Computational Mathematics for Learning and Data Analysis

Federico Matteoni

A.A. 2021/22

Index

0.1 Introduction

Exam: project (groups of 2) + oral exam.
This course's goals is to make sense of the huge amounts of data, take something big and unwieldy and produce something small that can be used, a **mathematical model**.
The mathematical model should be accurate, computationally inexpensive and general, but generally is not possible to have all three. General models are convenient (work once, apply many), they are parametric so we need to learn the right values of the parameters. Fitting is finding the model that better represents the phenomenon given a family of possible models (usually, infinitely many). Is an optimization model and usually is the computational bottleneck.
ML is better than fitting because fitting reduces the training error, the empirical risk, but ML reduces the test error, so the generalization error.
Solve general problem $min_{x \in S} f(x)$, with Poloni solve $min_{x \in R^n} ||Ax - b||_2$ which is easier and can be solved exactly.

Quick recap of linear algebra

Matrix - Vector multiplication, with $A \in R^{4\times 3}, c \in R^3, b \in R^4$

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ \vdots & \vdots & \vdots \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \qquad \begin{array}{l} b_i = \sum_{j=1}^{4} A_{ij} c_j \\ A_{11}c_1 + A_{12}c_2 + A_{13}c_3 = b_1 \end{array}$$

or linear combination of the columns

$$\begin{bmatrix} A_{11} \\ A_{21} \\ A_{31} \\ A_{41} \end{bmatrix} c_1 + \begin{bmatrix} A_{12} \\ A_{22} \\ A_{32} \\ A_{42} \end{bmatrix} c_2 + \begin{bmatrix} A_{13} \\ A_{23} \\ A_{33} \\ A_{43} \end{bmatrix} c_3 + \begin{bmatrix} A_{14} \\ A_{24} \\ A_{34} \\ A_{44} \end{bmatrix} c_4 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

with c_1, c_2, c_3 and c_4 called coordinates.

Basis: tuple of vectors $v_1, v_2, \ldots, v_n \mid$ you can write all vectors b in a certain space as a linear combination $v_1\alpha_1 + v_2\alpha_2 + \ldots + v_n\alpha_n$ with **unique** $a_1, \ldots, a_n$. The canonical basis is

$$c_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad c_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad c_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad c_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

and, for example

$$\begin{bmatrix} 3 \\ 5 \\ 7 \\ 9 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \cdot 3 + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot 5 + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot 7 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot 9$$

Image ImA = set of vectors b that we can reach with A

Kernel $KerA$ = set of vectors $x \mid Ax = 0$ ($x = 0$ is certainly one, there may be others)

Invertible A if this problem has exactly one solution.
$\forall\, b \in R^m$, A must be square and the columns of A are a basis of $R^m \Rightarrow x = A^{-1}b$ where A^{-1} is another square matrix $\mid A \cdot A^{-1} = A^{-1} \cdot A = I$ identity matrix (1 on the diagonal, 0 otherwise)
Implementation detail: `inv(A) * b` is not the best choice. Better: in Python `scipy.linalg.solv(A, b)` or, in Matlab, `A b`.

Cost, with $A \in R^{m\times n}, B \in R^{n\times p}, C \in R^{m\times p}$ (vectors $\Leftrightarrow n \times 1$ matrices), then the cost of multiplication is $mp(2n-1)$ floating point ops (*flops*), or $O(mnp)$.
In particular, A, B squared $\Rightarrow AB$ costs $O(m^3)$. With A, v vector $\Rightarrow Av$ costs $O(m^2)$. Faster alternatives are not worth it usually. And remember that $AB \neq BA$ generally, and also that $CA = CB \not\Rightarrow A = B$ with C matrix.
If there's $M \mid MC = I$, then $A = (MC)A = (MC)B = B$ (multiplying *on the left* by M on both sides)

Why a real valued function? Strong assumption, given x' and x'', I can always tell which one I like best (**total order** of R). Often more than one objective function, with contrasting and/or incomparable units (ex: loss function vs regularity in ML).
But R^k with $k > 1$ has no total order $\Rightarrow$ no *best* solution, only non-dominated ones.
Two practical solutions: maximize return with budget on maximum risk or maximize...
Even with a single objective function optimization is hard, impossible if f has no minimum in X (so, the problem P is unbounded below. Hardly ever happens in ML, because loss and regularization are ≥ 0
Also impossible if $f > -\infty$ but $\nexists x$, for example in $f(x) = e^x$. However plenty of ϵ-approximate solutions (ϵ-optima).
On PC $x \in R$ is in fact $x \in Q$ with up to 16 digits precision, so approximation errors are unavoidable anyway. Exact algebraic computation is possible but usually slow, and ML is going the opposite way (less precision: floats, half, small integer weights…).
Anyway finding the exact x_* is impossible in general.

Optimization need to be approximate Absolute gap, relative gap…but in general computing the gap is hard because we don't know f_*, which is what we want to estimate. So it's hard to estimate how good a solution is. Could argue that this is the "issue" in optimization: compute an estimate of f_*.

Optimization at least possible The f's spikes can't be arbitrarily narrow, so f cannot change too fast
f Lipsichitz continuous (L-c) on X: $\exists\, L > 0 \mid |f(x) - f(y)| \leq L|x - y| \quad \forall x, y \in X$
f L-c $\Rightarrow$ doesn't "jump" and one ϵ-optimum can be found with $O(\frac{LD}{\epsilon})$ evaluations by uniformly sampling X with step $\frac{2\epsilon}{L}$
Bad news: no algorithm can work in less that $\Omega(\frac{LD}{\epsilon})$, but it's the worst case of f (constant with one spike).
Number of steps is inversely proportional to accuracy: just not doable for small ϵ. Dramatically worse with $X \subset R^n$.
Also generally L is unknown and not easy to estimate, but algorithms actually require/use it.

Local Optimization Even if I stumble in x_* how do I recognize it? This is the difficult thing. Simpler to start with a weaker condition: x_* is the local minimum if it solves $min\{f(x) \mid f \in X(x_*, \epsilon) = [x_* - \epsilon, x_* + \epsilon]\}$ for some $\epsilon > 0$.
Stronger notion: **strict** local minimum if $f(x_*) < f(y)$
f (strictly) unimodal on X if has minimum $x_* \in X$ and it is (strictly) decreasing on the left $[x_-, x_*]$ and (strictly) increasing on the right $[x_*, x_+]$
Most functions are not unimodal, but they are if you focus on the attraction basin of x_* and restrict there. Unfortunately it's true for every local optimum, they all look the same.
Once in the attraction basin, we can restrict it by evaluating f in two points and excluding a part. How to choose the part so that the algorithm go as fast as possible? Each iteration dumps the left or the right part, don't know which $\Rightarrow$ should be equal $\Rightarrow$ select $r \in (\frac{1}{2}, 1), x'_- = x'_- + (1-r)D, x'_+ = x_- + rD$
Faster if r larger $\Rightarrow r = \frac{D}{2} + \epsilon = x'_\pm = x_- + \frac{D}{2} \pm \epsilon$ but next iteration will have two entirely different x'_-, x'_+ to evaluate f on.

Optimally choosing the iterates