Università degli Studi di Milano
Master Degree in Computer Science

# Information Management course

Teacher: Alberto Ceselli

Lecture 09: 13/11/2012

L. C. Molina, L. Belanche, A. Nebot
“Feature Selection Algorithms: A Survey and Experimental Evaluation”, IEEE ICDM (2002)

and

L. Belanche, F. Gonzales “Review and Evaluation of Feature Selection Algorithms in Synthetic Problems”, arXiv – available online (2011)

# Feature Selection Algorithms

- Introduction
- Relevance of a feature
- Algorithms
- Description of fundamental FSAs
- Generating weighted feature orders
- Empirical and experimental evaluation

# Algorithms for Feature Selection

- A FSA can be seen as a “computational approach to a definition of relevance”
  - Let X be the original set of features, $|X| = n$
  - Let $J(X')$ be an evaluation measure to be optimized:

    $J: X' \subseteq X \to \mathbb{R}$

  (1) Set $|X'| = m < n$; find $X' \subset X$ such that $J(X')$ is maximum

  (2) Set a value $J_0$; find $X' \subset X$ such that $|X'|$ is minimum, and $J(X') \geq J_0$

  - Find a compromise between (1) and (2)
- Remark: an optimal subset of features in not necessarily unique
- Characterization of FSAs
  - Search organization
  - Generation of successors
  - Evaluation measure

# Characterization of FSAs search organization

- General strategy with which the space of hypothesis is explored
- Search space: all possible subsets of features
- A partial order in the search space can be defined, as $S1 \prec S2$ if $S1 \subset S2$
- Aim of search: explore only a part of all subsets of features → for each subset relevance should be <u>upper</u> and <u>lower</u> bounded (estimates or heuristics)
  - Let L be a (labeled) list of (weighted) subsets of features → *states*
  - L maintains the current list of (partial) solutions, and the labels indicate the corresponding evaluation measure

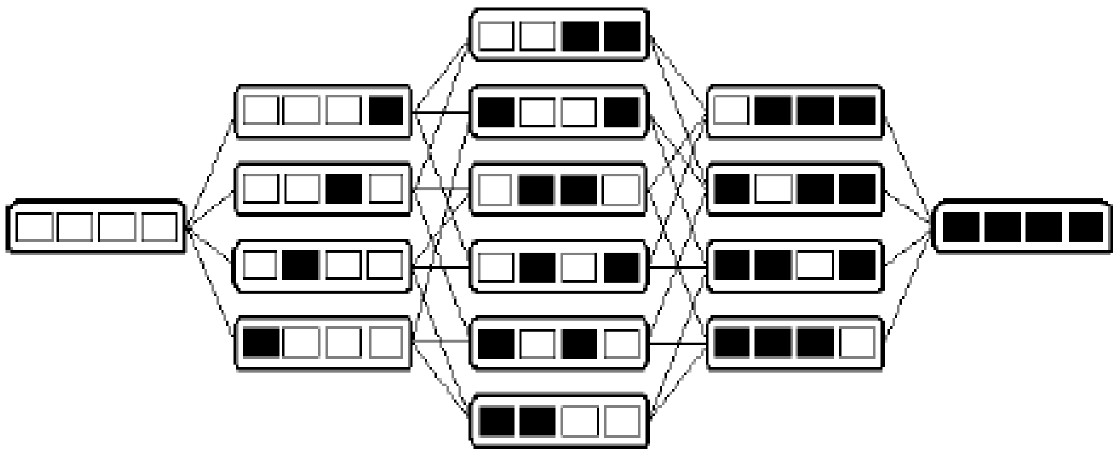


Figure 1. States in the binary search space involving 4 features. A black square represents the inclusion of a feature in the state and a white square represents its exclusion.

