Kayvan_Shah_Assignment 2_solution

February 2, 2023

1 Assignment 2: Exploratory Data Analysis and K Nearest Neighbors Classification

For this assignment you will perform exploratory data analysis to visualize Fisher's Iris dataset using Scikit Learn. And, you will explore the bias/variance trade-off by applying k-nearest neighbors classification to the Iris dataset and varying the hyperparameter k.

Documentation for Scikit Learn: + The top level documenation page is here: https://scikit-learn.org/stable/index.html + The API for the KNearestNeighborsClassifier is here: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsC + The User Guide for KNearestNeighborsClassifier is here: https://scikit-learn.org/stable/modules/neighbors.html#classification + Scikit Learn provides many Jupyter notebook examples on how use the toolkit. These Jupyter notebook examples can be run on MyBinder: https://scikit-learn.org/stable/auto_examples/index.html

For more information about the Iris dataset, see this page https://en.wikipedia.org/wiki/Iris_flower_data_set.

[3]:
```
%%bash
pip install -U scikit-learn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-
packages (1.2.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-
packages (from scikit-learn) (1.7.3)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-
packages (from scikit-learn) (1.21.6)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.8/dist-
packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.8/dist-packages (from scikit-learn) (3.1.0)
```

[4]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn import neighbors
from sklearn.model_selection import train_test_split
```

```python
from pandas import DataFrame
```

Load Iris dataset

[5]:
```python
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

[6]:
```python
print("Number of instances in the iris dataset:", X.shape[0])
print("Number of features in the iris dataset:", X.shape[1])
print("The dimension of the data matrix X is", X.shape)
```

```
Number of instances in the iris dataset: 150
Number of features in the iris dataset: 4
The dimension of the data matrix X is (150, 4)
```

[7]:
```python
X
```

[7]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
```

```
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
```

```
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
```

```
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

The y vector length is 150. It has three unique values: 0, 1 and 2. Each value represents a species of iris flower.

[8]:
```
y
```

[8]:
```
array([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, 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])
```

[9]:
```
y.shape
```

[9]:
```
(150,)
```

[10]:
```
dir(iris)
```

[10]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

[11]:
```python
iris.target_names
```

[11]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

[12]:
```python
iris.feature_names
```

[12]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

1.0.1 Extension: Show the summary table of iris data including min, max, median, quantiles

[13]:
```python
# Insert your answer here
import pandas as pd

iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

[14]:
```python
iris_df.describe()
```

[14]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
```

```
75%            1.800000
max            2.500000
```

1.1 Part 1Exploratory Data Analysis

1.1.1 Part 1a

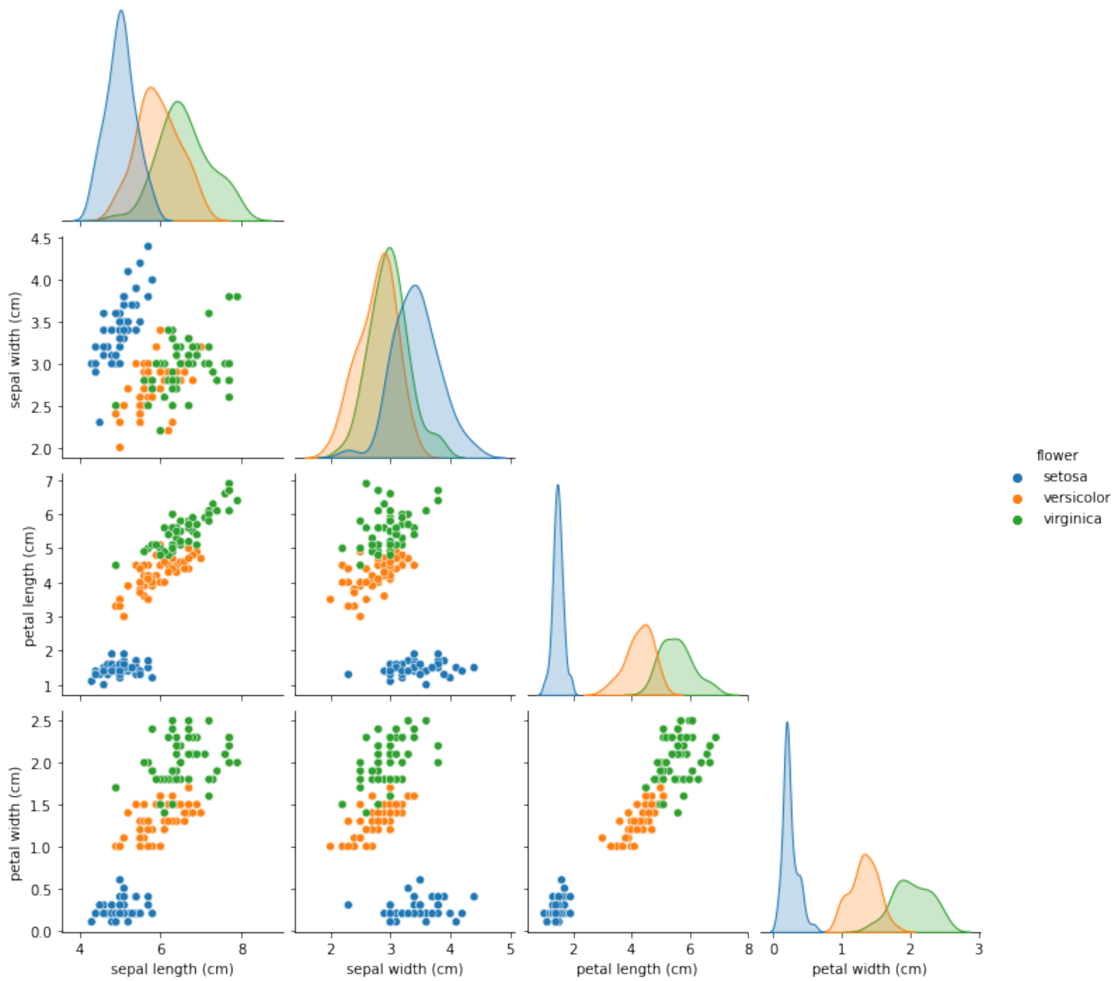
Generate scatter plots using each pair of the attributes as axis. You should generate $6 = \binom{4}{2}$ scatter plots.

