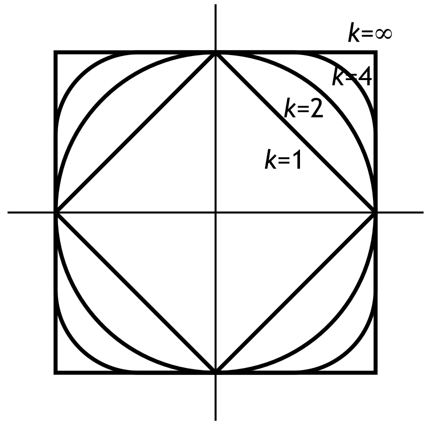
Distance-based methods

- ***L_k* norm** or *Minkowski metric* for *d*-dimensional vectors

$$L_k(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} \left| x_i - y_i \right|^k \right)^{1/k}$$

  - *k*=1: Manhattan distance $L_1(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} \left| x_i - y_i \right|$
  - *k*=2: Euclidean distance $L_2(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{d} \left(x_i - y_i \right)^2} = \sqrt{(\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y})} = \left\| \mathbf{x}-\mathbf{y} \right\|$
  - *k*=∞: $L_\infty(\mathbf{x},\mathbf{y}) = \max_i \left| x_i - y_i \right|$



Plot of all points at distance 1 from the origin for different values of *k* (*d*=2)

Distances and (in)variance

- Euclidean distance is invariant under translation and rotation, but sensitive to scaling
  - multiplying a dimension with a factor $f>1$ ($g<1$) will increase (decrease) its influence in the distance calculation
- Therefore we usually re-weight all dimensions to have unit variance

$$\sqrt{\sum_{i=1}^{d} \frac{\left(x_i - y_i\right)^2}{\sigma_i^2}} = \sqrt{(\mathbf{x}-\mathbf{y})^{\mathrm{T}}\mathbf{W}(\mathbf{x}-\mathbf{y})} \text{ with } \mathbf{W} = \begin{bmatrix} 1/\sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/\sigma_d^2 \end{bmatrix}$$

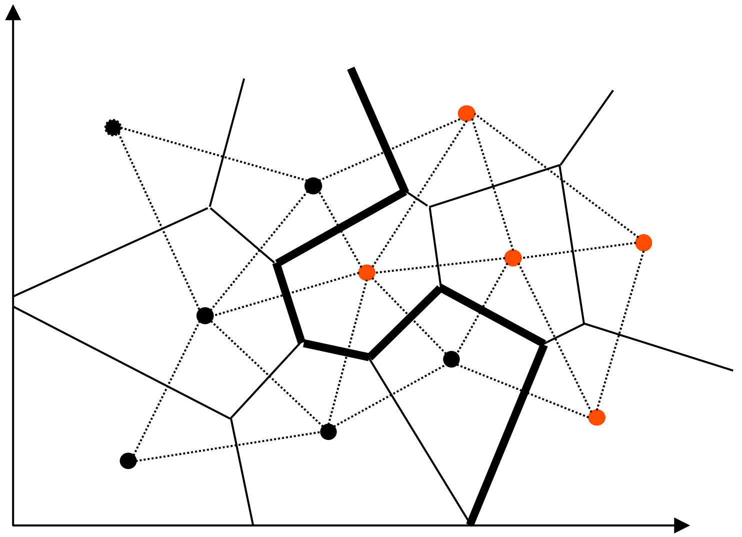
  - unit-distance circle becomes axis-parallel ellipse
- If we also want to eliminate correlations between features, we set **W** to the inverse of the covariance matrix (***Mahalanobis distance***)
  - unit-distance ellipse is rotated

Symbolic distance

- For equal-length bit strings, the ***Hamming distance*** is the number of positions in which the two strings differ
  - 101100 and 011101 have Hamming distance 3
  - corresponds to the L_1 distance
    - can also do L_k, just a monotonic transformation in this case
- Can be generalised in various ways
  - symbolic unordered features: 0 if same, 1 if different
  - symbolic ordered features: represent by (unit-variance) numbers
  - strings of different lengths (e.g. words): edit distance
- In what follows we assume some distance metric on feature vectors with possibly mixed numeric and symbolic features

