Master-MIRI
Topics on Optimization and Machine Learning (TOML)

José M. Barceló Ordinas
Departament d'Arquitectura de Computadors (UPC)

Bootstrapping and bagging …

- **Bootstrapping** is a sampling method that consists of creating a new training set by randomly sampling with replacement from the original sampling set,

- Example: let us assume that we have a training set consisting of the following data, **S=[1,2,3,4,5,6]**. The the following new training sets are valid using bootstrapping:

  S_1= [1,5,2,1,1,2]

  S_2= [4,1,3,3,4,4]

  S_3= [3,2,3,3,5,6]

  …

- The new training set can be of the same size or different size than the original, and as it can be observed, the samples of one of these new training sets can have repeated values, since we sample with replacements.

- In general, if the size of the training set is n, and n is large, it is expected to have (1-1/e) = 0.632, it is to say 63.2% of unique samples, and the rest 37.8% being repeated values.

Bootstrapping and bagging …

- **Ensemble methods:** use of several learning methods to obtain better performance. Ensemble is a supervised learning method (it is trained to later make predictions). Examples of ensemble methods: bagging, boosting, Bayes optimal classifier, Bayesian model averaging, etc.
- **Committees:** given that we train M different models, we make prediction using for example the average of the predictions made by each model. **Drawback:** averaging low-bias models, we reduce the error, but we need to add some variability (variance) since we have only one dataset. Thus, use bagging (select different data sets to introduce some variability)
- **Bagging** (**b**ootstrap **agg**regat**ing**) is an ensemble method designed to improve stability an accuracy of machine learning algorithms for classification and regression. It is composed of two parts:
  1. Bootstraping for producing multiple training sets
  2. Aggregation: for each different training set a model is set, where the final result is an ensemble of models (e.g. classifiers) in which each model votes with equal weight if the target is classification or the results are combined by averaging if the target is regression.

e.g. **Regression:** if we train separate copies $y_m(x)$, with m=1,…,M, of a predictive regression model, then the committee prediction is given by:

$$y_{com}(x) = 1/M \sum_{j=1...M} y_m(x)$$

- **How are the errors of bagging …**
  - If $y_m(x)$ is the outcome of a specific model, then $y_m(x) = f(x) + \epsilon_m(x)$, where **f(x) is the true regression function**, and the average sum of squares error is $E_x[(y_m(x)-f(x))^2] = E_x[\epsilon_m(x)^2]$. If we average over the M models, we obtain the average error of the models acting individually:

    $E_{AV} = 1/M \sum_{j=1...M} E_x[\epsilon_m(x)^2]$

  - Let's now obtain the committee error:

    $E_{COM} = E_x[(y_{com}-f(x))^2] = E_x[(1/M \sum_{j=1...M} y_m(x)-f(x))^2] = E_x[(1/M \sum_{j=1...M} \epsilon_m(x))^2]$,

If we assume that the errors $\epsilon_m(x)$ have zero mean and are uncorrelated:

$E_x[\epsilon_m(x)] = 0$

$E_x[\epsilon_m(x) \epsilon_n(x)] = 0$

We obtain:

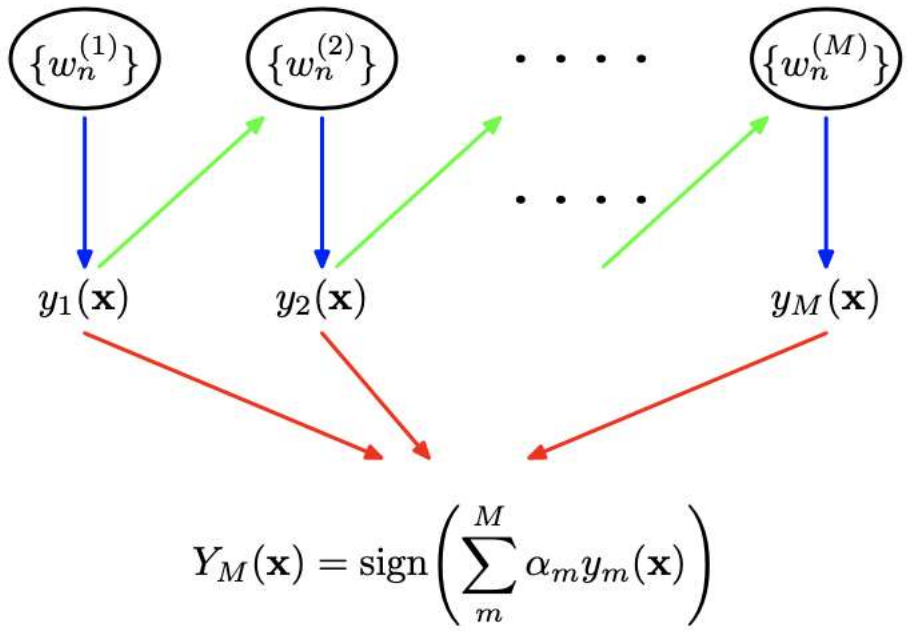
$E_{COM} = 1/M E_{AV}$,

  - However, in general, **the errors $\epsilon_m(x)$ are not uncorrelated** and the reduction in the overall error is small. However, it can be shown that in general $\mathbf{E_{COM} \leq E_{AV}}$.

  - To achieve improvements, it is better to use more sophisticated techniques such as **boosting**.

- **Boosting …**
  - **Boosting (AdaBoost):** assume a vector of experiments $\mathbf{x}^T_i=(x_{i1},\ldots,x_{iM})$, and outputs $t_i \in \{0,1\}$, from i=1,…,N, thus a classification problem. We train multiple models in sequence in which the error function used to train a particular model depends on the performance of previous models. In general, in the new training model more weight is given to data that has been misclassified. Initial weight $\{w^{(1)}_n\}=1/N$ for all n=1,…,N.

Schematic illustration of the boosting framework. Each base classifier $y_m(\mathbf{x})$ is trained on a weighted form of the training set (blue arrows) in which the weights $w_n^{(m)}$ depend on the performance of the previous base classifier $y_{m-1}(\mathbf{x})$ (green arrows). Once all base classifiers have been trained, they are combined to give the final classifier $Y_M(\mathbf{x})$ (red arrows).



- **Boosting …**
  - **AdaBoost algorithm** (**Ada**ptive **boost**ing): train new classifier using a dataset in which the weighting coefficients are adjusted according to the performance of the previously trained classifier, so as to give more weight to the misclassified data points. Algorithm:

1. Initialize the data weighting coefficients $\{w^{(1)}_n\}=1/N$ for n=1,…, N data points.

2. For m=1, …, M:
   a. Fit a classifier $y_m(x)$ to the training data by minimizing the weighted error function

      $$J_m = \Sigma_{n=1\ldots N}\, w^{(m)}_n\, I(y_m(x) \neq t_n)$$

      where $\mathbf{I(y_m(x) \neq t_n)}$ is the indicator function and equals to 1 if $y_m(x) \neq t_n$ and 0 otherwise.
   b. Evaluate the quantities: $\epsilon_m = J_m / (\Sigma_{n=1\ldots N}\, w^{(m)}_n)$,
      and use these to evaluate $\alpha_m = \ln \{ (1- \epsilon_m)/\epsilon_m \}$
   c. Update the data weighting coefficients:

      $$w^{(m+1)}_n = w^{(m)}_n \exp \{\alpha_m I(y_m(x) \neq t_n)\}$$

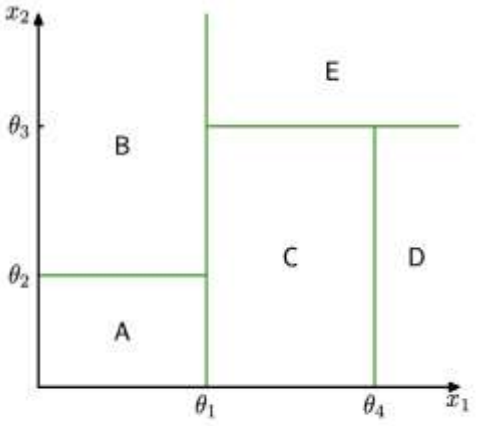
3. Make predictions using the final model:

   $$y_M(x) = \text{sign} (\Sigma_{m=1\ldots M}\, \alpha_m y_m(x))$$

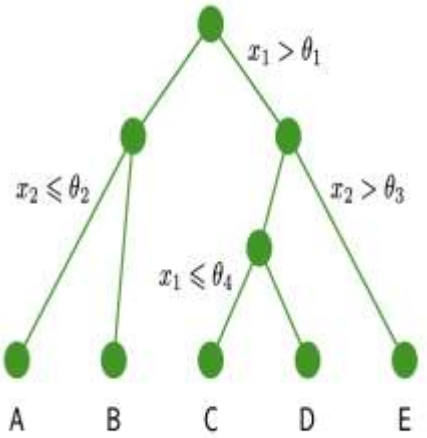
For m=1 since all $w^{(1)}_n$ are equal, is as having a single classifier, for then, weights $w^{(m)}_n$ increase for misclassified data points, and decrease for data that are correctly classified. Coefficients α_m give high weights to more accurate classifiers for the output.

- **Decision trees …**
  - A predictive model that uses a decision tree to go from observations about an item (branches) to conclusions about the item's target value.
  - Let's see an example (Bishop, chapter 14, section 14.4):

Illustration of a two-dimensional input space that has been partitioned into five regions using axis-aligned boundaries.



Binary tree corresponding to the partitioning of input space shown in Figure 14.5.



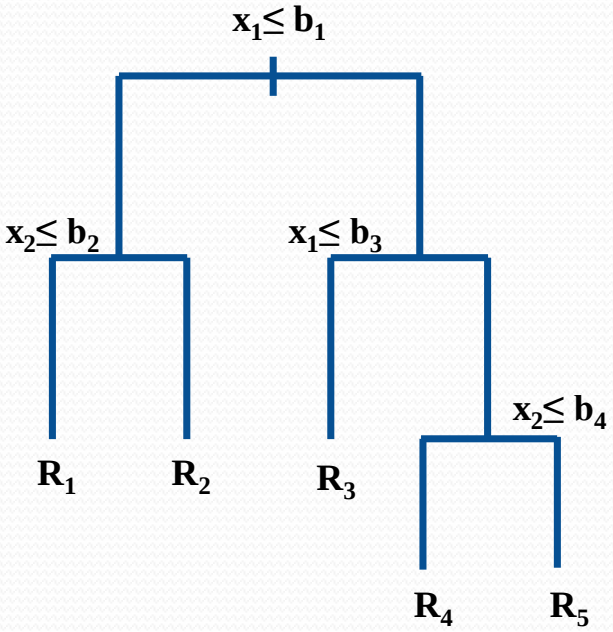
  - For learning such a model for the training set, we have to **determine the structure of the tree**, including the input variables at each node of the tree to make the split criterion and the value of the threshold parameter (in the example θ_i) for the split. Also we have to determine the values of the predictive variable (depending on whether we make a classification or a regression).

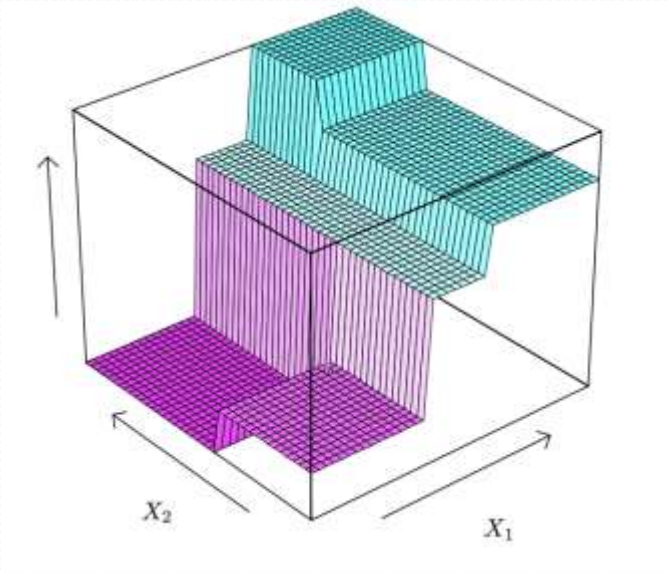
Decision trees …

- **Creating the tree structure:** let's have a D-dimensional vector of features $\mathbf{x}^T = (x_1,\ldots,x_D)$, and now $\{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ training data and labels $\{t_1, \ldots, t_N\}$.
- Calculating the optimal structure is infeasible due to the number of combination. Then, use a **greedy solution**, starting by a root node and adding nodes at each step of the process.
  - The idea is at each step look at candidate regions in input space that can be split, and also for each region, a choice of the D input variables to split, and the value of the threshold.
  - This is done by exhaustive search, noting that for a given choice of split variable and threshold, the optimal choice is given by the local average of the data,
  - Repeating the process for all possible choices of split variable and threshold, choose that one that gives smallest residual sum of squares error.
- For a deeper description of the process, see section 14.4 of Bishop's book, where it is explained a pruning method for an efficient search. Includes:
  - how to stop adding nodes to the tree (stopping criteria), and
  - a regularization parameter determines the trade-off between the sum-of-squares error and the complexity of the model as measured by the number of leaf nodes, and its value is chosen by cross-validation.

Decision trees regression … example

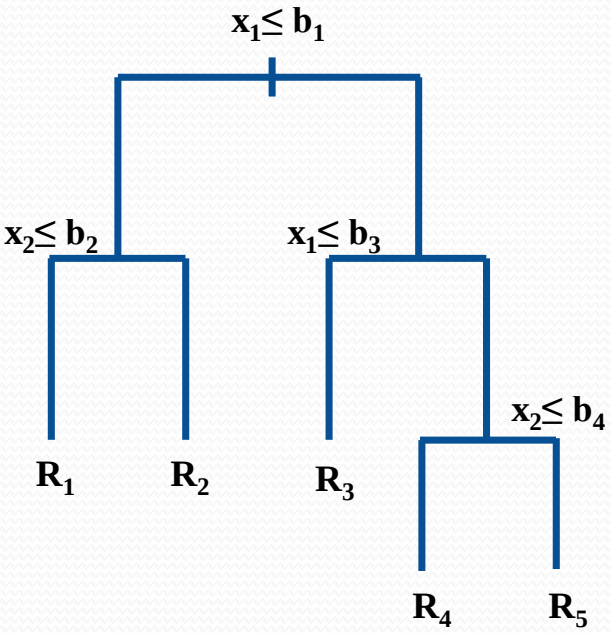
- Let us assume two features or predictor variables x_1 and x_2, and N data for each predictor, i.e., $\mathbf{X} \in R^{Nx2}$.
  1. The first step divides the data according on whether $x_1 \leq b_1$, if $x_1 \leq b_1$ goes to the left, otherwise goes to the right,
  2. What is on the **left** is then divided in two according to whether $x_2 \leq b_2$, resulting in R_1 and R_2. What is on the **right** is then divided in two according to whether $x_1 \leq b_3$,
  3. What is on the **left** results in R_3, what is on the **right** is again divided in two according to whether $x_2 \leq b_4$,
  4. What is on the **left** is results in R_4 and what is on the **right** is results in R_5.

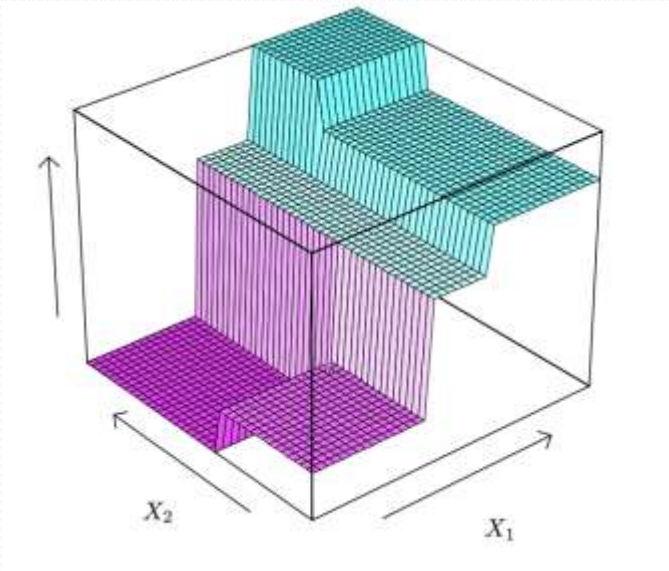




Decision trees regression … example

- To make a prediction $\tilde{y}$ on data (x_1,x_2), we begin on the root of the tree, and checks x_1. Then, following the tree the results will belong to one of the R_1, R_2, R_3, R_4, R_5 results. That is the prediction, it is to say $\tilde{y}=R_i$.
- How to decide which variable X_i has to be split and with which value b_i ?





Decision trees regression … example

- How to decide which variable X_i has to be split and with which value b_i ? Several algorithms to do that. Assume vector of features $\mathbf{x}^T=(x_1,\ldots,x_D)$, and now $\{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ training data and labels $\{t_1, \ldots, t_N\}$. The most common method is the following: suppose we are at node k, and region R_k of N_k cases remains to be split at this node.
- Now let us assume that we want to divide the node in |T| leaves, with j=1,…, |T|, where R_j is the region having N_j data of the N_k data. We produce terms m_j that are means and terms $Q_j(T)$ that are sums of squares:

  $m_j = \Sigma_{Xi \in Rj}\, t_i/N_j$ and $Q_j(T) = \Sigma_{Xi \in Rj}\, (t_i - m_j)^2$ with j=, 1…,|T|.

- The algorithm chooses the splitting variable X_i and threshold b_k that minimizes:

  $C(T)= \Sigma_{j=1,\ldots,|T|}\, Q_j(T) + \lambda\, |T|,$

Where λ determines the trade-off between overall sum of squares and the complexity of the model as measured by the number |T| of leaf nodes, and its value is chosen using cross-validation,

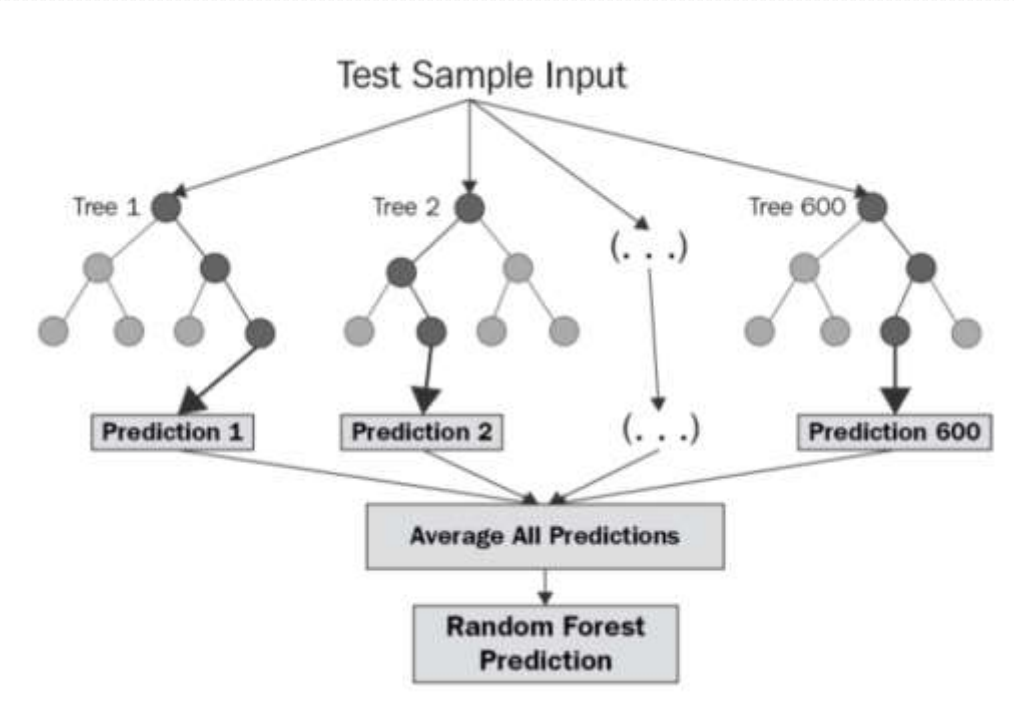
- When stop splitting ? → use cross-validation predicting the error
- If a branch stops which is the terminal value R_j for regression? → use the m_k,
- If classification instead of regression → use the cross-entropy or Gini index instead of sum of squares. If p_{jk} is the proportion of data to be assigned to region R_j and class k=1,..,K.

  cross-entropy: $Q_j(T) = \Sigma_{k=1,\ldots,K}\, p_{jk} \ln (p_{jk})$

  Gini-index: $Q_j(T) = \Sigma_{k=1,\ldots,K}\, p_{jk} (1-p_{jk})$

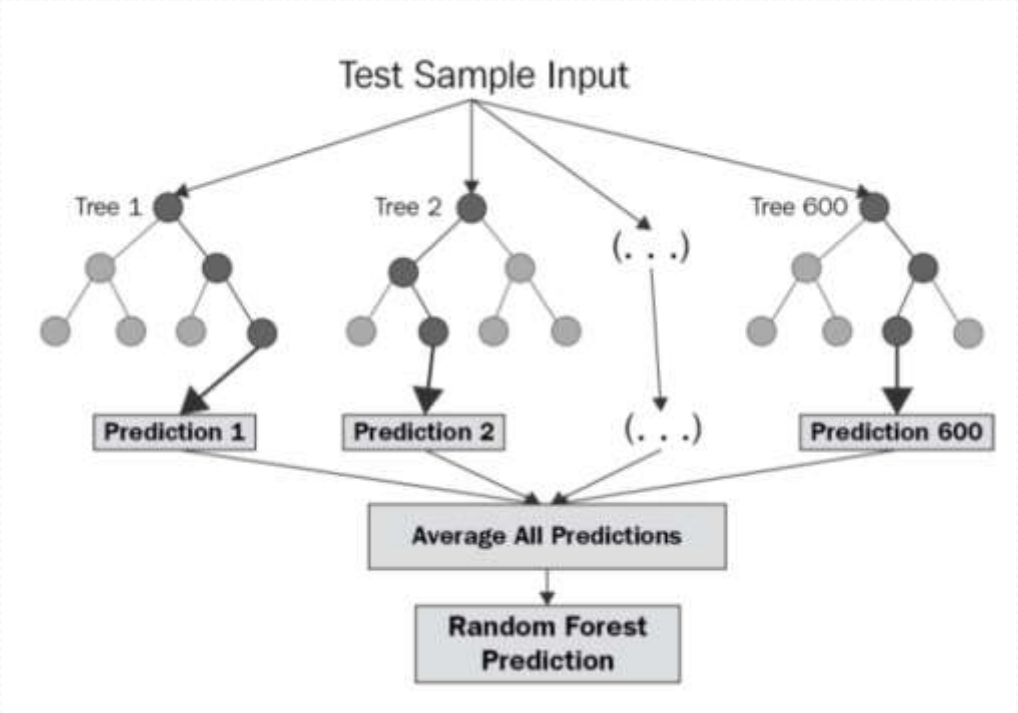
Decision trees … and Random Forest (RF)

- **Random Forest:** this is an ensemble machine learning method for classification and regression based on bagging and decision trees:
  - The idea is to create a bagging mechanisms in which M training sets are selected using bootstrapping and then M decision trees are averaged to obtain the final result,
  - Be careful, **random forest tends to overfit**: this causes high variance, which can be seen as high test errors on the test dataset, despite high accuracy on the training dataset,
  - Then, improved if the randomize over the number of features, define the maximum depth of the trees or increase the number of estimators (number of trees).



Decision trees … and Random Forest (RF)

- **Random Forest** in action:
  1. Pick k random data points from the training set,
  2. Build a decision tree associated to these k data points,
  3. Specify the number N of trees that you want to build and repeat step 1 and 2,
  4. For a new data point, make each of your N trees to predict and if you classify chose by majority voting and if you regress obtain the average,



Kernel methods: Dual representation …

- Let's go back to the **regression model**: $\mathbf{y(x,\beta) = x^T\beta = \beta^T x}$, (with β a M-dimensional vector) but now let consider that the regressors the dependent variable $y(\mathbf{x},\beta)$ depends on vectors $\boldsymbol{\phi}(\mathbf{x})$, thus $\mathbf{y(x,\beta) = \beta^T x = \beta^T \phi(x)}$. The specific case in which $\boldsymbol{\phi}(\mathbf{x})= \mathbf{x}$, represents the multiple linear regression, so in general $\boldsymbol{\phi}(\mathbf{x})$ can also express **a non-linear feature space**.
- Let us now express the **penalty function** with a regularization factor:

$$\mathbf{J(\beta)} = 1/2\ ||\mathbf{y(x,\beta)} - \mathbf{t}||^2_2 + \lambda/2\ ||\beta||^2 = 1/2\ \Sigma_{j=1\ldots N}\ (\beta^T \boldsymbol{\phi}(\mathbf{x_n}) - t_n)^2 + \lambda/2\ \beta^T\beta,$$

- Obtaining the gradient $\partial J(\beta)/\partial\beta = 0$:

$$\beta = -1/\lambda\ \Sigma_{j=1\ldots N}\ (\beta^T \boldsymbol{\phi}(\mathbf{x_n}) - t_n)\ \boldsymbol{\phi}(\mathbf{x_n}) = \Sigma_{j=1\ldots N}\ a_n\ \boldsymbol{\phi}(\mathbf{x_n}) = \boldsymbol{\Phi}^T\mathbf{a}\ ,$$

Where we call $\boldsymbol{\Phi}$ as the designed matrix whose n^{th} row is given by $\boldsymbol{\phi}(\mathbf{x_n})^T$ and define vector $\mathbf{a}=(a_1, \ldots, a_N)$ as a vector with coefficients:

$$a_n = -1/\lambda\ (\beta^T \boldsymbol{\phi}(\mathbf{x_n}) - t_n) \rightarrow \mathbf{a} = -1/\lambda\ (\beta^T \boldsymbol{\Phi} - \mathbf{t})$$

- Now, substituting in the penalty function, we can express $J(\beta)$ as:

$$\mathbf{J(\beta)} = 1/2\ (\mathbf{a^T K K a} - 2\mathbf{a^T K\ t} + \mathbf{t^T t}) + \lambda/2\ \mathbf{a^T K a}$$

Where **K** is the Gramm matrix defines as:

$$\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T \quad \text{which is a NxN symmetric matrix with elements } K_{nm} = \boldsymbol{\phi}(\mathbf{x_n})^T\boldsymbol{\phi}(\mathbf{x_m}) = k(\mathbf{x_n},\mathbf{x_m}),$$

The function **k(x,x')** is called a **Kernel function**.

Kernel methods: Dual representation …

If we set the gradient to J(β) to 0, we can obtain:

$$\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{t}$$

and for predicting values:

$$y(x,\beta) = \beta^T \phi(x) = \mathbf{a}^T \boldsymbol{\Phi}\, \phi(x) = \mathbf{k}(x)^T (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{t}$$ with $\mathbf{k}(x)$ with elements $k_n(\mathbf{x}_n, x) = \phi^T(\mathbf{x}_n)\, \phi(x)$.

As a conclusion, we have expressed a **dual formulation** for the least-square problem for multiple linear regression as a function to the **kernel function** k(x,x').

Differences between the original formulation and the dual formulation:

Original formulation: $\beta = (\mathbf{X}^T\mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$, we have to invert $\mathbf{X}^T\mathbf{X}$ which is a MxM matrix,

Dual Formulation: we have to invert $(\mathbf{K} + \lambda N \mathbf{I}_N)$ which is a NxN matrix,

1. Since N>>M, it seems that the dual formulation is less efficient that the original formulation.
2. However, the dual formulation is expressed as a function of a kernel function, and then we don't have to use explicitly the feature vector $\phi(\mathbf{x}_n)$, which allows to use a feature vector that can lie in an infinite or very high dimensionally space.

Kernel Methods: examples…

- Kernels $k(\mathbf{x},\mathbf{y})= \phi^T(\mathbf{x})\phi(\mathbf{y})$ have to be symmetric and positive semi-definite. We can observe that kernels are the scalar product of two vectors in a feature space $\phi(\mathbf{x})$.

1. $k(\mathbf{x},\mathbf{y}) = \phi^T(\mathbf{x})\,\phi(\mathbf{y}) = \mathbf{x}^T\mathbf{y}$ **(linear)** → $\phi^T(\mathbf{x}) = \mathbf{x}^T = (x_1, \ldots, x_M)$. Then, for example, for x_n, x_m we calculate the kernel as: $K_{nm} = k(x_n,x_m) = (x_{n1}, \ldots, x_{nM})(x_{m1}, \ldots, x_{mM}) = \Sigma_{\_\{i=1,\ldots,M\}}\, x_{ni}\, x_{mi}$,

2. $k(\mathbf{x},\mathbf{y}) = (c+\mathbf{x}^T\mathbf{y})^d$ **(d-polynomial)** with c>0 or $k(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^d$ → e.g., if d=2, $k(\mathbf{x},\mathbf{y})= (\mathbf{x}^T\mathbf{y})^2$, and $\mathbf{x}=(x_1,x_2)$, $\mathbf{y}=(y_1,y_2)$ → $\phi(\mathbf{x}) = (x^2_1, 2^{1/2}x_1x_2, x^2_2)$,

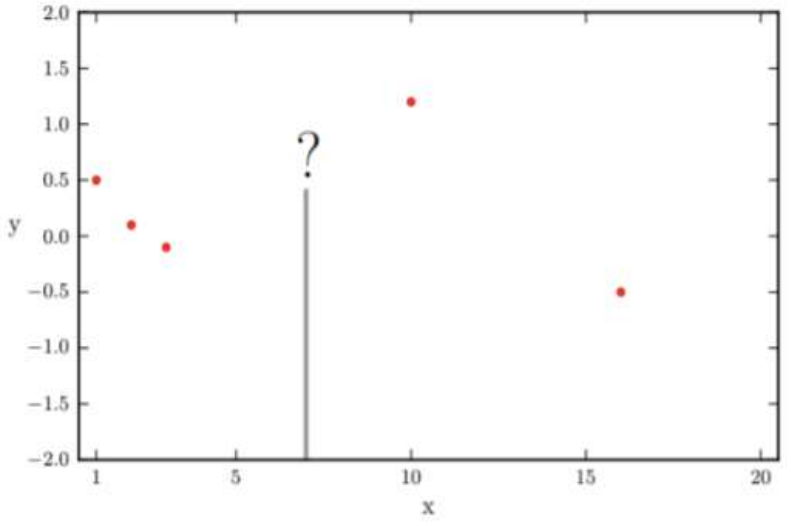
3. $k(\mathbf{x},\mathbf{y}) = \exp(-1/(2\sigma^2)(||\mathbf{x}-\mathbf{y}||^2)$ **(Gaussian kernel)** → the feature space if x is a 1-D vector is $\phi(\mathbf{x}) = \exp(-x/(2\sigma^2)\,(1, (1/1!\sigma^2)^{1/2}x, (1/2!\sigma^4)^{1/2}x^2, (1/3!\sigma^6)^{1/2}x^3, \ldots)$
   (infinite dimensional space)

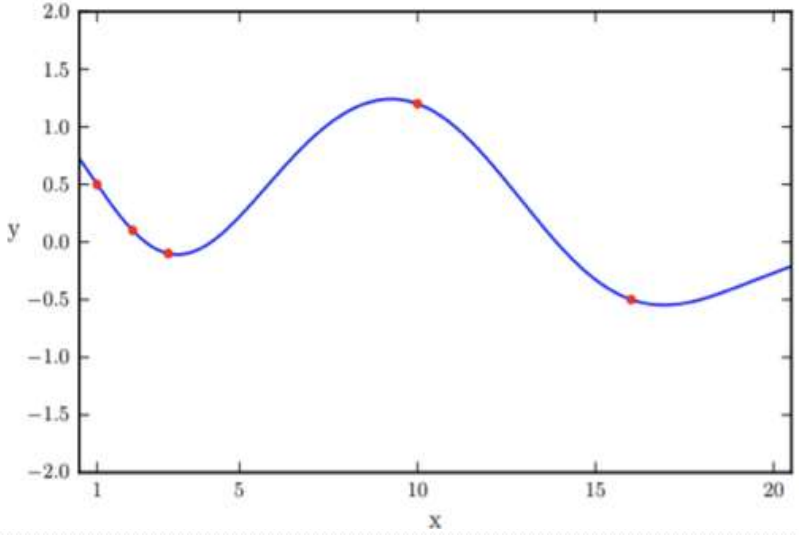
4. Other kernels: $k(\mathbf{x},\mathbf{y}) = \tanh(a\mathbf{x}^T\mathbf{y}+b)$ (**sigmoidal**), $k(\mathbf{x},\mathbf{y})= \exp(-1/(2\sigma^2)(||\mathbf{x}-\mathbf{y}||)$ (**exponential**), $k(\mathbf{x},\mathbf{y}) = \exp(-1/\sigma(||\mathbf{x}-\mathbf{y}||)$ (**Laplacian**), and others such as ∫spherical, wave, log, circular, Cauchy, etc.

It is possible to build complex kernels from combining simpler ones and following certain conditions or properties, see Bishop, ch 6, section 6.2.

Gaussian processes (GP): motivation …

- **Non-linear regression**: we have a set of data points (red dots in the left figure), and we want to predict a new value (with a question mark in the left figure) given a new input x. Same problem than linear regression, but now y(x) does not follow a linear model, but follows a non-linear model.
- GP extends multivariate Gaussian distributions to infinite dimensionality. The idea is that any data set can be thought as a sample from some multivariate (n-variate) Gaussian distribution, and the relation between one data and another is the covariate function $k(x_i,x_j)$





- **Gaussian processes: Let's see what happens with a linear regression …**
  - Assume the previous linear model $y(\mathbf{x}) = \beta^t\phi(\mathbf{x})$, where β is a M-dimensional vector, and now consider a prior distribution over β, following a Normal distribution:

    $\beta \sim N(0,\alpha^{-1}\mathbf{I})$ governed by the hyperparameter α, which represents the **precision** (inverse variance, $\alpha = 1/\sigma^2$),
  - We want to evaluate $y(\mathbf{x})$ over the training points $\mathbf{x}_1$, …, $\mathbf{x}_N$, (with each $\mathbf{x}_n=(x_{n1},\ldots, x_{nM})$), it is to say, the joint distribution of the function values $y(\mathbf{x}_1),\ldots, y(\mathbf{x}_N)$, that we call vector $\mathbf{y}$.
  - Vector $\mathbf{y}$ is given by $\mathbf{y} = \boldsymbol{\Phi}\beta$ where $\boldsymbol{\Phi}$ is the design matrix with elements $\boldsymbol{\Phi}_{nm} = \phi_m(\mathbf{x}_n)$ for m=1,…, M.
  - Note that vector $\mathbf{y}$ is a linear combination of Gaussian distribution variables β and then it also is Gaussian. We have to see what is its mean and variance to characterize it,

    $E[\mathbf{y}] = E[\boldsymbol{\Phi}\beta] = \boldsymbol{\Phi}E[\beta] = 0$

    $cov[\mathbf{y}] = E[\mathbf{y}^T\mathbf{y}] = \boldsymbol{\Phi}\, E[\beta\beta^T]\, \boldsymbol{\Phi}^T = \boldsymbol{\Phi}\alpha^{-1}\boldsymbol{\Phi}^T = \alpha^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \mathbf{K}$,

  With $\mathbf{K} = k(\mathbf{x}_n,\mathbf{x}_m) = \alpha^{-1}\,\phi(\mathbf{x}_n)\,\phi(\mathbf{x}_m)$, where $k(\mathbf{x}_n,\mathbf{x}_m)$ is a kernel function

  - **Definition**: a **Gaussian process** is a probability distribution over functions $y(\mathbf{x})$ evaluated at points $\mathbf{x}_1$, …, $\mathbf{x}_N$, that jointly have a Gaussian distribution, and therefore are characterized by their 2nd order statistics, it is to say, if its mean is zero, it is defined by its covariance matrix:

    $\mathbf{y} \sim p(\mathbf{y}) = N(0,\mathbf{K})$ being matrix **K** derived from a kernel function

Gaussian processes … and with a non-linear regression …

- We now have observed values $\mathbf{x}=(\mathbf{x_1}, \ldots, \mathbf{x_N})$, and targeted values $\mathbf{t}=(t_1, \ldots, t_N)$ to which we assume that come with some noise:

  $$t_n = y_n + \epsilon_n,$$

  where: $\mathbf{y_n}= y(\mathbf{x_n})$, and the error $\epsilon_n= N(0,\gamma^{-1})$, where γ is an hyperparameter that represents the precision (inverse of variance) of the noise.

- Since the noise is independent at each point, the joint distribution of vector **t** given **y** is given by an isotropic (covariance matrix only has diagonal values) Gaussian distribution:

  $$\mathbf{t}|\mathbf{y} \sim p(\mathbf{t}|\mathbf{y}) = N(\mathbf{t}|\mathbf{y}, \gamma^{-1}\mathbf{I_N}),$$ with $\mathbf{I_N}$ the NxN identity (diagonal) matrix

- Moreover the distribution of $p(\mathbf{y}) = N(0,\mathbf{K})$ is Gaussian with the covariance matrix **K** derived from a kernel function, meaning that those entry points $\mathbf{x_n}$ and $\mathbf{x_m}$ that are similar will produce $y(\mathbf{x_n})$ and $y(\mathbf{x_m})$ more strongly correlated than entry dissimilar points.

- Now, the objective is to predict new values t_{N+1} from new data $\mathbf{x_{N+1}}$, and for that we need to calculate the predictive distribution $p(t_{N+1}|\mathbf{t})$:
  1. calculate the marginal distribution $p(\mathbf{t})$, then
  2. calculate $p(\mathbf{t}_{N+1})$, and finally
  3. condition on **t**, $p(t_{N+1}|\mathbf{t})$

Gaussian processes: … and with a non-linear regression …

- to predict new values t_{N+1} from new data $\mathbf{x_{N+1}}$, and for that we need to calculate the predictive distribution $p(t_{N+1}|\mathbf{t})$, and for that we need to :

1. calculate the marginal distribution $p(\mathbf{t})$:

$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})\, p(\mathbf{y}) d\mathbf{y} = N(0,\mathbf{C})$

where the covariance matrix **C** is given by $C_{nm} = C(\mathbf{x_n},\mathbf{x_m}) = k(\mathbf{x_n},\mathbf{x_m}) + \gamma^{-1}\delta_{nm}$, with δ_{nm} the Kronecker delta function ($\delta_{nm}=1$ if n=m, $\delta_{nm}=0$ otherwise),

2. Then, we obtain $p(\mathbf{t}_{N+1}) = p(t_1,\ldots, t_N, t_{N+1}) = N(0,\mathbf{C}_{N+1})$

with $\boldsymbol{C}_{N+1} = \begin{pmatrix} \boldsymbol{C}_N & \boldsymbol{k} \\ \boldsymbol{k}^T & c \end{pmatrix}$, where $\mathbf{C}_N$ is the NxN covariance matrix given above, vector **k** (column vector) has elements $k(x_n,x_{N+1})$, $\mathbf{k}^T$ is its transpose (row vector), and the scalar $c=k(x_{N+1},x_{N+1})$,

3. Finally, we obtain $p(t_{N+1}|\mathbf{t}) = N(m(x_{N+1}), \sigma^2(x_{N+1}))$ that is a Gaussian distribution with mean $m(x_{N+1})$ and variance $\sigma^2(x_{N+1})$, where:

$m(x_{N+1}) = \mathbf{k}^T\mathbf{C}^{-1}_N\mathbf{t}$

$\sigma^2(x_{N+1}) = c - \mathbf{k}^T\mathbf{C}^{-1}_N\mathbf{t}$

- **Gaussian processes: … and with a non-linear regression …**
  - Restrictions on Gaussian processes:
    1. Matrix **C** has to be positive definite. Since **C** is given by $C_{nm} = C(\mathbf{x_n}, \mathbf{x_m}) = k(\mathbf{x_n}, \mathbf{x_m}) + \gamma^{-1}\delta_{nm}$,
    2. The eigenvalues λc_i of C are given by $\lambda c_i = \lambda_i + \gamma^{-1}$, with λ_i being the eigenvalues of matrix **K**, obtained from a kernel function,
    3. Then, if matrix K is positive semidefinite, then any λ_i will be positive or zero. Even if it is zero, since $\gamma^{-1} > 0$, then $\lambda c_i > 0$, and matrix **C** will be positive definite.

  - **Problems with Gaussian processes**:
    - scale badly with N, since we need to construct a covariance matrix of size NxN. Then, use of approximate methods that have better behaviour when N scales,
    - the predictions of the Gaussian process depends on the choice of the covariance matrix **C**, and then on the choice of **K**, the kernel function. For solving this issue, there is a set of techniques based on the likelihood of $p(\mathbf{t}|\theta)$ where θ represents the hyperparameters of the Gaussian process. This likelihood process ends with a non-convex optimization problem with numerous maxima, and again we have to resort to approximations.

- **Gaussian processes: Example taken from "Gaussian Processes for Regression: a quick introduction", M. Ebden.**
  - Let us assume x∈R, with N=6, **x**=(-1.50, -1.00, -0.75, -0.40, -0.25, 0.00), with the response y (our t in the theory), and error bars of σ= 0.3 (γ^{-1}=1/0.3), and we want to estimate x*=0.2

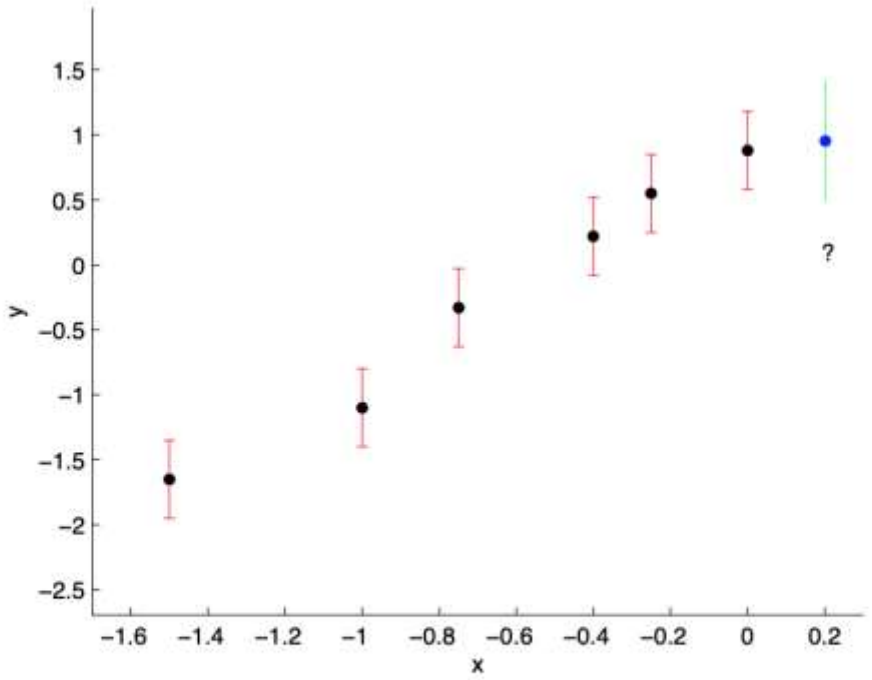


Figure 1: Given six noisy data points (error bars are indicated with vertical lines), we are interested in estimating a seventh at $x_* = 0.2$.

- **Gaussian processes: Example taken from Gaussian Processes for Regression: a quick introduction, M. Ebden.**
  - Let us assume x∈R, with N=6, **x**=(-1.50, -1.00, -0.75, -0.40, -0.25, 0.00), with the response y (our t in the theory), and error bars of σ= 0.3 ($\gamma^{-1}=1/\sigma^2$), and we want to estimate x*=0.2,
  - Choosing an squared exponential kernel defined as $k(\mathbf{x},\mathbf{y})= \sigma_f \exp(-1/(2l^2)(||\mathbf{x}-\mathbf{y}||^2)$ where the maximum allowable covariance is defined by σ_f, and choosing an appropriate σ_f and l,
  - Remember that with $\boldsymbol{C}_{N+1} = \begin{pmatrix} \boldsymbol{C}_N & \boldsymbol{k} \\ \boldsymbol{k}^T & c \end{pmatrix}$. Then, we calculate the covariance kernel matrix $\mathbf{C}_N= \mathbf{K}= K_{mn}$,

$$K = \begin{bmatrix} 1.70 & 1.42 & 1.21 & 0.87 & 0.72 & 0.51 \\ 1.42 & 1.70 & 1.56 & 1.34 & 1.21 & 0.97 \\ 1.21 & 1.56 & 1.70 & 1.51 & 1.42 & 1.21 \\ 0.87 & 1.34 & 1.51 & 1.70 & 1.59 & 1.48 \\ 0.72 & 1.21 & 1.42 & 1.59 & 1.70 & 1.56 \\ 0.51 & 0.97 & 1.21 & 1.48 & 1.56 & 1.70 \end{bmatrix}.$$

Moreover, $\mathbf{k}^T=k(x_n,x^*)$= (0.31, 0.68, 0.92, 1.25, 1.38, 1.54, and $c= k(x_{N+1},x_{N+1})$= 1.70.

- **Gaussian processes: Example taken from Gaussian Processes for Regression: a quick introduction, M. Ebden.**
  - Now, for x*=0.2

    $m(x_{N+1}) = \mathbf{k}^T\mathbf{C}^{-1}_N\mathbf{t} = 0.95$

    $\sigma^2(x_{N+1}) = c - \mathbf{k}^T\mathbf{C}^{-1}_N\mathbf{t} = 0.21$

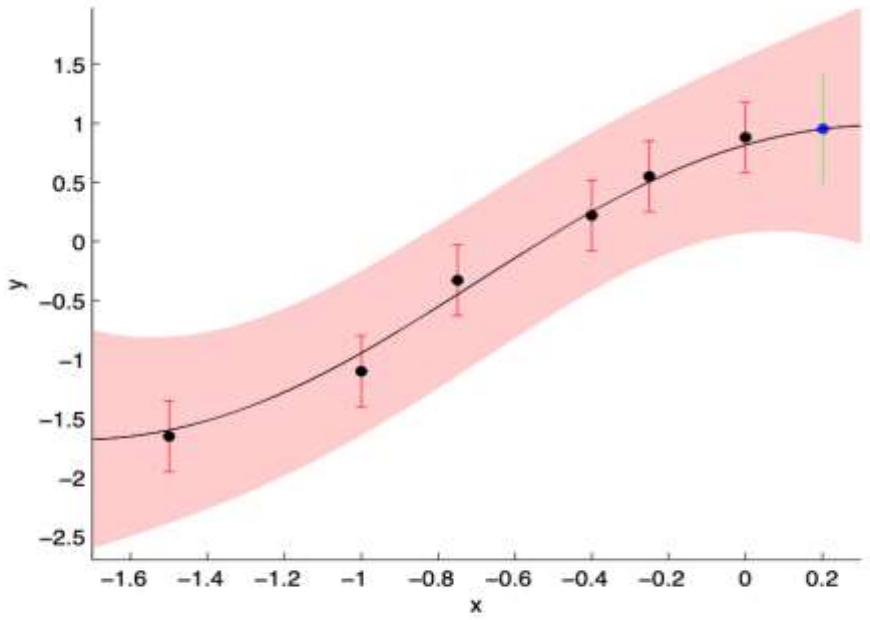


Figure 2: The solid line indicates an estimation of y_* for 1,000 values of x_*. Pointwise 95% confidence intervals are shaded.

- **Gaussian processes: Example taken from Gaussian Processes for Regression: a quick introduction, M. Ebden.**
  - How to select the hyperparameters ? In this example the hyperparameters are $\theta = \{\sigma_f, l, \sigma\}$.
  - The answer is to use the maximum likelihood (or MAP if you define a prior $p(\theta)$ in a Bayesian framework) over $p(\mathbf{t}|\mathbf{x}, \theta)$. The likelihood $p(\mathbf{t}|\mathbf{x}, \theta)$ is given by:

$$\ln p(\mathbf{t}|\mathbf{x}, \theta) = -1/2 \ln(|\mathbf{C}_N|) - 1/2\, \mathbf{t}^T\mathbf{C}^{-1}_N\mathbf{t} - N/2 \ln(2\pi),$$

Now, obtaining the derivative of $\ln p(\mathbf{t}|\mathbf{x}, \theta)$ with respect θ, we can obtain the parameters θ. We can also use our favourite gradient descent method to solve this.

In any case, take care because the problem is in general **non-convex**.

  - **Gaussian processes for classification:** GP for regression can be transformed to a GP for classification modifying the output of the GP to an appropriate non-linear activation function. We don't go in this problem in this course, but if you are interested in section 6.4.5 of Bishop's book you can find how to work it out.