Template matching

Each class is represented by one template (prototype).

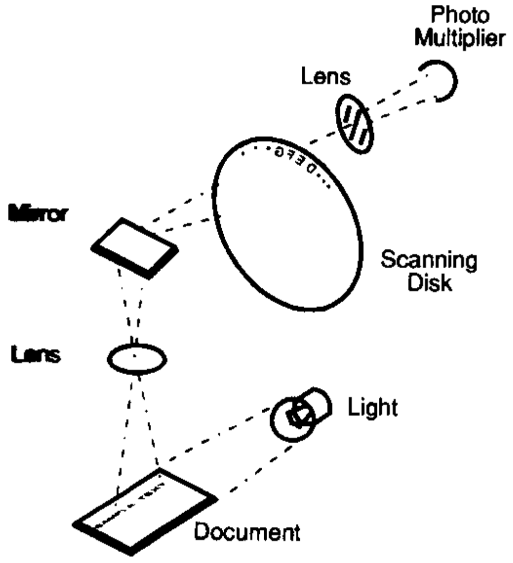
It is assumed that classified objects are templates with addition of noise.

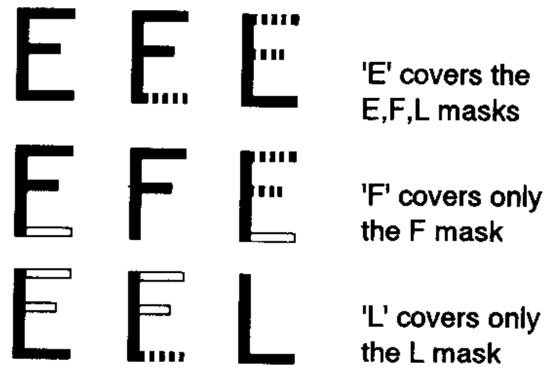
We compare object to be classified with each template seeking:

- Maximum correlation
- Minimum error

Historically it was the first approach to pattern classification.

There were fantastic hardware implementations.




'E' covers the E,F,L masks
'F' covers only the F mask
'L' covers only the L mask

Minimum-distance classification

Let us replace *template* by its feature vector. Let the index of the template be class label.

Instead of templates themselves we have for each class its feature vector: t^1, t^2 ... t^L (where L is number of classes in our problem).

When we want to classify an object we should first compute its feature vector. Let us denote it by *x*.

In the next step we compute L distances between *x* and each template t^i: $d^i = d(x, t^i)$

The final step is to find minimum distance between x and class templates:

$$c = \arg\min_i d^i$$

c is classification result (class label).

Distance function (1)

Distance function must satisfy 4 axioms:

1. $d(x,y) = 0 \equiv x = y$ (reflexivity)
2. $d(x,y) \geq 0$ (distances are non-negative)
3. $d(x,y) = d(y,x)$ (symmetry)
4. $d(x,y) + d(y,z) \geq d(x,z)$ (triangle inequality)

Distance functions (2)

Hamming distance

$$d_H(x,y)=\sum_{i=1}^{d} x_i \oplus y_i$$

City-block distance (Manhattan

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

Euclidean distance

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

Minkowski metric

$$d_m(x,y)=\left[\sum_{i=1}^{d} \left| x_i - y_i \right|^s \right]^{1/s}$$

Levenshtein (edit) distance

Nearest neighbour classification

Training set

$$T = \{ \langle x^k, i^k \rangle, k=1,2, \ldots N \}$$

x^k *k^{th}* object feature vector,
i^k *k^{th}* object class label

N – size of the training set

Nearest neighbour of x:

$$x^m \in T: \min d(x, x^k) = d(x, x^m) \text{ for } k=1,2, \ldots N$$

We decide, that x belongs to the same class as i^m - its nearest neighbour.

This classifier is abbreviated 1-NN, because only 1 nearest neighbour decides about the classification result.

