知識情報学 第2回演習サンプルプログラム ex2.ipynb

- Programmed by Wu Hongle, 監修　福井健一
- Last update: 2018/09/14
- Checked with Python 3.8.8, scikit-learn 1.0
- MIT License

k近傍法による分類と識別面のプロット

In [31]:
```
%matplotlib inline
from sklearn import datasets
import numpy as np
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

データの読み込みと標準化

- 【課題2】近傍数kを変更してみましょう.
- 【課題3】使用する特徴量(d1,d2)を変更してみましょう.

In [32]:
```
# K近傍法の近傍数パラメータ k
neighbors = 5
# テストデータ分割のための乱数のシード（整数値）
random_seed = 1
#  テストデータの割合
test_proportion = 0.3
# Iris データセットをロード
iris = datasets.load_iris()
# 使用する特徴の次元を(Irisの場合は0,1,2,3から)2つ指定. d1とd2は異なる次元を指定す
d1 = 0
d2 = 1
# d1,d2列目の特徴量を使用
X = iris.data[:, [d1, d2]]
# クラスラベルを取得
y = iris.target
# z標準化
X_std = scale(X)
```

課題1(a) データを学習データとテストデータに分割

- train_test_split()を使用し， 変数test_proportionの割合をテストデータとし， 変数random_seedを乱数生成器の状態に設定
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

In [33]:
```
X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=test_prc
```

課題1(b) クラスKNeighborsClassifierを使用してk近傍法のインスタンスを生成

- 近傍数kは上で指定したneighborsを使用

In [34]:
```
knn = KNeighborsClassifier(n_neighbors=neighbors)
```

k近傍法のモデルに学習データを適合

In [35]:
```
knn.fit(X_train, y_train)
```

Out[35]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

正答率の算出

In [36]:
```
acc_train = accuracy_score(y_train, knn.predict(X_train))
acc_test  = accuracy_score(y_test, knn.predict(X_test))
print('k=%d, features=(%d,%d)' % (neighbors, d1, d2))
print('accuracy for training data: %f' % acc_train)
print('accuracy for test data: %f' % acc_test)
```

