Introductory Applied Machine Learning

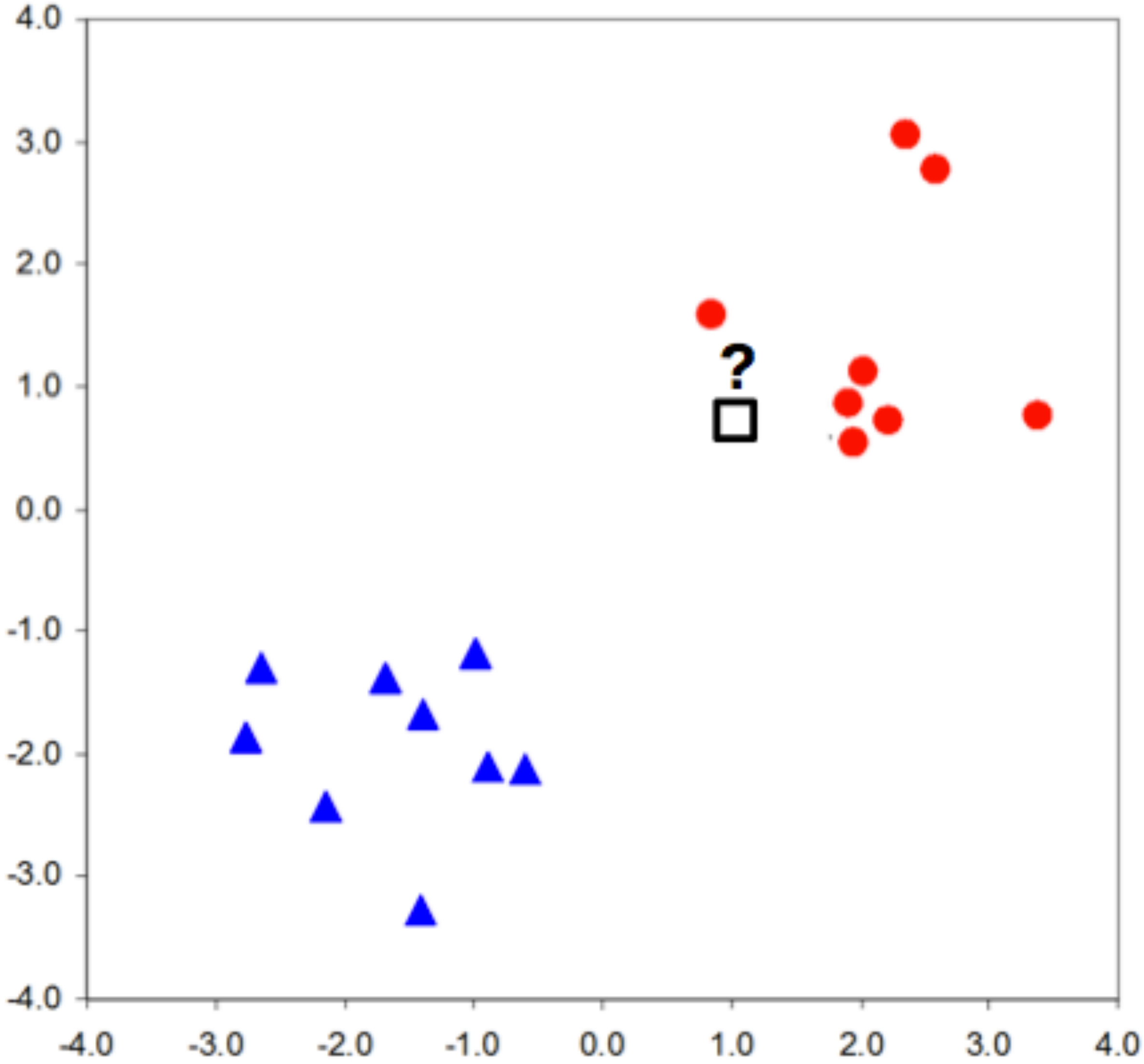
Nearest Neighbour Methods

Victor Lavrenko and Nigel Goddard
School of Informatics

Overview

- Nearest neighbour method
  - classification and regression
  - practical issues: k, distance, ties, missing values
  - optimality and assumptions
- Making kNN fast:
  - K-D trees
  - inverted indices
  - fingerprinting
- References: W&F sections 4.7 and 6.4

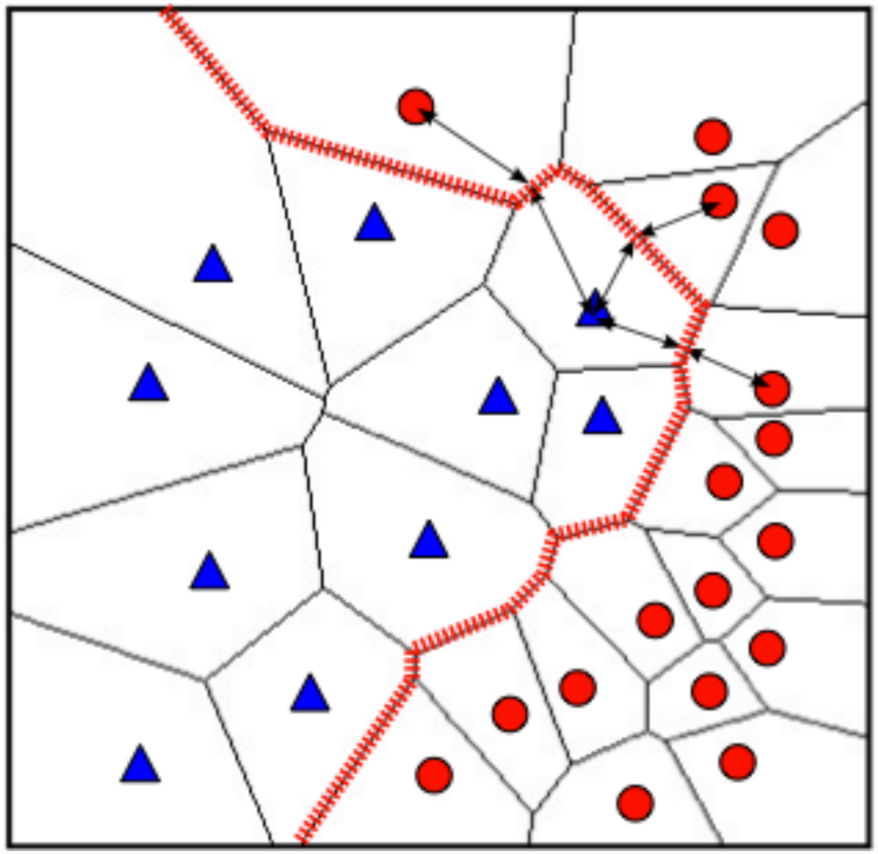
Intuition for kNN



- set of points (x,y)
  - two classes
- is the box red or blue
- how did you do it
  - use Bayes rule?
  - a decision tree?
  - fit a hyperplane?
- nearby points are red
  - use this as a basis for a learning algorithm

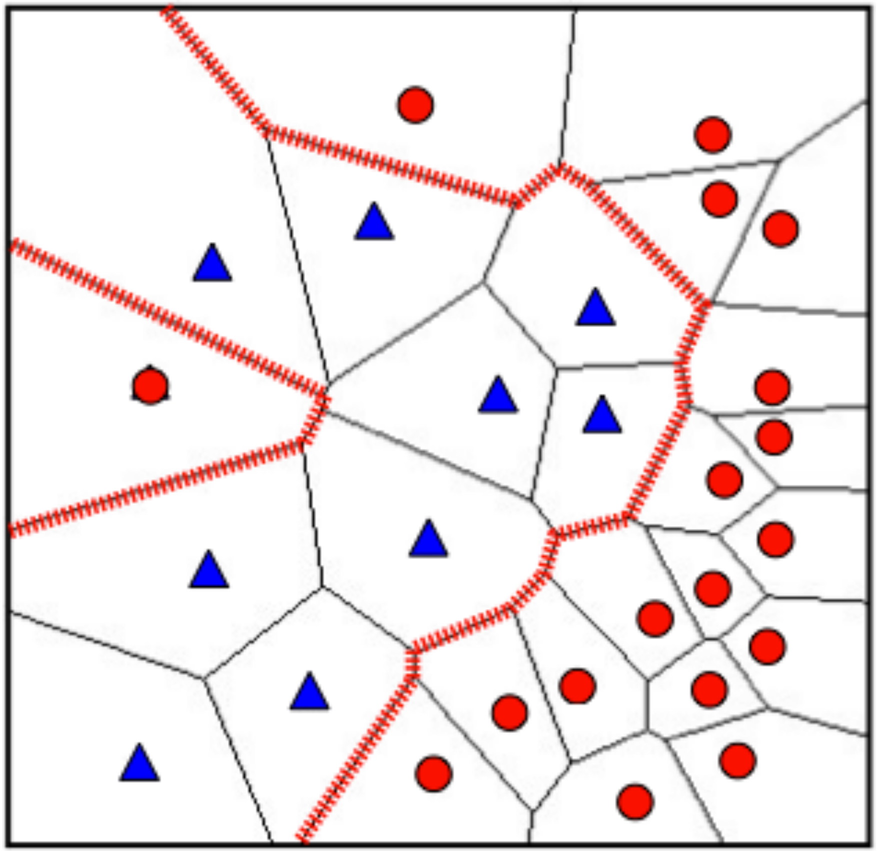
Nearest-neighbor classification

- Use the intuition to classify a new point x:
  - find the most similar training example x'
  - predict its class y'
- Voronoi tesselation
  - partitions space into regions
  - boundary: points at same distance from two different training examples
- classification boundary
  - non-linear, reflects classes well
  - compare to NB, DT, logistic
  - impressive for simple method



Nearest neighbour: outliers

- Algorithm is sensitive to outliers
  - single mislabeled example dramatically changes boundary
- No confidence P(y|x)
- Insensitive to class prior
- Idea:
  - use more than one nearest neighbor to make decision
  - count class labels in k most similar training examples
    - many “triangles” will outweigh single “circle” outlier

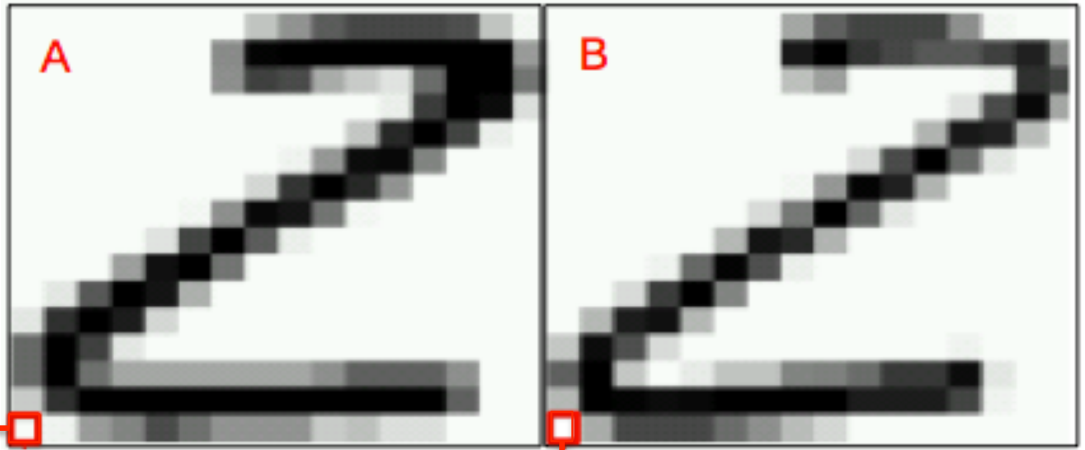


kNN classification algorithm

- Given:
  - training examples $\{x_i, y_i\}$
    - x_i … attribute-value representation of examples
    - y_i … class label: {ham,spam}, digit {0,1,…9} etc.
  - testing point x that we want to classify
- Algorithm:
    - compute distance $D(x,x_i)$ to every training example x_i
    - select k closest instances $x_{i1}...x_{ik}$ and their labels $y_{i1}...y_{ik}$
    - output the class y^* which is most frequent in $y_{i1}...y_{ik}$

Example: handwritten digits

- 16x16 bitmaps
- 8-bit grayscale
- Euclidian distance
  - over raw pixels



$$D(A,B)=\sqrt{\sum_r \sum_c \left(A_{r,c}-B_{r,c}\right)^2}$$

- Accuracy:
  - 7-NN ~ 95.2%
  - SVM ~ 95.8%
  - humans ~ 97.5%

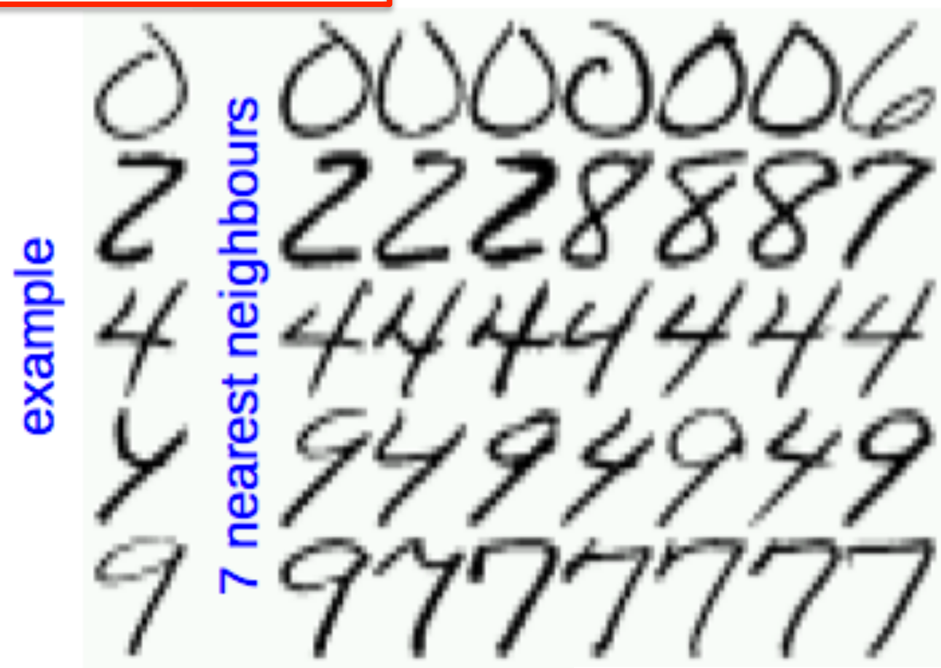


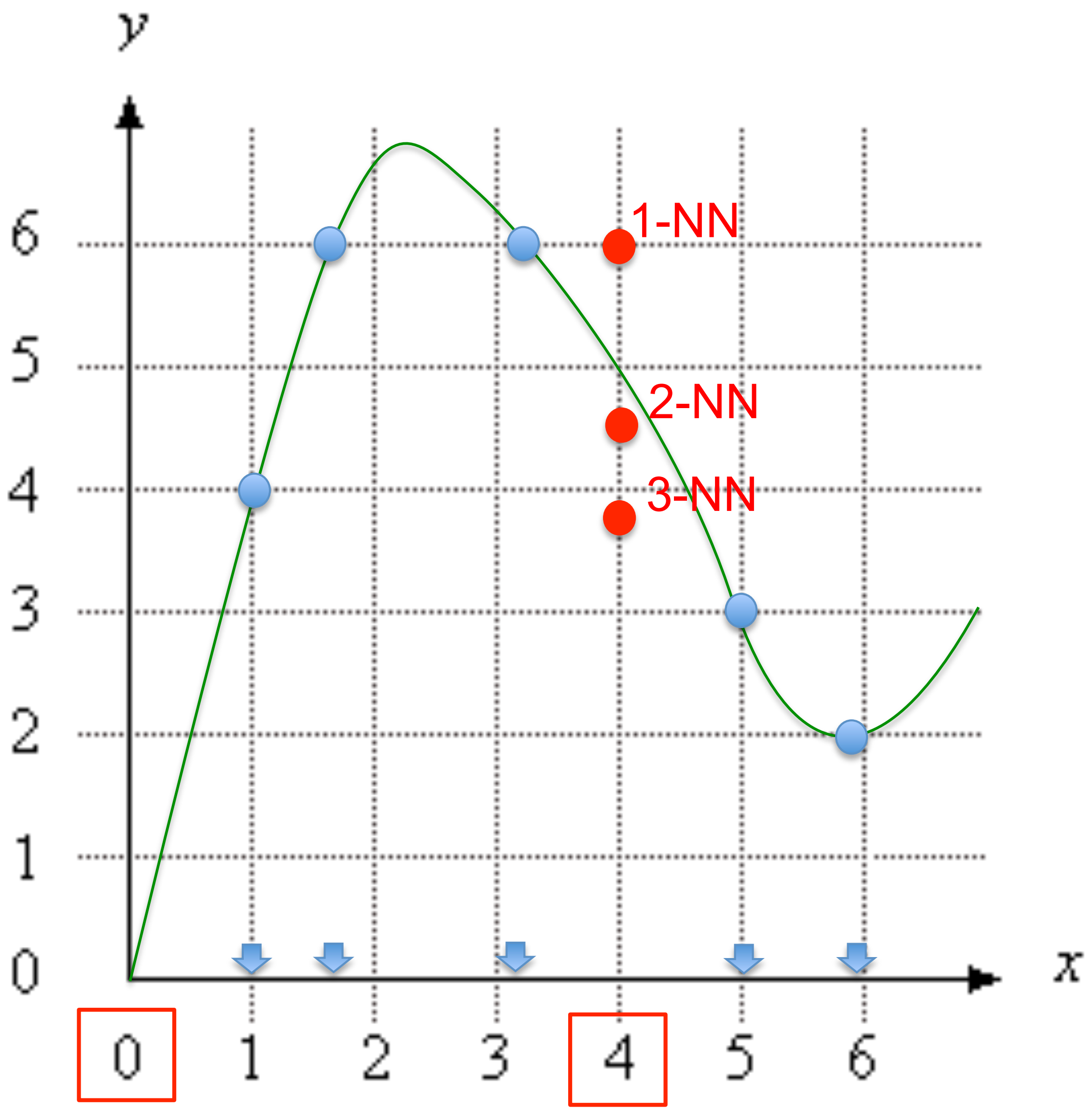
Figure © Sam Roweis, 2006

kNN regression algorithm

- Given:
  - training examples $\{x_i, y_i\}$
    - x_i … attribute-value representation of examples
    - y_i … real-valued target (profit, rating on YouTube, etc)
  - testing point x that we want to predict the target
- Algorithm:
    - compute distance $D(x,x_i)$ to every training example x_i
    - select k closest instances $x_{i1} \ldots x_{ik}$ and their labels $y_{i1} \ldots y_{ik}$
    - output the mean of $y_{i1} \ldots y_{ik}$:

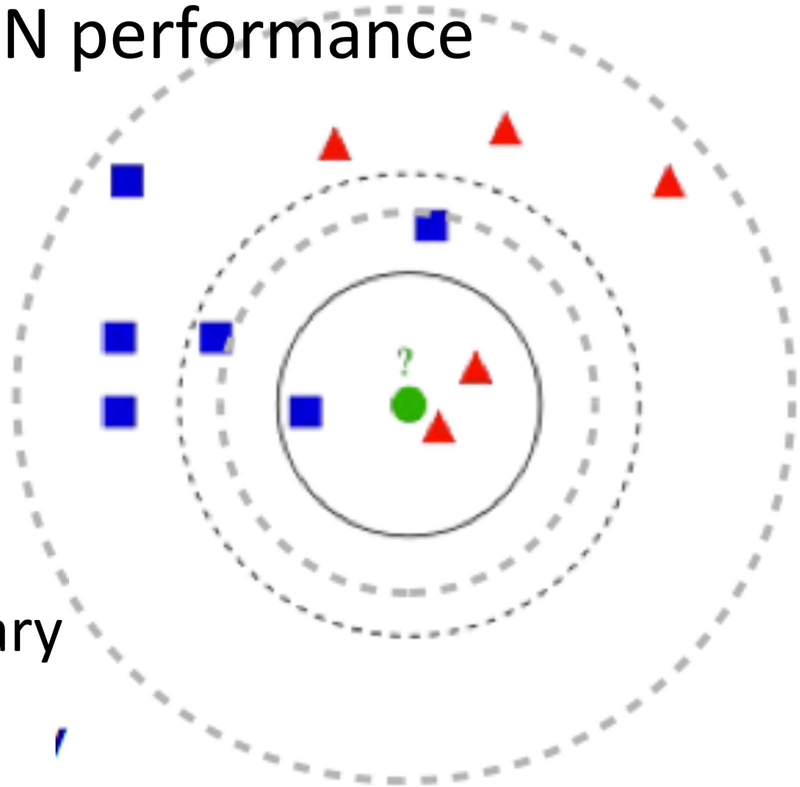
$$\hat{y} = f(x) = \frac{1}{k} \sum_{j=1}^{k} y_{i_j}$$

Example: kNN regression in 1-d



Choosing the value of k

- Value of k has strong effect on kNN performance
  – large value → everything classified as the most probable class: P(y)
  – small value → highly variable, unstable decision boundaries
    - small changes to training set → large changes in classification
  – affects "smoothness" of the boundary



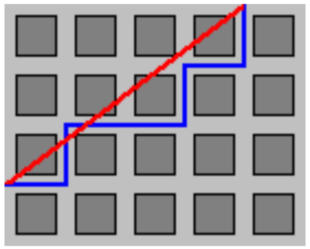
- Selecting the value of k
  – set aside a portion of the training data (validation set)
  – vary k, observe training → validation error
  – pick k that gives best generalization performance

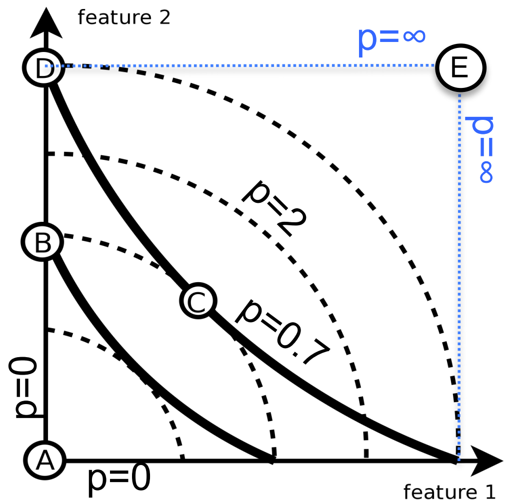
Distance measures

- Key component of the kNN algorithm
  - defines which examples are similar & which aren't
  - can have strong effect on performance
- Euclidian (numeric attributes): $D(x,x') = \sqrt{\sum_d \left|x_d - x'_d\right|^2}$
  - symmetric, spherical, treats all dimensions equally
  - sensitive to extreme differences in single attribute
    - behaves like a "soft" logical OR
- Hamming (categorical attributes): $D(x,x') = \sum_d 1_{x_d \neq x'_d}$
  - number of attributes where *x, x'* differ

Distance measures (2)

- Minkowski distance (p-norm): $D(x,x') = \sqrt[p]{\sum_d |x_d - x'_d|^p}$
  - p=2: Euclidian
  - p=1: Manhattan
  - p=0: Hamming ... logical AND
  - p=∞: $\mathbf{max}_d\ |x_d - x'_d|$ ... logical OR



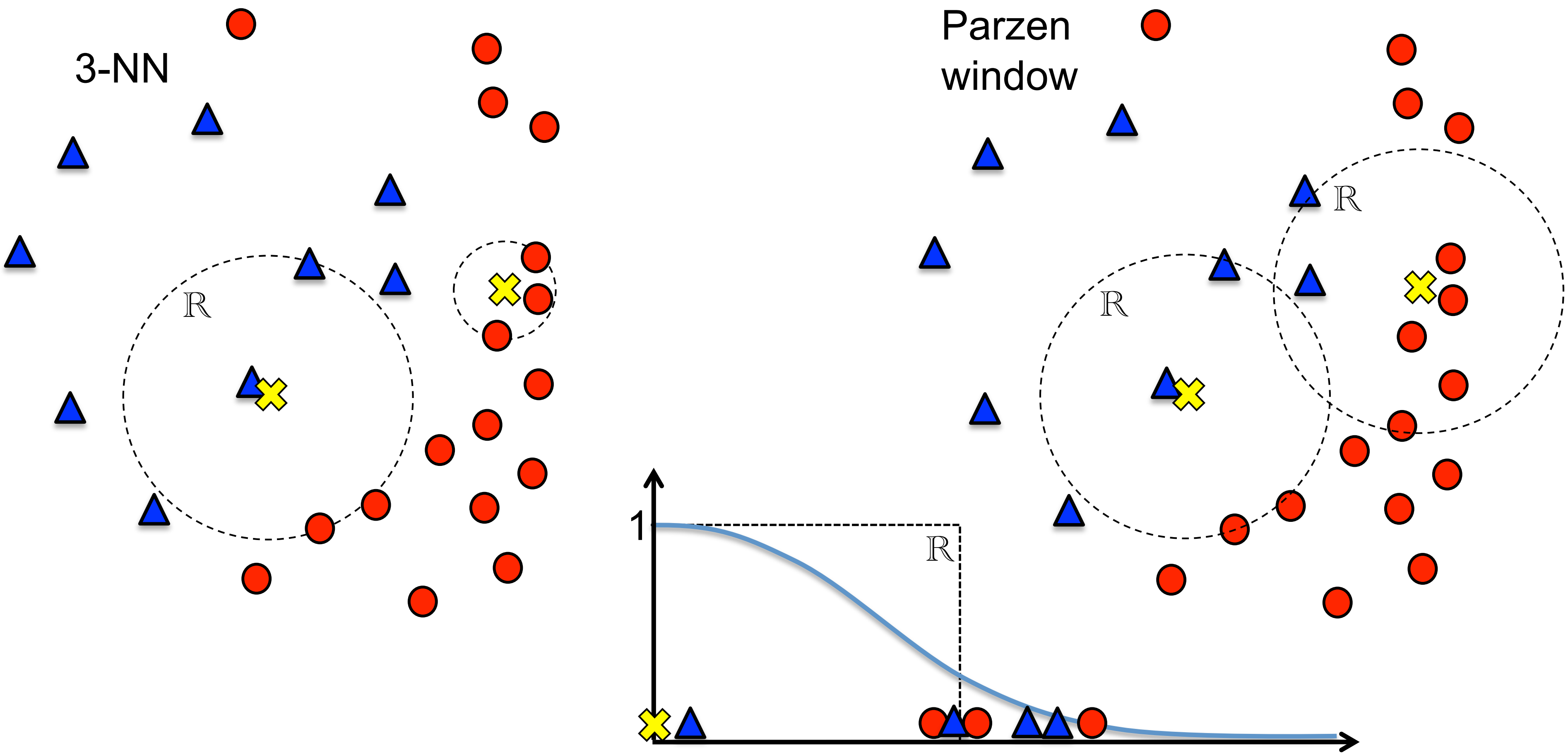


- Kullback-Leibler (KL) divergence:
  - for histograms (x_d>0, $\Sigma_d x_d$ = 1): $D(x,x') = -\sum_d x_d \log \frac{x_d}{x'_d}$
  - asymmetric, excess bits to encode x with x'
- Custom distance measures (*BM25* for text)

kNN: practical issues

- Resolving ties:
  - equal number of positive/negative neighbours
  - use odd k (doesn't solve multi-class)
  - breaking ties:
    - random: flip a coin to decide positive / negative
    - prior: pick class with greater prior
    - nearest: use 1-nn classifier to decide
- Missing values
  - have to "fill in", otherwise can't compute distance
  - key concern: should affect distance as little as possible
  - reasonable choice: average value across entire dataset

kNN, Parzen Windows and Kernels



$$P(y \mid x) = \frac{1}{|R(x)|} \sum_{x_i \in R(x)} 1_{y_i = y} = \frac{\sum_i 1_{y_i = y} \cdot 1_{x_i \in R(x)}}{\sum_i 1_{x_i \in R(x)}} = \frac{\sum_i 1_{y_i = y} \cdot K(x_i, x)}{\sum_i K(x_i, x)}$$

kNN pros and cons

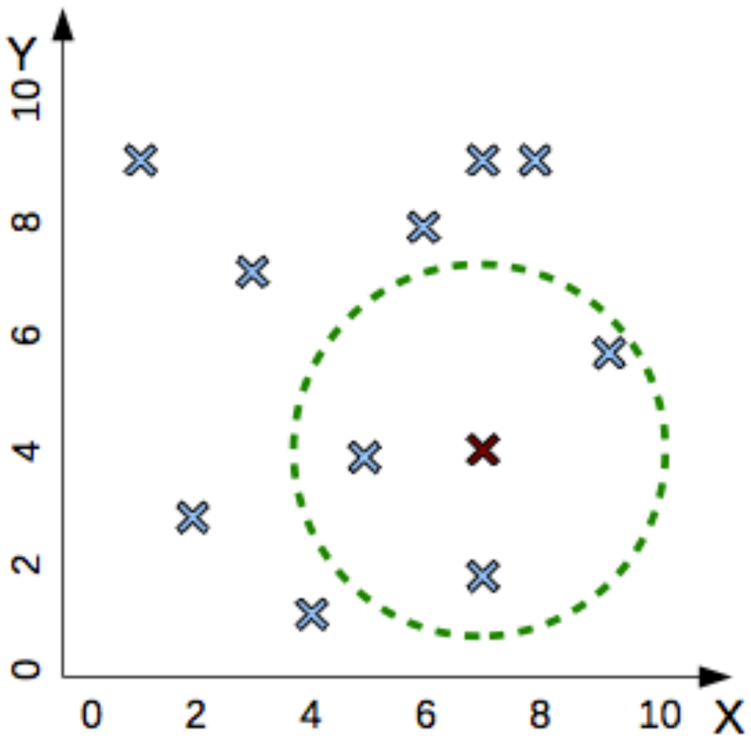
- Almost no assumptions about the data
  - smoothness: nearby regions of space → same class
  - assumptions implied by distance function (only locally!)
  - non-parametric approach: “let the data speak for itself”
    - nothing to infer from the data, except *k* and possibly D()
    - easy to update in online setting: just add new item to training set
- Need to handle missing data: fill-in or create a special distance
- Sensitive to class-outliers (mislabeled training instances)
- Sensitive to lots of irrelevant attributes (affect distance)
- Computationally expensive:
  - space: need to store all training examples
  - time: need to compute distance to all examples: O(*nd*)
    - *n* … number of training examples, *d* … cost of computing distance
    - *n* grows → system will become slower and slower
    - expense is at *testing*, not *training* time (bad)

Summary: kNN

- Key idea: nearby points → same class
  - important to select good distance function
- Can be used for classification and regression
- Simple, non-linear, asymptotically optimal
  - does not make assumptions about the data
  - "let the data speak for itself"
- Select k by optimizing error on held-out set
- Naïve implementations slow for big datasets
  - use K-D trees (low-d) or inverted lists (high-d)

Why is kNN slow?

What you see



Find nearest neighbors
of the testing point (red)

What algorithm sees

- Training set:

  {(1,9), (2,3), (4,1),
  (3,7), (5,4), (6,8),
  (7,2), (8,8), (7,9), (9,6)}

- Testing instance:

  (7,4)

- Nearest neighbors?

  compare one-by-one
  to each training instance

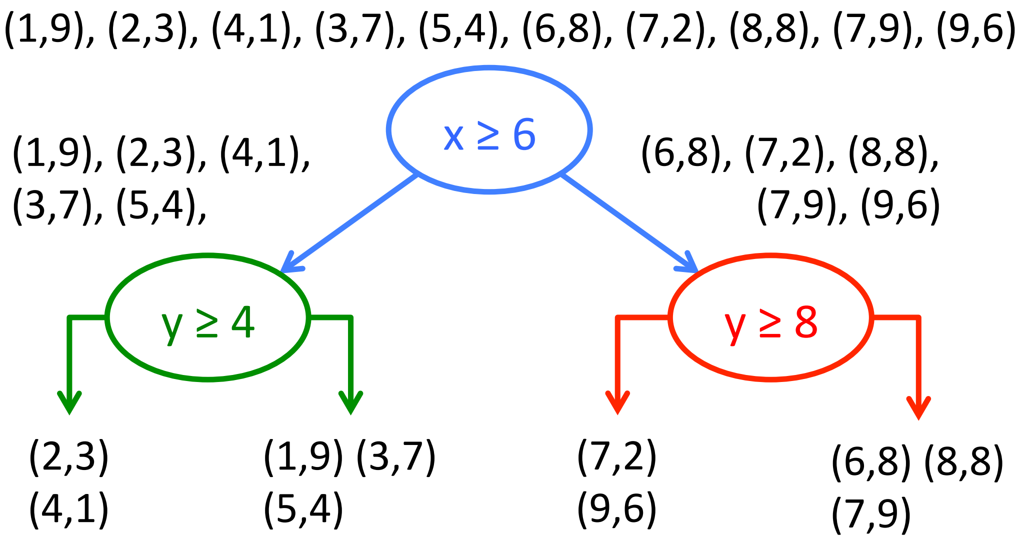
- n comparisons
- each takes d operations

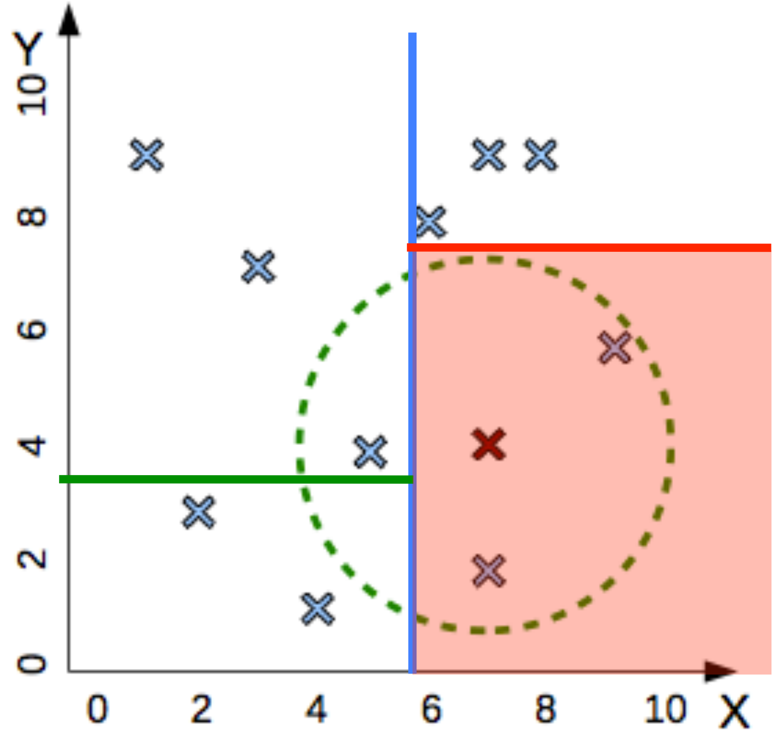
Making kNN fast

- Training: O(d), but testing: O(nd)
- Reduce d: dimensionality reduction
  - simple feature selection, other methods O(d^3)
- Reduce n: don't compare to **all** training examples
  - idea: quickly identify m<<n potential near neighbors
    - compare only to those, pick k nearest neighbors → O(md) time
  - **K-D trees**: low-dimensional, real-valued data
    - O ($d \log_2 n$), only works when d << n, inexact: may miss neighbors
  - **inverted lists**: high-dimensional, discrete data
    - O ($n'd'$) where d'<<d, n'<<n, only for sparse data (e.g. text), exact
  - **locality-sensitive hashing**: high-d, discrete or real-valued
    - O($n'd$), n'<<n … bits in fingerprint, inexact: may miss near neighbors

K-D tree example

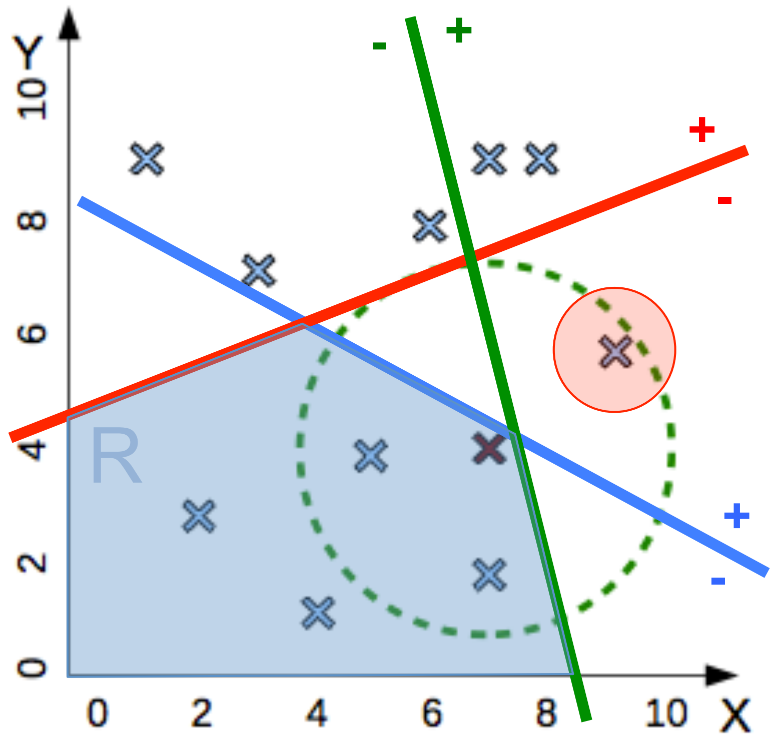
- Building a K-D tree from training data:
  - pick random dimension, find median, split data, repeat
- Find NNs for new point (7,4)
  - find region containing (7,4)
  - compare to all points in region





Locality-Sensitive Hashing (LSH)

- Random hyper-planes $\mathbf{h}_1 \ldots \mathbf{h}_k$
  - space sliced into 2^k regions (polytopes)
  - compare **x** only to training points in the same region R
- Complexity: $O(kd + dn/2^k)$
  - $O(kd)$ to find region R, $k \ll n$
    - dot-product **x** with $\mathbf{h}_1 \ldots \mathbf{h}_k$
  - compare to $n/2^k$ points in R
- Inexact: missed neighbors
  - repeat with different $\mathbf{h}_1 \ldots \mathbf{h}_k$
- Why not K-D tree?



Inverted list example

- Data structure used by search engines (Google, etc)
  - list all training examples that contain particular attribute
  - assumption: most attribute values are zero (sparseness)
- Given a new testing example:
  - merge inverted lists for attributes present in new example
  - O(dn): d … nonzero attributes, n … avg. length of inverted list

D1: "send your password"	spam
D2: "send us review"	ham
D3: "send us password"	spam
D4: "send us details"	ham
D5: "send your password"	spam
D6: "review your account"	ham

new email: "account review"