# Characterization of FSAs search organization

We consider three types of search:

- Exponential search ($|L| > 1$):
  - Search cost $O(2^n)$
  - Extreme case: exhaustive search
  - If given S1 and S2 with $S1 \subseteq S2$ then $J(S1) \geq J(S2)$

    → then J() is monotonic and branch-and-bound is optimal!
  - A* with heuristics is another option
- Sequential search ($|L| = 1$):
  - Start with a certain state and select a certain successor
  - Never backtrack
  - Search cost is polynomial, but no optimality guarantee
- Random search ($|L| > 1$):
  - Pick a state and change it somehow (local search)
  - Escape from local minima with random (worsening) moves

# Characterization of FSAs generation of successors

Five operators can be used to move from a state to the next

- Forward: start with X' = empty set
  - Given a state X', pick a feature $x \notin X'$ such that $J(X' \cup \{x\})$ is largest
  - Stop when $J(X' \cup \{x\}) = J(X')$, or $|X'|$ = certain card., or …
- Backward: start with X' = X
  - Given a state X', pick a feature $x \in X$ such that $J(X' \setminus \{x\})$ is largest
  - Stop when $J(X' \setminus \{x\}) = J(X')$, or $|X'|$ = certain card., or …
- Generalized Forward and Backward: consider <u>sets</u> of features for addition / removal at each step
- Compound: perform f consecutive forward moves and b consecutive backward moves
- Random

# Characterization of FSAs evaluation measures

- Several <u>problem dependent</u> approaches
- What counts is the relative values assigned to different subsets: e.g. classification
  - Probability of error: what's the behavior of a classifier using the subset of features?
  - Divergence: probabilistic distance among the class-conditional probability densities
  - Dependence: covariance or correlation coefficients
  - Interclass distance: e.g. dissimilarity
  - Information or Uncertainty: exploit entropy measurements on single features
  - Consistency: an inconsistency in X' and S is defined as two instances in S that are equal when considering only the features in X', but actually belong to different classes (aim: find the minimum subset of features leading to zero inconsistencies)

# Characterization of FSAs evaluation measures

- Example: Consistency
  - an inconsistency in X' and S is defined as two instances in S that are equal when considering only the features in X', but actually belong to different classes (aim: find the minimum subset of features leading to zero inconsistencies)

    $IC_{X'}(A) = X'(A) – \max_k X'_k(A)$

    $X'(A)$ = number of instances of S equal to A when only the features in X' are considered

    $X'_k(A)$ = number of instances of S <u>of class k</u> equal to A when only the features in X' are considered
  - Inconsistency rate:

    $IR(X') = \sum_{A \in S} IC_{X'}(A) / |S|$
  - $J(X') = 1 / ( IR(X') + 1 )$
- N.B. IR is a monotonic measure

# General schemes for feature selection

- Main forms of relation between FSA and “inducer”
  - Embedded scheme: the external method has its own FSA (e.g. decision trees or ANN)
  - Filter scheme: the feature selection takes place before the induction step
  - Wrapper scheme: FSA uses subalgorithms (e.g. learning algorithms) as internal routines

