Statistical Learning

Scribe : Antoine Pouille

October 3, 2014

Contents

1 Supervised learning, course of October the 3rd

Notations for these courses: n: number of training points, p: number of dimensions of the points

1.1 Main principles seen in the previous classes

Training data: (x_i, y_i) where

- $x_i \in \mathbb{R}^p$
- y_i are **labels**:
  - $\{-1, +1\}$ for **binary classification**
  - $\mathbb{R}$ for **regression**
  - $\{1, .., M\}$ for **multiclass classification**

1.1.1 Goal

Learn some function $\hat{f}$ such that $\hat{f}(x) \simeq y$ for a new data x and its unknown label y.

For instance, linear functions:

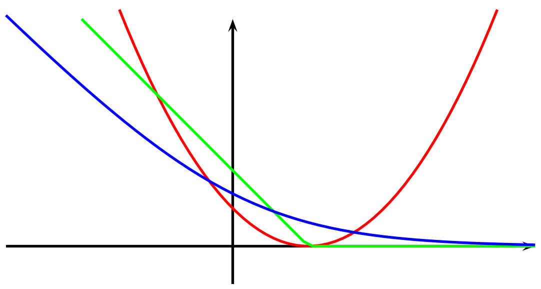
- $\hat{f}(x) = \theta^T x + b$ (regression)
- $\hat{f}(x) = sign(\theta^T x + b)$ (classification)

1.1.2 Empirical risk minimisation

$$\min_{\theta \in R^p} \frac{1}{n} \sum_{i=1}^{n} L(y_i, \theta^T x_i) + \lambda \Omega(\theta)$$

Example of loss functions L:

- $L(u, \hat{u}) = (u - \hat{u})^2$ (ridge)
- $log(1 - e^{-u\hat{u}})$ (logistic)
- $max(1 - u\hat{u}, 0)$ (hinge)



Regularisation term:

- $\Omega(\theta) = ||\theta||_2^2$ (ridge)
- $\Omega(\theta) = ||\theta||_1$ (lasso)

1.1.3 Bias-variance decomposition

Assume $Y = f(X) + \epsilon$ with $E(\epsilon) = 0$

Let us consider some test point x, with a label y to predict:
$E[(y - \hat{f}(x))^2] = E[(y - f(x))^2] + E[(\hat{f}(x) - E(\hat{f}(x)))^2] + E[(f(x) - E(\hat{f}(x)))^2]$
Bayes error, variance, squared bias
Cross-validation

1.2 Optimisation principles

1.2.1 Basic definitions

$f : \mathbb{R}^p \to \mathbb{R}$

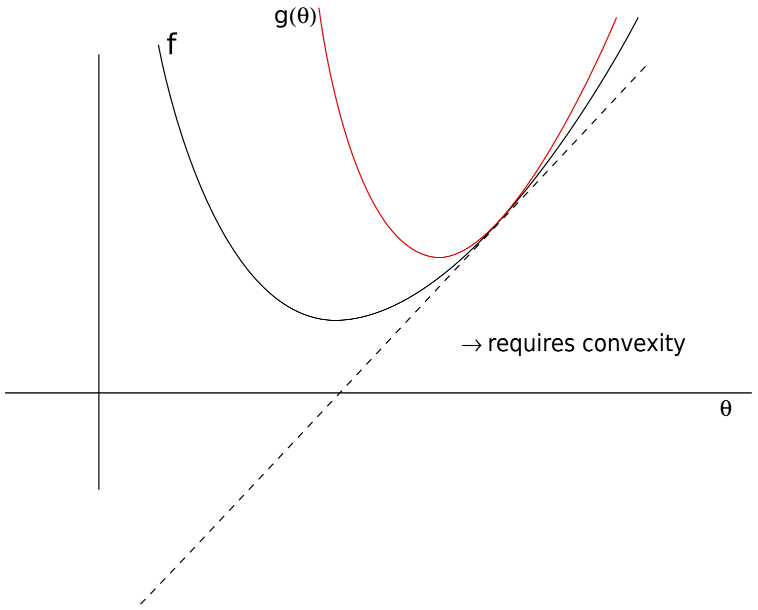
Gradient: $\nabla f = (\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_p})$

Convex functions

$f^* = min_{\theta \in R^p} f(\theta)$

Properties of convex functions:

- local optimum is global
- f is above its tangents: $f(\theta) \geq f(\theta_0) + (\nabla f(\theta_0))^T(\theta - \theta_0)$
- if f is differentiable, $\nabla f(\theta) = 0$ iff θ is a global optimum
- $f(\theta) - f^*$ is easy to control through duality gaps



1.2.2 Linear systems

Simpler than optimising a function: find x such that $Ax = b$. Just use LAPACK in your favourite language.

If $A \in \mathbb{R}^{n \times n}$ then solving is $O(n^3)$ in general

If n is very large, minimise $min_{x \in R^n} ||Ax - b||_2^2$ by using a conjugate gradient (approximate solution)

Example of ridge regression, where X is a n times p matrix:

$min_{\theta \in R^p} ||Y - X\theta||_2^2 + \lambda ||\theta||_2^2$

...

$\theta^* = (X^T X + \lambda I)^{-1} X^T Y$. Complexity is $O(p^3)$.

If n is smaller than p, we can solve this in $O(n^3)$ instead:

- assume that there exists a solution $\theta^* = X^T z^*$, where $z \in R^n$

  $\nabla f(\theta) = 0 \Leftrightarrow (X^T X - \lambda I) X^T z = X^Y$
  $\Leftrightarrow (X^T X X^T + \lambda X^T) z = X^T Y$
  $\Leftrightarrow (X^T (X X^T + \lambda I) z = X^T Y$
  $\Leftarrow (X X^T + \lambda I) z = Y$
  $\Leftarrow z^* = (X X^T + \lambda I)^{-1} Y$
  This tells us that $\theta^* = X^T (X X^T + \lambda I)^{-1} Y$
  Complexity is now $O(n^3)$

1.2.3 Gradient descent

h is k-Lipschitz continuous if for all x, y, $||h(x) - h(y)|| \leq k||x - y||$
Cauchy-Schwarz: $x^T y \leq ||x||||y||$

Lemma: If f is differentiable and its gradient is L-Lipschitz continuous, then for all $\theta_0, \theta \in R^p$,
$f(\theta) \leq f(\theta_0) + (\nabla f(\theta_0))^T (\theta - \theta_0) + \frac{L}{2}||\theta - \theta_0||_2^2$

First order approximation + quadratic term = $g(\theta)$

Proof of the lemma
$f(\theta) - f(\theta') = \int_{t=0}^{1} \nabla f(t\theta + (1-t)\theta')^T (\theta - \theta') dt$ (*)
$h : [0,1] \to \mathbb{R}$
$t \to f(t\theta + (1-t)\theta')$
$h(1) - h(0) = \int_0^1 h'(t) dt$
$\Leftrightarrow$
(*) $f(\theta) - f(\theta') - f(\theta') \nabla f(\theta')^T (\theta - \theta')$
$\leq \int_{t=0}^{1} \left[\nabla f(t\theta + (1-t)\theta') - \nabla f(\theta')\right]^T (\theta - \theta') dt$
$\leq \int_{t=0}^{1} ||\nabla f(t\theta + (1-t)\theta') - \nabla f(\theta')|| dt ||\theta - \theta'||_2$
$\leq \int_0^1 tL||\theta' - \theta||_2^2 dt = \frac{L}{2}||\theta - \theta'||_2^2$
End of proof.

- Gradient descent algorithm

```
Input $\theta_0 \in \mathbb{R}^p, \eta_t \in R$
for t=1...T
    $\theta_{t+1} \to \theta_t - \eta_t \nabla f(\theta_t)$
end
```

