Linear Regression

Machine Learning

Daniele Loiacono

POLITECNICO MILANO 1863

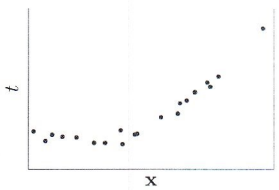
References

- This slides are based on material of prof. Marcello Restelli
- *Pattern Recognition and Machine Learning*, Bishop
  - Chapter 1 (1.1, 1.2, 1.3)
  - Chapter 3 (3.1, 3.3)

What is regression?

- Learn an **approximation** of function $f(x)$ that maps input x to a continuous output t from a dataset $\mathcal{D}$

$$\mathcal{D} = \{\langle x, t\rangle\} \Rightarrow t = f(x)$$

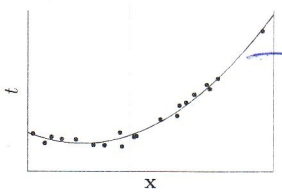


What is regression?

- Learn an **approximation** of function $f(x)$ that maps input x to a continuous output t from a dataset $\mathcal{D}$

$$\mathcal{D} = \{\langle x, t\rangle\} \Rightarrow t = f(x)$$

- How do we model f?
- How do we evaluate our approximation?
- How do we optimize our approximation?



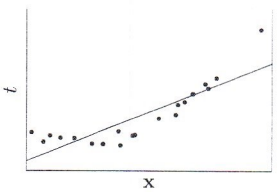
we need to decide how to model **f**: we need to find a model for the approximation for which we can search the parameters

true function that generates the points (plus some gaussian noise)

Note: we'll always have some noise, even in processes that are not stochastic

Linear Regression

- In linear regression, $f(x)$ is modeled with linear functions
  - Linear models can be easily **explained**
  - A linear regression problem can be solved **analytically**
  - Linear functions can be extended to model also **non-linear relationships**
  - More sophisticated methods are based on linear regression



we can solve the optimization problem analytically. With this property we've sure that if we don't achieve good results it's not because of the optimization algorithm (this is **very big**)

Linear Regression: model

- The simplest linear model can be defined as:

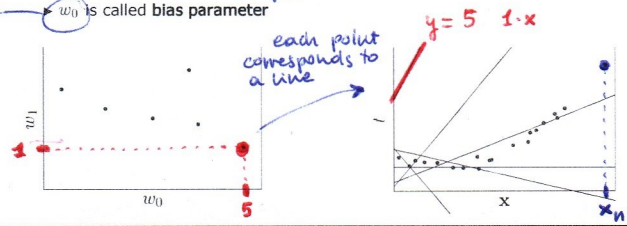
$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{D-1} w_j x_j = \mathbf{w}^T \mathbf{x}$$

  - $\mathbf{x} = (1, x_1, \ldots, x_{D-1})$: input
  - w_0 is called **bias parameter**

needed to model functions that have an offset if on the input

Suppose $D = 2$ (dimension). The dataset becomes:

$$D = \begin{bmatrix} 1, x_1 \\ 1, x_2 \\ \vdots \\ 1, x_n \end{bmatrix}$$



In this case the hypothesis space is the space of all the possible values of the two weights: w_0, w_1. One point in this hypothesis space (i.e. one couple (w_0, w_1)) represents a model: $y = w_0 + w_1 x$. How can we evaluate all of the models (points) and select the best one?

Remember: dimension = (1) + # variables (for the constant term (w_0))

Machine Learning — Daniele Loiacono

Linear Regression: loss function and optimization

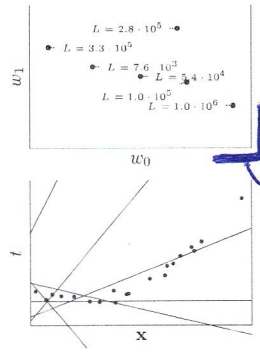
- A convenient **error loss function** is the **sum of squared errors** (SSE):

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, \mathbf{w}) - t_n)^2$$

  - the sum in L is also called **residual sum of squares** (RSS) and can be written as the sum of **residual errors**:

$$RSS(\mathbf{w}) = \|\epsilon\|_2^2 = \sum_{i=1}^{N} \epsilon_i^2$$

  - closed-form optimization of L can be easily obtained



Given a point (w_0, w_1) we can compute $L(\underline{w})$ by computing the sum of the squared errors that our approximation is doing w.r.t. the actual target value. The 1/2 term is just for convenience (when we'll optimize).

To each point (model, line) corresponds a value $L(\underline{w})$. The best model is the one for which $L(\underline{w})$ is smaller.

Machine Learning — Daniele Loiacono

Linear Models and Basis Functions :

improve linear models with basis functions

Machine Learning — Daniele Loiacono

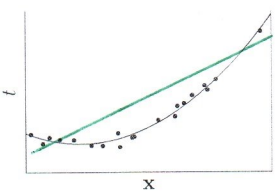
Linear Models

- A linear combination of the input variables is not enough to model data...
- ... but we just need a regression model that is **linear in the parameters**
- We can define a model using non-linear **basis functions**:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

  - $\boldsymbol{\phi}(\mathbf{x}) = (1, \phi_1(\mathbf{x}), \ldots, \phi_{M-1}(\mathbf{x}))^T$

The model y must be linear w.r.t. the weight vector $\underline{w}$. If so, we can still solve the optimization problem analytically and we can still explain the contribution of each term. How does it make sense? Because the loss function is optimized in $\underline{w}$ (because the loss function is a function in $\underline{w}$, is not a function w.r.t. the input variables $\underline{x}$)

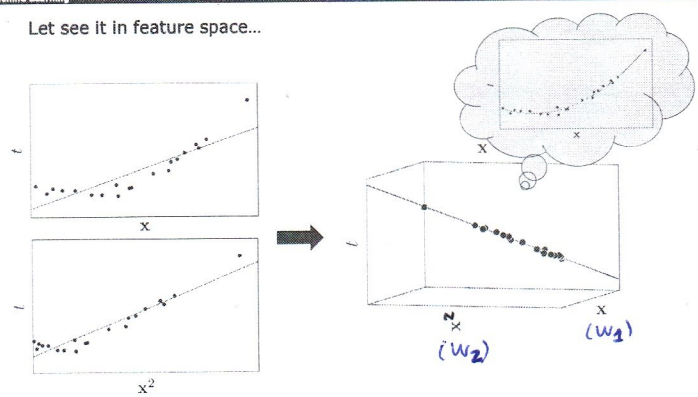


$y(x, w) = w_0 + w_1 x$

$\phi_j(\cdot)$ = nonlinear function, called basis function, that is just transformation of the original input. We can have how many we want of these basis functions.

