演習：K-means (NumPy)

In [1]:

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

データ生成

In [2]:

```
def gen_data():
    # 100x2の行列を3つ作成する
    x1 = np.random.normal(size=(100, 2)) + np.array([-5, -5])  # 左下の領域に点を集める
    x2 = np.random.normal(size=(100, 2)) + np.array([5, -5])  # 右下の領域に点を集める
    x3 = np.random.normal(size=(100, 2)) + np.array([0, 5])  # 上の領域に点を集める
    X = np.vstack((x1, x2, x3))  # 縦方向に結合
    return X
```

In [3]:

```
# データ作成
X_train = gen_data()
```

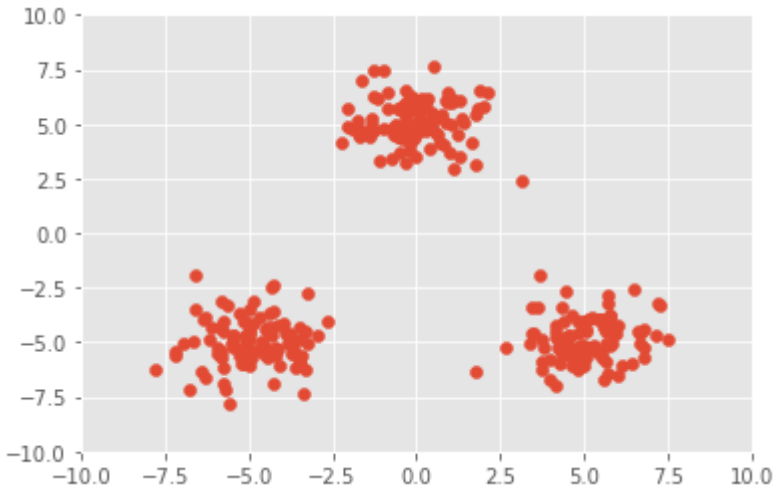
In [4]:

```
print(X_train.shape)
```

```
(300, 2)
```

In [5]:

```
# データ描画
plt.scatter(X_train[:, 0], X_train[:, 1])
plt.xlim(-10, 10)
plt.ylim(-10, 10)
plt.show()
```



学習

k-meansアルゴリズムは以下のとおり

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

In [6]:

```
def distance(x1, x2):
    """ 2つの行列間の距離を求める """
    d = np.sum((x1 - x2)**2, axis=1)
    return d
```

In [7]:

```
n_clusters = 3  # クラスタ数
iter_max = 100  # クラスタ計算の最大繰り返し回数
```

In [8]:

```
np.random.choice(len(X_train), n_clusters, replace=False)
```

Out[8]:

```
array([291, 228,  50])
```

In [9]:

```
# 手順1. 各クラスタ中心をランダムに初期化
centers = X_train[np.random.choice(len(X_train), n_clusters, replace=False)]
```

In [10]:

```
print(centers.shape)
print(centers)
```

```
(3, 2)
[[-5.46040544 -4.82037931]
 [-0.62337427  4.91963261]
 [-0.22963931  4.61617471]]
```

In [11]:

```
for it in range(iter_max):
    prev_centers = np.copy(centers)  # 前回の結果を保持（後で比較するため）
    D = np.zeros((len(X_train), n_clusters))

    # 手順2. 各データ点に対して、各クラスタ中心との距離を計算
    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る
    cluster_index = np.argmin(D, axis=1)

    # 手順3. 各クラスタの中心を計算
    for k in range(n_clusters):
        index_k = cluster_index == k
        centers[k] = np.mean(X_train[index_k], axis=0)

    # 収束判定 centersの値が近い場合は繰り返しを終了
    if np.allclose(prev_centers, centers):
        print(f'iter: {it}, closed. centers:\n{centers}')
        break

    if it % 10 == 0:
        print(f'iter: {it}, centers: {centers}')
```

```
iter: 0, centers: [[-1.32749723 -5.14723561]
 [-0.73858297  5.55499855]
 [ 2.89399228  0.78898428]]
iter: 4, closed. centers:
[[-4.87189834 -4.9708154 ]
 [-0.03877571  5.12490432]
 [ 5.06419495 -4.84271856]]
```

