UE DS52 : Foundations of Data Science
Optimization

Serge Iovleff

UTBM

March 20, 2024

"All learning problems are essentially optimization problems on data"

Christopher G. Atkeson, Professor at CMU

Plan

Motivations

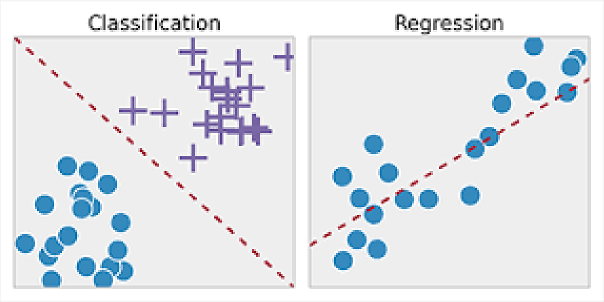
- In a learning process with data, we have
  - A model with a set of parameters $\boldsymbol{\theta}$ to be determined
  - A set of data $\mathcal{D}_n$ to learn from (i.e. a sample)
  - A cost function $R(\boldsymbol{\theta}, \mathcal{D}_n)$ that measures the performance of our model
  - Assumptions about data and parameters which are expressed using equalities and inequalities of the form $f(\boldsymbol{\theta}, \mathcal{D}_n) = 0$ and $g(\boldsymbol{\theta}, \mathcal{D}_n) \geq 0$.
- What is the general approach to solving these problems?

Plan

1. Motivations
    - Supervised Learning and Optimization
    - Unsupervised learning and optimization
2. Introduction to Optimization
3. Fundamentals of unconstrained optimization
    - Convexity
    - Linear Search Methods
    - Step length
4. An example: Logistic Regression
5. Constrained optimization

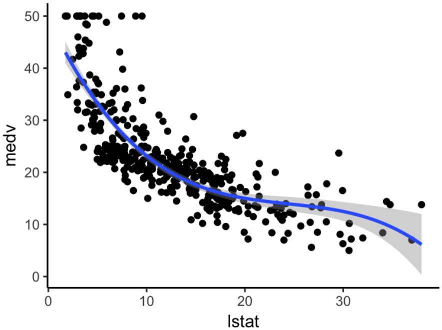
Supervised Learning

- Learning algorithms are used to automate **decision-making** processes by generalizing examples of decision making observed in the past.
- There are two main types of supervised learning problems, known as *classification* and *regression*.



Definition (Regression)

The objective is to predict a continuous number, a real number in mathematical terms (or a floating-point number in programming terms).



More formally, the input is a collection of

$$(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$$

of n observations, where each (X_i, Y_i) belongs to a space $\mathcal{X} \times \mathcal{Y}$ with $\mathcal{Y} \subset \mathbb{R}$.

The goal is to find a function $g : \mathcal{X} \to \mathcal{Y}$ (a "regressor", the formula that predicts the number)

Examples of regression models

- The **Ordinary Least Square** (OLS) model. Mathematically, it solves the problem

$$\min_{\boldsymbol{\beta}} ||\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}||_2^2$$

- The **Lasso** model. It consists of a linear model with an added regularization term. The objective function to minimize is

$$\min_{\boldsymbol{\beta}} \frac{1}{2n} ||\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}||_2^2 + \alpha ||\boldsymbol{\beta}||_1$$

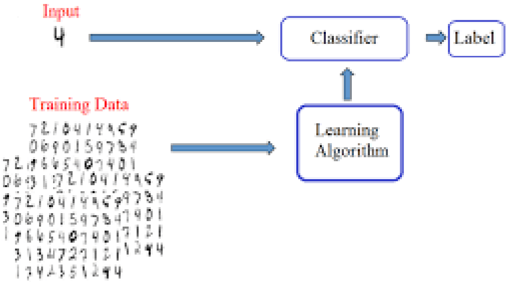
- The **support vector machine** (SVM). Here, we are penalizing samples whose prediction is at least ϵ away from their true target

$$\begin{aligned} &\min_{\boldsymbol{\beta}, b, \zeta, \zeta^*} \frac{1}{2} \boldsymbol{\beta}^T \boldsymbol{\beta} + C \sum_{i=1}^{n} (\zeta_i + \zeta_i^*) \\ &\text{subject to } y_i - \boldsymbol{\beta}^T \phi(\mathbf{x}_i) - b \leq \varepsilon + \zeta_i, \\ &\boldsymbol{\beta}^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ &\zeta_i, \zeta_i^* \geq 0, i = 1, \ldots, n \end{aligned} \tag{1}$$

with $\phi(x)$ a **kernel** function.