[15]:
```
## Insert your answer here...
iris_df_pp = iris_df.copy(deep=True)
iris_df_pp["flower"] = y
iris_df_pp["flower"] = iris_df_pp["flower"].map({0:"setosa",1:"versicolor",2:
 ↪"virginica"})
iris_df_pp.head()
```

[15]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   flower
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa
```

[16]:
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.pairplot(iris_df_pp, hue="flower", corner=True)
```

[16]: <seaborn.axisgrid.PairGrid at 0x7feba20c1f40>



1.1.2 Extension: Draw a boxplot of sepal length (cm), sepal width (cm), petal length (cm), petal width (cm). Use color to show the different target class.

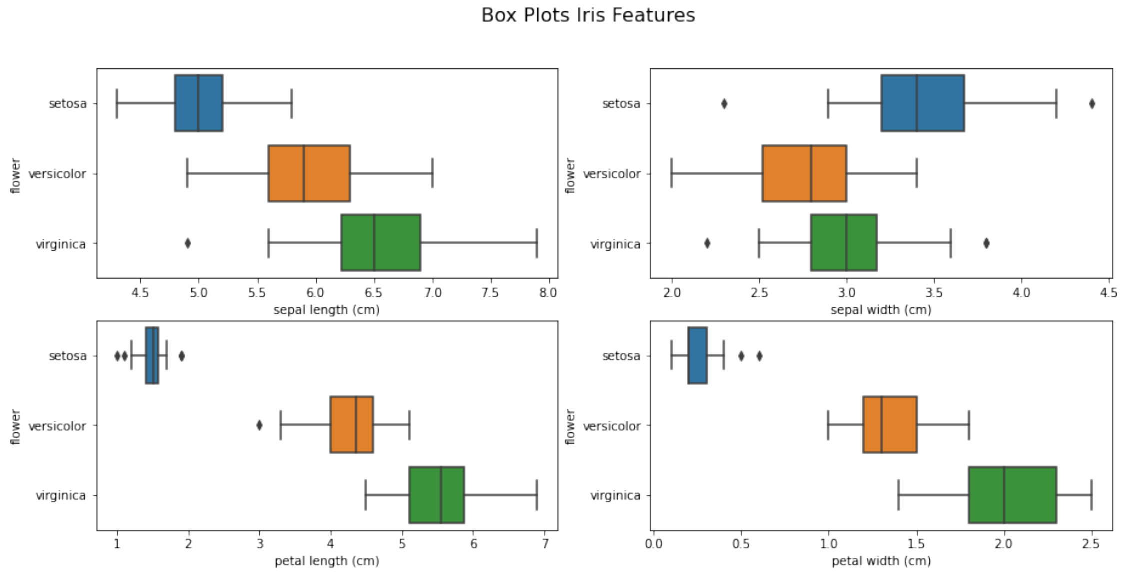
Some links to help you:

https://seaborn.pydata.org/generated/seaborn.boxplot.html

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html

[17]:
```
# Insert your code ...
# Creating subplots
fig, ax = plt.subplots(2, 2, figsize=(15, 7))
fig.suptitle("Box Plots Iris Features", fontsize=16)

axes = [ax[0,0], ax[0,1],
        ax[1,0], ax[1,1]]

for i, feat in enumerate(iris.feature_names):
```

```
    sns.boxplot(data=iris_df_pp, x=feat, y="flower", ax=axes[i])
```



1.1.3 Part 1b

If you were to draw linear decision boundaries to separate the classes, which scatter plot do you think will have the least error and which the most?

1.1.4 Insert your 1b answer here

Scatter Plot with:

- Most Error: `sepal width (cm) vs sepal length (cm)`
- Least Error: `sepal width (cm) vs petal length (cm)`

1.1.5 Part 1c

Scatter plots using two attributes of the data are equivalent to project the four dimensional data down to two dimensions using axis-parallel projection. Principal component analysis (PCA) is a technique to linearly project the data to lower dimensions that are not necessarily axis-parallel. Use PCA to project the data down to two dimensions.

Documentation for PCA: + API https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.ht + User guide https://scikit-learn.org/stable/modules/decomposition.html#pca

[18]:
```
### Insert your code here
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
dimension_reducer = pca.fit(iris_df)
```

```
data = pd.DataFrame(dimension_reducer.transform(iris_df), columns=[f"comp_{i}"␣
↪for i in range(2)])
data["flower"] = iris_df_pp["flower"].copy(deep=True)
data.head()
```

[18]:

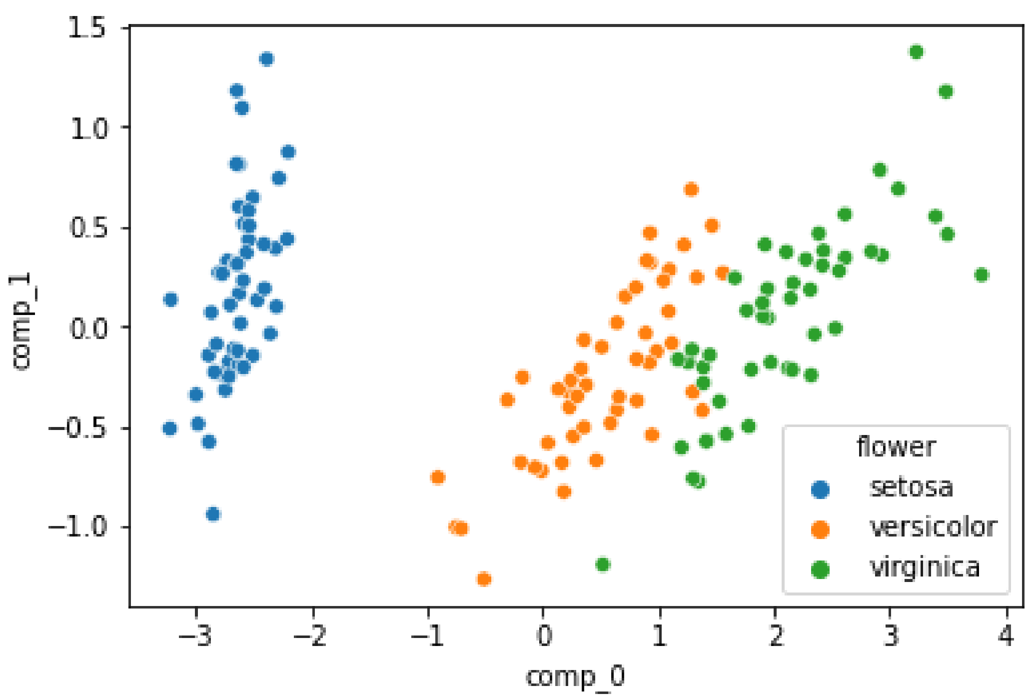
```
     comp_0    comp_1  flower
0 -2.684126  0.319397  setosa
1 -2.714142 -0.177001  setosa
2 -2.888991 -0.144949  setosa
3 -2.745343 -0.318299  setosa
4 -2.728717  0.326755  setosa
```

1.1.6 In the case of the Iris dataset, does PCA do a better job of separating the classes?

[19]:

```
sns.scatterplot(data=data, x="comp_0", y="comp_1", hue="flower")
```

[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7feb9f7622b0>



1.1.7 Insert your answer

PCA with `n_components=2` does a better job separating the 3 classes with minimal region of overlap between ***versicolor*** & ***virginica***

1.2 Part 2 K Nearest Neighbor

Split the dataset into train set and test set. Use 67 percent of the dataset for training, and use 33 percent for testing.

[20]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42,
)
```

[21]:
```
print("Number of instances in the train set:", X_train.shape[0])
print("Number of instances in the test set:", X_test.shape[0])
```

```
Number of instances in the train set: 100
Number of instances in the test set: 50
```

1.2.1 Part 2a Create a KNeibhorsClassifier with `n_neighbors = 5`. And, train the classifier using the train set.

[22]:
```
### Insert you answer here
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)
```

[22]:
```
KNeighborsClassifier()
```

[23]:
```
print("Using", model.n_neighbors, "neighbors:")
print("The train accuracy score is:", model.score(X_train, y_train))
print("The test accuracy score is :", model.score(X_test, y_test))
```

```
Using 5 neighbors:
The train accuracy score is: 0.96
The test accuracy score is : 0.98
```

1.2.2 Part 2b Tuning hyperparameter k

As we have seen in class, hyperparameter k of the K Nearest Neighbors classification affects the inductive bias. For this part train multiple near neighbor classifier models, store the results in a DataFrame. The plot plot training error and testing error versus N/k, where N = 100.

1.2.3 Extension: Use different metric for knn classification.

```
- 1). Euclidean distance
- 2). Manhattan distance
- 3). Chebyshev distance.
```

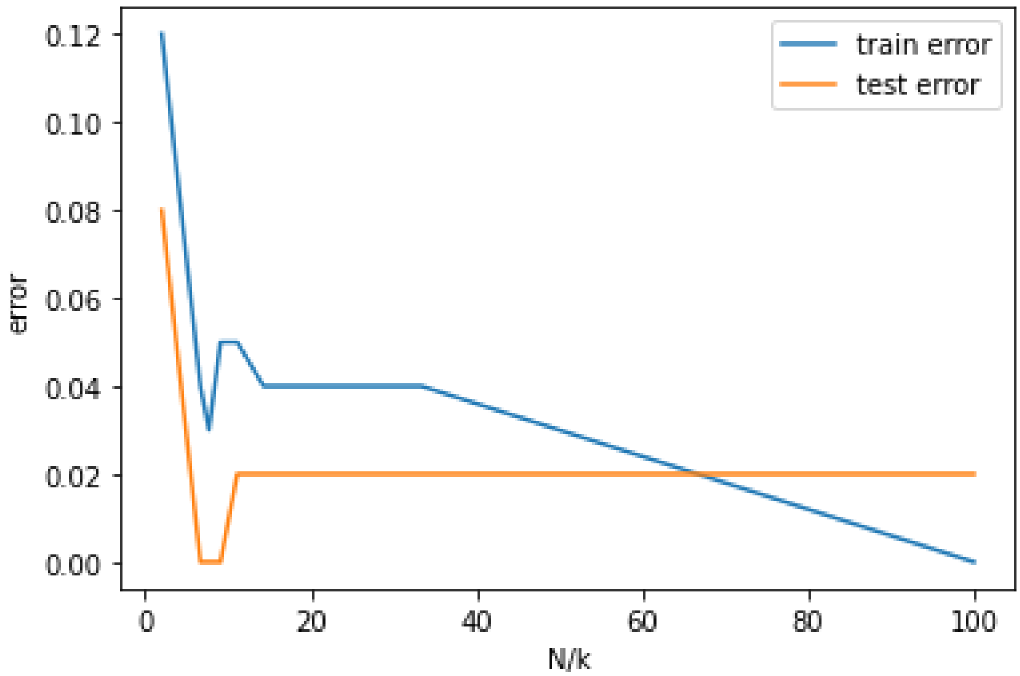
Distance Metrics Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.dist

[24]:
```python
k_list = [1, 3, 5, 7, 9, 11, 13, 15, 50]

def get_train_test_error(k_list: list=k_list, metric=None):
    train = []
    test = []
    n_k = []

