COMPSCI 5100
ML & AI for Data Science

Ali Gooya

Classification: Part II

Some slides are adapted from those of Dr. Ke Yuan

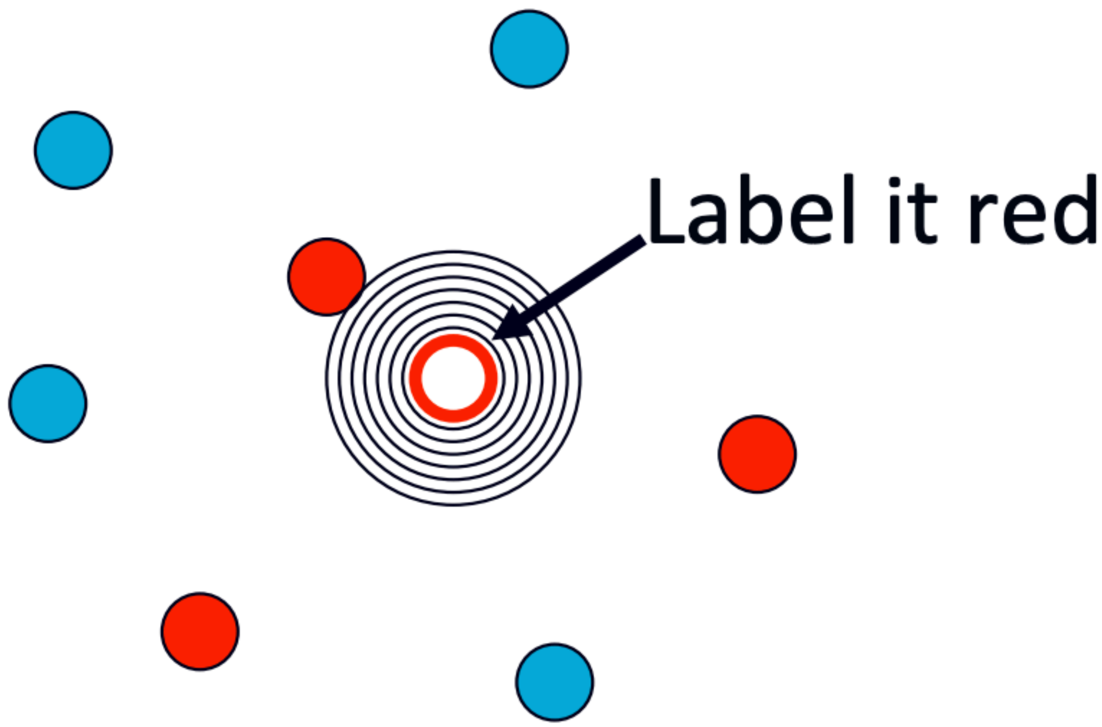
Notations

- N training samples (raw data or features): $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_N$
- Each sample associated with a label: $y_1, y_2, \cdots y_N$
- Binary classification: $y_n \in \{0,1\}$ where $n = 1, 2 \cdots N$
- Multiclass classification: $y_n \in \{1,2,\cdots C\}$
- Test sample: $\mathbf{x}_{new}$
- Task: Assign a label y_{new} to $\mathbf{x}_{new}$ where

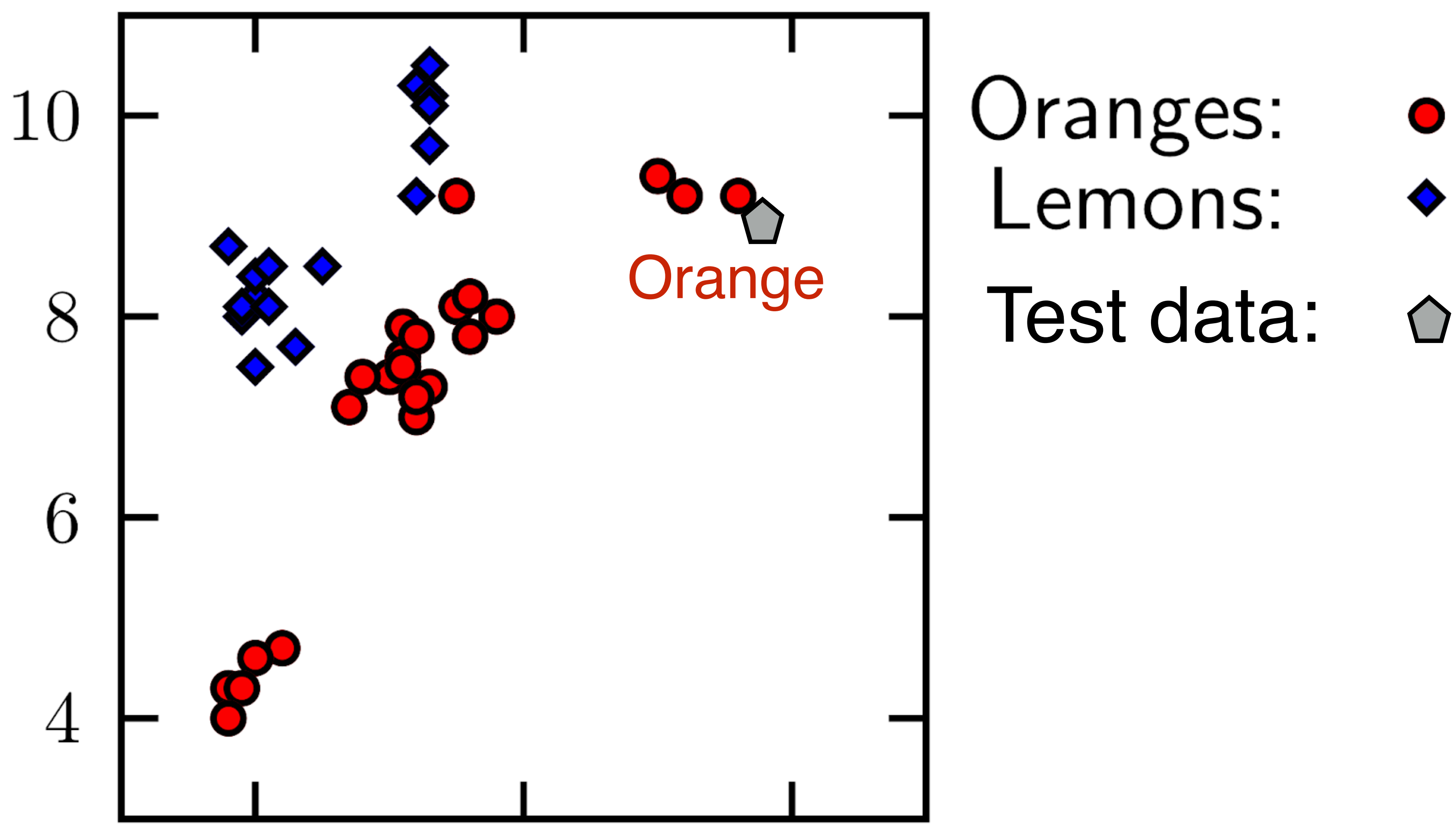
  $y_{new} \in \{1,2...C\}$ (multi-class) or $y_n \in \{0,1\}$ (if binary)