Definition (Supervised classification)

The objective is to predict a class label, which is a choice from a predefined list of possibilities.



Formally, we have a collection as input

$$\mathcal{D}_n = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}$$

of n observations where each (X_i, Y_i) belongs to a space $\mathcal{X} \times \mathcal{Y}$ with $\#\mathcal{Y} = K$.

The aim is to find a function $g : \mathcal{X} \to \mathcal{Y}$ (a "classifier", the method that will predict the label)

Examples of classification models

- **Logistic Regression**. The target y_i takes values in the set $\{0, 1\}$

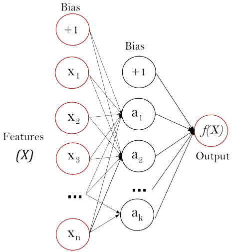
$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(-y_i \log(\mathbf{X}_i^T \boldsymbol{\beta}) - (1 - y_i) \log(1 - \mathbf{X}_i^T \boldsymbol{\beta}) \right),$$

- **Support vector Machine** (SVM). The goal is to find $\boldsymbol{\beta} \in \mathbb{R}^p$ and $\beta_0 \in \mathbb{R}$ such that the prediction given by $\text{sign}(\boldsymbol{\beta}^T \phi(\mathbf{x}) + \beta_0)$ is correct for most samples.

$$\begin{aligned} \min_{\boldsymbol{\beta}, \beta_0, \zeta} \frac{1}{2} \boldsymbol{\beta}^T \boldsymbol{\beta} + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to } y_i(\boldsymbol{\beta}^T \phi(\mathbf{x}_i) + \beta_0) \geq 1 - \zeta_i, \\ \zeta_i \geq 0, i = 1, ..., n \end{aligned} \tag{2}$$

with ϕ a kernel function.

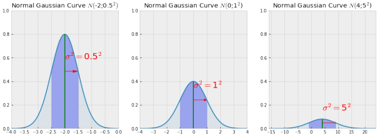
- **Neural Networks** (NN)



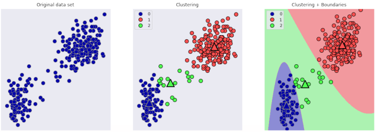
Plan

In unsupervised learning, we consider inputs $\mathcal{D}_n = (\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n)$, but we have neither the target outputs Y_i, nor the rewards of the environment.
There are three main categories of unsupervised learning

- **Density Estimation** (mostly using ML)

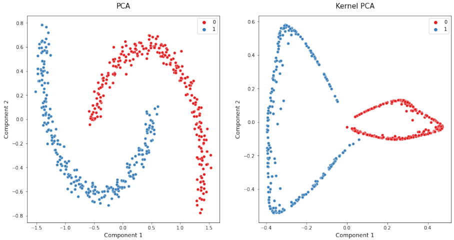


- The most common density estimation involves **clustering** algorithms, which divide the dataset into K distinct groups of similar elements.

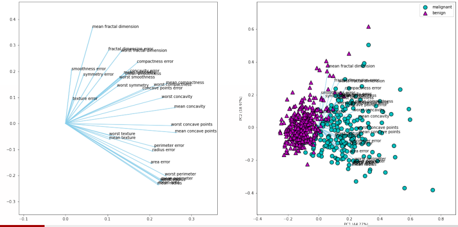


$$\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k f(\mathbf{x}_i; \boldsymbol{\theta}_k) \right)$$

- **Data transformations**. These are algorithms that create a new representation of the data easier to understand for humans or/and other machine-learning algorithms compared with the original original data representation.



- One of the most common transformations in unsupervised unsupervised learning is **dimensionality reduction**. The best-known technique is **PCA**.



Plan

Presentation

- A general optimization problem look like this:

$$\min_{\mathbf{x}\in\mathbb{R}^p} f(\mathbf{x}) \quad \text{s.t.} \begin{cases} c_i(x) = 0, i \in \mathcal{E} & \text{equality constraints (scalar)} \\ c_i(x) \le 0, i \in \mathcal{I} & \text{inequality constraints (scalar)} \end{cases} \tag{3}$$

with $\mathbf{x}$: vector; $f(\mathbf{x})$: objective function (scalar). Note that $\max\ f = -\min\ -f$.

- Example

$$\min_{x_1,x_2} (x_1-2)^2 + (x_2-1)^2 \quad \text{s.t.} \begin{cases} x_1^2 - x_2 \le 0 \\ x_1 + x_2 - 2 \le 0 \end{cases} \tag{4}$$

- Simplex problem

$$\max_{x,y} x + 3y \quad \text{s.t.} \begin{cases} 2y \le 25 - x \\ 4y \ge 2x - 8 \\ y \le 2x - 5 \\ x, y \ge 2 \end{cases} \tag{5}$$

Important remarks

- Optimization algorithms are **iterative**: build sequence of points that converges to the solution. Needs good initial point (often by prior knowledge).
- Desiderata for algorithms:
  - **Robustness**: perform well on wide variety of problems in their class, for any starting point;
  - **Efficiency**: little computer time or storage;
  - **Accuracy**: identify solution precisely (within the limits of fixed-point arithmetic).

  They conflict with each other
- General comment about optimization (Fletcher): “fascinating blend of theory and computation, heuristics and rigour”.
  - No universal algorithm: a given algorithm works well with a given class of problems.
  - Necessary to adapt a method to the problem at hand (by experimenting).
  - Not choosing an appropriate algorithm $\Rightarrow$ solution found very slowly or not at all.
- **Course contents**: derivative-based methods for continuous optimization

Plan

Plan

Problem: $\min f(\mathbf{x}),\ \mathbf{x} \in \mathbb{R}^p$.

- Global minimizer: $f(\mathbf{x}^\star) \leq f(\mathbf{x}),\ \forall \mathbf{x}\ \mathbb{R}^p$.
- Local minimizer: $\exists$ Neighborhood $\mathcal{N}$ of $\mathbf{x}^\star$: $f(\mathbf{x}^\star) \leq f(\mathbf{x}),\ \forall \mathbf{x} \in \mathcal{N}$.
- Strict (or strong) local minimizer: $f(\mathbf{x}^\star) < f(\mathbf{x}),\ \forall \mathbf{x} \in \mathcal{N} \backslash \{\mathbf{x}^\star\}$.
- First-order **necessary** conditions : $\mathbf{x}^\star$ is a local minimizer, if f continuously differentiable in an open neighborhood of $\mathbf{x}^\star \rightarrow \nabla f(\mathbf{x}^\star) = 0$. Not a sufficient condition, ex: $f(x) = x^3$
- Second-order **necessary** conditions: $\mathbf{x}^\star$ is a local minimizer if f is twice continuously differentiable in an open neighborhood of $\mathbf{x}^\star \Rightarrow \nabla f(\mathbf{x}^\star) = 0$ and $\nabla^2 f(\mathbf{x}^\star)$ is positive semi-definite. Not sufficient condition, ex: $f(x) = x^3$.
  Proof by contradiction : if $\nabla^2 f(\mathbf{x}^\star)$ is not positive semi-definite then f decreases along the direction where ∇^2 is not positive semi-definite.
- Second-order **sufficient** condition: $\nabla^2 f$ continuous in an open neighborhood of $\mathbf{x}^\star$, $\nabla f(\mathbf{x}^\star) = 0$, $\nabla^2 f(\mathbf{x}^\star)$ positive definite $\Rightarrow \mathbf{x}^\star$ is a strict local minimizer of f. (Not necessary condition, ex: $f(x) = x^4$ at $\mathbf{x}^\star = 0$.)
  Proof : Taylor-expand f around $\mathbf{x}^\star$.

Convex functions

Definition (Convex sets and functions)

A set $\mathcal{D} \subset \mathbb{R}^p$ is convex if for any pair $(\mathbf{x}_1, \mathbf{x}_2) \in \mathcal{D}^2$ and for all $\alpha \in [0, 1]$, we have

$$\alpha \mathbf{x}_1 + (1-\alpha)\mathbf{x}_2 \in \mathcal{D}$$

A function $f : \mathbb{R}^p \to \mathbb{R}$ is **convex** if

- Its domain is convex
- $f(\mathbf{x}) = f(\alpha \mathbf{x}_1 + (1-\alpha)\mathbf{x}_2) \leq \alpha f(\mathbf{x}_1) + (1-\alpha) f(\mathbf{x}_2)$

Definition (Positive semi-definite matrices)

A matrix H of dimension (p, p) is positive semi-definite if

$$\mathbf{z}^T H \mathbf{z} \geq 0, \qquad \forall \mathbf{z} \in \mathbb{R}^p$$

There is a link between convex functions and their second derivative matrices

Theorem (Convexity criterion)

A function $f : \mathbb{R}^p \to \mathbb{R}$ of class $\mathcal{C}^2$ is convex if its Hessian matrix $H(\mathbf{x}) = \nabla^2 f(\mathbf{x})$ is positive semi-definite at any point of the domain.

Examples of convex functions

- Linear/affine functions
$$x \to \mathbf{x}^T\mathbf{b} + c,\ \mathbf{b} \in \mathbb{R}^p,\ c \in \mathbb{R}$$
- Quadratic functions
$$x \to \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c,\ \mathbf{b} \in \mathbb{R}^p,\ c \in \mathbb{R}$$
with $\mathbf{A}$ semi-definite positive.
- The entropy function
$$x \in \mathbb{R}^+,\ x \to -x\ln(x)$$
- The d norm defined for $\mathbf{x} \in \mathbb{R}^p$ by $\|\mathbf{x}\|_p = \left(\sum_{i=1}^p x_i^p\right)^{1/p}$
- Log-sum-exp (aka softmax, a smooth approximation to the maximum function often used on machine learning)
$$f(\mathbf{x}) = \log\left(\sum_{i=1}^n \exp(x_i)\right)$$

Why convex functions?

For practical reasons :

- Many machine learning problems arise from the minimization of a convex criterion and provide significant results for the initial statistical task.
- Many optimization problems admit a convex reformulation (SVM classification or regression, LASSO regression, ridge regression, etc.).

For reasons inherent to the algorithms :

- local minimum = global minimum. This is important because in general, descent methods use $\nabla f(\mathbf{x})$ (or something similar), which is local information about f.
- There are many fast algorithms for optimizing convex of convex functions, some of which are independent of the dimension of the original problem.

Plan

Iteration: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ where α_k is the **step length** (how far to move along $\mathbf{p}_k$), $\alpha_k > 0$; $\mathbf{p}_k$ is the s**earch direction**.

- The **linear search** is one of the two classic approaches classical approaches to forcing the convergence algorithms for calculating a minimum (the other being the **confidence regions** [not seen in this talk])
- $\mathbf{p}_k$ *descent direction* at $\mathbf{x}_k \Rightarrow \mathbf{p}_k^T \nabla f(\mathbf{x}_k) < 0$ guarantees that f can be reduced along $\mathbf{p}_k$ (for a sufficiently small step):
  Proof:

$$
\begin{aligned}
f(\mathbf{x}_k + \alpha \mathbf{p}_k) = & \quad f(\mathbf{x}_k) + \alpha \mathbf{p}_k^T \nabla f(\mathbf{x}_k) + \mathcal{O}(\alpha^2) && \text{(Taylor's theorem)} \\
< & \quad f(\mathbf{x}_k) && \text{for all sufficiently small } \alpha > 0
\end{aligned}
$$

Gradient descent direction

The direction along which f decreases most rapidly, is $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$.

- First-order optimization methods
  - $\Rightarrow$ We approach f by its linearized form $f(\mathbf{x}_k + \alpha \mathbf{p}_k) \approx f(\mathbf{x}_k) + \alpha \mathbf{p}^T \nabla f(\mathbf{x}_k)$
  - $\Rightarrow$ $\nabla f(\mathbf{x}_k)$ is orthogonal to the contour line $f(\mathbf{x}) = c$ with $c = f(\mathbf{x}_k)$.
- The function f decreases most strongly in the direction opposed to that of the gradient of f, i.e.

$$\min_{\mathbf{p}} \mathbf{p}^T \nabla f(\mathbf{x}) \quad s.t. \; \|\mathbf{p}\| = 1 \quad \Rightarrow \quad \mathbf{p} = -\frac{\nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|}$$

Algorithm 1 Steepest descent algorithm

Choose a starting point $\mathbf{x}_0 \in \mathbf{dom} f$
$k = 0$
repeat
    $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ ▷ Calculating the gradient at point $\mathbf{x}_k$
    $\alpha_k \approx \arg\min_\alpha f(\mathbf{x}_k + \alpha \mathbf{p})$ ▷ Optimization of α
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ ▷ Update
    $k = k + 1$
until stop if stopping criterion is met ($\|\nabla f(\mathbf{x}_k)\| < \varepsilon$)

Newton direction

The Newton direction is $\mathbf{p}_k = -H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$.

- This corresponds to assuming f is locally quadratic and jumping directly to its minimum. By Taylor's theorem

$$f(\mathbf{x}_k + \mathbf{p}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{p} + \frac{1}{2}\mathbf{p}^T \nabla^2 f(\mathbf{x}_k)\mathbf{p} = Q_k(\mathbf{p})$$

- Minimize a quadratic function

$$\mathbf{p} = \underset{\mathbf{p}}{\operatorname{argmin}}\, Q(\mathbf{p}) = \underset{\mathbf{p}}{\operatorname{argmin}} \left[c + \mathbf{g}^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p} \right]$$

- The solution to this problem is given explicitly by (take derivative with respect to $\mathbf{p}$)

$$\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}$$

- Note: Taking $\mathbf{H} = \mathbf{I}$, we get the Steepest descent algorithm.
- What's wrong with this method?

  $\Rightarrow$ We need to calculate the inverse of the matrix Hessian matrix at each iteration , this can be very time-consuming in terms of computation time, or even be infeasible

Quasi-Newton direction: BFGS

For most algorithms, $\mathbf{p}_k = -B_k^{-1}\nabla f(\mathbf{x}_k)$ where B_k is symmetric non-singular:

- Approximate the Hessian matrix using the following ideas:
  - Hessians change slowly
  - Derivatives interpolate
- At each step, compute finite differences $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$
- Update $\mathbf{B}_{k+1} \approx \mathbf{H}_{k+1}^{-1}$ using formula

$$\mathbf{B}_{k+1} = (\mathbf{I}_p - \rho_k \mathbf{s}_k \mathbf{y}_k^T)\mathbf{B}_k(\mathbf{I}_p - \rho_k \mathbf{s}_k \mathbf{y}_k^T) + \rho_k \mathbf{s}_k \mathbf{s}_k^T \tag{6}$$

with $\rho_k = \mathbf{y}_k^T \mathbf{s}_k$.

Algorithm 2 BFGS algorithm

1: **Choose** a starting point $\mathbf{x}_0 \in \mathbf{dom} f$ and inverse Hessian approximation $\mathbf{B}_0$ (ex. $\mathbf{B}_0 = \mathbf{I}_p$)
2: $k = 0$
3: **while** $\|\nabla f(\mathbf{x}_k)\| > \varepsilon$ **do**
4: Compute: $\mathbf{p}_k = -\mathbf{B}_k \nabla f(\mathbf{x}_k)$
5: Compute: $\alpha_k \approx \arg\min_\alpha f(\mathbf{x}_k + \alpha \mathbf{p}_k)$
6: Update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
7: Compute: $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$
8: Update: $\mathbf{B}_{k+1}$ using equation (6).
9: $k = k + 1$
10: **end while**

Plan

Step length

Here, we deal with how to choose the step length given the search direction $\mathbf{p}_k$.

- Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function whose minimum is being sought. At iteration k, the algorithm has found a $\mathbf{x}_k$ position and calculated a direction of descent $\mathbf{p}_k$. The (one-dimensional) problem to be solved is

$$\min_{\alpha} \phi_k(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$$

- We can try to minimize ϕ exactly by solving $\phi_k(\alpha) = 0$ (no closed form)
  - Each evaluation of ϕ requires an evaluation of f (expensive?)
  - If we don't care about the number of computation of f, we can use any uni-dimensional minimization method:
    - Root finding methods if ϕ' is available (expensive ?): Secant, Steffenson's, Brent's methods, Newton (if ϕ'' is available), etc.
    - Golden search, Powell's quadratic, etc.
- If the computation of f is expensive, we're rather looking for a sufficient decrease of ϕ

Step length: The backtracking algorithm

- A popular inexact line search condition stipulates that α_k should first of all give sufficient decrease in the objective function f, as measured by the following inequality (*Armijo condition*):

$$f(\mathbf{x}_k + \alpha \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha \nabla f(\mathbf{x}_k)^T \mathbf{p}_k$$

  for some constant c_1.

- The sufficient decrease condition is not enough by itself to ensure that the algorithm makes reasonable progress. To rule out unacceptably short steps we introduce a second requirement, called the curvature condition

$$\nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^T \mathbf{p}_k \geq c_2 \alpha \nabla f(\mathbf{x}_k)^T \mathbf{p}_k$$

- The sufficient decrease and curvature conditions are known collectively as the **Wolfe conditions**.

Algorithm 3 backtracking algorithm

1: **Choose** $\alpha > 0$ ($\alpha = 1$ for Newton methods), $\rho \in (0, 1)$, $c \in (0, 1)$
2: **while** ($f(\mathbf{x}_k + \alpha \mathbf{p}_k) \leq f(\mathbf{x}_k) + c\alpha \nabla f(\mathbf{x}_k)^T \mathbf{p}_k$) **do**
3: $\quad \alpha \leftarrow \rho \alpha$
4: **end while**

Plan

The logistic model

The objective is to predict a binary outcome using predictors.

- We want to predict $y \in \{0, 1\}$ using predictor $\mathbf{x}$
- Logistic Regression is a discriminative model, because it models the posterior probabilities $\mathbb{P}(y|\mathbf{x})$ directly.
- The model is

$$\begin{aligned} \mathbb{P}(y = 1|\mathbf{x}) &= h(\mathbf{x}^T \boldsymbol{\beta}) \\ \mathbb{P}(y = 0|\mathbf{x}) &= 1 - h(\mathbf{x}^T \boldsymbol{\beta}) \end{aligned}$$

  with $h : \mathbb{R} \to (0, 1)$ an **inverse link** function [more about this in another session].
- We have a Bernoulli likelihood with independent observations. So given a sample, $(y_i, \mathbf{x}_i)_{i=1}^n$, the likelihood is given by

$$l_n(\boldsymbol{\beta}) = \sum_{i=1}^{n} y_i \log h(\mathbf{x}_i^T \boldsymbol{\beta}) + (1 - y_i) \log(1 - h(\mathbf{x}_i^T \boldsymbol{\beta})).$$

First derivatives

Compute ∇l_n

- Taking the first derivative with respect to β_j, we get

$$
\begin{aligned}
\frac{\partial l_n}{\partial \beta_j} &= \sum_{i=1}^{n} \frac{y_i}{h(\mathbf{x}_i^T\boldsymbol{\beta})} h'(\mathbf{x}_i^T\boldsymbol{\beta})x_{ij} - \frac{1-y_i}{1-h(\mathbf{x}_i^T b)} h'(\mathbf{x}_i^T\boldsymbol{\beta})x_{ij} \\
&= \sum_{i=1}^{n} x_{ij} h'(\mathbf{x}_i^T\boldsymbol{\beta}) \left(\frac{y_i}{h(\mathbf{x}_i^T\boldsymbol{\beta})} - \frac{1-y_i}{1-h(\mathbf{x}_i^T\boldsymbol{\beta})} \right) \\
&= \sum_{i=1}^{n} x_{ij} \frac{h'(\mathbf{x}_i^T\boldsymbol{\beta})}{h(\mathbf{x}_i^T\boldsymbol{\beta})(1-h(\mathbf{x}_i^T\boldsymbol{\beta}))} (y_i - h(\mathbf{x}_i^T\boldsymbol{\beta})).
\end{aligned}
$$

- Now let's suppose we're using the canonical link function $g = \text{logit}$. Then $h(x) = \frac{1}{1+e^{-x}}$, so $h' = h(1-h)$ [Proove it !] which means this simplifies to

$$
\frac{\partial l_n}{\partial \beta_j} = \sum_{i=1}^{n} x_{ij}(y_i - h(\mathbf{x}_i^T\boldsymbol{\beta}))
$$

- So

$$
\nabla l_n(\boldsymbol{\beta}) = \mathbf{X}^T(\mathbf{y} - \hat{\mathbf{y}}).
$$

Second derivatives

Compute $\nabla^2 l_n$.