    for k in k_list:
        if metric:
            model = KNeighborsClassifier(n_neighbors=k, metric=metric)
        else:
            model = KNeighborsClassifier(n_neighbors=k)
        model.fit(X_train, y_train)
        train.append(1 - model.score(X_train, y_train))
        test.append(1 - model.score(X_test, y_test))
        n_k.append(100/k)

    result = pd.DataFrame(
        data = {
            "N/k": n_k,
            "train error": train,
            "test error":test
        },
    )
    return result
```

[25]:
```python
### Insert your code
# Use the `result` to store the DataFrame
# euclidean
result = get_train_test_error(metric="euclidean")
```

[26]:
```python
result.plot(x='N/k', y=['train error', 'test error'], ylabel='error')
```

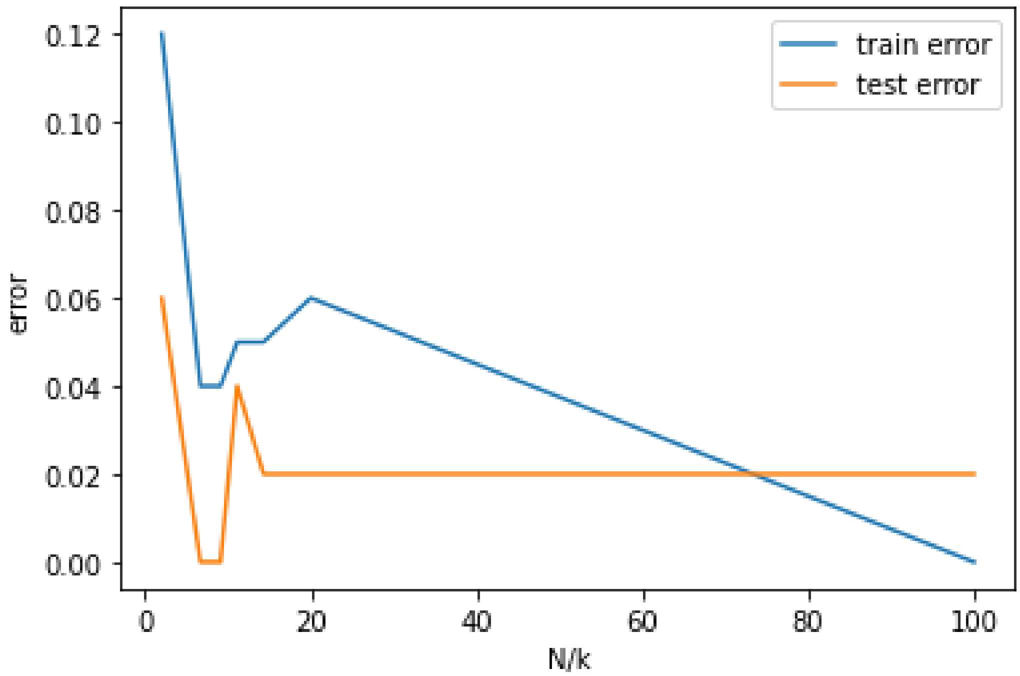
[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7feb9a8672b0>



[27]:
```
### Insert your code
# Use the `result` to store the DataFrame
# manhattan
result = get_train_test_error(metric="manhattan")
```

[28]:
```
result.plot(x='N/k', y=['train error', 'test error'], ylabel='error')