Machine Learning — Daniele Loiacono

Let see it in feature space...



For example we can take the green line and add x^2: $y(x, w) = w_0 + w_1 x + w_2 x^2$. In this way:

$\phi_1(x) = x$

$\phi_2(x) = x^2$

The model is still linear

(in this case should be a plane, in D-dim it should be an hyperplane

Machine Learning — Daniele Loiacono

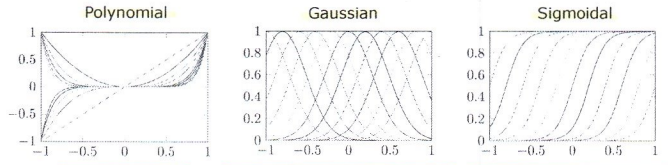
How can we find basis functions? We can do that in a problem-specific way (when we already have some knowledge about the problem and so we know that we could expect some linear correlation between the target and some specific mapping of the input variables) or we can try to apply some standard basis functions:

→ (the choice of which one to actually use in the model can be done only by comparing the performances of the model)
→ MODEL SELECTION

basis functions are a good way to add previous knowledge in the modelling

Basis Functions

- Some examples of basis function (assuming single-variable input)

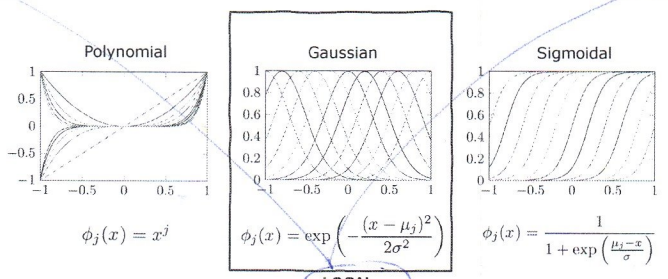


$\phi_j(x) = x^j$ | $\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2}\right)$ | $\phi_j(x) = \frac{1}{1+\exp\left(\frac{\mu_j - x}{\sigma}\right)}$

These functions come with a parameter: the grade of the polynomial is a parameter, the mean and variance of the gaussian are parameters, μ_j and σ in the sigmoidal are parameters. However, these parameters do not go in $\underline{w}$, they're not learned from data. These parameters are **design choices**.

Basis Functions

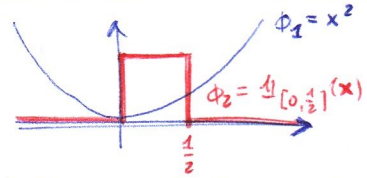
- Some examples of basis function (assuming single-variable input)



$\phi_j(x) = x^j$ | $\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2}\right)$ | $\phi_j(x) = \frac{1}{1+\exp\left(\frac{\mu_j - x}{\sigma}\right)}$

If we consider a polynomial basis function, e.g. $\phi(x) = x^2$, we have that $\phi(x) > 0$ in all the domain. In the case of gaussian we have that if we go far enough from the peak of the gaussian then the function goes almost to zero.

if values are here then their contribution is almost negligible (if values are more extreme then they're for sure negligible)
→ LOCAL (restricted to a neighborhood of the peak of the gaussian)



in the case $y(x,w) = w_0 + w_1\phi_1 \downarrow w_0 + w_1 x^2$

NO matter what is x, for sure it will contribute to y as $w_1 x^2$. In: $y(x,w) = w_0 + \phi_2 w_1$, we have that x will contribute as w_1 only if $x \in [0, \frac{1}{2}]$. That's the meaning of **LOCAL**.

How to find the best weight vector w.r.t. the loss function:

Least Squares

(in the case with basis functions, if we want the original version we consider the basis as identities)

Ordinary Least Squares

- For linear models, a closed-form optimization of the RSS, known as **least squares**, starting from the matrix form of the loss function:

$$L(\mathbf{w}) = \frac{1}{2}RSS(\mathbf{w}) = \frac{1}{2}(\mathbf{t} - \Phi\mathbf{w})^T(\mathbf{t} - \Phi\mathbf{w})$$

: $\underline{\varepsilon} := (\underline{t} - \underline{\Phi w})$ = vector of errors ($\underline{\varepsilon}^T\underline{\varepsilon} = \|\underline{\varepsilon}\|_2^2$)

  - where $\Phi = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_N))^T$ and $\mathbf{t} = (t_1, \ldots, t_N)^T$

$$\underline{\Phi} = \begin{bmatrix} 1 & \phi_1(\underline{x}^1) & \phi_2(\underline{x}^1) & \cdots & \phi_{M-1}(\underline{x}^1) \\ 1 & \phi_1(\underline{x}^2) & \phi_2(\underline{x}^2) & \cdots & \phi_{M-1}(\underline{x}^2) \\ \vdots & & & & \\ 1 & \phi_1(\underline{x}^n) & \phi_2(\underline{x}^n) & \cdots & \phi_{M-1}(\underline{x}^n) \end{bmatrix} = \begin{bmatrix} 1 & \phi_{11} & \phi_{12} & \cdots & \phi_{1(M-1)} \\ 1 & \phi_{21} & \phi_{22} & \cdots & \phi_{2(M-1)} \\ \vdots & & & & \\ 1 & \phi_{n1} & \phi_{n2} & \cdots & \phi_{n(M-1)} \end{bmatrix}, \quad \underline{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}, \quad \underline{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{M-1} \end{bmatrix}$$

each row is a sample

corresponding target for each sample

vector of weights

Ordinary Least Squares

- For linear models, a closed-form optimization of the RSS, known as **least squares**, starting from the matrix form of the loss function:

$$L(\mathbf{w}) = \frac{1}{2}RSS(\mathbf{w}) = \frac{1}{2}(\mathbf{t} - \Phi\mathbf{w})^T(\mathbf{t} - \Phi\mathbf{w})$$

: $L(\cdot)$ is quadratic w.r.t. $\underline{w}$

  - where $\Phi = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_N))^T$ and $\mathbf{t} = (t_1, \ldots, t_N)^T$

$$\frac{\partial(f(x)^2)}{\partial x} = 2f(x)\cdot f'(x)$$

- We can compute first a second derivative of $L(w)$ to find the optimal w

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\Phi^T(\mathbf{t} - \Phi\mathbf{w}) \qquad \frac{\partial^2 L(\mathbf{w})}{\partial \mathbf{w}\partial \mathbf{w}^T} = \Phi^T\Phi$$

The concept of convexity is achieved only when this matrix is positive semidefinite (the eigenvalues are ≥ 0). A matrix of the form $\underline{\Phi}^T\underline{\Phi}$ is for sure positive semidef.

Since we're in a ∪ situation, the stationary point will be a minimum

$$\frac{\partial L(\underline{w})}{\partial \underline{w}} = 0 \Rightarrow \hat{\mathbf{w}}_{OLS} = \left(\Phi^T\Phi\right)^{-1}\Phi^T\mathbf{t}$$

→ ∪ ✓ ✗

Note: $\Phi^T\Phi$ is positive semidefinite. In order to invert the matrix we need it to be positive definite
→ $\hat{w}_{OLS}$ exists when $\Phi^T\Phi$ is invertible, and so when it is positive definite (all eigenvalues > 0)

When we have an eigenvalue equal to zero? (not invertible) for example when we have linear dependence among features (we can identify the case and remove the redoundant feature).
The other case (for eigen = 0) is that we have more features than data.

Ordinary Least Squares

- For linear models, a closed-form optimization of the RSS, known as **least squares**, starting from the matrix form of the loss function:

$$L(\mathbf{w}) = \frac{1}{2}RSS(\mathbf{w}) = \frac{1}{2}(\mathbf{t} - \mathbf{\Phi w})^T(\mathbf{t} - \mathbf{\Phi w})$$

  - where $\mathbf{\Phi} = (\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_N))^T$ and $\mathbf{t} = (t_1, \ldots, t_N)^T$

- We can compute first a second derivative of $\mathcal{L}(w)$ to find the optimal w

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{\Phi}^T(\mathbf{t} - \mathbf{\Phi w}) \qquad \frac{\partial^2 L(\mathbf{w})}{\partial \mathbf{w}\partial \mathbf{w}^T} = \mathbf{\Phi}^T\mathbf{\Phi}$$

$$\Rightarrow \quad \mathbf{w}_{OLS} = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t}$$

Assuming $(\Phi^T\Phi)$ non singular
Complexity $O(NM^2 + M^3)$ — It's very expensive

Sequential Learning / Online learning

: when we don't have all the data in advance and we want to update the training each time we get additional data **or** when we cannot perform OLS because it's not feasible

- Closed-form optimization (OLS) is not feasible with large dataset
- Instead, a **stochasitcs** (or **sequential**) gradient descent is possible
- **Least Mean Square** (LMS) algorithm:

$$L(\mathbf{x}) = \sum_n L(x_n)$$

$$\Rightarrow \quad \mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \alpha^{(n)}\nabla L(x_n)$$

$$\Rightarrow \quad \mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \alpha^{(n)}\left(\mathbf{w}^{(n)T}\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right)\boldsymbol{\phi}(\mathbf{x}_n)$$

  - α is called learning rate and to guarantee convergence: *

$$\sum_{n=0}^{\infty}\alpha^{(n)} = +\infty \quad \text{(not too fast)} \qquad \sum_{n=0}^{\infty}\alpha^{(n)^2} < +\infty \quad \text{(not too slow)}$$

can be variable (adjustable) (we expect it to decrease)

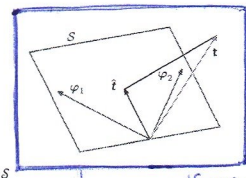
We work on one sample. We compute the loss function on the one sample we're working on. We compute the derivative of the loss function w.r.t. only that one sample and we apply a correction:

Why is it called stochastic gradient descent? Because the loss function computed on a single data point is not equal to the loss function computed on the whole data points. And so the gradient computed on one point is different from the gradient computed on the whole set.

* We want the learning rate to not decrease too fast (because if so we don't have enough time to reach the optimal solution) but we want it to decrease fast enough (otherwise we don't have convergence).

Geometric Interpreation of OLS

- Let t be the N-dimensional target vector
- Let φ_j be the j-th column of matrix Φ
  - $\varphi_1, \ldots, \varphi_M$ identify a linear subspace $\mathcal{S}$
- Let $\hat{t}$ be the N-dimensional vector computed as Φw
  - $\hat{t}$ is a linear combination of φ_j and lies in $\mathcal{S}$
- OLS finds $\hat{t}$ minimizing the SSE with respect to t
  - $\hat{t}$ represents the projection of t onto the subspace $\mathcal{S}$



We look at Φ as:

$$\Phi = \left[\underline{\varphi}_1\ \underline{\varphi}_2 \cdots \underline{\varphi}_{M-1}\right]$$

feature 1 computed for all the samples in the dataset

$$\hat{\mathbf{t}} = \mathbf{\Phi}\hat{\mathbf{w}} = \mathbf{\Phi}\left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t}$$

Hat Matrix (H)

output of our model: $\hat{\underline{t}} = \underline{y}(\Phi, \hat{w}_{OLS})$

this means that $\hat{\underline{t}}$ is just a linear combination of the columns of Φ

If we only have 2 features then $\hat{\underline{t}}$, which is a linear combination of $\underline{\varphi}_1$ and $\underline{\varphi}_2$, will be on the plane generated from $\underline{\varphi}_1$ and $\underline{\varphi}_2$. However, $\underline{t}$ (the true target vector) is not constrained to be on the plane. What we can guarantee (minimizing the squared error between $\hat{\underline{t}}$ and $\underline{t}$) is that $\hat{\underline{t}}$ is the projection of $\underline{t}$ on the subspace generated by $\underline{\varphi}_1$ and $\underline{\varphi}_2$. This means that hat matrix is a (linear) projector. The properties of our approximation depend on how we design our features space. Depending on how we choose the features space we can get a better/worse approximation.

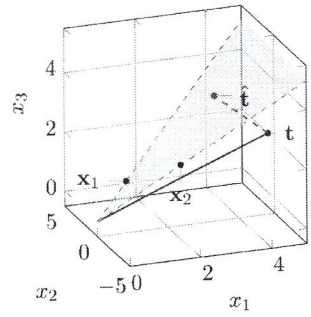
Geometric Interpreation of OLS: an example

- Let N=3 and M=2

$$\mathbf{\Phi} = \mathbf{X} = \begin{pmatrix} \underline{\varphi}_1 & \underline{\varphi}_2 \\ 1 & 2 \\ 1 & -2 \\ 1 & 2 \end{pmatrix}$$

Once we fix $\underline{\varphi}_1$ and $\underline{\varphi}_2$ we cannot get an approximation that is not a linear comb. of $\underline{\varphi}_1$ and $\underline{\varphi}_2$. This leads to the fact that the approx of the target for sample 1 and sample 3 will be the same (since they're identical).

$$\mathbf{t} = \begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix} \quad \text{real target} \qquad \hat{\mathbf{t}} = \begin{pmatrix} 3.5 \\ 1 \\ 3.5 \end{pmatrix} \quad \text{approx.}$$



There are two things we cannot take into account: if the target depends on some variables that are not represented by the feature and the noise (always present).

Multiple Outputs → we want to predict multiple values

- What happens if our regression problem has multiple outputs, i.e., **t** is not scalar
- It is possible to solve independently a regression problem for each problem
- Yet, it is possible to use the same set of basis functions:

$$\hat{\mathbf{W}} = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{T} \qquad \mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ \vdots & & & \\ t_{n1} & t_{n2} & \cdots & t_{nm} \end{bmatrix}$$

t_{1m}: m-th target of the first sample
t_{nm}: m-th target of the n-th sample

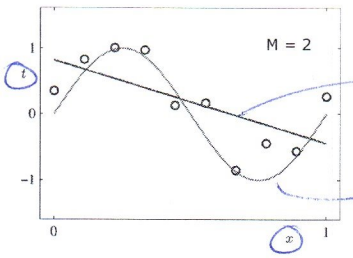
  - Where each column of matrix **T** and $\hat{W}$ are respectively the target vector of each and the weight vector for each output
- The solution above can be easily **decoupled** for each output **k**: (t_k is a vector)

$$\hat{\mathbf{w}}_k = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t}_k$$

  - as a benefit $(\Phi^T\Phi)^{-1}$ can be computed only **once**

Regularization

How do we design the linear models? An example



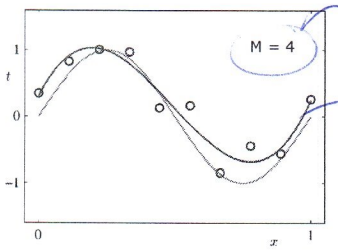
1-order polynomial model

approximation of $g(\cdot)$: $y = w_0 + w_1 x$

real function from which we have the samples

$t = g(x) + \varepsilon$: t = targets (samples), $g(\cdot)$ = true function

How do we design the linear models? An example



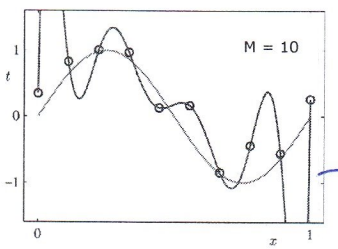
3-order polynomial model

features (including the constant)

approximation obtained with least squares using a 3-order polynomial model

$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

How do we design the linear models? An example



9-order polynomial model

With this model the precision is accurate only in the already given samples → not good

How can we mitigate? REGULARIZATION

What is regularization?

- What happens to model parameters when complexity increases?

	$M=1$	$M=2$	$M=3$	$M=10$
$\hat{w}_0$	0.19	0.82	0.31	0.35
$\hat{w}_1$		-1.27	7.99	232.37
$\hat{w}_2$			-25.43	-5321.83
$\hat{w}_3$				48568.31
$\hat{w}_4$				-231639.30
$\hat{w}_5$				640042.26
$\hat{w}_6$				-1061800.52
$\hat{w}_7$				1042400.18
$\hat{w}_8$				-557682.99
$\hat{w}_9$				125201.43

As we add parameters we can see that their value increases (in absolute value). This is because to have a highly-changing function we have to have something that changes a lot w.r.t. x → highly-valued weights (big values in absolute value). To keep the function smooth (and not so "crazy") we can add an additional term to the loss function that has the role of keeping the weights as small as possible.

Regularization

- How do we extend the loss function?

$$L(\mathbf{w}) = L_D(\mathbf{w}) + \lambda L_W(\mathbf{w})$$

this additional part has the role of keeping the weights small

  - $L_D(w)$ is the usual loss function (e.g., RSS)
  - $L_W(w)$ accounts for model complexity
  - λ is the **regularization** coefficient → the higher the coefficient the more important will be to keep the weights small

This is not a parameter. This is a HYPERPARAMETER

- How do we design $L_w(w)$?
  - Ridge Regression
  - Lasso

Again: why $\underline{w}$ should be small?
if $\underline{w}$ is small then to small variations in the features space will correspond small variations in the target space

Ridge Regression

- In **ridge regression**:

$$L_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} = \frac{1}{2}\|\mathbf{w}\|_2^2$$

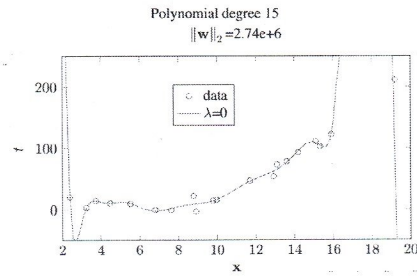
$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - \mathbf{w}^T\phi(\mathbf{x}_i)\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

$\|\underline{w}\|_2^2 = \sum_{j=0}^{M-1} w_j^2$

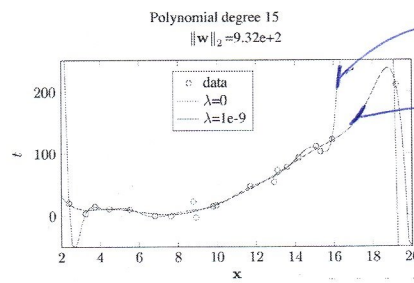
- The loss function is still quadratic with respect to w and closed-form optimization is still possible:

$$\hat{\mathbf{w}}_{ridge} = \left(\lambda\mathbf{I} + \mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t}$$

Ridge Regression: quadratic example



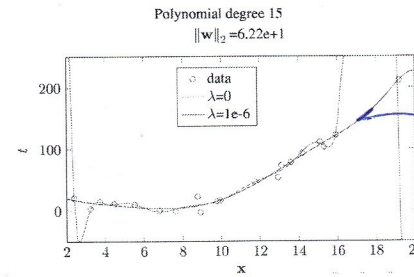
Ridge Regression: quadratic example



no regularization

ridge regression with $\lambda = 1e-9$

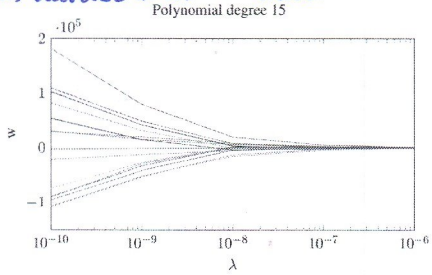
Ridge Regression: quadratic example



ridge regression with $\lambda = 1e-6$

Ridge Regression: quadratic example

What happens with the increasing of λ?
The weights decrease in absolute value.



Lasso

- Another common regularization method is **lasso**:

$$L_W(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_1 = \frac{1}{2}\sum_{j=0}^{M-1}|w_j|$$

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_1 \quad : \|\mathbf{w}\|_1 = \sum_{j=0}^{M-1}|w_j|$$

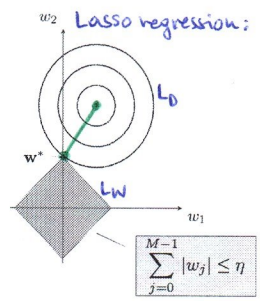
- In this case, closed-form optimization is not possible
- Nevertheless, lasso typically leads to **sparse** regression models: when regularization coefficient (λ) is large enough, some **components** of $\hat{\mathbf{w}}$ become **equal to zero**
- We can see regularization equivalent to minimizing $\mathcal{L}_D(w)$ subject to constraint:

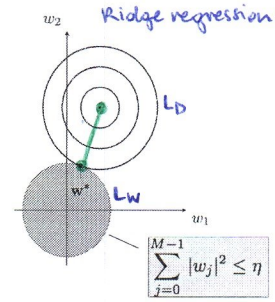
$$\sum_{j=0}^{M-1}|w_j| \leq \eta$$

This is very good. If we can have less components then we have simplest models and easier to understand

Lasso vs Ridge Regression

- Why lasso leads to **sparse** model? Let's visualize constraint of lasso and ridge





Considering the same η for the two cases, in the lasso case it is more likely to end up on one axis.

Until now we considered a direct approach. What if we want to procede in a probabilistic way?

Least Squares and Maximum Likelihood

(probabilistic approach)

Maximum Likelihood (ML)

- We can deal with regression in a **probabilistic way**
  - We define a probabilistic model that maps inputs (x) to outputs (t)
  - Such probabilisic model, $y(x, w)$, will include some **unknown parameters** (w)
  - Then, we model the **likelihood**, i.e., the probability that observed data $\mathcal{D}$ is generated by a given set of **parameters** (w): (under the chosen probabilistic model)

$$p(\mathcal{D}|\mathbf{w})$$

  - Finally, we can estimate **parameters** (w) by maximizing the likelihood:

$$\mathbf{w}_{ML} = \arg\max_{\mathbf{w}} p(\mathcal{D}|\mathbf{w})$$

Maximum Likelihood (ML) for linear regression

- Our probabilistic model can be defined as:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + \epsilon$$

  - we assumed a linear model for $y(x, w)$
  - we assumed $\epsilon \sim \mathcal{N}(0, \sigma^2)$

  + Hp. all the points are independent among each other (iid)

Given these assumptions and the data we observed, we can evaluate the weights $\underline{w}$. How? We choose the weights $\underline{w}$ that maximize the probability of the data.

- Given a dataset $\mathcal{D}$ of N samples with inputs $\mathbf{X}=\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and ouptuts $\mathbf{t}=\{t_1,\ldots,t_N\}^T$:

$$p(\mathcal{D}|\mathbf{w}) = p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}\left(t_n|\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n), \sigma^2\right)$$

(known, unknown) → goal: maximize this probability w.r.t. $\underline{w}$

We want to find:

$$\underline{w}_{ML} = \arg\max_{\underline{w}} p(D|\underline{w})$$

How? Instead of maximizing $p(D|\underline{w})$ we maximize its logarithm.

Maximum Likelihood (ML) for linear regression (cont.)

- To find $\mathbf{w}_{ML}$ it is convenient to maximize the log-likelihood:

$$\mathcal{N}\left(t_n|\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n), \sigma^2\right) = \frac{1}{(2\pi\sigma^2)^{1/2}} exp\left\{-\frac{1}{2\sigma^2}(t_n - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n))^2\right\}$$

$$\ell(\mathbf{w}) = \ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \sum_{n=1}^{N} \ln p(t_n|\mathbf{x}_n, \mathbf{w}, \sigma^2) = -\frac{N}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}RSS(\mathbf{w})$$

RSS: residual sum of squares

- To solve the optimization problem, we equal the gradient to zero:

$$\nabla\ell(\mathbf{w}) = \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n)^T - \mathbf{w}^T\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^T\right) = 0$$

$$\mathbf{w}_{ML} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}$$

OLS

This means that when we apply Ordinary least Squares we implicitly do the maximum likelihood approach (with the assumptions of linearity and $\epsilon \sim N(0, \sigma^2)$)

This does not provide any tool to get an estimate of the uncertainty level that we have on the estimates

Bayesian Linear Regression

Bayesian approach

1. We formulate our knowledge about the world in **a probabilistic way**:
   i. We define the **model** that expresses our knowledge qualitatively
   ii. Our model will have some **unknown parameters**
   iii. We capture our **assumptions** about unknown parameters with the **prior distribution** over those parameters before seeing the data
2. We observe the **data**
3. We compute the posterior probability distribution for the parameters, given observed data

$$p(parameters|data) = \frac{p(data|parameters)p(parameters)}{p(data)}$$

4. We use the posterior distribution to:
   a. Make predictions by averaging over the posterior distribution
   b. Examine/Account for uncertainty in the parameter values
   c. Make decisions by minimizing expected posterior loss

prior on the parameters (before we observe the data what do we know?)

This allows not only to find the most probable value of weights but also to assess the uncertainty that we have in our estimate of the parameters.

overall likelihood of the data. We don't model it, we obtain it by marginalizing $p(data|param)$

Parameters Posterior Distribution

- The **posterior distribution** for the model parameters can be found by combining the prior with the likelihood for the parameters given data:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$$

(Bayes theorem)

Parameters Posterior Distribution

- The **posterior distribution** for the model parameters can be found by combining the prior with the likelihood for the parameters given data:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$$

prior

  - $p(\mathbf{w})$ is the **prior** probability over the parameter – what we know before observing the data

Parameters Posterior Distribution

- The **posterior distribution** for the model parameters can be found by combining the prior with the likelihood for the parameters given data:

likelihood

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$$

  - $p(D|\mathbf{w})$ is the **likelihood** – the probability of observing the data (D) given some value of the parameters (**w**)

Parameters Posterior Distribution

- The **posterior distribution** for the model parameters can be found by combining the prior with the likelihood for the parameters given data:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$$

normalizing constant

$$p(\mathcal{D}) = \int p(\mathcal{D}|\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}$$

  - $P(D)$ is the **marginal likelihood** and acts as **normalizing constant**

Parameters Posterior Distribution

- The **posterior distribution** for the model parameters can be found by combining the prior with the likelihood for the parameters given data:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$$

posterior

  - $p(\mathbf{w}|D)$ is the **posterior probability** of parameters w given training data
  - the most probable value of **w** given the data will be the mode of the posterior, also known as **maximum a posteriori** (MAP).

Bayesian Linear Regression

- How to model the **prior**? Assuming a Gaussian likelihood, a **conjugate prior** is the most convenient choice:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_0, \mathbf{S}_0)$$

→ when we choose a prior we have to choose a mean and a variance for the vector of weights. How to choose them? it depends on the situation. Often this choice is based on previous knowledge. This settings allow easily to include in the model any previous knowledge we have on the problem. However we can choose what we want for $\underline{w}_0$ and S_0.

- As a result, the **posterior** is still Gaussian:

$$p(\mathbf{w}|\mathbf{t}, \Phi, \sigma^2) \propto \mathcal{N}(\mathbf{w}|\mathbf{w}_0, \mathbf{S}_0)\mathcal{N}(\mathbf{t}|\Phi\mathbf{w}, \sigma^2\mathbf{I})$$

(MOST NEUTRAL CHOICE :)
If we don't have any previous knowledge it's good to choose $\underline{w}_0 = \underline{0}$ and $S_0 = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} = \sigma^2 I$

This is also the choice that we do for the likelihood ($\mathcal{N}(\underline{t}|\Phi\underline{w}, \sigma^2 I)$)

no dependencies between the weights and same variance for all the weights

Bayesian Linear Regression

- How to model the **prior**? Assuming a Gaussian likelihood, a **conjugate prior** is the most convenient choice:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_0, \mathbf{S}_0)$$

- As a result, the **posterior** is still Gaussian:

$$p(\mathbf{w}|\mathbf{t}, \mathbf{\Phi}, \sigma^2) \propto \mathcal{N}(\mathbf{w}|\mathbf{w}_0, \mathbf{S}_0)\mathcal{N}(\mathbf{t}|\mathbf{\Phi}\mathbf{w}, \sigma^2\mathbf{I})$$

$$\Rightarrow \begin{cases} p(\mathbf{w}|\mathbf{t}, \mathbf{\Phi}, \sigma^2) = \mathcal{N}(\mathbf{w}|\mathbf{w}_N, \mathbf{S}_N) \\ \mathbf{w}_N = \mathbf{S}_N\left(\mathbf{S}_0^{-1}\mathbf{w}_0 + \frac{\mathbf{\Phi}^T\mathbf{t}}{\sigma^2}\right) \\ \mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} + \frac{\mathbf{\Phi}^T\mathbf{\Phi}}{\sigma^2} \end{cases}$$

← conjugate → still gaussian

Note: if we have a strong assumption in the prior, this strong assumption will be reflected also in the posterior

How to represent no previous knowledge? $\underline{w}_0 = \underline{0}$ and S_0 = infinity (so we're saying we are infinitely not sure about what we're saying). In this way the posterior mean and variance will depend only on the data:

$$\begin{cases} \underline{w}_N = S_N \cdot \frac{\Phi^T \underline{t}}{\sigma^2} \\ S_N^{-1} = \frac{\Phi^T\Phi}{\sigma^2} \end{cases}$$

which, if we substitute, brings:

$$\underline{w}_N = (\Phi^T\Phi)^{-1}\Phi\underline{t}$$

which is again **OLS!**

So, maximum likelihood / OLS is the specific case of the bayesian regression framework that corresponds to the case where we don't have any previous knowledge on weights.