Nearest Neighbour

- Simplest of all classifiers: **1-Nearest Neighbour**
- **Simple idea:** Label a new sample the same as its closest data point



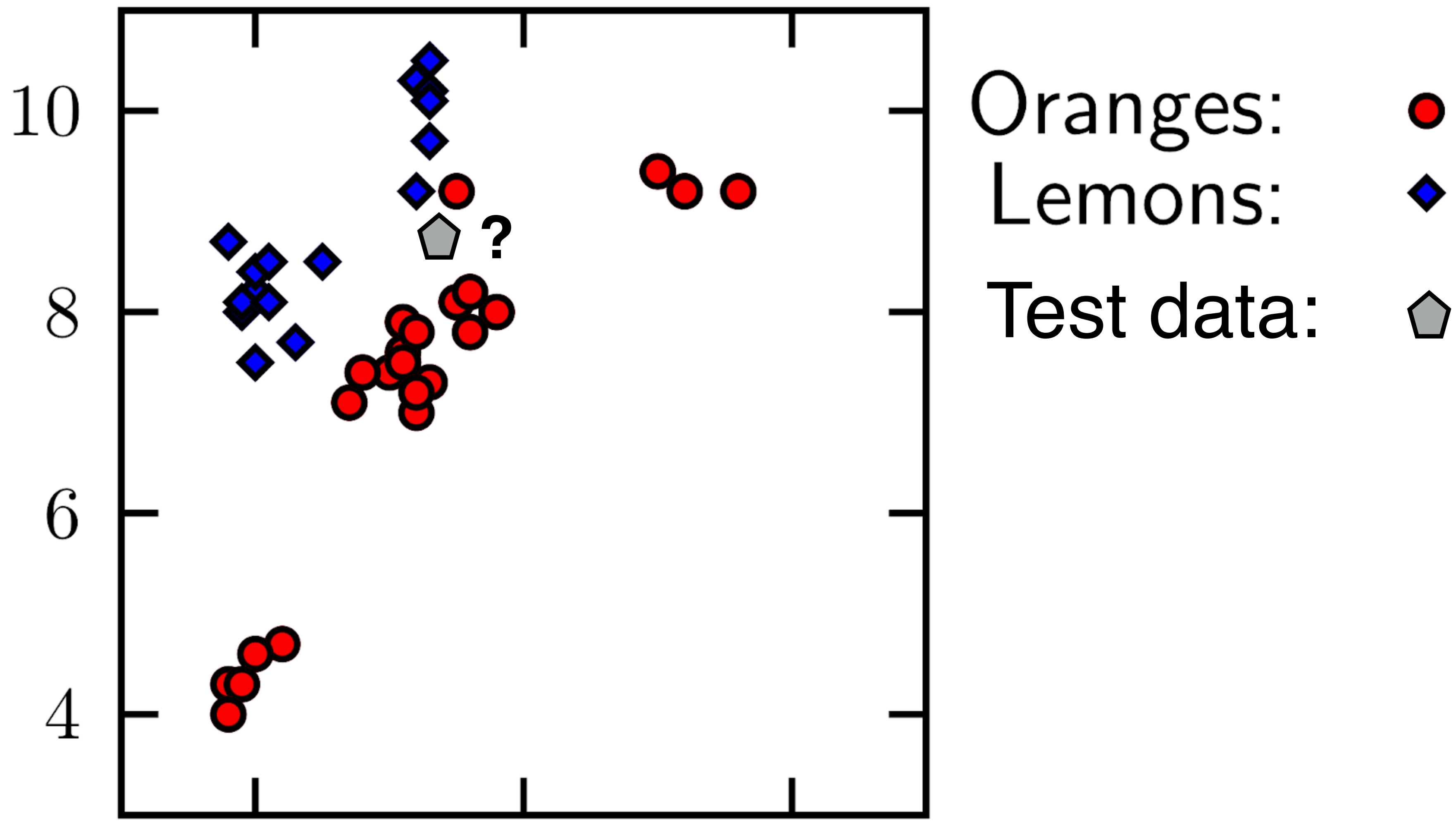
1-Nearest Neighbour



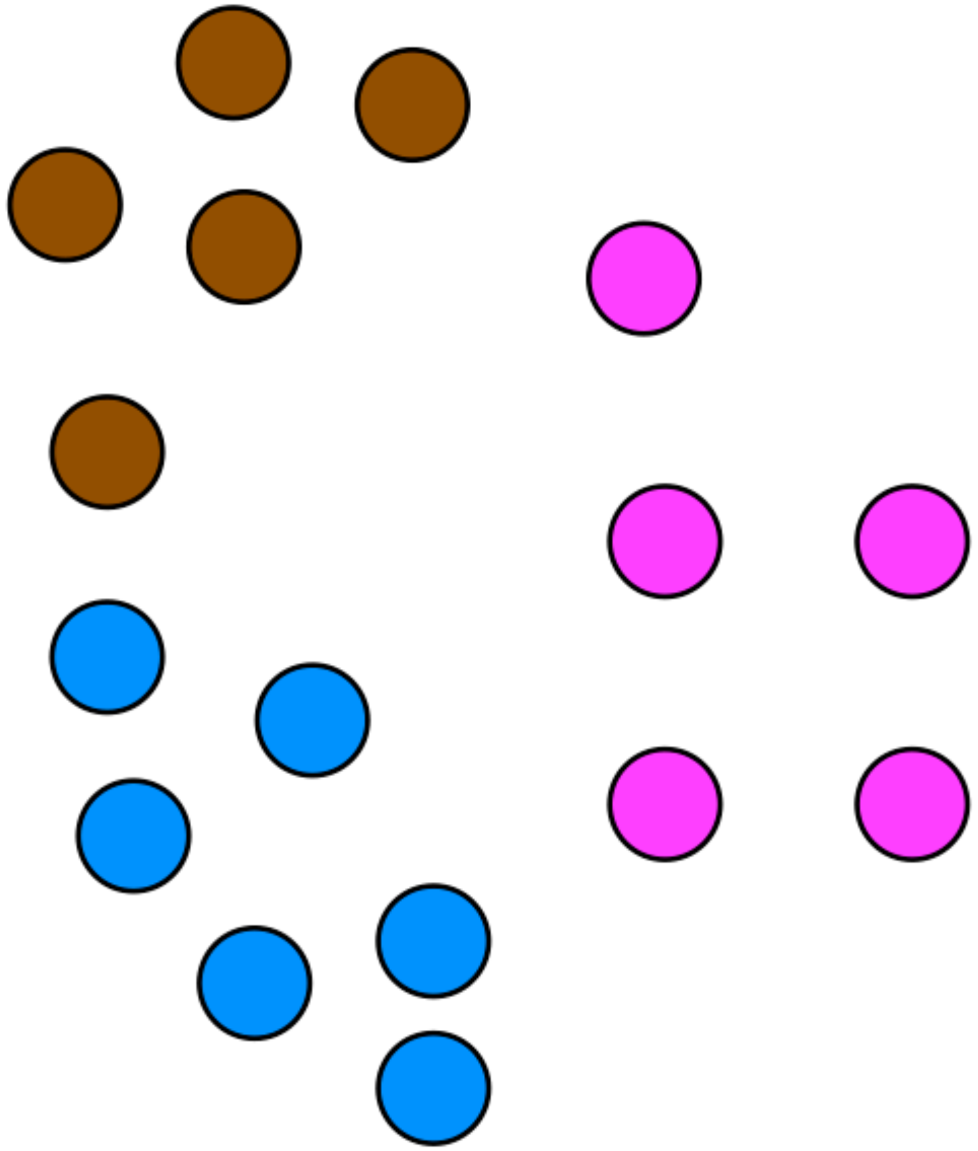
$$y_{new} \leftarrow y_{i*}$$

$$i* = \underset{i}{\operatorname{argmin}}\ dist(\mathbf{x}_i, \mathbf{x}_{new})$$