```

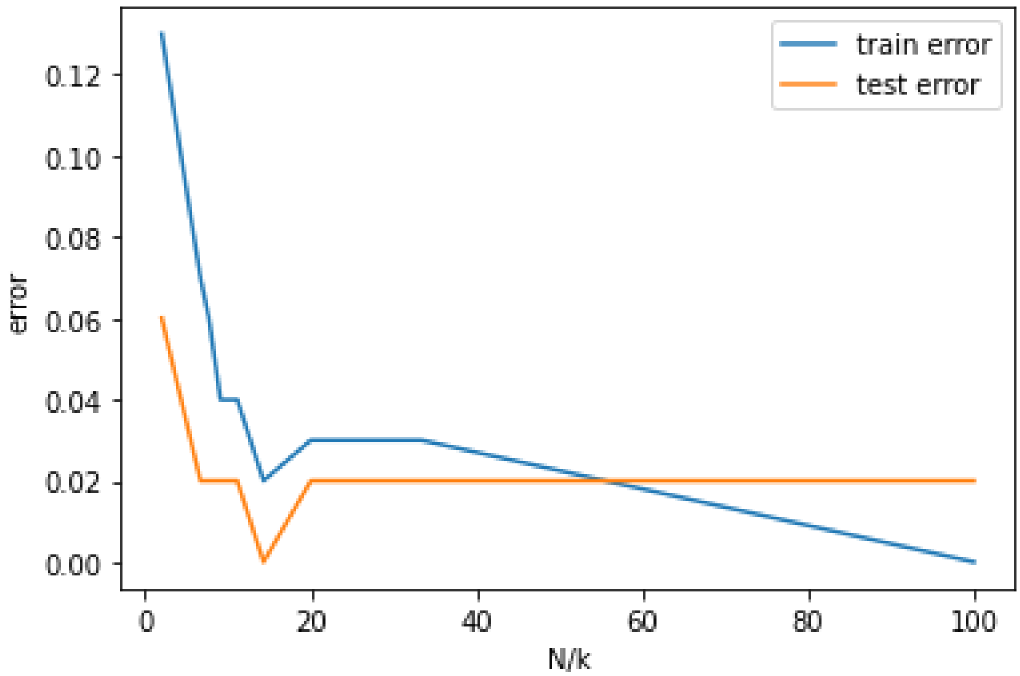
[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7feb9a843100>



[29]:
```
### Insert your code
# Use the `result` to store the DataFrame
# chebyshev
result = get_train_test_error(metric="chebyshev")
```

[30]:
```
result.plot(x='N/k', y=['train error', 'test error'], ylabel='error')
```

[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7feb9a7bd310>



1.2.4 Part 2c Plot decision boundaries of K Nearest Neighbors

Use Scikit Learn's DecisionBoundaryDisplay class to visualize the nearest neighbor boundaries as k is varied.

https://scikit-learn.org/stable/modules/generated/sklearn.inspection.DecisionBoundaryDisplay.html#sklearn.insp

[31]:
```
k_list = [1, 3, 5, 7, 9, 11, 13, 15, 50]
```

Simplify the problem by using only the first 2 attributes of the dataset

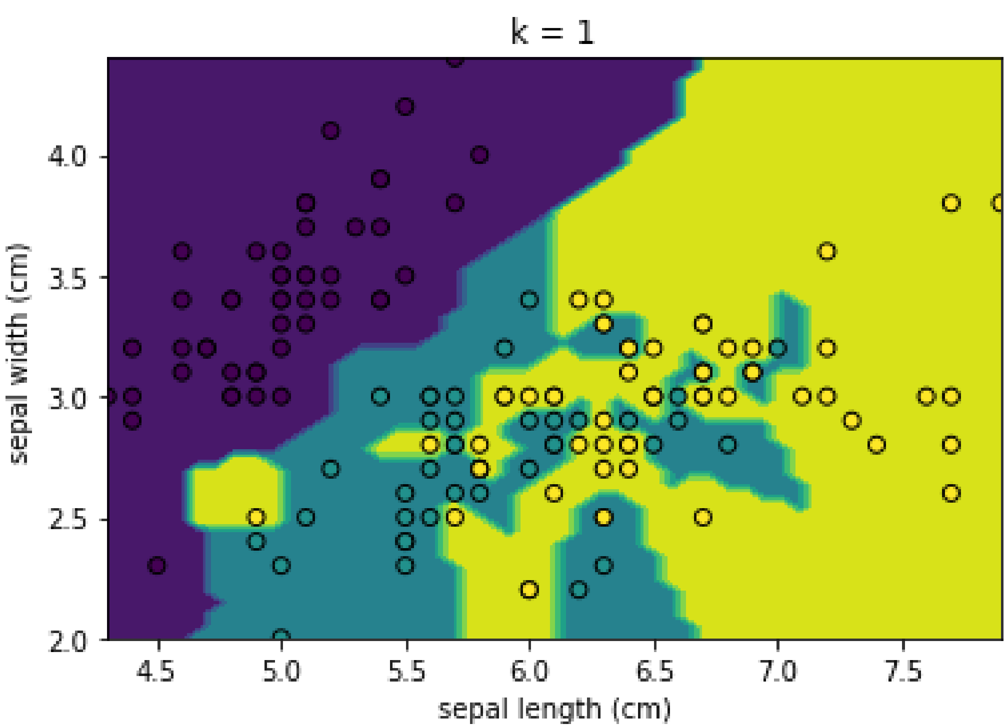
[32]:
```
X2 = iris.data[:, :2]
```

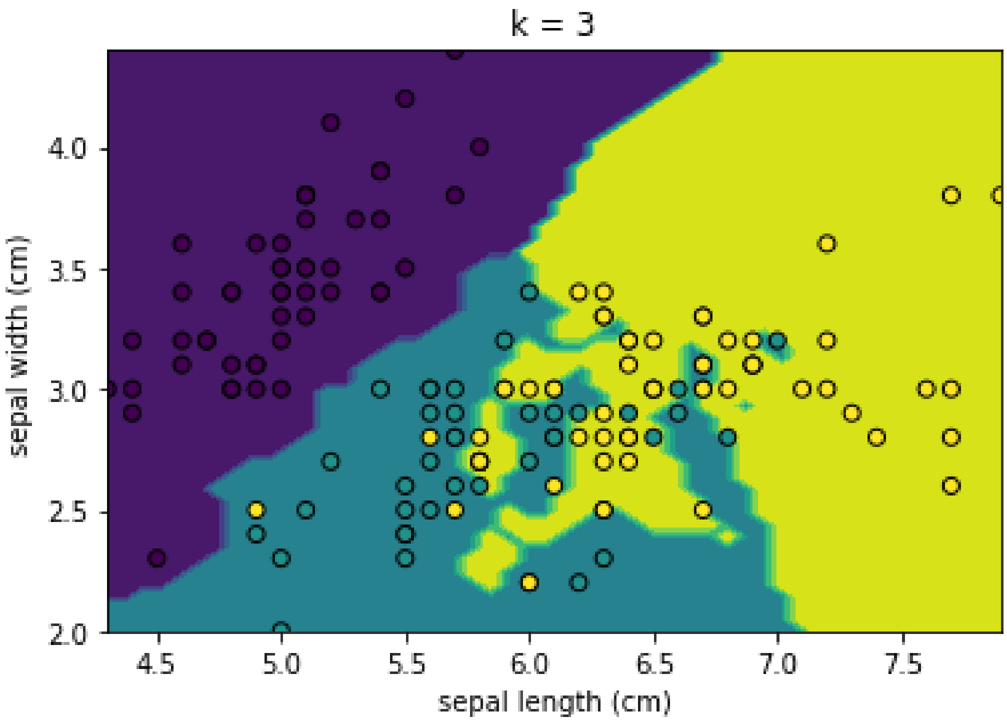
[33]:
```
### Insert your code here
import numpy as np
from sklearn.inspection import DecisionBoundaryDisplay

