HAP/LAP

Introduction to Machine Learning

Classification

Olatz Arbelaitz: olatz.arbelaitz@ehu.eus
www.aldapa.eus

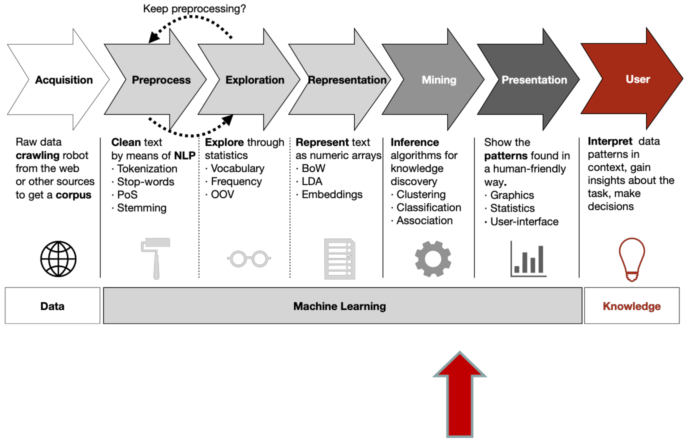
Erasmus Mundus

HAP/LAP

Topics

Classification

HAP/LAP



Classification

Classification process

HAP/LAP

- Division of the corpora (*Test options*)
  - train / test
  - Cross-validation
- **Classifier** (*Classify*)
  - Set parameters
- Evaluation (*Classifier Output*)
  - Confusion matrix
  - Precision/recall
  - Microaveraging/macroaveraging

Classifiers

HAP/LAP

Lazy: based on instances (knn) ← **Similarity function**
IB1, IBk

Bayes: based on Bayes theorem ← **Measuring probabilities**
NaiveBayes

Trees: based on trees
J48, RandomForest

Rules:
PART, OneR

Typical methods of Artificial Intelligence (Trees, Rules)

Functions: based on linear functions ← **Calculating functions**
MultilayerPerceptron,
SMO, Winnow

Meta: classifier combination
AdaBoostM1, vote, stacking ← **Combination of simple classifiers**

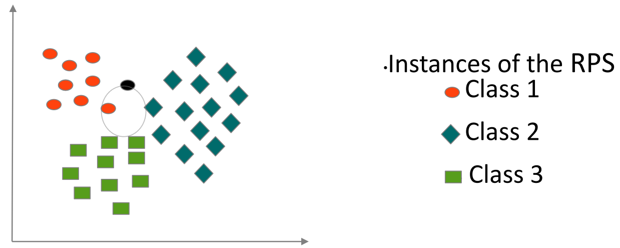
Based on instances: K-NN classifier

HAP/LAP

k Nearest Neigbour (**k**-NN)

k-NN is a **lazy** classifier. The classifier is based on the learning instances stored in memory, there is not model of the categories built

1-NN The class of the new instance to be classified will be the class of its nearest neighbour in the reference pattern set (RPS)



Based on instances: K-NN classifier

k Nearest Neighbour (**k**-NN): the majority class within the K nearest instances in the RPS is chosen

Procedure to classify new instances

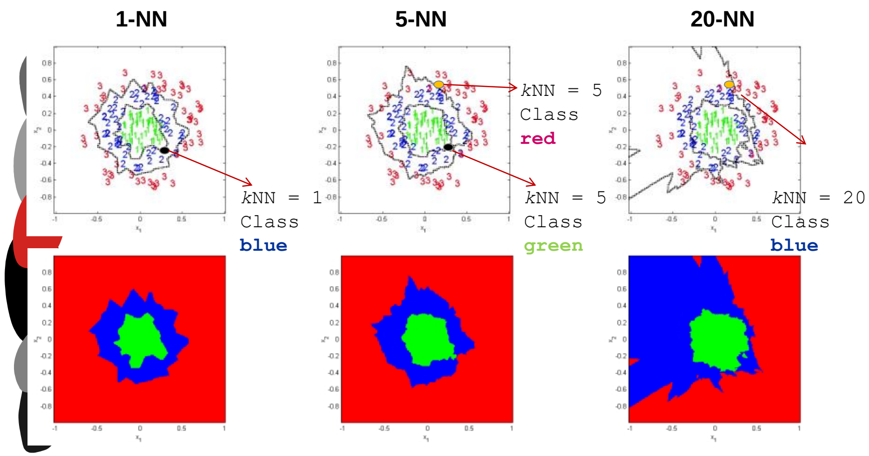
1.- Calculate the **distance** between the instance to classify and all the instances in the RPS
For example Euclidean distance (n dimensions)
2.- Select the **k nearest** instances (smallest distance)
3.- Assign as class the **majority class** within the k instances

Main parameters

Value of **k**, **number of examples** used to classify
Distance or similarity measure used to compare instances
Criteria **to select** the k nearest instances
Criteria **to decide** the class of the new instances

*k*NN versus *1*NN

HAP LAP



Based on instances

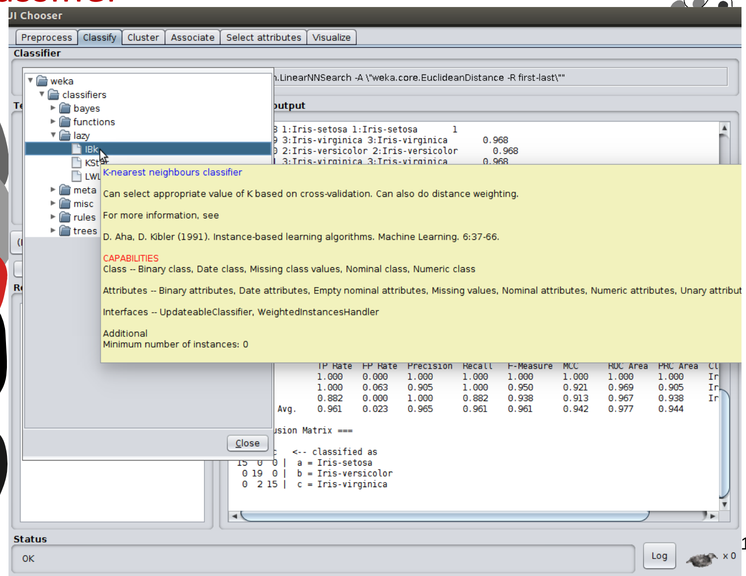
Weka: by default distance: **Euclidean**

measures how far/near the elements are in a vector space

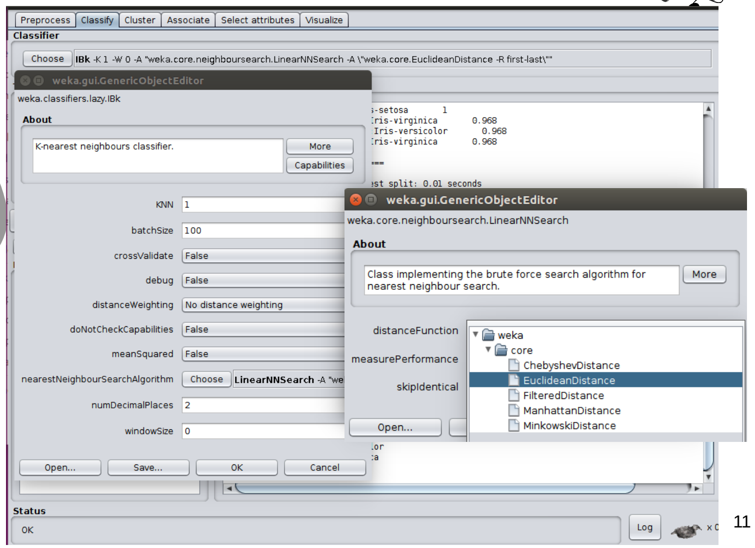
In **Lazy** classifiers:

- *IBk:* the *k* most similar. Normalizes numeric attributes (between 0-1) to apply distances. If K>1 we can weight distances.
  - Many distance options
- *windowSize*: can be used to limit the number of instances to be kept

Classifier



Classifier



k-NN. Example

HAP/LAP

Example (text classification)	**category**
d_1: ill(3), player, doctor(5), health(2)	**health**
d_2: nurse, play(3), doctor(4), health(2)	**health**
d_3: player(4), play(2), doctor(2), ball(3)	**sport**
d_t: ill(3), play(4), doctor(2), health	**???**

dictionary

Nurse
Ill
Player
Play
Doctor
Health
Ball

$$|d_j, d_z| = \sqrt{\sum_{i=1}^{n} (w_{ji} - w_{zi})^2}$$

d_1: 0 3 1 0 5 2 0 $|d_1 - d_t| =$

d_2: 1 0 0 3 4 2 0 $|d_2 - d_t| =$

d_3: 0 0 4 2 2 0 3 $|d_3 - d_t| =$

d_t: 0 3 0 4 2 1 0

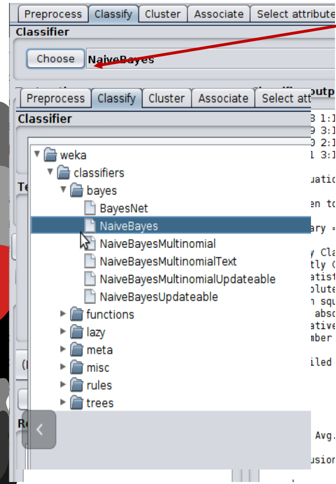
Knn. Assignment

Open ReutersGrainTrain

After a feature selection process, for the best set try k-NN algorithm and adjust K parameter

	k = 1	k = 3	k = 5	K =
RGT_(percentage_split)				
RGT_(CV-10 fold)				

Types of classifiers

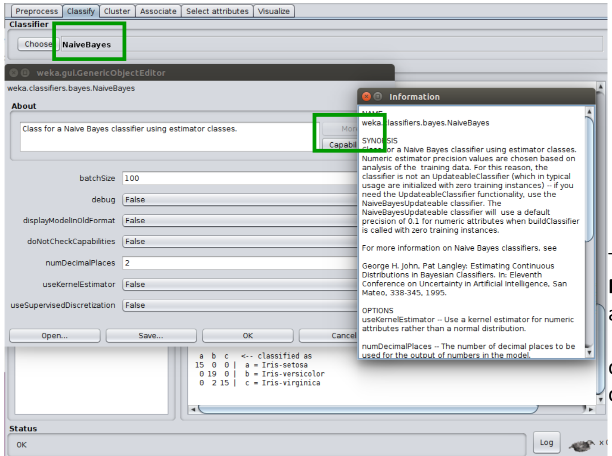


HAP/LAP

Lazy: based on instances (knn)
IB1, IBk
Bayes: based on Bayes theorem
NaiveBayes
Trees: based on trees
J48, NBTree, RandomForest
Rules:
PART, OneR
Functions: based on linear functions
MultilayerPerceptron,
SMO, Winnow
Meta: classifier combination
AdaBoostM1, vote, stacking

Naive Bayes

HAP/LAP



To see the concrete **Parameters** of a classifier and a short description about them click in the name of the classifier in bold and then click the "*More*" button

Example: document classification(spam)

http://en.wikipedia.org/wiki/Naive_Bayes_classifier#Document_Classification

Naive Bayes

Classifier based on Bayes Theorem making calculations simpler when:

The database has many features

There are not enough examples to calculate probabilities of all feature combinations.

Revision:

P(wi): prior probability of class wi
(quantifies the probability of a class without any extra information)

P(x|wi): density function probability conditioned to the class
(quantifies the probability of **x** having a concrete value knowing the class it belongs to)

P(wi|x): **posterior probability**
(quantifies the probability of an instance to belong to a class)

p(x): probability of instance x (unconditional)
(distribution of the instances)

Naive Bayes

HAP/LAP

Bayes theorem

$0 <= P(w_i) <= 1$ $\quad \Sigma P(w_i) = 1 \; (i=1,\dots,c)$ $\quad \Sigma P(x|w_i) = 1 \; (i=1,\dots,c)$

Given the prior probabilities and the density functions the **posterior probability** can be calculated **(≈classification):**

$$P(\omega_i|x) = \frac{P(x|\omega_i) * P(\omega_i)}{P(x)}$$

where

$$P(x) = \sum_{i=1}^{c} P(x|\omega_i) * P(\omega_i)$$

Classification with Naive Bayes

$$w_{NB} = \arg\max_{w_i \in C} P(w_i) \prod_{k=1..F} P(x_f \mid w_i)$$

$$\hat{y} = \arg\max_{k \in \{1,..,k\}} p(class_k) \prod_{i=1} p(x_i|class_k)$$

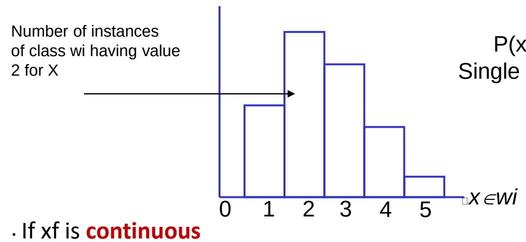
Naive Bayes

- Features are supposed to be **independent**

  As a consequence the probability for many features can be calculated as follows

  $P(x1,\dots,xF|wi) = P(x1|wi) \cdot \dots \cdot P(xf|wi)$

- If xf is **discrete** P(xf|wi) is estimated based on the relative instance frequency of class wi taking value xf (mass function, histograms)



- If xf is **continuous**
  1. Discretize and treat it as discrete
  2. estimate P(xf|wi) assuming a gaussian (normal) distribution- (only mean and variance required)

NaiveBayes

HAP LAP

Bayes theorem: calculating conditional probabilities

P(A|B) = P(A) x P(B|A) / P(B)

P(A): Prob. of A, P(A|B): Prob. of A given B is true and P(B|A): Prob. of B given A is true.

Example: is the web page containing word *mode* in French or in English?
And the one containing *maison*?

P(French|mode) = P(French) x P(mode|French) / P(mode).

Given a word (*mode*), the probability of the document being in *French* P(French|mode) is the following: probability of the document to be in *French* P(French) and the probability of having word *mode* if the document is in *French* P(mode/French) divided by the proportion of documents containing word *mode* P(mode).

P(French) = 0.08 P(mode|French) = 0.62 P(mode) = 0.15 | P(maison|French) = 0.92 P(maison) = 0.08

P(French|mode) = 0.08 x 0.62 / 0.15 = 0.33

P(French|maison) = 0.08 x 0.92 / 0.08 = 0.92

NaiveBayes

HAP LAP

- **Bag of Words assumption** => assume the position of the words in the document doesn't matter.
- **Conditional Independence** => Assume the feature probabilities **P(xi | cj)** are independent given the class **c**.
- To calculate **P(d | c) x P(c)**, we calculate P(xi | c) for each xi in **d**, and multiply them together.
- Then we multiply the result by P(c) for the current class. We do this for each of our classes, and choose the class that has the maximum overall value.

P (ci) = [N documents that have been classified as ci] / [N documents]

P (wi | cj) = [count(wi, cj)] / [Σw∈V count (w, cj)]
Laplace Smoothing
adding 1 to the numerator and modifying the denominator as such:
P (wi | cj) = [count(wi, cj) + 1] / [Σw∈V(count (w, cj) + 1)]
P (wi | cj) = [count(wi, cj) + 1] / [Σw∈V(count (w, cj)) + |V|]
where |V| is our vocabulary size (

NaiveBayes

	Doc	Words	Class
Training	1	Chinese Beijing Chinese	c
	2	Chinese Chinese Shanghai	c
	3	Chinese Macao	c
	4	Tokyo Japan Chinese	j
Test	5	Chinese Chinese Chinese Tokyo Japan	?

Priors:

$P(c) = \frac{3}{4}$

$P(j) = \frac{1}{4}$

Conditional Probabilities:

P(Chinese|*c*) = (5+1) / (8+6) = 6/14 = 3/7

P(Tokyo|*c*) = (0+1) / (8+6) = 1/14

P(Japan|*c*) = (0+1) / (8+6) = 1/14

P(Chinese|*j*) = (1+1) / (3+6) = 2/9

P(Tokyo|*j*) = (1+1) / (3+6) = 2/9

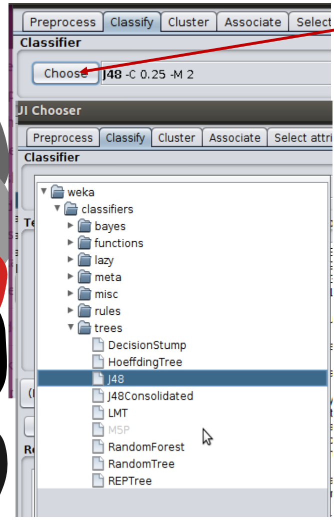
P(Japan|*j*) = (1+1) / (3+6) = 2/9

Choosing a class:

P(c|d5) ∝ 3/4 * (3/7)³ * 1/14 * 1/14 ≈ 0.0003

P(j|d5) ∝ 1/4 * (2/9)³ * 2/9 * 2/9 ≈ 0.0001

Types of classifiers



Lazy: based on instances (knn)
IB1, IBk

Bayes: based on Bayes theorem
NaiveBayes

Trees: based on trees
J48, NBTree, RandomForest

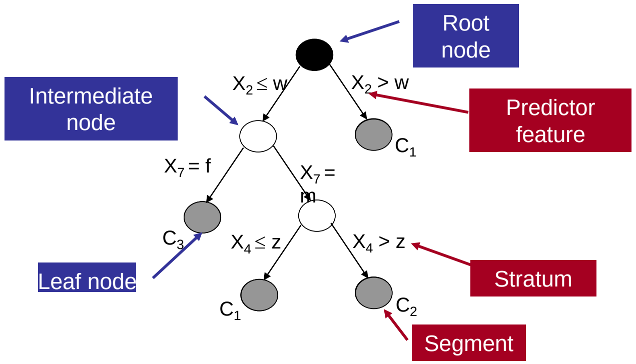
Rules:
PART, OneR

Functions: based on linear functions
MultilayerPerceptron,
SMO, Winnow

Meta: classifier combination
AdaBoostM1, vote, stacking

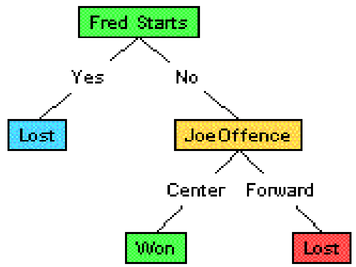
Decision trees

- Based on "Divide and conquer" algorithm
- A classifier in the form of a tree structure
  - Decision node: specifies a test on a single attribute
  - Leaf node: indicates the value of the target attribute
  - Arc/edge: split of one attribute
  - Path: a disjunction of test to make the final decision



Decision trees

- Decision trees classify instances or examples by starting at the root of the tree and moving through it until a leaf node.



- Problem: **Overfitting** (good in learning, worse in generalization)

Decision trees

Input: Training set E (labelled instances)
Output: Decision tree (T)
Algorithm

begin
  If all the examples in *E* are of the same category *Cj*
   then Result **simple node** labelled as *Cj*
  else
   begin
    Select a **feature** Xi with values *xi1,...,xil*
    Partition *E* in *E1,...,El* according to the values of *Xi*
    Build **subtrees** *T1,...,Tl* for*E1,...,El*
    The result is **a tree with root *Xi* and subtrees *T1,...,Tl***
    The branches between *Xi* and the subtrees are labelled with *xi1,...,xil*
   end
end

Decision trees

- **At each node:** selection of an attribute to split- choosing the most "useful" attribute for classifying examples.
  - Nominal features: as many branches as values
  - Numeric features: ≤, > // >, = , < // <, segment, >
- How to decide what is "useful"? Example: information gain
  - measures how well a given attribute separates the training examples according to their target classification
  - At each node, choose to divide the attribute with the largest information gain
- Stopping rule
  - Every attribute has already been included along this path through the tree, or
  - The training examples associated with this leaf node all have the same target attribute value (i.e., their entropy is zero).

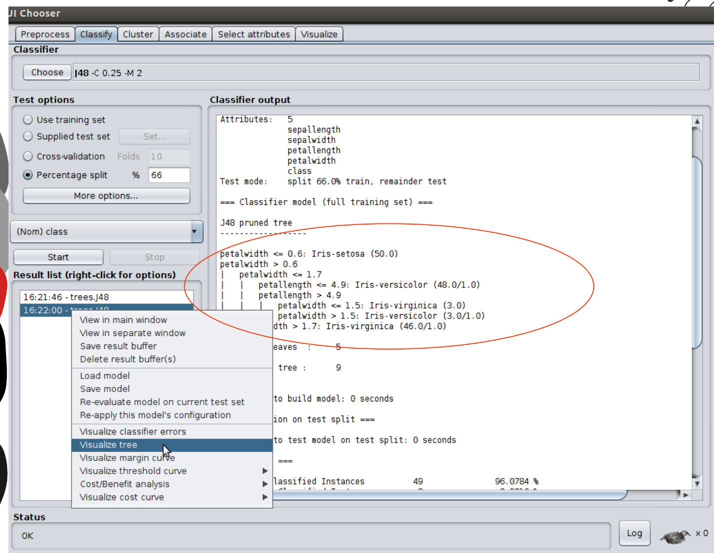
Classifier

Implementation of C4.5 (Quilan): J48

Two branches in numeric attributes (≤, >) see *iris.arff*

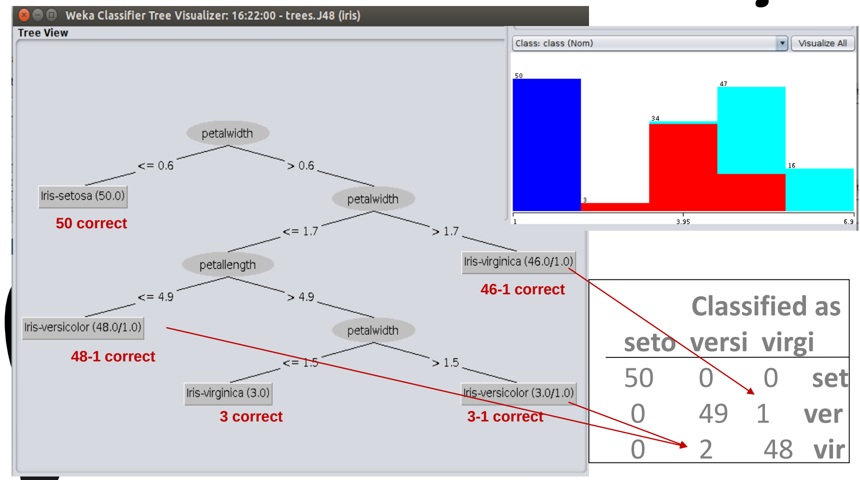
With nominal attributes: every value see *soybean.arff*

Decision trees: J48

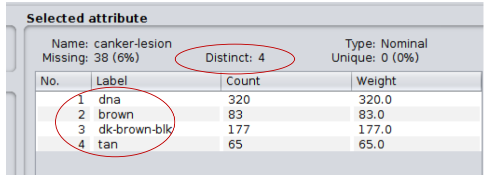


AP

Visualize tree

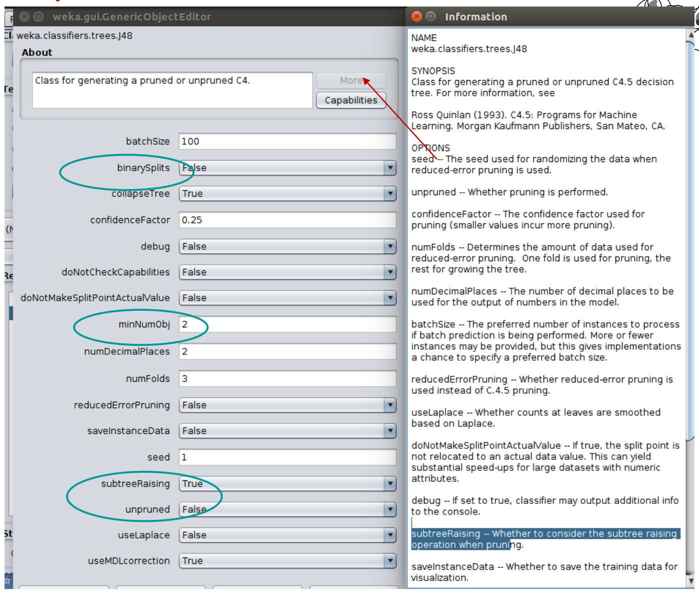


J48
soybean.arff



leafspot-size = lt-1/8
  canker-lesion = dna
| | leafspots-marg = w-s-marg
| | | seed-size = norm: bacterial-blight (21.0/1.0)
| | | seed-size = lt-norm: bacterial-pustule (3.23/1.23)
| | leafspots-marg = no-w-s-marg: bacterial-pustule (17.91/0.91)
| | leafspots-marg = dna: bacterial-blight (0.0)
| canker-lesion = brown: bacterial-blight
  canker-lesion = dk-brown-blk: phytophthora-rot (4.78/0.1
  canker-lesion = tan: purple-seed-stain (11.23/0.23)
....

J48. Options



J48. Options

Pruning

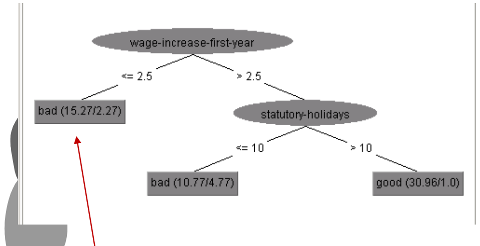
Prepruning (forward pruning)

decide where to cut when building the tree

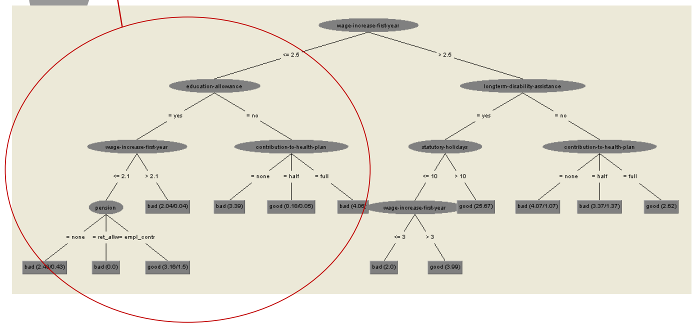
Postpruning (backward pruning)

- generate the tree and then analyze to decide where to cut
- most used option
- Two options in each node
  - subtree replacement → replace with leaves
  - subtree raising → remove subtrees and replace with the lower part → reclassify→ time

J48. Options. *labor.arff*

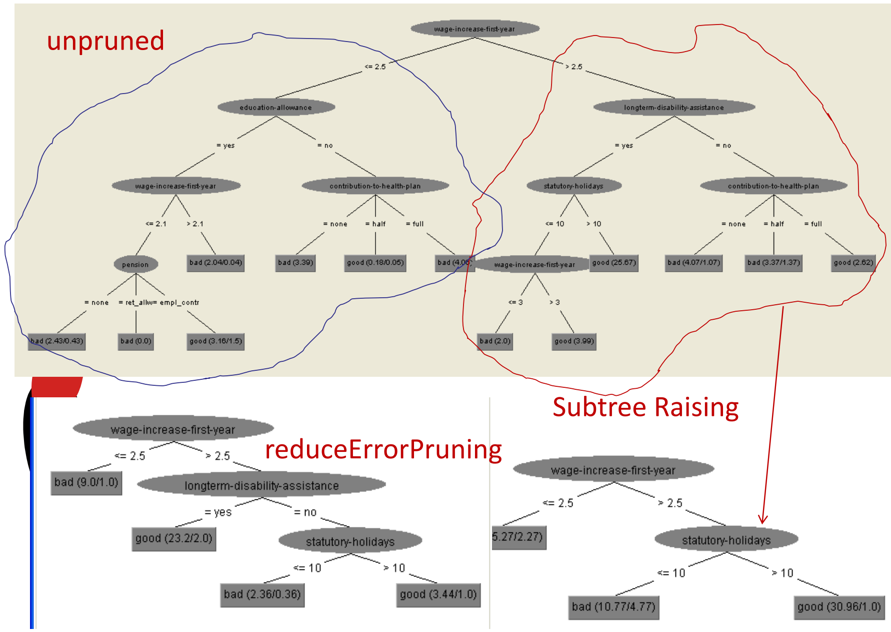


Subtree replacement
5 leaves bad → bad
2 leaves good
F-measure = 0.89



unpruned

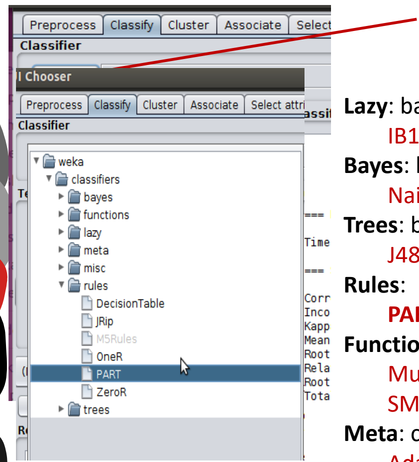
F-measure = 0.89



Other tree options

- ***Random Tree***:
  - K features are selected randomly in each node (try *soybean.arff*)
- ***ADTree***: *Freund, Y., Mason, L (1999)*
  - Two category branches
  - Boosting iterations: adds 3 nodes in each iteration
- ***NBTrees:*** *Ron Kohavi (1996)*
  - NB classiefiers in leaf nodes
- ***DecisionStump:***
  - Generates binary trees of a single level.
  - To use in Boosting methods
- ***Id3:*** *R. Quinlan (1986)*
  - Only nominal attributes
  - Information gain → Gain Ratio
- ***Random Forest***:
  - forest of random trees
  - Bagging (multiple classifier system)

Types of classifiers



HAP/LAP

Lazy: based on instances (knn)
IB1, IBk
Bayes: based on Bayes theorem
NaiveBayes
Trees: based on trees
J48, NBTree, RandomForest..
Rules:
PART, OneR....
Functions: based on linear functions
MultilayerPerceptron,
SMO, Winnow
Meta: classifier combination
AdaBoostM1, vote, stacking

Decision rules

- Divide-and-conquer technique
- Find rules that group instances of a class and discard those that are not of the class
- Overfitting (good in learning but not generalization capacity)

- Maximize the ratio: *p/t*
  t: number of examples covered by the rule and *p* those that are positive among them
- Information gain: *p[log p/t – log P/T]*
  gain with the new rule
  t and *p* the same, and *P* and *T* same value after introducing the new rule
- To introduce new rules:
  - as many positive examples as possible
  - as few negative as possible

Decision rules

- Rule induction

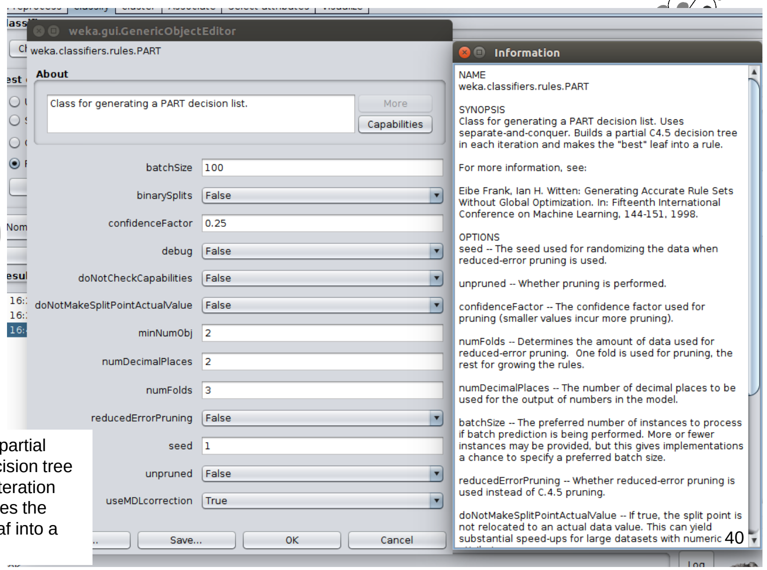
Example iris: 3 rules

```
if petalwidth <= 0.6 then Iris-setosa
else     if petalwidth <= 1.7 AND petallength <= 4.9
  then Iris-versicolor
          else Iris-virginica
```

Example TC

```
  cyclist & Sky & … then sport
else      if crisis & euro & … then economy
       else …
```


Decision rules: PART

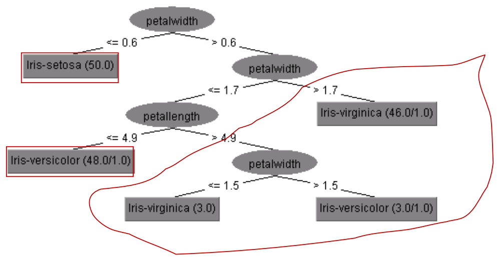


Builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule

Decision rules: PART

HAP/LAP

PART decision list------------------ 3 rules
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth <= 1.7 AND petallength <= 4.9: Iris-versicolor (48.0/1.0)
: Iris-virginica (52.0/3.0)

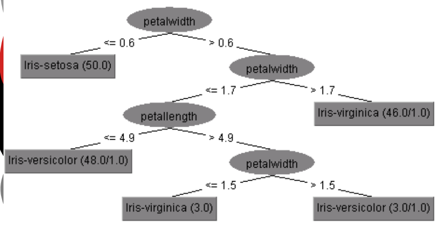


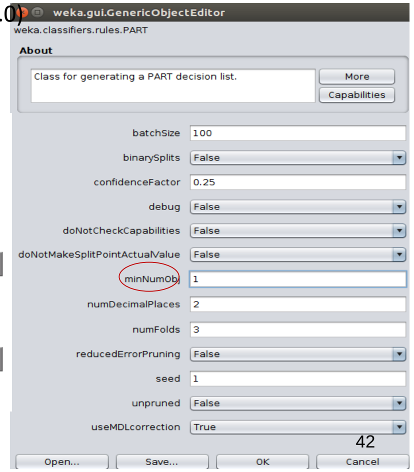
Iris-virginica

Decision rules: PART

HAP/LAP

petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 1.7 AND petallength > 4.8: Iris-virginica (43.0)
petallength <= 4.7 AND petalwidth <= 1.5: Iris-versicolor (42.0)
sepalwidth <= 3: Iris-virginica (11.0/4.0)
: Iris-versicolor (4.0)
Number of Rules : 5





Decision rules: other options

HAP/LAP

JRIP: similar to RIPPER (Repeated Incremental Pruning to Produce Error Reduction), in accuracy, number of rules and time. Not in memory

OneR: R.C. Holte (1993)

- Generates a single rule
- Prediction based on the feature with minimum error
- Discretizes numeric attributes
- Simple rules obtain often better results

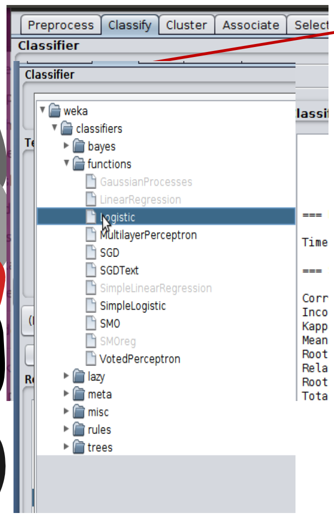
DecisionTable: Ron Kohavi (1995)

- Uses Best-first search to explore feature set
- Cross-validation for evaluation
- 1-nn can be used when instances can not be classified with the information in the table

Rules:
petalwidth class
===================
'(1.75-inf)' Iris-virginica
'(0.8-1.75]' Iris-versicolor
'(-inf-0.8]' Iris-setosa
===================

Types of classifiers

HAP/LAP



Lazy: based on instances (knn)
IB1, IBk
Bayes: based on Bayes theorem
NaiveBayes
Trees: based on trees
J48, NBTree, RandomForest..
Rules:
PART, OneR....
Functions: based on linear functions
MultilayerPerceptron,
SMO, Logistic
Meta: classifier combination
AdaBoostM1, vote, stacking

Functions

HAP/LAP

Classifiers that can be represented with mathematical equations:

Regression

To predict based on numeric classes and attributes
Ex.: cost of a flat based on size, number of bedrooms, ...

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Classification

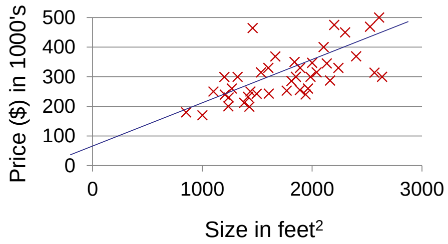
The values of the class are discrete

Functions

HAP/LAP

Example: cost of the flat according to size
A single attribute: size(x)

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…



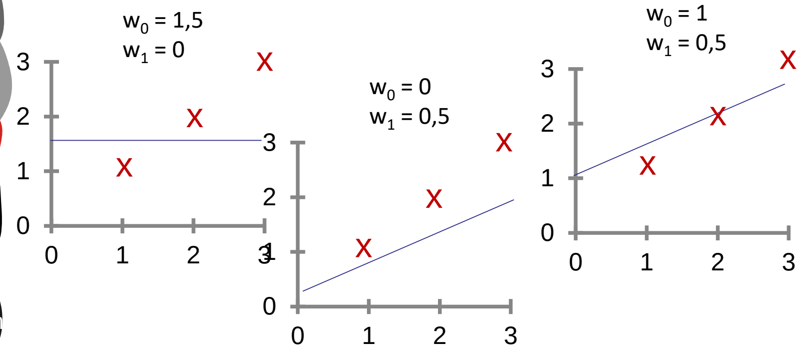
Linear function (hypothesis)

$h_w(x) = w_0 + w_1 x$ (w_i = weights)

Functions

HAP/LAP

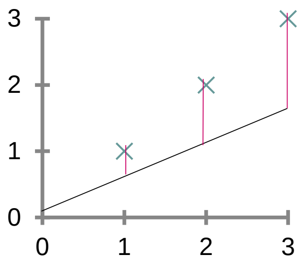
Example: $h_w(x) = w_0 + w_1 x$



Functions

HAP/LAP

How to calculate the cost of the prediction?



Linear function

$h_w(x) = w_0 + w_1 x_1$

w_i = weights

Cost function:

$$J(w_0, w_1) = 1/2m \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)})^2$$

m: number of instances

Functions

HAP/LAP

Linear regression

To predict with numeric classes and attributes.

- **The class** is represented as a linear combination of the features with predicted weights

$C = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$ weights are calculated from the training set

class weights features

- **The predicted class** for each example is calculated in the following way: ($x_0 = 1$)

$w_0 x_0^{(1)} + w_1 x_1^{(1)} + w_2 x_2^{(1)} + \dots + w_n x_n^{(1)} = \sum w_j x_j^{(1)}$

find w_j coefficients that minimize the squared difference between the predicted value and the real value

Predicted value

$$\sum_{i=1}^{m} (k^{(i)} - \sum_{j=0}^{n} w_j a_j^{(i)})^2$$

m: instances
n: number of features

Real value

Example: CPU.arff

Data

MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	class
125,	256,	6000,	256,	16,	128,	198
29,	8000,	32000,	32,	8,	32,	269
29,	8000,	32000,	32,	8,	32,	220
29,	8000,	32000,	32,	8,	32,	172

....

To calculate w coefficients, the following values need to be minimized:

$$(198 - \sum w_j x_j)^2 + (269 - \sum w_j x_j)^2 + (220 - \sum w_j x_j)^2 + (172 - \sum w_j x_j)^2 + \ldots$$

```
class =

      0.0491 * MYCT +
      0.0152 * MMIN +
      0.0056 * MMAX +
      0.6298 * CACH +
      1.4599 * CHMAX +
    -56.075
```


Functions: Artificial Neural Networks

Idea:

- model mathematically the human intellectual capacities.
- massively parallel computation schemas
- Unstable classifiers appropriate for multiple classifier systems

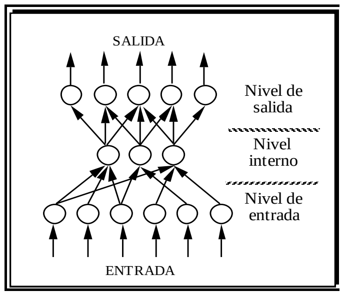
Universal approach property:

Some of the models, (Multilayer Perceptron (MLP), Radial Basis Function (RBF)), if an infinite number of patterns can be used, have the capacity to approach any discriminate function with a certain precision.

Functions: Artificial Neural Networks

Multilayer Perceptron (MLP)

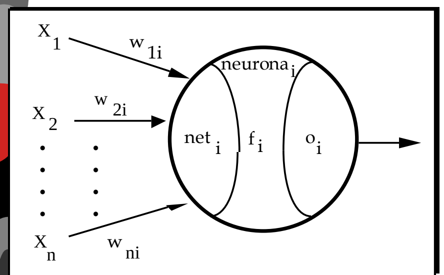
Is a feedforward network where every neuron in a layer is connected to every neuron in next layer. The structure would be the following:



Functions: Artificial Neural Networks

The basic structure of an ANN is a single neuron

When the ANN has a single neuron it is called simple linear perceptron.



$$o = f\left(\sum_{j=1}^{N} w_j * x_j + w_0\right)$$

f is an identity type function, sign, sigmoid, etc.

Functions

HAP/LAP

Perceptron

At the beginning, all the weights of the features of category c_i are identical: w_{ki}

For new examples to learn (d_j) the classifier classifies with the weights the classifier has and:

If the classification is correct: no action

If the classification is not correct :

If $d_j \in c_i$ then $w_{ki} := w_{ki} + \alpha$ $(\alpha > 0)$

If $d_j \notin c_i$ then $w_{ki} := w_{ki} - \alpha$ $(\alpha > 0)$

$(w_{ki} \rightarrow$ all t_k where $w_{kj} = 1)$

If the weight of the feature (w_{ki}) diminishes in the learning process, feature (t_k) is not useful for classification and it can be removed (on-the-fly term space reduction)

Perceptron. Example

- Document classification
  - d_i = … book presentation in Koldo Mitxelenan …
  - d_l = … Europes´ economy… the euro has risen
  - d_j = … the writer will talk about the book in Koldo Mitxelena
- Representation (lemmas)
  - d_i = … book present Koldo Mitxelena …
  - d_l = … Europe economy euro have rise…
  - d_j = … writer talk about book Koldo Mitxelena …
- Features:
  - **{writer, book, present, novel, … , read, Koldo, Mitxelena, have, Europe, economy, euro, rise, …}**
- Category:
  - **Culture**

Perceptron. Example

- {x_1=writer, x_2=book, x_3=present, x_4=novel, x_5=read, x_6=Koldo, x_7=Mitxelena, x_8=have , x_9=Europe, x_{10}=economy, x_{11}=euro, x_{12}=rise, …}

  $d_i = (x_1, x_2, \ldots, x_{12})$ = {0,1,1,0,0,1,1,0,0,0,0,0} **Culture**

  d_l = {0,0,0,0,0,0,0,1,1,1,1,1} **Economy**

  d_j = {1,1,0,0,0,1,1,1,0,0,0,0,0} **Culture**

- Algorithm

  Beginning $w_k = 0.1$; $x_0 = 1$; $\alpha = 0.8$

  $d_i \rightarrow f(x) = x_0w_0 + x_1w_1 + x_2w_2 + \ldots + x_{12}w_{12} = 0.5$ $f(x) > 0 \rightarrow$ culture YES

  $d_l \rightarrow f(x) = x_0w_0 + x_1w_1 + x_2w_2 + \ldots + x_{12}w_{12} = 0.6$ $f(x) > 0 \rightarrow$ culture YES

  Error $\rightarrow$ recalculating weights $(-\alpha)$ $w_0 - w_7 = 0.1$; $w_8 - w_{12} = -0.7$

  $d_j \rightarrow f(x) = x_0w_0 + x_1w_1 + x_2w_2 + \ldots + x_{12}w_{12} = -0.2$ $f(x) < 0 \rightarrow$ culture NO

  Error $\rightarrow$ recalculating weights $(+\alpha)$ $w_0 - w_2 = 0.9$; $w_3 - w_5 = 0.1$; $w_6 - w_7 = 0.9$; $w_8 = 0.1$; $w_9 - w_{12} = -0.7$

- Classifier

  hyperplane $\rightarrow$ weight vector

Functions

HAP/LAP

Winnow

To recalculate weights $(\alpha > 1, 0 < \beta < 1)$

Multiplication instead of addition subtraction

Positive winnow

If the classification is correct: no action

If the classification is not correct

If $d_j \in c_i$ then $w_{ki} := w_{ki} \times \alpha$ $(\alpha > 1)$

If $d_j \notin c_i$ then $w_{ki} := w_{ki} \times \beta$ $(0 < \beta < 1)$

Balanced winnow

Two weights for each term (+ and -)

If the classification is not correct:

if dj $\in$ ci then $w_{ki} := w_{ki}^{+} \times \alpha$ $(\alpha > 1)$

$w_{ki} := w_{ki}^{-} \times \beta$ $(0 < \beta < 1)$

if dj $\notin$ ci then $w_{ki} := w_{ki}^{-} \times \alpha$

$w_{ki} := w_{ki}^{+} \times \beta$

Functions

HAP/LAP

SVM (Support Vector Machine)

Problems of linear classifiers:

- The boundaries between two classes are linear
- Too simple for many practical applications

SVM- uses linear models to define non linear boundaries between classes

- Projects the input data using non linear mapping
- Converts the instance space to another space
- In the new space, the classes are linearly separable but in the original they are not.

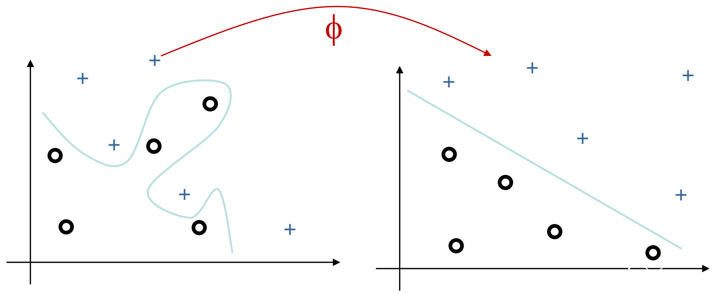
Linear model: *maximun margin hyperplane*

***Weka*: functions/SMO classifier** it is slow, CacheSize = 0

Call weka with: java -Xms512M -jar weka.jar

Functions

- Hyperplane discriminators: separate positive and negative examples by an hyperplane



ϕ function

Kernels

- **Kernel** Function defined in the new space

  any x, z ∈ X $\quad K = \langle \phi(x) \cdot \phi(z) \rangle$
- The Kernel represents the similarity between two objects
- Depending on function f different kernels can be generated different learning algorithms can be used
- Advantages:
  - Not necessary to maintain the feature vectors of the data
  - Adequate to work with complex spaces
  - Good result with high dimensional problems(TC)
  - Efficient ways of calculating internal product exist

Kernels

- Document classification
  - d_i = … book presentation in Koldo Mitxelenan …
  - d_l = … Europes' economy… the euro has risen
  - d_j = … the writer will talk about the book in Koldo Mitxelenan
- Representation (lemas)
  - d_i = … book present Koldo Mitxelena …
  - d_l = … Europe economy euro have rise…
  - d_j = … writer talk about book Koldo Mitxelena …
- Features:
  - **{writer, book, present, nobel, … , read, Koldo, Mitxelena, have, Europe, economy, euro, rise, …}**

Kernels

$d_1 = x \rightarrow \phi(x) = \{0,1,1,0,0,....,0,1,1,0,0,0,0,0\}$
$d_2 = z \rightarrow \phi(z) = \{1,1,0,0,0,....,0,1,1,1,0,0,0,0\}$
$d_3 = v \rightarrow \phi(x) = \{0,0,0,0,0,....,0,0,0,1,1,1,1,1\}$

Kernel function to measure similarity between d_1 and d_2

$d_1 = x \rightarrow \phi(x) = \{0,1,1,0,0,....,0,1,1,0,0,0,0,0\}$
$d_2 = z \rightarrow \phi(z) = \{1,1,0,0,0,....,0,1,1,1,0,0,0,0\}$

$\boxed{k(x,z) = \langle \phi(x) \cdot \phi(z) \rangle = \textbf{3 identical words}}$

Kernel function to measure similarity between d_2 and d_3

$d_2 = z \rightarrow \phi(z) = \{1,1,0,0,0,....,0,1,1,1,0,0,0,0\}$
$d_3 = x \rightarrow \phi(x) = \{0,0,0,0,0,....,0,0,0,1,1,1,1,1\}$

$\boxed{k(x,z) = \langle \phi(x) \cdot \phi(z) \rangle = 1 \rightarrow \textbf{a single word}}$

Kernels

- In the new space of the learning set, the **similarity** of each document with the rest of documents is represented.

$$K_{training} = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) & \dots & k(\mathbf{x}_1,\mathbf{x}_m) \\ k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) & \dots & k(\mathbf{x}_2,\mathbf{x}_m) \\ \dots & \dots & \dots & \dots \\ k(\mathbf{x}_m,\mathbf{x}_1) & k(\mathbf{x}_m,\mathbf{x}_2) & \dots & k(\mathbf{x}_m,\mathbf{x}_m) \end{bmatrix}$$

Kernel Gram Matrix

- It is not necessary to maintain all the original information

Kernels

- Three documents (d1, d2, and d3) and 13 words in the dictionary (t1, t2, … t12,t13)

term by document

$$D = [d1, d2, d3] = \begin{array}{c} \\ t1 \\ t2 \\ \dots \\ t13 \end{array} \begin{array}{c} \begin{array}{ccc} d1 & d2 & d3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ \dots & \dots & \dots \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

doc by doc

d1 and d2 share 3 words

$$G = D'D = \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ 0 & 0 & \dots & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ \dots & \dots & \dots \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 3 & 0 \\ 3 & 5 & 1 \\ 0 & 1 & 5 \end{pmatrix}$$

K (Gram)

- Size: 500 doc x 10.000 word → 5.000.000 elements in matrix D
  250.000 elements in matrix G

Kernels

- The inner product does not take into account the semantic relationship between terms:
  d1 = health, hospital, doctor… d2 = nurse, pills, …
  k(d1,d2) = <ϕ(d1) • ϕ(d2) > = 0 identical words
- It is possible to measure similarity between words instead of document similarity
  - $d1 = x \rightarrow \phi(x) = \{0,1,1,0,0,....,0,1,1,0,0,0,0,0\}$
  - $d2 = z \rightarrow \phi(z) = \{1,1,0,0,0,....,0,1,1,1,0,0,0,0\}$
  - $d3 = v \rightarrow \phi(x) = \{0,0,0,0,0,....,0,0,0,1,1,1,1,1\}$

term by term

0 docs with words t1 and t13

$$DD' = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ \dots & \dots & \dots \\ 1 & 1 & 0 \\ \dots & \dots & \dots \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ 0 & 0 & \dots & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & \dots & 0 \\ 1 & 2 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ 1 & 2 & & \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix} \quad T$$