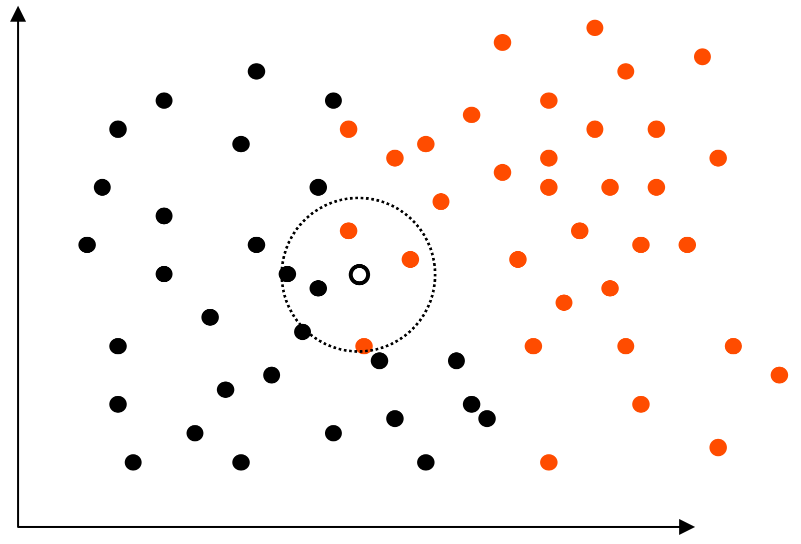
A close neighbour…

- ***Nearest-neighbour classification***: store a set of exemplars (e.g. the training data), and assign a test instance to the class of its nearest exemplar
  - instance space divided by ***Voronoi tesselation***
  - piecewise linear decision boundary



Increasing the number of neighbours

- ***k-Nearest-neighbour classification***: assign a test instance to the majority class among its *k* nearest neighbours
  - smoother decision boundary than with *k*=1
  - *k*→∞: all instances are assigned most common class in training set (i.e., using prior distribution only)
  - *k* can be tuned using separate test set



To classify a test instance, draw the smallest hypersphere around it that contains *k* neighbours; then assign majority class among those

Example: 2-nearest-neighbour

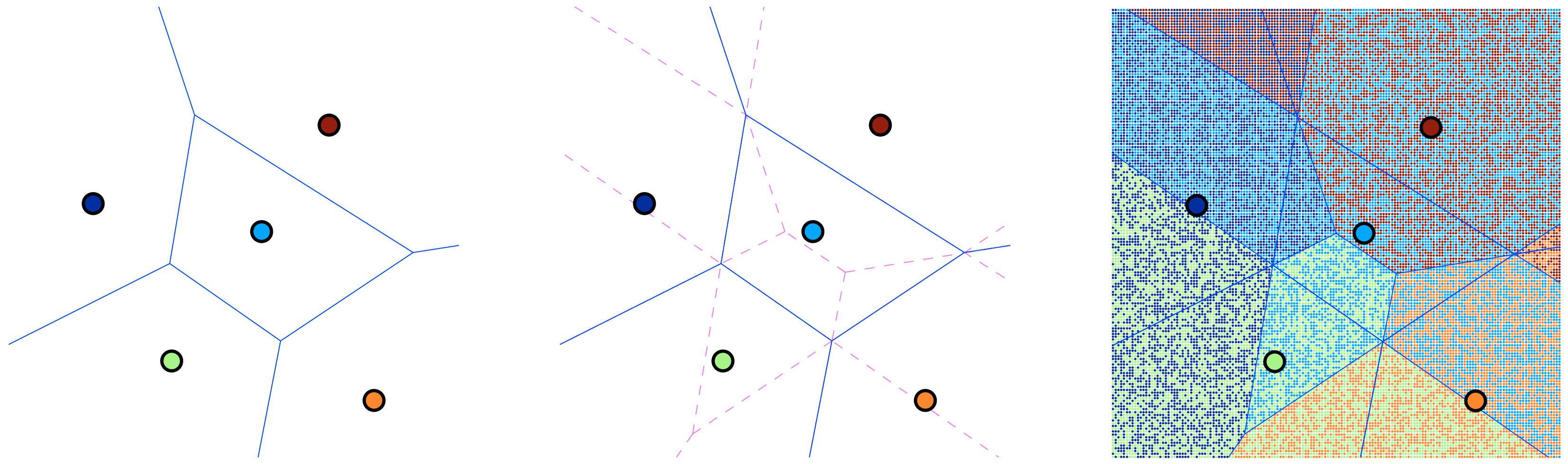


Figure 8.8. **(left)** Voronoi tesselation for five exemplars. **(middle)** Taking the two nearest exemplars into account leads to a further subdivision of each Voronoi cell. **(right)** The shading indicates which exemplars contribute to which cell.