feat_1, feat_2 = np.meshgrid(
    np.linspace(X2[:, 0].min(), X2[:, 0].max()),
    np.linspace(X2[:, 1].min(), X2[:, 1].max())
)
grid = np.vstack([feat_1.ravel(), feat_2.ravel()]).T
```

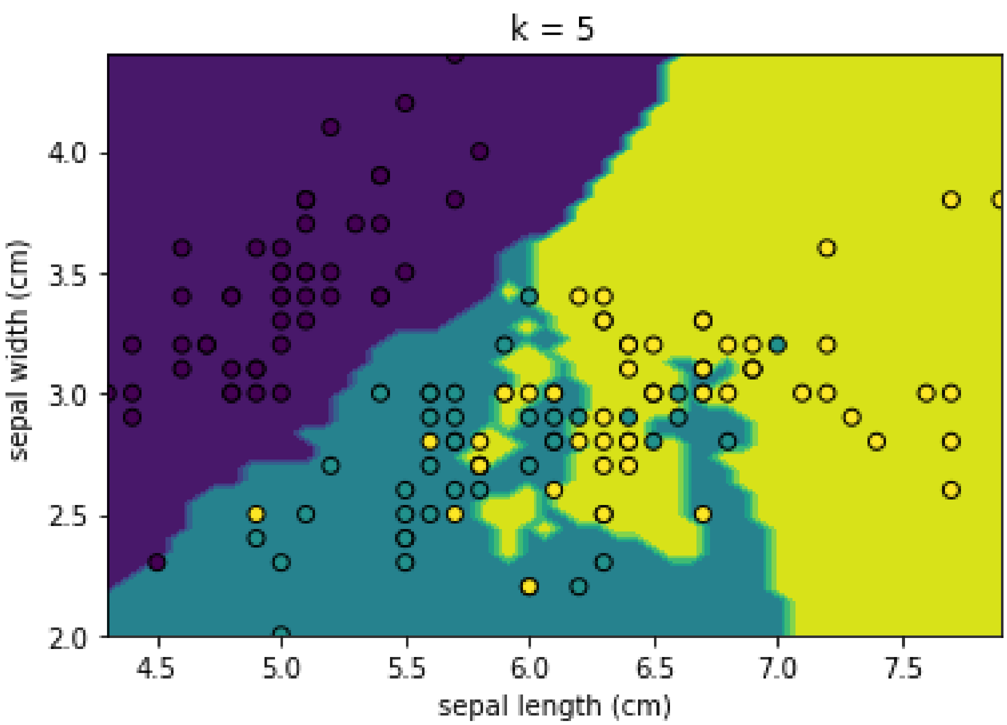
[34]:
```
ax_ = [[0,0], [0,1], [0,2],
       [1,0], [1,1], [1,2],
       [2,0], [2,1], [2,2]]
```

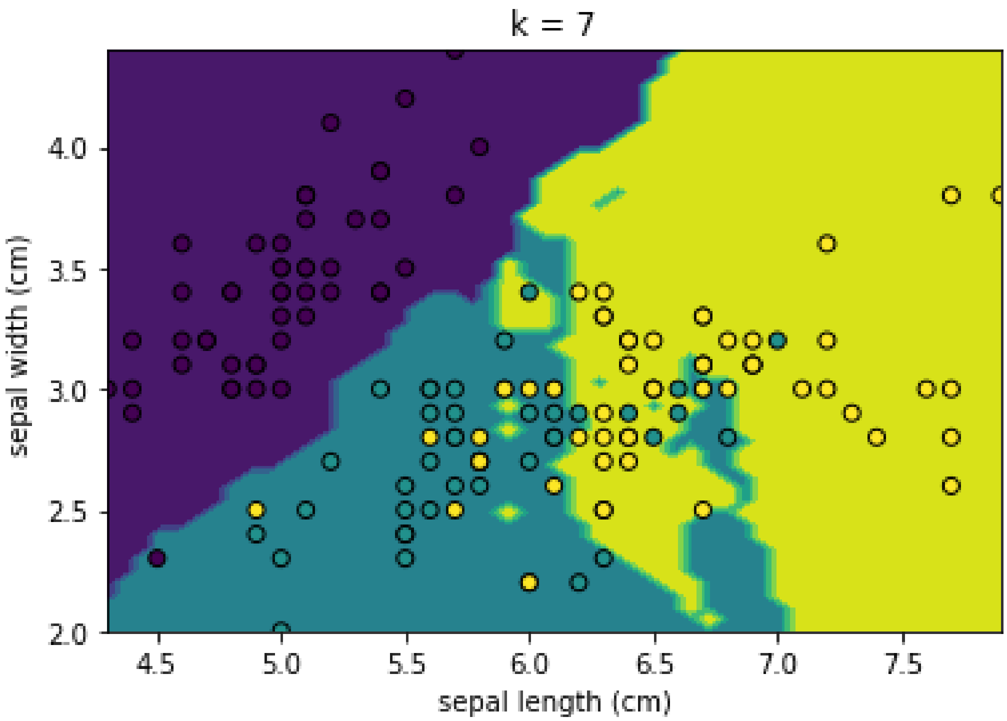
```
def get_pred():
    y = []
    for k in k_list:
        model = KNeighborsClassifier(n_neighbors=k)
        model.fit(X2, iris.target)
        y_pred = np.reshape(model.predict(grid), feat_1.shape)
        y.append(y_pred)
    return y