2 docs with words t2 and t7

- Similarity between words. Different words but they appear together

Kernels

- Identity (linear kernel) $K(\mathbf{x},\mathbf{z}) = \langle \mathbf{x}\cdot\mathbf{z}\rangle$
- Polynomial (polinomy of degree d) $K(\mathbf{x},\mathbf{z}) = \left(\langle \mathbf{x}\cdot\mathbf{z}\rangle + c\right)^d$

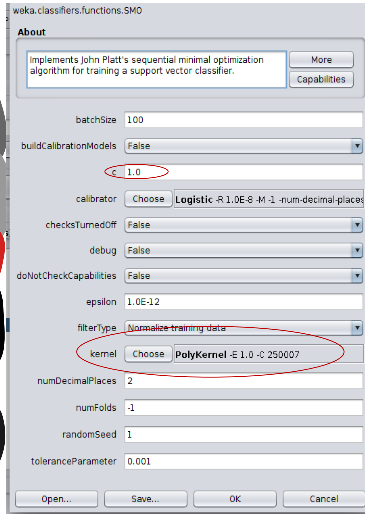
$$\langle \mathbf{x}\cdot\mathbf{z}\rangle^2 = \left(\sum_{i=1}^N x_i z_i\right)^2 = \left(\sum_{i=1}^N x_i z_i\right)\cdot\left(\sum_{j=1}^N x_j z_j\right) = \sum_{i=1}^N \sum_{j=1}^N x_i x_j z_i z_j = \sum_{i,j=1}^N (x_i x_j)(z_i z_j)$$

- Gaussian kernel (Radial Basis Function, RBF)

$$K(\mathbf{x},\mathbf{z}) = \exp\left(-\|\mathbf{x}-\mathbf{z}\|^2 / 2\sigma^2\right)$$

- Sigmoid $K(\mathbf{x},\mathbf{z}) = \tanh\left(\kappa\langle \mathbf{x}\cdot\mathbf{z}\rangle + \vartheta\right)$

SMO (sequential minimal optimization)



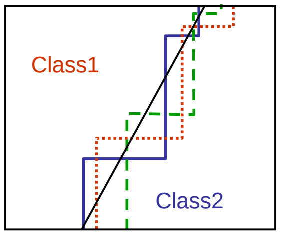
Weka
- Normalizes attributes
- Converts nominal attributes to binary

Select C
- small C : great error tolerance → many training examples missclassified
- bigger C: results will improve
- very big C : great importance to the training data → overfitting

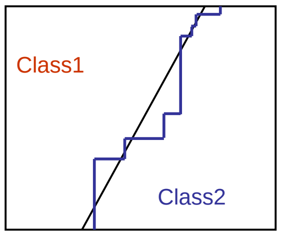
Select Kernel
- Polinomial (PolyKernel): exponent
- Gaussian kernel (RBF)

Multiple classifier systems

Motivation:
- **Statistical**: For small training samples the algorithm might find different hypothesis with the same performance. Taking into account several classifiers reduces the risk of selecting the wrong classifier.
- **Representational**: In many learning problems the objective function can not be represented for none of the classifiers.
- **DT example:**



Decision Boundary (DB) of 3 individual DTs



Corresponding DB of the voting classifier

Multiple classifier systems

T classifiers $\Phi_1, \Phi_2, \ldots, \Phi_T$, to solve the same task

Classification based on different learning methods

result: <u>combination</u> of the result of *k* classifiers

- Voting in classification tasks
- Average result if it is numeric

T classifiers of the same type

Bagging: same classifier different subsamples. All classifiers same weight

Boosting: complementary classifiers. (sequential learning)

Random subespace methods: classifiers built with different sets of features

Weka: Vote (select T different classifiers)

AdaBoost, Bagging, Random subespace (select a classifier)

Multiple classifier systems: bagging

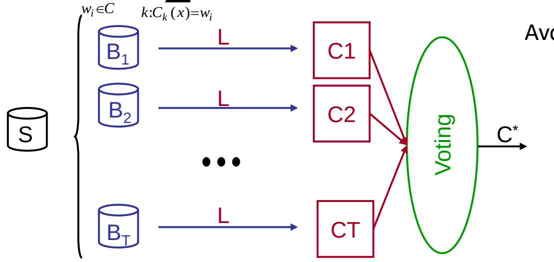
Bagging: Bootstrap aggregating, (Breiman en 1996)

Classifiers built using bootstrap samples (with replacement)
Generates T subsamples of size n' (n' ≤n) from S (learning sample of size n)
Builds T classifiers and combines the outputs

Final decision: voting of all individual classifiers for classification
Average for regression

$$C^*(x) = \arg\max_{w_i \in C} \sum_{k: C_k(x)=w_i} 1$$ /* The most voted class*/

Avoids overfitting



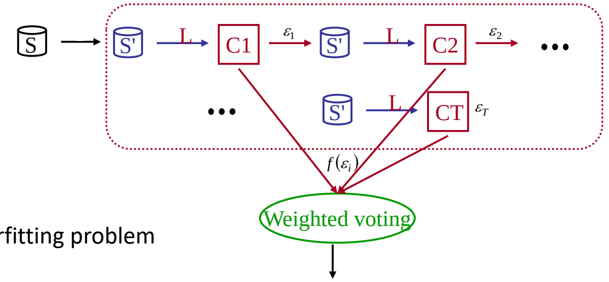
Multiple classifier systems: boosting

Boosting: proposed with the aim of stregthening of weak classifiers (Schapire 1990)
AdaBoost (Adaptive Boosting) introduced in by 1996 Freund&Schapire.

T classifiers are built sequentially. Each of the instances in the sample has a weight which changes depending on whether its classification is correct or incorrect.
resampling vs *reweighting*

- Weight increments for incorrectly classified examples
- Weight decrements for correctly classified examples

Final decision: weighted voting



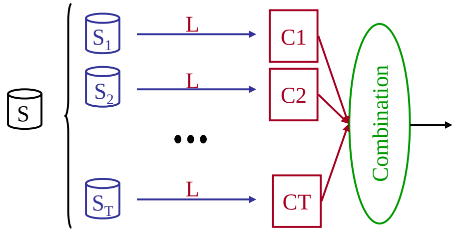
Overfitting problem

Multiple classifier systems: RSM

Random subespace method (RSM) (Ho 1995: *Random Decision Forests*)

To build individual classifiers based on different subespaces
Subespace: classification space based on a subset of the original set of features
The methodology is applicable to any type of classifier
Final decision: average of class memebership probabilities produced by each individual classifier.

$$C^*(x) = \arg\max_{w_i \in C} \frac{1}{T} \sum_{j=1}^{T} \hat{P}_j(w_i \mid x)$$



Example

- *soybean.arff* (%66)
  - J48; RandomTree; IB1
  - AdaBoost: J48, IB1, RTree;
  - Bagging: J48, IB1, RTree;
  - Vote (J48+RandomTree); (IB1+J48+RandomTree)

	Basic	AdaBoost	Bagging	Vote
IB1				
J48				
RandomTree				
J48+RTree				
IB1+J48+RTree				

Example

- *ReutersTrainGrain_WV.arff, soybean.arff, spam.arff* (%66)
  - IBK, Naive Bayes, J48, PART
  - F-measure/accuracy

	Ibk (k= ??)	Naive Bayes	J48	PART
RGT_WV	F-m	F-m	F-m	F-m
	Acc	Acc	Acc	Acc
Soybean	F-m	F-m	F-m	F-m
	Acc	Acc	Acc	Acc
spam	F-m	F-m	F-m	F-m
	Acc	Acc	Acc	Acc