# General algorithm for feature selection

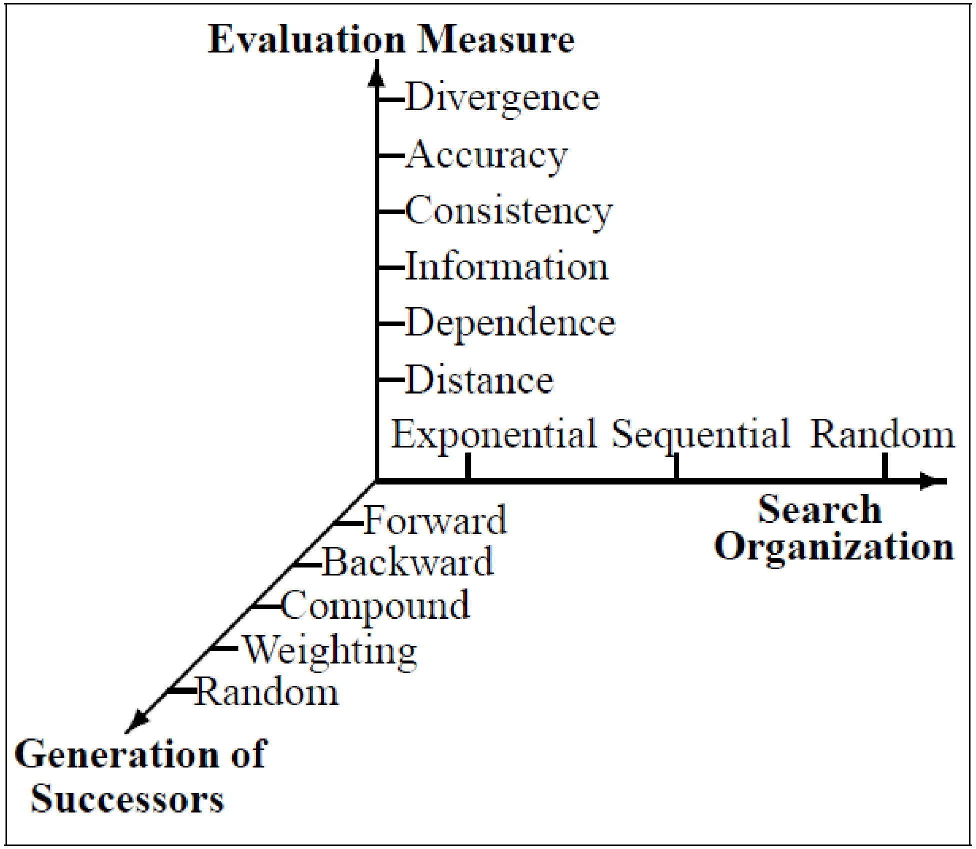
```
Input:
    S  – data sample with features X, |X| = n
    J  – evaluation measure to be maximized
    GS – successor generation operator
Output:
    Solution – (weighed) feature subset

L := Start_Point(X);
Solution := {best of L according to J};
repeat
    L := Search_Strategy(L, GS(J), X);
    X' := {best of L according to J};
    if J(X') ≥ J(Solution) or (J(X') = J(Solution)
            and |X'| < |Solution|)
    then Solution := X';
until Stop(J, L)
```

# Characterization of a FSA

Each algo can be represented as a triple <Org, GS, J>

- Org: search organization
- GS: Generation of Successors
- J: Evaluation measure



# Feature Selection Algorithms

- Introduction
- Relevance of a feature
- Algorithms
- Description of fundamental FSAs
- Generating weighted feature orders
- Empirical and experimental evaluation

# Las Vegas Filter (LVF) <random, random, any>

```
Input:
   max − the maximum number of iterations
   J − evaluation measure
   S(X) − a sample S described by X , |X| = n
Output:
   L − all equivalent solutions found

L    := []           // L stores equally good sets
Best := X            // Initialize best solution
J0   := J(S(X))      // minimum allowed value of J
repeat max times
    X' := Random_SubSet(Best)   // |X'| ≤ |Best|
    if J(S(X')) ≥ J0 then
        if |X'| < |Best| then
            Best := X'
            L := [X']            // L is reinitialized
        else if |X'| = |Best| then
            L := append(L, X')
        end
    end
end
end
```

# Las Vegas Incremental (LVI) <random, random, consist.>

```
Input:
  max – the maximum number of iterations
  J – evaluation measure
  S(X) – a sample S described by X, |X| = n
  p – initial percentage
Output:
  X' – solution found

S0 := portion(S, p)   // Initial portion
S1 := S \ S0          // Test set
J0 := J(S(X))         // Minimum allowed value of J
repeat forever
     X' := LVF (max, J, S0(X))
    if J(S(X')) ≥ J0 then stop
    else
        C := {elements in S1 with low
              contribution to J using X'}
        S0 := S0 ∪ C
        S1 := S1 \ C
    end
end
```

Rule of thumb: $p = 10\%$

# SBG/SFG <sequential, F/B, any>

**Input**:
$S(X)$ – a sample $S$ described by $X, |X| = n$
$J$ – evaluation measure
**Output**:
$X'$ – solution found

$X' := \emptyset$ *// forward*
$X' := X$ *// backward*
**repeat**
$x' := argmax\{J(S(X' \cup \{x\})) \mid x \in X \setminus X'\}$ *// forward*
$x' := argmax\{J(S(X' \setminus \{x\})) \mid x \in X'\}$ *// backward*
$X' := X' \cup \{x'\}$ *// forward*
$X' := X' \setminus \{x'\}$ *// backward*
**until** no improvement in $J$ in last $j$ steps
**or** $X' = X$ *// forward*
**or** $X' = \emptyset$ *// backward*