k-nearest neighbour: variations

- In *k*-NN, each of the *k* nearest neighbours $\mathbf{x}_i$ contributes equally to the vote for the most likely class of the test instance $\mathbf{x}$
- ***Distance-weighted k-NN***: weight the vote with a ***kernel*** $K(\mathbf{x}-\mathbf{x}_i)$ that decreases with the distance
  - e.g. Gaussian kernel $K(\mathbf{d}) \propto e^{-\|\mathbf{d}\|^2}$
- Using distance-weighting, we can also let $k \to \infty$ and have **all** training instances contribute to the vote
  - **global** rather than local method
- All these methods can also be used for ***estimation***, where the dependent variable is a scalar rather than a class

Discussion

- *k*-nearest neighbour methods take **all** features into account. With large numbers of features this causes problems
  - irrelevant features dominate distance calculations
  - training set covers only a fraction of instance space, which causes overfitting: 'curse of dimensionality'

- This can be addressed in several ways
  - stretch/shrink dimensions if it improves performance on a separate test set
  - perform **feature selection** in a pre-processing step —> later lecture

Clustering

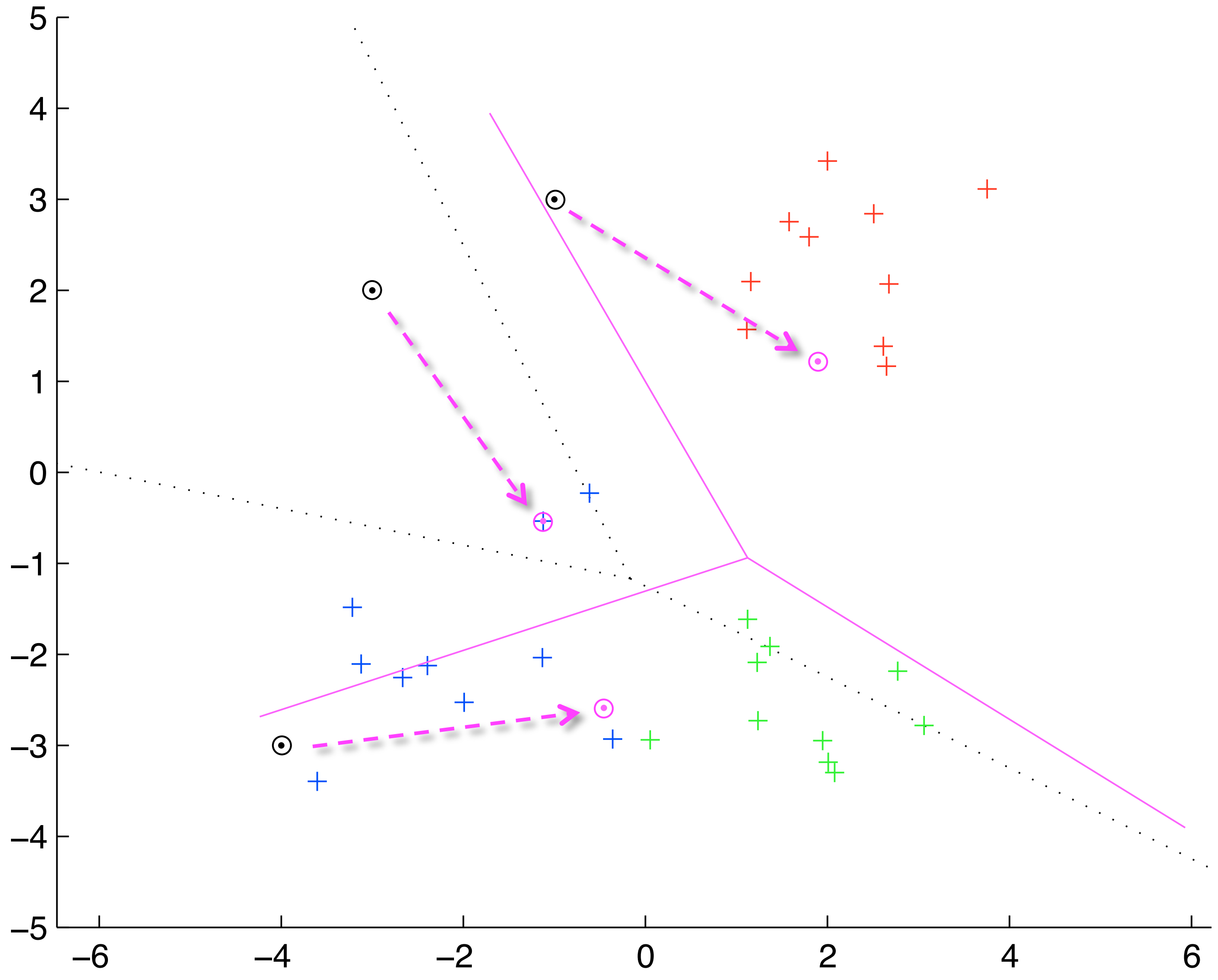
- **Clustering** involves segmenting the instance space into regions of similar objects
    - no guidance through labelled training instances -> ***unsupervised learning***
- We can again use the idea of a distance metric -> ***K*-means clustering** (this lecture)
- We can also use the class as a ‘hidden’ variable -> **Gaussian mixture models** (later)