1-NN Properties

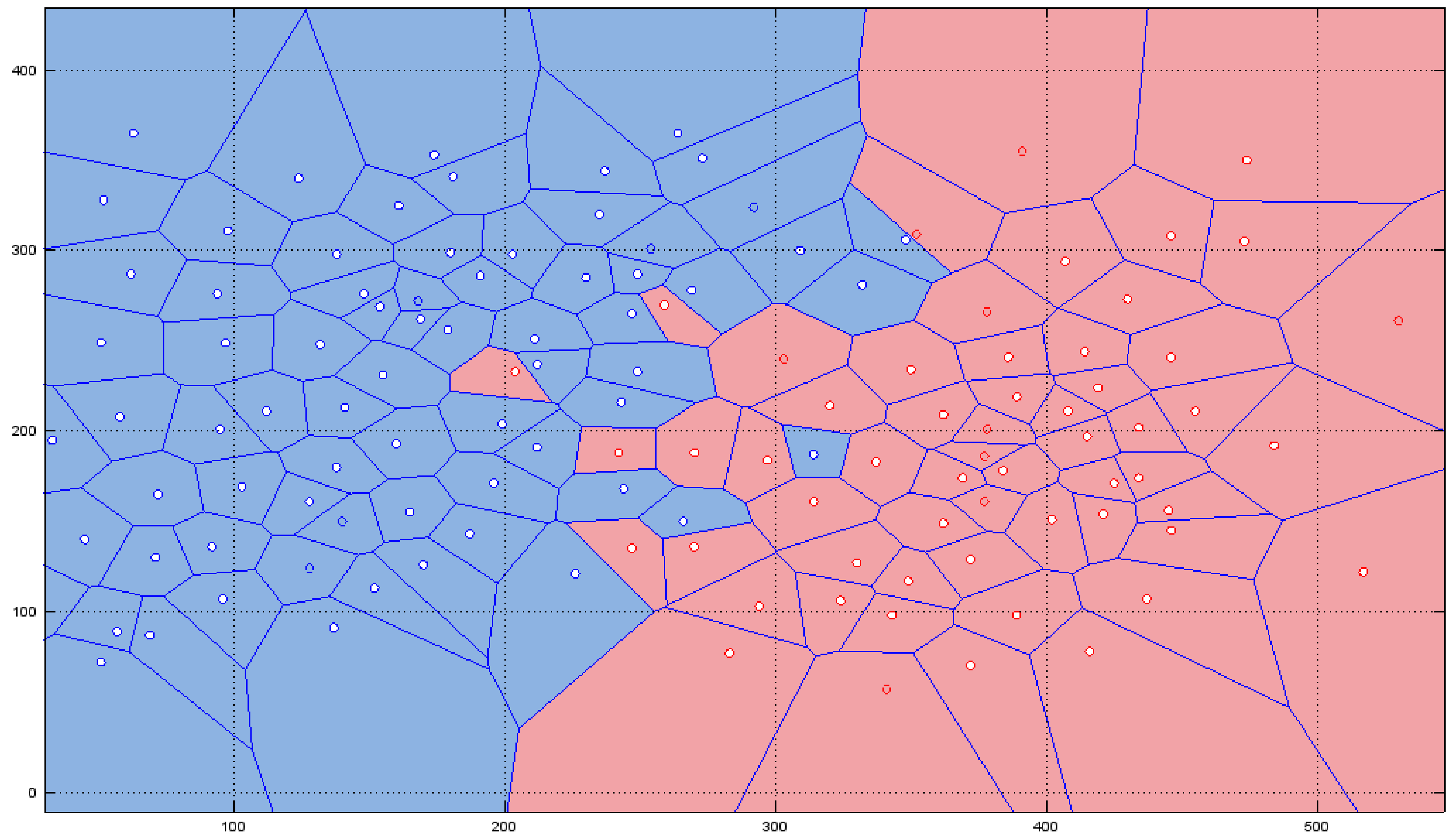
Advantages of 1-NN classification:

- Direct use of the training set
- Low probability of error
- Simple implementation

1-NN classification shortcomings:

- Large computational complexity for large N
- Sensitivity to misclassified samples
- "random" behaviour at class boundaries

Voronoi diagram



k-NN classification

Majority voting of *k* nearest neighbours.

Set of *k* neighbours:

$$T^k = \{ <x^\nu, i^\nu>, \nu=1,2, \dots k \}$$

Subset of T^k containing elements of class i:

$$T^{k,i} = \{ <x^\nu, i^\nu>, \nu=1,2, \dots k \wedge i^\nu=i \}$$

Membership function:

$$F^i(x) = \# T^{k,i} \quad i=1, 2, \dots L$$

Decision: x belongs to class i^m where $F^{im}(x) = \max F^i(x)$

k selection for k-NN classification

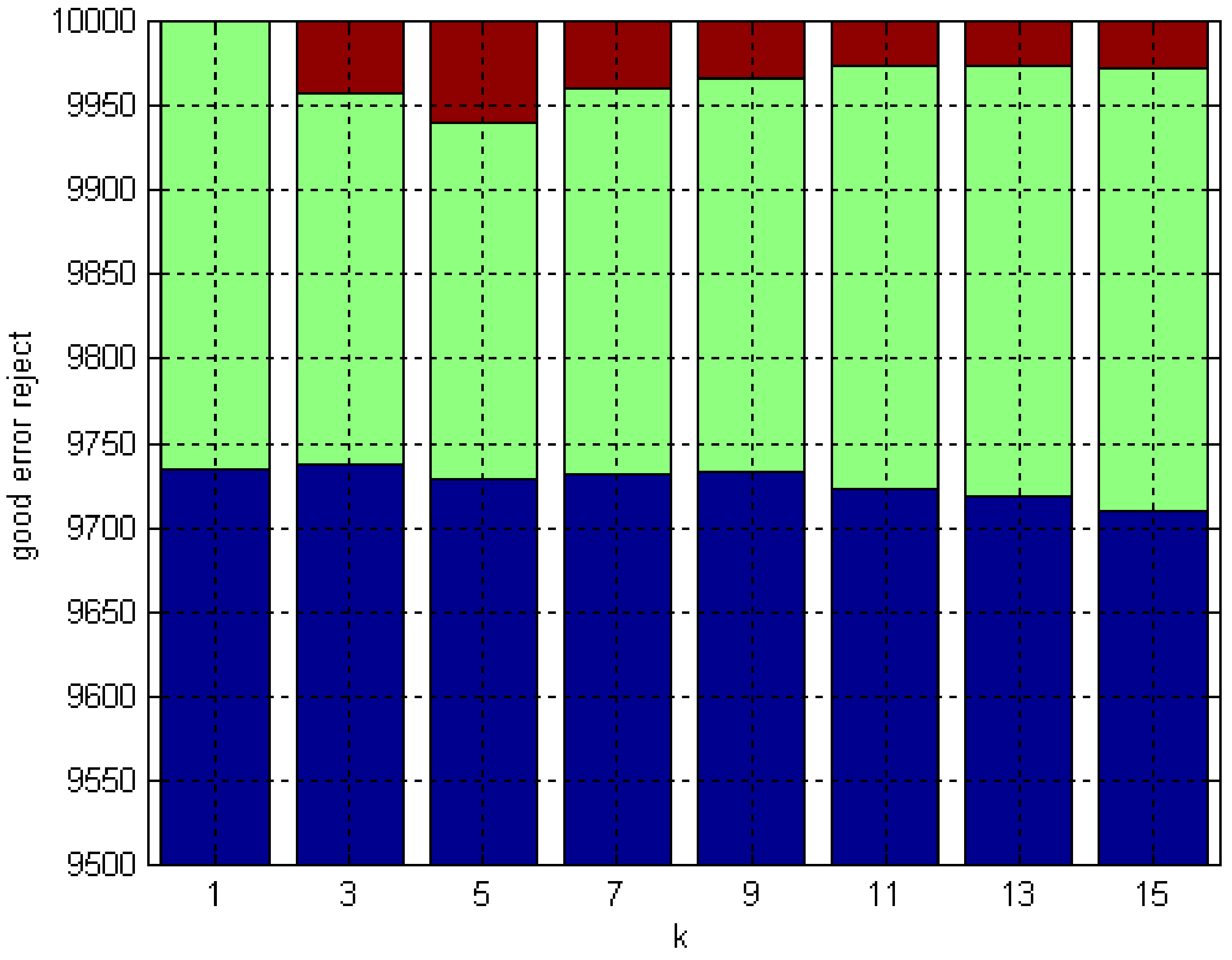
We have two contradictory requirements for k:

- large, to minimize probability of misclassification,
- small - with relation to training set size - to achieve true class boundaries.

For a fixed size training set we should experiment with different k values and select the value giving best classification results on test set.

Generally, we will observe the shift of the errors into the reject decision; it is desirable when the reject option has lower cost than error in classification. Next figure shows the classification coefficients for different *k*s. (Handwritten digits were classified in 40 dimensions using Euclidean distance).

What *k* value should we select?



Data preprocessing

In nearest neighbor classifier it is evident that all features should have the same order of magnitude – but it’s easy to forget it.

The first step in designing classifier (any classifier! not just NN) is learning the data. The goal is twofold:

1. To detect outliers (if any)
2. To make decision if normalization should be performed or not

Usually some simple checking will do:

1. Compute mean and variance of features
2. Compare mean and median of features
3. Check minimum and maximum features’ values
4. Plotting histogram of individual features

Data normalization

In many cases it is enough to scale the input data. We divide feature values in most cases by:

- Square root of variance
- Total feature values range (max – min)

If you normalize the training set you have to normalize in exactly the same way data to be classified.

