K nearest neighbours

Kairit Sirts

14.02.2014

K nearest neighbour (KNN) classification

- Suppose we have a set of N points belonging to C different classes.
- For each class we have N_c points such that

$$\sum_c N_c = N$$

- Suppose we want to classify a new point **x**.
- From the training set we find K **nearest** points to the point **x**.
- We know the correct labels of these points.
- We assign **x** the **majority label** of the neighbouring points.

KNN classification with K = 3

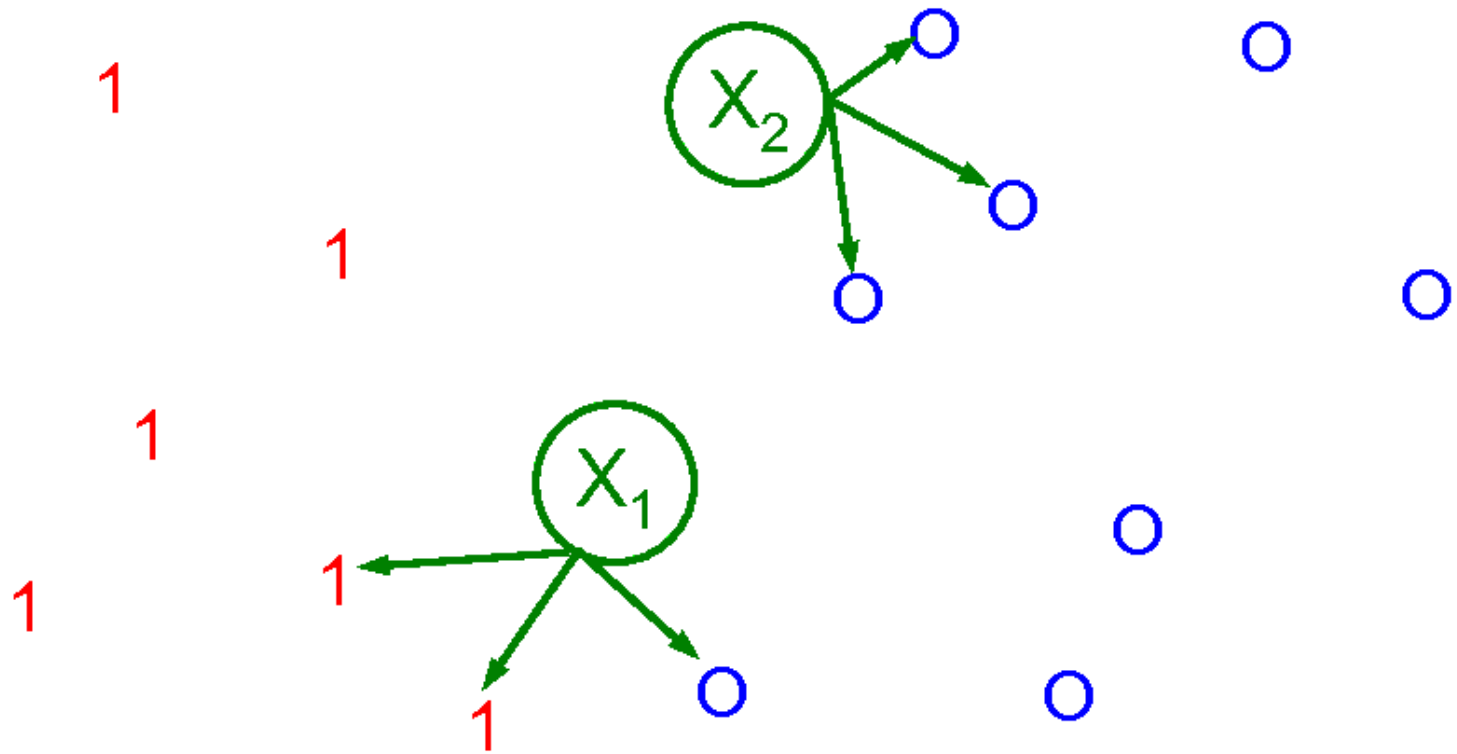

Figure 1.14(a) from Machine Learning: A Probabilistic Approach (Murphy).

Formally

- When we have training data D and some fixed K then the value of the label y of a new point $\mathbf{x}$ has the probability:

$$p(y = c|\mathbf{x}, D, K) = \frac{1}{K} \sum_{i \in N_K(\mathbf{x}, D)} \mathbf{1}\{y_i = c\}$$

- $N_K(\mathbf{x}, D)$ are the indices of the K nearest points to $\mathbf{x}$ in D.

KNN classifier

- KNN is a **supervised** method.
- It is a **nonparametric** learning method.
  - In nonparametric models the number of parameters is not fixed but rather grows with the amount of data.
- KNN is also called a **memory-based** learner because essentially the algorithm just memorizes the training data.
- K is hyperparameter in this model.

KNN example

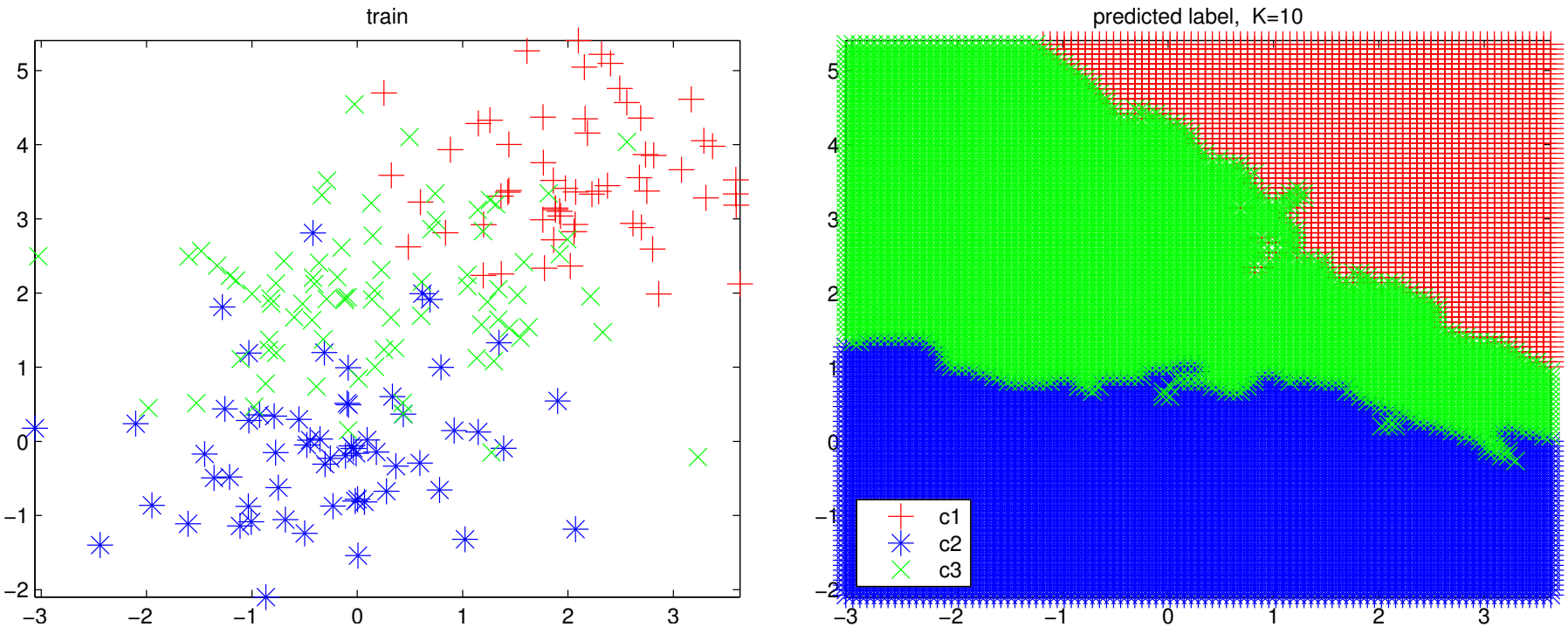

Figure 1.15(a) and (d) from Machine Learning: A Probabilistic Approach (Murphy).

Decision boundary

- The lines separating different colors on last slide are called **decision boundaries**.
- They show for this specific model into which class the test examples will be classified.
- Decision boundaries characterize the **complexity** of the model:
  - When the decision boundary is too complex we might be overfitting.
  - When the decision boundary is too smooth there might be underfitting.
- With KNN we can use K to control the complexity of the decision boundary.
- Cross-validation can be used to select the best value for K.

What means **nearest**?

- The distance between two points must be assessed according to some **distance measure**.
- There are many ways for computing the distance between two points.
  - Euclidean distance (2-norm)
  - Cosine distance
  - Hamming distance
  - Levenshtein distance
  - Manhattan distance (1-norm)
  - Chebyshev distance (∞-norm)
  - Mahalanobis distance
- Most distances are **metrics** but this must not always be true.

Distance metric

A distance measure d is a **metric** if the following conditions are satisfied:

1. non-negativity: $d(x, y) \geq 0$
2. identity: $d(x, y) = 0$ if and only if $x = y$
3. symmetry: $d(x, y) = d(y, x)$
4. triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Distances in Euclidean space

Euclidean distance (2-norm)

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Manhattan distance (1-norm)

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

Chebyshev distance (∞-norm)

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

Cosine similarity and distance

- Cosine similarity measures the angle between vectors.
- Most commonly used in high-dimensional positive spaces.

$$sim_C(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

- It ranges from values -1 (exactly opposite) to 1 (exactly same).
- Cosine distance is obtained from cosine similarity:

$$d_C(\mathbf{x}, \mathbf{y}) = 1 - sim_C(\mathbf{x}, \mathbf{y})$$

Levenshtein distance

- Also called string edit distance (SED).
- Used to measure the distance between two strings.
- SED is minimum number of single-character edits required to change one string into another.
- Edit operations include:
  - insertions
  - deletions
  - substitutions
- Can be implemented using dynamic programming.
- SED(kitten, sitting) = 3: **k** i t t **e** n $\rightarrow$ **s** i t t **i** n **g**
  - substitute **k** with **s**
  - substitute **e** with **i**
  - insert **g**

Hamming distance

- Like Levenshtein distance with substitution operation only.
- Frequently used with binary data.

1		**1**		0		1		**1**		
1		**0**		0		1		**0**		
0	+	**1**	+	0	+	0	+	**1**	=	**2**

- Can also be used with categorical data.

red		**tall**		5		
red		**short**		5		
0	+	**1**	+	0	=	**1**

Data normalization

- Distance depends on the scale of the features.
- Changing the scale of one feature (for example from cm to mm) while keeping other the same the distances and nearest neighbours will be different.
- A common way to avoid it is to **normalize** the data:
  - Compute mean mu_j and standard deviation σ_j in each of the j dimensions.
  - Rescale all elements such that:

$$x_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

Mahalanobis distance

- Mahalanobis distance takes into account the covariance between dimensions.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})}$$

- Σ is the covariance matrix of features.
- This formula appears in the exponent of the Gaussian distribution.
- When covariance is idendity matrix then Mahalanobis distance reduces to Euclidean distance.
- With diagonal covariance matrix the resulting measure is normalized Euclidean distance.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^{m} \frac{(x_j - y_j)^2}{\sigma_j^2}}$$

Curse of dimensionality

- KNN-s can work well with good distance metric and enough labelled training data.
- But they do not perform well in high dimensional settings due to **curse of dimensionality**.
- Suppose data is distributed uniformly in d-dimensional unit cube.
- We choose a point $\mathbf{x}$ and form a cube around including some fraction f of all points.
- What is the expected edge length of this cube?

$$E_d[s(f)] = f^{1/d}$$

Curse of dimensionality

- Let's fix $f = 0.01$

d	s
1	0.01
2	0.10
3	0.22
4	0.32
5	0.40
6	0.46
7	0.52
8	0.56
9	0.60
10	0.63

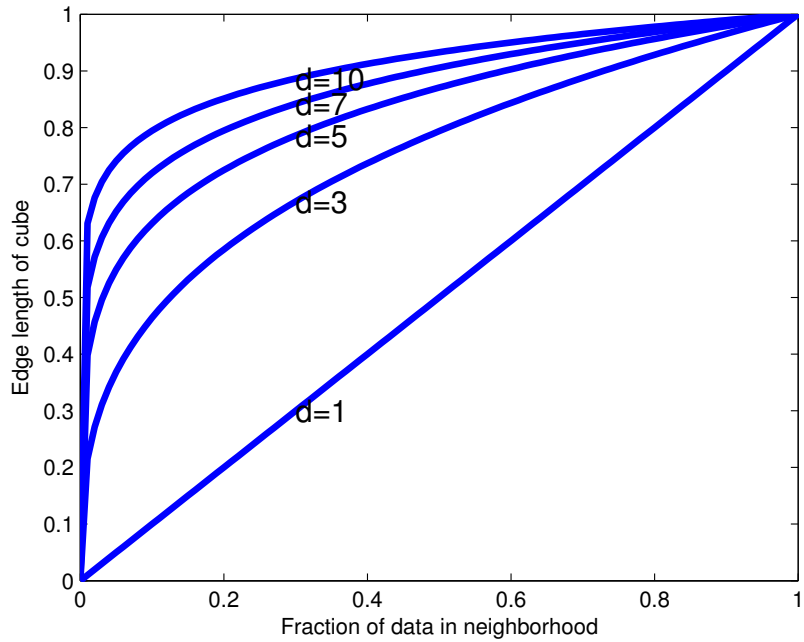

Figure 1.16(b) from Machine Learning: A Probabilistic Approach (Murphy).

- Considering that our cube has edge length only one the "nearest neighbours" for high dimensional data are actually not very near.