+ コード + テキスト

RAM ディスク

k近傍法

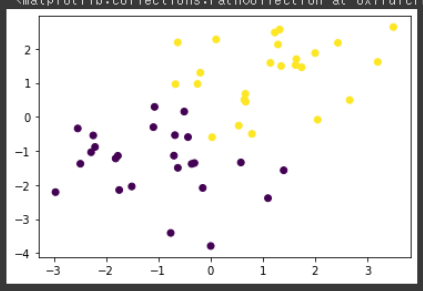
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

訓練データ生成

```
def gen_data():
  #50の要素を持つベクトルを2列に変換して各要素から-1
  x0 = np.random.normal(size=50).reshape(-1,2) - 1
  #50の要素を持つベクトルを2列に変換して各要素から+1
  x1 = np.random.normal(size=50).reshape(-1,2) + 1.

  x_train = np.concatenate([x0,x1])
  y_train = np.concatenate([np.zeros(25),np.ones(25)]).astype(np.int)
  return x_train,y_train

# r1,r2 = gen_data()
# print(np.mean(r1[0]))
# print(np.mean(r1[1]))
```

```
X_train, ys_train = gen_data()
print(np.mean(X_train[::,0]))
print(np.mean(X_train[::,1]))
# print(X_train)
plt.scatter(X_train[:,0], X_train[:,1], c=ys_train)
```

```
0.07510731065377 03
-0.04581431581136874
<matplotlib.collections.PathCollection at 0x7fd1cfff36d0>
```



学習：k近傍法において、「学習」相当の処理はなし

予測するデータ点との、距離が最も近いk個の、訓練データのラベルの最頻値を割り当てる。

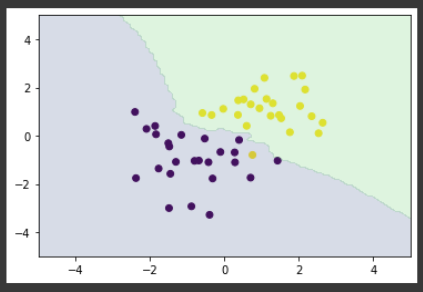
```
#引数1,2の「差の二乗」の合計を返却
def distance(x1, x2):
  return np.sum((x1 - x2)**2, axis=1)

#k近傍法予測
def knc_predict(n_neighbors, x_train, y_train, X_test):
```

0秒 完了時間: 21:50

```
  return np.sum((x1 - x2)**2, axis=1)

#k近傍法予測
def knc_predict(n_neighbors, x_train, y_train, X_test):
  ##### np.empty() #####
  #用途：未初期化の配列を生成する。
  #第１引数：shape
  #第２引数：要素のタイプ(初期値float)
  y_pred = np.empty(len(X_test), dtype=y_train.dtype) #X_testと同サイズの行列を初期化生成
  j = 0
  for i, x in enumerate(X_test):
    distances = distance(x, X_train)
    nearest_index = np.argsort(distances)[:n_neighbors]
    #scipy.stats.mode：最頻値（モード）
    mode, uu = stats.mode(y_train[nearest_index])
    y_pred[i] = mode
    # if i>5000 and i<6000:
    #   print(y_pred[i])
  return y_pred
```

```
23
24 def plt_result(x_train, y_train, y_pred):
25   xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
26   xx = np.array([xx0, xx1]).reshape(2, -1).T
27
28   plt.scatter(x_train[:,0], x_train[:,1], c=y_train)
29   plt.contourf(xx0, xx1, y_pred.reshape(100,100).astype(dtype=np.float), alpha=0.2, levels=np.linspace(0, 1, 3))
30
31
```

```
[5]  1 n_neighbors = 3
     2
     3 xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
     4 X_test = np.array([xx0, xx1]).reshape(2, -1).T
     5
     6 # print(np.array([xx0, xx1]))
     7 # print(np.array([xx0, xx1]).flatten())
     8 # print(np.array([xx0, xx1]).reshape(2,-1))
     9 y_pred = knc_predict(n_neighbors, X_train, ys_train, X_test)
    10 plt_result(X_train, ys_train, y_pred)
    11
```

