Instance-based Learning or Nearest Neighbor Models

PROF. JACQUES SAVOY
UNIVERSITY OF NEUCHATEL

Overview

Introduction

Distance measures

Classification / regression

Computation

Nearest Shrunken Centroids

Instance-based Representation

Simplest form of learning: *rote learning* (plain memorization).

- Training instances are searched for existing instances that most closely *resembles* new example.
- The predicted class is given by the nearest training example.
- The instances themselves represent the knowledge.
- Similarity function defines what's "learned".

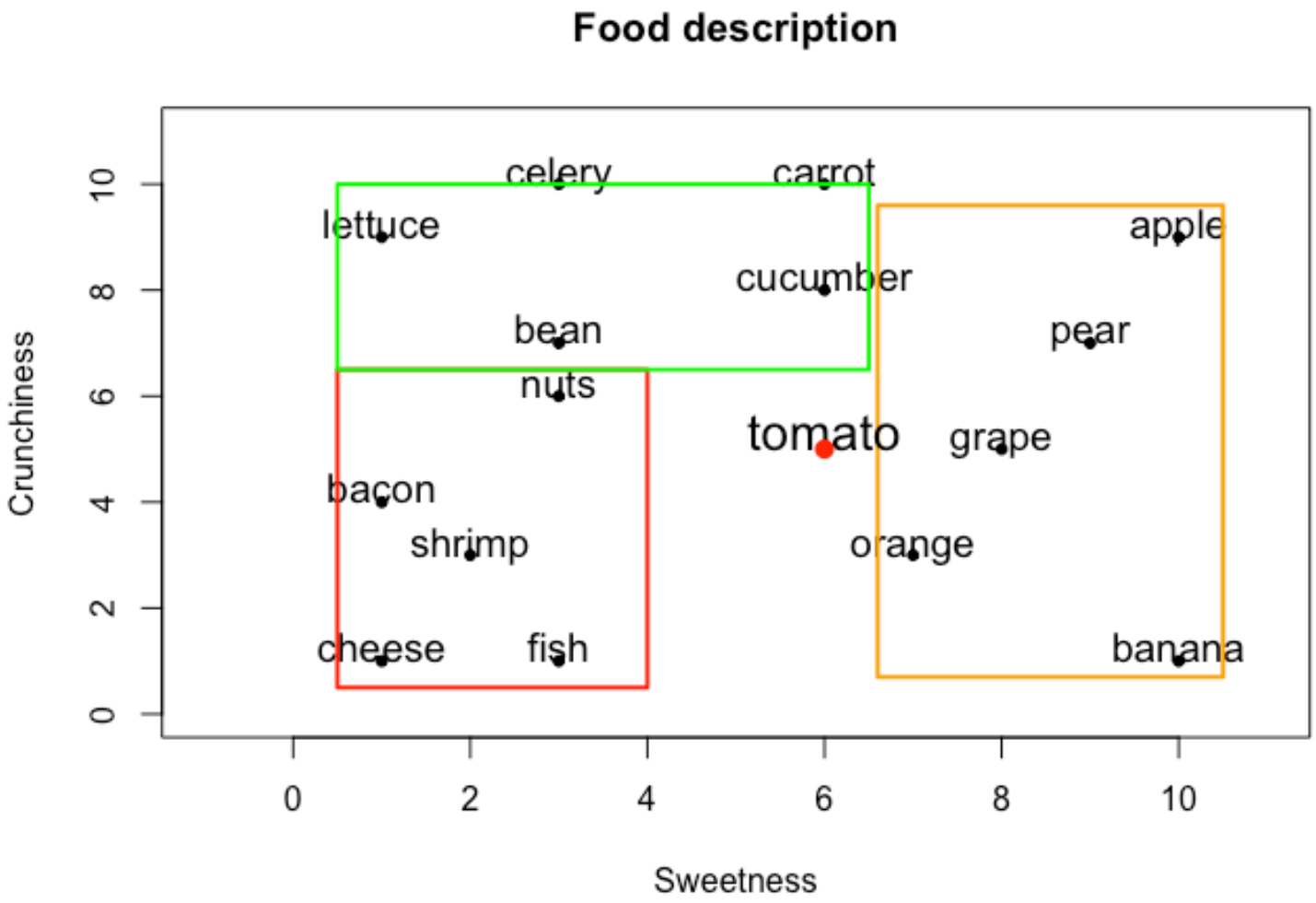
Instance Spa

Represent the instance space in 2D.

Question:
Label for:
`tomato(6, 5)`

Small distance = higher the similarity.

How to compute the distance between two instances?



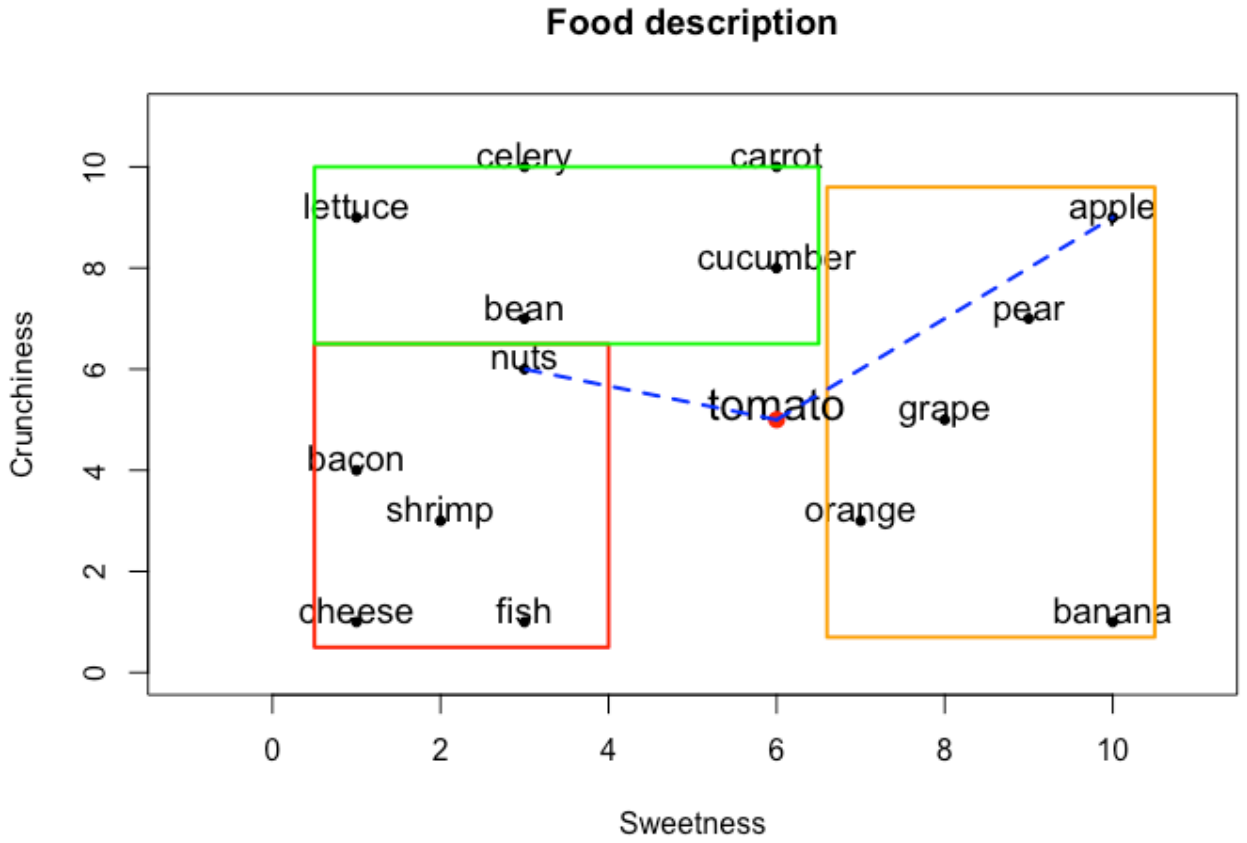
Instance Space

Represent the instance space in 2D.

Question:
Label for:
`tomato(6, 5)`

Small distance = higher the similarity.

How to compute the distance between two instances?



Overview

Introduction

Distance measures

Classification / regression

Efficient Computation

Nearest Shrunken Centroids

Define a Distance Function

Simplest case: one numeric attribute

- Distance is the difference between the two attribute values involved (or a function thereof)

Feature \ Object	Nuts	Apple	Tomato
Sweetness	3	10	6

Distance (Tomato, Nuts) = 6 - 3 = 3

Distance (Tomato, Apple) = 6 - 10 = -4

Distance (Apple, Tomato) = 10 - 6 = 4

Use the absolute value (L_1-norm). All distances must be positive.

Let try another distance measure… the Euclidian distance!

Distance Function

Several numeric attributes: normally, Euclidean distance is used (default in various packages).

Feature \ Object	Nuts	Apple	Tomato
Sweetness	3	10	6
Crunchiness	6	9	5

$$distance(Tomato, Nuts) = \sqrt{|6-3|^2 + |5-6|^2}$$
$$= \sqrt{3^2 + 1^2} = \sqrt{10}$$

Computing the square root is not always necessary (only the rank is important).

Distance Function

A "new" ISO standard can be used to measure the sweetness of a product.

Feature \ Object	Nuts	Apple	Tomato
Sweetness	3,000	10,000	6,000
Crunchiness	6	9	5

$$distance(Tomato, Nuts) = \sqrt{|6,000 - 3,000|^2 + |5 - 6|^2}$$
$$= \sqrt{3,000^2 + 1^2} = \sqrt{9,000,001}$$

Large values have an clear impact on the distance.