# SBG/SFG <sequential, F/B, any>

**Input**:
$S(X)$ – a sample $S$ described by $X, |X| = n$
$J$ – evaluation measure
**Output**:
$X'$ – solution found

$X' := \emptyset$ *// forward*
$X' := X$ *// backward*
**repeat**
$x' := argmax\{J(S(X' \cup \{x\})) \mid x \in X \setminus X'\}$ *// forward*
$x' := argmax\{J(S(X' \setminus \{x\})) \mid x \in X'\}$ *// backward*
$X' := X' \cup \{x'\}$ *// forward*
$X' := X' \setminus \{x'\}$ *// backward*
**until** no improvement in $J$ in last $j$ steps
**or** $X' = X$ *// forward*
**or** $X' = \emptyset$ *// backward*

# Focus <exponential, forward, consist.>

```
Input:
   S(X) – a sample S described by X, |X| = n
   J – evaluation measure (consistency)
   J0 – minimum allowed value of J
Output:
   X' – solution found

for i ∈ [1..n] do
    for each X' ⊂ X , with |X'| = i do
          if J(S(X')) ≥ J0 then stop
    end
end
```

# Sequential Floating FS <exponential, F+B, consist.>

**Input:**

$S(X)$ – a sample $S$ described by $X, |X| = n$

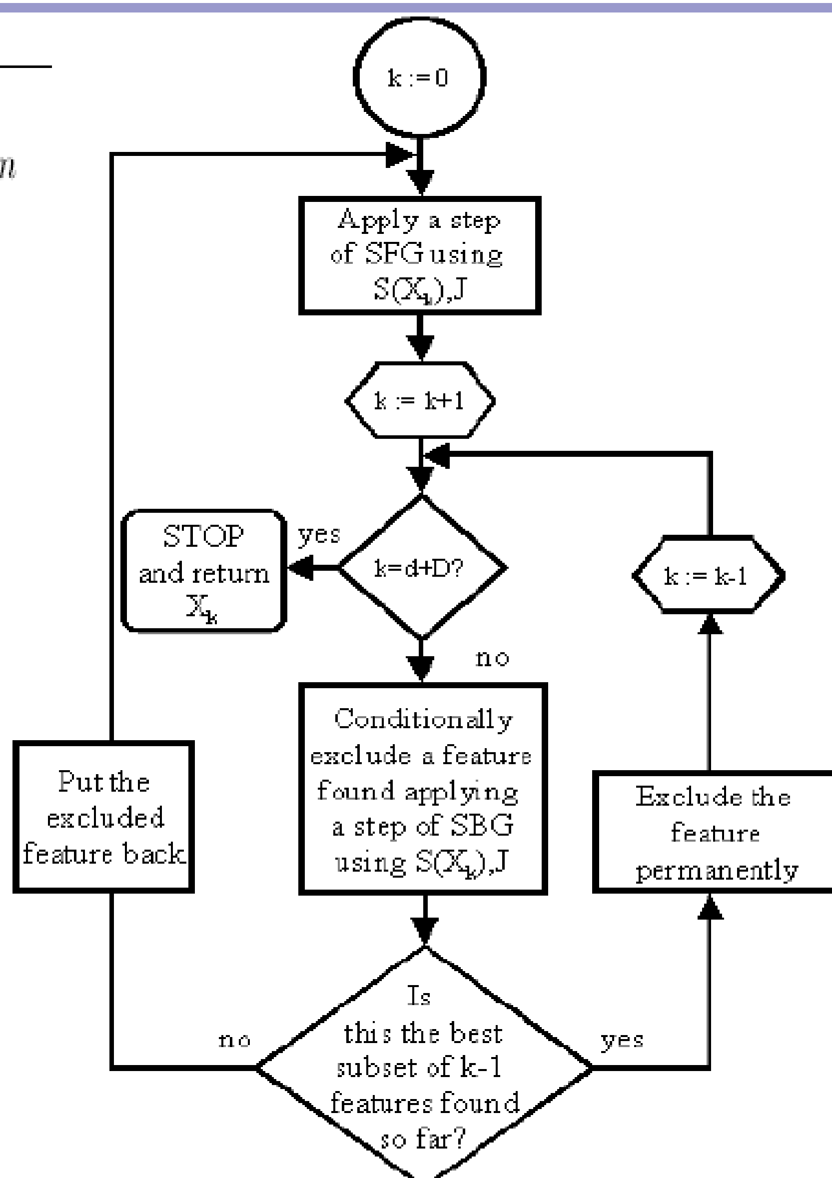
$J$ – evaluation measure

$d$ – desired size of the solution

$D$ – maximum deviation allowed with respect to $d$

**Output:**

solution of size $d \pm D$



# (Auto) branch&bound <exponential,backward,monotonic>

```
Input:
  S(X) − a sample S described by X, |X| = n
  J − evaluation measure (monotonic)
Output:
  L − all equivalent solutions found

procedure ABB (S(X): sample; var L′: list
    of set)
  for each x in X do
    enqueue(Q, X \ {x})  // remove a feature at a time
