count)

    ]
    state.append(ax.scatter(test_kmeans.centroids[:,0], test_kmeans.centroids [:
 ↪,1], color = 'yellow', marker = '*', s = 200, edgecolors = 'black' ))
    container.append(state)
ani = ArtistAnimation(fig, container, interval=500, blit=False)
ani.save("stuff/clusters01.mp4")
plt.close()
Video("stuff/clusters01.mp4")
```

[22]:
```
<IPython.core.display.Video object>
```

[ ]:
```

```

[24]:
```
centers_3d = [[0, 2.5, 2], [2, 2.5,1], [1, 1.5,-1], [3,0,-1], [-1,0,-1],␣
 ↪[1,0,2]]
X_3d, y_3d = datasets.make_blobs(n_samples=10000, centers=centers_3d,␣
 ↪cluster_std=0.3, n_features = 3)
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(111, projection='3d')
for i in range(len(centers_3d)):
    ax.scatter(X_3d[y_3d == i][:,0], X_3d[y_3d == i][:,1], X_3d[y_3d == i][:
 ↪,2], color = colors_set[i], alpha = 0.5, marker = 'o', edgecolors = 'black')

ax.set_title("Generated clusters")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.legend([str(i) for i in range(len(centers_3d))])
pass
```



, . , “ ” “ ” ,

[26]:
```
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(111, projection='3d')
container = [] #


clusters_count = 6 #
test_kmeans = KMeans(clusters_count) #
iter_count = 10 #

for i in range(iter_count):
    test_kmeans.test_this(X_3d,1)
```

```
    state = [
            ax.scatter(X_3d[test_kmeans.closest == k][:,0], X_3d[test_kmeans.
 ↪closest == k][:,1], X_3d[test_kmeans.closest == k][:,2], color =␣
 ↪colors_set[k], alpha = 0.05, marker = 'o',  zorder = -2 )
            for k in range(clusters_count)
           ]
    state.append(ax.scatter(test_kmeans.centroids[:,0], test_kmeans.centroids [:
 ↪,1], test_kmeans.centroids [:,2], color = 'yellow', marker = '*', ␣
 ↪edgecolors = 'black' , s = 800, alpha = 1, zorder = 2))
    container.append(state)
ani = ArtistAnimation(fig, container, interval=500, blit=False)
ani.save("stuff/clusters02.mp4")
plt.close()
Video("stuff/clusters02.mp4")
```

[26]: <IPython.core.display.Video object>

, , **k - means**

[18]:
```
x,y = datasets.make_moons(1000)
fig, ax=plt.subplots()
container = [] #


clusters_count = 2 #
test_kmeans = KMeans(clusters_count) #
iter_count = 10 #

for i in range(iter_count):
    test_kmeans.test_this(x,1)
    state = [

            ax.scatter(x[test_kmeans.closest == k,0], x[test_kmeans.closest ==␣
 ↪k,1], marker = 'o', color = colors_set[k], alpha = 0.3, edgecolors = 'black')
            for k in range(clusters_count)

    ]
    state.append(ax.scatter(test_kmeans.centroids[:,0], test_kmeans.centroids [:
 ↪,1], color = 'yellow', marker = '*', s = 200, edgecolors = 'black' ))
    container.append(state)
ani = ArtistAnimation(fig, container, interval=500, blit=False)
ani.save("stuff/clusters03.mp4")
plt.close()
Video("stuff/clusters03.mp4")
```

[18]: <IPython.core.display.Video object>

[ ]:

[27]:
```
#
colors_rgb = [
    'rgb(220, 20, 60)', 'rgb(0, 255, 127)', 'rgb(128, 128, 0)',
    'rgb(0, 139, 139)', 'rgb(255, 165, 0)', 'rgb(0, 0, 128)',
    'rgb(148, 0, 211)', 'rgb(255, 215, 0)', 'rgb(123, 104, 238)'
              ]
pyo.init_notebook_mode() #
```

[29]:
```
centers_3d = [[0, 2.5, 2], [2, 2.5,1], [1, 1.5,-1], [3,0,-1], [-1,0,-1],␣
↪[1,0,2], [1,1,1]]
X_3d, y_3d = datasets.make_blobs(n_samples= 5000, centers=centers_3d,␣
↪cluster_std=0.45, n_features = 3)

clusters_count = 7 #
test_kmeans = KMeans(clusters_count) #
iter_count = 10 #
test_kmeans.fit_transform(X_3d)