NN classifier quality assessment

Cross-validation procedure:

- Data set is partitioned into K subsets
- Each of the K subsets is used as the testing set, while remaining subsets form the training set
- The estimate of our NN classifier recognition (or error) coefficient is the mean value from our K tests.

Stratification – we should pay attention to number of samples belonging to different classes in each of K subsets.

Leave-one-out method is the “extreme” cross-validation: the test set contains just one sample!

Projection method for finding NN

Friedman et al. (1975)

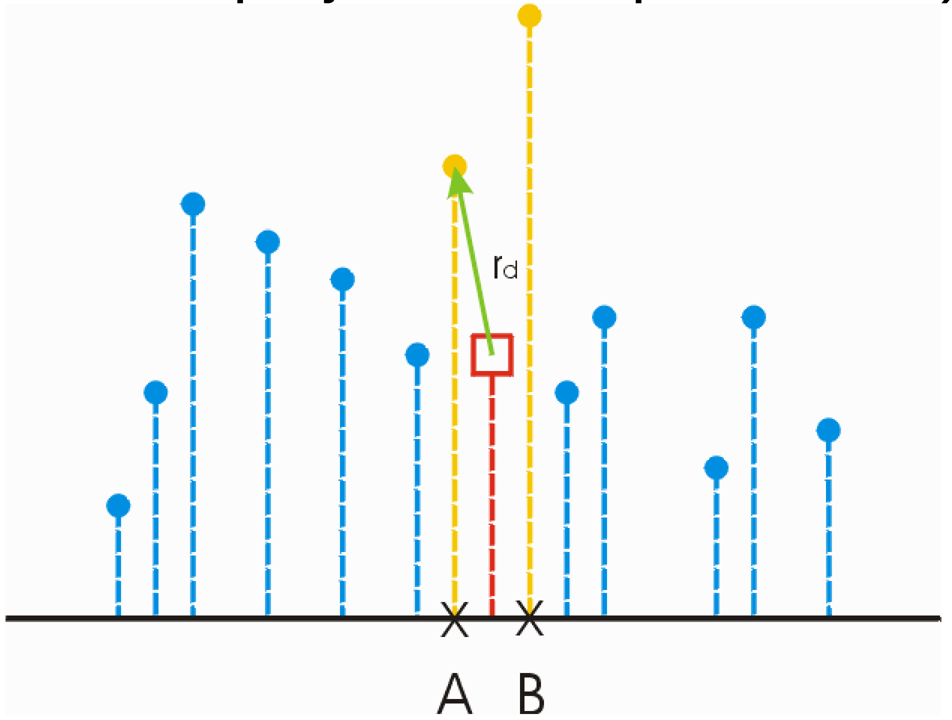
Use projection of each sample on one axis to reduce number of NN candidates.

Preprocessing (2D case):

1. Projection of all samples in training set on one of the axes. Sorting of points' projections.

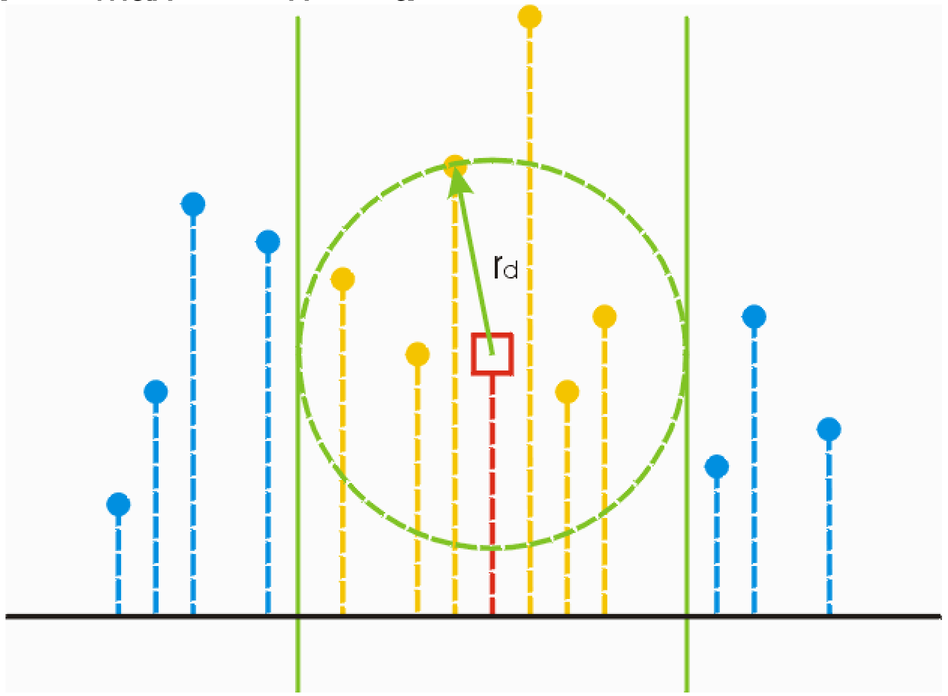
Searching for nearest neighbour of x:

1. Localize x on the axis (P_x- projection).
2. Find two nearest P_x (on both sides) projections of samples from training set (these are projections of points A, B).

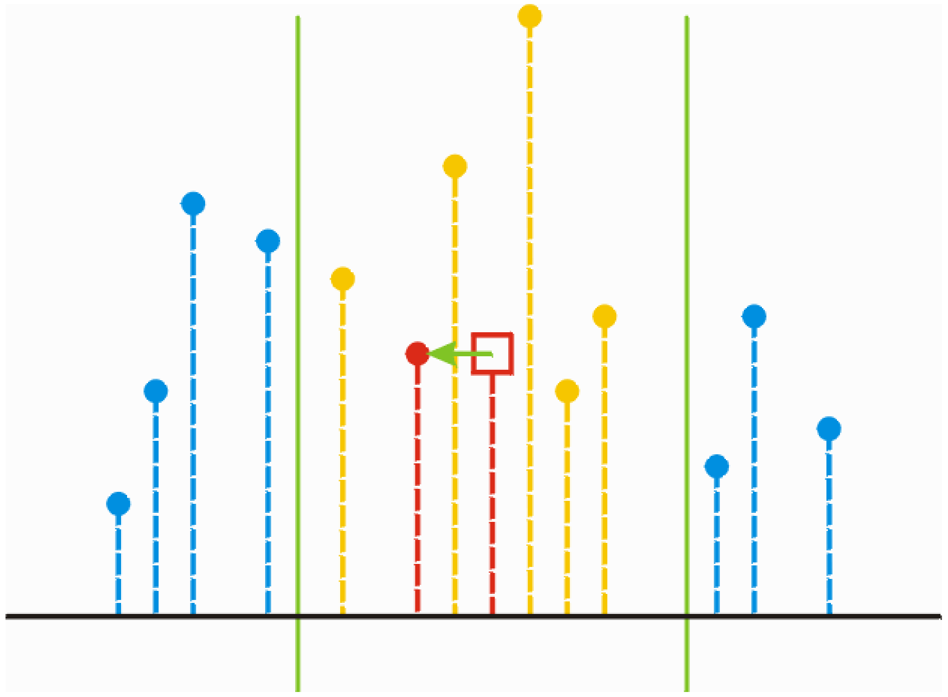


3. Compute minimum distance $\mathbf{r_d}$ between x (not P_x) and points A, B.

4. Determine boundaries of NN search range on the axis:
   $<G_{min} = P_x + \mathbf{r_d}, G_{max} = P_x - \mathbf{r_d}>$



5. Compute distances between x and all samples, which projections are in search range (Kd set).
6. Find the smallest distance in Kd set ($\rightarrow$NN).



