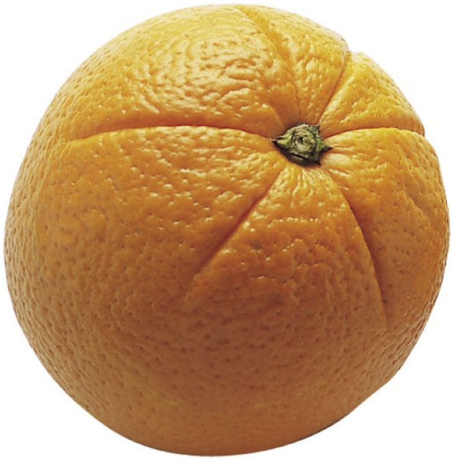
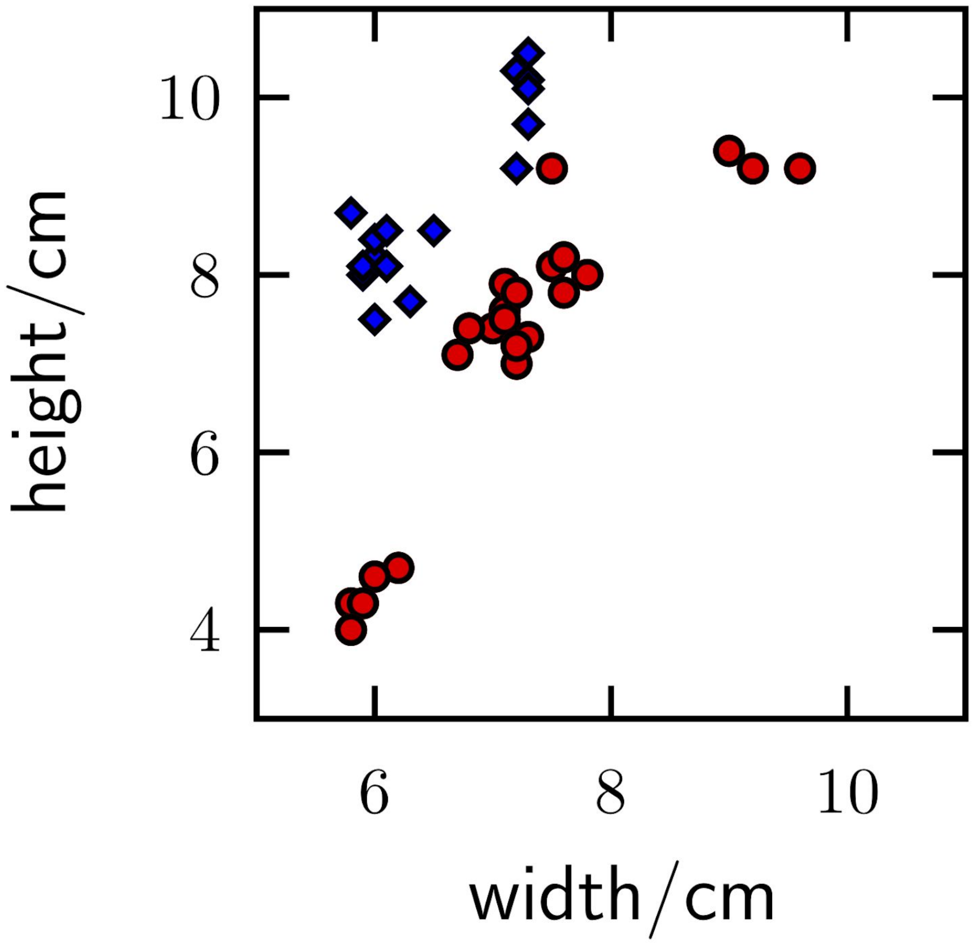
Machine Learning & Artificial Intelligence for Data Scientists: Classification (Part1)

Ke Yuan
https://kyuanlab.org/
School of Computing Science

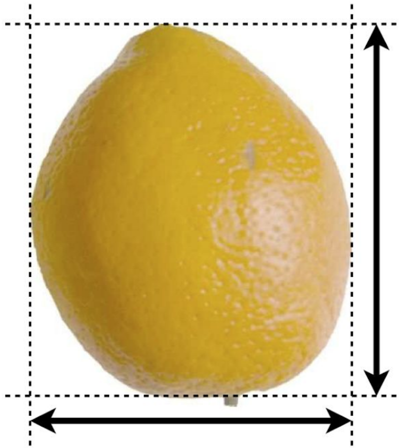
Again some data, and a problem



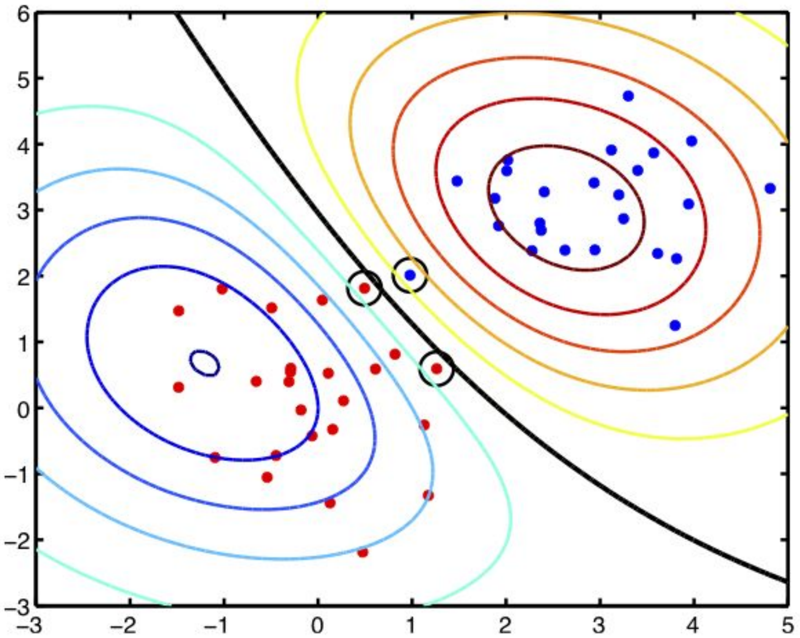
Source: https://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/


height/cm
width/cm
4
6
8
10
6
8
10


Oranges:
Lemons:



Classification



- A set of N objects with attributes (usually vector) $\mathbf{x}_n$.
- Each object has an associated response (or label) t_n.
- Binary classification: $t_n = \{0, 1\}$ or $t_n = \{-1, 1\}$,
  - (depends on algorithm).
- Multi-class classification: $t_n = \{1, 2, \ldots, K\}$.

Probabilistic v non-probabilistic classifiers

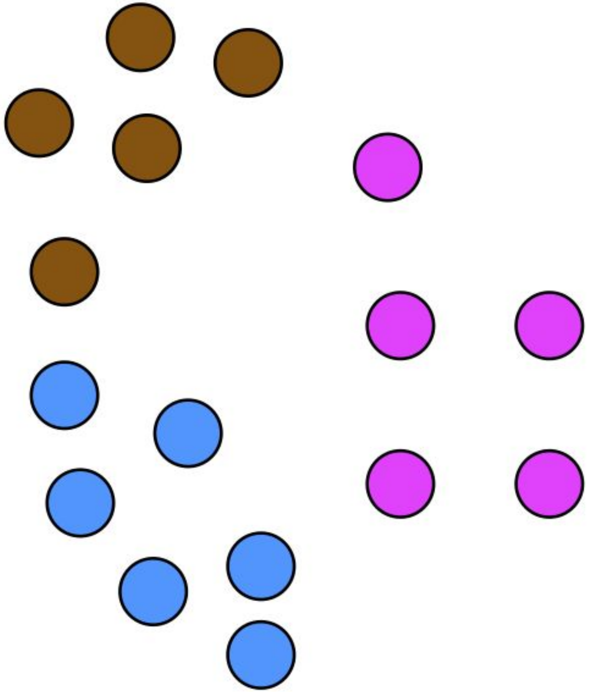
Classifier is trained on $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and $t_1, \ldots, t_N$ and then used to classify $\mathbf{x}_{\text{new}}$.

- Probabilistic classifiers produce a probability of class membership $P(t_{\text{new}} = k | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t})$
  - e.g. binary classification: $P(t_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t})$ and $P(t_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t})$.
- Non-probabilistic classifiers produce a hard assignment
  - e.g. $t_{\text{new}} = 1$ or $t_{\text{new}} = 0$.
- Which to choose depends on application....

Probabilistic v non-probabilistic classifiers

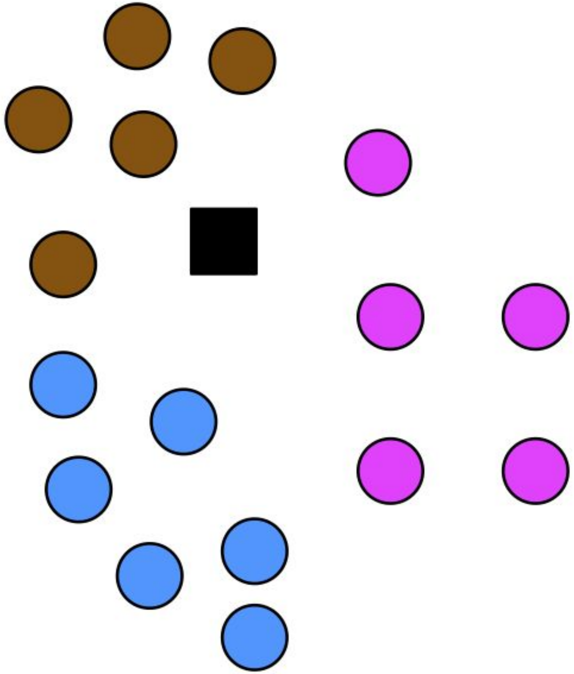
- Probabilities provide us with more information – $P(t_{\text{new}} = 1) = 0.6$ is more useful than $t_{\text{new}} = 1$.
  - Tells us how **sure** the algorithm is.
- Particularly important where cost of misclassification is high and imbalanced.
  - e.g. Diagnosis: telling a diseased person they are healthy is much worse than telling a healthy person they are diseased.
- Extra information (probability) often comes at a cost.

Algorithm 1: K-Nearest Neighbours (KNN)



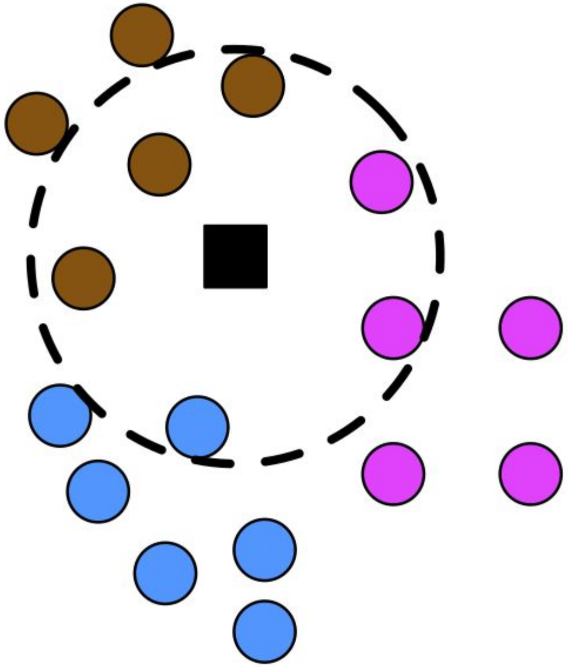
Training data from 3 classes.

KNN



Test point.

KNN

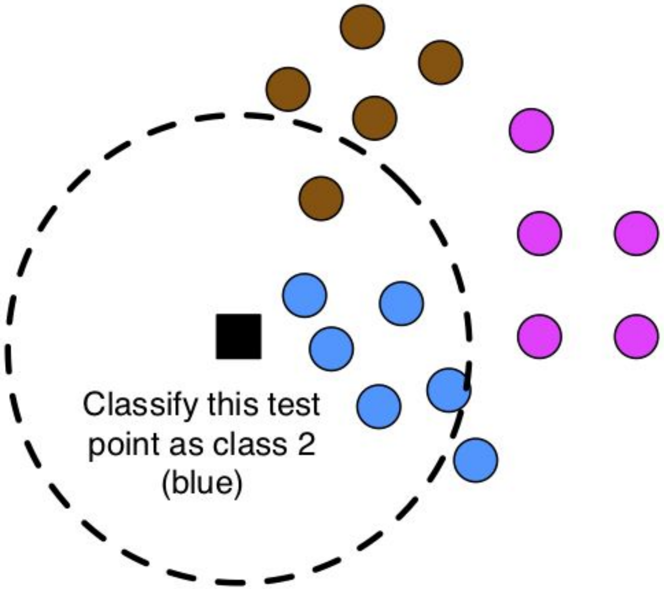


Find $K = 6$ nearest neighbours.

KNN

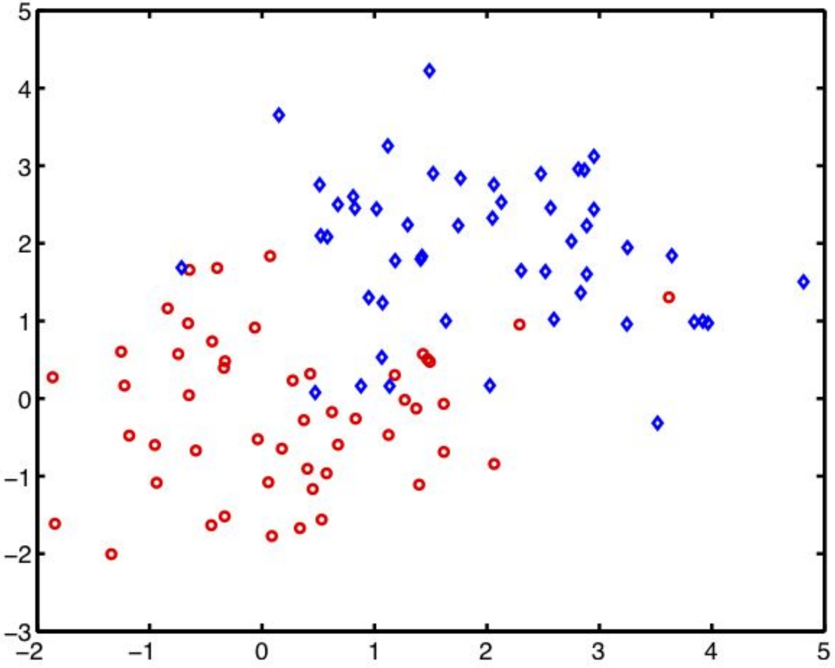
Class one has most votes – classify $\mathbf{x}_{new}$ as belonging to class 1.

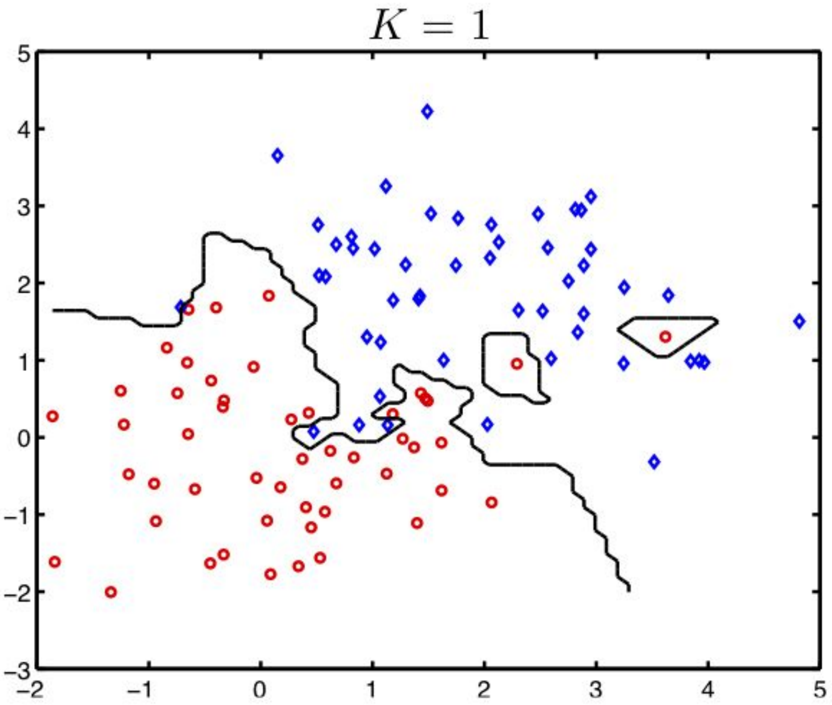
KNN



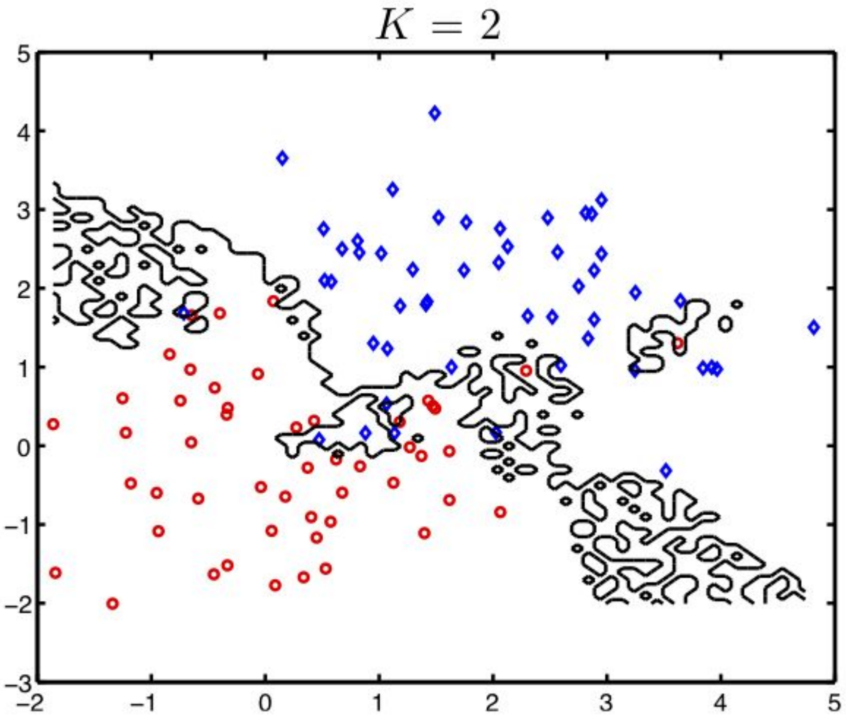
Second example – class 2 has most votes.

KNN: Binary data

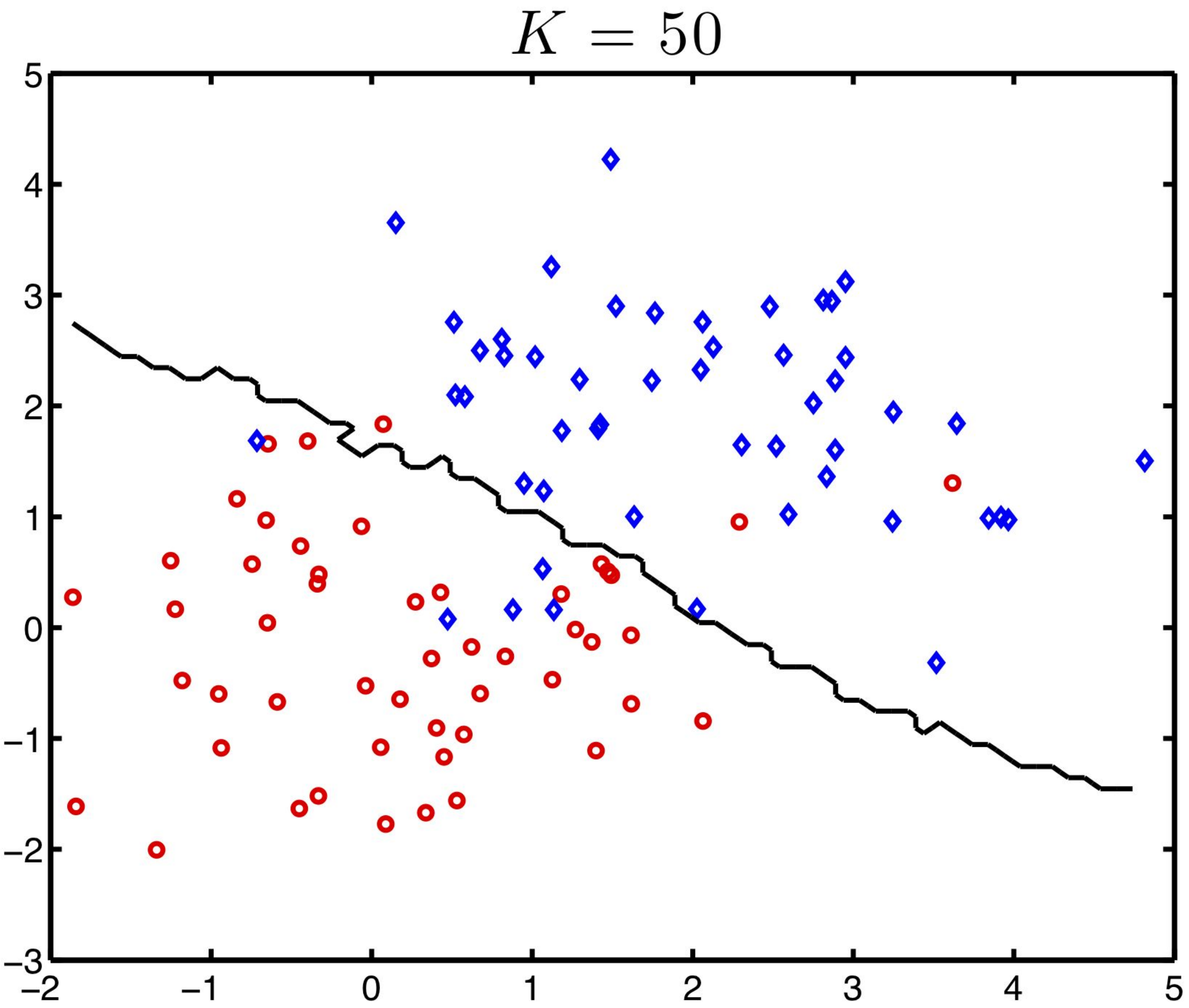




- 1-Nearest Neighbour.
- Line shows decision boundary.
- Too complex – should the islands exist?



- 2-Nearest Neighbour.
- What's going on?
- Lots of ties – random guessing.



$K = 50$
5
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
4
5

Problem with KNN

- Class imbalance
  - As K increases, small classes will disappear!
  - Imagine we had only 5 training objects for class 1 and 100 for class 2.
  - For $K \geq 11$, class 2 will **always** win!
- How do we choose K?
  - Right value of K will depend on data.
  - Cross-validation!

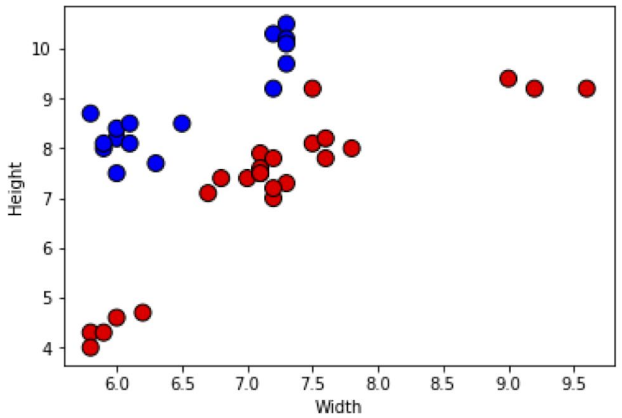
In [2]:
```python
import numpy as np
%matplotlib inline
import pylab as plt
from matplotlib.colors import ListedColormap

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

data = np.loadtxt('orange_lemon.txt', delimiter=',') # load fruit data
X = data[:,1:3]
t = data[:,0]
plt.scatter(X[:, 0], X[:, 1], c=t, cmap=cmap_bold, edgecolor='k', s=100)
plt.xlabel('Width')
plt.ylabel('Height')
```

Out[2]:
```
Text(0, 0.5, 'Height')
```

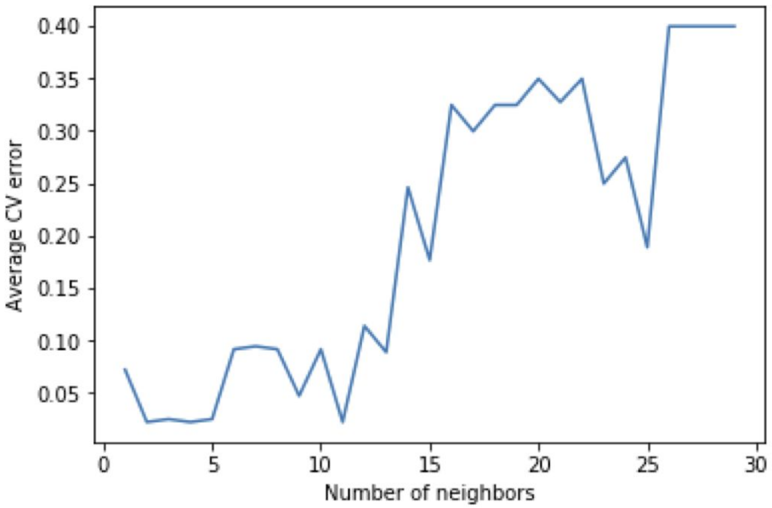
We can see it with oranges and lemons



```python
In [17]: cv_scores = []
         for i in range(1,30,1):
             knn_cv = KNeighborsClassifier(n_neighbors=i)
             cv_scores.append(1-np.mean(cross_val_score(knn_cv, X, t, cv=5)))

         plt.plot(np.arange(1,30,1),cv_scores)
         plt.xlabel('Number of neighbors')
         plt.ylabel('Average CV error')
         print(np.min(cv_scores))
```

```
0.0222222222222143
```

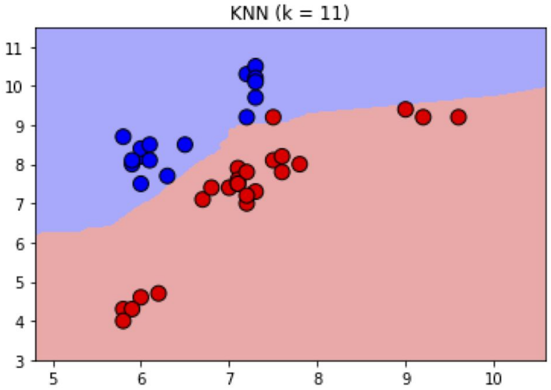


5-fold CV to select K

In [4]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

n_neighbors = 11
clf = KNeighborsClassifier(n_neighbors)
clf.fit(X, t)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=t, cmap=cmap_bold,
            edgecolor='k', s=100)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("KNN (k = %i)"
          % (n_neighbors))
```

Out[4]:
```
Text(0.5, 1.0, 'KNN (k = 11)')
```



In [3]:
```python
# 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].
h = .02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
```

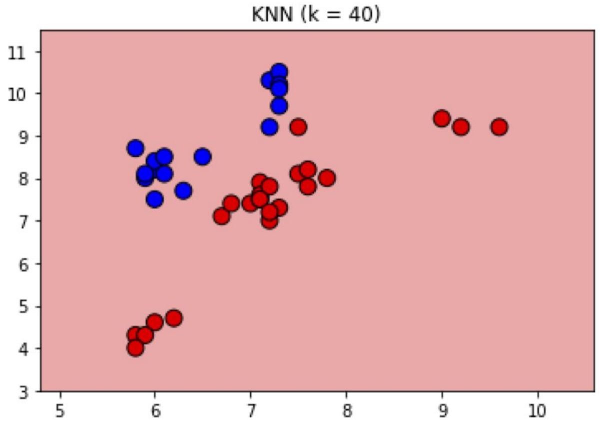
K = 11

In [6]:

```python
n_neighbors = 40
clf = KNeighborsClassifier(n_neighbors)
clf.fit(X, t)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=t, cmap=cmap_bold,
            edgecolor='k', s=100)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("KNN (k = %i)"
          % (n_neighbors))
```

Out[6]:

```
Text(0.5, 1.0, 'KNN (k = 40)')
```



K = 40

KNN summary

- Non-probabilistic.
- Fast.
- Only one parameter to tune (K).
- Important to tune it well....
- ...can use CV.
- There is a probabilistic version.
  - Not covered in this course.
- Now onto a (different) probabilistic classifier...