fig = go.Figure(data=[
    go.Scatter3d(
        x=X_3d[test_kmeans.closest == i][:,0],
        y=X_3d[test_kmeans.closest == i][:,1],
        z=X_3d[test_kmeans.closest == i][:,2], mode = 'markers',
        marker = {'color':colors_rgb[i]}, opacity = 0.10)
    for i in range(clusters_count)
] + [go.Scatter3d(
        x = test_kmeans.centroids[:,0],
        y = test_kmeans.centroids[:,1],
        z = test_kmeans.centroids[:,2],
        mode = 'markers', opacity = 1,
        marker = {'color' : 'rgb(255, 69, 0)', 'size' : 20, 'line' : {'width' :␣
↪2, 'color' : 'black'}}
    )]
)

fig.show()
#                    ,  .          WebGL
```

[31]:
```
import plotly.express as px
import plotly.offline as pyo
```

```
X_3d, y_3d = datasets.make_blobs(n_samples= 5000, centers = 8, cluster_std= 1,␣
 ↪n_features = 3)

clusters_count = 8 #
test_kmeans = KMeans(clusters_count) #
iter_count = 10 #
test_kmeans.fit_transform(X_3d)



fig = go.Figure(data=[
    go.Scatter3d(
        x=X_3d[test_kmeans.closest == i][:, 0],
        y=X_3d[test_kmeans.closest == i][:, 1],
        z=X_3d[test_kmeans.closest == i][:, 2], mode = 'markers',
        marker = {'color' : colors_rgb[i]}, opacity = 0.10)
    for i in range(clusters_count)
] + [go.Scatter3d(
        x = test_kmeans.centroids[:, 0],
        y = test_kmeans.centroids[:, 1],
        z = test_kmeans.centroids[:, 2],
        mode = 'markers', opacity = 1,
        marker = {'color' : 'rgb(255, 69, 0)', 'size' : 20, 'line' : {'width' :␣
 ↪2, 'color' : 'black'}}
     )]
)

fig.show()
#                      ,  .          WebGL
```

,   ,

[32]:
```
centers_3d = [[0, 2.5, 2], [2, 2.5,1], [1, 1.5,-1], [3,0,-1], [-1,0,-1],␣
 ↪[1,0,2], [1,1,1]]
X_3d, y_3d = datasets.make_blobs(n_samples= 5000, centers=centers_3d,␣
 ↪cluster_std=0.45, n_features = 3)

clusters_count = 7 #
def norm (x,axis):
    return np.linalg.norm(x, ord = np.inf,axis = axis)
test_kmeans = KMeans(clusters_count, metric= norm) #
iter_count = 10 #
test_kmeans.fit_transform(X_3d)


fig = go.Figure(data=[
```

```
    go.Scatter3d(
        x=X_3d[test_kmeans.closest == i][:,0],
        y=X_3d[test_kmeans.closest == i][:,1],
        z=X_3d[test_kmeans.closest == i][:,2], mode = 'markers',
        marker = {'color':colors_rgb[i]}, opacity = 0.10)
    for i in range(clusters_count)
] + [go.Scatter3d(
        x = test_kmeans.centroids[:,0],
        y = test_kmeans.centroids[:,1],
        z = test_kmeans.centroids[:,2],
        mode = 'markers', opacity = 1,
        marker = {'color' : 'rgb(255, 69, 0)', 'size' : 20, 'line' : {'width' :␣
 ↪2, 'color' : 'black'}}
     )]
)

fig.show()
#                      ,  .          WebGL
```

[33]:
```
#            N    ,        n_disks          R,                h
def generate_disks(N, R, h, n_disks):
    radius = R * np.sqrt(np.random.rand(N, 1))
    theta = 2 * np.pi * np.random.rand(N, 1)
    x = radius * np.cos(theta)
    y = radius * np.sin(theta)
    distribution_z = np.linspace(0, n_disks*h, n_disks)
    points = [(distribution_z[np.random.choice(range(distribution_z.
 ↪shape[0]))]+np.random.normal(0,1)) for i in range(N)]
    z = np.array(points).reshape(-1,1)
    return np.hstack([x, y, z])
```

[34]:
```
X_3d = generate_disks(5000,5,20,5) #

clusters_count = 5 #
def norm (x,axis):
    return np.linalg.norm(x, ord = np.inf,axis = axis)

test_kmeans = KMeans(clusters_count, metric= norm) #
iter_count = 10 #
test_kmeans.fit_transform(X_3d)




fig = go.Figure(data = [
    go.Scatter3d(
        x=X_3d[test_kmeans.closest == i][:, 0],
```