Attribute values must be normalized!

Normalization

We can normalize the values between 0 and 1 using the following rule. For an attribute with value w_k, we may compute its new normalizes value w'_k as:

$$w'_k = \frac{w_k - min(w_j)}{max(w_j) - min(w_j)}$$

Nominal attributes (e.g., color): distance is set to 1 if values are different, 0 if they are equal.

Ordinal value: same as nominal or the distance between the values can be estimated as for example as (0, 1, 2, 3) or (0, 0.5, 2, 4), …

Normalization Z score

We can also normalize the values by transforming them to their Z score values (standardization).

For an attribute with value w_k, the Z score value w'_k is:

$$w'_k = \frac{w_k - \mu_w}{\sigma_w} \approx \frac{w_k - \hat{\mu}_w}{\hat{\sigma}_w} = \frac{w_k - \bar{x}_w}{s_w}$$

Subtracting the (estimated) mean and divide by the (estimated) standard deviation.

No real min or max value.

We expect 68% of the values between [mean - sd; mean + sd].
or 95% of the values between [mean - 2 *sd; mean + 2 * sd].

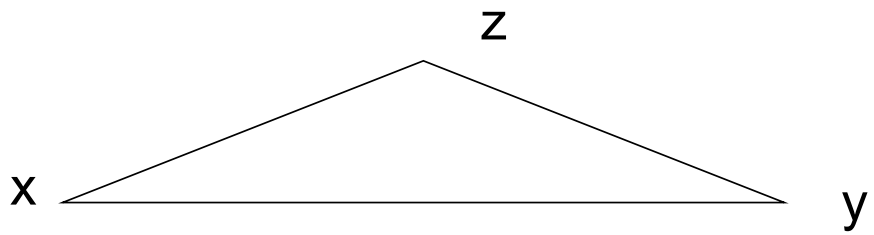
Axioms of a Distance Metric

Various possible functions to measure a *distance*.

d is a *distance metric* (or simply a *distance function* or a *metric*)
if it is a function from a pair of points to real numbers such that:

1. $d(x,y) \geq 0$.
2. $d(x,y) = 0$ iff $x = y$.
3. $d(x,y) = d(y,x)$.
4. $d(x,y) \leq d(x,z) + d(z,y)$ (*triangle inequality*).

Look at some functions (assuming having only numerical features).



Example of Distance Measures

According to the L^1 norm

Manhattan distance = distance if you had to travel along coordinates only "walking round the blocks".

$$Dist_{Manhattan}(A,B) = \sum_{i=1}^{t} |w_{iA} - w_{iB}|$$

Canberra is a "normalized" version of Manhattan (but can be larger than 1.0)

$$Dist_{Canberra}(A,B) = \sum_{i=1}^{t} \frac{|w_{iA} - w_{iB}|}{|w_{iA}| + |w_{iB}|}$$

Example of Distance Measures

According to the L^1 norm

We usually obtain good results with the Manhattan and Tanimoto distance functions.

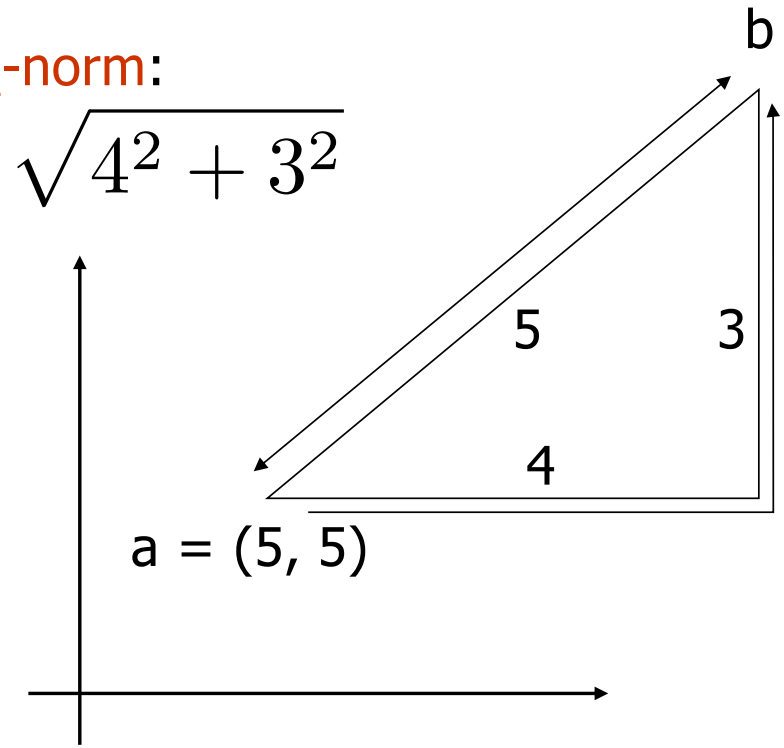
Usually, we assume that all values are positive.

$$Dist_{Tanimoto}(A,B) = \frac{\sum_{i=1}^{t} |w_{iA} - w_{iB}|}{\sum_{i=1}^{t} max(w_{iA}, w_{iB})}$$

Examples of Euclidean Distances

L_2-norm:

$$dist(a,b) = \sqrt{4^2 + 3^2}$$



L_1-norm:

dist(a, b) = 4 + 3 = 7

Example of Distance Measures

According to the L^2 norm

The most common notion of “distance”, distance "as the crow flies"

$$Dist_{Euclidian}(A,B) = \sqrt[2]{\sum_{i=1}^{t} |w_{iA} - w_{iB}|^2}$$

Clark, a "normalized" version of the Euclidian distance.

$$Dist_{Clark}(A,B) = \sqrt[2]{\sum_{i=1}^{t} \left(\frac{|w_{iA} - w_{iB}|}{w_{iA} + w_{iB}} \right)^2}$$

Example of Distance Measures

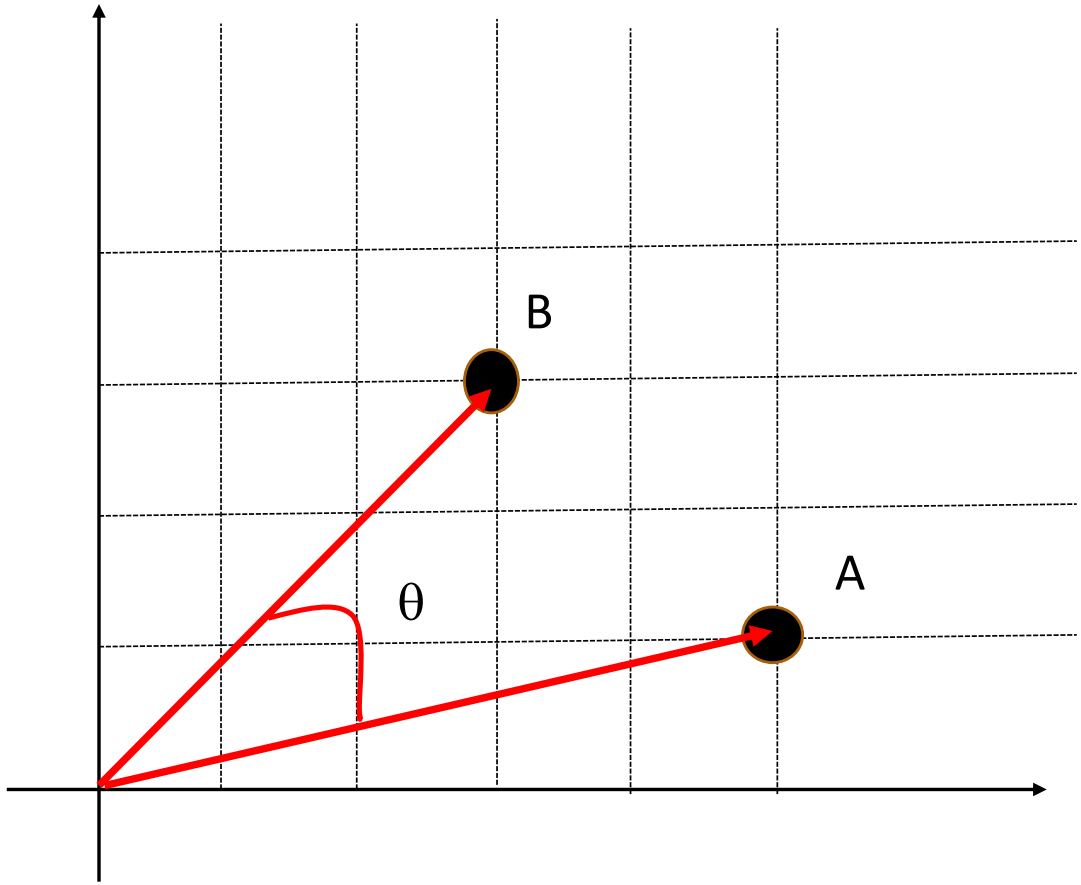
Cosine distance, a well-known and frequently used function.

