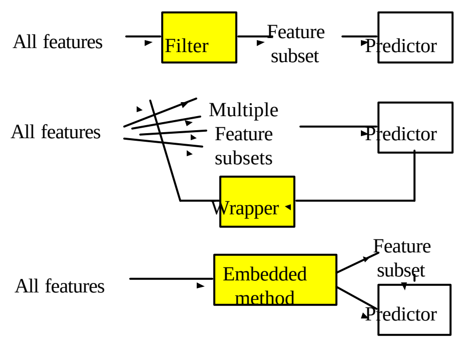
Lecture 9: Embedded Methods

Isabelle Guyon *guyoni@inf.ethz.ch*
André Elisseeff *AEL@zurich.ibm.com*

Chapter 5: Embedded methods

Filters, Wrappers, and Embedded methods



Filters

Methods:

- Criterion: Measure feature/feature subset "relevance"
- Search: Usually order features (individual feature ranking or nested subsets of features)
- Assessment: Use statistical tests

Results:

- Are (relatively) robust against overfitting
- May fail to select the most "useful" features

Wrappers

Methods:

- Criterion: Measure feature subset "usefulness"
- Search: Search the space of all feature subsets
- Assessment: Use cross-validation

Results:

- Can in principle find the most "useful" features, but
- Are prone to overfitting

Embedded Methods

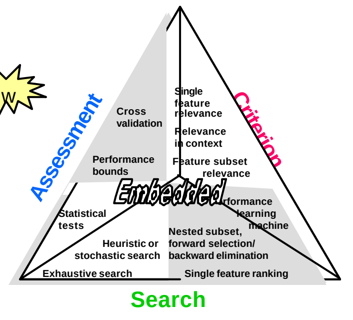
Methods:

- Criterion: Measure feature subset "usefulness"
- Search: **Search guided by the learning process**
- Assessment: Use cross-validation

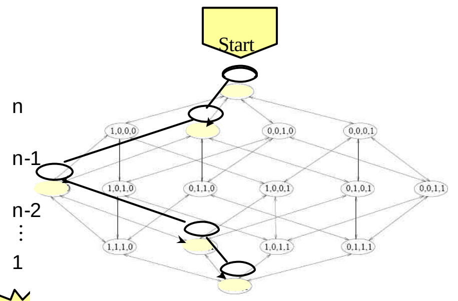
Results:

- Similar to wrappers, but
- Less computationally expensive
- Less prone to overfitting

Three "Ingredients"



Forward Selection



Guided search: we do not consider alternative paths.

Forward Selection with GS

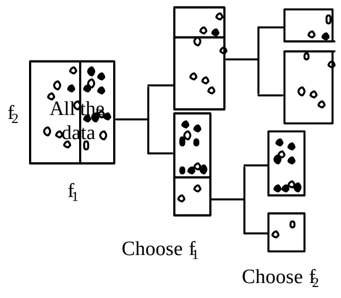
Stoppiglia, 2002. Gram-Schmidt orthogonalization.

- Select a first feature $X_{?(1)}$ with maximum cosine with the target $\cos(\mathbf{x}_i, \mathbf{y}) = \mathbf{x}.\mathbf{y}/||\mathbf{x}|| \; ||\mathbf{y}||$
- For each remaining feature X_i
  - Project X_i and the target Y on the null space of the features already selected
  - Compute the cosine of X_i with the target in the projection
- Select the feature $X_{?(k)}$ with maximum cosine with the target in the projection.

Embedded method for the linear least square predictor

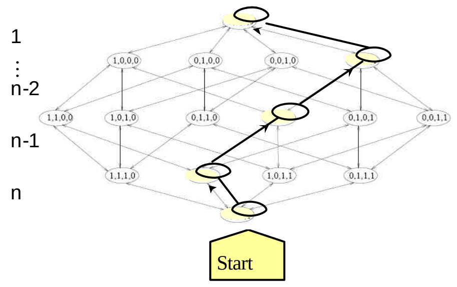
Forward Selection w. Trees

- Tree classifiers,
  like *CART (Breiman, 1984)* or *C4.5 (Quinlan, 1993)*



At each step, choose the feature that "reduces entropy" most. Work towards "node purity".

Backward Elimination



Backward Elimination: RFE

RFE-SVM, Guyon, Weston, et al, 2002

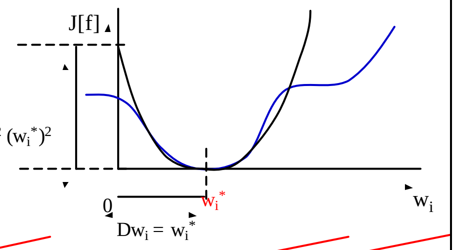
Start with all the features.

- Train a learning machine f on the current subset of features by minimizing a risk functional J[f].
- For each (remaining) feature X_i, estimate, without retraining f, the change in J[f] resulting from the removal of X_i.
- Remove the feature $X_{?(k)}$ that results in improving or least degrading J.

Embedded method for SVM, kernel methods, neural nets.

OBD *(LeCun et al, 1990)*

$DJ \cong ½\, \partial^2 J/\partial w_i^2\, (w_i^*)^2$



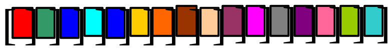
$$DJ = \cancel{\Sigma_i \partial J/\partial w_i\, Dw_i} + ½\, \Sigma_i\, \partial^2 J/\partial w_i^2\, (Dw_i)^2 + \cancel{\text{cross-terms}} + \cancel{O(\|D\mathbf{w}\|^3)}$$

Simple case: linear classifier + J quadratic form of $\mathbf{w} \Rightarrow DJ \propto w_i^2$

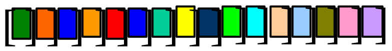
RFE for ridge regression and SVM: remove input with smallest w_i^2

Nested Subset Methods

- Forward selection



- Backward elimination



- Feature ranking (filters)

Complexity Comparison

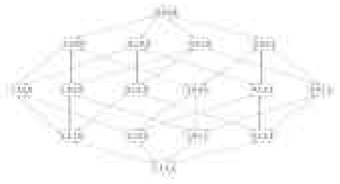
Generalization_error ≤ Validation_error + ε(C / m)

Method	**Number of subsets tried**	**Complexity C**
Exhaustive search wrapper	2^n	n
Nested subsets greedy wrapper	n(n+1)/2	log n
Feature ranking or embedded methods	n	log n

m: number of validation examples, n: number of features.

Scaling Factors

Idea: Transform a discrete space into a continuous space.



$s = [\sigma_1, \sigma_2, \sigma_3, \sigma_4]$

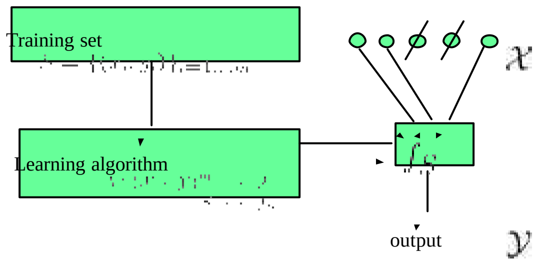
• Discrete indicators of feature presence: $\sigma_i \in \{0, 1\}$

• Continuous scaling factors: $\sigma_i \in$ IR

Now we can do gradient descent!

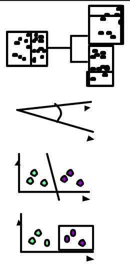
Embedded methods (alternative definition)

- Definition: an embedded feature selection method is a *machine learning algorithm* that returns a model using a limited number of features.



Examples

- Forward selection with Decision trees
- Forward selection with Gram-Schmidt
- Any algorithm producing a model where "sensitivity" analysis can be done:
  - Linear system: remove feature *i* if w_i is smaller than a fixed value.
  - Others, e.g. parallelepipeds: remove dimension where width is below a fixed value.



Note: embedded methods use the specific structure of the model returned by the algorithm to get the set of "relevant" features.

Design strategies

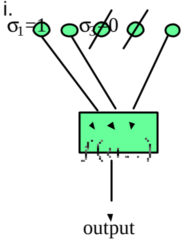
- As previously suggested: use tricks and intuition. Might work but difficult. Still can produce very smart algorithms (decision trees).
- Other means: interpret feature selection as a model selection problem. In that context, we are interested in finding the set of features such that the model is the "best".

Feature selection as model selection - 1

- Let us consider the following set of functions parameterized by α and where σ 2 $\{0,1\}^n$ represents the use (σ_i=1) or rejection of feature i.

$$\sigma \circ x = (\sigma_1 x_1, \dots, \sigma_n x_n)$$

$$\begin{aligned} f : \Lambda \times \mathbb{R}^n &\longrightarrow \mathbb{R} \\ (\alpha, \sigma \circ x) &\longmapsto f(\alpha, \sigma \circ x) \end{aligned}$$



Example (linear systems, α=w):

$$w.x + b = \sum_{i=1}^{n} w_i x_i \sigma_i = \sum_{\sigma_i \neq 0} w_i x_i$$

Feature selection as model selection - 2

- We are interested in finding α and σ such that the generalization error is minimized:

$$\min_{\sigma, \alpha} R(\alpha, \sigma)$$

where

$$R(\alpha, \sigma) = \int L(f(\alpha, \sigma \circ x), y) dP(x, y)$$

Sometimes we add a constraint: # non zero σ_i's · s_0

Problem: the generalization error is not known…

Feature selection as model selection - 3

- The generalization error is not known directly but bounds can be used.
- Most embedded methods minimize those bounds using different optimization strategies:
  - Add and remove features
  - Relaxation methods and gradient descent
  - Relaxation methods and regularization

Example of bounds (linear systems):

Non separable $R(w,\sigma) \leq \frac{1}{m}\sum_{k=1}^{m} L(w.x_k + b, y_k) + O(\frac{r\|w\|}{\sqrt{m}})$

Linearly separable $R(w,\sigma) \leq O(\frac{r^2\|w\|^2}{m})$

Feature selection as model selection -4

- How to minimize $\min_{\sigma,\alpha} R(\alpha,\sigma)$?

Most approaches use the following method:

1. Set $\sigma = (1, .., 1)$
2. Compute $\alpha^* = \arg\min_\alpha R(\alpha,\sigma)$
3. Compute $\sigma^* = \arg\min_\sigma R(\alpha^*,\sigma)$
4. Set $\sigma \leftarrow \sigma^*$ and go back to 2.

This optimization is often done by relaxing the constraint
σ 2 $\{0,1\}^n$
as σ 2 $[0,1]^n$

Add/Remove features 1

- Many learning algorithms are cast into a minimization of some regularized functional:

$$\underbrace{\min_{\alpha} R(\alpha,\sigma)}_{G(\sigma)} = \min_{\alpha} \sum_{k=1}^{m} L(f(\alpha, \sigma \circ x_k), y_k) + \Omega(\alpha)$$

Empirical error

Regularization capacity control

- What does $G(\sigma)$ become if one feature is removed?
- Sometimes, G can only increase… (e.g. SVM)

Add/Remove features 2

- It can be shown (under some conditions) that the removal of one feature will induce a change in G proportional to:

$$\sum_{k=1}^{m} \left(\frac{\partial f}{\partial x^i}\right)^2 (\alpha, x_k)$$

Gradient of f wrt. i^th feature at point x_k

- Examples: SVMs $\frac{\partial f}{\partial x^i} \propto w_i$

! RFE ($\Omega(\alpha) = \Omega(w) = \sum_i w_i^2$)

Add/Remove features - RFE

- Recursive Feature Elimination

1. Set $F = \{1, \dots, n\}$
2. Get w^* as the solution on a SVM on the data set restricted to features in F
3. Select top features as ranked by the $|w_i^*|$'s
4. Back to 2.