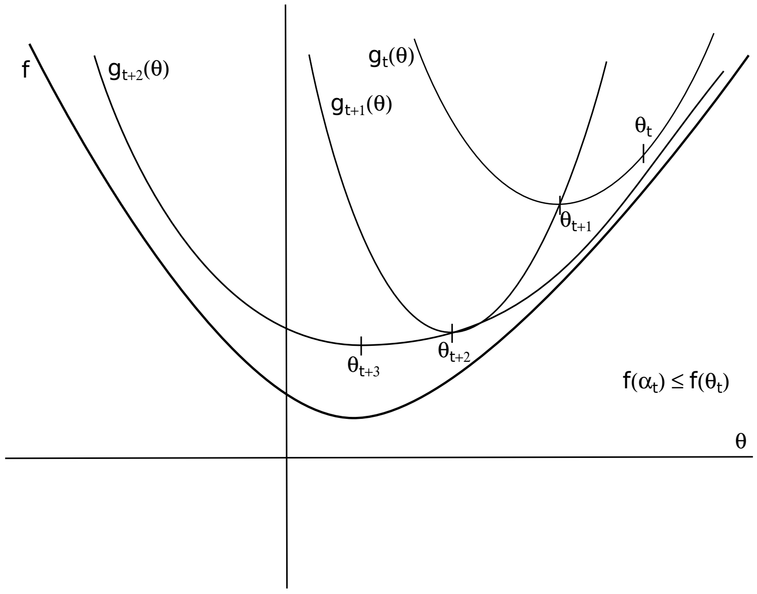
  If $\eta_t = \frac{1}{L}$, then we minimise the $g_t(\theta)$ (quadratic function) at each step.

  $g_t(\theta) = f(\theta_t) + \nabla f(\theta_t)^T (\theta - \theta_t) + L/2||\theta - \theta_t||_2^2$

  then the minimum of g_t is achieved by $\theta_t - \frac{1}{L}\nabla f(\theta_t)$

  $\nabla g_t(\theta) = 0 \Leftrightarrow \nabla f(\theta_t) + L(\theta - \theta_t) = 0$

  $\Leftrightarrow \theta = \theta_t - \frac{1}{L}\nabla f(\theta_t)$



Proposition: if f is convex, differentiable and ∇ f is L-Lipschitz, then gradient descent with $\eta_t = \frac{1}{L}$ gives $f(\theta_t) - f^* \leq \frac{L||\theta^* - \theta_0||_2^2}{2t}$ (**convergence rate**)

Proof of the proposition: $f(\theta) \leq g_t(\theta) = f(\theta_t) + (\nabla f(\theta_t))^T(\theta - \theta_t) + \frac{L}{2}||\theta - \theta_t||_2^2$
We can see that θ_{t+1} minimises $g_t(\theta)$, so we directly have $g_t(\theta) = g_t(\theta_{t+1}) + \frac{L}{2}||\theta - \theta_{t+1}||_2^2$
$f(\theta_{\mathrm{t+1}}) \leq g_t(\theta_{\mathrm{t+1}}) = g_t(\theta^*) - \frac{L}{2}||\theta^* - \theta_{\mathrm{t+1}}||_2^2$
$= f(\theta_t) + \nabla f(\theta_t)^T(\theta^* - \theta_t) + \frac{L}{2}||\theta^* - \theta_t||_2^2 - \frac{L}{2}||\theta^* - \theta_{t+1}||_2^2$
As $f(\theta_t) + \nabla f(\theta_t)^T(\theta^* - \theta_t) \leq f^*$ (convexity)
$f(\theta_{t+1}) - f^* \leq \frac{L}{2}||\theta^* - \theta_t||_2^2 - \frac{L}{2}||\theta^* - \theta_{t+1}||2^2$
$T(f(\theta_t) - f^*) \leq [\sum_{t=0}^{T-1} f(\theta_{t+1})] - Tf^* \leq \frac{L}{2}||\theta^* - \theta_0||_2^2 - \frac{L}{2}||\theta^* - \theta_T||_2^2 \leq \frac{L}{2}||\theta^* - \theta_0||_2^2$
End of proof.

- Extension: optimisation with constraints (projective gradient descent)
  Let C be convex set: we want to compute $min_{\theta \in C} f(\theta)$

  It's actually simple: do the same thing, but project each θ_t on C. That is, replace

  $\theta_{t+1} \leftarrow \theta_t - \frac{1}{L}(\nabla f)(\theta_t)$

  by

  $\theta_{t+1} \leftarrow \Pi_C(\theta_t - \frac{1}{L}(\nabla f)(\theta_t))$

  where Π_C is the euclidian projection on C

- Proximal gradient descent

Let's say you want to optimise a non-smooth function, for instance:

$min_{\theta \in \mathbb{R}^p} f(\theta) + \Omega(\theta)$

where $\Omega(\theta) = ||\theta||_1$: we can't use the gradient on Ω.

Then you need to replace (***) by $\theta_{t+1} = argmin_{\theta \in \mathbb{R}^p} \frac{1}{2}||\theta - (\theta_t - \frac{1}{L}\nabla f(\theta_t)||_2^2 + \frac{1}{L}\Omega(\theta) \Rightarrow$ called proximal gradient descent

This operator is called the **proximal** operator.

1.2.4 Newton method

When f is twice differentiable, we can define the Hessian: $\nabla^2 f(\theta) = [\frac{\partial^2 f}{\partial x \partial y}]$ which is a p times p matrix.

Idea: instead of doing a first-order approximation, we use a quadratic approximation.

Algorithm:

```
Input $\theta_0 \in \mathbb{R}^p$
for t = 1...T
$f(\theta) \simeq f(\theta_t) + \nabla f(\theta_t)^T(\theta - \theta_t) + \frac{1}{2}(\theta - \theta_t)^T \nabla^2 f(\theta_t)(\theta - \theta_t)$
Minimize the quadractic approximation $\theta_{t+1} \leftarrow \theta_t - \nabla^2 f(\theta_t)^{-1} \nabla f(\theta_t)$
endfor
```

Each iteration costs $O(p^3)$, which is a lot. However, the convergence rate is very good:

If the smallest eigenvalue of the Hessian is always greater than $l > 0$, and the Hessian is M-Lipschitz continuous, then there exists a constant C such that $||\theta_{t+1} - \theta^*||_2 \leq C||\theta_t - \theta^*||_2^2$.

For large scale problems, there are variants L-BFGS.

1.2.5 Stochastic Gradient Descent

We are often minimizing a function of the form:

$f_n(\theta) = \frac{1}{n}\sum_{i=1}^n L(y_i, X_i, \theta)$. (empirical risk)

$f(\theta) = \mathbb{E}_{(y,X)}[L(y, X, 0)]$ (expected risk)

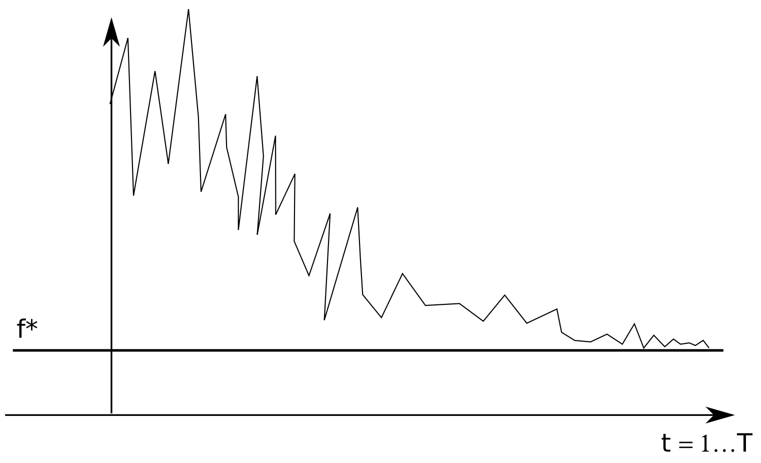
Remark: complexity per iteration of gradient descent for minimizing for is $O(M)$ because $\nabla f_n(\theta) = \frac{1}{n}\sum_{i=1}^n \nabla_\theta L(y_i, X_i, \theta)$

Algorithm

```
for t=1...T
Draw$(y_t, X_t)$
Update $\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla_\theta L(y_t, X_t, \theta_t)$
end for
```

The interpretation of $\eta_t \geq 0$ is the "step-size", and of $\nabla_\theta L(y_t, X_t, \theta_t)$ is a noisy gradient.

Typical behaviour $f_n(\theta)$



Remarks:

- $\mathbb{E}[f(\theta_t)] - f^* = O\left(\frac{1}{\sqrt{t}}\right)$ for convex functions (under technical assumptions)
- this is a " slow" rate but the cost per iteration is cheap
- useful for large-scale machine learning
- in practice, fast to get low-precision solution, and slow for high precision

1.3 Non-parametric approaches

1.3.1 Nearest Neighbor Approach (NN)

Assume that you have training data (Y_i, X_i) with $i = 1 \ldots n$ with $Y_i \in \{1, \ldots, M\}$.

Given some text data $X \in \mathbb{R}^P$, the NN classification rule is $\hat{i} \in argmin_{i=1\ldots n} d(X, X_i)$, $haty_{NN}(x) = y_{\hat{i}}$.

Variant k-NN: $\hat{y}_{k-NN}(x) = Vote(\hat{y}_{\hat{i}_1}, \ldots, \hat{y}_{\hat{i}_k})$.

Remarks:

- there is no "learning" of parameters
- requires "appropriate" distance functions
- simple but $O(n)$ complexity at test time

Figures of **Elements of Statistical Learning** by Hastie et al, page 466 (485 in the PDF): The classification shown on the second figure is too precise, while the first one can account for noisy or wrong training data. $\rightarrow$ There are approximate NN techniques that do some pre-computation. For instance "KD-trees", "local sensitive hashing LSH".

1.3.2 Smoothing techniques for regression

$\hat{f}(x) = \sum_{\text{i}=1}^{n} \frac{\kappa_\theta(X, X_i) Y_i}{\sum_{j=1}^{n} \kappa_\theta(X, Xj)}$

$= \sum_{i=1}^{n} \eta_i Y_i$ $(\sum \eta_i = 1$ and $\eta_i \geq 0)$

K_σ is a weight function, for instance a smoothing kernel. Example: $K_\sigma(X, X_i) = e^{\frac{-1}{2\sigma^2}||X - X_i||_2^2}$

This is called the Nadaraya-Watson estimator. Book by Wasserman "Non-parametric statistics"

1.3.3 Application in image processing: non-local means by Buades et al. 2005

Noisy image; pixel y_i.

Hypothesis: a pixel can be explained by a "patch" X_i around it (10x10 pxels), $\Rightarrow X_i \in \mathbb{R}^{100}$.

Non local means approach consider all the patches (X_i, y_i) as "training set". The denoised pixel i is obtained by $\hat{y}_j = \sum_{i=1}^{n} \frac{\kappa_\theta(X_i,X_j)}{\sum_{k=1}^{n} \kappa_\theta(X_i,X_k)} Y_i$

$Z_i = \mu + \epsilon_i$ with $\mathbb{E}[\epsilon_i] = 0$ and ϵ_i are i.i.d. $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} z_i$

$\hat{\mu} - \mu = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i = O\left(\frac{1}{\sqrt{n}}\right)$

Idea: to denoise a pixel, look for similar patches in the rest of the images (assign high weight to similar patches, low weight otherwise).

Then, take the (weighted) average of the pixels at the center of the patches. This works if the noise has a zero average, and is independent.

1.3.4 Theorem by Cover and Hart, 1967

"Asymptotically, the error rate of 1-NN is never more than twice the Bayes error rate".

The data (y, x) is drawn according to some probability distribution. In particular, we have some conditional probabilities $\mathbb{P}[Y = y|x]$ (with $x \in \mathbb{R}^p$, $y \in 1 \ldots M$)

Bayes classifier: $\hat{y}_{Bayes}(x) = argmax_{c=1...M}\mathbb{P}[Y = c|X = x]$

This is not possible in practice, because we don't know the conditional probabilities.

$min_{f:\mathbb{R}^p \to \{1...M\}}\mathbb{E}_{(X,Y)}[1_{\{f(X)\neq Y\}}]$
$= \mathbb{E}_X\mathbb{E}_{\{Y|X\}}[1_{\{f(X)\neq Y\}}]$
$= \mathbb{E}_X\left[\sum_{\text{i}=1}^{n} \mathbb{P}[Y = c|X]1_{\{f(X)\neq Y\}}\right]$
$= \mathbb{E}_X[1 - \mathbb{P}[Y = f(X)|X]]$

This is minimum for $f(X) = argmax_{i=1...M}\mathbb{P}[Y = c|X] = \hat{y}_{Bayes}(X)$