```
k=5, features=(0,1)
accuracy for training data: 0.847619
accuracy for test data: 0.688889
```

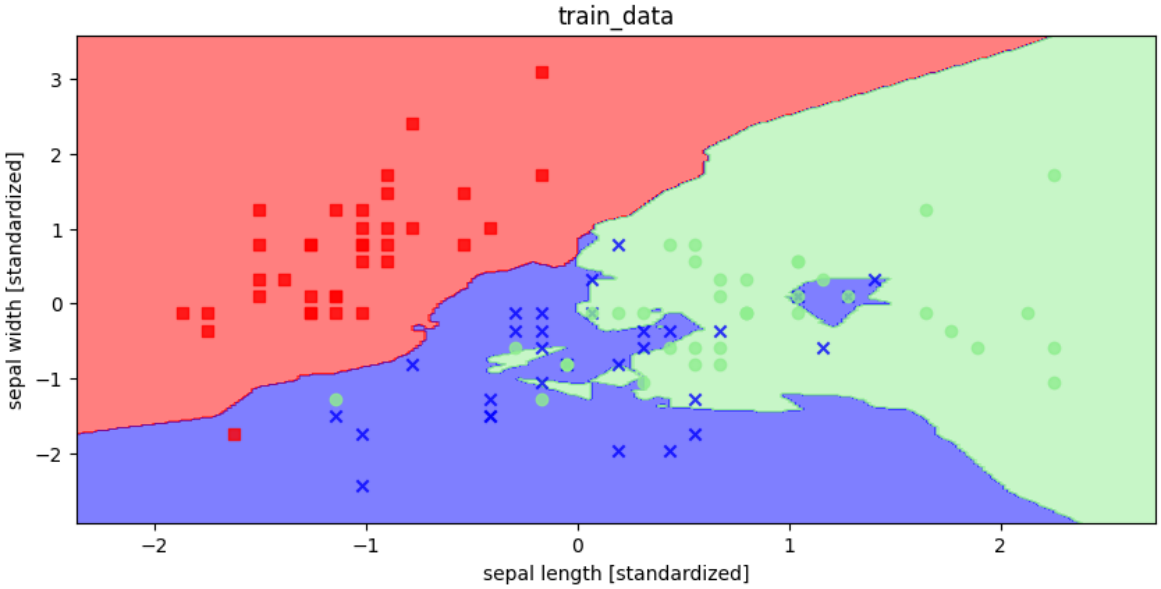
識別境界面をプロットする関数

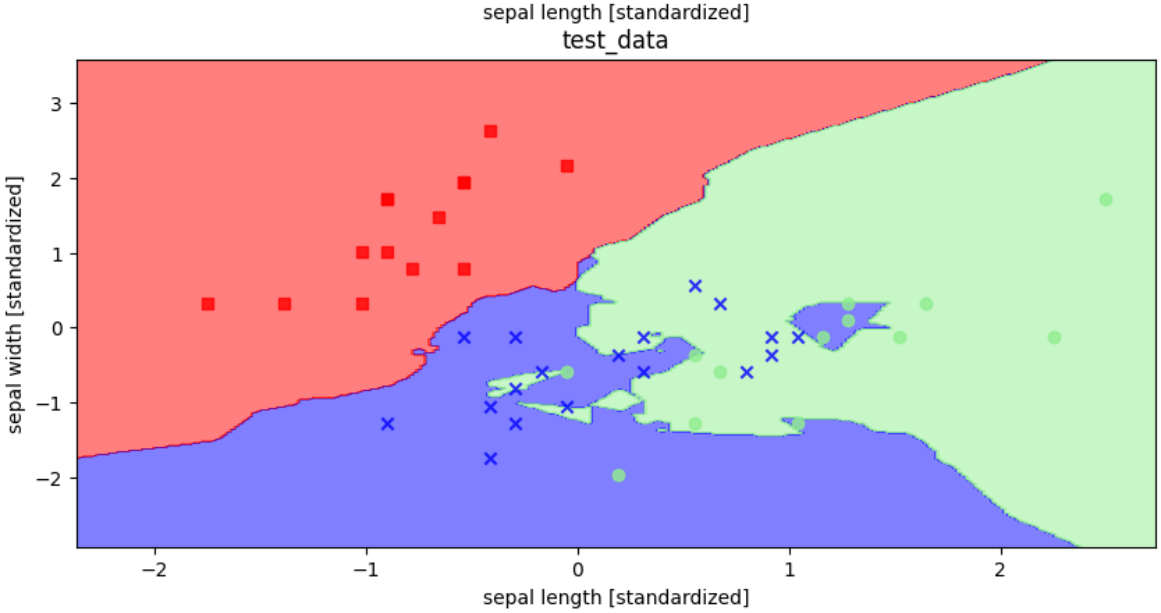
各格子点に対してk近傍法で識別を行い，識別結果に応じて色を付けている

In [37]:
```
def plot_decision_boundary(title=None):
    x1_min, x1_max = X_train[:, 0].min() - 0.5, X_train[:, 0].max() + 0.5
    x2_min, x2_max = X_train[:, 1].min() - 0.5, X_train[:, 1].max() + 0.5
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, 0.02),
                           np.arange(x2_min, x2_max, 0.02))

    Z = knn.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)

    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    plt.figure(figsize=(10,10))
    plt.subplot(211)

    plt.contourf(xx1, xx2, Z, alpha=0.5, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y_train)):
```

```
        plt.scatter(x=X_train[y_train == cl, 0], y=X_train[y_train == cl, 1],
                    alpha=0.8, c=colors[idx],
                    marker=markers[idx], label=cl)


    plt.xlabel('sepal length [standardized]')
    plt.ylabel('sepal width [standardized]')
    plt.title('train_data')

    plt.subplot(212)

    plt.contourf(xx1, xx2, Z, alpha=0.5, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y_test)):
        plt.scatter(x=X_test[y_test == cl, 0], y=X_test[y_test == cl, 1],
                    alpha=0.8, c=colors[idx],
                    marker=markers[idx], label=cl)


    plt.xlabel('sepal length [standardized]')
    plt.ylabel('sepal width [standardized]')
    plt.title('test_data')

