K-Nearest Neighbors

Created by Philip Graff for EN.601.475/675 Spring 2020

In [1]:
```
# import required packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import datasets
from scipy import stats
```

Initialize

We begin by loading the data and defining the model.

In [2]:
```
# load iris data
iris = datasets.load_iris()
```

In [3]:
```
# look at first two dimensions
X = iris.data[:, :2]
y = iris.target
```

In [4]:
```
X[:5,:]
```

Out[4]:
```
array([[5.1, 3.5],
       [4.9, 3. ],
       [4.7, 3.2],
       [4.6, 3.1],
       [5. , 3.6]])
```

In [5]:
```
class kNN(object):

    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X = X
        self.y = y

    def predict(self, x):
        distance = np.linalg.norm(self.X - x, axis=1)
        k_nearest = np.argsort(distance)[:self.k]
        k_nearest_vals = self.y[k_nearest]
        return stats.mode(k_nearest_vals)[0][0]
```

Decision Bounday

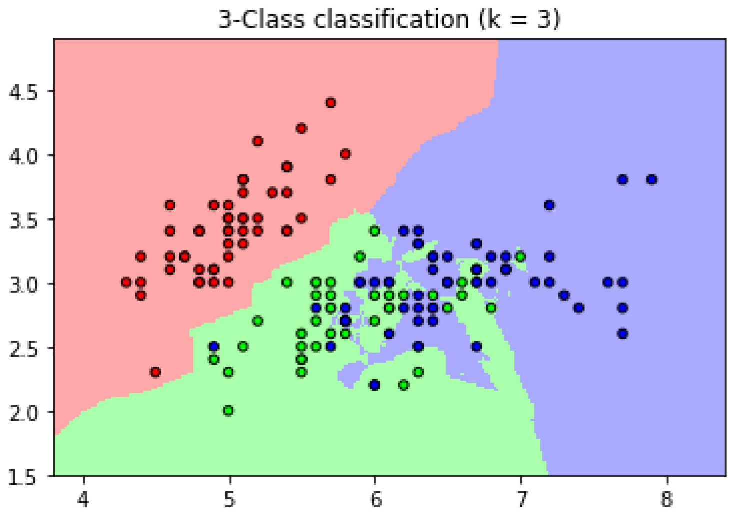
We want to investigate the decision boundary, so we:

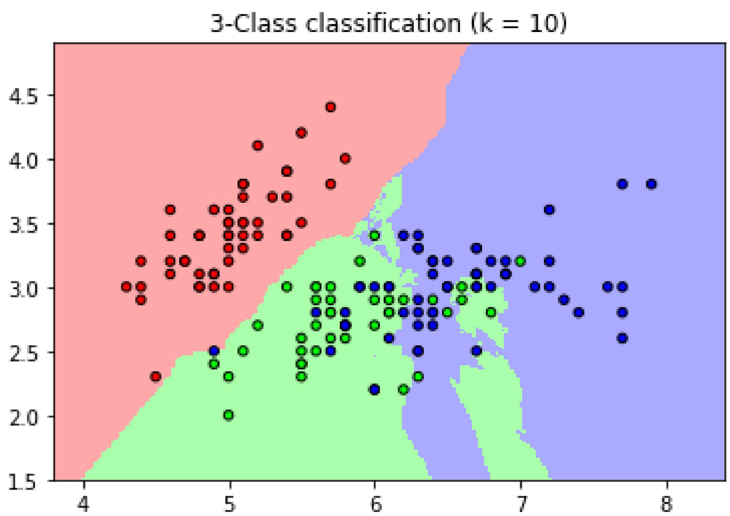
1. Train the model
2. Define a mesh in the parameter space
3. Evaluate the model along the mesh
4. Plot results
5. Repeat 1-4 for different model settings (k)

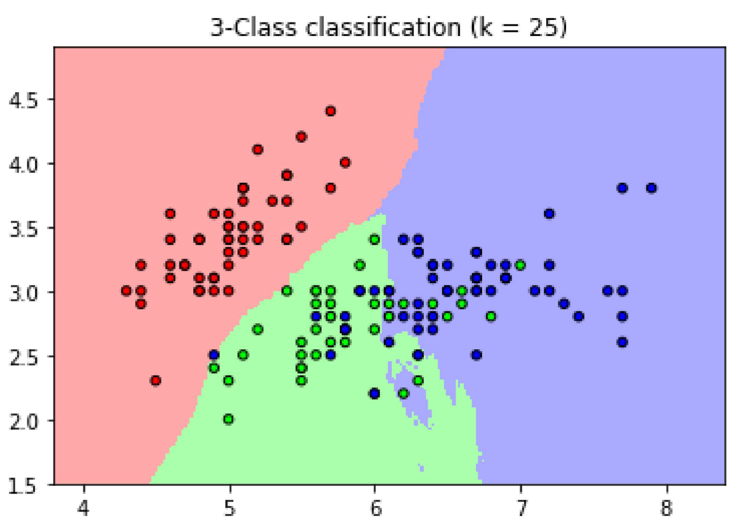
In [6]:
```
# step size in the mesh
h = .02

# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

In [7]:
```
def plotBoundary(n_neighbors):
    # build and fit the model
    model = kNN(n_neighbors)
    model.fit(X[:,:2], y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h
))
    Xin = np.c_[xx.ravel(), yy.ravel()]
    Z = np.array([model.predict(Xin[i,:2]) for i in range(Xin.shape[0])])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i)" % (n_neighbors))
    plt.show()
```

In [8]:
```
for n in [3, 10, 25]:
    plotBoundary(n)
```







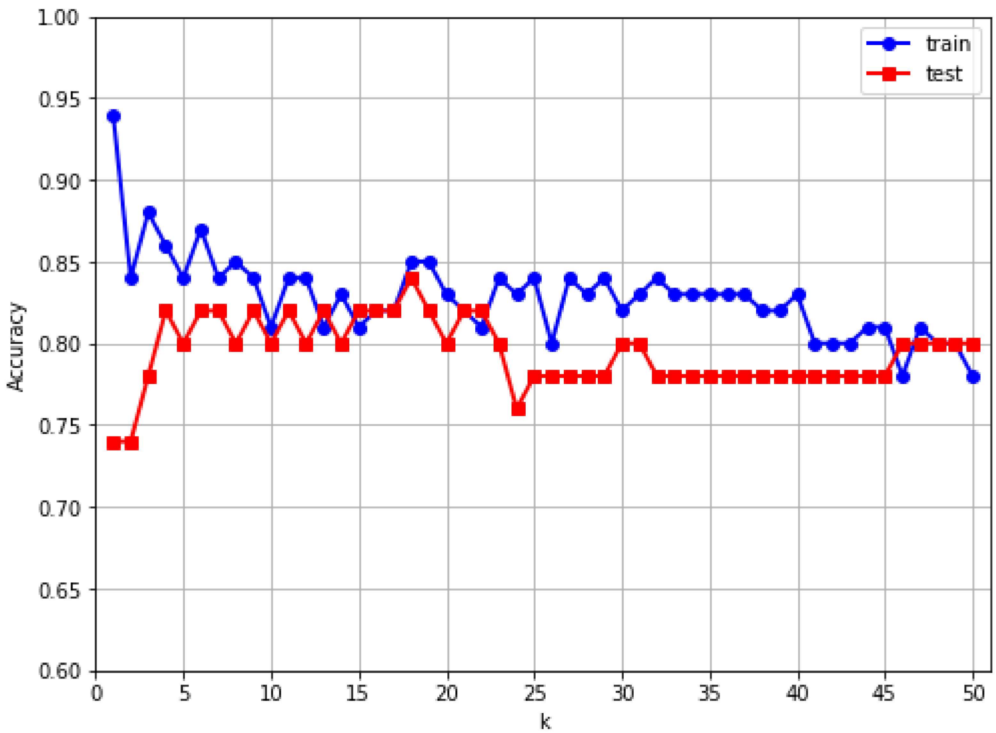
Accuracy

To measure how well this model can predict and find the optimal value for `k` , let's look at its prediction accuracy.

In [9]:
```
def getAccuracy(n, train, test):
    model = kNN(n)
    model.fit(X[train], y[train])
    train_pred = np.array([model.predict(X[train[i]]) for i in range(len(train
))])
    test_pred = np.array([model.predict(X[test[i]]) for i in range(len(test
))])
    train_acc = np.sum(train_pred == y[train]) / (1.0 * len(train))
    test_acc = np.sum(test_pred == y[test]) / (1.0 * len(test))
    return train_acc, test_acc
```

In [10]:
```
t = np.arange(X.shape[0])
np.random.shuffle(t)
train_size = 100
train = t[:train_size]
test = t[train_size:]
train_acc = []
test_acc = []
kmax = 50
krange = range(1,kmax+1)
for n in krange:
    xa, ya = getAccuracy(n, train, test)
    train_acc.append(xa)
    test_acc.append(ya)
```

In [11]:
```
plt.figure(figsize=(8,6))
plt.plot(krange, train_acc, '-ob', lw=2, label='train')
plt.plot(krange, test_acc, '-sr', lw=2, label='test')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.xlim([0,kmax+1])
plt.ylim([0.6,1])
plt.xticks(np.arange(kmax+1,step=5))
plt.grid()
plt.show()
```

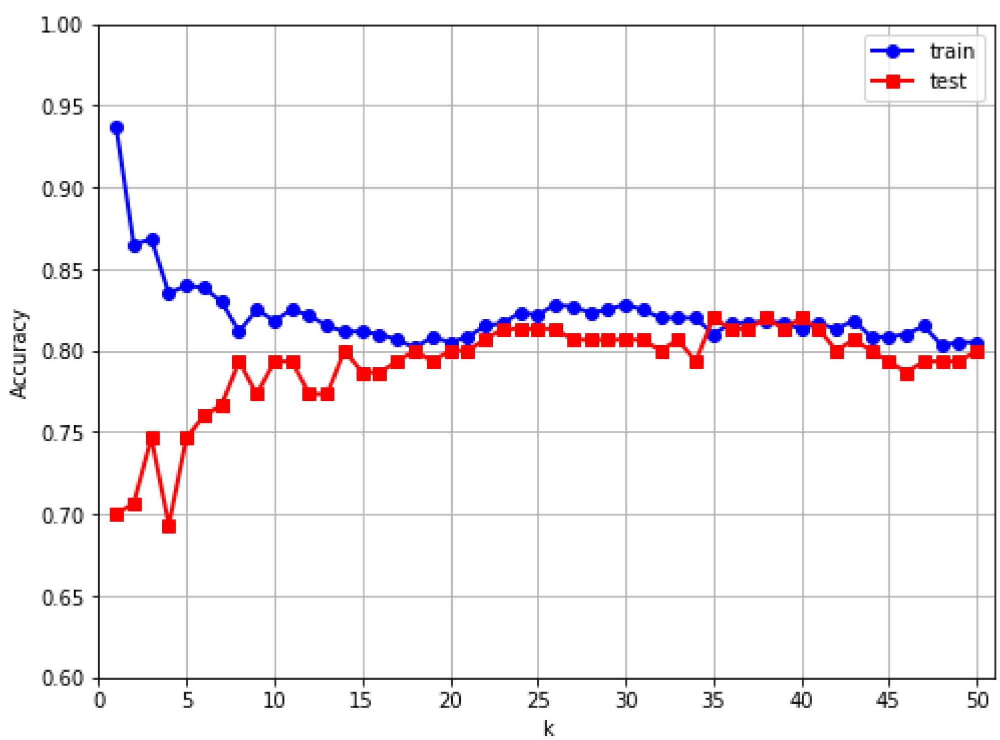


Cross Validation

The evaluation of accuracy has a lot of noise to it. We can use cross validation -- in this case with 5 folds -- to evaluate the accuracy in a more stable manner.

In [12]:
```
t = np.arange(X.shape[0])
train_acc_f = []
test_acc_f = []
nfolds = 5
kmax_f = 50
krange_f = range(1,kmax_f+1)
for n in krange_f:
    fold_train_acc = 0.0
    fold_test_acc = 0.0
    for fold in range(nfolds):
        train = t[t % nfolds != fold]
        test = t[t % nfolds == fold]
        xa, ya = getAccuracy(n, train, test)
        fold_train_acc += xa / (1.0 * nfolds)
        fold_test_acc += ya / (1.0 * nfolds)
    train_acc_f.append(fold_train_acc)
    test_acc_f.append(fold_test_acc)
```

In [13]:
```
plt.figure(figsize=(8,6))
plt.plot(krange_f, train_acc_f, '-ob', lw=2, label='train')
plt.plot(krange_f, test_acc_f, '-sr', lw=2, label='test')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.xlim([0,kmax_f+1])
plt.ylim([0.6,1])
plt.xticks(np.arange(kmax_f+1,step=5))
plt.grid()
plt.show()
```



In [ ]: