Lecture 19: K Nearest Neighbour & Locally Weighted Regression

Andreas Wichert

Department of Computer Science and Engineering

Técnico Lisboa

Parametric Models

- We select a hypothesis space and adjust a fixed set of parameters with the training data, *y = y(**x**, **w**)*
- We assume that the parameters ***w*** summarise the training (compression)
- This methods are called parametric models
- Example:
    - Liner Regression
    - Non Linear Regression
    - Perceptron
    - Stochastic Gradient Descent
- When we have a small amount of data it makes sense to have a small set of parameters (avoiding overfitting)
- When we have a large quantity of data, overfitting is less an issue

Histogram

- To construct a histogram
  - Divide the range between the highest and lowest values in a distribution into several bins of equal size
  - Toss each value in the appropriate bin of equal size
  - The height of a rectangle in a frequency histogram represents the number of values in the corresponding bin

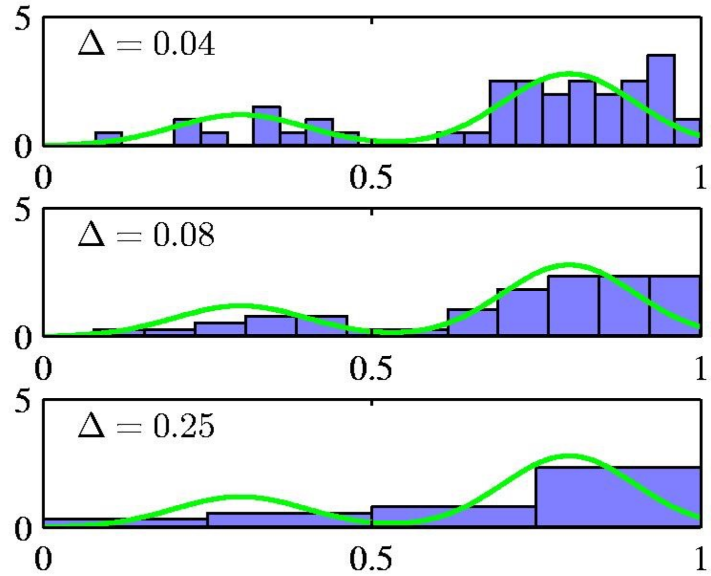
Nonparametric Methods

Histogram methods

partition the data space into distinct bins with widths and count the number of observations, n_i, in each bin.

$$p_i = \frac{n_i}{N\Delta_i}$$

Often, the same width is used for all bins,



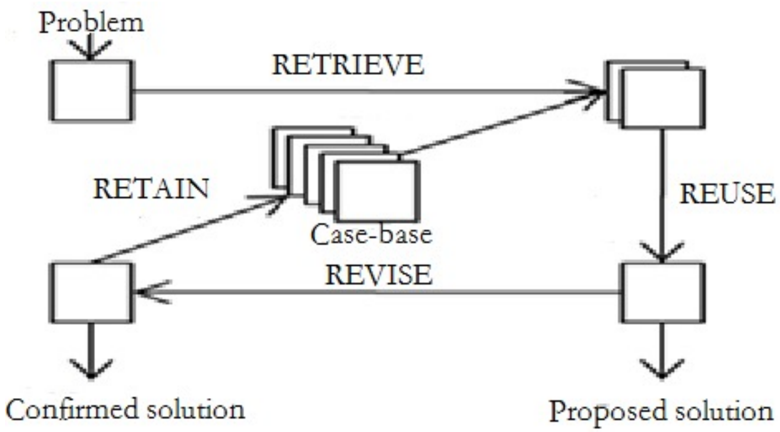
- In a D-dimensional space, using M bins in each dimension will require M^D bins!

Non Parametric Learning

- A non parametric model is one that can not be characterised by a fixed set of parameters
- A family of non parametric models is Instance Based Learning
- Instance based learning is based on the memorisation of the database
- There is not a model associated to the learned concepts
- The classification is obtained by looking into the memorised examples
- When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance

Case-based reasoning

- Instance-based methods can also use more complex, symbolic representations
- In case-based learning, instances are represented in this fashion and the process for identifying neighbouring instances is elaborated accordingly



- The cost of the learning process is *0*, all the cost is in the computation of the prediction
- This kind learning is also known as *lazy learning*
- One disadvantage of instance-based approaches is that the cost of classifying new instances can be high
  - Nearly all computation takes place at classification time rather than learning time
- Therefore, techniques for efficiently indexing training examples are a significant practical issue in reducing the computation required at query time

- A distance function is needed to compare the examples similarity
- The most popular metrics are the Taxicab or Manhattan metric d_1 with

$$d_1(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_1 = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_m - y_m|$$

- and the Euclidean metric

$$d_2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{|x_1 - y_1|^2 + |x_2 - y_2|^2 + \cdots + |x_m - y_m|^2}.$$

- This means that if we change the distance function, we change how examples are classified

K-Nearest Neighbor

- In nearest-neighbor learning the target function may be either discrete-valued or real valued
- Learning a discrete valued function

- $f : \Re^d \rightarrow V$, *V* is the finite set $\{v_1, \ldots\ldots, v_n\}$

- For discrete-valued, the *k*-NN returns the most common value among the k training examples nearest to x_q.
- $Data = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \cdots, , (\mathbf{x}_N, t_N)\}$

$$f(\mathbf{x}_\eta) = t_\eta = v_\eta$$

K-Nearest Neighbor

$$Data = \{(\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_1)), \cdots, , f(\mathbf{x}_N, (\mathbf{x}_N))\}$$

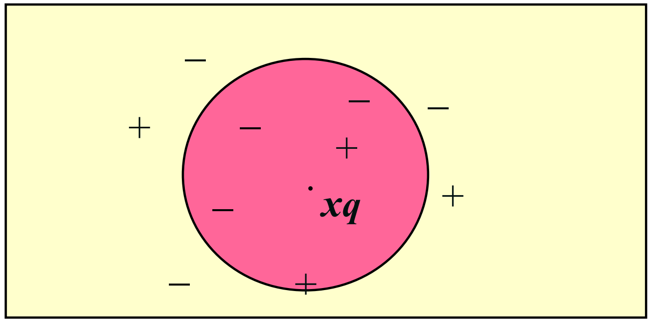
- Training algorithm
  - For each training example ***(x**,f(**x**))* add the example to the list
- Classification algorithm
  - Given a query instance x_q to be classified
    - Let $x_1,..,\mathbf{x}_k$ *k* instances which are nearest to $\mathbf{x}_q$

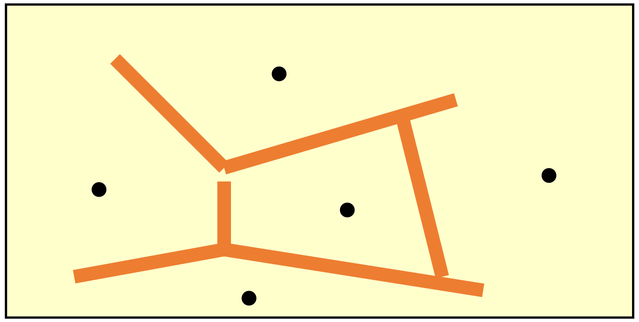
$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\arg\max} \sum_{i=1}^{k} \delta(v, f(x_i))$$

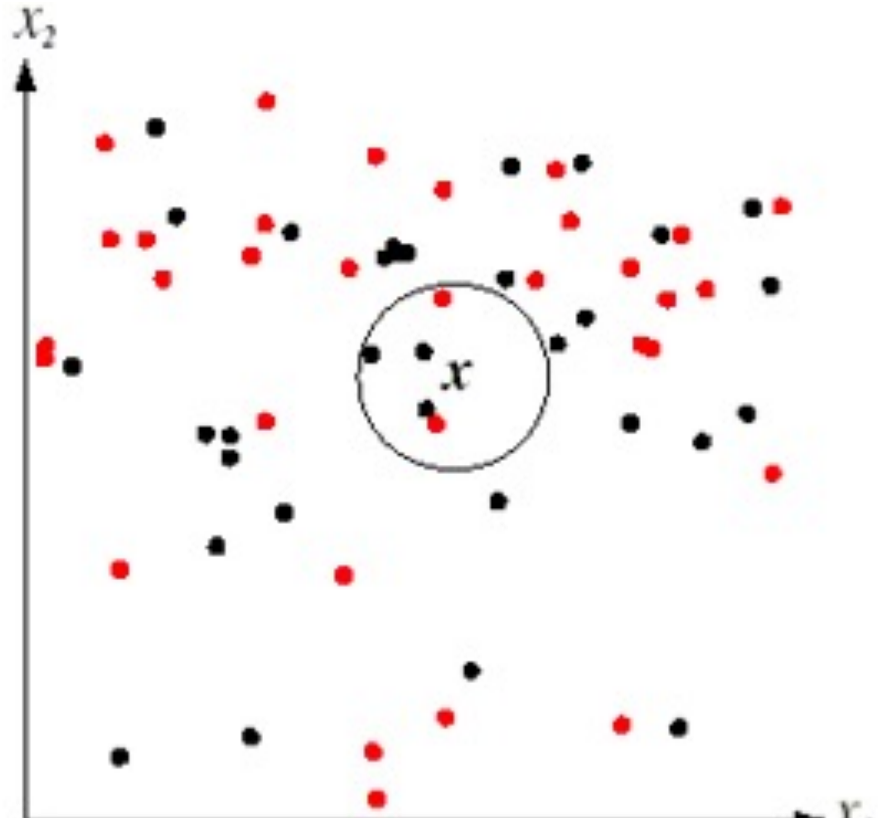
    - Where $\delta(a,b)=1$ if $a=b$, else $\delta(a,b)= 0$ (Kronecker function)

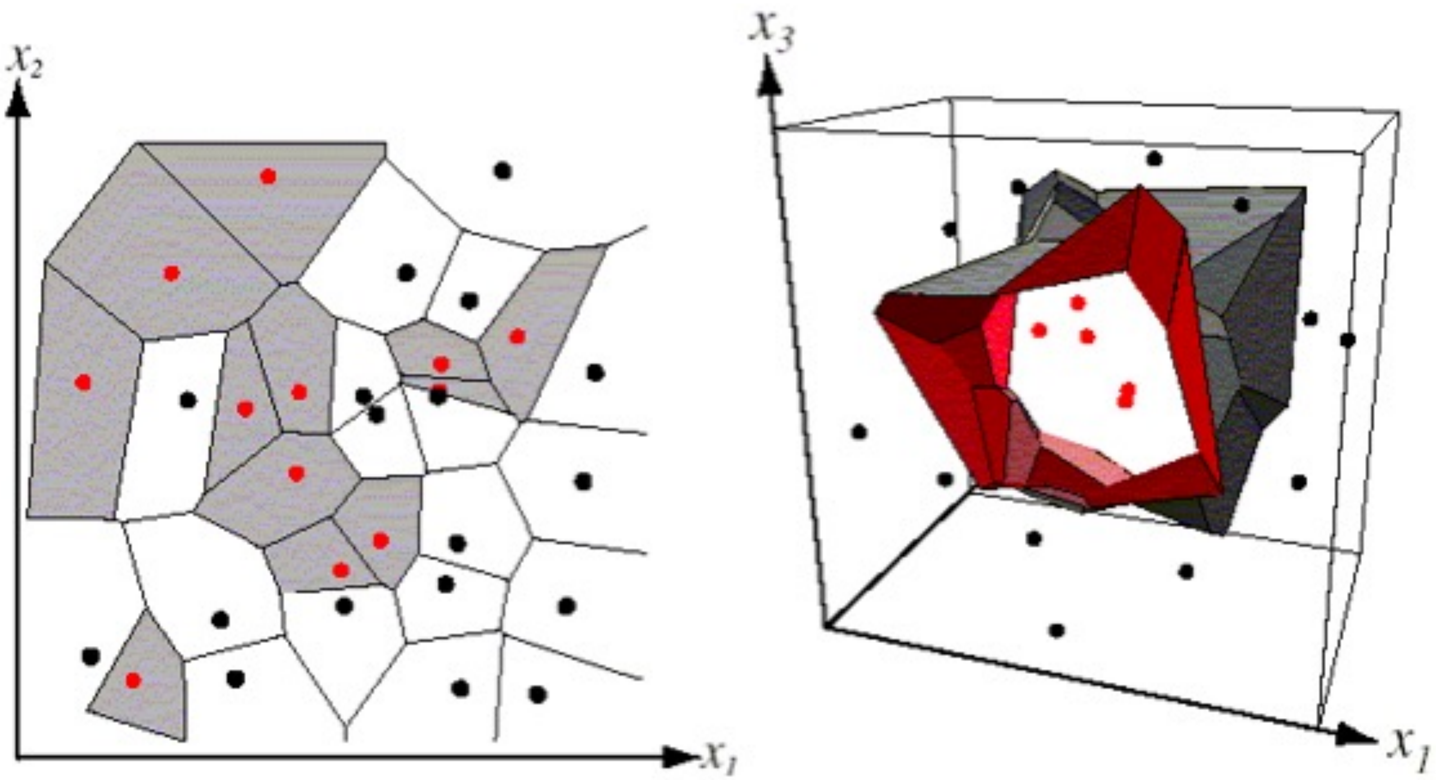
Definition of Voronoi diagram

- The decision surface induced by 1-NN for a typical set of training examples.



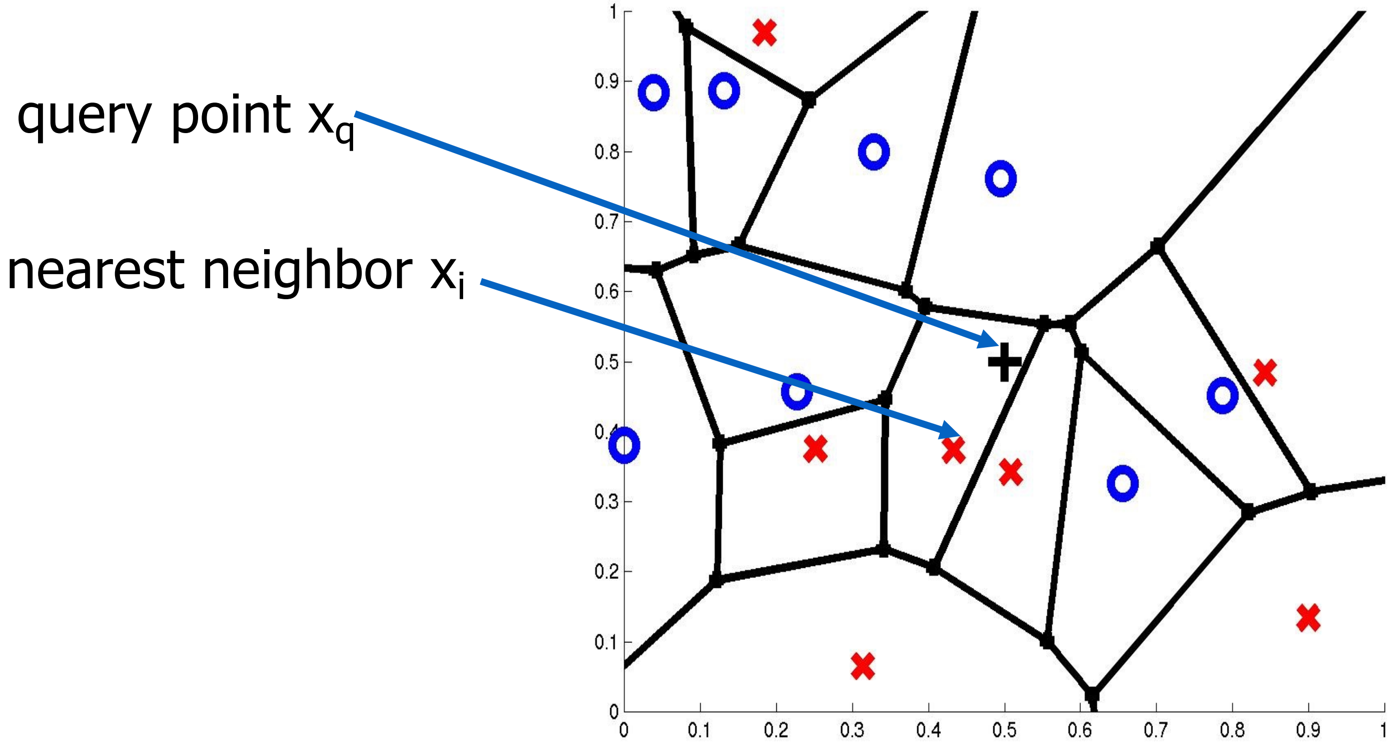



X_2
x
X_1



- kNN rule leeds to partition of the space into cells (Vornoi cells) enclosing the training points labelled as belonging to the same class
- The decision boundary in a Vornoi tessellation of the feature space resembles the surface of a crystall

1-Nearest Neighbor

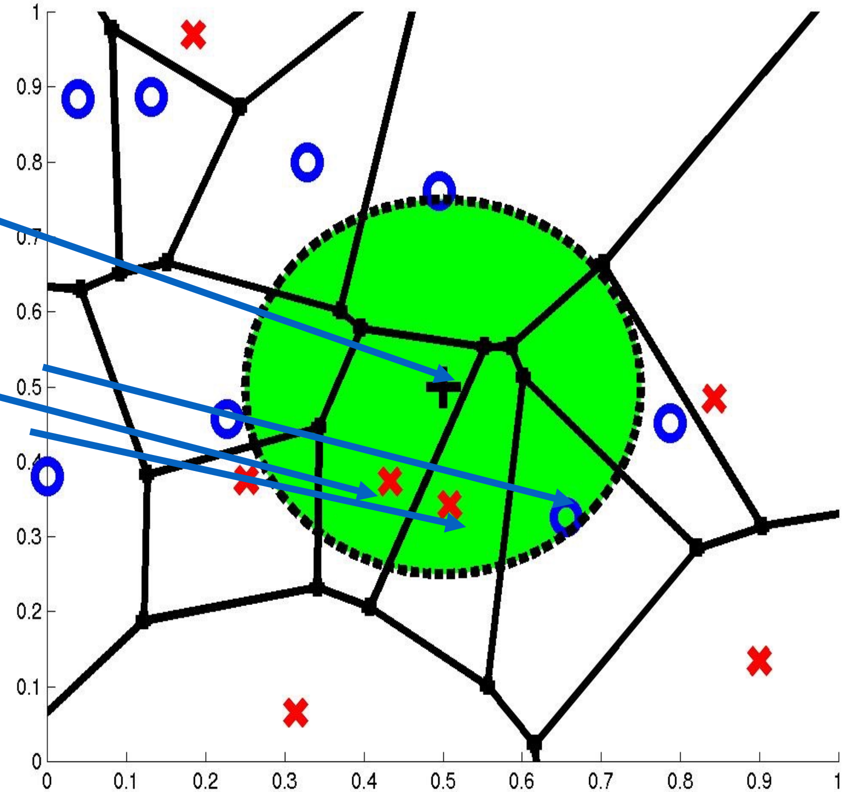


3-Nearest Neighbors

query point x_q

3 nearest neighbors

2x,1o

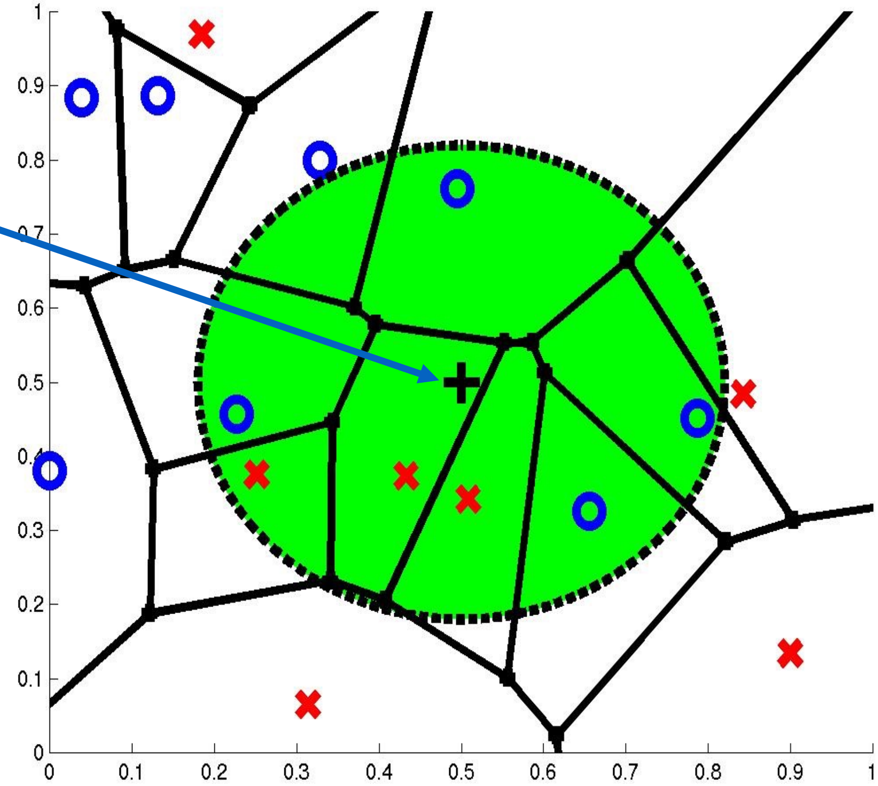


7-Nearest Neighbors

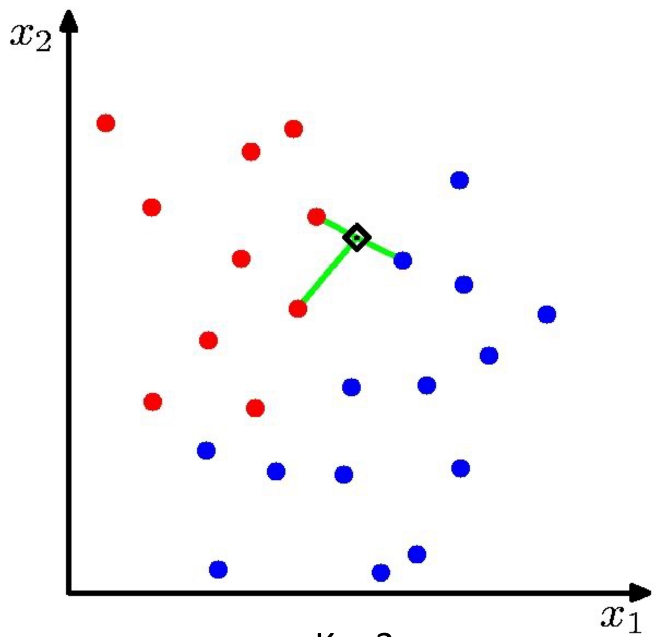
query point x_q

7 nearest neighbors

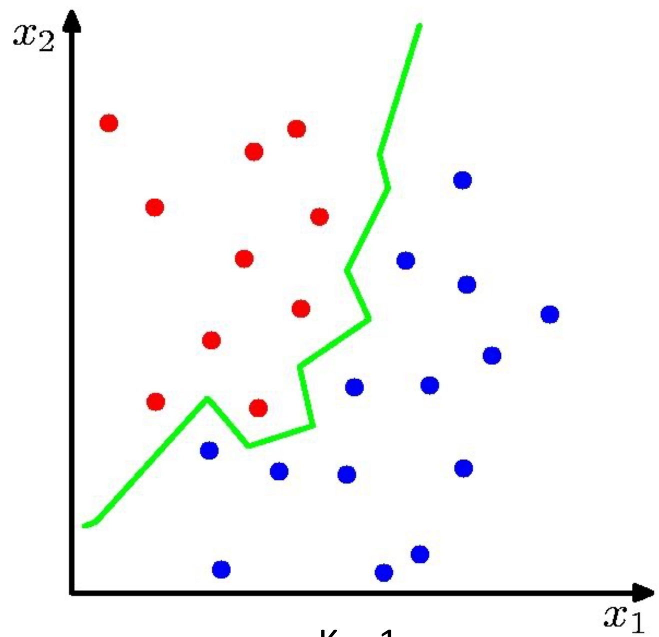
3x,4o



K-Nearest-Neighbours for Classification (2)

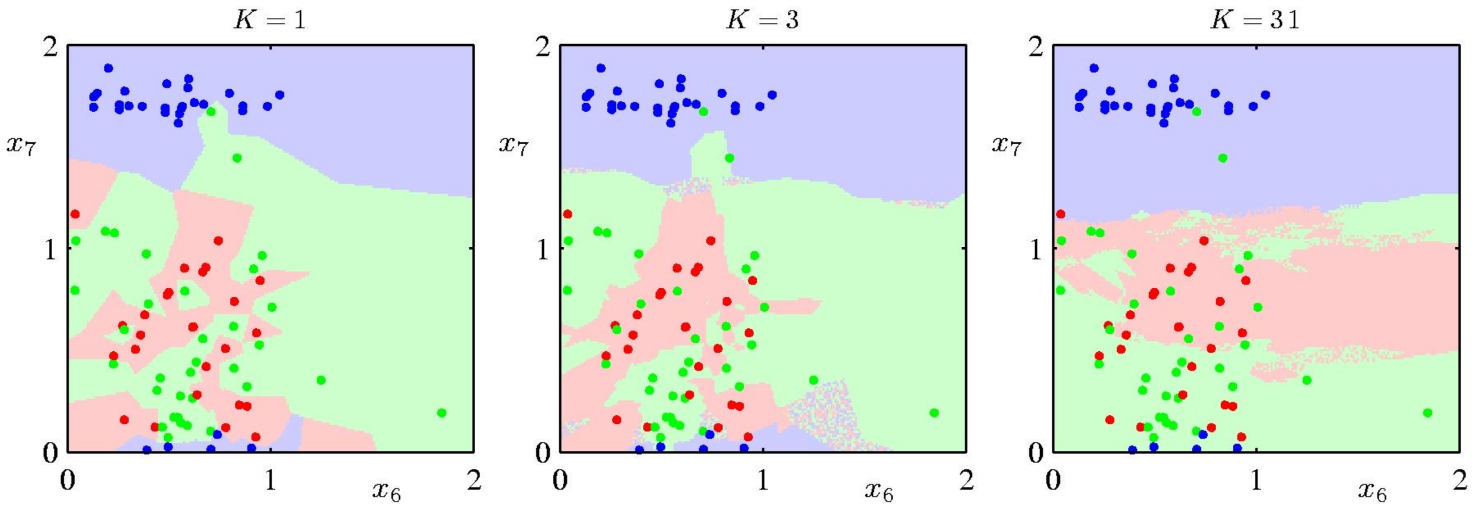


K = 3



K = 1

K-Nearest-Neighbours for Classification



• K acts as a smother

• For $N \to \infty$, the error rate of the 1-nearest-neighbour classifier is never more than twice the optimal error (obtained from the true conditional class distributions).

Distance Weighted

- Refinement to kNN is to weight the contribution of each *k* neighbor according to the distance to the query point x_q
  - Greater weight to closer neighbors
  - For discrete target functions

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

$$w_i = \begin{cases} \dfrac{1}{d(x_q, x_i)^2} & if \quad x_q \neq x_i \\ 1 & else \end{cases}$$

How to determine the good value for *k*?

- Determined experimentally
- Start with *k=1* and use a test set to validate the error rate of the classifier
- Repeat with *k=k+2 (For two classes)*
- Choose the value of *k* for which the error rate is minimum

- Note: *k* should be odd number to avoid ties

Curse of Dimensionality

- Imagine instances described by 20 features (attributes) but only 3 are relevant to target function
- Curse of dimensionality: nearest neighbor is easily misled when instance space is high-dimensional
- Dominated by large number of irrelevant features

Possible solutions

- Stretch j-th axis by weight z_j, where $z_1,\dots,z_n$ chosen to minimize prediction error (weight different features differently)
- Use cross-validation to automatically choose weights $z_1,\dots,z_n$
- Note setting z_j to zero eliminates this dimension altogether (feature subset selection)
- PCA (later)

Disatvantages

- One disadvantage of instance-based approaches is that the cost of classifying new instances can be high
- How can we reduce the classification costs (time)?

- Therefore, techniques for **efficiently indexing** training examples are a significant practical issue in reducing the computation required at query time (High Dimensional Indexing, tree indexing will not work!)
- Compression, reduce the number of representatives (LVQ)
  - Two Examples…

Epsilon similarity

- For a range query vector y from a collection of s vectors

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_s$$

all vectors $\mathbf{x}_i$ that are ϵ-similar according to the distance function d are searched

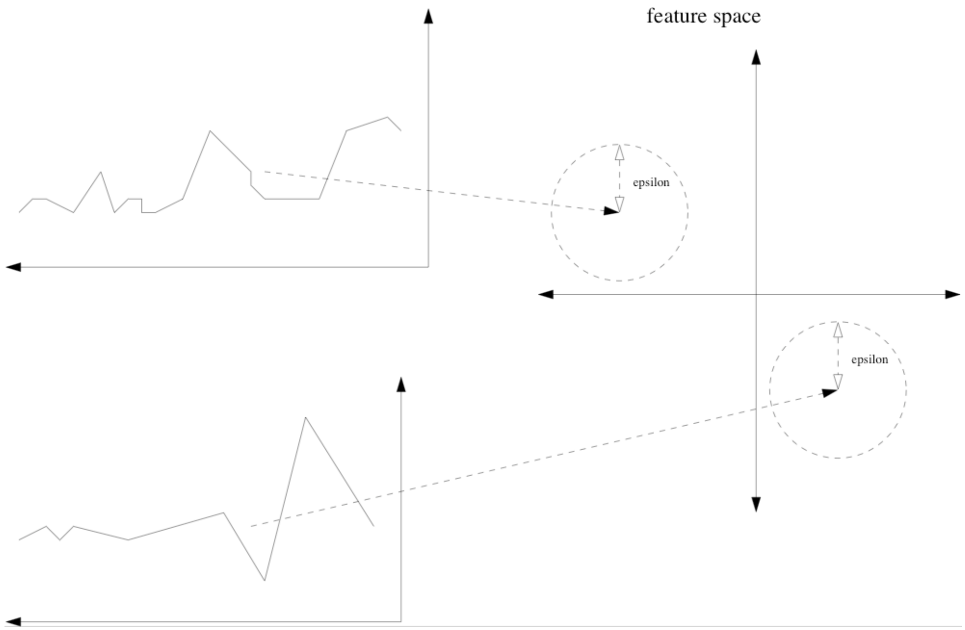
$$d(\mathbf{x}_i, \mathbf{y}) < \epsilon.$$

GEneric Multimedia INdexIng

Christos

Faloutsos

QBIC 1994



- a feature extraction function maps the high dimensional objects into a low dimensional space
- objects that are very dissimilar in the feature space, are also very dissimilar in the original space

Lower bounding lemma

- $$d_{feature}(F(O_1), F(O_2)) \leq d(O_1, O_2)$$

- if distance of similar “objects“ is smaller or equal to ε in original space
- then it is as well smaller or equal ε in the feature space

Range query

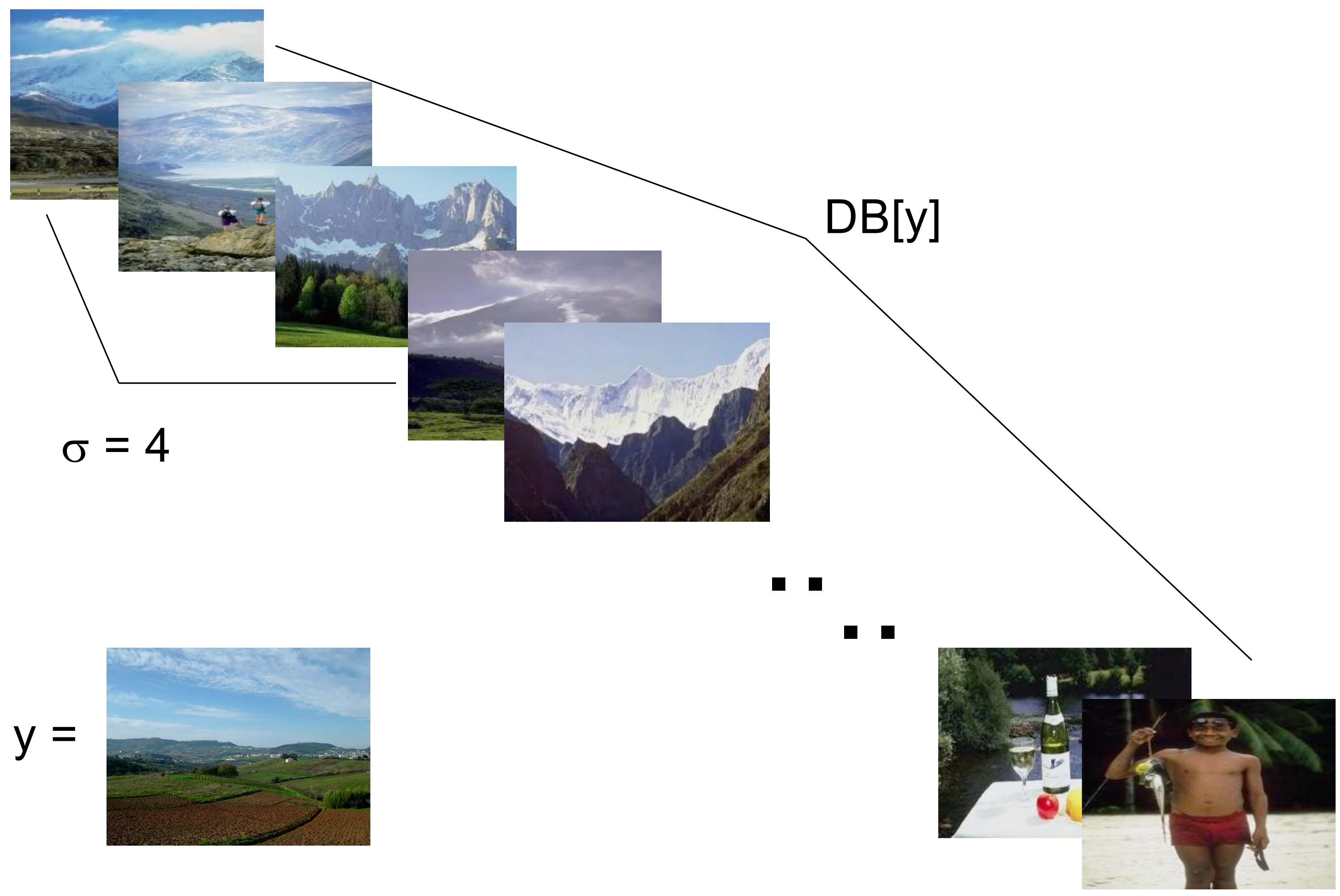
$$\{\vec{x}^{(i)} \in DB | i \in \{1..s\}\}$$

$$d[y]_n := \{d(x^{(i)}, y) \mid \forall n \in \{1..s\} : d[y]_n \leq d[y]_{n+1}\}$$

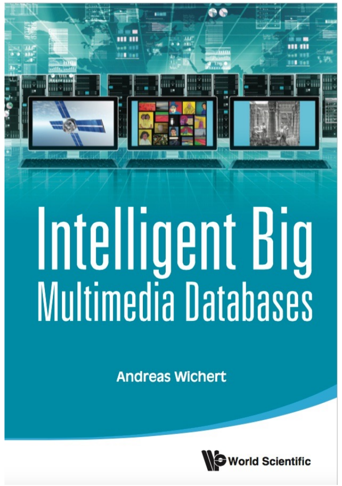
- Range query: search covers all points in the space whose Euclidian distance to the query ***y*** is smaller or equal to ε

$$DB[y]_\varepsilon := \{x^{(i)} \in DB \mid d[y]_n = d(x^{(i)}, y) \leq \varepsilon\}$$

$$\sigma = |DB[y]_\varepsilon|$$


DB[y]
σ = 4
y =

Linear subspace sequence



- Sequence of subspaces with, $V=U_0$ and

$$U_0, U_1, U_2, \ldots, U_n$$

$$U_0 \supset U_1 \supset U_2 \supset \ldots \supset U_n$$

$$dim(U_0) > dim(U_1) > dim(U_2) \ldots > dim(U_n)$$

- Lower bounding lemma,

$$d(U_n(x_1), U_n(x_2)) \leq \ldots \leq d(U_1(x_1), U_1(x_2)) \leq d(U_0(x_1), U_0(x_2))$$

- Example,

$$\mathbf{R}^{\mathbf{m}} \supset \mathbf{R}^{\mathbf{m-1}} \supset \mathbf{R}^{\mathbf{m-2}} \supset \ldots \supset \mathbf{R}^{\mathbf{1}}$$

DB in subspace

$$\{U_k(\vec{x})^{(i)} \in U_k(DB) | i \in \{1..s\}\}$$

$$d[U_k(y)]_n := \{d(U_k(x^{(i)}), U_k(y)) \mid \forall n \in \{1..s\} : d[U_k(y)]_n \leq d[U_k(y)]_{n+1}\}$$

$$\mathrm{U}_k(DB[y])_\varepsilon := \{U_k(x)_n^{(i)} \in U_k(DB) \mid d[U_k(y)]_n = d(U_k(x)^{(i)}, U_k(y)) \leq \varepsilon$$

$$U_k(\sigma) = |U_k(DB[y]_\varepsilon)|$$

$$U_0(\sigma) < U_1(\sigma) < U_2(\sigma) < \ldots < U_{(n)}(\sigma) < s$$

Computing costs

$$U_1(\sigma) \cdot m + U_2(\sigma) \cdot dim(U_1) + \ldots + s \cdot dim(U_n) =$$

$$U_1(\sigma) \cdot dim(U_0) + U_2(\sigma) \cdot dim(U_1) + \ldots + s \cdot dim(U_n) =$$

$$= \sum_{i=1}^{n} U_i(\sigma) \cdot dim(U_{(i-1)}) + s \cdot dim(U_n)$$

Orthogonal projection

- Corresponds to the mean value of the projected points
- Distance *d* between projected points in $\boldsymbol{R^m}$ corresponds to the distance d_u in the orthogonal subspace ***U*** multiplied by a constant *c*

$$c = \sqrt{\frac{m}{f}}$$

Orthogonal projection

P: $\boldsymbol{R}^m \rightarrow \boldsymbol{U}$

- $w^{(1)}, w^{(2)}, .., w^{(m)}$ Orthonormalbasis of $\boldsymbol{R}^m$
- $w^{(1)}, w^{(2)}, .., w^{(f)}$ Orthonormalbasis of $\boldsymbol{U}$

(Gram-Schmidt orthogonalization process)

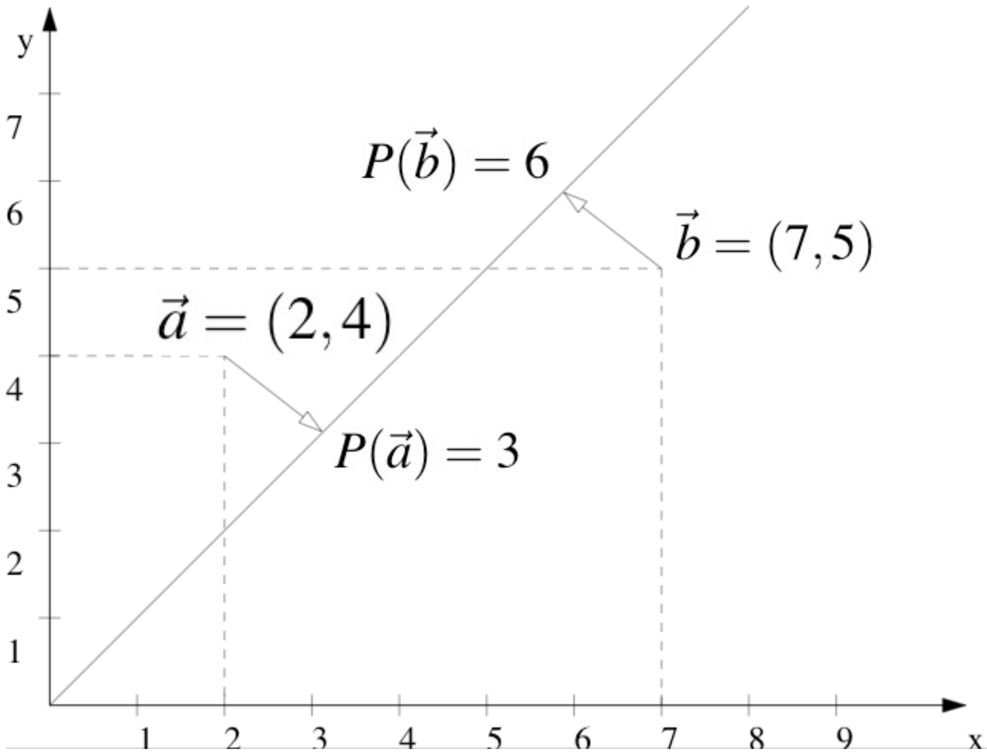
$$\vec{x} = \sum_{i=1}^{f} < \vec{x}, w^{(i)} > \cdot w^{(i)} + \sum_{i=f+1}^{m} < \vec{x}, w^{(i)} > \cdot w^{(i)}$$

$$P(\vec{x}) = \sum_{i=1}^{f} < \vec{x}, w^{(i)} > \cdot w^{(i)} \qquad O(\vec{x})^{\perp} = \sum_{i=f+1}^{m} < \vec{x}, w^{(i)} > \cdot w^{(i)}$$

- Because of the Pythagorean theorem lower bound lemma follows

$$||\vec{x}||^2 = ||P(\vec{x})||^2 + ||O(\vec{x})^{\perp}||^2 \quad \Rightarrow \quad ||\vec{x}|| \geq ||P(\vec{x})||$$

$$U=\{(x_1,x_2) \in R^2 \mid x_1=x_2\}$$



$$d_u(P(\vec{a}),P(\vec{b})) = \sqrt{|6-3|^2}$$

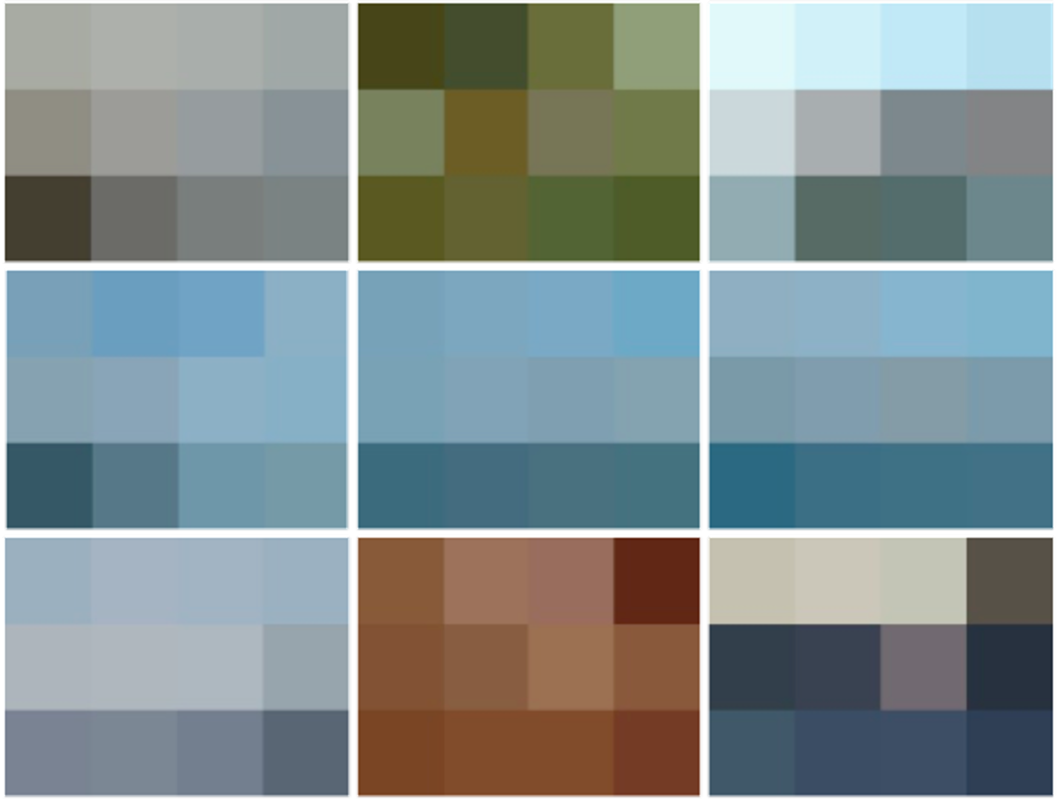
$$c = \sqrt{\frac{m}{f}} \qquad c = \sqrt{2}$$

$$d(P(\vec{a}),P(\vec{b})) = 3 \cdot \sqrt{2} \leq d(\vec{a},\vec{b}) = \sqrt{26}$$

- In this case, the lower bounding lemma is extended:
- let O_1 and O_2 be two objects; *F()*, the mapping of objects into *f* dimensional subspace *U*
- *F()* should satisfy the following formula for all objects, where *d* is a distance function in the space *V* and d_U in the subspace *U*

$$d_U(F(O_1), F(O_2)) \leq d(F(O_1), F(O_2)) \leq d(O_1, O_2)$$

U_3 (4*3)

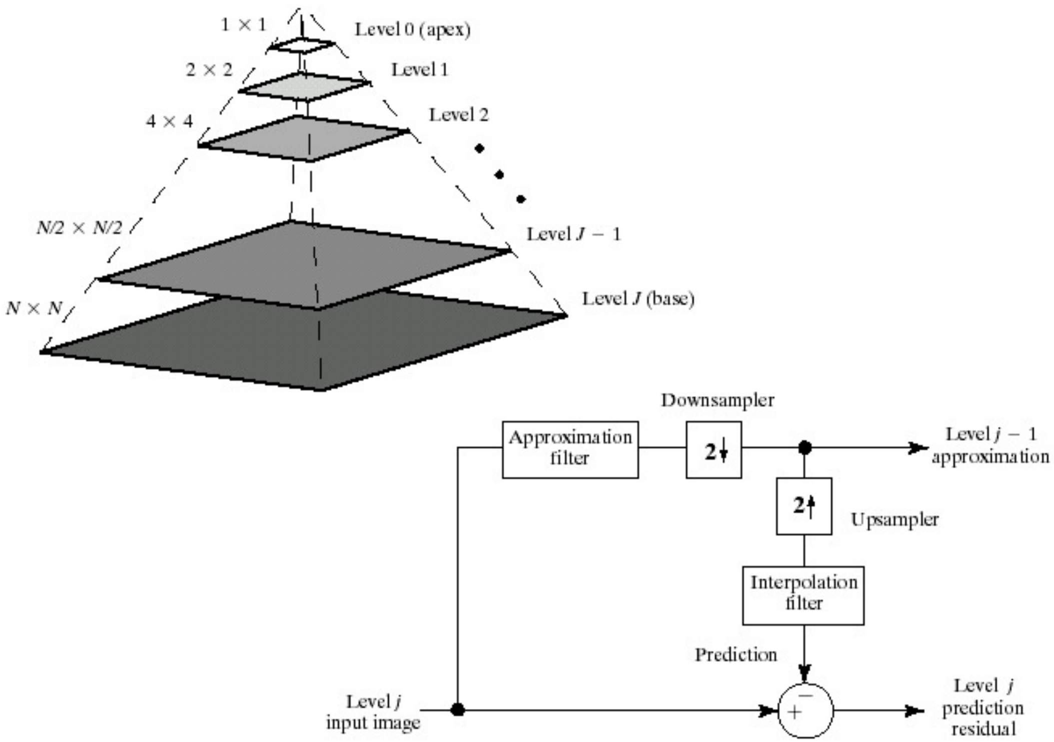


U_2 (8*6)

U_1 (40*30)

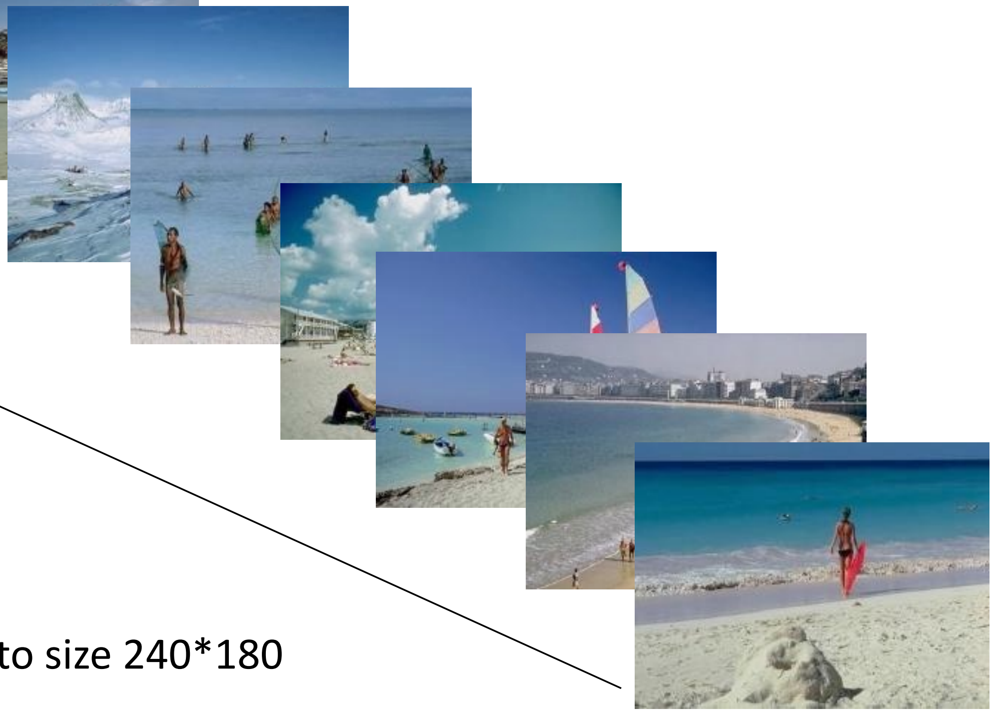
U_0 (240*180)

Image Pyramid



a
b

FIGURE 7.2 (a) A pyramidal image structure and (b) system block diagram for creating it.



y=

$$\sigma = 10$$

$$U_0(DB)_{22540}$$

- 15.7 less complex
- (would be ≈ 20 sec)
  - 1000 images, scaled to size 240*180
  - (DB[y] ≈ 5.3 min)

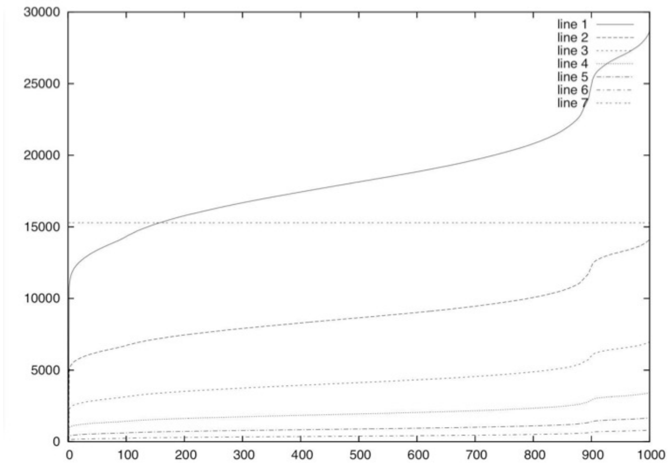
Hierarchy of subspaces

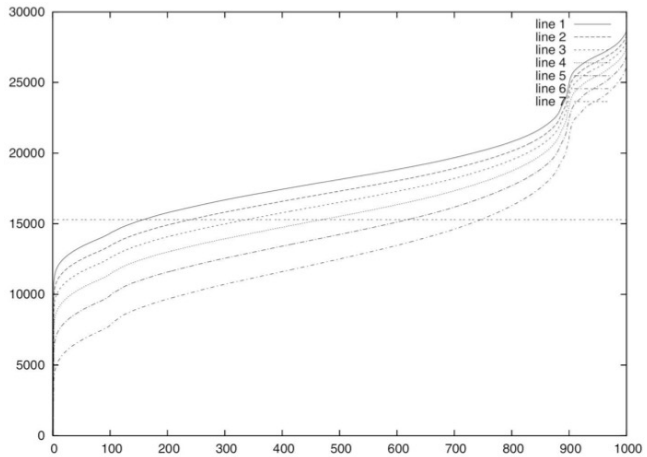
$$U_0 \supset U_1 \supset U_2 \supset U_3$$

- The distance between objects $d=d_{U0}$ in the space U_0 can be obtained from the distance d_{Uk} between objects in the orthogonal subspace U_k by multiplying the distance d_{Uk} by a constant

$$c_k = \sqrt{\frac{dim(U_0)}{dim(U_k)}}$$

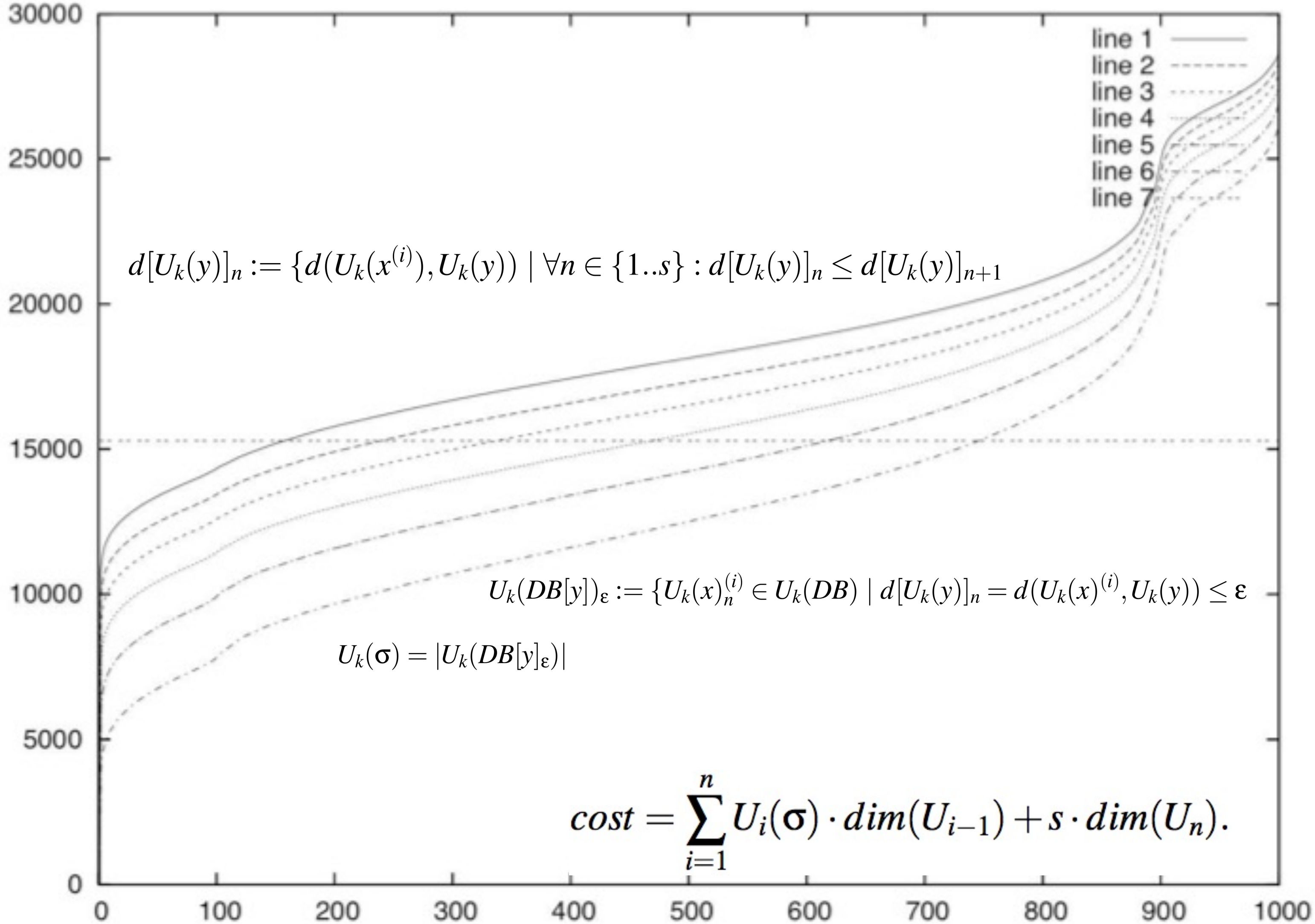
Euclidian distance for a query **y** to the elements of DB





- Distance in U_k Distance in U_0

$$d_{U_k}(F_{0,k}(O_1), F_{0,k}(O_2)) \leq d_{U_0}(F_{0,k}(O_1), F_{0,k}(O_2))$$



$$d[U_k(y)]_n := \{d(U_k(x^{(i)}), U_k(y)) \mid \forall n \in \{1..s\} : d[U_k(y)]_n \leq d[U_k(y)]_{n+1}$$

$$U_k(DB[y])_\varepsilon := \{U_k(x)_n^{(i)} \in U_k(DB) \mid d[U_k(y)]_n = d(U_k(x)^{(i)}, U_k(y)) \leq \varepsilon$$

$$U_k(\sigma) = |U_k(DB[y]_\varepsilon)|$$

$$cost = \sum_{i=1}^{n} U_i(\sigma) \cdot dim(U_{i-1}) + s \cdot dim(U_n).$$

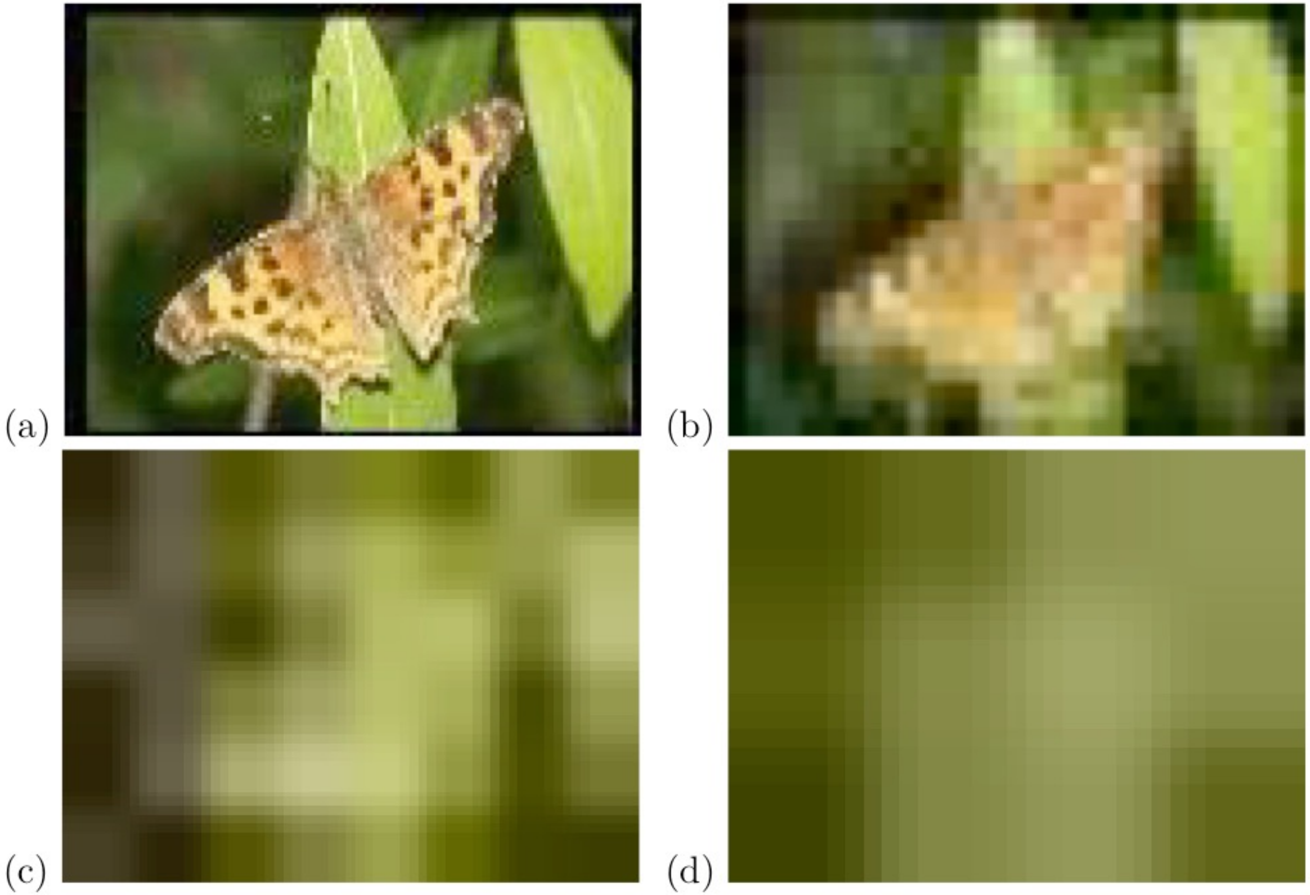
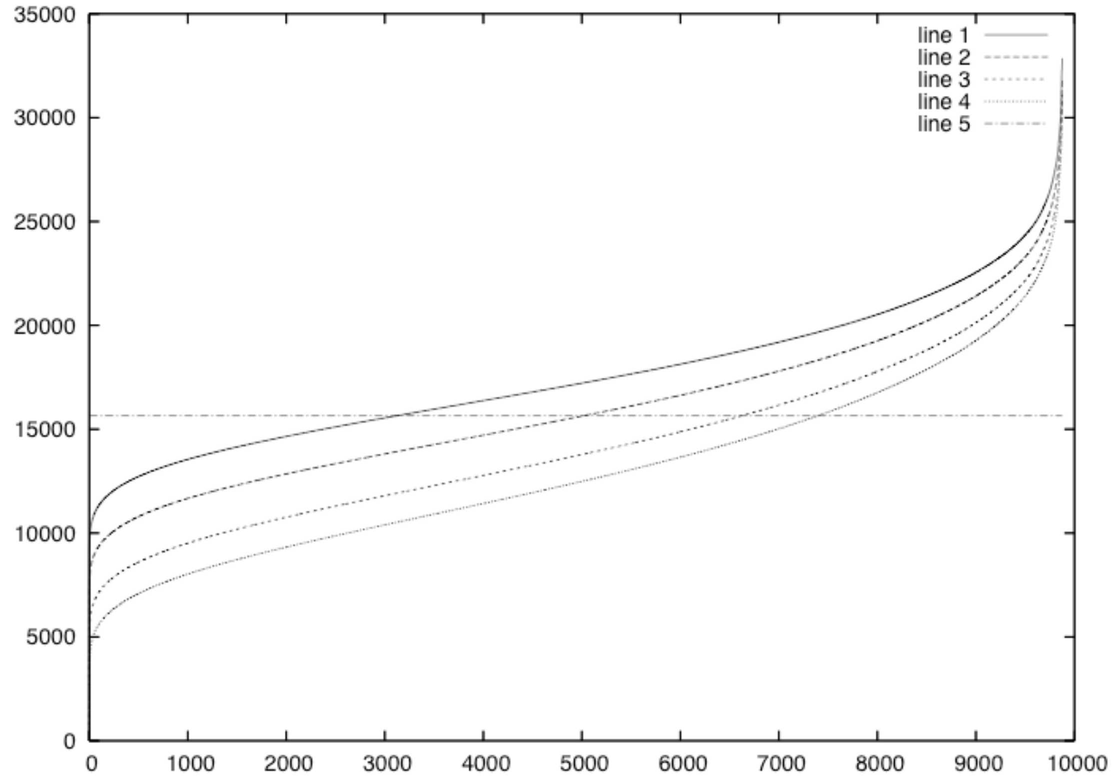
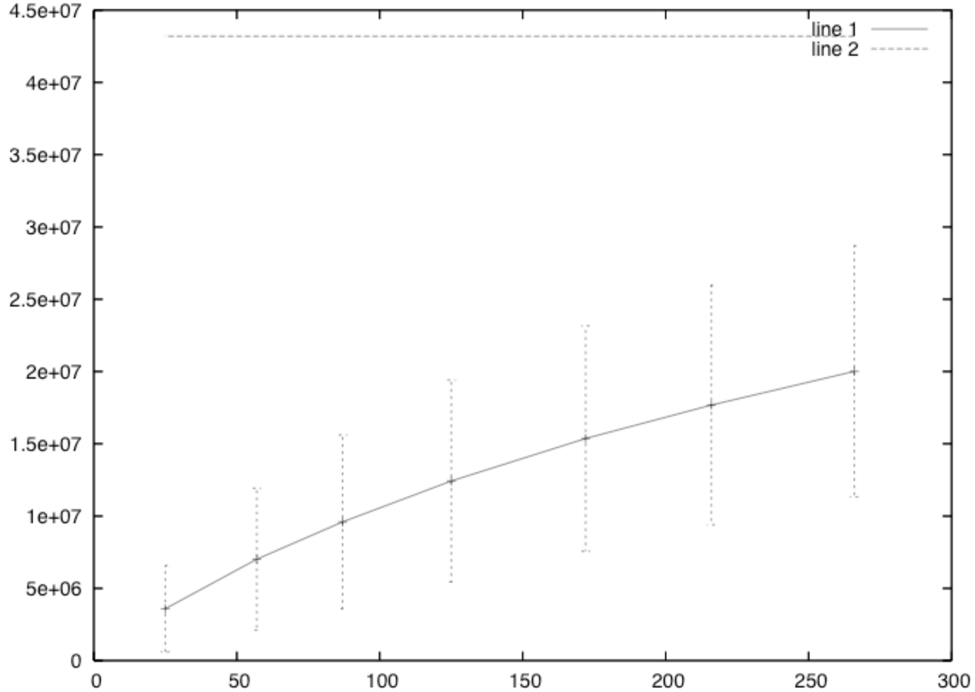


Figure 6: (a) Image of an butterfly, with the size 128×96. (b) Image of the butterfly, resolution 32×24. (c) The image of the butterfly resolution 8×6. (d) The image of the butterfly, resolution 4×3.



Database consists of ***9.876*** web-crawled color images

Mean Euclidian distance for a query ***y*** to the elements of DB



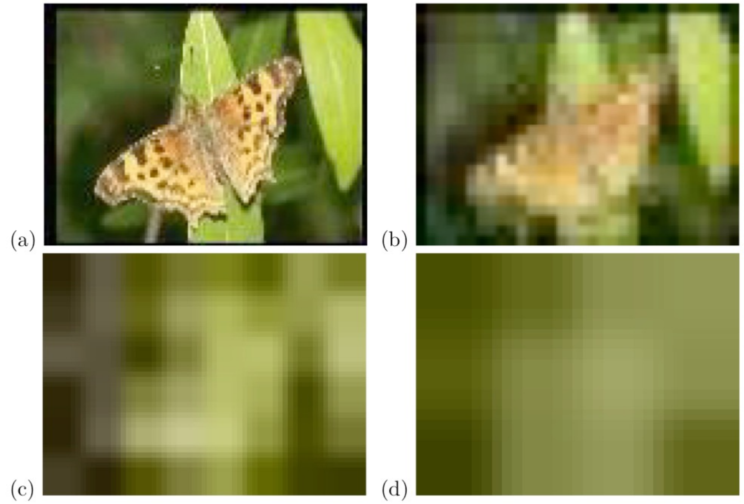


Figure 6: (a) Image of an butterfly, with the size 128×96. (b) Image of the butterfly, resolution 32 × 24. (c) The image of the butterfly resolution 8 × 6. (d) The image of the butterfly, resolution 4 × 3.

- Mean computing costs using the hierarchical subspace method
- Error bars indicate the standard deviation
- The x-axis indicates the number of the most similar images which are retrieved and the y-axis, the computing cost
- Database consists of ***9.876*** web-crawled color images

Logarithmic Cost

The cost are

logarithmic in dimension ***dim*** and the number of points ***N***

log(N)+log(dim)

Andreas Wichert and Catarina Moreira, ***Projection Based Operators in lp space for Exact Similarity Search***, Fundamenta Informaticae, Annales Societatis Mathematicae Polonae, 136(4): 461-474, 2015

doi:10.3233/FI-2015-1166

Locally Weighted Regression

- We can extend this method from classification to regression
- Instead of combining the discrete predictions of k-neighbours we have to combine continuous predictions

- Averaging
- Local linear regression
  - K-NN linear regression fits the best line between the neighbors
    A linear regression problem has to be solved for each query (least squares regression)
- Local weighted regression

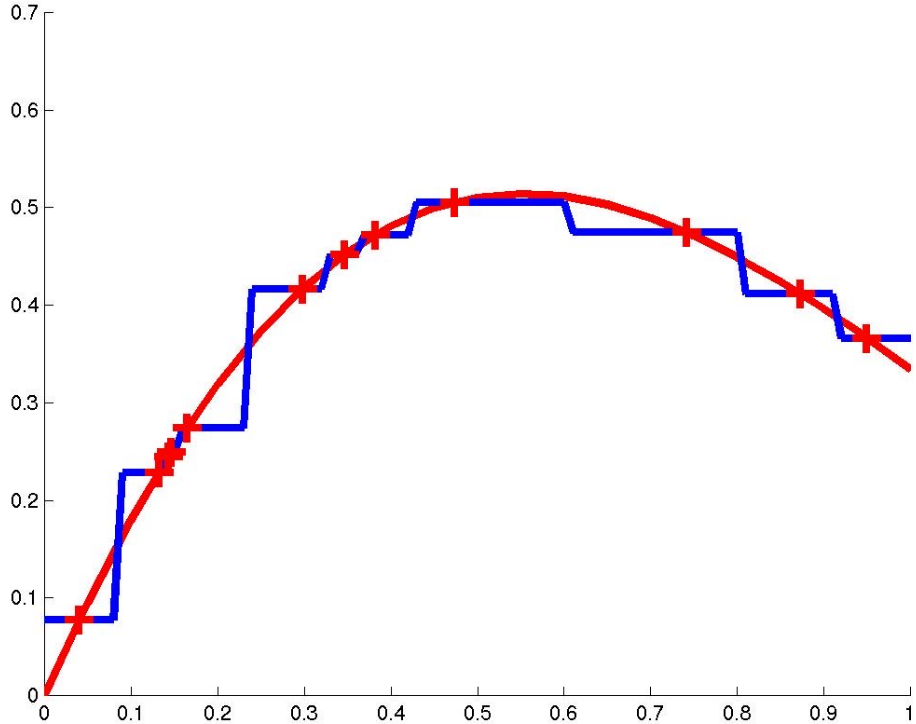
Continuous-valued target functions

- kNN approximating continous-valued target functions
- Calculate the mean value of the *k* nearest training examples rather than calculate their most common value

$$f : \Re^d \rightarrow \Re \qquad \hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

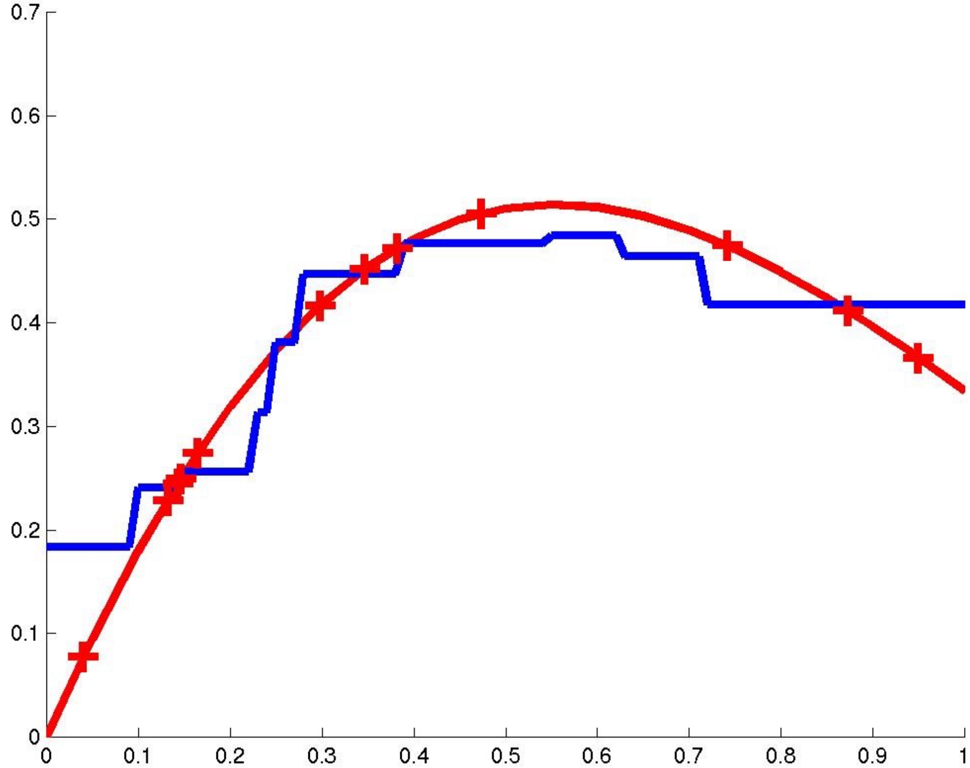
Nearest Neighbor (continuous)

1-nearest neighbor



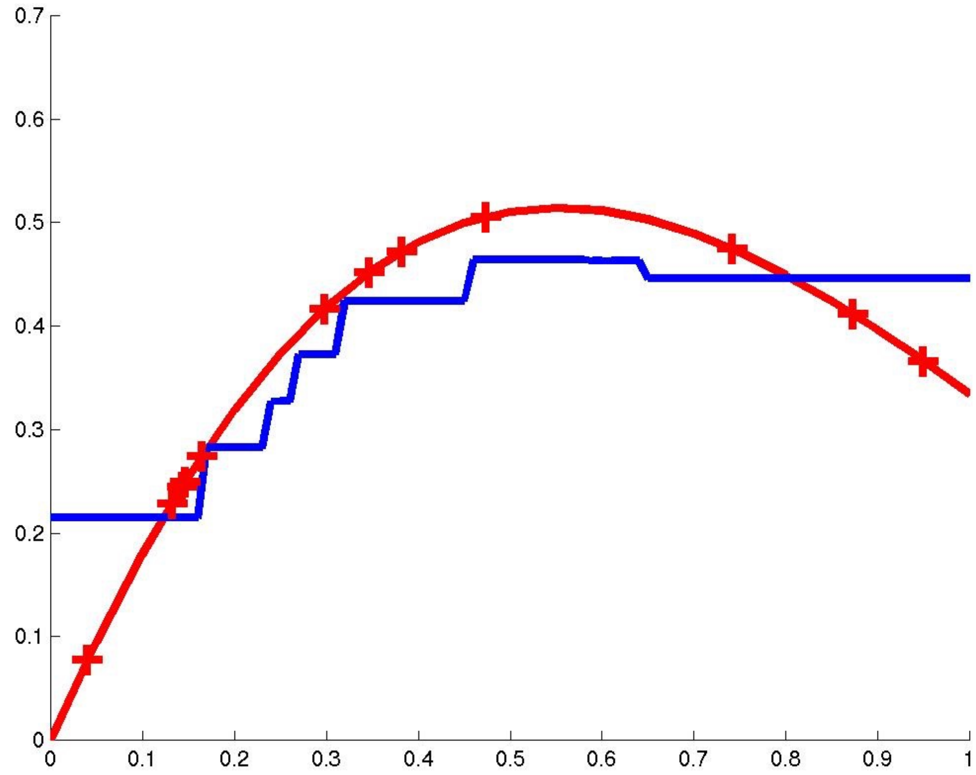
Nearest Neighbor (continuous)

3-nearest neighbor



Nearest Neighbor (continuous)

5-nearest neighbor



Distance Weighted

- For real valued functions

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

$$w_i = \begin{cases} \dfrac{1}{d(x_q, x_i)^2} & if \quad x_q \neq x_i \\ 1 & else \end{cases}$$

Weighted Linear Regression

- How shall we modify this procedure to derive a local approximation rather than a global one?
- The simple way is to redefine the error criterion *E* to emphasize fitting the local training examples
- Minimize the squared error over just the k nearest neighbors

Linear Unit (Before)

- Simpler linear unit with a linear activation function

$$o = \sum_{j=0}^{D} w_j \cdot x_j = net_0.$$

- We can define the training error for a training data set D_t of N elements with

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} (t_k - o_k)^2 = \frac{1}{2} \cdot \sum_{k=1}^{N} \left(t_k - \sum_{j=0}^{D} w_j \cdot x_{k,j} \right)^2$$

One starts the process at some randomly chosen point $\mathbf{w}^{initial}$ and modifies the weights if required with the learning rule

$$w_j^{new} = w_j^{old} + \Delta w_j$$

and

$$\Delta w_j = -\eta \cdot \frac{\partial E}{\partial w_j}.$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{2} \cdot \sum_{k=1}^{N} (t_k - o_k)^2 = \frac{1}{2} \cdot \sum_{k=1}^{N} \frac{\partial}{\partial w_j} (t_k - o_k)^2$$

The update rule for gradient decent is given by

$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

Now we have the basis functions for dimension $D = 1$ with $\phi_0(x) = 1$

$$\hat{f}(x_k) = \sum_{j=0}^{M-1} w_j \cdot \phi_j(x_k) = \mathbf{w}^T \Phi(x_k)$$

$$\hat{f}(x_k) = w_0 + w_1 \cdot \phi_1(x_k) + w_2 \cdot \phi_2(x_k) + \cdots + w_{M-1} \cdot \phi_{M-1}(x_k)$$

$$\hat{f}(x_k) = o_k, \quad f(x_k) = t_k$$

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} (f(x_k) - \hat{f}(x_k))^2$$

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} (f(x_k) - \hat{f}(x_k))^2$$

The update rule for gradient decent is given by

$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (f(x_k) - \hat{f}(x_k)) \cdot \phi_j(x_k)$$

Minimize the squared error over just the k nearest neighbours:

$$E(\mathbf{w}) = \sum_{x_i \ is \ kNN \ of \ x_q} (f(x_k) - \hat{f}(x_k))^2$$

$$\Delta w_j = \eta \cdot \sum_{x_i \ is \ kNN \ of \ x_q} (f(x_k) - \hat{f}(x_k)) \cdot \phi_j(x_k)$$

Minimize the squared error over the entires D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q

$$E(\mathbf{w}) = \sum_{k=1}^{N} f(x_k) - \hat{f}(x_k))^2 \cdot K(d(x_q, x_k))$$

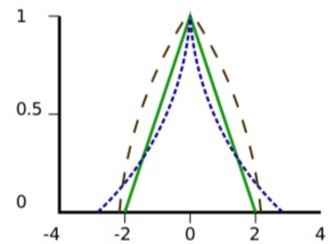
$$E(\mathbf{w}) = \sum_{k=1}^{N} f(x_k) - \hat{f}(x_k))^2 \cdot K(d(x_q, x_k))$$

Kernel K function is the function of distance that is used to determine the weight of each training example.

Local weighted regression uses a function K to weight the contribution of the neighbours depending on the distance, this is done using a kernel function

Kernel functions have a width parameter that determines the decay of the weight (it has to be adjusted)

Weighted Linear Regression

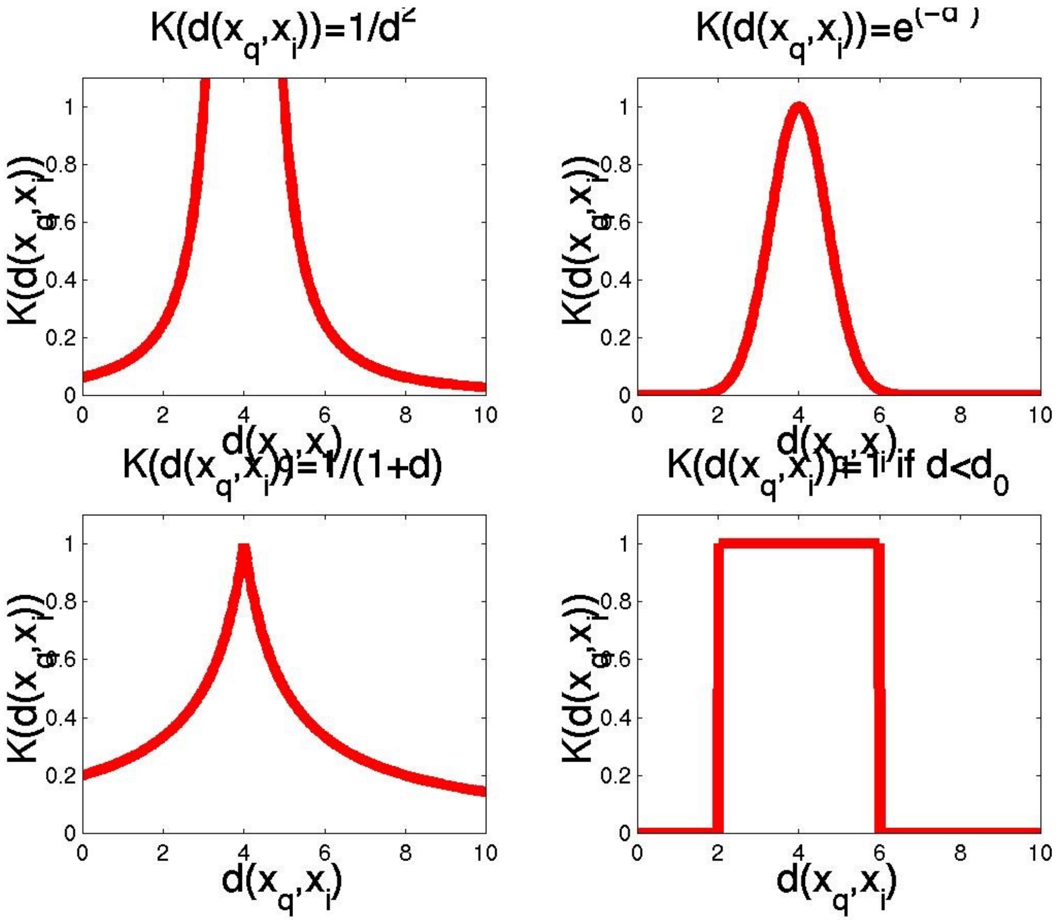


- Kernel functions have a width parameter that determines the decay of the weight (it has to be adjusted)
- A weighted linear regression problem has to be solved for each query (gradient descent search)

- Combine both approaches

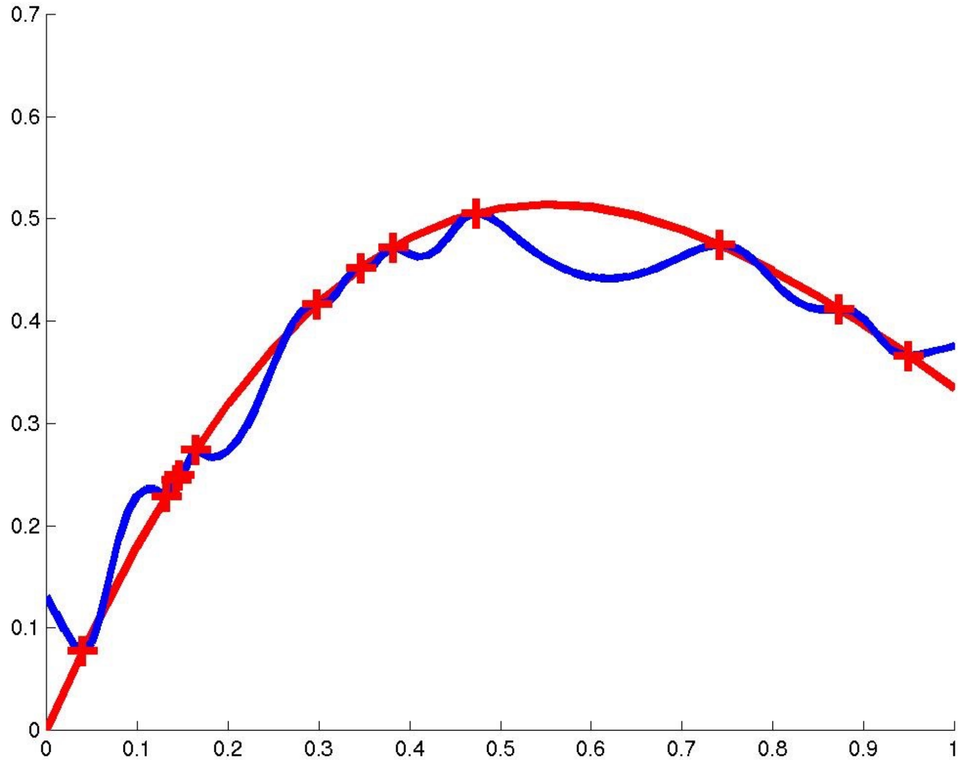
$$E(\mathbf{w}) = \sum_{x_i \ is \ kNN \ of \ x_q} f(x_k) - \hat{f}(x_k))^2 \cdot K(d(x_q, x_k))$$

Kernel Functions



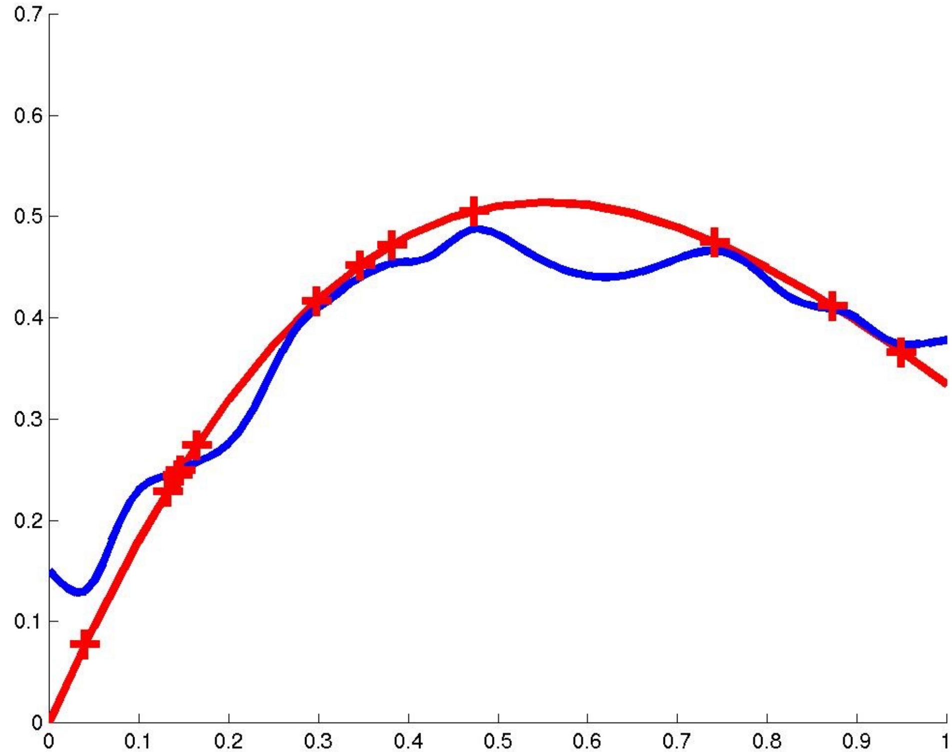
Distance Weighted NN

$K(d(x_q, x_i)) = 1/\ d(x_q, x_i)^2$



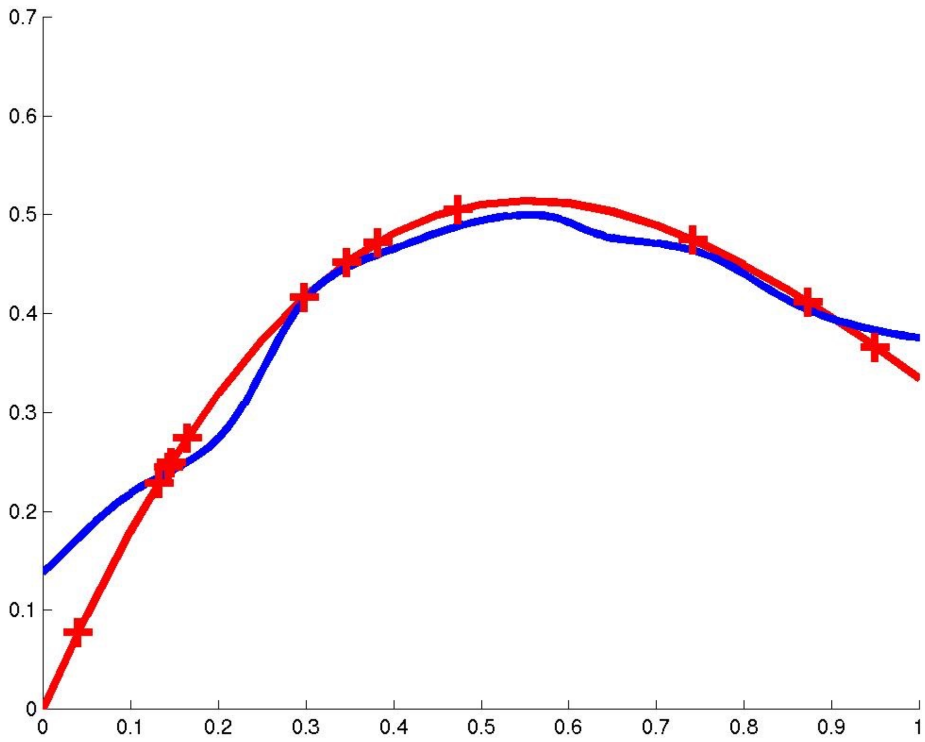
Distance Weighted NN

$K(d(x_q,x_i)) = 1/(d_0+d(x_q,x_i))^2$



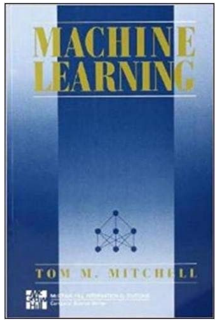
Distance Weighted NN

$K(d(x_q,x_i)) = \exp(-(d(x_q,x_i)/\sigma_0)^2)$

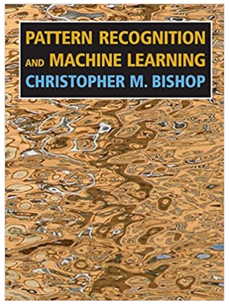


- Can fit low dimensional, very complex, functions very accurately
- Training, adding new data, is almost free
- Doesn’t forget old training data
- Lazy: wait for query before generalizings
- Lazy learner can create local approximations

Literature

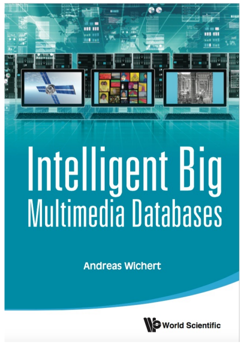


- Tom M. Mitchell, Machine Learning, McGraw-Hill; 1st edition (October 1, 1997)
  - Chapter 8



- Christopher M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer 2006
  - Section 2,5

Literature (Additional)



- Intelligent Big Multimedia Databases, A. Wichert, World Scientific, 2015
  - *Chapter 6: Low Dimensional Indexing*
  - *Chapter 7: Approximative Indexing*
  - *Chapter 8: High Dimensional Indexing*