Minimize estimate of R(α, σ) wrt. α

Minimize the estimate R(α,σ) wrt. σ and under a constraint that only limited number of features must be selected

Add/Remove feature summary

- Many algorithms can be turned into embedded methods for feature selections by using the following approach:

1. Choose an objective function that measure how well the model returned by the algorithm performs
2. "Differentiate" (or sensitivity analysis) this objective function according to the σ parameter (i.e. how does the value of this function change when one feature is removed and the algorithm is rerun)
3. Select the features whose removal (resp. addition) induces the desired change in the objective function (i.e. minimize error estimate, maximize alignment with target, etc.)

What makes this method an 'embedded method' is the use of the structure of the learning algorithm to compute the gradient and to search/weight relevant features.

Add/Remove features when to stop

- When would you stop selecting features?
  - When objective function has reached a plateau?
    - What happens for the bound $r^2||w||^2$ when features are removed?
  - Using a validation set?
    - What size should you consider?
  - Don't stop, just rank features?

Gradient descent - 1

- How to minimize $\min_{\sigma,\alpha} R(\alpha, \sigma)$?

Most approaches use the following method:

1. Set $\sigma = (1, \dots, 1)$
2. Compute $\alpha^* = \arg\min_\alpha R(\alpha, \sigma)$
3. Compute $\sigma^* = \sigma - \lambda \nabla_\sigma R(\alpha^*, \sigma)$
4. Set $\sigma \leftarrow \sigma^*$ and go back to 2

Would it make sense to perform just a gradient step here too?

Gradient step in $[0,1]^n$.

Gradient descent 2

Advantage of this approach:

- can be done for non-linear systems (e.g. SVM with Gaussian kernels)
- can mix the search for features with the search for an optimal regularization parameters and/or other kernel parameters.

Drawback:

- heavy computations
- back to gradient based machine algorithms (early stopping, initialization, etc.)

Gradient descent summary

- Many algorithms can be turned into embedded methods for feature selections by using the following approach:

1. Choose an objective function that measure how well the model returned by the algorithm performs
2. Differentiate this objective function according to the σ parameter
3. Performs a gradient descent on σ. At each iteration, rerun the initial learning algorithm to compute its solution on the new scaled feature space.
4. Stop when no more changes (or early stopping, etc.)
5. Threshold values to get list of features and retrain algorithm on the subset of features.

Difference from add/remove approach is the search strategy. It still uses the inner structure of the learning model but it scales features rather than it selects them.

Design strategies (revisited)

- Directly minimize the number of features that an algorithm uses (focus on feature selection directly and forget generalization error).
- In the case of linear system, feature selection can be expressed as:

$$\min_{w} \sum_{i=1}^{n} 1_{w_i \neq 0}$$

Subject to $y_k\,(w.x_k + b) \geq 0$

Feature selection for linear system is NP hard

- Amaldi and Kann (1998) showed that the minimization problem related to feature selection for linear systems is NP hard: the minimum cannot be approximated within $2^{\log^{1-\varepsilon}(n)}$ for all $\varepsilon > 0$, unless NP is in DTIME($n^{\text{polylog}(n)}$).
- Is feature selection hopeless?
- How can we approximate this minimization?

Minimization of a sparsity function

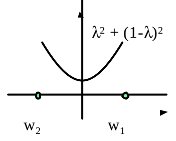
- Replace $\sum_{i=1}^{n} 1_{w_i \neq 0}$ by another objective function:
  - l_1 norm: $\longrightarrow \|w\|_1 = \sum_{i=1}^{n} |w_i|$
  - Differentiable function: $\longrightarrow \sum_{i=1}^{n} (1 - \exp^{-\alpha|w_i|})$
- Do the optimization directly!