First compute the cosine similarity.

$$Sim_{Cosine}(A,B) = \cos(\theta) = \frac{\sum_{i=1}^{t} w_{iA} \cdot w_{iB}}{\sqrt{\sum_{i=1}^{t} w_{iA}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{iB}^2}}$$

$$Dist_{Cosine}(A,B) = \frac{Sim_{Cosine}(A,B)^{-1}}{\pi}$$

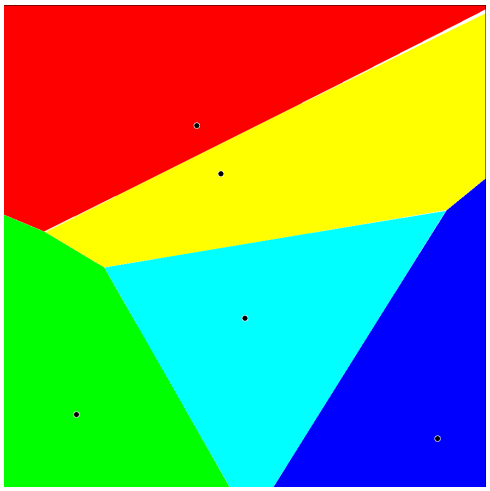
Example of Cosine Similarity



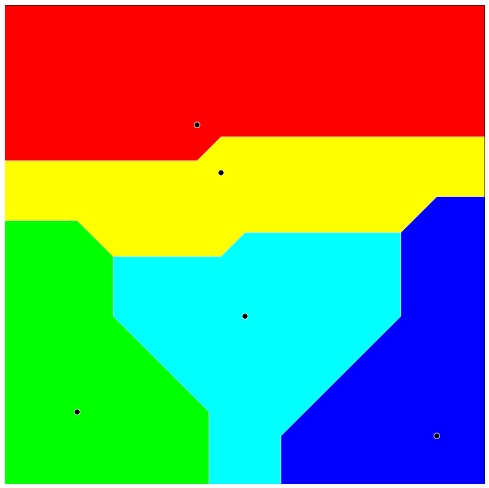
Not all the Same?

With five points, what is the region cover by each of them when the distance function is

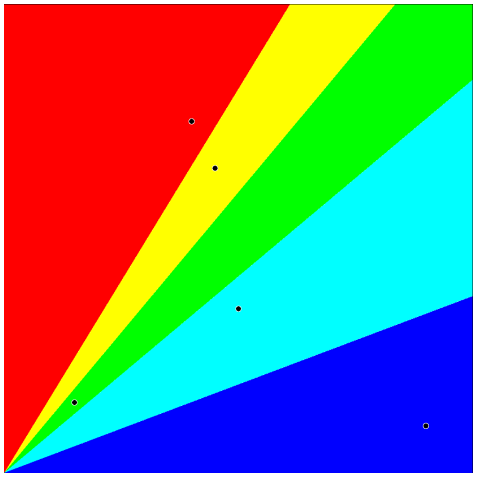
Euclidian



Manhattan



Cosine



Distance Measures for Sets

Definition of Jaccard similarity.

$$Sim_{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Similarity values are between 0 and 1.

Example:

- Note by "1" the presence of the feature
  A = **1** 0 1 **1 1**
  B = **1** 0 0 **1 1**.
- Size of intersection = 3, size of union = 4, therefore Jaccard similarity (not distance) = 3/4.
- Ignore the 0-0 matching!

d(x,y) = 1 – (Jaccard similarity)

In our example 1 - 3/4 = 1/4.

Distance Measures for Sets

Definition of Dice similarity.

$$Sim_{Dice}(A, B) = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

Similarity values are between 0 and 1.

Example:

- Note by "1" the presence of the feature
  A = **1** 0 1 **1 1**
  B = **1** 0 0 **1 1**.
- Size of intersection = 3, size of A = 4, size of B = 3, therefore Dice similarity (not distance) = (2*3) / (4+3) = 6/7.
- Ignore the 0-0 matching!

d(x,y) = 1 – (Dice similarity)

In our example 1 - 6/7 = 1/7.

Edit Distance

The edit distance of two strings is the number of inserts and deletes of characters needed to turn one into the other.

Equivalently:

$d(x,y) = |x| + |y| - 2|LCS(x,y)|$.

- LCS = *longest common subsequence* = longest string obtained both by deleting from *x* and deleting from *y*.

Example*:* *x* = *abcde* ; *y* = *bcduve*.

Turn *x* into *y* by deleting *a*, inserting *u* after d inserting *v* after *u*.

- Edit-distance = 3.

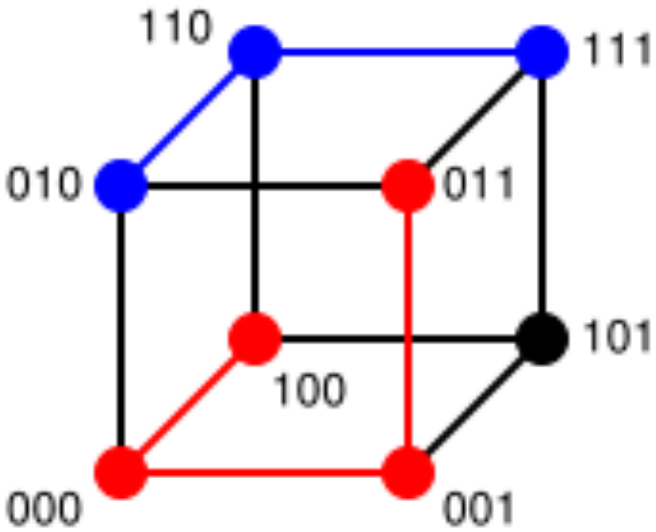
Or, LCS(x,y) = *bcde*.

$|x| + |y| - 2|LCS(x,y)| = 5 + 6 - 2 \cdot 4 = 3$.

Other Distances

Hamming distance between two strings of *equal length* is the number of positions for which the corresponding symbols are different.

It measures the minimum number of *substitutions* required to change one into the other, or the number of *errors* that transformed one string into the other.



With Mixed Data Types?

Previous distance definitions assume that all *p* predictors are of the same type (usually numerical). Then we can use the Euclidian or Manhattan function.

With different types (nominal and numerical)?

- Dichotomize all variables -> all are binary data.
- Apply two similarity functions (one for categorical data, one for numerical ones
  -> combined the two values.
- Apply a new distance function, e.g., Gower's distance function.

With Mixed Data Types?

With different types (nominal and numerical)?

Example, for the object X and Y, and for the *i*th attribute:

if numerical: $|w_{iX} - w_{iY}|$ = Positive value

if nominal (binary): $I(w_{iX} \neq w_{iY})$ = {0, 1}

And the final distance is just the sum of the *p* contributions.

Could be important to normalize the numerical values before applying the distance computation (e.g., function `scale()` in R).

Gower Distance Function

Generalization of the previous idea.

Each predictor will contribute with a value [0,1].

For the object X and Y, and for the *i*th attribute, its contribution is d^i_{XY}

if nominal (binary): $d^i_{XY} = w_k \cdot I(w_{iX} \neq w_{iY})$

if numerical: $d^i_{XY} = w_k \cdot \frac{|w_{iX} - w_{iY}|}{max_k(w_{ik}) - min_k(w_{ik})}$

in which a weight w_k is assigned to the *k*th predictor (usually all = 1).

Ordinal variable: as numerical one.

Final distance is just the sum of the *p* contributions divided by *p*.

In presence of a NA value, ignore the attribute.

In the presence of a binary asymmetric attribute, ignore when both are 0.

Gower Function in R

```
Distance Gower:
        1       2       3       4
2   0.0833
3   0.3333  0.4167
4   0.7500  0.6667  0.4167
5   0.9167  1.0000  0.5833  0.3333
```

	virus	location	density
#1	0	1	0.5
#2	0	1	0
#3	1	1	0.5
#4	1	0	0
#5	1	0	2

Distance Gower:

D(#1, #2) = (0 + 0 + ((0.5-0) / (2-0))/ 3 = (0.5/2) / 3 = 0.25/3 = 0.0833

Overview

Introduction

Distance measures

Classification / regression

Efficient Computation

Nearest Shrunken Centroids

k-NN Model

A simple model (lazy learning, instance-based learning, case-based learning, …).

A non-parametric model: y = f(x). No constraint on f() (any curve).

Methods: *nearest neighbor* (NN)*, k-nearest neighbor* (k-NN)

with *k* = 1, 3, … odd number

We suggest having a rather "large" value for *k*
(when having a lot of training examples)

Given a *k* value, we can determine regions belonging to the same class.

These regions are not rectangular (decision tree) but may have a general form (depending of the value of *k*).

Interesting case...

Two numerical attributes.
Values between 0.0 and 5.0.