    plt.suptitle(title)
    plt.show()
```

In [38]:
```
plot_decision_boundary()
```



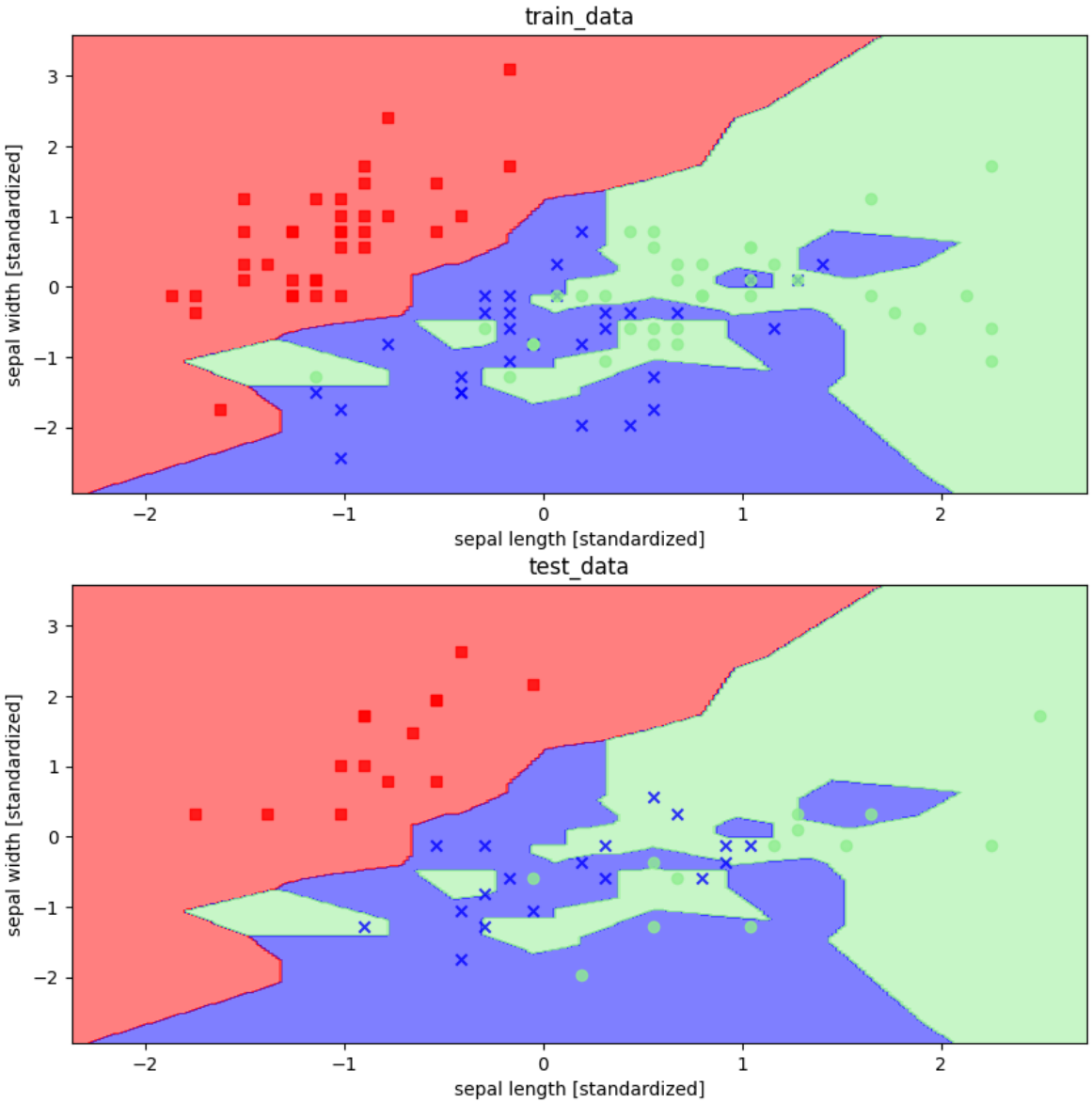