K-means clustering

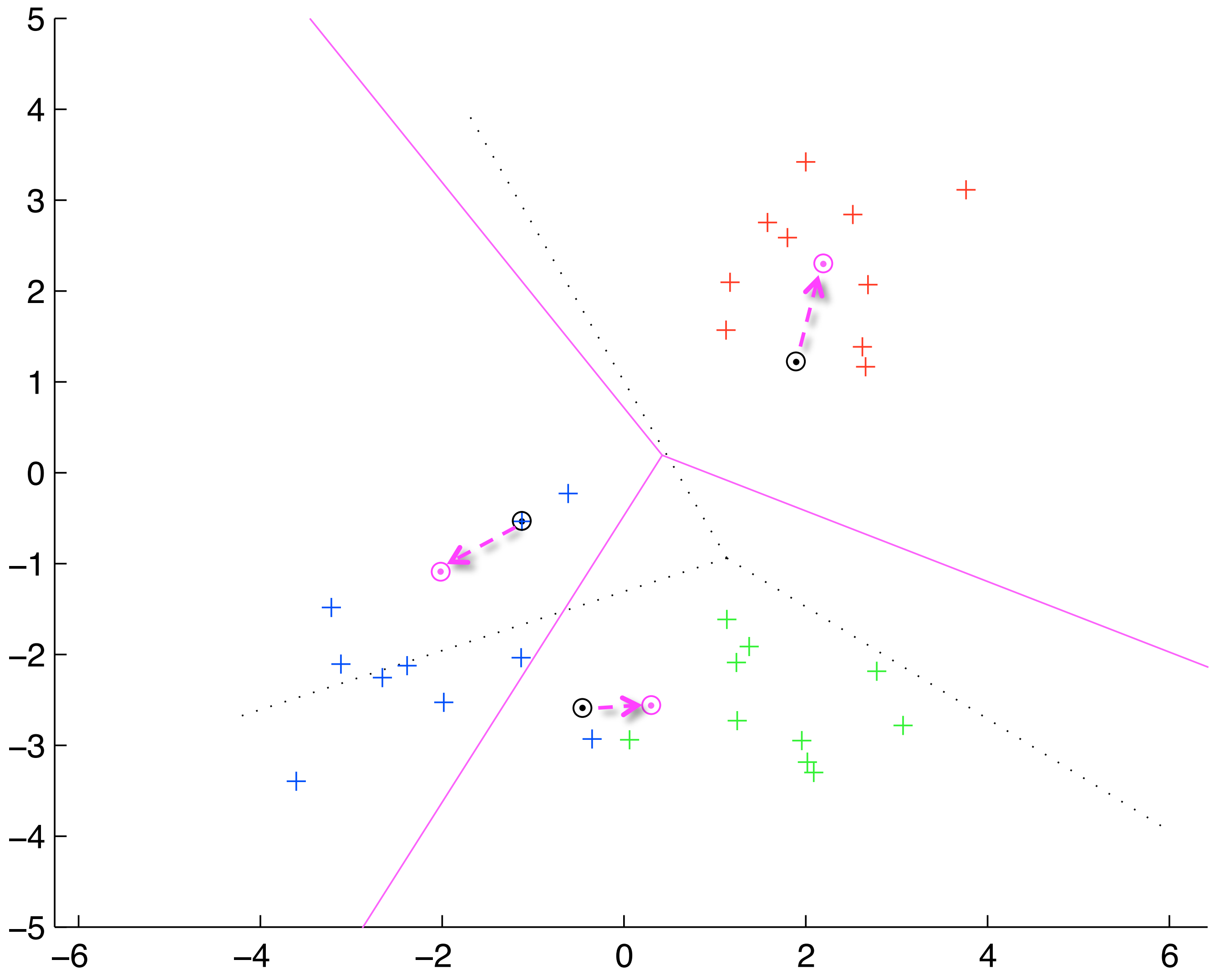
```
function KMeans(Instances,K)
randomly initialise K vectors $\mu_1 \ldots \mu_K$;
repeat
        assign each $x \in Instances$ to the nearest $\mu_j$;
        recompute each $\mu_j$ as the mean of the
                instances assigned to it;
until no change in $\mu_1 \ldots \mu_K$;
return $\mu_1 \ldots \mu_K$;
```