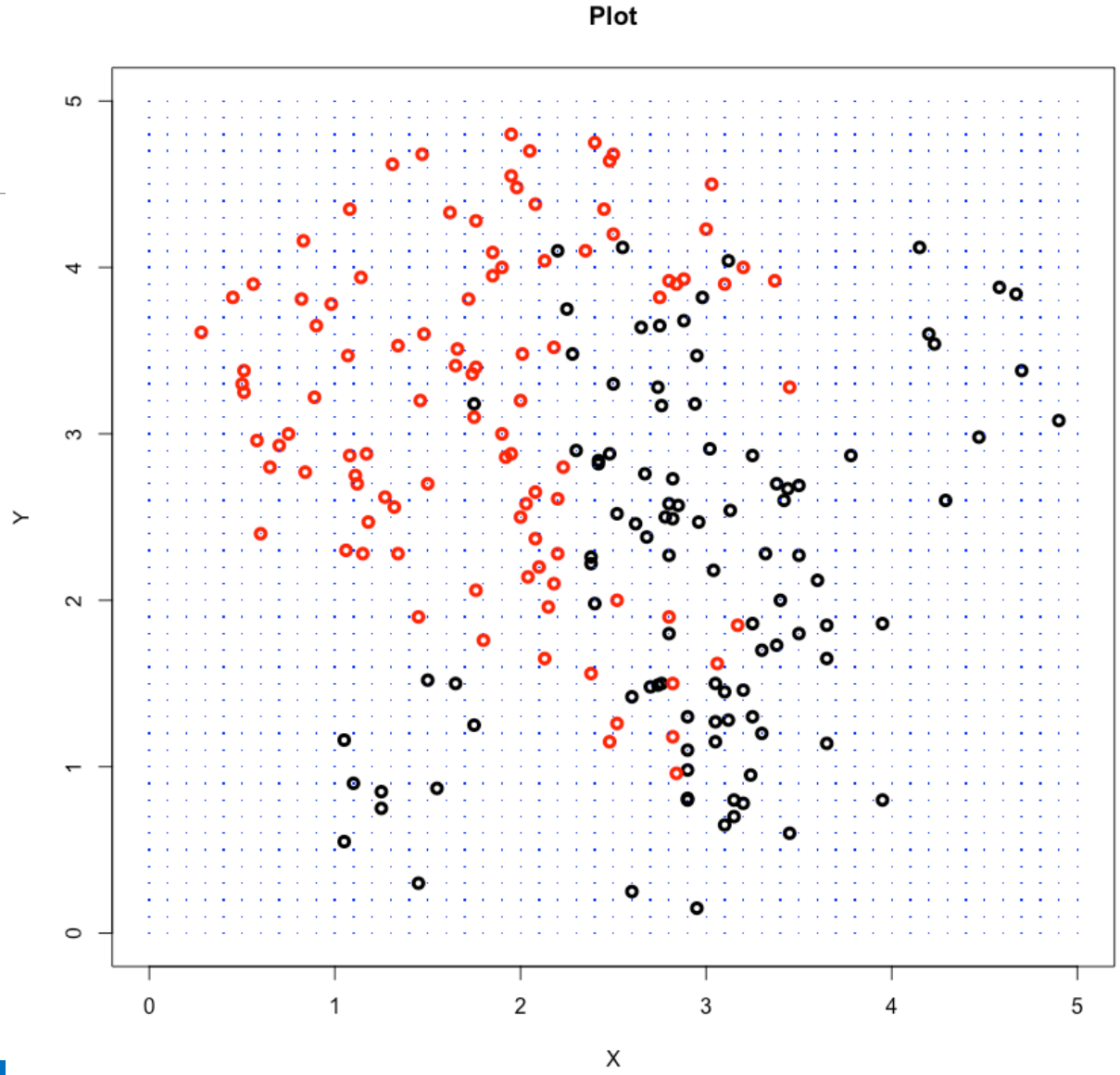
Two possible categories (red/black).

Same set for learning and testing.
K=3. Success rate: 92.5%.

K=1? Success rate: 100%.

Different sets for learning and testing.
K=3. Success rate: 86%.

Which estimation is the "best"?



Instance-based

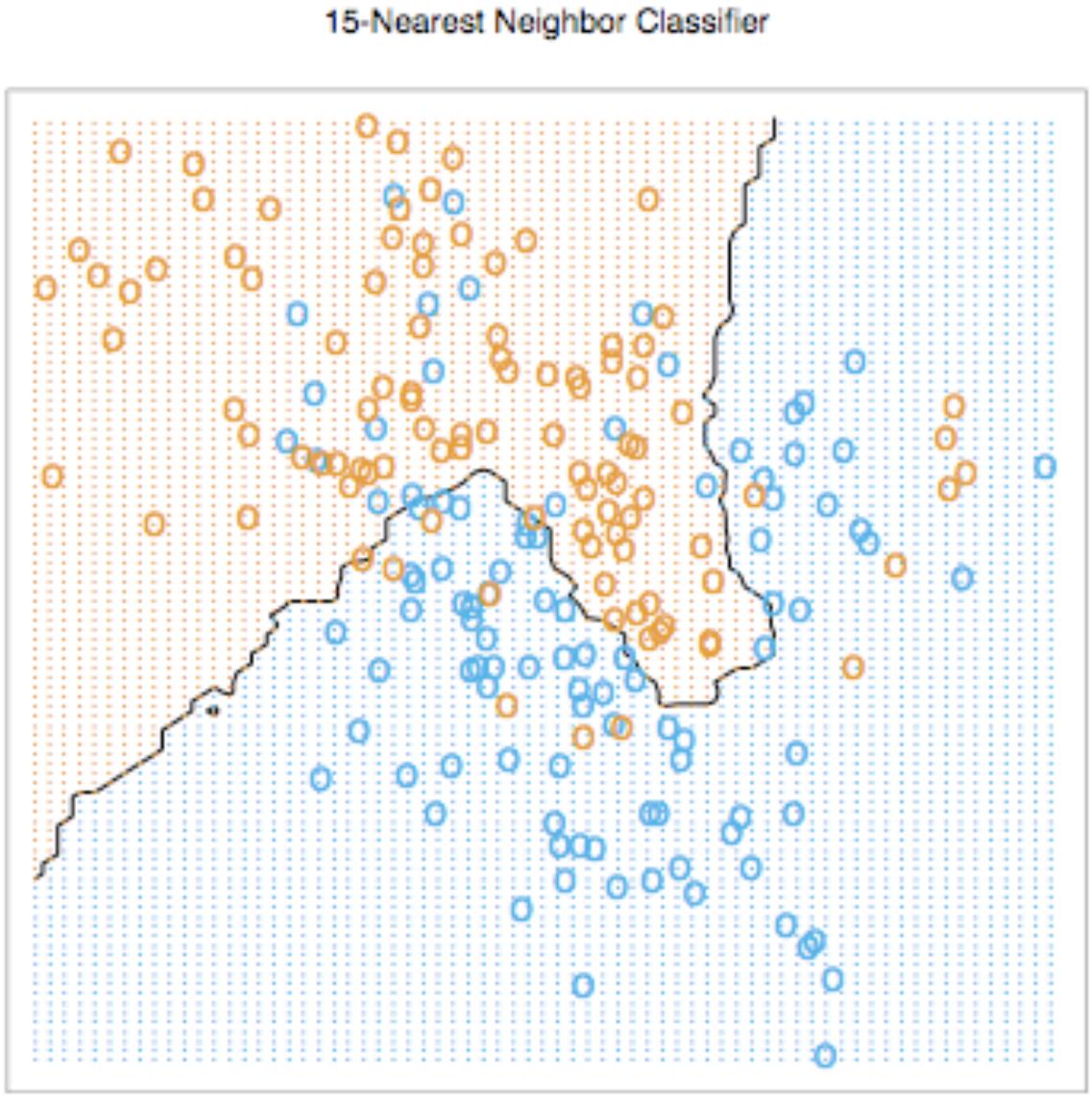
Binary Response

1-Nearest Neighbor

Instance-based R

Binary Response

15-Nearest Neighbors



k-NN Regression

Similar to the k-NN for classification.

Define a distance measure (or similarity measure). E.g., Euclidian distance.

Determine the *nearest neighbor* (NN, implicitly 1-NN)*, k-nearest neighbor* (*k*-NN) with $k = 1, 2, 3, \ldots$

To determine the target value, compute the mean over all *k* nearest neighbors.

We suggest having a rather “large” value for *k* (when having a lot of training examples).

k-NN Classification & Regression

The problem is to define a distance metric (or measure).

What to do when facing with NA values?

- Considering the NA as one possible value? Possible in classification.
- Replacing by the mean value of the corresponding attribute?
- Ignoring this attribute?

How to select the best value for *k?*
(the hyper-parameter (or meta-parameter) for this model).

- Having a theory explaining this choice?
- By trying different values for *k*.

Overview

Introduction

Distance measures

Classification / regression

Efficient Computation

Nearest Shrunken Centroids

Computation

Simplest way of finding the nearest neighbor (*n* instances, *p* predictors):
linear scan of the data (brute force approach) $O(n \cdot p)$

- Classification takes time proportional to the product of the number of instances in training and test sets thus we have $O(n_1 \cdot n_2 \cdot p)$.

We can do better by having a *better* representation of our training set.

Nearest neighbor search can be done more efficiently using appropriate data structures.

We will discuss different methods that represent training data in a tree structure: pruning, IB3, *k*D-tree and ball tree (but shortly).

Pruning

Assume k = 1 and only the training set is used.

Assume some exemplars are already stored.

When a new instance is analyzed:

- If the closest exemplar produce the good answer, discard this instance
- Otherwise, stored this instance (and call it exemplar).

A single exemplar is enough to describe a region.

But: cold-start problem.

Training set is not noisy.

The order of the instances could be important.

IB3 (Instance-based learner version 3)

With *k* = 1, noisy exemplars have a clear impact. Solution: Increase *k*.

For each exemplar (stored), keep the number of correct and wrong decisions.

Have two thresholds on the accuracy rate. Work with *k* = 1.

- If the exemplar performance is below the lower limit, remove it.
- If the exemplar performance is larger than the upper limit, use it in prediction.
- Between the two limits, continue monitoring its performance.

Threshold? Built a confidence interval with the Bernoulli process (see decision tree).

Reject: α = 12.5%. Accept: α = 5%.

E.g., thresholds: remove when < 16%, accept when > 65%

*k*D-Tree

The *k*D-tree is a binary tree (to store a set of points in a *k*-dimensional space).

- Each intern node may have up to two children.
- Divide the hyperspace by an hyperplane at each level (a line if we are in a 2D space).
- Do it recursively.

All splits are made parallel to one of the axes.

The hyperplanes are not decision boundaries.

See computer graphics textbooks.

*k*D-Tree

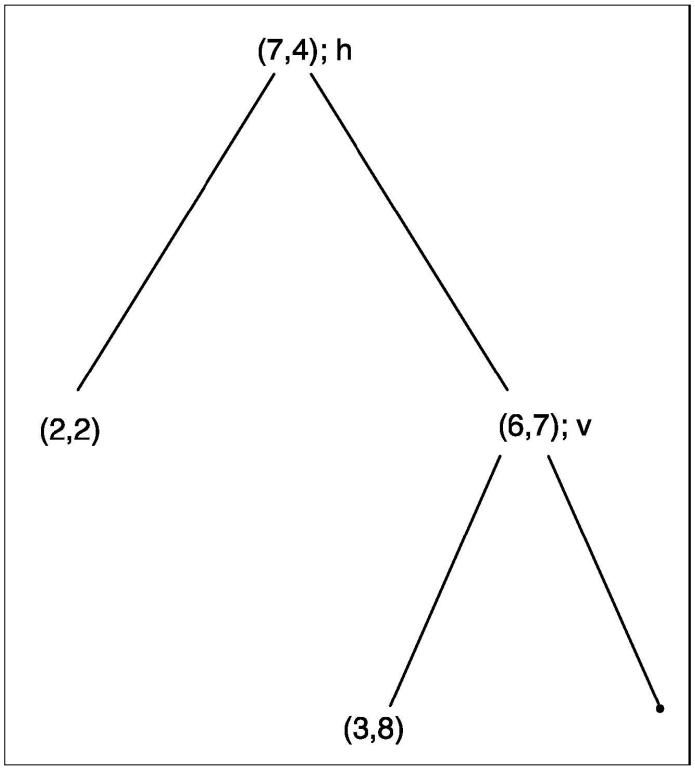
Example

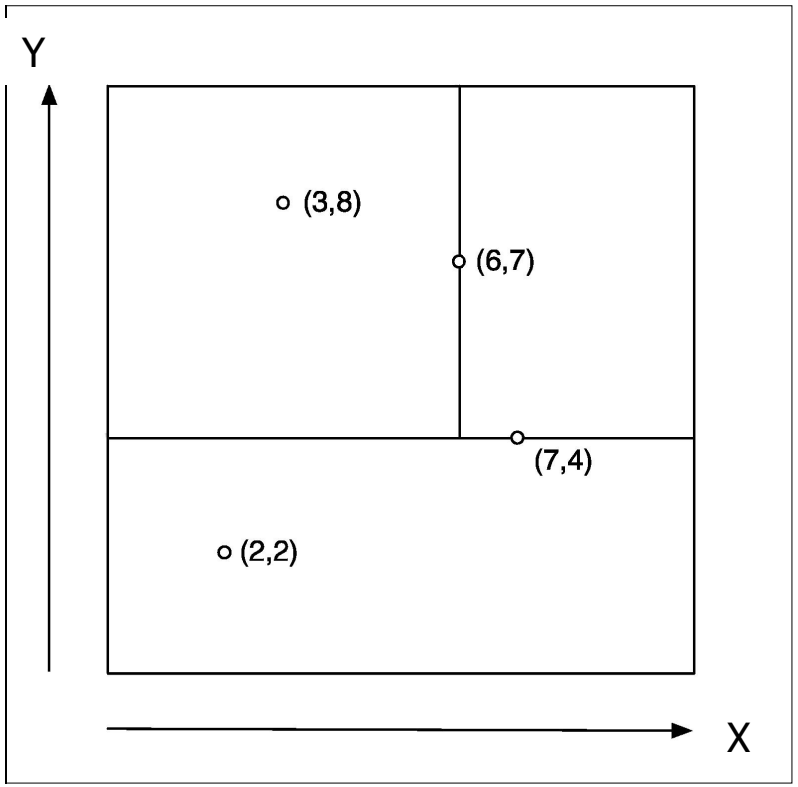
Objects	X	Y
a	2	2
b	3	8
v	6	7
h	7	4

- Chose an attribute (Y) to draw the first line (hyperplane).
- Select the instance *h* (7,4) as root.
  Smaller than Y=4 on the left, larger on the right
  (view as arbitrary decision for the moment).

*k*D-Tree

*k*D-tree (*k*=2) example with 4 examples





*k*D-Tree

With our example:

- each leaf contains either one point or none.
- each sibling branches are not necessarily developed at the same depth.
  Is it important?

How can we search into such a tree when having a new instance?

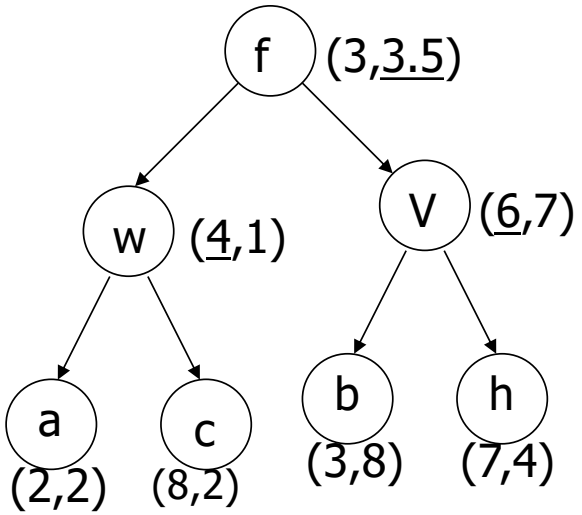
At each node, we must know the split direction (“X" or “Y" in our example) and the branches with upper or lower values.

First we add three instances in our *k*D-tree.

*k*D-Tree

We add new instances (*w*, *c*, *f*), we rebuild the tree.

A new example (star).



Objects	X	Y
a	2	2
b	3	8
v	6	7
h	7	4
w	4	1
c	8	2
f	3	3.5
☼ (new)	8	3

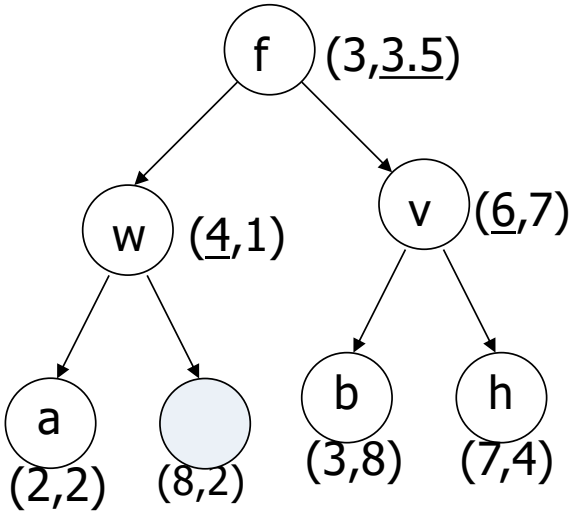
*k*D-Tree

How can we search into such a tree when having a new instance (e.g., ☼(8,3))?

The star is the new instance.

The leaf node of the region containing the target is colored grey.

This is a good first approximation.

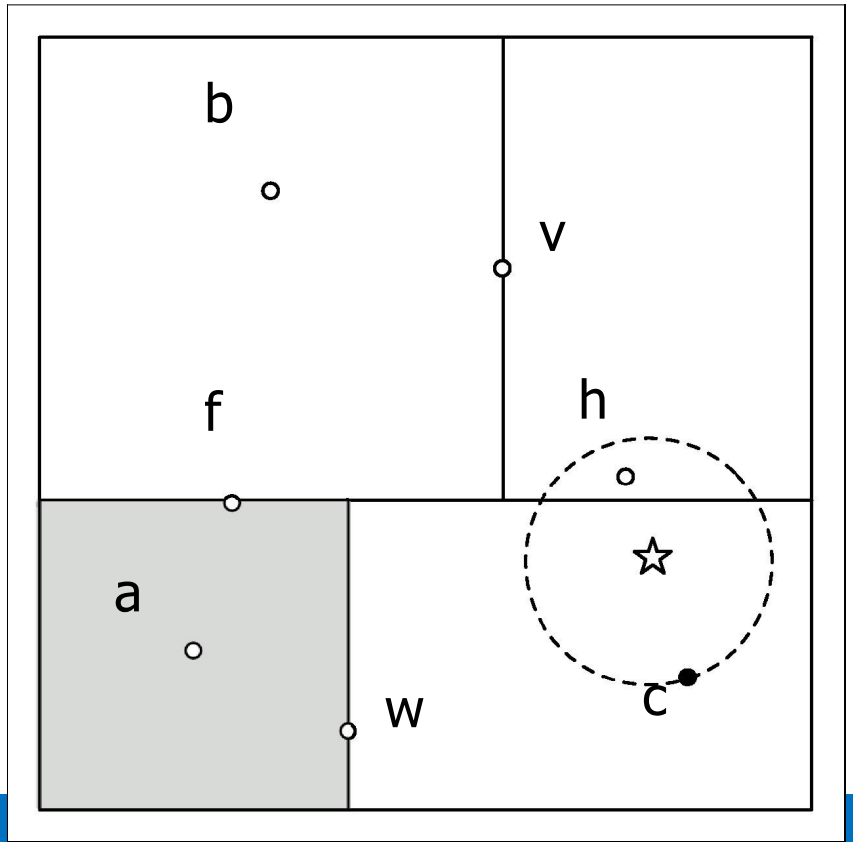


*k*D-Tree

$$Dist(c, \star) = \sqrt{(8-8)^2 + (3-2)^2} = 1$$

We have a potential nearest neighbor (with point *c*) and a potential shortest distance (distance(c,star)=1).

But the whole circle in not inside the rectangle!

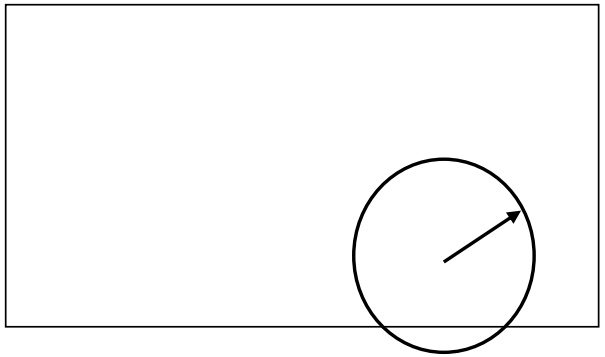


*k*D-Tree

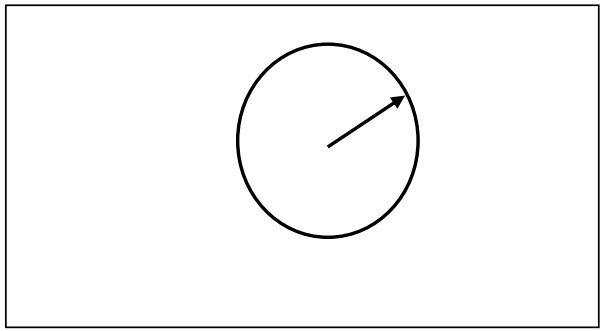
Where to find the possible NN?

Define the radius (distance between the new instance and the point in the leaf node, or between the star and the black point). Then you need some graphical testing functions.

Can you determine (fast) if the circle is inside a rectangle?

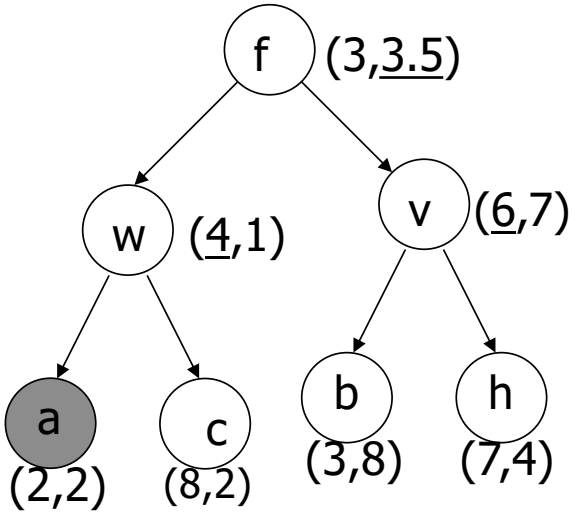


Can you determine (fast) if a point is inside a circle?



*k*D-Tree

- From this candidate (c) with a potential shortest distance distance (1) we must backtrack to find the rectangle that intersect with the circle.
- Backtracking procedure might go back to the root!
- Here, we visit rectangle “a”.



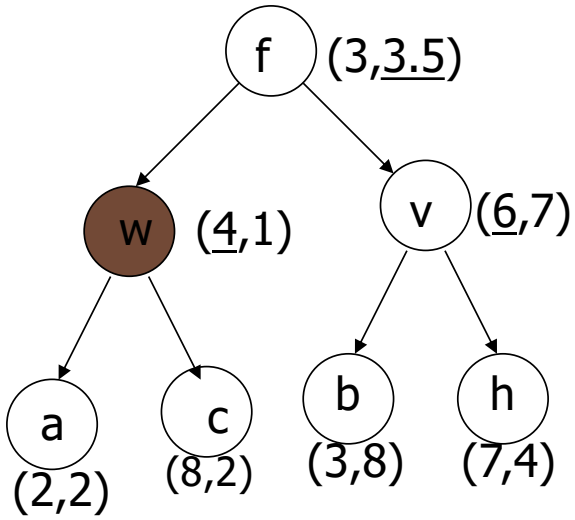
*k*D-Tree

Determine if it is possible for a closer neighbor to lie within the node's sibling (shaded). In our example, we have no intersection with the circle.

If not, back up to the parent node (w, then f) and check *the* other siblings. (In our case, we must do it).

Check if the aunt (v) intersects the circle and if yes, check for a closer point (its daughters, *b* and *h*).

(it does in our example).



*k*D-Tree

Complexity depends on depth of tree, given by logarithm of number of nodes $O(\log_2(n))$

Amount of backtracking required depends on quality of tree ("square" (better) vs. large rectangles). But usually $O(\log_2(n))$ … if the tree is reasonable (well built).

How to build a good tree? Need to find good split point and split direction.

A splitting strategy:

- Split direction: direction with greatest variance
- Split point: median value along that direction (the coordinates are easy to find).

Using value closest to mean (rather than median) can be better if data is skewed (but you need to find the point closer to it).

Can apply this recursively.

*k*D-Tree

Big advantage of instance-based learning: classifier can be updated incrementally

- Just add new training instances!

Can we do the same with *k*D-trees?

Heuristic strategy:

- Find leaf node (rectangle) containing new instance.
- Place instance into leaf if leaf is empty.
- Otherwise, split leaf according to the longest dimension (to preserve squareness).

Tree should be rebuilt occasionally (i.e. if depth grows to twice the optimum depth) to allow a better balance.

*k*D-Tree & Ball Tree

Problem in *k*D-trees: corners with a new instance close to one of them.

Observation: no need to make sure that regions don't overlap.

Can use balls (hyperspheres) instead of hyper-rectangles.

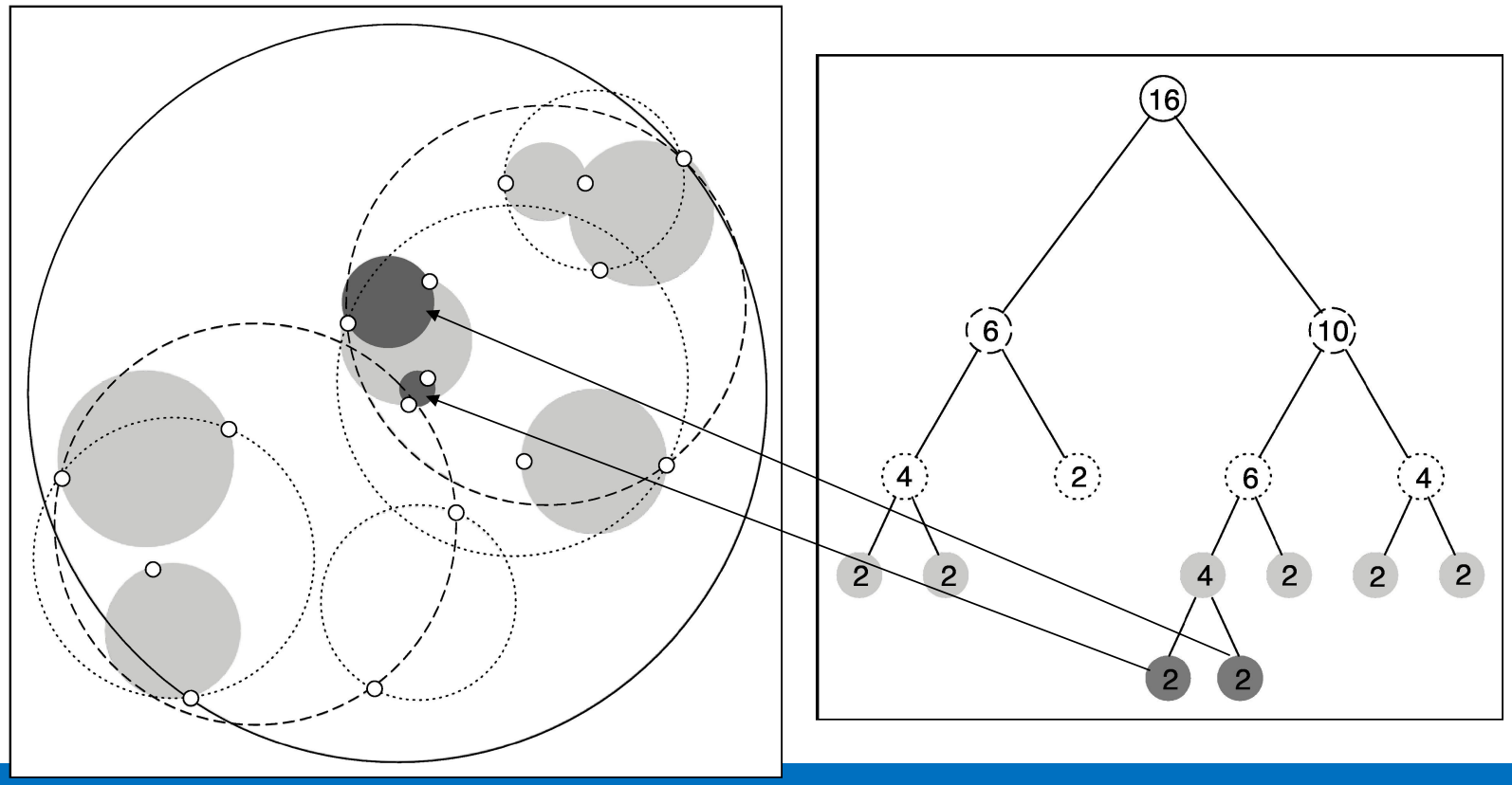
A *ball tree* organizes the data into a tree of *k*-dimensional hyperspheres.

Normally allows for a better fit to the data and thus more efficient search.

Point can belong to two circles but assigned to only one.

Ball Tree

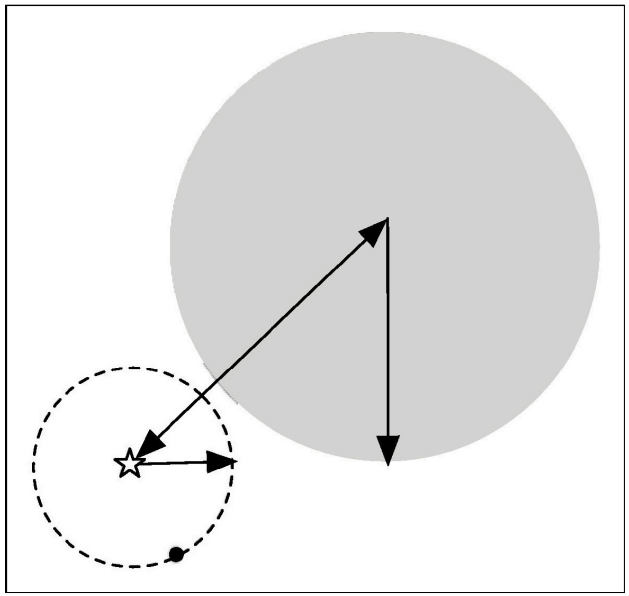
Example with 16 instances



Ball Tree

Nearest neighbor search is done using the same backtracking strategy as in kD-trees (top-down approach to define the leaf).

Ball can be ruled out from consideration if distance from target point (star) to ball's center exceeds ball's radius plus current upper bound (its nearest neighbor).



Overview

Introduction

Distance measures

Classification / regression

Computation

Nearest Shrunken Centroids

Nearest Shrunken Centroids

Based on *k*-nn approach.

Instead of considering all instances (for computing the distance with the new instance), build a profile for each class: profile-based NN (because *k*=1).

Previously, we have instance-based *k*-nn.

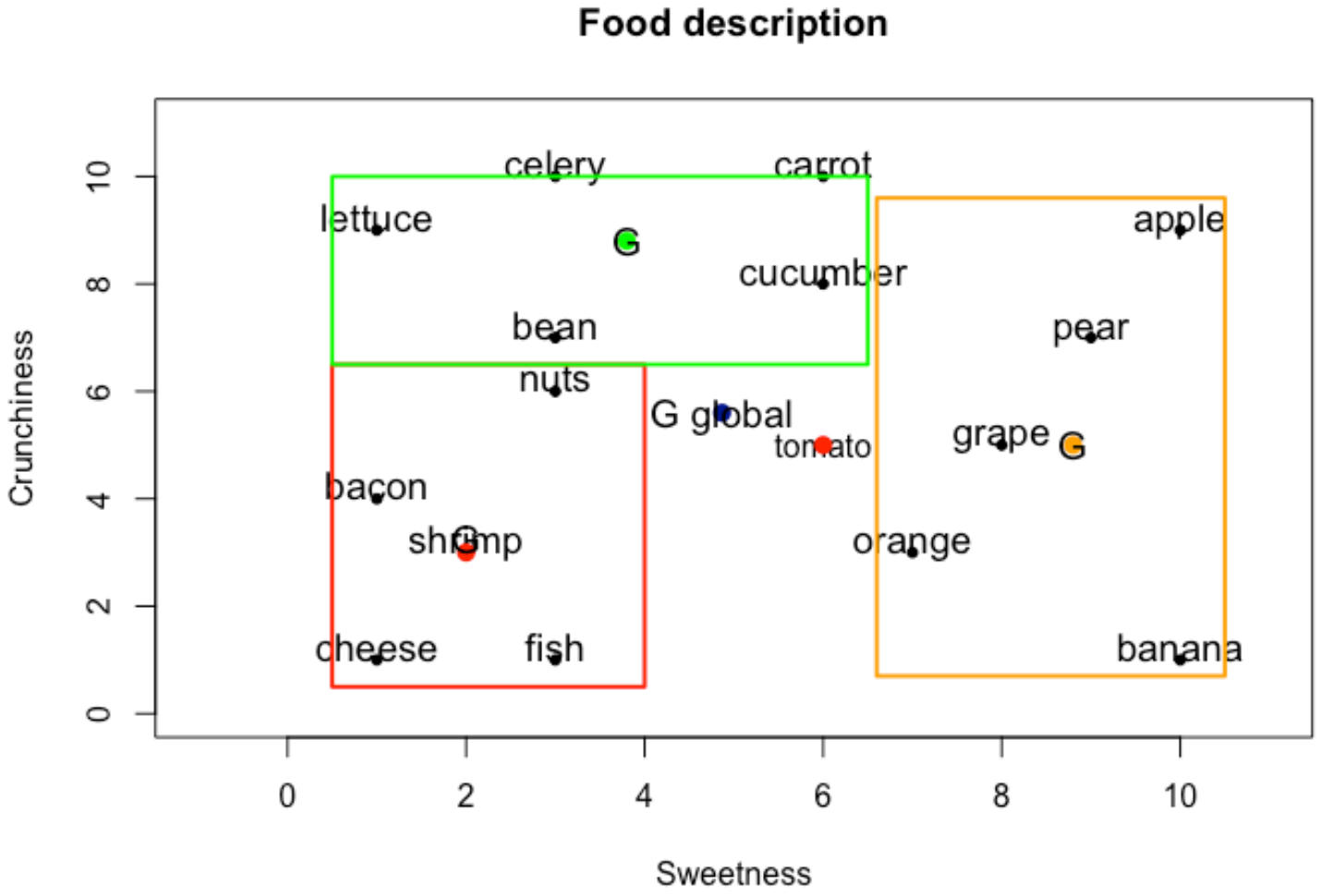
Each class is represented by centroid (profile, gravity center of the class).

Ignore small discriminant values and thus remove irrelevant features (for a given class).

Effective in different applications.

Nearest S

Distance with profile-based model



Nearest Shrunken Centroids

Given a set of features i = 1, 2, …, *p*.

Given a set of instances j = 1, 2, …, *n*.

Given a set of classes (labels) k = 1, 2, … *K*.

Let C_k indicates all the n_k observations with the label *k*.

We can define the prior of each class by n_k/n.

Gravity center (of the entire sample) as G = $[\bar{x}_1, \bar{x}_2, \ldots \bar{x}_p]$

with the mean for the *i*th feature

$$\bar{x}_i = \frac{1}{n} \cdot \sum_{j=1}^{n} x_{ij}$$

Nearest Shrunken Centroids

For the *k*th class and the *i*th feature, we compute its mean:

$$\bar{x}_{ik} = \frac{1}{n_k} \cdot \sum_{j \in C_k} x_{ij}$$

And we can generate the gravity center (profile) for each class with those means.

Thus the profile (centroid, gravity center) for class *k* is:

$G_k = [\bar{x}_{1k}, \bar{x}_{2k}, \ldots \bar{x}_{pk}]$

But if for the *i*th attribute, the difference between $\bar{x}_i$ and $\bar{x}_{ik}$ is small, the class *k* does not differ from the global mean.

Nearest Shrunken Centroids

Normalize within each class with

$$d_{ik} = \frac{\bar{x}_{ik} - \bar{x}_i}{m_k \cdot s_i}$$

Numerator: the difference between the mean of the *k*th class and the *i*th feature and the global mean for that feature.

Denominator: the standard deviation of this *i*th feature.

When d_{ik} close to 0 the *i*th feature not useful for that *k*th class.

Nearest Shrunken Centroids

And the denominator is the estimation of the standard error of the numerator.

$$d_{ik} = \frac{\bar{x}_{ik} - \bar{x}_i}{m_k \cdot s_i}$$

For the *i*th feature, estimated variance s_i^2

$$\hat{\sigma}_i^2 = s_i^2 = \frac{1}{n-K} \cdot \sum_{k=1}^{K} \sum_{j \in C_k} (x_{ij} - \bar{x}_{ik})^2$$

With $m_k = \sqrt{1/n_k - 1/n}$

d_{ik} is a *t*-statistic for the *i*th feature comparing class *k* to the average.

An Example

Three target classes with p = 6 features, and K = 3 classes {C1, C2, and C3}.

	F1	F2	F3	F4	F5	F6	Target
ex1	2	0.4	10	4	4	5	C1
ex2	4	0.5	10.5	4	3	4	C1
ex3	3	0.6	9.5	3	5	6	C1
ex4	11	0	0	7	20	6	C2
ex5	12	0.2	0	8	22	4	C2
ex6	22	0.4	5	10	31	4	C3
ex7	21	0.3	6	12	32	5	C3
ex8	20	0.6	5	11	33	6	C3

An Example

Compute the mean for each class and each feature

Mean	F1	F2	F3	F4	F5	F6
C1	3.0	0.5	10.0	3.67	4.0	5.0
C2	11.5	0.1	0.0	7.5	21.0	5.0
C3	21.0	0.43	5.33	11.0	32.0	5.0
gravity	11.88	0.38	5.75	7.38	18.75	5.0

Similar means for two classes (C1 and C3), different for class C2.

Same mean for all classes, F6 is an uninterested feature.

sum^2	F1	F2	F3	F4	F5	F6
	0.9	0.0173	0.233	0.633	1.2	1.2

m_k	**C1**	**C2**	**C3**
	0.678	0.79	0.678

An Example

For the p = 6 features, and K = 3 classes, the d_{ik} values.

d_{ik}	F1	F2	F3	F4	F5	F6
C1	-14.6	10.6	26.9	-8.65	-18.16	0
C2	-0.62	-20.07	-31.17	0.25	2.37	0
C3	14.97	4.97	-2.638	8.45	16.31	0

Shrink towards zero the uninteresting feature for some classes, where Δ is a parameter.

$$d'_{ik} = sign(d_{ik}) \cdot (|d_{ik}| - \Delta)_+$$

Set Δ = 5.