y = get_pred()
```

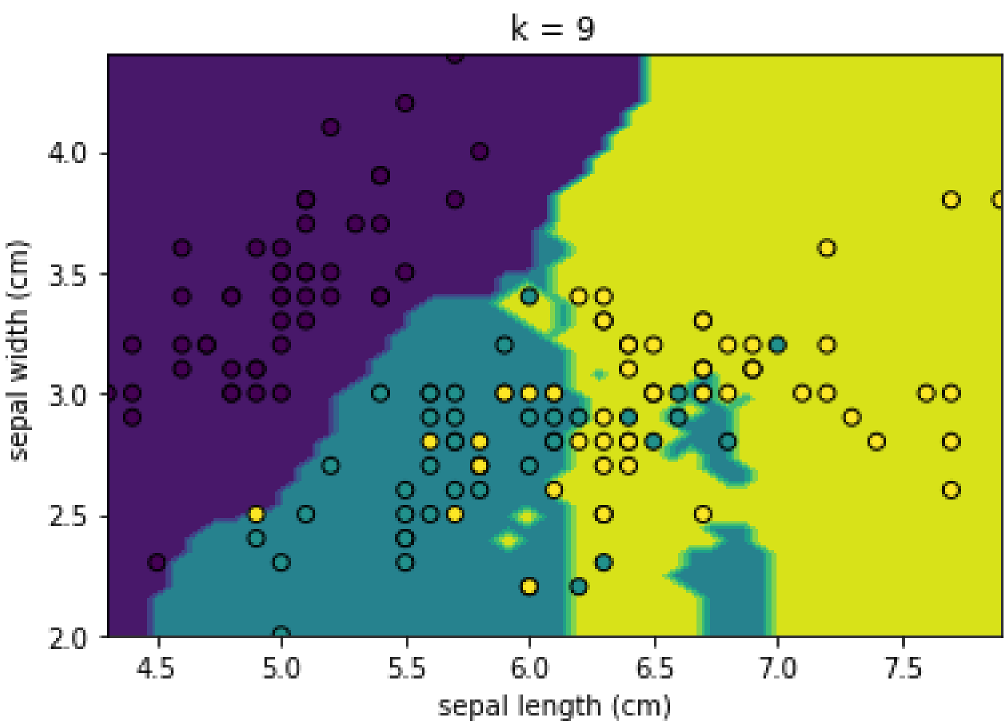
[35]:
```
for i in range(len(k_list)):
    display = DecisionBoundaryDisplay(
        xx0=feat_1, xx1=feat_2, response=y[i],
        xlabel=iris.feature_names[0],
        ylabel=iris.feature_names[1],
    )
    display.plot()
    display.ax_.scatter(
        iris.data[:, 0], iris.data[:, 1], c=iris.target, edgecolor="black"
    )
    plt.title(f"k = {k_list[i]}")
```

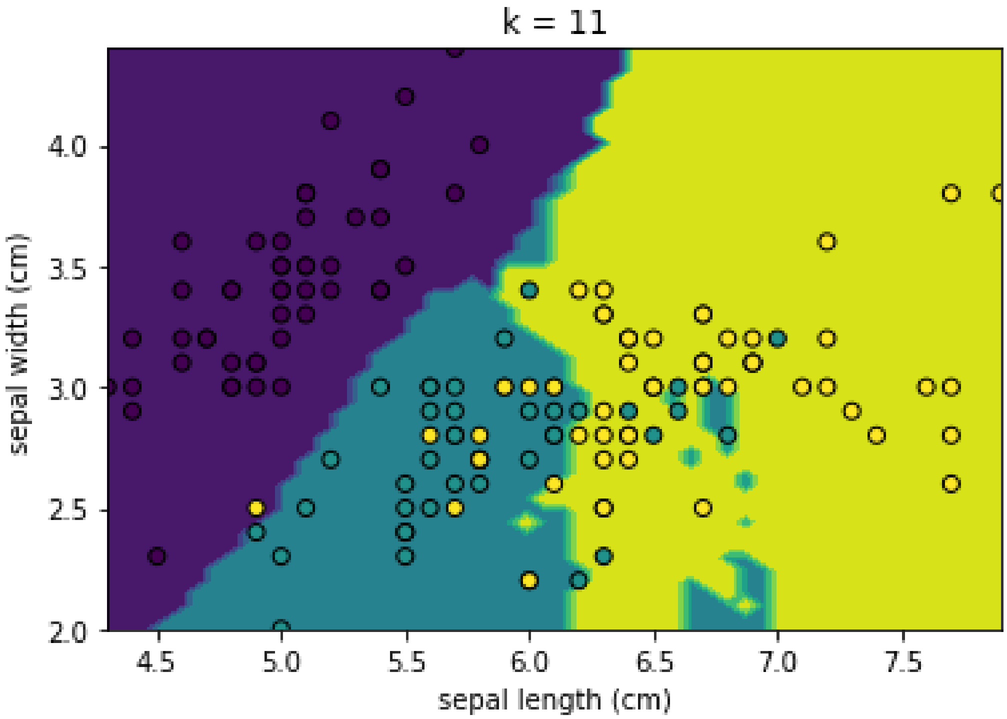


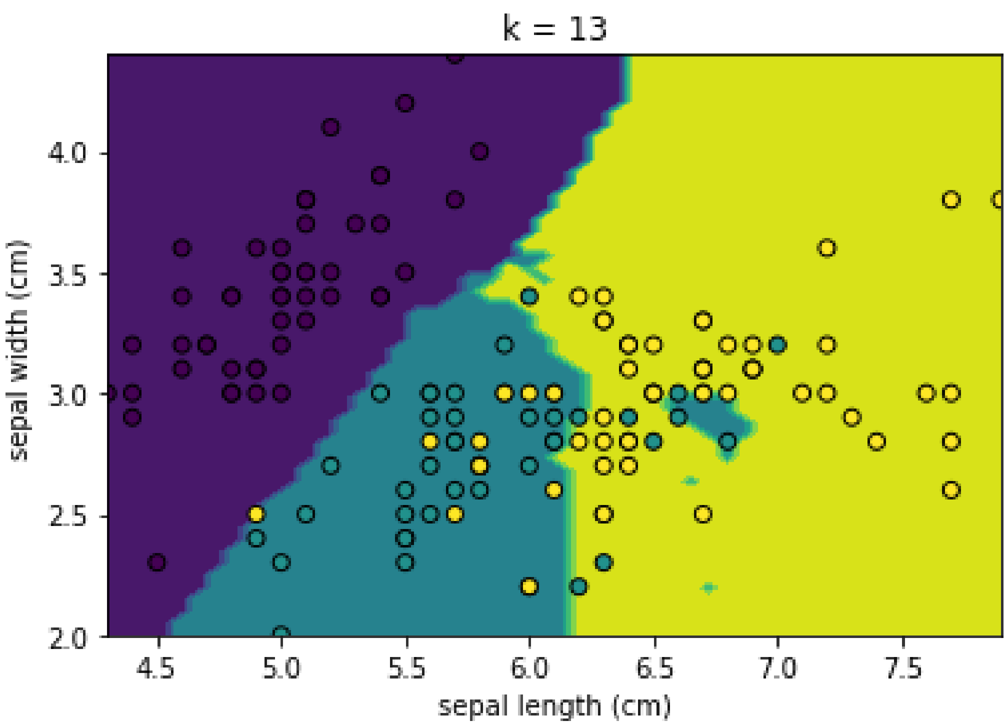

k = 3
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5