課題2-2 近傍数kの影響について，識別境界および正解率 （Accuracy）の観点で考察しなさい

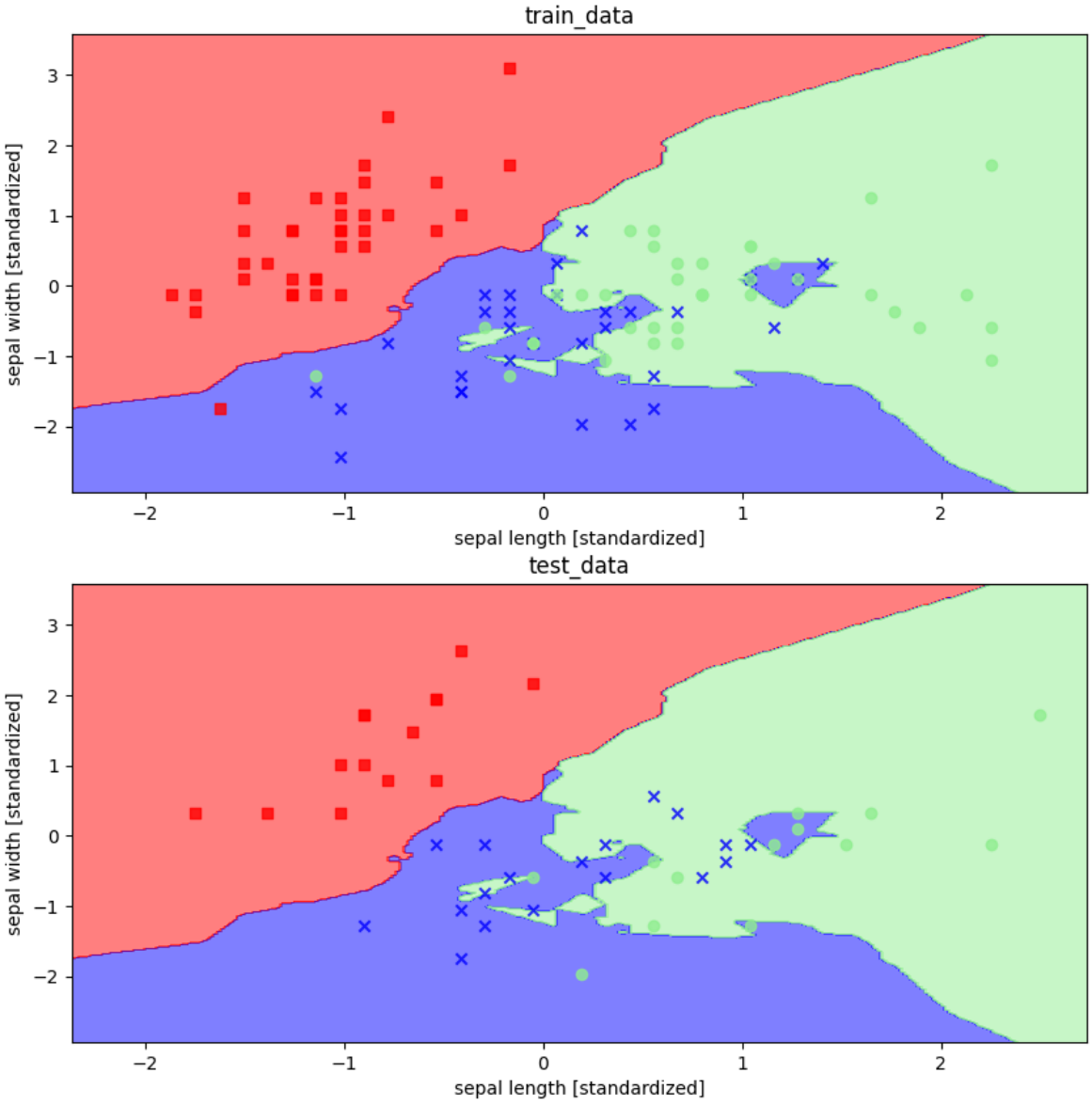
In [39]:
```
# 1から20までのkの値を試す
k_values = range(1, 21)