1-Nearest Neighbour

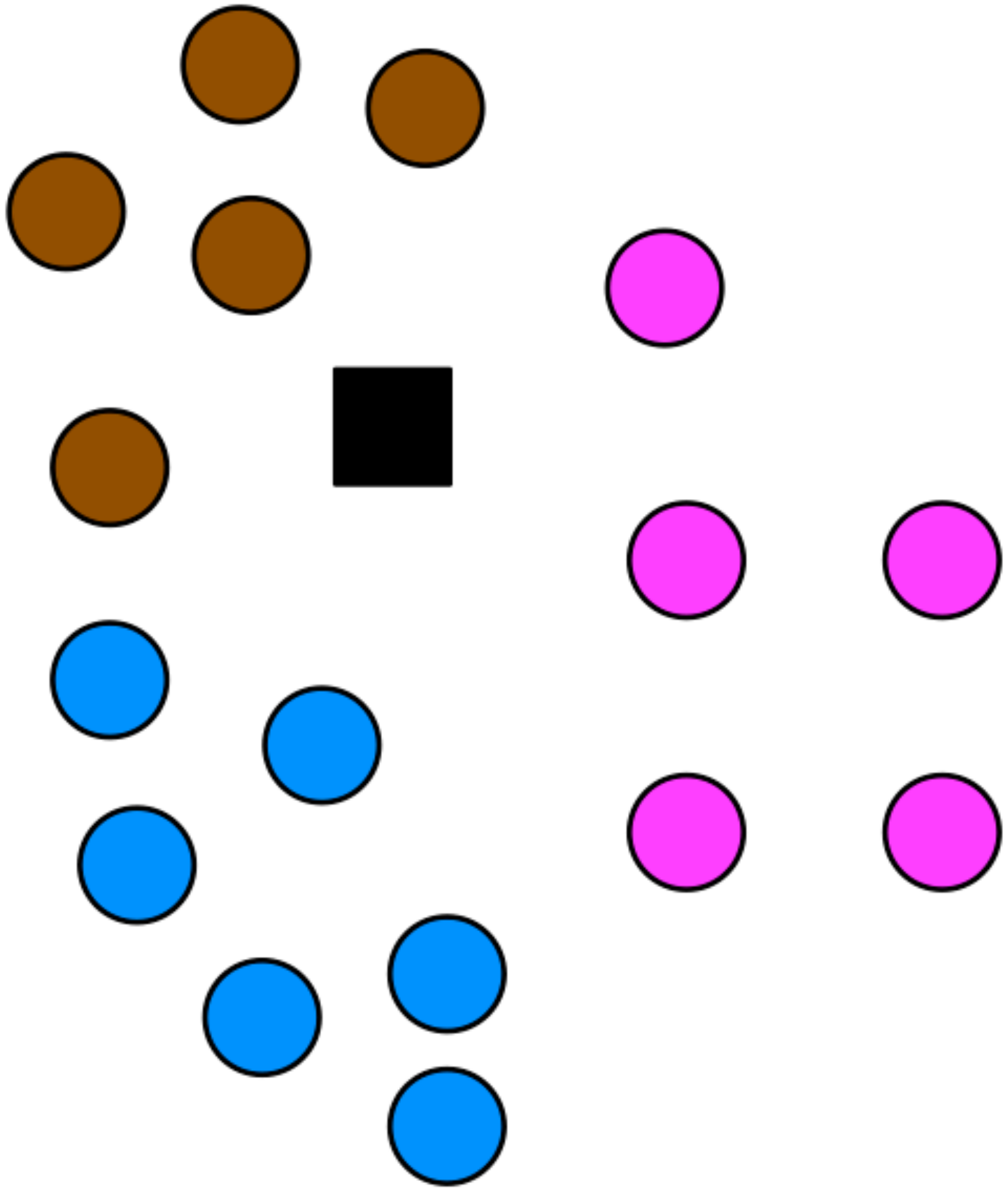


K-Nearest Neighbour



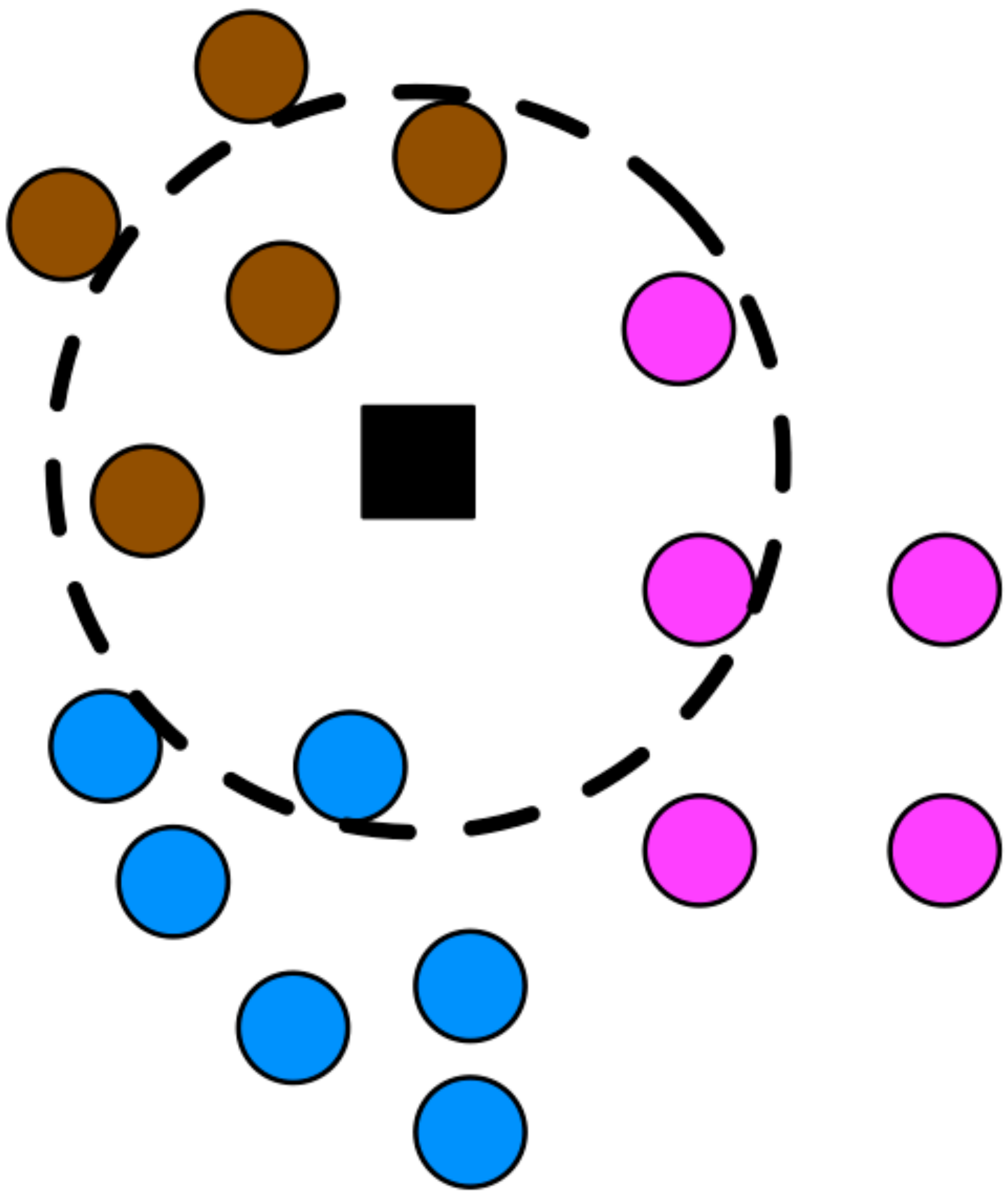
Training data from 3 classes.

K-Nearest Neighbour



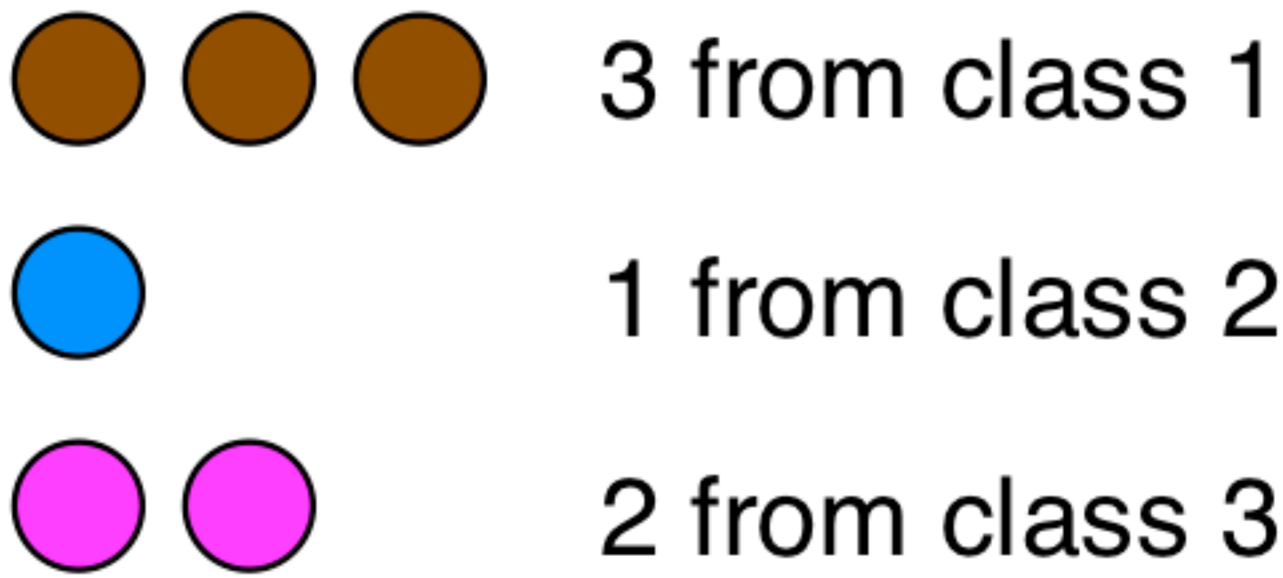
Test point.

K-Nearest Neighbour



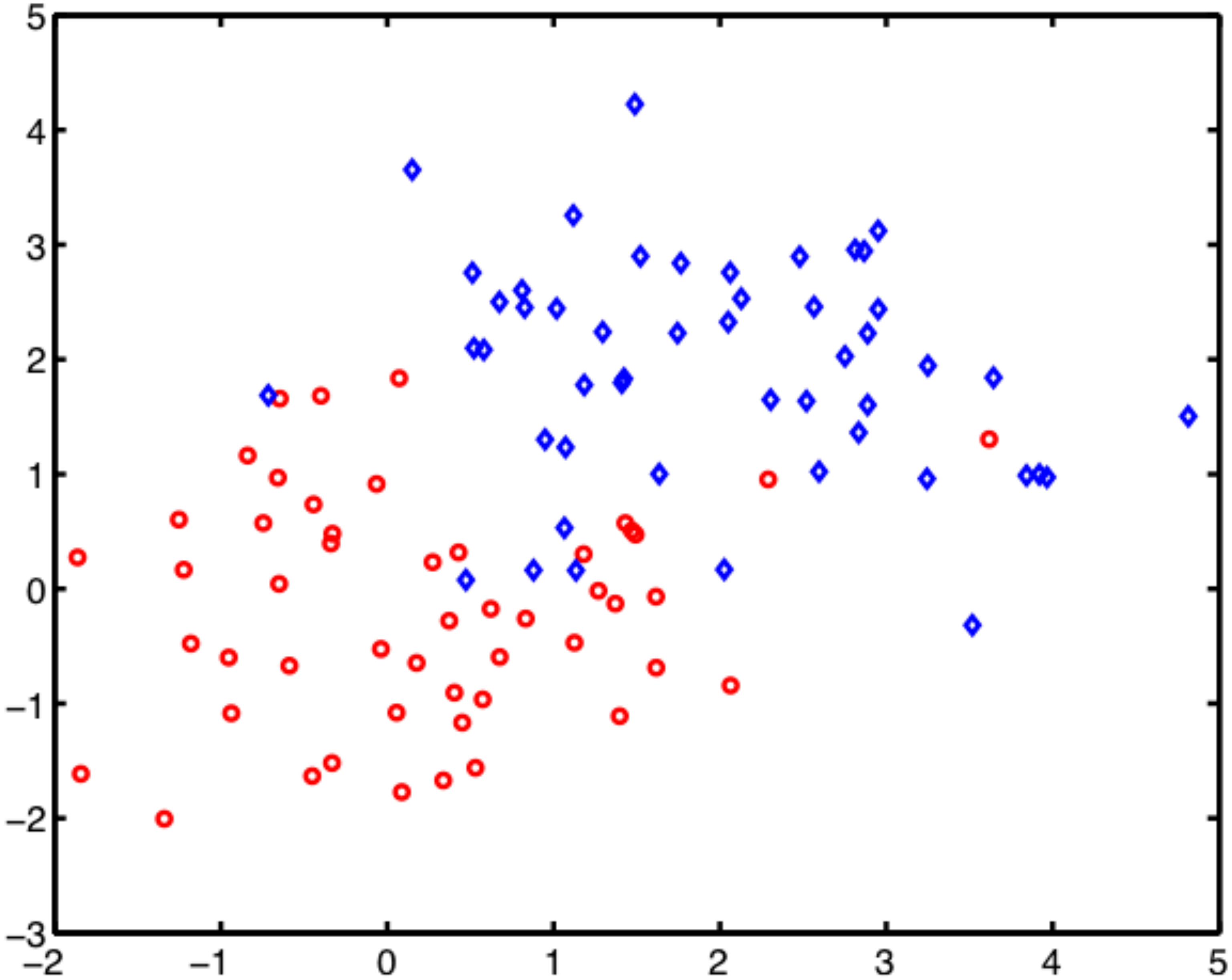
Find $K = 6$ nearest neighbours.

K-Nearest Neighbour

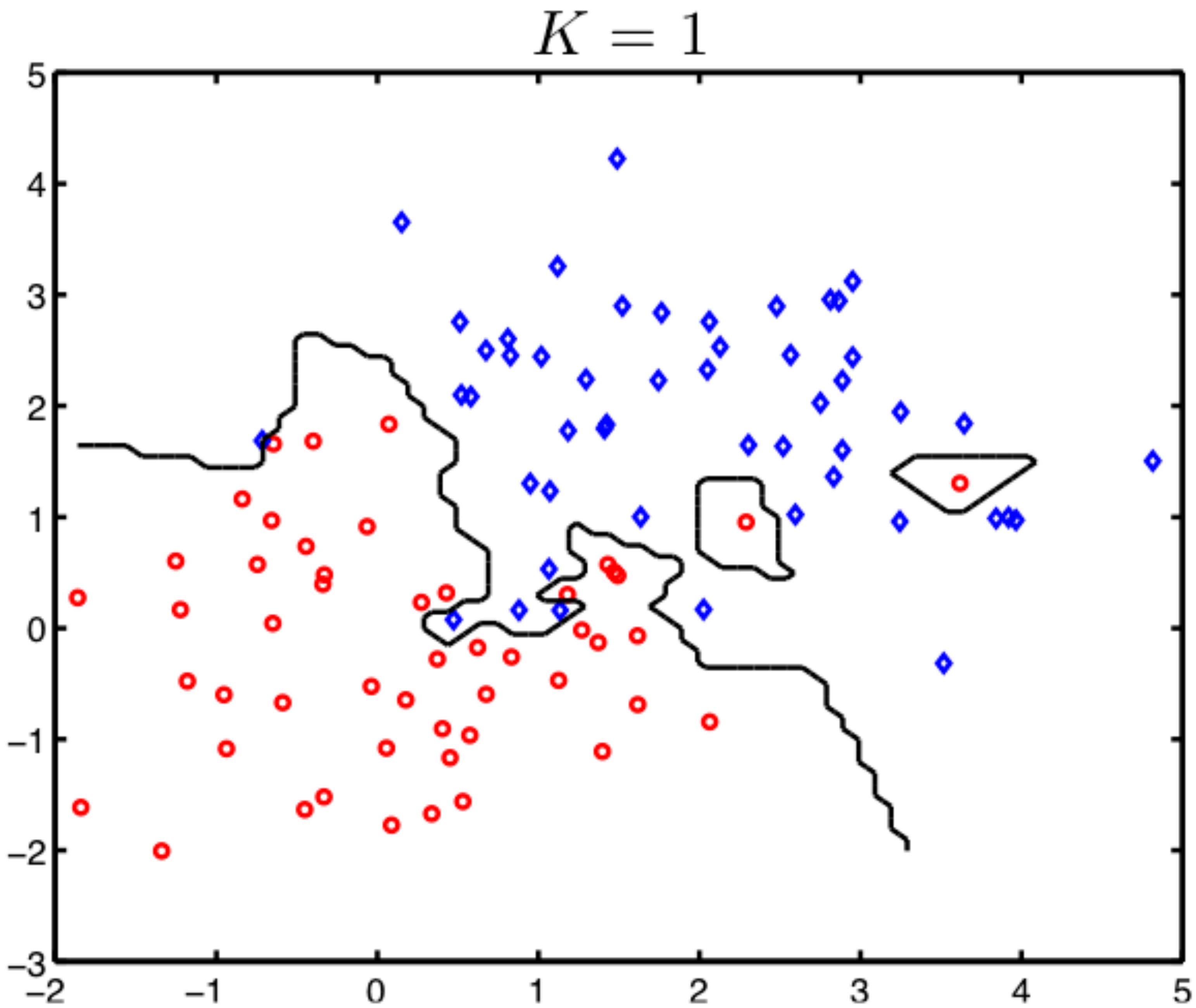


Class one has most votes – classify $\mathbf{x}_{new}$ as belonging to class 1.

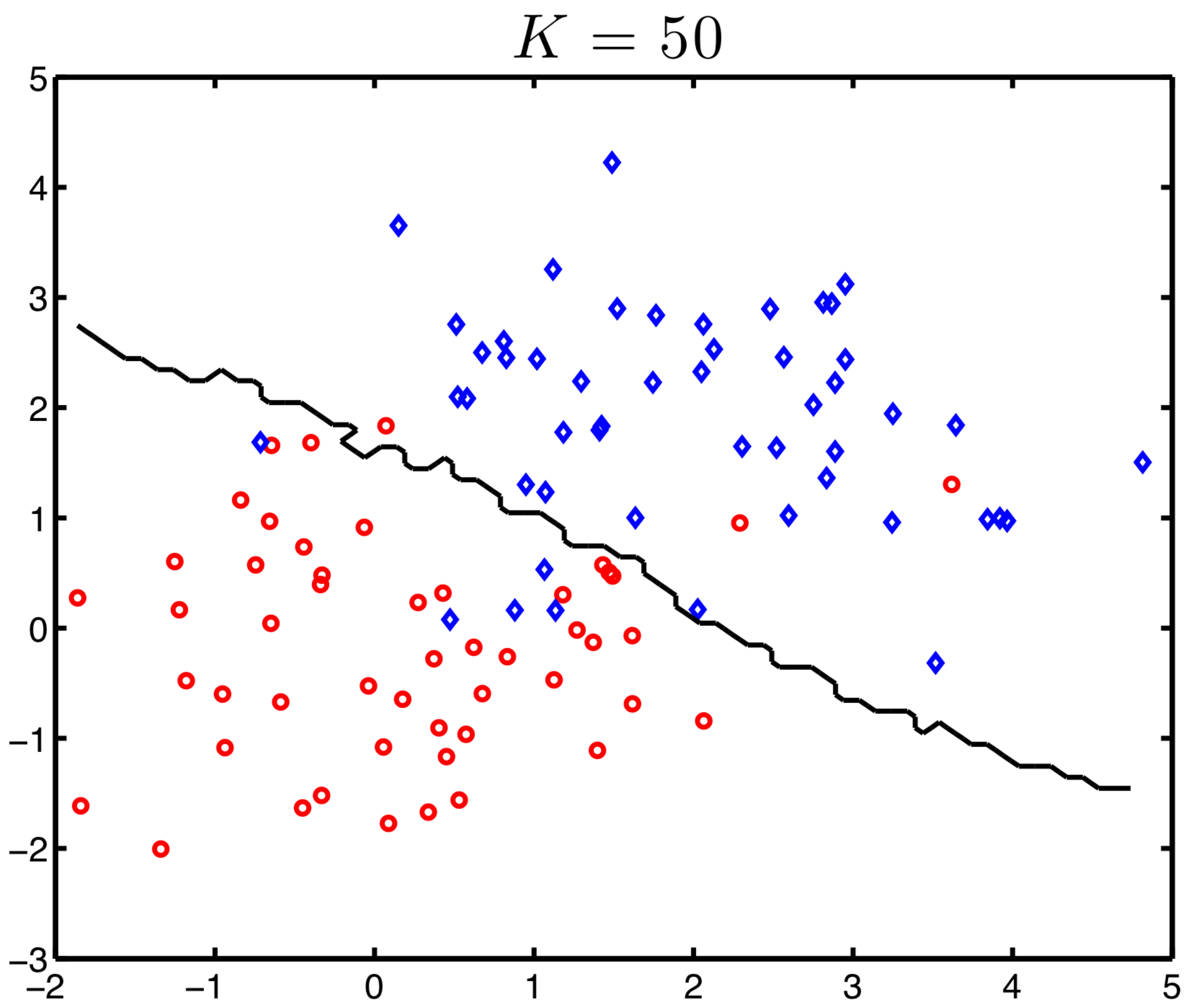
K-Nearest Neighbour



K-Nearest Neighbour



K-Nearest Neighbour



K-Nearest Neighbour

- 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!

K-Nearest Neighbour

- **What is the training process?** Are we learning any parameters?
    - 'Training' in K-NN = 'Memorizing' training data

- **How do we compute the distance between samples?**
    - Any distance metric should work. Squared $L2$ norm is common for real-valued features.
    - $\|\mathbf{x}_1 - \mathbf{x}_{new}\|_2^2 = \sum_i \left(\mathbf{x}_1(i) - \mathbf{x}_{new}(i)\right)^2$
    - Choice of distance metric may change results (but not much).

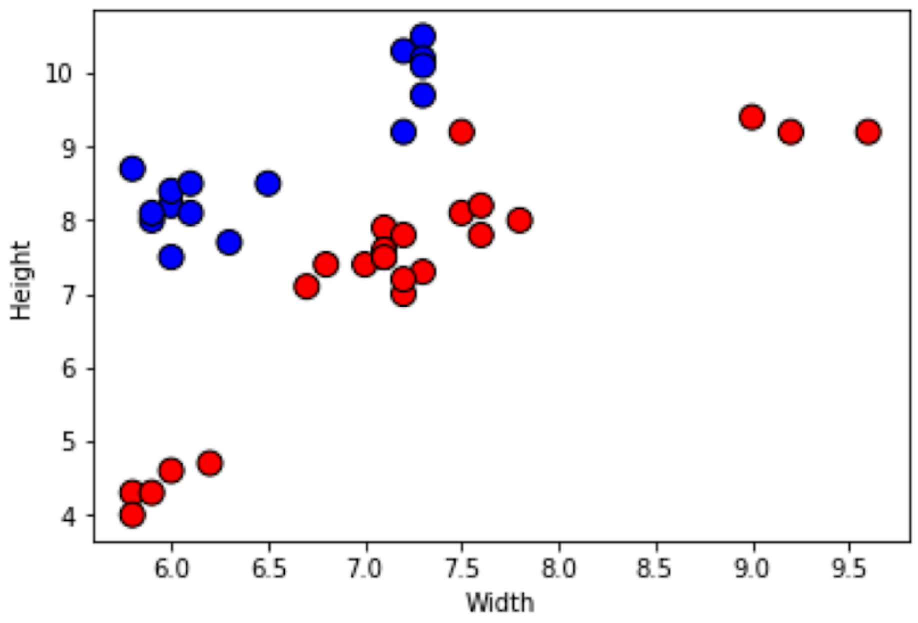
K-NN (orange & lemon example)

In [2]:
```
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')
```

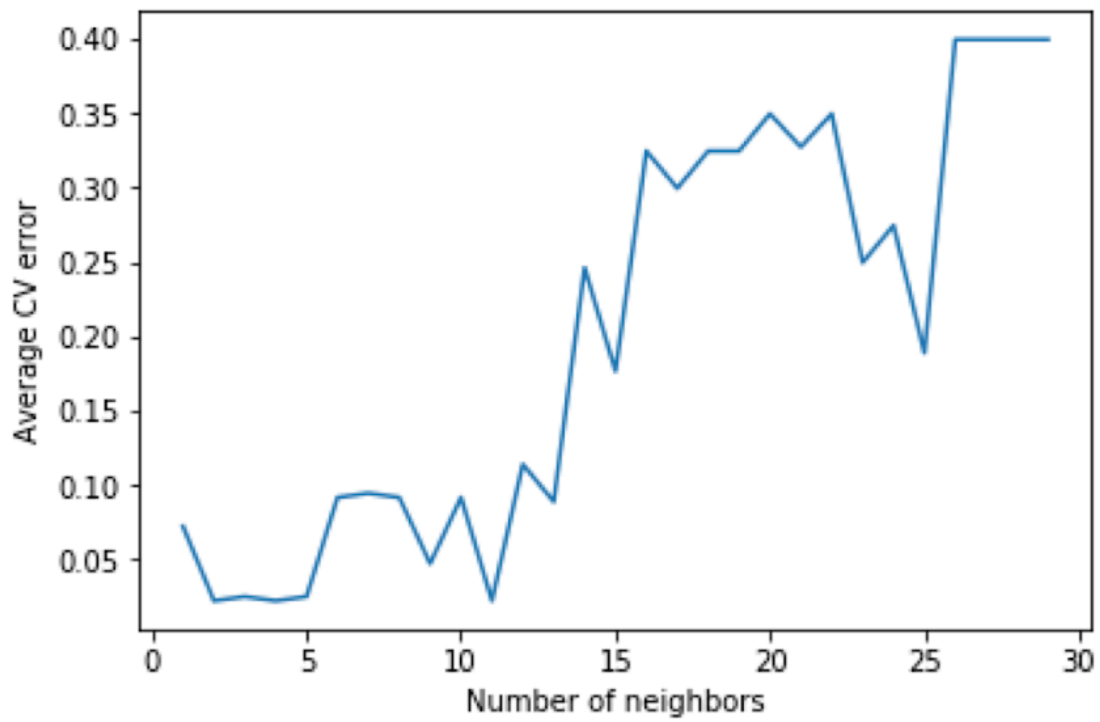


K-NN (orange & lemon example)