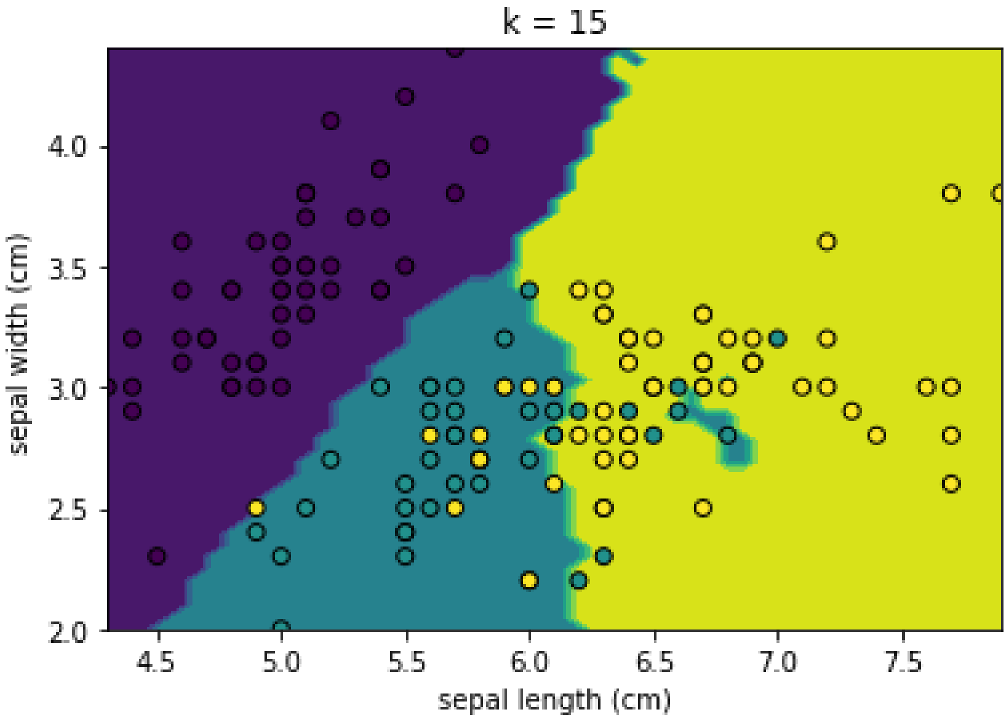

k = 5
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5

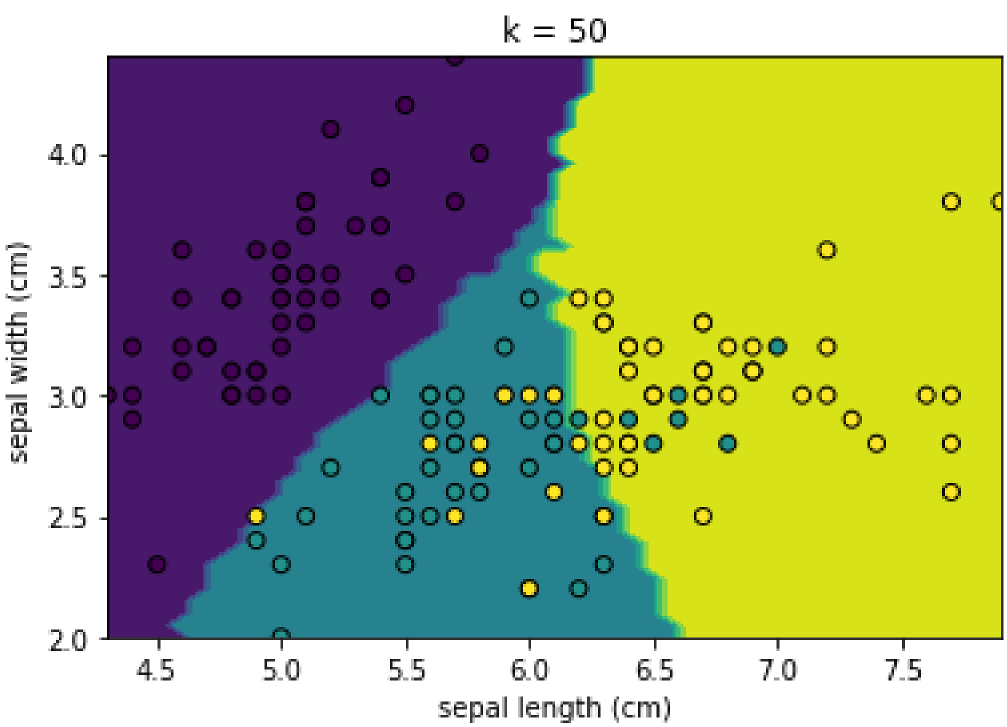

k = 7
4.0
3.5
3.0
2.5
2.0
sepal width (cm)
4.5
5.0
5.5
6.0
6.5
7.0
7.5
sepal length (cm)


k = 9
4.0
3.5
3.0
2.5
2.0
sepal width (cm)
4.5
5.0
5.5
6.0
6.5
7.0
7.5
sepal length (cm)


k = 11
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5


k = 13
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5


k = 15
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5


k = 50
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5