- Notes:
  - different initialisation schemes are possible for the centroids
  - alternatively, we can partition the instances into *K* groups and calculate the initial means from those

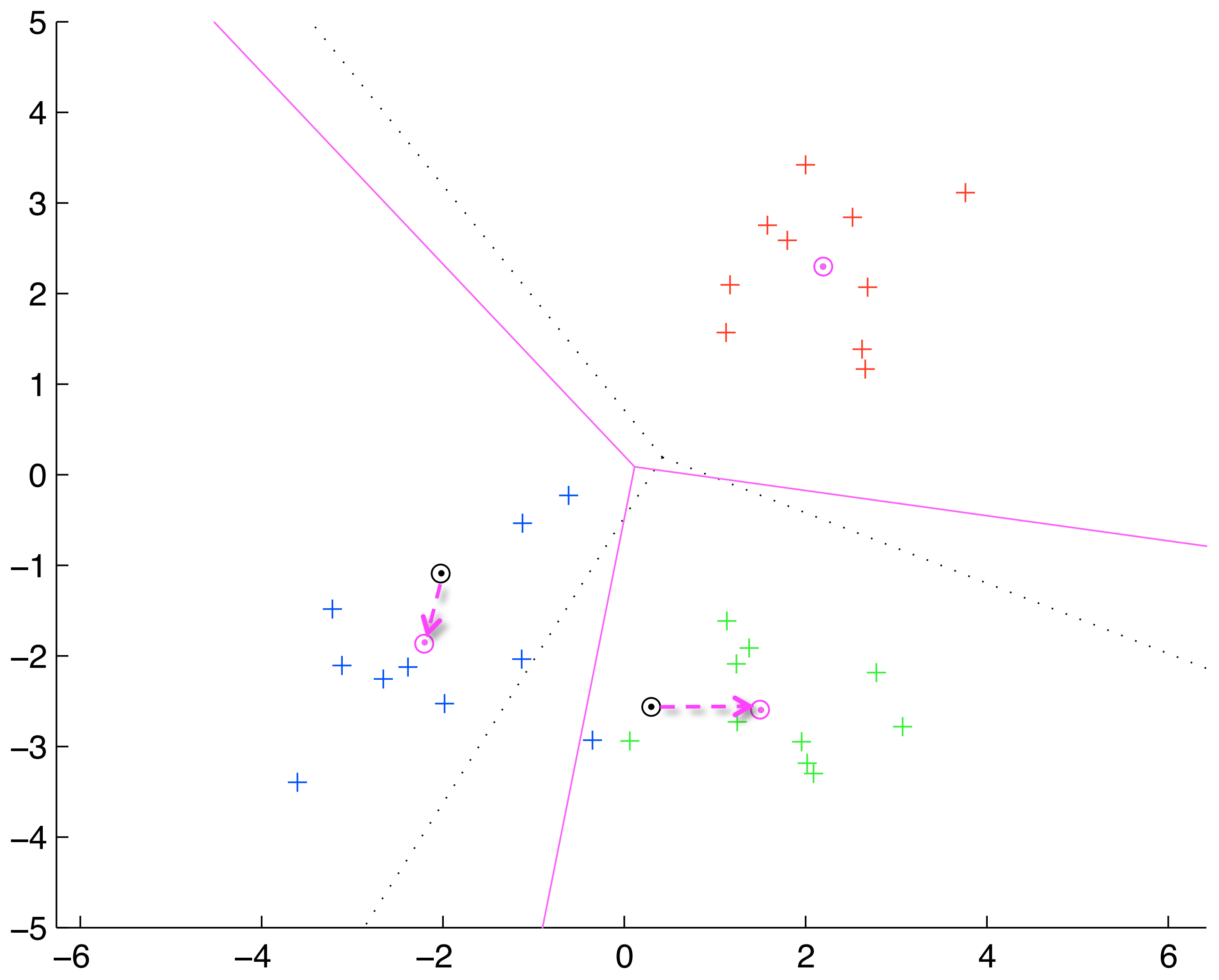
K-means example



K-means example



K-means example

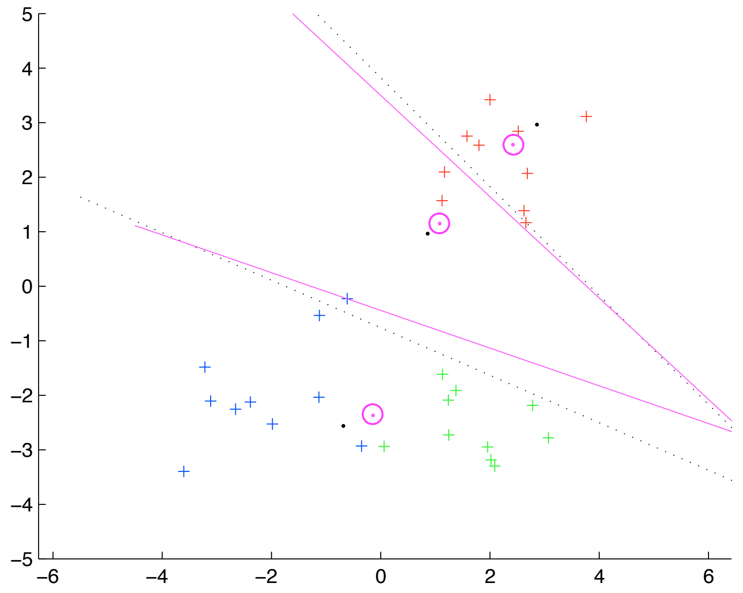


Analysis

- What does it do?
  - ***within-cluster scatter***: total squared distance between points $\mathbf{x}_i$ and centroid $\boldsymbol{\mu}_k$ in k-th cluster: $\sum_i \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$
  - K-means attempts to find a configuration $\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_K$ that minimises within-cluster scatter summed over all clusters
    - this is equivalent to maximising the between-cluster scatter (move all data points to their centroid and calculate total squared distance to the global centroid)

- Does it work?
  1. The algorithm terminates.