7. If --k, "remove" from the training set point (NN) found in step 6 and return to step 2.

Extension to d dimensions

Preprocessing

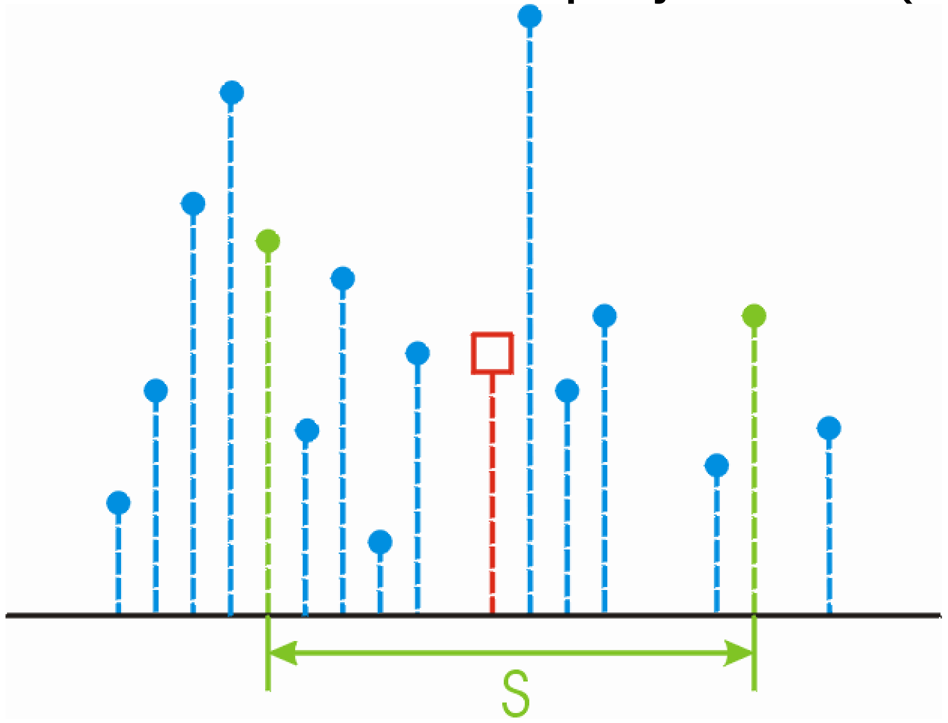
1. Project all samples **on all axes**; sort projections.
2. Compute expected number of calculations (***n***) for uniform distribution (the worst case).

Determination of *n*

Manhattan	1	$(kd!)^{\frac{1}{d}} N^{1-\frac{1}{d}}$
Euklidesowa	2	$\sqrt{\pi}\left[kd\left(\frac{d}{2}-1\right)!\right]^{\frac{1}{d}}(2N)^{1-\frac{1}{d}}$
Max	∞	$k^{\frac{1}{d}} N^{1-\frac{1}{d}}$

Searching for nearest neighbour of x:

1. Localize x on all axes.
2. Find (***n/2***)th projections on both sides of x.
3. Compute distance between these projections (***S***).

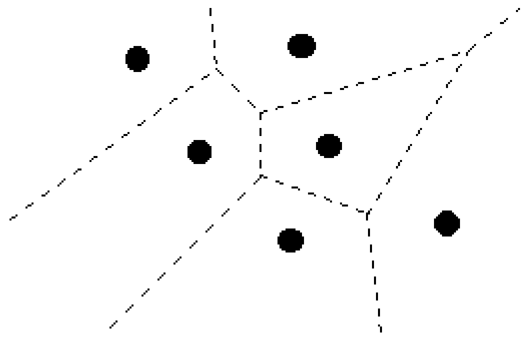


4. Determine local projection desity ***D=n/S*** in the neighbourhood of x projection.

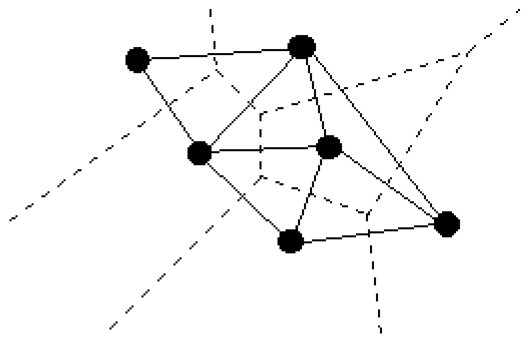
5. Find axis with the lowest projection density ***D***. This axis will be used in further steps (6-9). P_x is the projection of x on the axis.
6. Find two nearest P_x (on both sides) projections of samples from training set (these are projections of points A, B).
7. Compute minimum distance $\mathbf{r_d}$ between x (not P_x) and points A, B.
8. Determine boundaries of NN search range on the axis:
   $<G_{min} = P_x + \mathbf{r_d}, G_{max} = P_x - \mathbf{r_d}>$
9. Compute distances between x and all samples, which projections are in search range (Kd set).
10. Find the smallest distance in Kd set ($\rightarrow$NN).
11. If --k, "remove" from the training set point (NN) found in step 10 and return to step 2.