- Furthermore, still using h

$$\frac{\partial^2 l_n}{\partial \beta_k \partial \beta_j} = -\sum_{i=1}^{n} x_{ij} \frac{\partial}{\partial \beta_k} h(\mathbf{x}_i^T \boldsymbol{\beta}) = -\sum_{i} x_{ij} x_{ik} \left[h(\mathbf{x}_i^T \boldsymbol{\beta})(1 - h(\mathbf{x}_i^T \boldsymbol{\beta})) \right].$$

- Let

$$\begin{aligned} W &= \operatorname{diag}\left(h(\mathbf{x}_1^T \boldsymbol{\beta})(1 - h(\mathbf{x}_1^T \boldsymbol{\beta})), \ldots, h(\mathbf{x}_n^T \boldsymbol{\beta})(1 - h(\mathbf{x}_n^T \boldsymbol{\beta})) \right) \\ &= \operatorname{diag}\left(\hat{y}_1(1 - \hat{y}_1), \ldots, \hat{y}_n(1 - \hat{y}_n) \right). \end{aligned}$$

- Then we have

$$\nabla^2 l_n = -\mathbf{X}^T W \mathbf{X}.$$

- As $\hat{y}_i \in (0, 1)$, $-\mathbf{X}^T W \mathbf{X}$ will always be strictly negative definite, although numerically if $\hat{y}_i$ gets too close to 0 or 1 then we may have weights round to 0 which can make H negative semidefinite and therefore computationally singular .

Newton-Raphson algorithm

Use iterative weighted least squares regression .

- Create the working response $\mathbf{z} = W^{-1}(\mathbf{y} - \hat{\mathbf{y}})$ and note that
$$\nabla l_n = \mathbf{X}^T(y - \hat{y}) = \mathbf{X}^T W \mathbf{z}.$$
- All together this means that we can optimize the log likelihood by iterating
$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + (\mathbf{X}^T W^{(k)} \mathbf{X})^{-1} \mathbf{X}^T W^{(k)} \mathbf{z}^{(k)}$$
- Remark: $(\mathbf{X}^T W^{(k)} \mathbf{X})^{-1} \mathbf{X}^T W^{(k)} \mathbf{z}^{(k)}$ is exactly $\hat{\boldsymbol{\beta}}$ for a weighted least squares regression of $\mathbf{z}^{(k)}$ on $\mathbf{X}$.

Plan

Formalisation

- Constraints on the solution can be added in the form of equalities or inequalities of the form
$$\mathbf{g}(\mathbf{x}) = \mathbf{0}; \qquad \mathbf{h}(\mathbf{x}) \geq 0$$
- Example: how to solve the problem?
$$\begin{aligned} \min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) &= \theta_1^2 + \theta_2^2 - 1 \\ \text{s.t. } g(\boldsymbol{\theta}) &: \theta_1 + \theta_2 - 1 = 0 \end{aligned}$$
- We can transform it into an unconstrained problem
- We can use the Lagrange multiplier method.
- We write the Lagrangian
$$\mathcal{L}(\mathbf{x}, \lambda) = 1 - \theta_1^2 - \theta_2^2 + \lambda(\theta_1 + \theta_2 - 1)$$

$\Rightarrow$ We're maximizing a function of three variables. If the problem has a solution, the constraint is verified because
$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \Leftrightarrow \theta_1 + \theta_2 - 1 = 0$$

- What is the solution?

General Formulation

- For a problem written in the form

$$\begin{array}{lll} \min_{\boldsymbol{\theta}} & f(\boldsymbol{\theta}) & \\ \text{s.t.} & \mathbf{g}(\boldsymbol{\theta}) & = \mathbf{0} \\ & \mathbf{h}(\boldsymbol{\theta}) & \geq \mathbf{0} \end{array}$$

- We have the Lagrangian

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{\theta}) + \boldsymbol{\lambda}^T \mathbf{g}(\boldsymbol{\theta}) + \boldsymbol{\mu}^T \mathbf{h}(\boldsymbol{\theta})$$

- In Machine Learning, the problem is ideal if

$$\begin{array}{lll} \min_{\boldsymbol{\theta}} & f(\boldsymbol{\theta}) & \text{is a convex function} \\ \text{s.c.} & \mathbf{g}(\boldsymbol{\theta}) = \mathbf{0} & \text{is linear} \\ & \mathbf{h}(\boldsymbol{\theta}) \geq \mathbf{0} & \text{is a convex set} \end{array}$$

To be continued...

© LizzyChrome

Remember: Each time we fit a learning algorithm on a training dataset, we are solving an optimization problem

No magic