クラスタリング結果

In [12]:

```
def plt_result(X_train, centers, xx):
    """ クラスタリングの結果を描画 """
    # データを散布図で描画
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_pred, cmap='spring')
    # クラスタ中心を描画
    plt.scatter(centers[:, 0], centers[:, 1],
                s=200, marker='X', lw=2, c='black', edgecolor="white")

    pred = np.empty(len(xx), dtype=int)

    for i, x in enumerate(xx):
        d = distance(x, centers)
        pred[i] = np.argmin(d)

    # クラスタ領域の描画
    plt.contourf(xx0, xx1, pred.reshape(100, 100), alpha=0.2, cmap='spring')
```

In [13]:

```
# データの属するクラスタを予測する
y_pred = np.empty(len(X_train), dtype=int)

for i, x in enumerate(X_train):
    d = distance(x, centers)  # クラスタ中心との距離を計算
    y_pred[i] = np.argmin(d)  # 各クラスタ中心から最も距離の近いクラスタに属する
```

In [14]:

```
print(y_pred.shape)
print(y_pred)
```

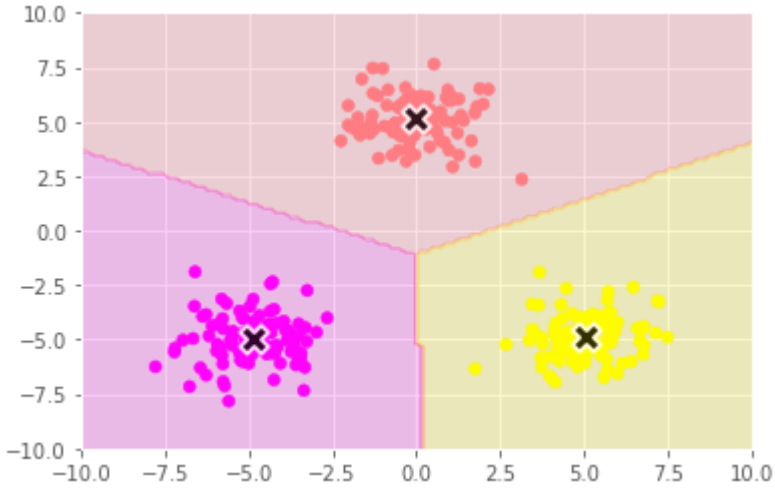
```
(300,)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 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 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]
```

In [15]:

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

In [16]:

```
plt_result(X_train, centers, xx)
```



クラスタ中心の決定は各データ点のかたまりのほぼ中心になっており、
データ点の色を見てもデータ点のかたまりから外れた点が別の色になっているということもなく、うまく分類出来ているように見える。

（比較用）sklearnでの実装

In [17]:

```
kmeans = KMeans(n_clusters=3, random_state=0)
```

In [18]:

```
kmeans.fit(X_train)
```

Out[18]:

```
KMeans(n_clusters=3, random_state=0)
```

In [19]:

```
labels = kmeans.labels_
print(f'labels: {labels}')
```

```
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]
```

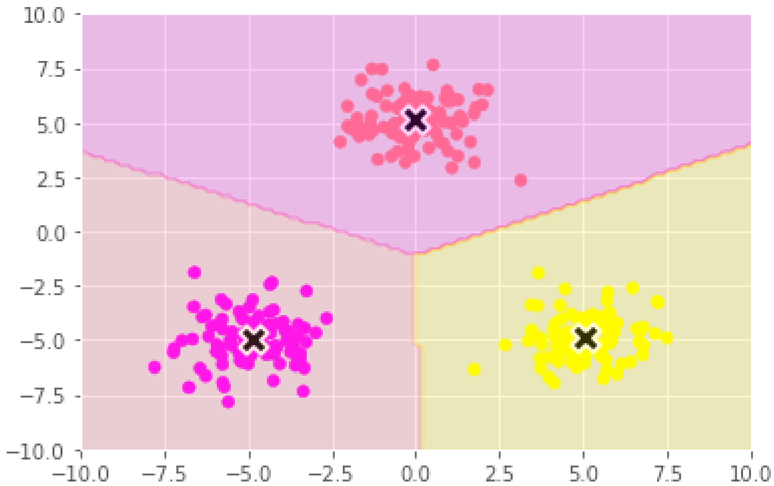
In [20]:

```
centers = kmeans.cluster_centers_
print(f'cluster_centers: {centers}')
```

```
cluster_centers: [[-0.03877571  5.12490432]
 [-4.87189834 -4.9708154 ]
 [ 5.06419495 -4.84271856]]
```

In [21]:

```
plt_result(X_train, centers, xx)
```



NumPy実装のものと同じ結果となった。