Condensation of the training set

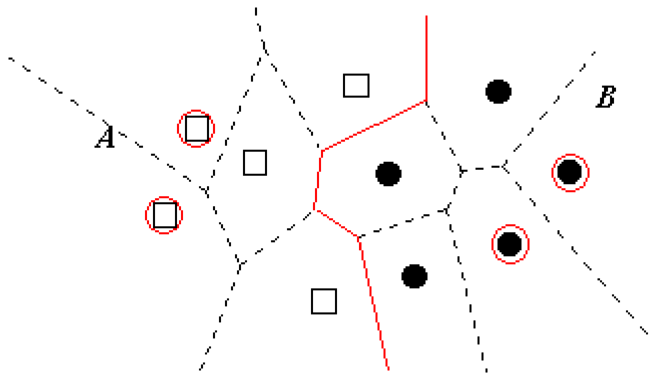
Voronoi diagram - for given set of nodes it is space division into regions, in which all points are closer to region's node than to any other node.



Delaunay triangulation - it's dual of Voronoi diagram. If two Voronoi regions have common edge, then nodes of these regions are connected. Such nodes are called Voronoi (Delaunay) neighbours.



Decision boundary for 1-NN classification belongs to Voronoi diagram.



Voronoi regions' nodes, which edges do not belong to decision boundary are superfluous. They can be removed from training set.

Condensing algorithm

1. Compute Delaunay triangulation for training set.
2. For each node, mark if all its Delaunay neighbours belong to the same class as the node considered.
3. Remove all *marked* nodes. The remaining nodes comprise condensed training set.

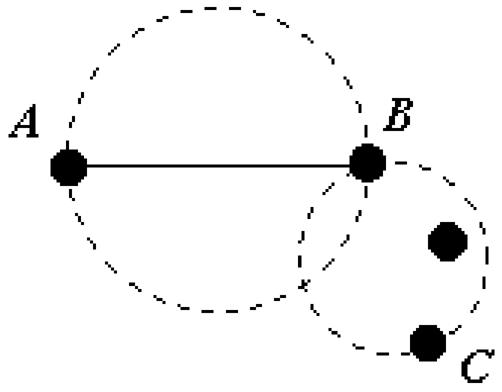
Proximity graphs

Proximity graph represents spatial distribution of the points belonging to the set.
In such a graph two points are connected if they are sufficiently near - in the sense of some metric.

Voronoi graph
Gabriel graph
Relative neighbourhood graph

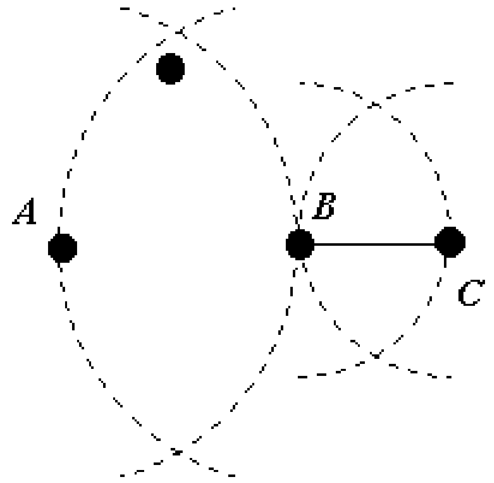
Gabriel graph

Gabriel graph: two points A and B of the set are connected with edge *ab*, if no other point in the set lies in the circle of diameter AB.