$(x)_+$ = x is x > 0,
= 0 otherwise

d'_{ik}	F1	F2	F3	F4	F5	F6
C1	-14.6	10.6	26.9	-8.65	-18.16	0
C2	0	-20.07	-31.17	0	0	0
C3	14.97	0	0	8.45	16.31	0

Nearest Shrunken Centroids

We can rewrite the previous equation as:

$$\bar{x}_{ik} = \bar{x}_i + m_k \cdot s_i \cdot d_{ik}$$

But the values d_{ik} must be shrunken towards zero, generating new d'_{ik} values

$$\bar{x}'_{ik} = \bar{x}_i + m_k \cdot s_i \cdot d'_{ik}$$

with $$d'_{ik} = sign(d_{ik}) \cdot (|d_{ik}| - \Delta)_+$$

and $(x)_+$ is the positive part if $x > 0$, 0 otherwise.
Small values of d_{ik} will be set to 0 (ako feature selection procedure).

Nearest Shrunken Centroids

Classification:

The new instance x* is compared with the profile (centroid) of each class.

The discriminate value for the class *k* is :

$$\delta_k(x^*) = \sum_{i=1}^{p} \frac{(x_i^* - \bar{x}_i')^2}{s_i^2} - 2 \cdot \log \pi_k$$

and the min value for *k* defined the class.

Where the second term is a correction for the class prior probability with $\sum_{i=1}^{K} \pi_i = 1$

An Example

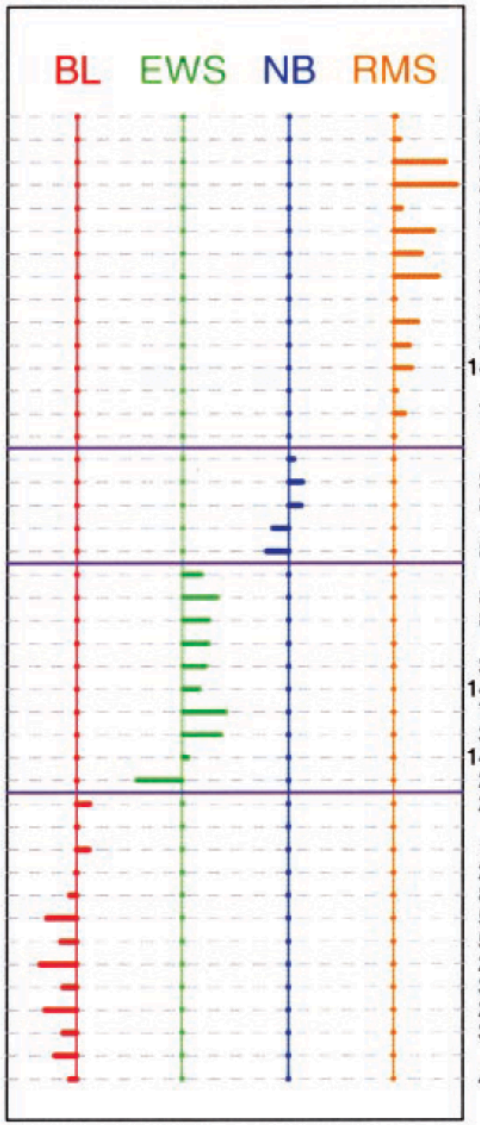
For the p = 6 features, and K = 3 classes, the x'_{ik} values or the three profiles (centroids).

x'_{ik}	F1	F2	F3	F4	F5	F6
C1	-10.04	8.70	44.78	1.21	-1.74	5
C2	11.60	-17.93	5.75	7.53	21.86	5
C3	34.41	9.44	0.65	25.45	37.15	5

Nearest Shrunken

Ignore some of the features (gene) for each class

d'_{ik} values for the four possible diseases



813841	tissue plasminogen activator
859359	quinone oxidoreductase homolog
207274	insulin-like growth factor 2
296448	insulin-like growth factor 2 (somatomedin A)
898219	homolog of mouse mesoderm specific transcript
784224	fibroblast growth factor receptor 4
796258	sarcoglycan alpha (dystrophin-associated glycoprotein)
244618	EST
789253	presenilin 2 (Alzheimer disease 4)
298062	troponin T2, cardiac muscle isoforms
461425	myosin MYL4
1409509	troponin T1, slow skeletal muscle isoforms
42558	L-arginine:glycine amidinotransferase
769716	neurofibromin 2 (mutated in neurofibromatosis type 2)
25725	farnesyl-diphosphate farnesyltransferase 1
44563	growth associated protein 43 (GAP43)
325182	N-cadherin (neuronal)
812105	ALL1-fused gene from chromosome 1q
41591	meningioma 1 (disrupted in balanced translocation)
810057	cold shock domain protein A
52076	neuroblastoma protein (NOE1)
866702	Fas-associated protein tyrosine phosphatase 1
814260	follicular lymphoma variant translocation protein 1
43733	glycogenin 2
357031	tumor necrosis factor alpha-induced protein 6
1435862	MIC2 surface antigen (CD99)
770394	IgG Fc fragment receptor transporter, alpha chain
377461	caveolin 1 (caveolae protein)
1473131	transducin-like enhancer of split 2
295985	EST
241412	E74-like factor 1 (ets domain transcription factor)
80109	major histocompatibility complex, class II, DQ alpha 1
183337	major histocompatibility complex, class II, DM alpha
233721	insulin-like growth factor binding protein 2
897788	receptor type protein tyrosine phosphatase F
563673	antiquitin 1
504791	glutathione S-transferase A4
212542	cDNA DKFZp586J2118
365826	growth arrest-specific protein 1
204545	EST
308163	EST
21652	alpha 1 catenin (cadherin-associated protein)
486110	profilin 2

Nearest Shru

Varying the amount of shrinkage and the effect on the number of features (genes).

cv in red
test in blue
(training in green)

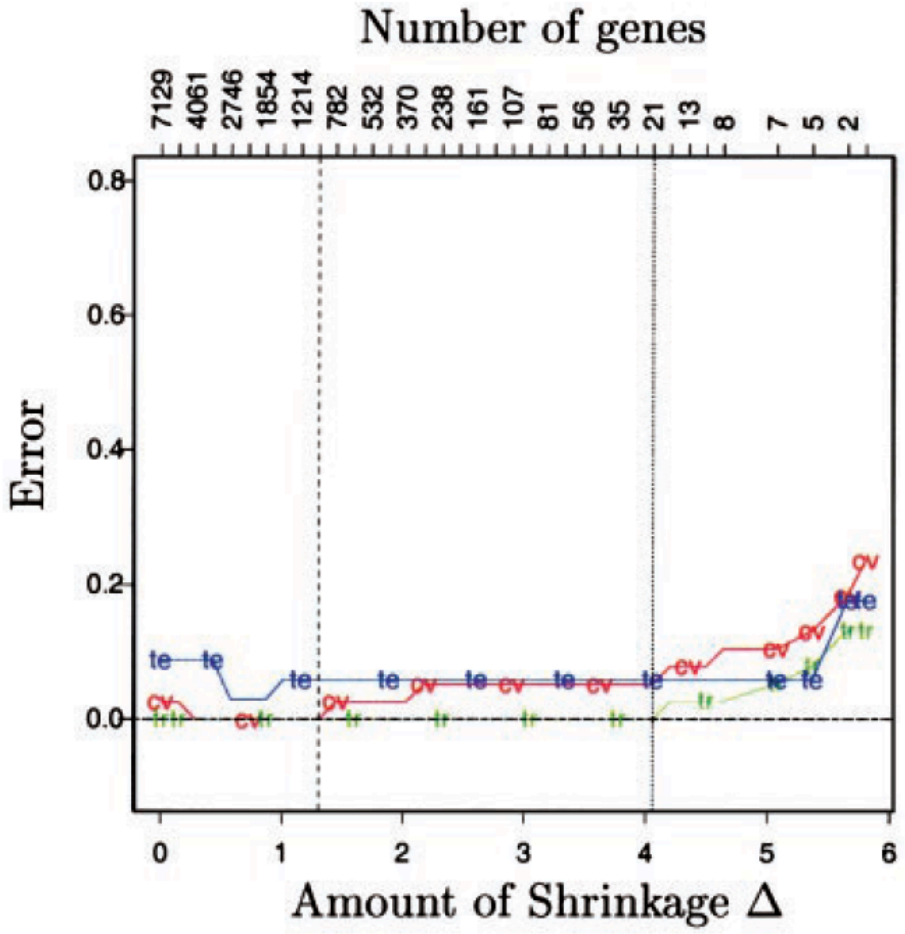


Fig. 6. Leukemia classification: training (tr, green), cross-validation (cv, red), and test (te, blue) errors. The value $\Delta = 4.06$ yields a subset of 21 genes.

Conclusion

K-NN is a non-parametric method (y = f(x)). Can generate very complex boundaries.

Simple and often very accurate.

Unlike parametric model:

- need a larger amount of data (see curve of dimensionality problem).

Two main strategies:

- Instance-based: consider all instances as points.
- Profile-based: generate a centroid for each category/class, then compute the distance with those centroids only.

Used for classification and regression!

Conclusion

K-NN requires a good distance measure must be selected: no clear theory.

Choice depending on attributes types (nominal, numerical, both, with missing values).

No feature selection: take *all* attributes (feature selection must be done before).

- Remedy: attribute selection or weights
- Apply Nearest Shrunken Centroids.

When many irrelevant attributes: poor results.

When too many redundant attributes: biased according to some attributes.

Computing intensive (many features, many observations). Use kD-tree (or variants).

K = 1? A clear risk of over-fitting.