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))
```

```
0.022222222222222143
```

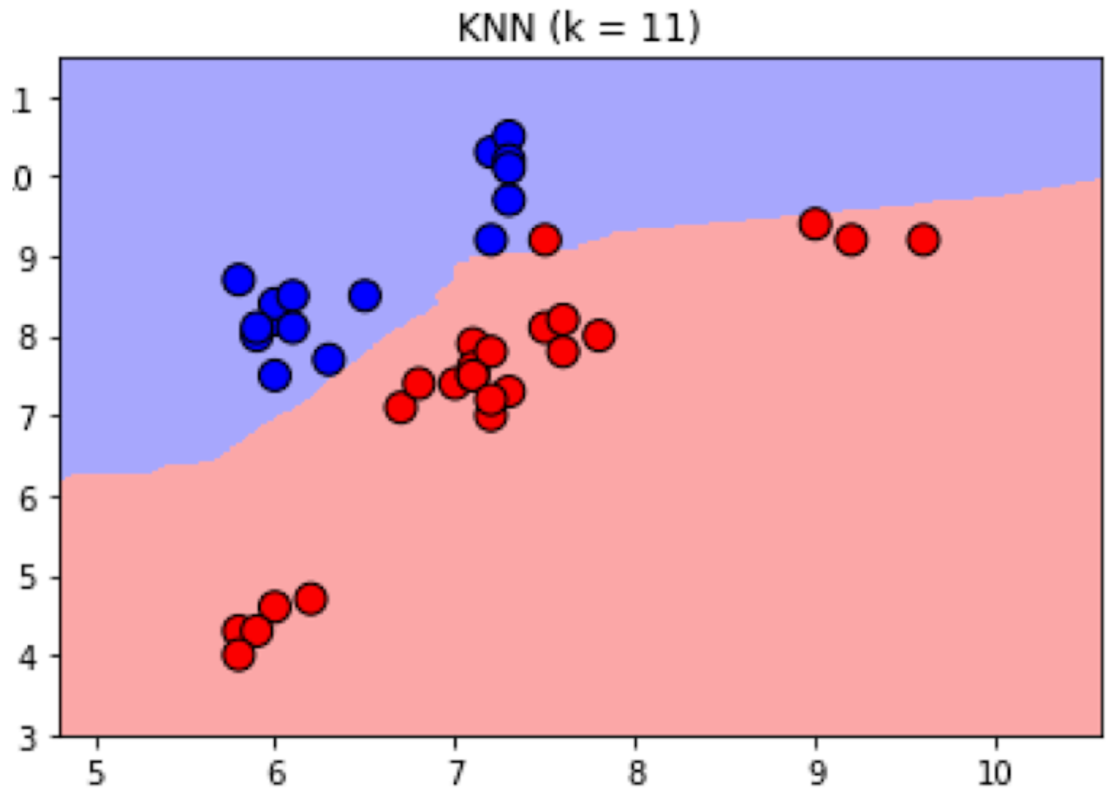


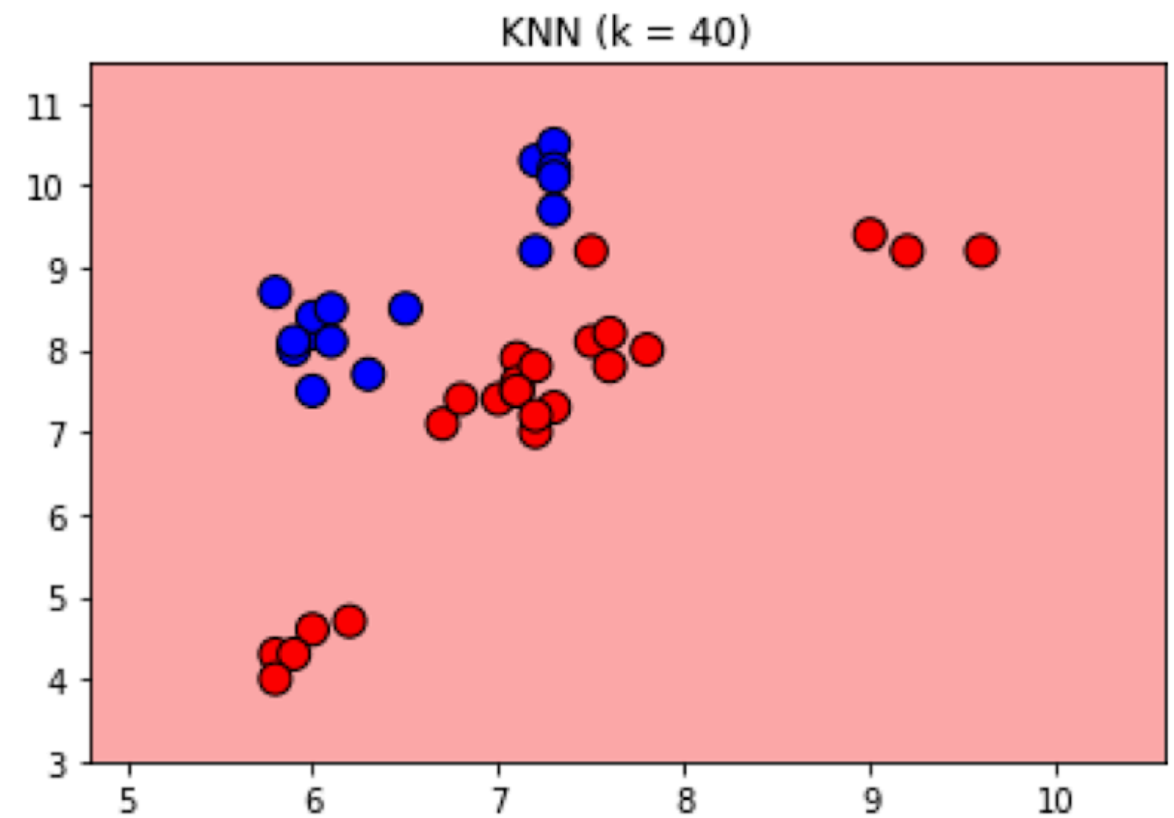
5 fold cross validation to choose K

Error = 1 - accuracy

$$\text{Accuracy} = \frac{\text{correct classficiation}}{\text{total test sampels}}$$

K-NN (orange & lemon example)





K-NN summary

Simple

- Only 1 parameter to tune
- simple to implement
- Fast training (rather no training)

...but **inefficient**

- Inference time may be large if N is large - **Not ideal**
- Large memory requirement