  2. It finds a **local optimum** from which no further improvement is possible by making local changes.
  3. It does not necessarily find a **global optimum**.

Local optimum with *K*-means



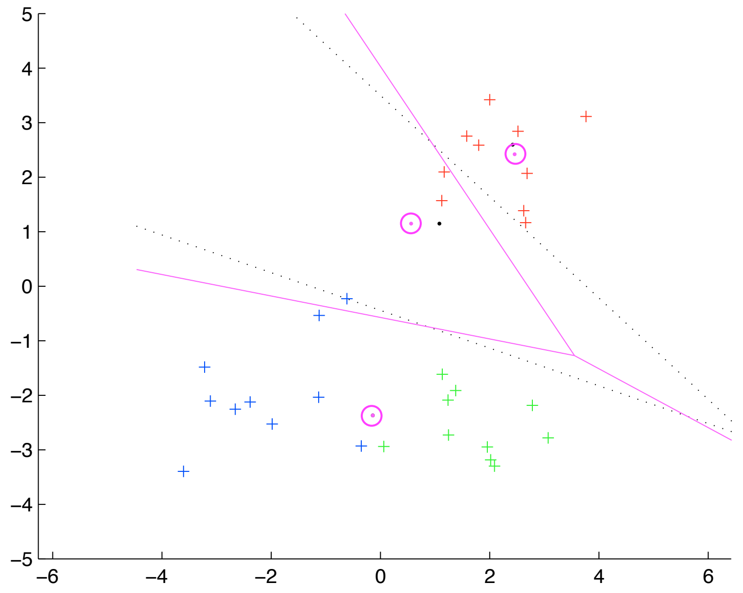


Figure 8.12. **(left)** First iteration of 3-means on the same data as Figure 8.11 with differently initialised centroids. **(right)** 3-means has converged to a sub-optimal clustering.

Discussion

- To be reasonably sure that we're not too far from the global optimum, we should run *K*-means a number of times with different initial configurations
  - or use more clever and/or domain-dependent initialisation schemes
- *K*-medoids is a variant of *K*-means in which cluster center is a **data point** (rather than the mean of data points) that minimises within-cluster scatter
  - allows to use distance metrics other than Euclidean distance, which tends to be sensitive to outliers
- We can also use "soft" assignment of data points to cluster centers –> Gaussian mixture models