Machine Learning — Daniele Loiacono

Bayesian Linear Regression an Maximum Likelihood

- Which parameters?
  - The maximum a-posteriori (**MAP**) is the obvious choice
  - When posterior is gaussian the **MAP** is equal to the mean
- When prior is **infinitely broad** MAP is equal to ML solution:

$$\lim_{\mathbf{S}_0 \to \infty} \mathbf{w}_N = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t} = \mathbf{w}_{ML}$$

$$\lim_{\mathbf{S}_0 \to \infty} \mathbf{S}_N^{-1} = \frac{\mathbf{\Phi}^T\mathbf{\Phi}}{\sigma^2}$$

$$\hat{\sigma}^2 = \frac{1}{N-M}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n))^2$$

  - The ML estimate of **w** has the **smallest variance** among linear unbiased estimates and the **lowest MSE** among linear **unbiased** estimates (Gauss-Markov). (theorem)

The great thing about bayesian approach is that, unlikely in the case of OLS or maximum likelihood, here we get two estimates: $\underline{w}_N$, S_N. We always got only $\underline{w}_N$. Now we have an estimate of the UNCERTAINTY we have on the weight vector.

Most easiest and **UNBIASED** estimate that we can get from data (notice that we have N-M because with M we account the degrees of freedom)

Machine Learning — Daniele Loiacono

Bayesian Linear Regression and Regularization

- What about regularization?
  - When $\mathbf{w}_0$=0 and $\mathbf{S}_0=\tau^2\mathbf{I}$, we have:

$$\ln p(\mathbf{w}|\mathbf{t}) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(t_i - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)\right)^2 - \frac{1}{2\tau^2}\|\mathbf{w}\|_2^2$$

  - In this case, **MAP** ($\mathbf{w}_N$) is equivalent to the solution of ridge regression ($\hat{\mathbf{w}}_{ridge}$) with $\lambda = \frac{\sigma^2}{\tau^2}$

The maximum aposteriori ($\underline{w}_N$) is the same as the result obtained with ridge regression if we assume the prior to have $\underline{0}$ mean and covariance matrix = $\tau^2\mathbf{I}$. In this case it'll result: $\lambda = \sigma^2/\tau^2$.

⇒ Bayesian regression also allows to account for regularization. Regularization can be seen as specific case of bayesian regression for which we make specific decisions for the prior.

MOREOVER Bayesian regression can be used "incrementally".

Machine Learning — Daniele Loiacono

Bayesian Linear Regression: an example

- How to exploit the Bayesian approach for sequential learning?
  - We compute **posterior** with initial data
  - When additional data is available, the **posterior** becomes the **prior**

$p(\underline{w}|D) = N(\underline{w}_N, S_N)$

then we obtain new data: D':

$p(\underline{w}'|D') = p(D'|\underline{w}')p(\underline{w}')$

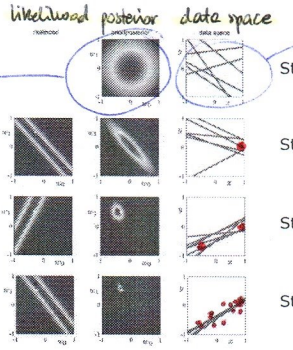
Machine Learning — Daniele Loiacono

Bayesian Linear Regression: an example

- How to exploit the Bayesian approach for sequential learning?
  - We compute **posterior** with initial data
  - When additional data is available, the **posterior** becomes the **prior**
- Let see an example
  - Let assume data is generated as $t(x) = -0.3 + 0.5x + \varepsilon$
  - Let assume that $x \sim U(-1,1)$ and $\varepsilon \sim \mathcal{N}(0,0.04)$
  - Let use as model: $y(x,\mathbf{w}) = w_0 + w_1 x$
  - Let assume as prior: $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}_0, \tau^2\mathbf{I})$ with $\tau^2 = 0.5$ and $\mathbf{w}_0 = [0,0]^T$ → this means that we're basically doing ridge regression

Machine Learning — Daniele Loiacono

Bayesian Linear Regression: an example (2)



these are all the possible models that we can sample from the posterior distribution

At the first step the posterior is basically the prior because we don't have any samples

(*) **Note**: likelihood and posterior have the weights w_0 and w_1 on the axis. We sample a couple from the joint posterior distribution of w_0 and w_1, we obtain a sample and so a line. The line is represented in the data space

as we keep having samples we get a better posterior and so the samples from the posterior (models) become more accurate

Machine Learning — Daniele Loiacono

Predictive Distribution

- With a Bayesian framework, it is possible to compute the probability distribution of the target for a new sample $\mathbf{x}^*$ (given the training data $\mathcal{D}$), by **integrating over the posterior distribution**:

$$p(t^*|\mathbf{x}^*, \mathcal{D}) = \mathbb{E}[t^*|\mathbf{x}^*, \mathbf{w}, \mathcal{D})] = \int p(t^*|\mathbf{x}^*, \mathbf{w}, \mathcal{D}) p(\mathbf{w}|\mathcal{D}) \mathrm{d}\mathbf{w}$$

MAP

- This is often called **predictive distribution**
- Unfortunately, computing the predictive distribution requires knowledge of the posterior distribution, which is usually **intractable**

However, in our case (linear bayesian regression, gaussian distribution for the prior, gaussian likelihood, etc.) we can obtain a solution analitically:

Machine Learning — Daniele Loiacono

Predictive Distribution for Bayesian Regression

(Model assumptions: $y(x) = \mathbf{w}^T \phi(x) + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2)$)

- With our assumptions, we can compute the **predictive distribution:**

$$p(t|\mathbf{x}, \mathcal{D}, \sigma^2) = \int p(t|\mathbf{x}, \mathbf{w}, \sigma^2) p(\mathbf{w}|\mathbf{w}_N, \mathbf{S}_N) \mathrm{d}\mathbf{w} = \int \mathcal{N}\left(t|\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}), \sigma^2\right) \mathcal{N}\left(\mathbf{w}|\mathbf{w}_N, \mathbf{S}_N\right) \mathrm{d}\mathbf{w}$$

$$p(t|\mathbf{x}, \mathcal{D}, \sigma^2) = \mathcal{N}\left(t|\mathbf{w}_N{}^T \boldsymbol{\phi}(\mathbf{x}), \sigma_N^2(\mathbf{x})\right)$$

$$\sigma_N^2(\mathbf{x}) = \sigma^2 + \boldsymbol{\phi}(\mathbf{x})^T \mathbf{S}_N \boldsymbol{\phi}(\mathbf{x})$$

data — parameters

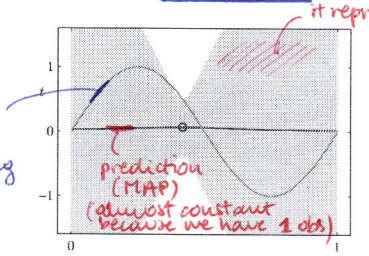
the mean does not change (it still the MAP), however we're interested in $\sigma_N^2(x)$ because that is the one that will tell about variance (thanks to that we can construct a confidence interval!)

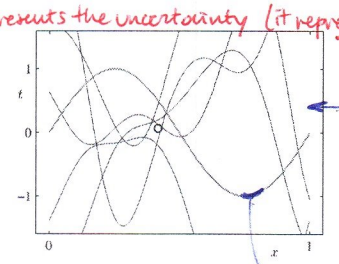
- when $N \to \infty$ the uncertainty associated to parameters (second term) goes to zero and the variance of predictive distribution depends only on variance of data (σ^2) (we cannot reduce σ^2: we cannot have a degree of uncertainty on the target that is less than the actual variance of the data)

Machine Learning — Daniele Loiacono

Predictive Distribution: an example

- Approximating a sinusoidal dataset with a linear model w/ 9 Gaussian basis functions: 1 sample observed





it represents the uncertainty (it represents a confident interval (pointwise)) The further from the observation, the larger the uncertainty

true function generating data

prediction (MAP) (almost constant because we have 1 obs)

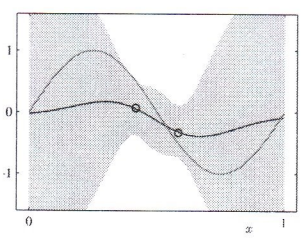
some possible functions that we can draw from the posterior (*)

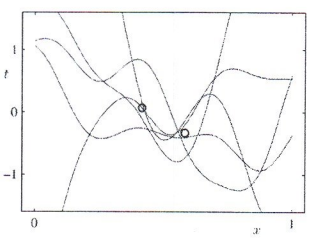
true function

Machine Learning — Daniele Loiacono

Predictive Distribution: an example

- Approximating a sinusoidal dataset with a linear model w/ 9 Gaussian basis functions: 2 samples observed

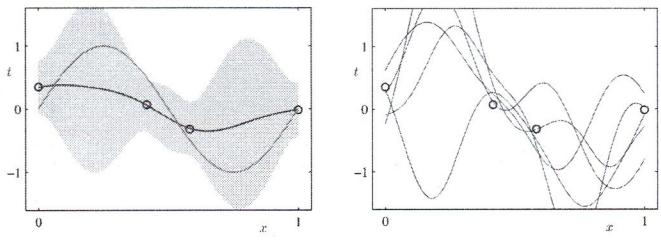




Machine Learning — Daniele Loiacono

Predictive Distribution: an example

- Approximating a sinusoidal dataset with a linear model w/ 9 Gaussian basis functions: 4 samples observed

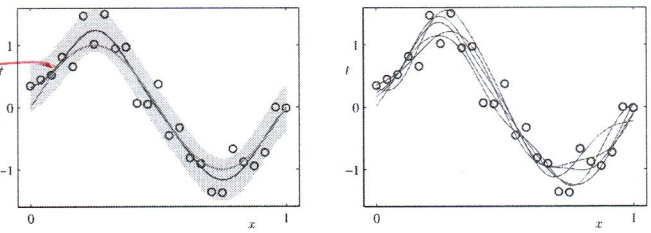


Predictive Distribution | Sampled Models

Predictive Distribution: an example

- Approximating a sinusoidal dataset with a linear model w/ 9 Gaussian basis functions: 25 samples observed

much smaller confidence intervals!



Predictive Distribution | Sampled Models

Challenges and Limitation of Linear Regression

Challenges

- Modeling Challenges
  - Our model should fit well all the functions (we think) are **likely**
  - The prior should not give zero or small probabilities to possible values, but at the same time also avoid spreading out the probability (**uninformative**)
- Computational Challenges
  - **Analytical integreation** is possible only using **conjugate priors** and works for **simple models**
  - **Approximated** approaches are instead to be used in the more general case, such as Gaussian (Laplace) approximation, Monte Carlo integration, and Variational approximation

Limitation of Fixed Basis Functions

- Linear models with fixed basis functions have several advantages
  - Allow closed-form solution
  - Tractable **Bayesian treatment**
  - Can model non-linear relationship with proper basis function
- However, they have also several limitations
  - Basis functions are not *adaptive* with respect to the training data
  - These models suffer of the **curse of dimensionality**

BIGGEST INCONVENIENT: (we cannot adapt $\phi_j(\cdot)$ to the data, we have to choose all of them in advance)