```
        y=X_3d[test_kmeans.closest == i][:, 1],
        z=X_3d[test_kmeans.closest == i][:, 2], mode = 'markers',
        marker = {'color' : colors_rgb[i]}, opacity = 0.10)
    for i in range(clusters_count)
] + [go.Scatter3d(
        x = test_kmeans.centroids[:,0],
        y = test_kmeans.centroids[:,1],
        z = test_kmeans.centroids[:,2],
        mode = 'markers', opacity = 1,
        marker = {'color' : 'rgb(255, 69, 0)', 'size':20, 'line' : {'width': 2,␣
↪'color' : 'black'}}

    )]
)

fig.show()
#                    ,  .          WebGL
```

0.3.3 Summary

(, , …)

, (“ ”) k - Means
. , , (DBSCAN,)

[ ]:

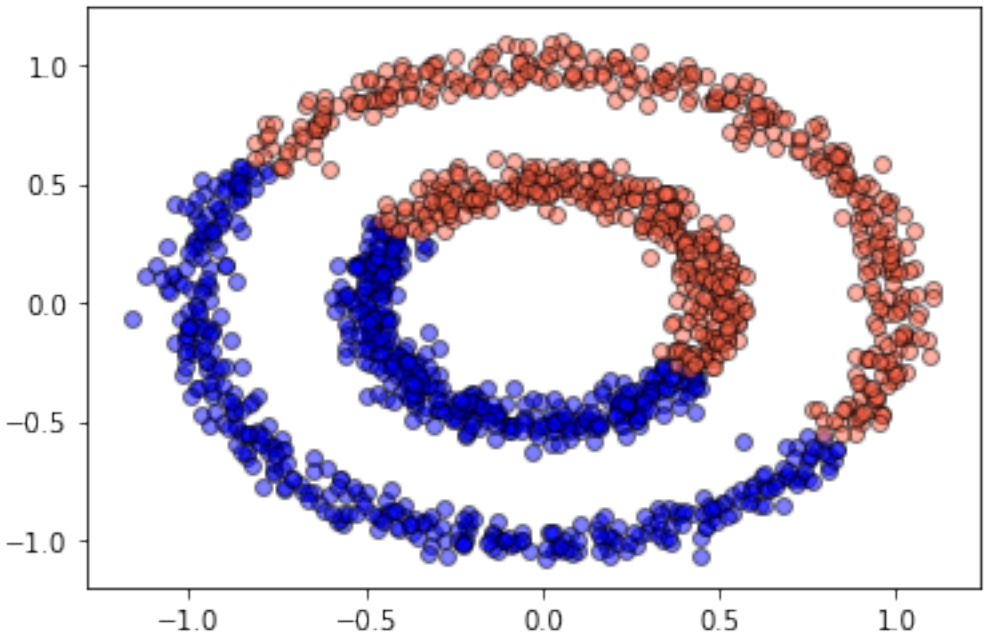
[ ]:

0.3.4 , k-means ,

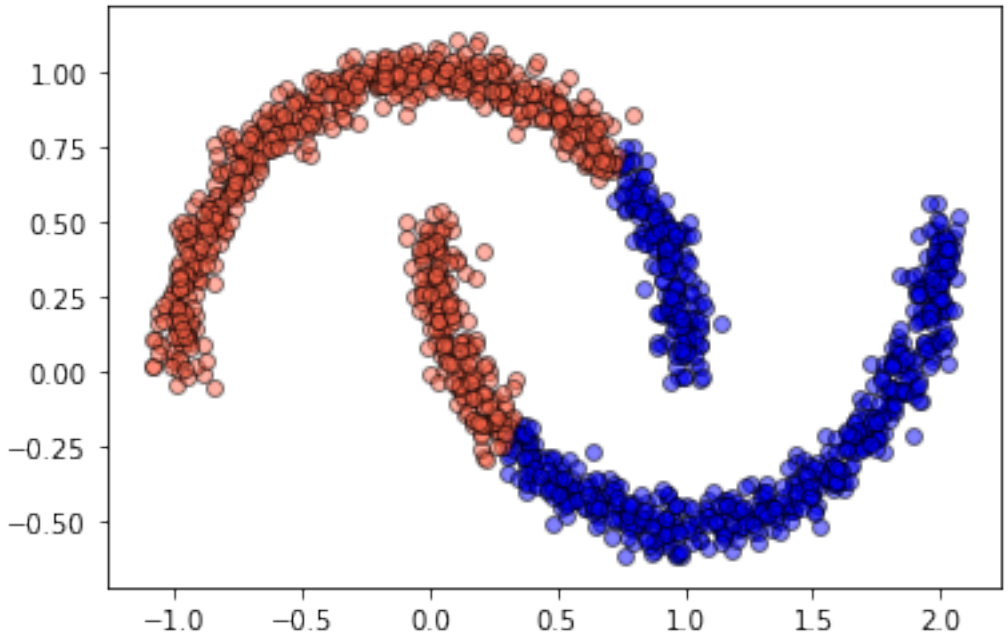
[30]:
```
n_samples = 1500
```

[33]:
```
noisy_circles = datasets.make_circles(n_samples=n_samples, factor=.5, noise=.05)
centers  = 2
kmeans = KMeans(2)
kmeans.fit_transform(noisy_circles[0])
for i in range(centers):
```

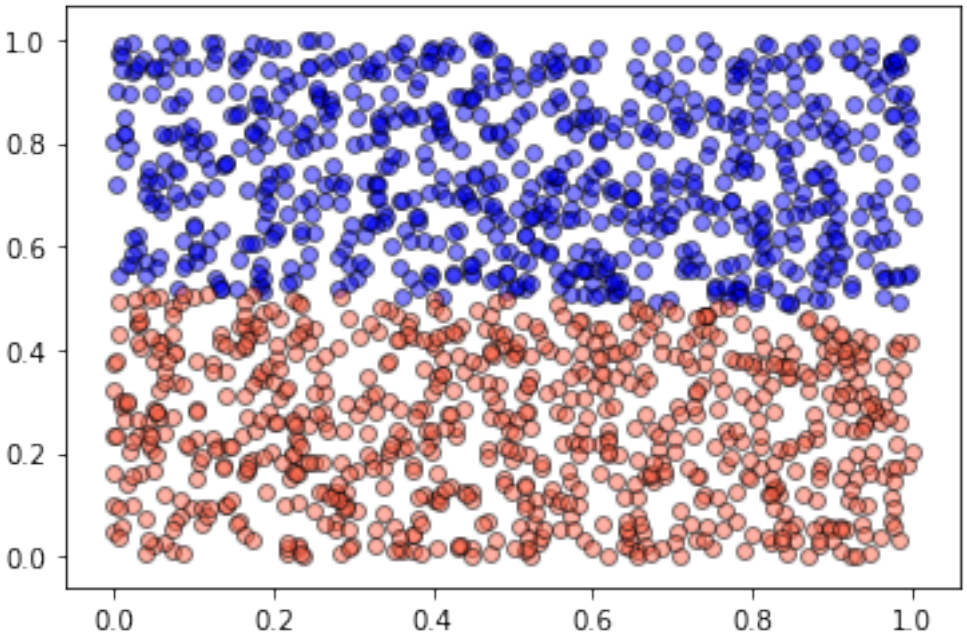
```
    plt.scatter(noisy_circles[0][kmeans.closest == i][:,0],␣
↪noisy_circles[0][kmeans.closest == i][:,1],color = colors_set[i], alpha = 0.
↪5, marker = 'o', edgecolors = 'black')
```



[34]:
```
noisy_moons = datasets.make_moons(n_samples=n_samples, noise=.05)
centers  = 2
kmeans = KMeans(2)
kmeans.fit_transform(noisy_moons[0])
for i in range(centers):
    plt.scatter(noisy_moons[0][kmeans.closest == i][:,0], noisy_moons[0][kmeans.
↪closest == i][:,1],color = colors_set[i], alpha = 0.5, marker = 'o',␣
↪edgecolors = 'black')
```



[35]:
```
#      ,                  .                ,                    .
#
no_structure = np.random.rand(n_samples, 2)
centers  = 2
kmeans = KMeans(2)
kmeans.fit_transform(no_structure)
for i in range(centers):
    plt.scatter(no_structure[kmeans.closest == i][:,0], no_structure[kmeans.
 ↪closest == i][:,1],color = colors_set[i], alpha = 0.5, marker = 'o',␣
 ↪edgecolors = 'black')
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

, “ , ”. ,

.

, .

[ ]:

[ ]:

[ ]: