GI01/4C55: Supervised Learning

7. Tree-based learning algorithms

November 21, 2005

Massimiliano Pontil

Today's plan

- Bayesian interpretation of regularization
  (cont. from last class)
- Classification and regression trees (CART)
- Multi variate regression splines (MARS)

Bibliography: These lecture notes are available at:
http://www.cs.ucl.ac.uk/staff/M.Pontil/courses/index-SL05.htm
Lecture notes are based on Hastie, Tibshirani & Friedman, Chapters 8.3 and 9.2,9-4

Regularization in Hilbert spaces

Consider the square loss regularization problem

$$\min_{\mathbf{w}} \left\{ \frac{1}{m} \sum_{i=1}^{m} (y_i - f_{\mathbf{w}}(\mathbf{x}))^2 + \lambda J(\mathbf{w}) \right\} \tag{1}$$

Our main example has been $f_{\mathbf{w}} = \mathbf{w}^\top \boldsymbol{\phi}$ where $\phi : \mathbb{R}^d \to \mathcal{W}$ and $\mathcal{W}$ is a Hilbert space (we mainly considered finite dimensional Hilbert spaces, $\mathcal{W} \equiv \mathbb{R}^N$) and $J(\mathbf{w}) = \mathbf{w}^\top \mathbf{w}$ but the observations we shall make here extend to general Hilbert spaces

- We shall now derive a Bayesian interpretation of regularization

Bayesian statistical methods (I)

The Bayesian approach to learning/inference assumes

- $P(S_m|f)$: a sampling probability model for the data (the conditional probability of S_m given f)
- $P(f)$: an **a priori** distribution over the possible functions / parameters

It then uses Bayes rule to compute the posterior distribution of a function given the data, $P(f|S_m)$. This represents the updated knowledge about f after we have seen the data and is used for prediction

Bayesian statistical methods (II)

Since the data are i.i.d. the conditional prob. factors as

$$\begin{aligned} P(S_m|f) &= \prod_{i=1}^{m} P(\mathbf{x}_i, y_i|f) \\ &= \prod_{i=1}^{m} P(\mathbf{x}_i)P(y_i|\mathbf{x}_i, f) \\ &= \prod_{i=1}^{m} P(\mathbf{x}_i)P(y_i|f(\mathbf{x}_i)) \propto \prod_{i=1}^{m} P(y_i|f(\mathbf{x}_i)) \end{aligned}$$

Hence $P(S_m|f)$ is essentially a model of the noise of the output data: the probability that, if the function underlying the data is f, by randomly sampling f at the sites $\{\mathbf{x}_i\}_{i=1}^m$ the set of measurements $\{y_i\}_{i=1}^m$ is obtained

Bayesian statistical methods (III)

If the noise is normally distributed with variance σ then the probability $P(S_m|f)$ can be written as:

$$P(S_m|f) \propto \prod_{i=1}^{m} P(y_i|f(\mathbf{x}_i)) \propto e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - f(\mathbf{x}_i))^2}$$

The posterior $P(f|S_m)$ can be computed via Bayes rule as

$$P(f|S_m) = \frac{P(S_m|f)P(f)}{P(S_m)} \propto P(S_m|f)P(f) \qquad (2)$$

Model of the prior

The prior $P(f)$ can be used to impose constraints on f assigning significant probability only to those functions that satisfy those constraints

Assuming a **non-informative prior** (improper prior)

$$P(f) = const$$

we see that maximizing the posterior distribution is equivalent to Maximum Likelihood

$$\max_f P(S_m|f) = \max_f e^{-\frac{1}{2\sigma^2}\sum_{i=1}^m (y_i - f(\mathbf{x}_i))^2}$$

which in turn is the same as least squares

$$\min_f \sum_{i=1}^m (y_i - f(\mathbf{x}_i))^2$$

Gaussian prior

If $f = \mathbf{w}^\top \boldsymbol{\phi}$ we define the norm $\|f\|^2 := \mathbf{w}^\top \mathbf{w}$ and assume the Gaussian prior of f

$$P(f) \propto e^{-\|f\|^2} \tag{3}$$

The function $f : \mathbb{R}^d \to \mathbb{R}$ can be seen as a Gaussian random field (Gaussian process in the space of functions above)

Following Bayes rule (2) the *a posteriori* probability of f is written as

$$P(f|S_m) \propto P(S_m|f)P(f) \propto e^{-(\frac{1}{2\sigma^2}\sum_{i=1}^m (y_i - f(\mathbf{x}_i))^2 + \|f\|^2)} \tag{4}$$

Gaussian prior (cont.)

We have that

$$\begin{aligned}\mathbf{E}_f[f(\mathbf{x})f(\mathbf{x}')] &= \mathbf{E}_{\mathbf{w}}[(\boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w})(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}'))] \\ &= \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{E}[\mathbf{w}\mathbf{w}^\top]\boldsymbol{\phi}(\mathbf{x}') \\ &= \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}') = K(\mathbf{x}, \mathbf{x}')\end{aligned}$$

Where we have used the fact that $\mathbf{E}[\mathbf{w}^\top \mathbf{w}] = \mathbf{I}_N$

Since the average function according to the prior is the zero function the kernel can be interpreted as the covariance matrix of between the function values $f(\mathbf{x})$ and $f(\mathbf{x}')$

MAP estimate

The **maximum a posteriori** (MAP) estimate considers the function that maximizes $P(f|S_m)$ in (4). This is the same as minimizing the exponent in (4),

$$\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2 + \|\mathbf{w}\|^2$$

hence the MAP is equivalent to minimizing our regularization problem,

$$\frac{1}{m}\sum_{i=1}^{m}(y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2 + \lambda\|\mathbf{w}\|^2$$

where $\lambda = \frac{2\sigma^2}{m}$

MAP estimate and Bayesian estimate

The MAP estimate is only one of several possible ones

In some cases, the Bayesian average

$$\hat{f} = \int f dP(f|S_m)$$

makes more sense (e.g. when the loss function is not convex)

In general, there is no nice analytical form for the integral and we need to either approximate the integrand function or do sampling

One can show that for the square loss the MAP and the average estimates coincide (exercise)

Tree-based learning algorithms

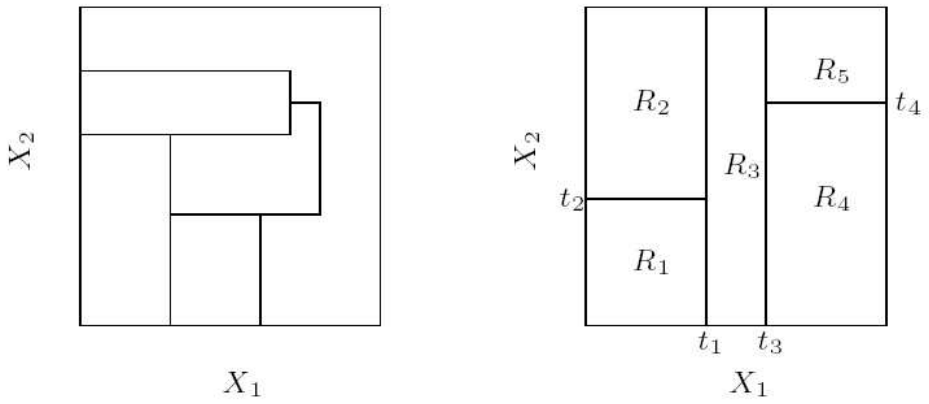
Tree methods attempt to partition the input space into a set of rectangles and fit a simple model (e.g. a constant) in each one

$$f(\mathbf{x}) = \sum_{n=1}^{N} c_n I\{\mathbf{x} \in R_n\}$$

- $\{R_n\}_{n=1}^N$: hyper-rectangles partitioning the input space: $\bigcup_{n=1}^N R_n = \mathbb{R}^d$, $R_n \cap R_\ell = \emptyset$
- $I\{\mathbf{x} \in R_n\} = 1$ if $\mathbf{x} \in R_n$ and 0 otherwise (indicator function)
- $\{c_n\}_{n=1}^N$: some real parameters. A natural choice is

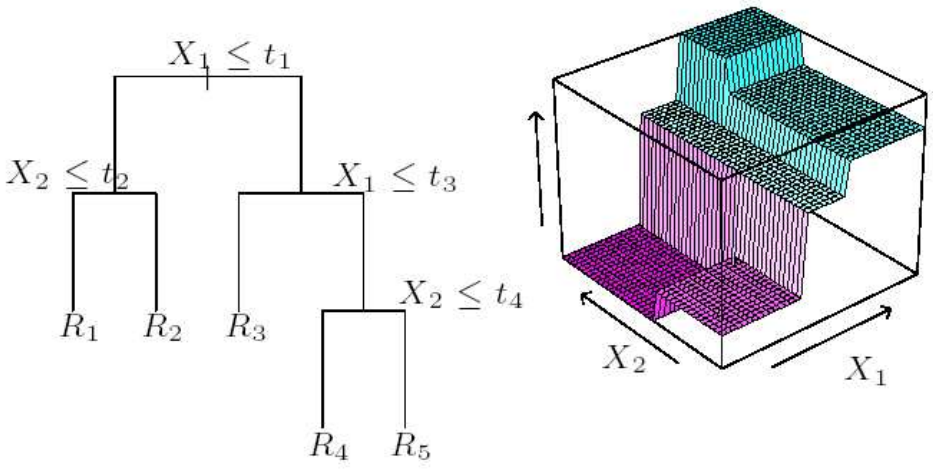
$$c_n = \text{ave}(y_i|\mathbf{x}_i \in R_n) \equiv \frac{\sum_{i=1}^m y_i I\{\mathbf{x}_i \in R_n\}}{\sum_{i=1}^m I\{\mathbf{x}_i \in R_n\}}$$

Recursive binary partitions



- Left plot: Each partition line has a simple description, yet partitions may be complicated to describe
- Right plot: recursive binary partition (first split the space into two regions then iterate in each region)

Tree representation



- Left plot: tree representation for the recursive binary partition above
- Right plot: prediction function $f(\mathbf{x}) = \sum_{n=1}^{5} c_n I\{\mathbf{x} \in R_n\}$

Some remarks

- A nice feature of a recursive binary tree is its **interpretability** (e.g. in medical science the tree stratifies the population into strata of high and low outcome on the basis of patient characteristics)

- Warning: risk for **overfitting**! (with enough splits the tree can fit arbitrarily well the data)

Key question: how to compute the tree (hence the regions R_n) and how to control overfitting?

Regression trees (I)

The algorithm needs to automatically decide on the splitting variables (1st or 2nd in above example) and split points. Recall that:

$$c_n = \text{ave}(y_i | \mathbf{x}_i \in R_n)$$

which corresponds to minimizing the square error in region n

Ideally we would like to solve the problem

$$\min_{R_1,\ldots,R_N} \left\{ \sum_{i=1}^{m} \left(y_i - \sum_{n=1}^{N} \text{ave}(y_j | \mathbf{x}_j \in R_n) I\{\mathbf{x}_i \in R_n\} \right)^2 \right\}$$

But it is generally computationally intractable. So we resort to an alternate heuristic procedure

Regression trees (II)

Let's find the first split in the tree

Define the pair of axis parallel half-planes:

$$R_1(j,s) = \{\mathbf{x}|x_j \leq s\}, \quad R_2(j,s) = \{\mathbf{x}|x_j > s\}$$

and search for optimal values $\bar{j}$ and $\bar{s}$ which solve the problem

$$\min_{j,s}\left\{\min_{c_1}\sum_{\mathbf{x}_i\in R_1(j,s)}(y_i-c_1)^2 + \min_{c_2}\sum_{\mathbf{x}_i\in R_2(j,s)}(y_i-c_2)^2\right\}$$

Regression trees (III)

$$\min_{j,s}\left\{\min_{c_1}\sum_{\mathbf{x}_i\in R_1(j,s)}(y_i-c_1)^2 + \min_{c_2}\sum_{\mathbf{x}_i\in R_2(j,s)}(y_i-c_2)^2\right\}$$

- The inner minimization is solved by

$$\bar{c}_1 = \text{ave}(y_i|\mathbf{x}_i \in R_1(j,s)) \qquad \bar{c}_2 = \text{ave}(y_i|\mathbf{x}_i \in R_2(j,s))$$

- For each splitting variable j the search for the best split point s can be done in $O(m)$ computations (can split between any two training points)

Thus the problem is solved in $O(dm)$ computations

Regression trees (IV)

In order to build the tree we recursively assign training points to each obtained region and repeat the above steps in each one

If we do not stop the process we clearly overfit the data! So, when should we stop it?

- follow a split only if it decreases the empirical error more than a threshold? No, there might be better splits below a "bad" node
- when a maximum depth of the tree is reached? No, this could underfit or overfit. We need to look at the data to determine a good size of the tree

Regression trees (V)

We choose the tree **adaptively** from the data:

- first grow a large tree $\hat{T}$ (stopping when a minimum number of data (e.g. 5) is assigned at each node)
- then prune the tree using a **cost-complexity pruning**, i.e. look for $T_\lambda \subseteq \hat{T}$ which minimizes

$$C_\lambda(T) = \sum_{n=1}^{|T|} m_n Q_n(T) + \lambda |T|$$

where T is a subtree of $\hat{T}$, n runs over leaf nodes of T (a subset of the nodes of $\hat{T}$) m_n is the number of data assigned to node n and $Q_n(T) = \frac{1}{m_n} \sum_{\mathbf{x}_i \in R_n} (y_i - c_n)^2$ (so the first term in C_λ is just the training error)

- Can show that there is a unique $T_\lambda \subseteq \hat{T}$ which minimizes C_λ

Regression trees (VI)

A C_λ is a kind of regularized empirical error: $T_0 = \hat{T}$, the original tree. Large values of λ result in smaller trees. A good value of λ can be obtained by cross validation

- **weakest link pruning**: successively collapse the internal nodes that produce the smallest per node increase in $\sum_{n=1}^{|T|} m_n Q_n(T)$ and continue till the root (single-node) tree is produced
- search along the produced list of trees for the one which minimizes C_λ

One can show that T_λ is in the produced list of subtrees, hence this algorithm gives the optimal solution

Classification trees (I)

When the output is a categorical variable (say $\mathcal{Y} = \{1, \ldots, K\}$) we use the same algorithm above with two important modifications

- For each region R_n we defined the empirical class probabilities:

$$p_{nk} = \frac{1}{m_n} \sum_{\mathbf{x}_i \in R_n} I(y_i = k)$$

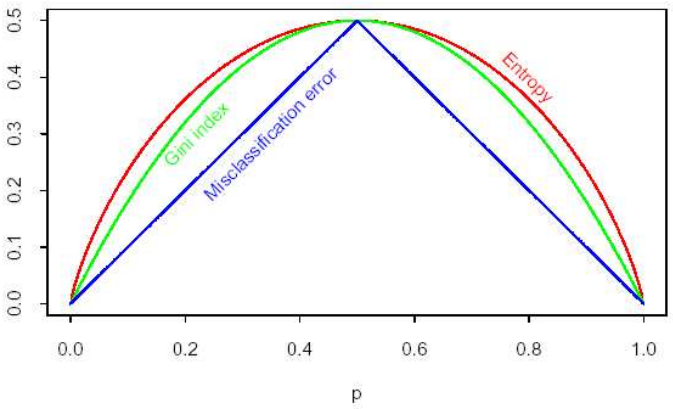
- We classify an input which falls in region n in the class with maximum probability

$$f(\mathbf{x}) = \text{argmax}_{k=1}^{K} \sum_{n=1}^{N} p_{nk} I\{\mathbf{x} \in R_n\}$$

- We use different measures $Q_n(T)$ of node impurity

Classification trees (II)

- Misclassification error:
  $1 - p_{nk(n)}$,
  $k(n) := \text{argmax}_{k=1}^{K} p_{nk}$
- Gini index:
  $\sum_k p_{nk}(1 - p_{nk})$
- Cross entropy:
  $\sum_k p_{nk} \log p_{nk}$



Cross entropy or Gini index are used to grow the tree (they are more sensitive in changes in the node probabilities). Misc. error is used to prune it.

MARS (I)

Multivariate adaptive regression splines (MARS) is an iterative algorithm which computes

$$f(\mathbf{x}) = c_0 + \sum_{n=1}^{N} c_n h_n(\mathbf{x})$$

where h_n are generated by taking products of one dimensional piecewise linear functions (splines) centered at the data:

$$\mathcal{C} = \left\{ (x_j - t)_+, (t - x_j)_+ : t \in \{x_{ij} : i = 1, \ldots, m\} \right\}$$

where x_j is the j-th component of vector $\mathbf{x}$ and x_{ij} j-th component of example $\mathbf{x}_i$

We discuss the MARS algorithm and its connection to CART

MARS (II)

Initialization: choose $f^{(0)} = c_0$ where c_0 is the average of the output data and set $\mathcal{M} = \{1\}$ (this set grows as the algorithm iterates and represents the basis functions which we insert in the model)

For $n = 1, \ldots, N$:

- Choose as the basis function h_n a function of the form

$$a_1(x_j - t)_+ g(\mathbf{x}) + a_2(t - x_j)_+ g(\mathbf{x}), \quad g \in \mathcal{M}$$

  which produces the largest decrease in the training error.
- Let $f^{(n)} = c_0 + \sum_{\ell=1}^{n} c_\ell h_\ell$ where the coefficients c_ℓ are recomputed to minimize the square error
- Add $(x_j - t)_+ g(\mathbf{x})$ and $(t - x_j)_+ g(\mathbf{x})$ to $\mathcal{M}$

MARS and CART

With two modifications MARS is equivalent to the tree growing algorithm in CART (discarding the pruning step)

- Replace the piecewise linear basis functions by step functions $I\{x - t > 0\}$ and $I\{t - x \geq 0\}$
- At each iteration remove from $\mathcal{M}$ the function which has been used to build a new model function

MARS can be used for classification as well