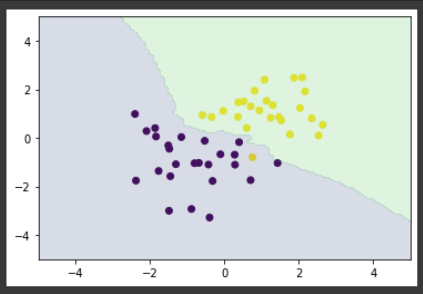


```
[6]  1 #sklにて実装
```

+ コード + テキスト

RAM ディスク

```
[6]  2 from sklearn.neighbors import KNeighborsClassifier
     3 knc = KNeighborsClassifier(n_neighbors).fit(X_train, ys_train)
     4 xx = np.array([xx0, xx1]).reshape(2, -1).T
     5 # print(xx)
     6 # xx = np.array([xx0, xx1]).reshape(-1, 2)
     7 # print(xx)
     8 plt_result(X_train, ys_train, knc.predict(xx))
     9
```



k平均クラスタリング(k-means)

```
[7]  1 %matplotlib inline
     2 import numpy as np
     3 import matplotlib.pyplot as plt
     4
```

データ作成

```
[8]  1 def gen_data():
     2   x1 = np.random.normal(size=(100, 2)) + np.array([-5, -5])
     3   x2 = np.random.normal(size=(100, 2)) + np.array([5, -5])
     4   x3 = np.random.normal(size=(100, 2)) + np.array([0, 5])
     5   #np.vstack:列方向に行列を結合（列数があってる必要ある。）
     6   #np.hstack:行方向に行列を結合（行数があってる必要ある。）
     7   return np.vstack((x1,x2,x3))
     8
     9 # a = np.array([5, -5])
    10 # b = np.random.normal(size=(100, 2))
    11 # c = a + b
    12
    13 # print(a)
    14 # print(b[:5])
```

```
# print(c[:5])
# print(gen_data().shape)
```

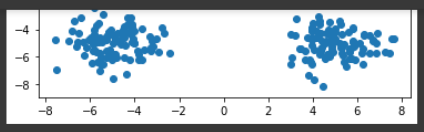
```
[9]  #データ作成
X_train = gen_data()
#データ描画
plt.scatter(X_train[:, 0], X_train[:, 1])
```

```
<matplotlib.collections.PathCollection at 0x7fd1d2159090>
```

+ コード + テキスト

RAM ディスク

[9]



学習(クラスタ中心の計算)

k-meansアルゴリズムは以下の通り。

1）各クラスタ中心の初期値を設定する。

2）各データ点に対して、各クラスタ中心との距離を計算し、最も距離が近いクラスタを割り当てる。

3）各クラスタの平均ベクトル（中心）を計算する。

4）収束するまで2，3の処理を繰り返す

```
[10]  def distance(x1, x2):
        #axis=1は集計する方向を示す。
        return np.sum((x1-x2)**2, axis=1)

      # a1 = np.array([[1,2,3],[4,5,6],[1,2,3],[4,5,6]])
      # a2 = np.array([[4,5,6],[1,2,3],[1,2,3],[4,5,6]])
      # np.sum((a1 - a2)**2, axis=0)

      n_clusters = 3
      iter_max = 100

      #1）各クラスタ中心の初期値を設定する。
      centers = X_train[np.random.choice(len(X_train), n_clusters, replace = False)]
      print("クラスタの初回中心(ランダム):")
      print(centers)
      print()
      for cnt in range(iter_max):
        #前回ループの各クラスタ中心を保存したうえで、centerの計算を開始
        prev_centers = np.copy(centers)
        D = np.zeros((len(X_train),n_clusters))

        #2-1）【中心再計算】各データ点に対して、各クラスタ中心との距離を再計算。
        #Dは各列に「3中心からの距離」を保持する。
        for i, x in enumerate(X_train):
          D[i] = distance(x, centers)

        #各データ点に、最も距離が近いクラスタを割り当て
        #argmin()は配列の最小値のインデックスを取得するもの
        #「D(距離配列)」から中心から最小距離を持つインデックスを選択し、リストとして取り出す。(0 or 1 or 2)
        #2-2）【再分類】最も距離が近いクラスタを割り当てる。
        cluster_index = np.argmin(D, axis=1)

        #各クラスタの中心を計算
        #3）各クラスタの平均ベクトル（中心）を再計算する。
        for k in range(n_clusters):
          index_k = cluster_index == k
          centers[k] = np.mean(X_train[index_k], axis=0)

        #ループ終了判定
        #前回ループのクラスタ中心と今回ループのクラスタ中心が同じであれば学習終了
        if np.allclose(prev_centers, centers):
          # print(cluster_index)
          # print(D)
          print(index_k)

          # print("最終ループ時の各点距離D:")
          # print(D)
```

