演習：k近傍法 (NumPy)

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.neighbors import KNeighborsClassifier
%matplotlib inline
plt.style.use('ggplot')
```

訓練データ作成

In [2]:

```
def gen_data():
    """ ランダムなデータを作成 """
    # 全体的に左寄りのデータを作成 (データ数50で2列なので、25x2の2次元行列が作れる)
    x0 = np.random.normal(size=50).reshape(-1, 2) - 1.  # 1. にすることでfloat型になる
    # 全体的に右寄りのデータを作成
    x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
    X_train = np.concatenate([x0, x1])  # x0とx1を結合

    # 1次元の0と1の配列を作成
    y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)

    return X_train, y_train
```

In [3]:

```
# 訓練用データを作成
X_train, y_train = gen_data()
```

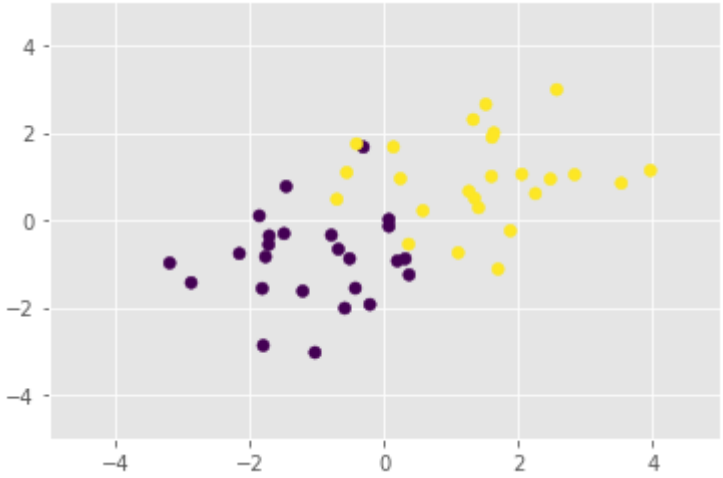
In [4]:

```
print(X_train.shape)
print(y_train.shape)
print(X_train[:5])  # 1列目がラベル0、2列目がラベル1
print(y_train[22:28])  # 0と1の境界辺り
```

```
(50, 2)
(50,)
[[ 0.19617648 -0.93587303]
 [-3.19930899 -0.98272722]
 [-0.31218008  1.6722665 ]
 [-0.5162186  -0.8834322 ]
 [-1.80392554 -2.87175339]]
[0 0 0 1 1 1]
```

In [18]:

```
# 作成データの散布図を描画
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plt.xlim(-5., 5.)
plt.ylim(-5., 5.)
plt.show()
```



予測

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

In [6]:

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

In [7]:

```
def knc_predict(n_neighbors, X_train, y_train, X_test):
    """ NumPyで分類器を作成 """
    y_pred = np.empty(len(X_test), dtype=y_train.dtype)

    for i, x in enumerate(X_test):
        distances = distance(x, X_train)  # テストデータごとに、学習データとの距離を計算
        nearest_index = distances.argsort()[:n_neighbors]
        mode, _ = stats.mode(y_train[nearest_index])  # 最頻値を出力
        y_pred[i] = mode

    return y_pred
```

In [8]:

```
def plot_result(X_train, y_train, y_pred):
    xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
    y_pred = y_pred.reshape(100, 100).astype(np.float)

    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
    plt.contourf(xx0, xx1, y_pred, alpha=0.2, levels=np.linspace(0, 1, 3))
    plt.show()
```

In [9]:

```
n_neighbors = 3  # 近傍の数
```

In [10]:

```
# テストデータを作成
xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
X_test = np.array([xx0, xx1]).reshape(2, -1).T
```

In [19]:

```
print(X_test.shape)
```

```
(10000, 2)
```

In [25]:

```
# 中身を確認
print(X_test[:3])
print(X_test[-3:])
print(X_test[4998:5003])
```

```
[[-5.        -5.       ]
 [-4.8989899 -5.       ]
 [-4.7979798 -5.       ]]
[[4.7979798 5.       ]
 [4.8989899 5.       ]
 [5.        5.       ]]
[[ 4.8989899  -0.05050505]
 [ 5.         -0.05050505]
 [-5.          0.05050505]
 [-4.8989899   0.05050505]
 [-4.7979798   0.05050505]]
```

In [11]:

```
y_pred = knc_predict(n_neighbors, X_train, y_train, X_test)
```

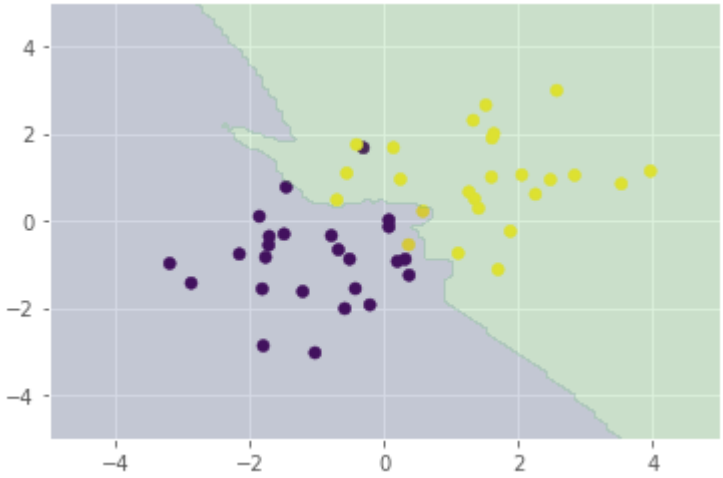
In [27]:

```
print(y_pred.shape)
print(y_pred[:10])
print(y_pred[-10:])
```

```
(10000,)
[0 0 0 0 0 0 0 0 0 0]
[1 1 1 1 1 1 1 1 1 1]
```

In [12]:

```
plot_result(X_train, y_train, y_pred)
```



考察： 特に、$0 \leq x \leq 1$ あたりが過剰適合しているように見える。
k近傍法では、近傍数を増やすとより滑らかな境界になっていくため、
n_neighbors の値を増やすことでモデルが改善すると思われる。

(比較用) sklearnでの実装

In [13]:

```
xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T
```

In [14]:

```
# 分類器を作成
knc = KNeighborsClassifier(n_neighbors=n_neighbors)
```

In [15]:

```
knc.fit(X_train, y_train)  # 学習
```

Out[15]:

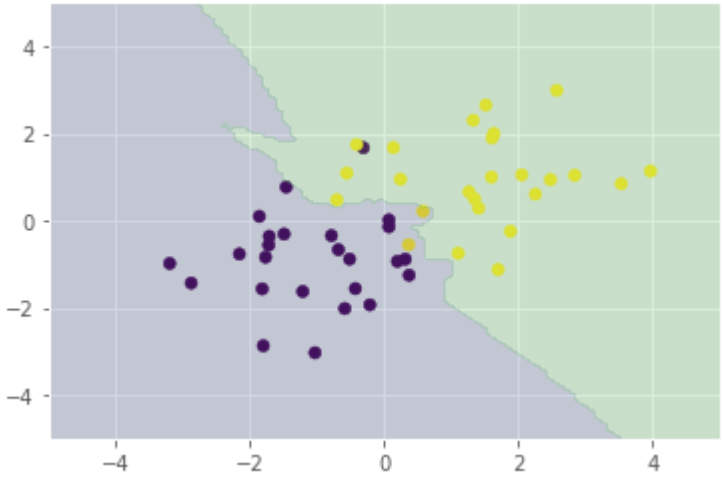
```
KNeighborsClassifier(n_neighbors=3)
```

In [16]:

```
y_pred =  knc.predict(xx)
```

In [17]:

```
plot_result(X_train, y_train, y_pred)
```



NumPy実装の結果とほぼ同じ結果となっている。

In [ ]:

```
```