# 正解率を保存するリスト
train_accuracies = []
test_accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # 訓練データとテストデータでの正解率を計算
    acc_train = accuracy_score(y_train, knn.predict(X_train))
    acc_test  = accuracy_score(y_test, knn.predict(X_test))

    train_accuracies.append(acc_train)
    test_accuracies.append(acc_test)
    if k in [1,5,10,15,20]:
        plot_decision_boundary(f'neighbors={k}')
```

```
plt.figure(figsize=(10, 6))
plt.plot(k_values, train_accuracies, label='Training Accuracy', marker='o')
plt.plot(k_values, test_accuracies, label='Test Accuracy', marker='o')
plt.title('Accuracy vs. Number of Neighbors (k)')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

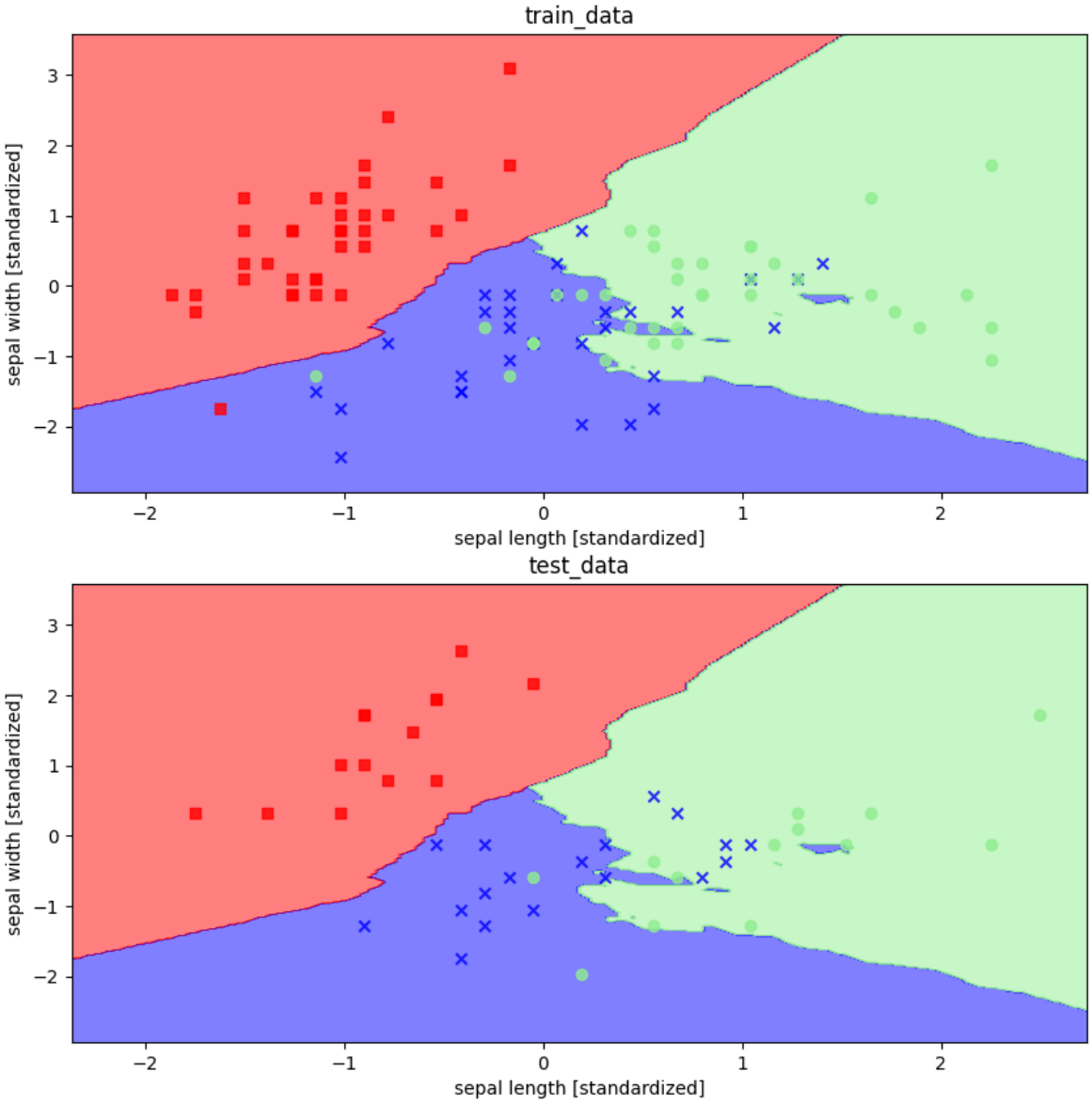
neighbors=1



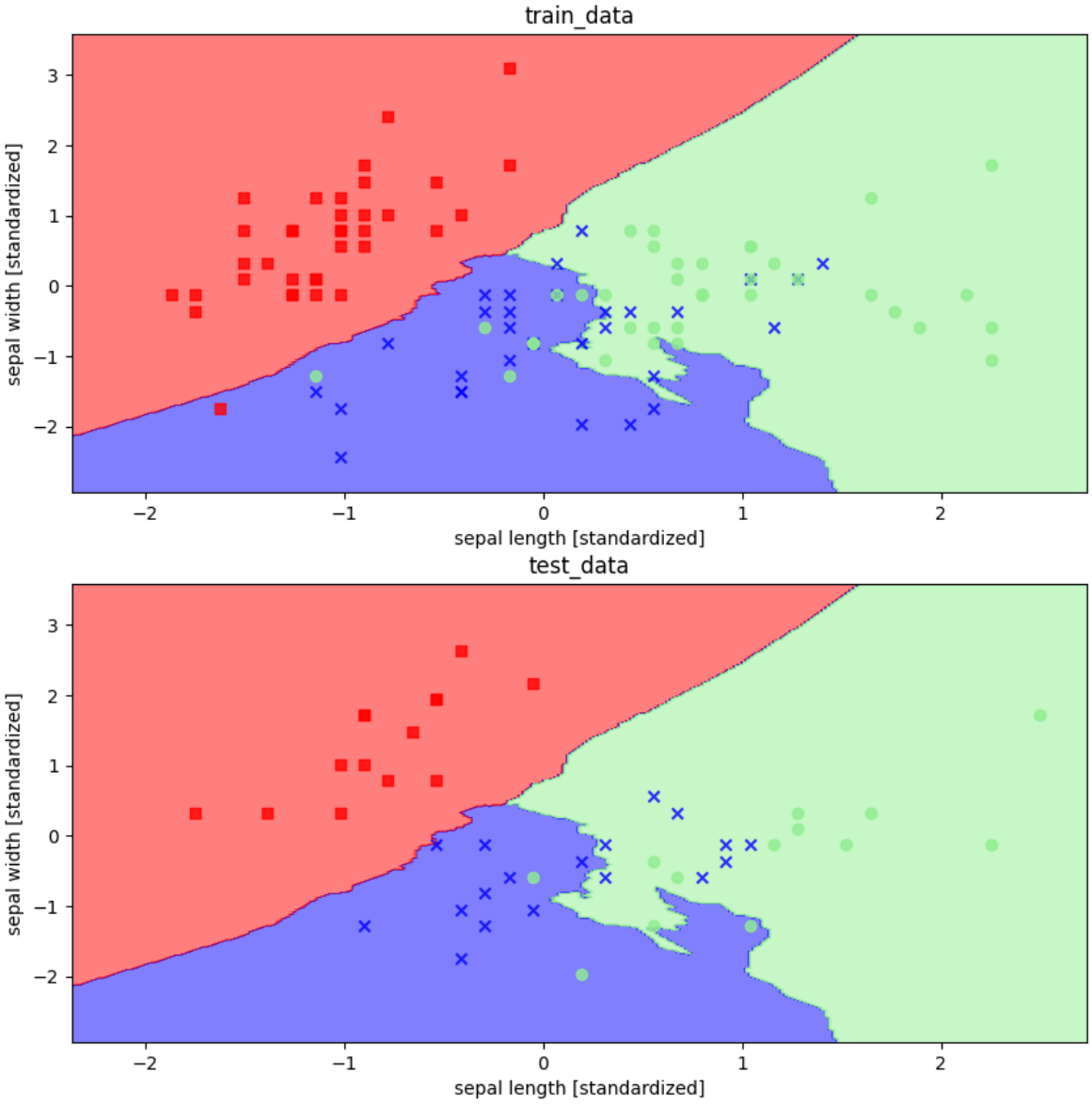
neighbors=5


train_data
sepal width [standardized]
sepal length [standardized]
test_data
sepal width [standardized]
sepal length [standardized]