+ コード + テキスト

RAM ディスク

```
[10]  # print("最終ループ時の各点距離D:")
      # print(D)
```

```
    # print("クラスタの最終中心(前回ループ):")
    # print(prev_centers)
    # print("クラスタの最終中心(今回ループ):")
    print(centers)
    # print("学習ループ回数 {0}回".format(cnt))

    break

# 収束する回数を観察すると、かなりばらつきがある。
# 2回で収束することもあるし、まれに20回かかることもある。



```

```
クラスタの初回中心(ランダム):
[[ 5.24258609 -3.64797256]
 [-5.76014887 -4.62902466]
 [-6.95262784 -4.85905037]]

[ True False  True False  True False False  True False  True False  True
 False False False  True False  True  True False  True False  True False
 False  True False False  True  True False False False False  True  True
 False False  True  True False  True False  True  True  True False  True
 False  True  True False  True  True  True  True False False False  True
 False  True False False False  True False False  True  True  True False
 False False  True  True  True False False False  True False  True  True
 False False False False  True False  True  True False False  True  True
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False]
[[ 2.48173039 -0.0319369 ]
 [-4.25830617 -4.52843155]
 [-5.74894409 -5.44363139]]
```

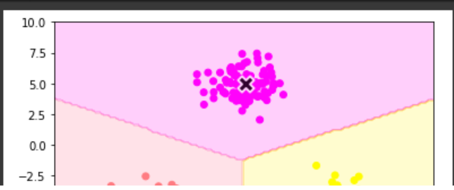
結果表示

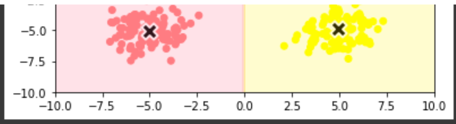
[11]
```
xx0, xx1 = np.meshgrid(np.linspace(-10, 10, 100),np.linspace(-10, 10, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

def plt_result(X_train, centers, xx, y_pred):
  #データを可視化
  plt.scatter(X_train[:, 0], X_train[:, 1], c=y_pred, cmap='spring')
```

+ コード + テキスト RAM ディスク

[25]
```
  #データを可視化
  plt.scatter(X_train[:, 0], X_train[:, 1], c=y_pred, cmap='spring')
  #中心を可視化
  plt.scatter(centers[:, 0], centers[:, 1], s=200, marker='X', lw=2, c='black', edgecolor='white')
  #領域の可視化
  pred = np.empty(len(xx), dtype=int)
  for i, x in enumerate(xx):
    d = distance(x, centers)
    pred[i] = np.argmin(d)
  plt.contourf(xx0, xx1, pred.reshape(100,100), alpha=0.2, cmap='spring')

```

[26]
```
# y_pred = np.empty(len(X_train), dtype=int)
# for i, x in enumerate(X_train):
#   d = distance(x, centers)
#   y_pred[i] = np.argmin(d)

#わざわざもう一回、X_trainをクラスタリング計算(y_predの算出)しない方法

# y_pred = cluster_index
plt_result(X_train, centers, xx, cluster_index)

```





+ コード + テキスト

RAM
ディスク

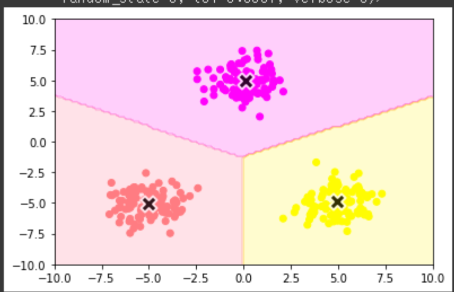
SKLで実装

[27]
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X_train)

print("labels:{}".format(kmeans.labels_))
print("cluster_centers:{}".format(kmeans.cluster_centers_))
kmeans.cluster_centers_

```

```
labels:[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0]
cluster_centers:[[ 0.08693337  4.93396956]
 [-5.02850621 -5.05362993]
 [ 4.97910161 -4.91153822]]
array([[ 0.08693337,  4.93396956],
       [-5.02850621, -5.05362993],
       [ 4.97910161, -4.91153822]])
```

[28]
```
plt_result(X_train, kmeans.cluster_centers_,xx,cluster_index)
kmeans.score
```

```
<bound method KMeans.score of KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)>
```



✓ 0 秒 完了時間: 21:50