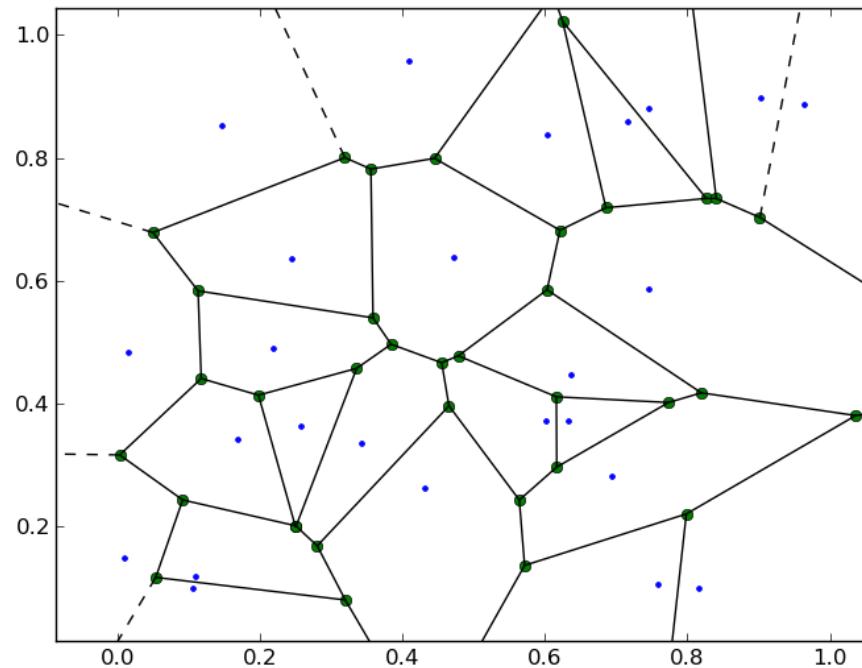


# Nearest Neighbour



Nearest Neighbour is a regression or classification algorithm that predicts whatever is the output value of the nearest data point to some query.

To find the nearest data point, we have to find the *distance* between the query and other points. So we have to decide how to define the *distance*.

# Minkowski distance

**Minkowski distance** If  $\chi \rightarrow \mathbb{R}^d$ ,  $x, y \in \chi$ , the Minkowski distance of order  $p > 0$  is defined as:

$$Dis_p(x, y) = \left( \sum_{j=1}^d |x_j - y_j|^p \right)^{1/p} = \|x - y\|_p$$

Where  $\|z\|_p = (\sum_{j=1}^d |z_j|^p)^{1/p}$  is the  $p$  – norm (sometimes denoted  $L_p$  norm) of the vector  $z$ .

# Minkowski distance

- The **2-norm** refers to the familiar **Euclidean distance**

$$Dis_2(x, y) = \sqrt{\sum_{j=1}^n (x_j - y_j)^2} = \sqrt{(x - y)^T (x - y)}$$

- The **1-norm** denotes **Manhattan distance**, also called **cityblock distance**:

$$Dis_1(x, y) = \sum_{j=1}^n |x_j - y_j|$$

# Minkowski distance

- If we now let  $p$  grow larger, the distance will be more and more dominated by the largest coordinate-wise distance, from which we can infer that  $Dis_{\infty} = \max_j |x_j - y_j|$ ; this is also called **Chebyshev distance**.
- You will sometimes see references to the **0-norm** (or  $L_0$  norm) which **counts the number of non-zero elements in a vector**. The corresponding distance then counts the number of positions in which vectors  $x$  and  $y$  differ. This is not strictly a Minkowski distance; however, we can define it as:

$$Dis_0(x, y) = \sum_{j=1}^d (x_j - y_j)^0 = \sum_{j=1}^d I[x_j \neq y_j]$$

under the understanding that  $x^0 = 0$  for  $x = 0$  and 1 otherwise.

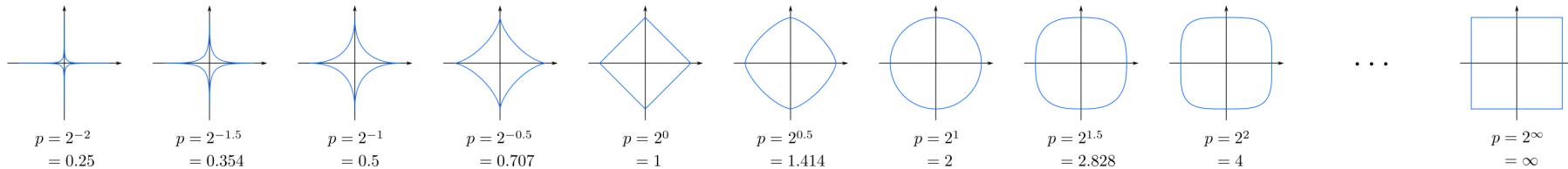
# Minkowski distance

Sometimes the data is not naturally in  $\mathbb{R}^d$ , but if we can turn it into Boolean features, or character sequences, we can still apply distance measures. For example:

- If  $x$  and  $y$  are binary strings, this is also called the *Hamming distance*. Alternatively, we can see the Hamming distance as the number of bits that need to be flipped to change  $x$  into  $y$ .  
    敲击
- For non-binary strings of unequal length this can be generalised to the notion of *edit distance* or *Levenshtein distance*.

# Circles and ellipses

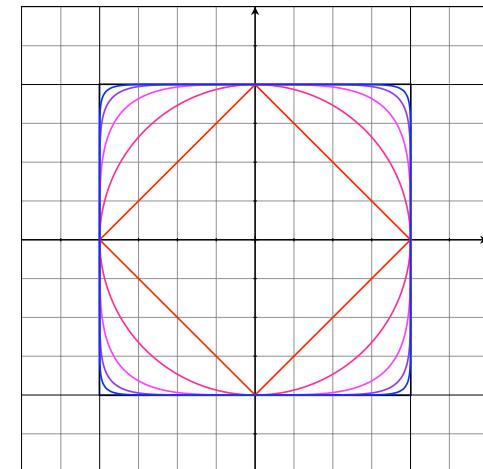
距离原点距离相同的点的集合(with different value of p)



混合

Unite circles with different order-p Minkowski distance

- Notice that for points on the coordinate axes all distances agree
- If we require a rotation invariant distance metric, then Euclidean distance is our only choice



# Distance metric

Distance metric Given an instance space  $\mathcal{X}$ , a distance metric is a function  $Dis : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^d$  such that for any  $x, y, z \in \mathcal{X}$ :

- distances between a point and itself are zero:  $Dis(x, x) = 0$
- all other distances are larger than zero: if  $x \neq y$  then  $Dis(x, y) > 0$
- distances are symmetric:  $Dis(y, x) = Dis(x, y)$
- **detours** can not shorten the distance (triangle inequality):  
**绕路**

$$Dis(x, z) \leq Dis(x, y) + Dis(y, z)$$

If the second condition is weakened to a non-strict inequality – i.e.,  $Dis(x, y)$  may be zero even if  $x \neq y$  – the function  $Dis$  is called a *pseudo-metric*.

# Means and distances

The arithmetic mean minimises squared Euclidean distance. The arithmetic mean  $\mu$  of a set of data points  $D$  in a Euclidean space is the unique point that minimises the sum of squared Euclidean distances to those data points.

**Proof.** We will show that  $\arg \min_y \sum_{x \in D} \|x - y\|^2 = \mu$ , where

$\|\cdot\|$  denotes the 2-norm. We find this minimum by taking the gradient (the vector of partial derivatives with respect to  $y$ ) of the sum and setting it to the zero vector:

$$\nabla_y \sum_{x \in D} \|x - y\|^2 = -2 \sum_{x \in D} (x - y) = -2 \sum_{x \in D} x + 2|D|y = 0$$

From which we derive  $y = \frac{1}{|D|} \sum_{x \in D} x = \mu$

# Means and distances

- Notice that minimising the *sum* of squared Euclidean distances of a given set of points is the same as minimising the *average* squared Euclidean distance.
- You may wonder what happens if we drop the square here: wouldn't it be more natural to take the point that **minimises total Euclidean distance** as exemplar?  
单变量的
- This point is known as the **geometric median**, as for univariate data it corresponds to the **median** or 'middle value' of a set of numbers. However, for multivariate data there is no closed-form expression for the geometric median, which needs to be calculated by successive approximation.

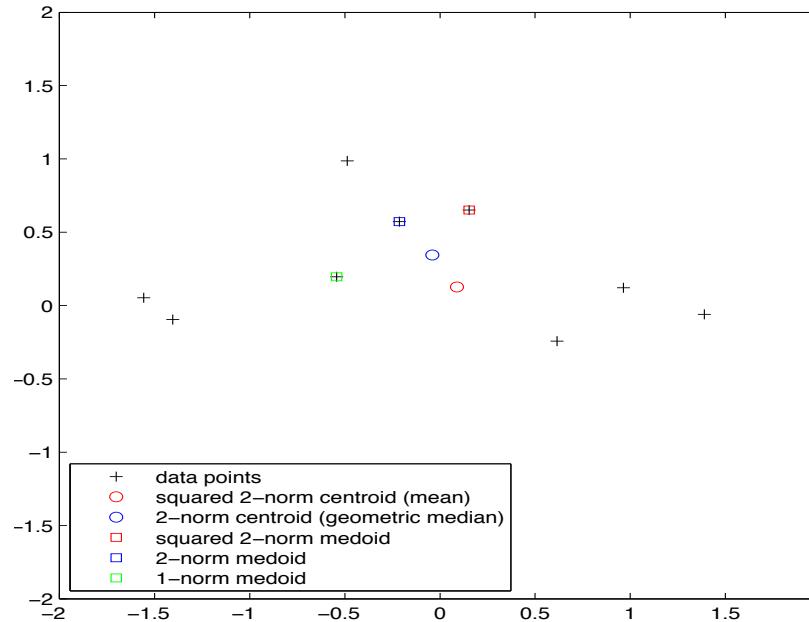
# Means and distances

centroid: calculated by algrithmn, does not have to occur in the data

medoid: alternative of centroid, one of the point chosen closest to centroid 模范点、标准

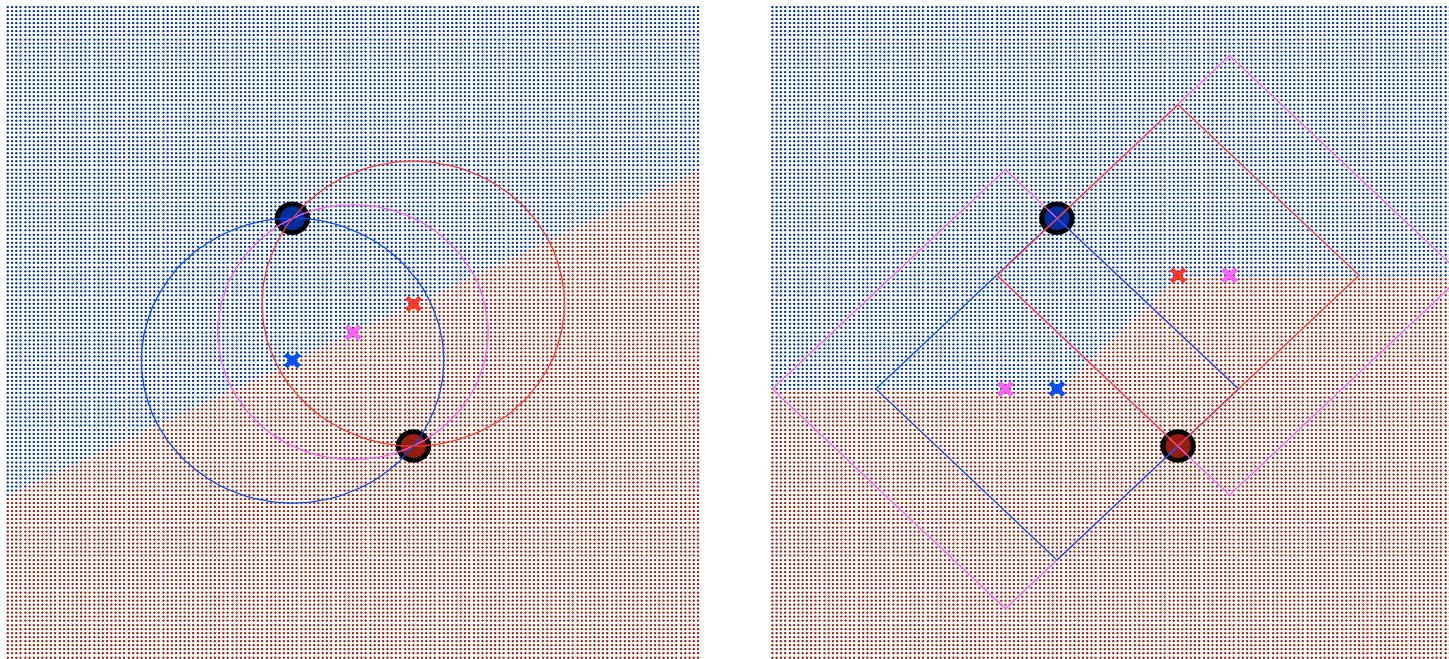
- In certain situations it makes sense to restrict an exemplar to be one of the given data points. In that case, we speak of a *medoid*, to distinguish it from a *centroid* which is an exemplar that doesn't have to occur in the data.
- Finding a *medoid* requires us to calculate, for each data point, the total distance to all other data points, in order to choose the point that minimises it. Regardless of the distance metric used, this is an  $O(n^2)$  operation for  $n$  points.
- So for medoids there is no computational reason to prefer one distance metric over another.
- There may be more than one medoid.

# Centroids and medoids



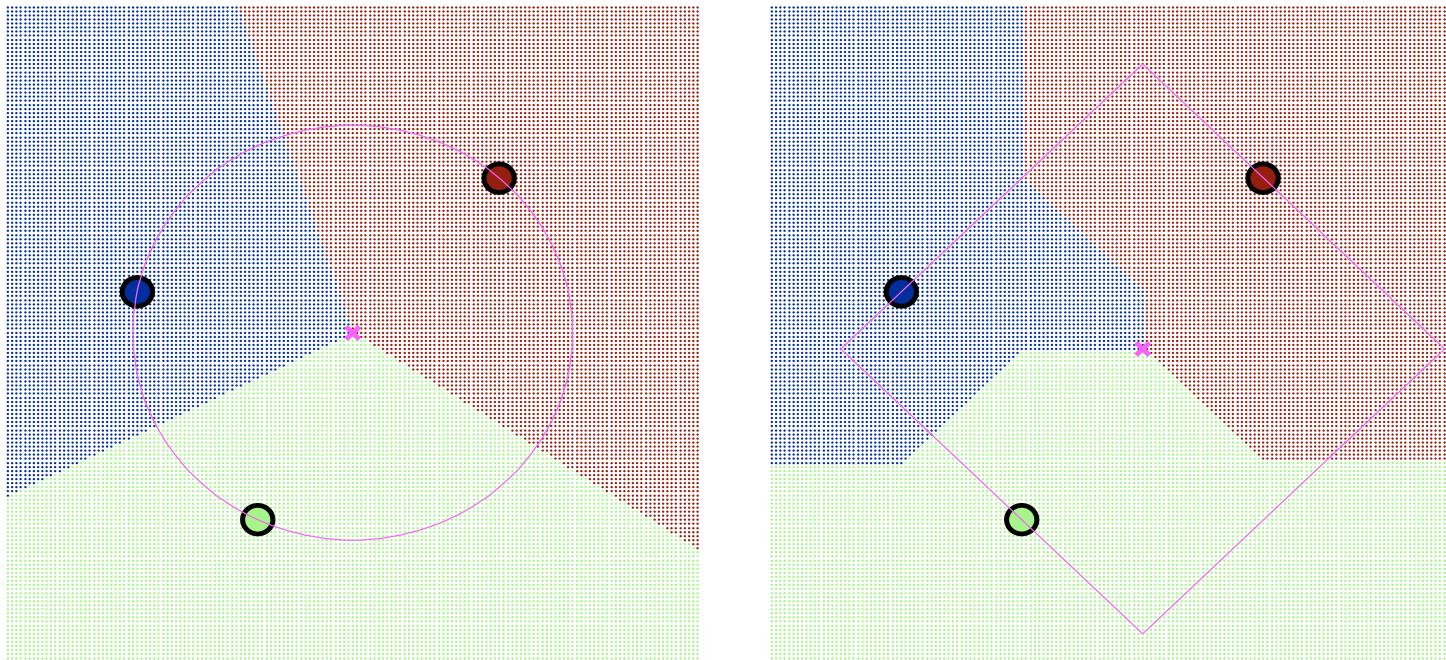
A small data set of 10 points, with circles indicating centroids and squares indicating medoids (the latter must be data points), for different distance metrics. Notice how the outlier on the bottom-right ‘pulls’ the mean away from the geometric median; as a result the corresponding medoid changes as well.

# Two-exemplar decision boundaries



(left) For two exemplars the nearest-exemplar decision rule with Euclidean distance results in a linear decision boundary coinciding with the perpendicular bisector of the line connecting the two exemplars.  
(right) Using Manhattan distance the circles are replaced by diamonds.

# Three-exemplar decision boundaries



(left) Decision regions defined by the 2-norm nearest-exemplar decision rule for three exemplars. (right) With Manhattan distance the decision regions become non-convex.

# Distance-based models

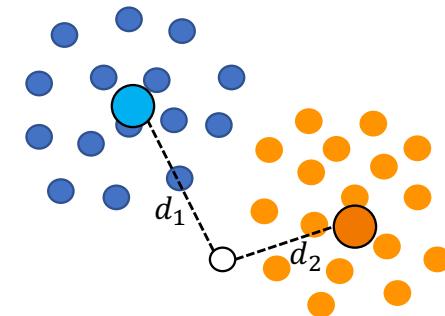
To summarise, the main ingredients of distance-based models are:

- **distance metrics**, which can be Euclidean, Manhattan, Minkowski or Mahalanobis, among many others;
- **exemplars**: centroids that find a centre of mass according to a chosen distance metric, or medoids that find the most centrally located data point; and
- **distance-based decision rules**, which take a vote among the  $k$  nearest exemplars.

# Nearest Centroid Classifier

# Nearest Centroid Classifier

This is a classifier based on minimum distance principle, where the class exemplars are just the centroids (or means)



Training: for training sample pairs  $\{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$  where  $x_i$  is the feature vector for sample  $i$  and  $y_i$  is the class label, class centroids are:

$$\mu_k = \frac{1}{|C_k|} \sum_{j \in C_k} x_j$$

Test

- a new unknown object with feature vector  $x$  is classified as class  $i$  if it is much closer to the mean vector of class  $k$  than to any other class mean vector

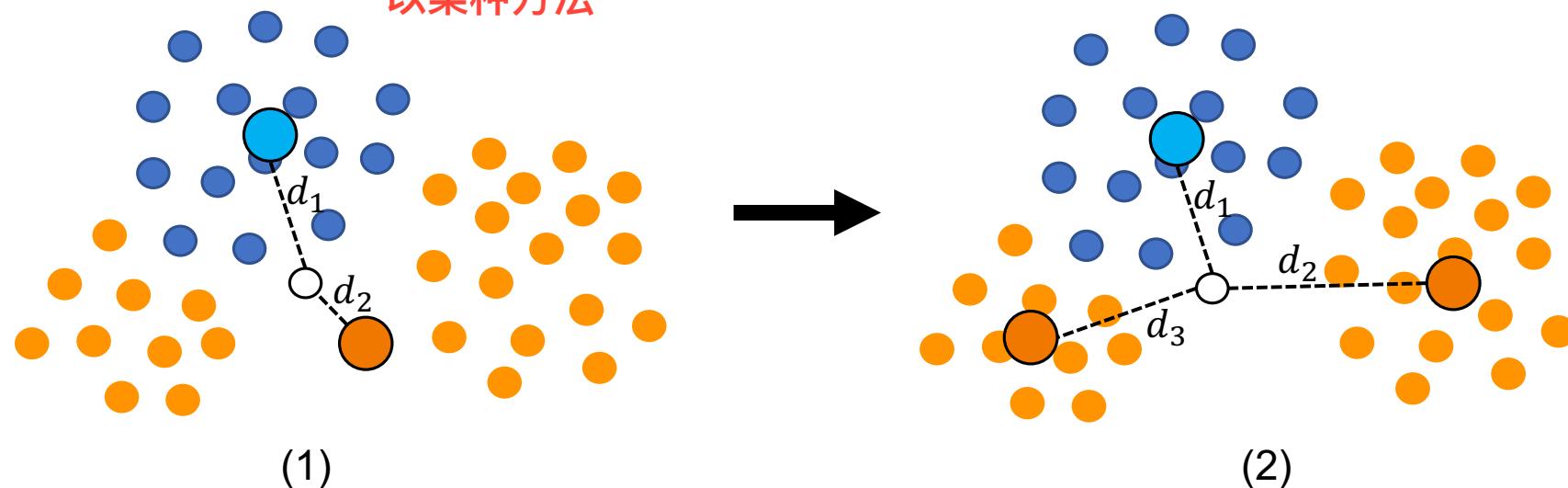
# Basic Linear Classifier & Nearest Centroid Classifier

- The basic linear classifier is distance-based.
- An alternative, distance-based way to classify instances without direct reference to a decision boundary is by the following decision rule: if  $x$  is nearest to  $\mu^{\oplus}$  then classify it as positive, otherwise as negative; or equivalently, classify an instance to the class of the nearest exemplar.
- If we use Euclidean distance as our closeness measure, simple geometry tells us we get exactly the same decision boundary.
- So the basic linear classifier can be interpreted from a distance-based perspective as constructing exemplars that minimise squared Euclidean distance within each class, and then applying a nearest-exemplar decision rule.

# Nearest Centroid Classifier

- What happens if a class has more than one mode? (similar to the image)
  1. If there is only one centroid per class, then it will perform poorly
  2. If we can somehow find different modes, we can define one centroid per each mode which helps the classifier

以某种方法



# Nearest Centroid Classifier

Advantages:

- Simple
- Fast
- works well when classes are compact and far from each other.

# Nearest Centroid Classifier

Disadvantages:

- For complex classes (eg. Multimodal, non-spherical) may give very poor results
- Can not handle outliers and noisy data well
- Can not handle missing data

# Nearest neighbour classification

# Nearest neighbour classification

死记硬背

- Related to the simplest form of learning: rote learning or memorization
  - Training instances are searched for instance that **most closely resembles** new or *query* instance
  - The instances themselves represent the knowledge
  - Called: *instance-based*, *memory-based* learning or *case-based* learning; often a form of *local* learning
- The *similarity* or *distance* function defines “learning”, i.e., how to go beyond simple memorization
- Intuitive idea — instances “close by”, i.e., neighbours or *exemplars*, should be classified similarly
- Instance-based learning is lazy learning
- Methods: *nearest-neighbour*, *k-nearest-neighbour*, *lowess*, . . .
- Ideas also important for *unsupervised* methods, e.g., clustering (later lectures)

# Nearest Neighbour

Stores all training examples  $\langle x_j, f(x_j) \rangle$ .

Nearest neighbour:

- Given query instance  $x_q$ , first locate nearest training example  $x_n$ , then estimate  $\hat{f}(x_q) \leftarrow f(x_n)$

$k$ -Nearest neighbour:

- Given  $x_q$ , take vote among its  $k$  nearest neighbours (if discrete-valued target function) (see next slide)
- take mean of  $f$  values of  $k$  nearest neighbours (if real-valued)

$$\hat{f}(x_q) \leftarrow \frac{\sum_{j=1}^k f(x_j)}{k}$$

# $k$ -Nearest Neighbour Algorithm

Training algorithm:

- For each training example  $(x_j, f(x_j))$ , add the example to the list *training \_examples*.

Classification algorithm:

- Given a query instance  $x_q$  to be classified,
  - Let  $x_1, \dots, x_k$  be the  $k$  instances from *training examples* that are *nearest* to  $x_q$  by the distance function
  - Return

V:set of all categories

$$\hat{f}(x_q) \leftarrow \arg \max_{v \in V} \sum_{i=1}^k \delta(v, f(x_i))$$

Where  $\delta(a, b) = 1$  if  $a = b$  and 0 otherwise.

# Distance function again

The distance function defines what is learned.

Instance  $x_j$  is described by a feature vector (list of attribute-value pairs)

$$x_j = (x_{j1}, \dots, x_{jd})^T$$

Where  $x_{jr}$  denotes the value of the  $r$ th attribute/feature of  $x_j$ .

Most commonly used distance function is *Euclidean distance* . . .

- distance between two instances  $x_i$  and  $x_j$  is defined to be

$$Dis(x_i, x_j) = \sqrt{\sum_{r=1}^d (x_{ir} - x_{jr})^2}$$

# Distance function again

Many other distance functions could be used . . .

- e.g., *Manhattan* or *city-block* distance (sum of absolute values of differences between attributes)

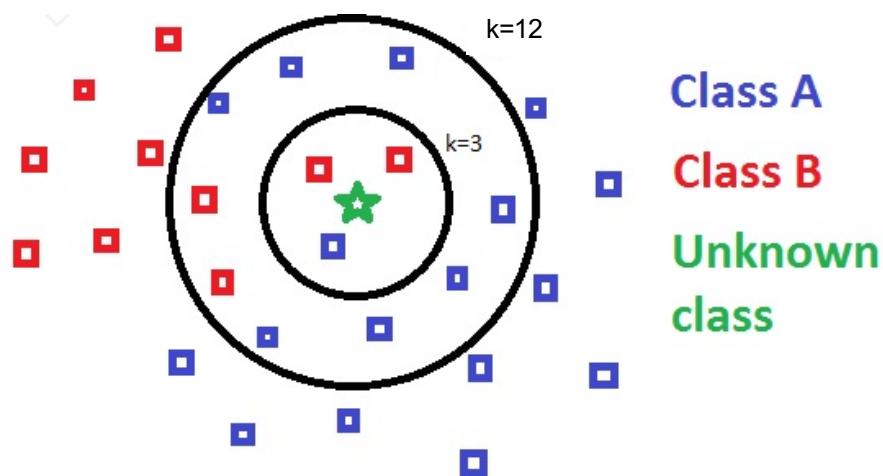
$$Dis(x_i, x_j) = \sum_{r=1}^d |x_{ir} - x_{jr}|$$

Vector-based formalization – use norm  $L_1$ ,  $L_2$ , ...

The idea of distance functions will appear again in *kernel methods*.

# kNN Example

- What is the predicted class for the green point given the data for



<https://towardsdatascience.com/knn – using – scikit – learn – c6bed765be75>

# Normalization and other issues

- Different attributes measured on different scales (for example one attribute/feature may have a range of [0,100] and another have a range of [-1,1])
- Need to be *normalized* (why ?)

$$x'_{jr} = \frac{x_{jr} - \min(x_{jr})}{\max(x_{jr}) - \min(x_{jr})}$$

where  $x_{jr}$  is the actual value of attribute/feature  $r$  and  $x'_{jr}$  is the normalised value.

- Nominal attributes: distance either 0 or 1

# When To Consider Nearest Neighbour

- Instances map to points in  $\mathbb{R}^d$
- Less than 20 attributes per instance
  - or number of attributes can be reduced . . .
- Lots of training data
- No requirement for “explanatory” model to be learned

# K-Nearest Neighbour

Advantages:

- Statisticians have used  $k$ -NN since early 1950s
- Can be very accurate
- Training is very fast
- Can learn complex target functions

# K-Nearest Neighbour

Disadvantages:

- Slow at query time: basic algorithm scans entire training data to derive a prediction
- “Curse of dimensionality”
- Assumes all attributes are equally important, so easily fooled by irrelevant attributes
  - Remedy: attribute selection or weights
- Problem of noisy instances:
  - Remedy: remove from data set
  - not easy – how to know which are noisy ?
- Needs homogenous feature type and scale
- Finding the optimal number of neighbors ( $k$ ) can be challenging

# Inductive Bias of $k$ -NN

The **inductive bias** (a.k.a. learning bias) of a learning algorithm is the set of assumptions that the learner uses to predict outputs given unseen inputs.

What is the inductive bias of  $k$ -NN ?

- an assumption that the classification of query instance  $x_q$  will be most similar to the classification of other instances that are nearby according to the distance function

# Nearest-neighbour classifier

- 1NN perfectly separates training data, so low bias but high variance
- By increasing the number of neighbours  $k$  we increase bias and decrease variance (what happens when  $k = m$ ?  $m$  is the number of observations)
- Easily adapted to real-valued targets, and even to structured objects (nearest-neighbour retrieval). Can also output probabilities when  $k > 1$

# Distance-Weighted kNN

- Might want to weight nearer neighbours more heavily ...
- Use distance function to construct a weight  $w_i$
- Replace the final line of the classification algorithm by:

$$\hat{f}(x_q) \leftarrow \arg \max_{v \in V} \sum_{i=1}^k w_i \delta(v, f(x_j))$$

Where

$$w_i = \frac{1}{Dis(x_q, x_i)^2}$$

$Dis(x_q, x_i)$  is distance between  $x_q, x_i$

# Distance-Weighted kNN

For real-valued target functions replace the final line of the algorithm by:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}$$

(denominator normalizes contribution of individual weights).

Now we can consider using all the training examples instead of just  $k$ :

- using all examples (i.e., when  $k = m$  and  $m$  is number of training samples) with the rule above is called *Shepard's method*

# Evaluation

如何理解调参数：调优参数（tuning parameters），需要人为设定，称为超参数（Hyperparameter）

梯度下降法中的学习速率 $\alpha$ ，迭代次数epoch，批量大小batch-size，k近邻法中的 $k$ （最相近的点的个数），决策树模型中树的深度，等等

Lazy learners do not construct an explicit model, so how do we evaluate the output of the learning process ?

- 1-NN – training set error is always zero !
  - each training example is always closest to itself
- $k$ -NN – overfitting may be hard to detect

Solution:

*Leave-one-out cross-validation (LOOCV)* – leave out each example and predict it given the rest:

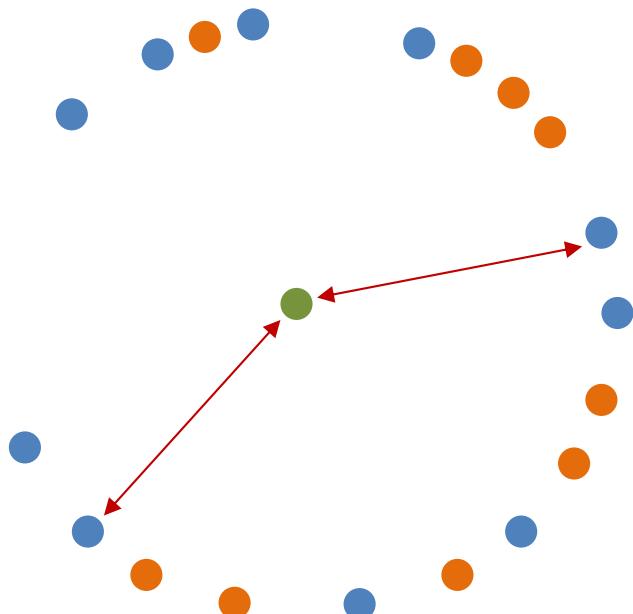
$$(x_1, y_1), (x_2, y_2), \dots, (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \dots, (x_m, y_m)$$

Error is mean over all predicted examples. Fast – no models to be built !

# kNN Computational Time

- kNN uses the training data as exemplars, so using simple search for prediction is  $O(n)$ !
- There are algorithms to search for neighbours more efficiently with  $O(\log n)$  but they do not work very well for above 10 dimensions (more than 10 features/attributes)
- For above 10 dimension, there are some approximate nearest neighbour approaches that can improve computation by orders of magnitude
- In high dimensional space (e.g. above 20 dimensions) even with using such algorithms, the kNN doesn't work well
- In high-dimensional spaces everything is far away from everything and so pairwise distances are uninformative (curse of dimensionality)

# When is kNN meaningful?

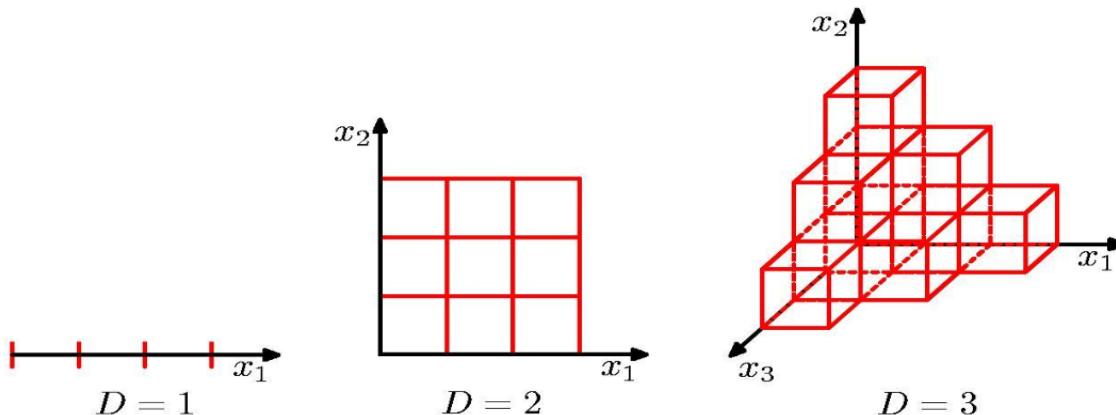


You may think that this is an exceptional example, and this doesn't really happen in practice!!

# Curse of Dimensionality

- It can be shown that as dimensions increase the effectiveness of distance metrics decrease and the concept of proximity may not be qualitatively meaningful as all points look equidistant
- This is one symptom of having high dimensional space (curse of dimensionality)
- There are also other problems arising from curse of dimensionality:
  - It becomes polynomially harder to estimate many parameters (e.g covariances)
  - It becomes more difficult to visualize data
  - Enormous amount of data is needed to train a model

# Curse of Dimensionality



- number of “cells” in the instance space grows exponentially in the number of features
- with exponentially many cells we would need exponentially many data points to ensure that each cell is sufficiently populated to make nearest-neighbour predictions reliably

# Curse of Dimensionality

Bellman (1960) coined this term in the context of dynamic programming

Imagine instances described by 20 attributes, but only 2 are relevant to target function — “similar” examples will appear “distant”.

*Curse of dimensionality*: nearest neighbour is easily mislead when dealing with high-dimensional  $x$  in terms of the number of features – problem of irrelevant attributes

One approach:

- Stretch  $j$ th axis by weight  $z_j$ , where  $z_1, \dots, z_d$  chosen to minimize prediction error
- Use cross-validation to automatically choose weights  $z_1, \dots, z_d$
- Note setting  $z_j$  to zero eliminates this dimension altogether

# Curse of Dimensionality

Some ideas to address this for instance-based (nearest-neighbour) learning

- Euclidean distance with weights on attributes

$$Dis(x_q, x_i) = \sqrt{\sum_{r=1}^d z_r (x_{qr} - x_{ir})}$$

- updating of weights based on nearest neighbour classification error
  - class correct/incorrect: weight increased/decreased
  - can be useful if not all features used in classification

See Moore and Lee (1994) “Efficient Algorithms for Minimizing Cross Validation Error”

# Instance-based (nearest-neighbour) learning

Recap – Practical problems of 1-NN scheme:

- Slow (but fast  $k$ -dimensional tree-based approaches exist)
  - Remedy: removing irrelevant data
- Noise (but  $k$ -NN copes quite well with noise)
  - Remedy: removing noisy instances
- All attributes deemed equally important
  - Remedy: attribute weighting (or simply selection)

# Some refinements of instance-based classifiers

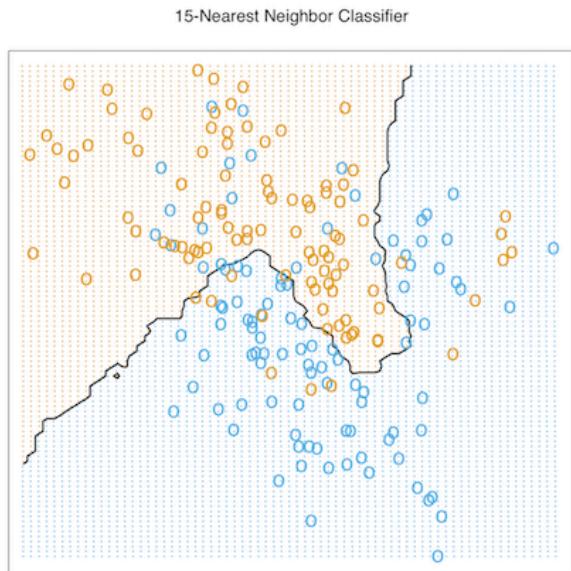
- Edited NN classifiers discard some of the training instances before making predictions
- Saves memory and speeds up classification
- IB2: incremental NN learner: only incorporates misclassified instances into the classifier
  - Problem: noisy data gets incorporated
- IB3: store classification performance information with each instance & only use in prediction if above a threshold

# Dealing with noise

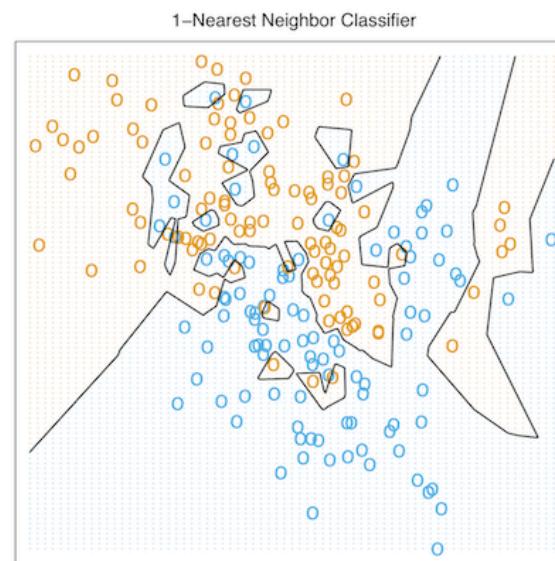
Use larger values of  $k$  (why ?) How to find the “right”  $k$  ?

- One way: cross-validation-based  $k$ -NN classifier (but slow)
- Different approach: discarding instances that don’t perform well by keeping success records of how well an instance does at prediction (IB3)
  - Computes confidence interval for an instance’s success rate and for default accuracy of its class
  - If lower limit of first interval is above upper limit of second one, instance is accepted (IB3: 5%-level)
  - If upper limit of first interval is below lower limit of second one, instance is rejected (IB3: 12.5%-level)

# kNN Example



**FIGURE 2.2.** The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (**BLUE** = 0, **ORANGE** = 1) and then fit by 15-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the 15-nearest neighbors.



**FIGURE 2.3.** The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (**BLUE** = 0, **ORANGE** = 1), and then predicted by 1-nearest-neighbor classification.

# kNN Example

- Automated MS-lesion segmentation by KNN
  - They have used some manually labeled image as the training set
  - They used 4 features: Intensity and voxel locations (x,y,z coordinates)
- 
- Ref: Anbeek et. Al, “Automated MS-lesion segmentation by K-nearest neighbor classification”, MIDAS journal, 2008

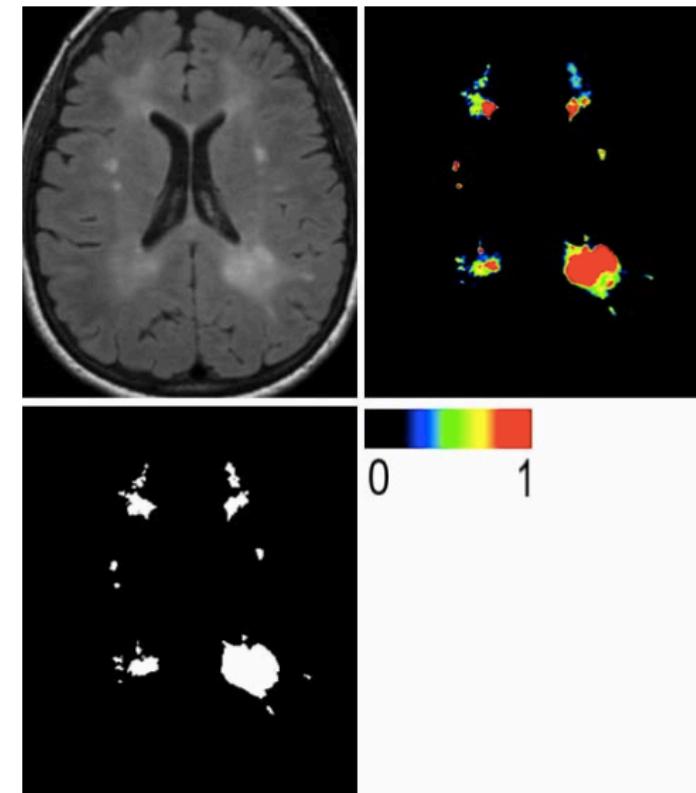


Figure 1 MS-lesion segmentation results. Top left: FLAIR image; top right: probabilistic segmentation, showing probability of lesion per voxel (see color bar); down left: binary segmentation, derived from probabilistic segmentation with threshold 0.4.

# Summary

- A framework for classification
- Classification viewed in terms of distance in feature space
- Distance-based
- A classifier as a linear model
- Nearest neighbour classifiers
- Later we will see how to extend by building on these ideas

# Acknowledgements

- Material derived from slides for the book  
“Elements of Statistical Learning (2nd Ed.)” by T. Hastie, R. Tibshirani & J. Friedman. Springer (2009) <http://statweb.stanford.edu/~tibs/ElemStatLearn/>
- Material derived from slides for the book  
“Machine Learning: A Probabilistic Perspective” by P. Murphy MIT Press (2012)  
<http://www.cs.ubc.ca/~murphyk/MLbook>
- Material derived from slides for the book “Machine Learning” by P. Flach Cambridge University Press (2012) <http://cs.bris.ac.uk/~flach/mlbook>
- Material derived from slides for the book  
“Bayesian Reasoning and Machine Learning” by D. Barber Cambridge University Press (2012)  
<http://www.cs.ucl.ac.uk/staff/d.barber/brml>
- Material derived from slides for the book “Machine Learning” by T. Mitchell McGraw-Hill (1997)  
<http://www- 2.cs.cmu.edu/~tom/mlbook.html>
- Material derived from slides for the course “Machine Learning” by A. Srinivasan BITS Pilani, Goa, India (2016)