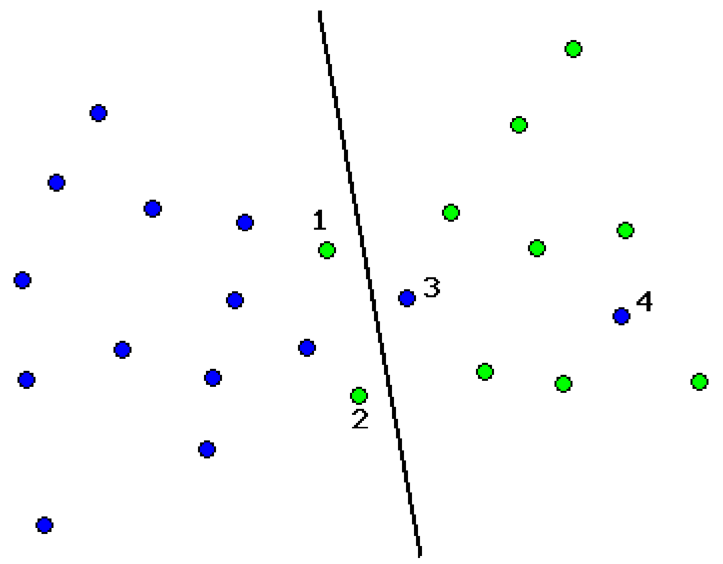
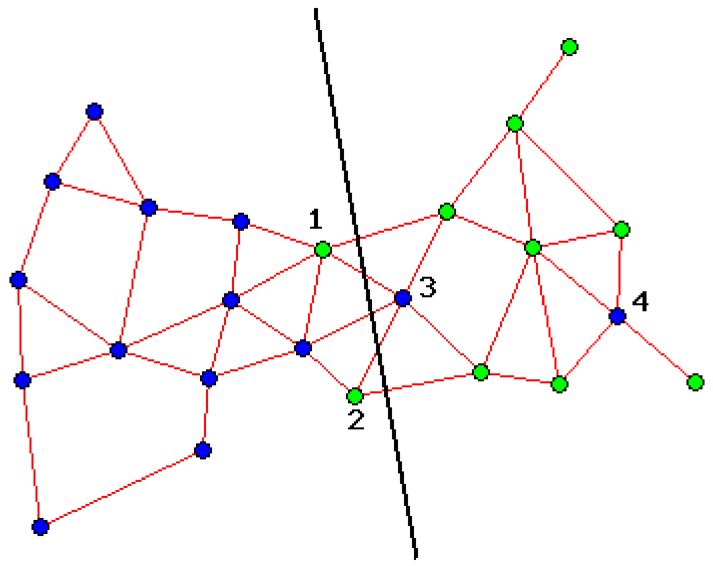
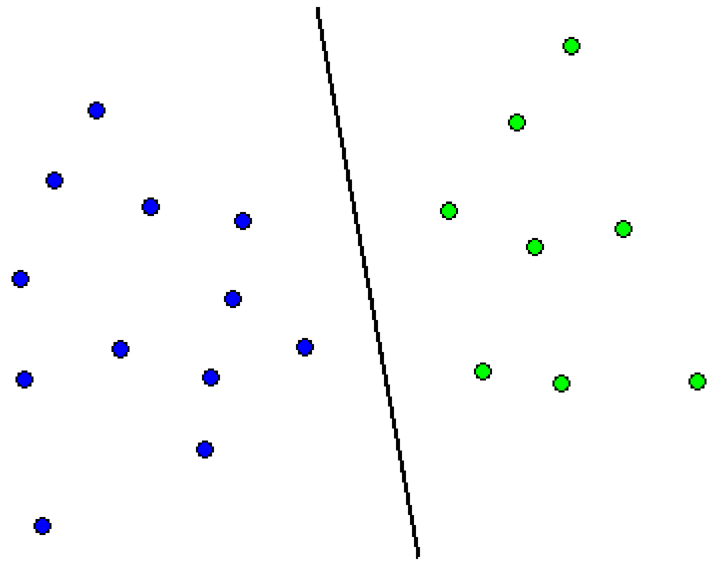
Relative neighbourhood graph

Relative neighbourhood graph: two points A and B of the set are connected with edge *ab*, if no other point in the set lies inside the *lune* of points A and B.
(Intersection of circles centered in A and B with diameter |AB|).



Editing algorithm

1. Build proximity graph of the training set.
2. For each graph's node:
   a. Find all neighbours of the node.
   b. Determine most frequent class in the neighbourhood (winning class)
   c. *Mark* if node belongs to class other than neighbourhood's winning class.
3. Remove *marked* nodes from the training set.


1
2
3
4

1
2
3
4


Training set reduction - MULTIEDIT

Devijver and Kittler (1982)

1. Diffusion
   Split randomly training set T into M subsets T_1, ... T_M, $M \geq 3$.
2. Classification
   Classify samples from set T_i using 1-NN method with $T_{(i+1) \bmod M}$ as the training set, i = 1, ... M.
3. Editing
   Remove all samples misclassified in step 2.
4. Blending
   Join remaining samples into new training set T.
5. Termination
   If M last iterations did not change set T terminate, in other cases return to step 1.

Condensing algorithm

We set up bins called Store and Grabbag. The first sample is placed in Store all the other samples are placed in Grabbag. We let n_k designate number of samples in Grabbag whenever step 1 of the algorithm is entered.

1. For i = 1, ... n_k, that is all samples in Grabbag
   Classify *i*th sample of Grabbag with 1-NN rule using Store as the training set. If classified *correctly* the sample is returned to Grabbag, otherwise it is placed in Store.

2. If no sample was transferred from Grabbag to Store in step 1, or Grabbag is empty then terminate; else return to step 1.

The final contents of Store constitute the condensed training set for 1-NN classifier.