  end
  while not empty(Q) do
    X′ := dequeue(Q)
    // X′ is legitimate if it is not a subset of a pruned state
    if legitimate(X′) and J(S(X′)) ≥ J0 then
      L′ := append(L′, X′)
      ABB(S(X′), L′)
    end
  end
end

begin
   Q := ∅            // Queue of pending states
   L′ := [X]         // List of solutions
   J0 := J(S(X))     // Minimum allowed value of J
   ABB (S(X), L′)    // Initial call to ABB
   k := smallest size of a subset in L′
   L := set of elements of L′ of size k
end
```

# Quick branch&bound <rndm/exp,rndm/back,monotonic>

- Use LVF to find a good solution
- Use ABB to explore efficiently the remaining search space

# Feature Selection Algorithms

- Introduction
- Relevance of a feature
- Algorithms
- Description of fundamental FSAs
- Generating weighted feature orders
- Empirical and experimental evaluation

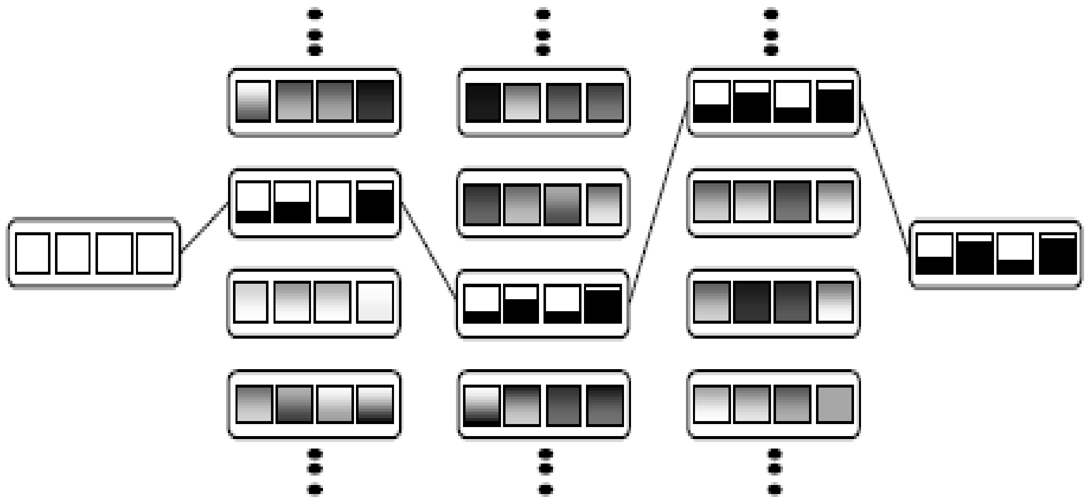


Figure 2. A path of states in the continuous search space involving 4 features. Relevances are represented as a degree of filling.

# Relief <random, weighting, distance>

```
Input:
  p – sampling percentage
  d – distance measure
  S(X) – a sample S described by X, |X| = n
Output:
  W – array of feature weights

initialize W[] to zero
do p|S| times
    A := Random_Element (S)
    A_nh := Near–Hit (A, S)
    A_nm := Near–Miss (A, S)
    for each i ∈ [1..n] do
      W[i] := W[i] + d_i(A, A_nm) − d_i(A, A_nh)
    end
end
```

Closest element to A in S in the same (hit) or a different (miss) class