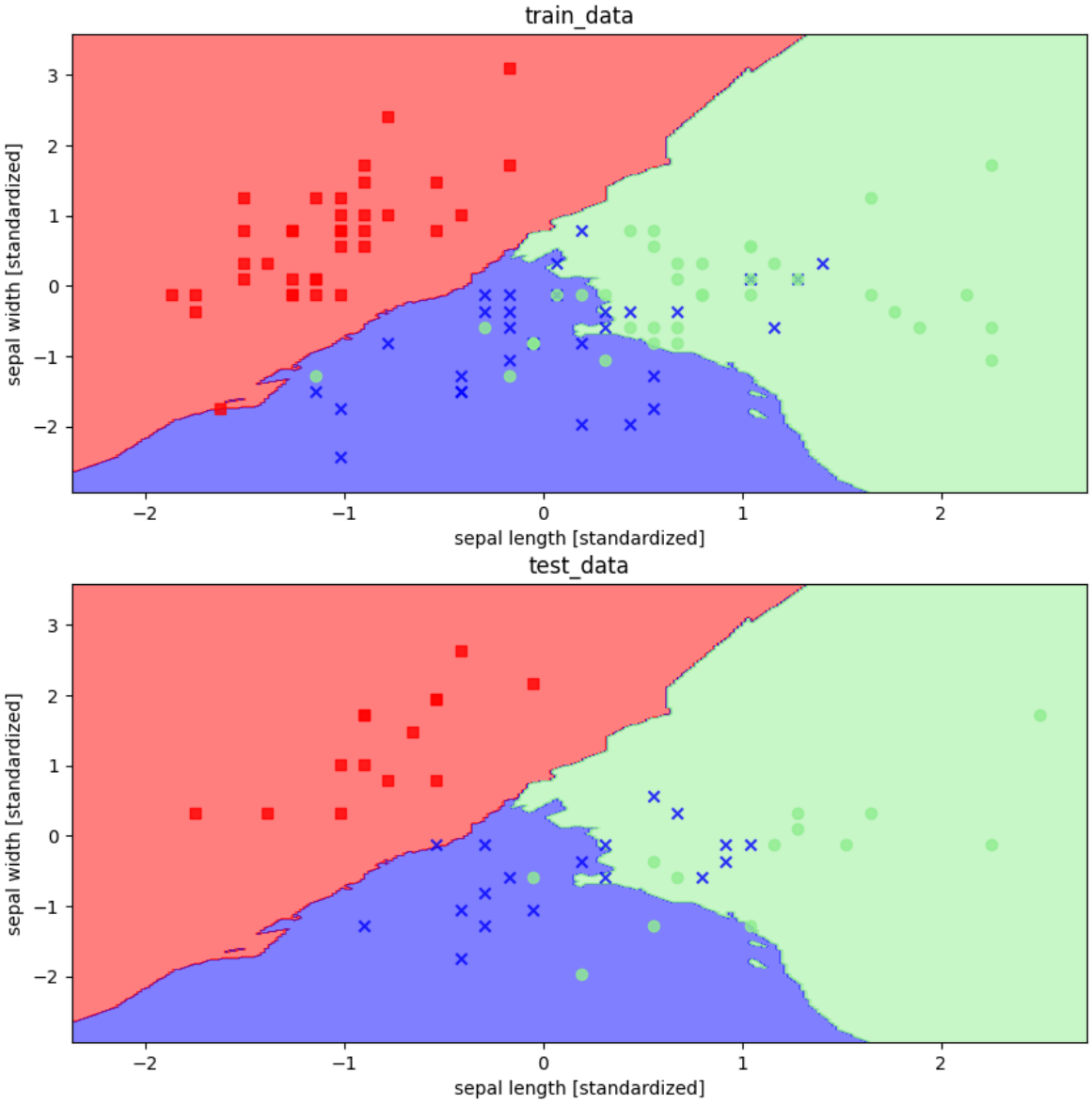
neighbors=10

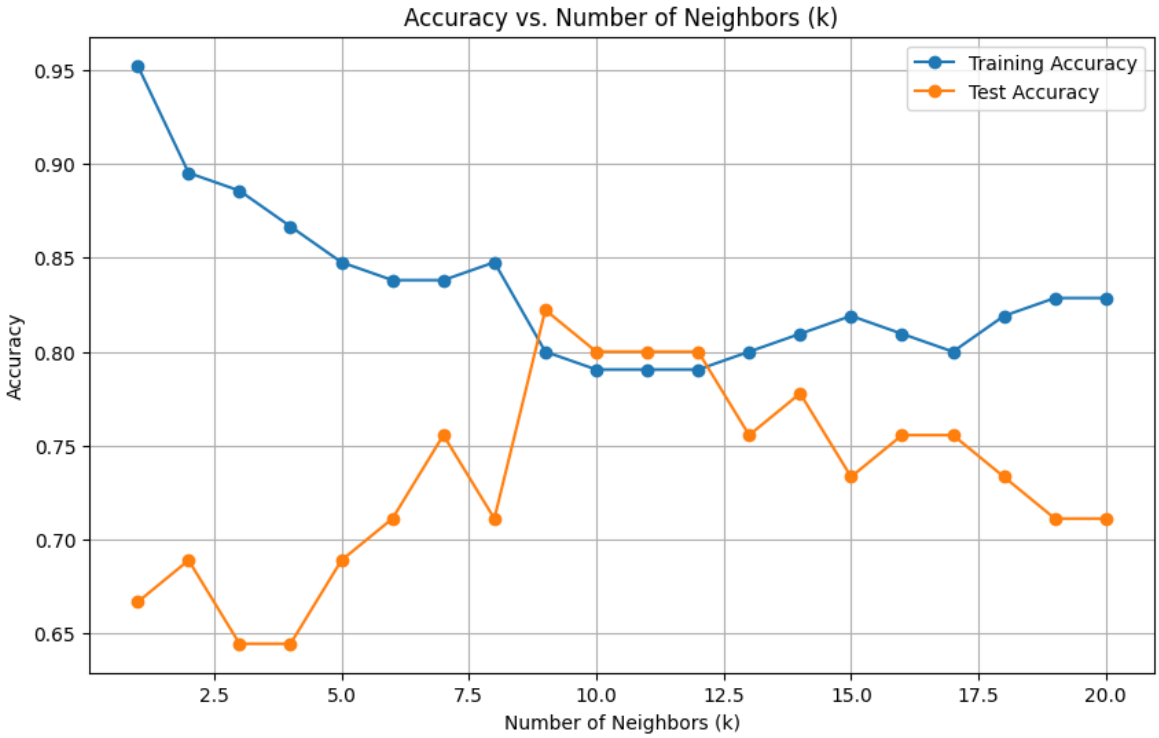


neighbors=15



neighbors=20





考察

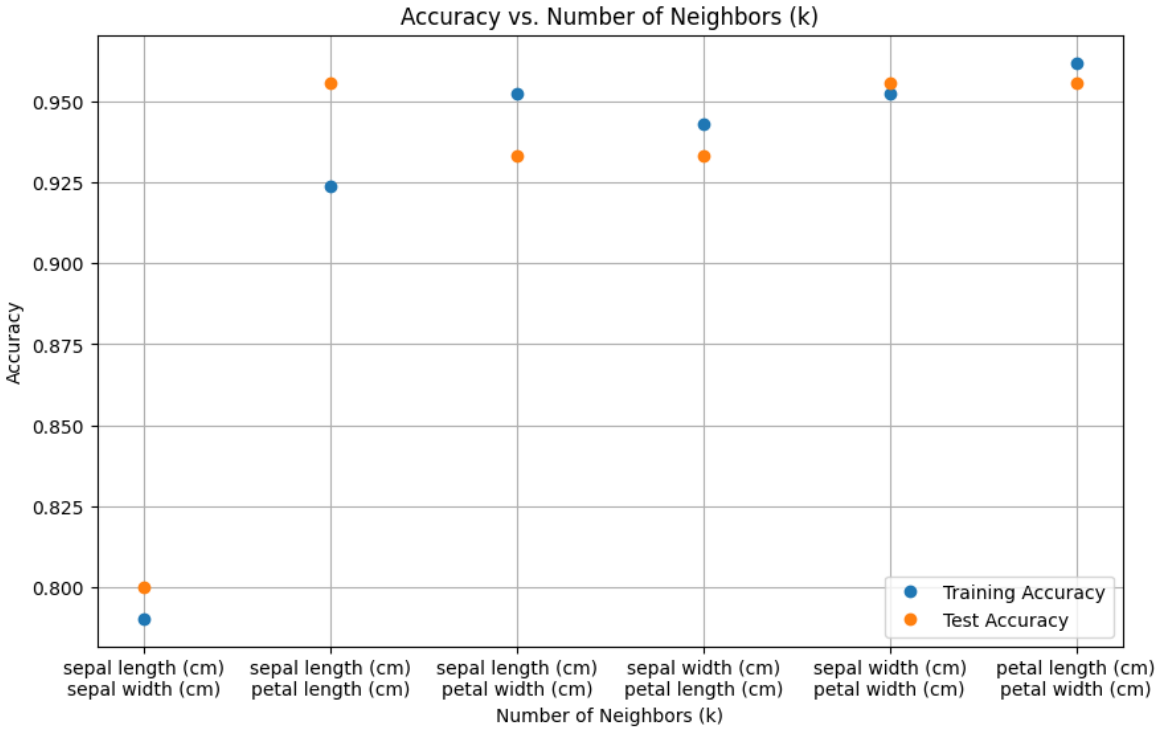
上の図は、近傍数1,5,10,15,20のときの識別境界の図と、1~20の近傍数の時の訓練、テストデータの正解率である。
近傍数が小さい場合、モデルは訓練データに過剰適合し複雑な識別境界を持ったため、テストデータでの正解率は低くなったと考えられる。
一方、近傍数が大きい場合、モデルは訓練データに対して一般化されたため、識別境界は滑らかになり、正解率が低下したと考えられる。

課題2-3. 使用する特徴量を変えて，特徴量と識別境界や正解率との関係を考察しなさい.

In [40]:
```
# K近傍法の近傍数パラメータ k
neighbors = 10 # 課題2-2より、近傍数10を使用

# テストデータ分割のための乱数のシード（整数値）
random_seed = 1
#  テストデータの割合
test_proportion = 0.3
# Iris データセットをロード
iris = datasets.load_iris()

# 使用したパラメータを保存するリスト
used_param = []
# 正解率を保存するリスト
train_accuracies = []
test_accuracies = []


for d1 in range(0,4):
    for d2 in range(d1+1,4):
        used_param.append(iris.feature_names[d1]+"\n"+iris.feature_names[d2])
```

```
        # d1,d2列目の特徴量を使用
        X = iris.data[:, [d1, d2]]
        y = iris.target
        X_std = scale(X)

        X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=

        knn = KNeighborsClassifier(n_neighbors=neighbors)
        knn.fit(X_train, y_train)

        # 訓練データとテストデータでの正解率を計算
        acc_train = accuracy_score(y_train, knn.predict(X_train))
        acc_test  = accuracy_score(y_test, knn.predict(X_test))

        train_accuracies.append(acc_train)
        test_accuracies.append(acc_test)
```

In [41]:
```
plt.figure(figsize=(10, 6))
plt.plot(used_param, train_accuracies, 'o', label='Training Accuracy')
plt.plot(used_param, test_accuracies, 'o', label='Test Accuracy')
plt.title('Accuracy vs. Number of Neighbors (k)')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



考察

'petal length (cm)' と 'petal width (cm)' の組み合わせがトレーニングデータ、テストデータ共に、最も高い正解率を示している。
これは、アヤメの品種を分類するのに非常に有効な特徴量の組み合わせであると考えられる。
一方、'sepal width (cm)' を含む組み合わせは正解率が低い傾向にある。
これは、この特徴量がアヤメの品種分類に対してあまり有用でないと考えられる。