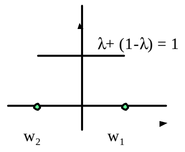
The l_1 SVM

- The version of the SVM where the margin term $||w||^2$ is replace by the l_1 norm $\sum_i |w_i|$ can be considered as an embedded method:
  - Only a limited number of weights will be non zero (tend to remove redundant features)
  - Difference from the regular SVM where redundant features are all included (non zero weights)

A note on SVM

- Changing the regularization term has a strong impact on the generalization behavior…
- Let $w_1=(1,0)$, $w_2=(0,1)$ and $w_\lambda=(1-\lambda)w_1+\lambda w_2$ for λ 2 [0,1], we have:
  - $||w_\lambda||^2 = (1-\lambda)^2 + \lambda^2$) minimum for $\lambda = 1/2$
  - $|w_\lambda|_1 = (1-\lambda) + \lambda$





The gradient descent

- Perform a constrained gradient descent on:

$$\min_{w,b} \sum_{k=1}^{m} 1 - \exp(-\alpha v_i)$$

Under the constraints:

$$y_k(w.x_k + b) \geq 0,\ w_i \leq v_i,\ -w_i \leq v_i$$

A direct approach

- Replace $\sum_{i=1}^{n} \mathbb{I}_{w_i \neq 0}$ by $\sum_i \log(\varepsilon + |w_i|)$
- Same idea as gradient descent but using another approximation.
- Boils down to the following multiplicative update:

```
1. Set $\sigma = (1, \ldots, 1)$

2. Get $w^*$ solution of an SVM on data set
   where each input is scaled by $\sigma$.

3. Set $\sigma = w^* * \sigma$

4. back to 2.
```

Embedded method - summary

- Embedded methods are a good inspiration to design new feature selection techniques for your own algorithms:
  - Find a functional that represents your prior knowledge about what a good model is.
  - Add the \sigma weights into the functional and make sure it's either differentiable or you can perform a sensitivity analysis efficiently
  - Optimize alternatively according to \alpha and \sigma
  - Use early stopping (validation set) or your own stopping criterion to stop and select the subset of features
- Embedded methods are therefore not too far from wrapper techniques and can be extended to multiclass, regression, etc…

Exercise Class

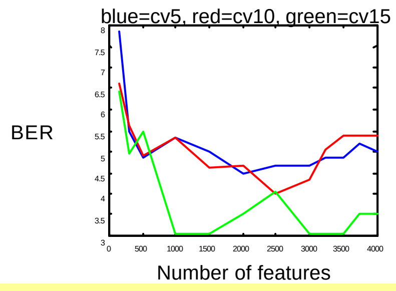
Homework 8: Solution

- Baseline model: 5% BER (trained on training data only)
- Best challenge entries ~3% BER
- Tips to outperform the challengers:
  - Train on (training + validation) set => double the number of examples
  - Vary the number of features

```
my_classif=svc({'coef0=1', 'degree=1', 'gamma=0',
  'shrinkage=0.5'});
my_model=chain({s2n('f_max=???'), normalize,
  my_classif})
```

  - Select best model by CV

Difficulty: Good CV



- **Get 1 point if you make an entry with less than 5% error**
- **Get 2 points if you make an entry with less than 4% error**

Filters Implemented

- @s2n
- @Relief
- @Ttest
- @Pearson (Use Matlab corrcoef. Gives the same results as Ttest, classes are balanced.)
- @Ftest (gives the same results as Ttest. Important for the pvalues: the Fisher criterion needs to be multiplied by num_patt_per_class or use anovan.)
- @aucfs (ranksum test)

Exercise - 1

- Consider the 1 nearest neighbor algorithm. We define the following score:

$$\sum_{k=1}^{m} ||x_k - x_{s(k)}||^2 - \lambda ||x_k - x_{d(k)}||^2$$

- Where s(k) (resp. d(k)) is the index of the nearest neighbor of x_k belonging to the same class (resp. different class) as x_k.

Exercise - 1 (cont.)

- 1. Motivate the choice of such a function to upper bound the generalization error (qualitative answer)
- 2. How would you derive an embedded method to perform feature selection for 1 nearest neighbor using this functional?
- 3. Motivate your choice (what makes your method an 'embedded method' and not a 'wrapper' method)

Exercise - 2

- Design an RFE algorithm in a multi-class set-up (hint: choose a regular multi-class SVM, add the \sigma scaling factors into the functional and compute the gradient).
- Discuss the advantages/drawback of this approach when compared to using many two classes RFE algorithms in a one-against-the rest approach.