SMM - Section 5 - Linear Systems

Simone Montali - monta.li

29 settembre 2020

1 Linear systems

1.1 Introduction to linear systems

Now, using the matrix decompositions, we are able to solve linear systems. A linear system can be written as:

$$Ax = b$$

where A is a mtrix of size $m \times n$, with $m \geq n$ and x is a column vector of length n and b is a column vecotr of length m. This form can be expanded as a list of equations. We are interested in the existence/uniqueness of a solution, the numerical methods to find it (there can be more methods) and the conditioning of the problem. We are looking for two features: precision of the solution (related to an estimate of error, the numerical solution always has an error, therefore different algorithms produce different errors), and time of solution (an algorithm is efficient if it requires the lesser possible time).
We can now divide the solution in two categories: square and non-square systems.

1.2 Square linear systems

We now have a linear system

$$Ax = b$$

with A of dimension $n \times n$, x and b of dimension n exists and is unique iff one of the following conditions holds:

- A is non-singular
- $\text{rank}(A) = n$
- The system $Ax = 0$ only admits the solution $x = 0$

In principle, we can check that the solution is computable by dividing b by A, thus multiplying $A^{-1}b$. The computation of the inverse of A is, sadly, computationally hard. So other solutions are needed. One of these is the *Cramer's rule*, where each component x_i of the solution is the ratio between $\frac{detA_i}{detA}$. This method is very costly too, therefore it is not widely used. The numerical methods follow two different approaches: the **direct methods** (which are more precise but computationally costly) and the **iterative methods** (which require a thoretically infinite number of steps, so they are less precise but faster).

1.2.1 Direct methods

These methods are based on the factorization of the matrix, like the LU factorization (computational cost $O\left(\frac{n^3}{3}\right)$). The solution is therefore computable by solving two triangular systems: $Ly = b$ and $Ux = y$. Why is this kind of complicated method used? Because the solution of a triangular system is much less expensive, and it is solvable by forward or backward substitution (which have computational cost of $O\left(\frac{n^2}{2}\right)$). In the case of the pivoting algorithm, we multiply both terms in $Ax = b$ by P, thus obtaining $PAx = Pb$ and then $Ly = Pb, Ux = y$.

1.2.2 Iterative methods

The basic idea, here, is constructing a sequence of vectors that converge to the exact solution.

$$x^* = lim_{k\to\infty}x_k$$

The convergence means that each component converges to a component of the exact solution. We've got to have a starting guess x_0. The sequence of vector is obtained by applying the same function (or set of operators) iteratively. One might ask: *when can I stop the iteration?* This is the most critical part. **Stopping criteria** are the conditions that allow the machine to stop, and they're based on some quantity related to the residual, defined by $r_k = b - AX_k$ at iteration k. We can even base on the absolute criterion $||x_{k+1} - x_k|| \leq \tau$.

1.2.3 Sparse matrices

A sparse matrix is a matrix with a very low percentage of non-null elements, i.e. the majority of items are equal to 0. These are important for data representation, since only the non-null elements are stored, saving a lot of memory.

1.3 Inherent errors in linear systems

Always remember that inherent errors are due to errors in the data representation and do not depend on the algorithm. Even if we start with a really small representation error, we can have large output errors. We distinguish between well posed (where $|\Delta y| \approx |\Delta x|$) and ill posed (where $|\Delta y| >> |\Delta x|$) problems. In the second case, even with small representation errors, there's a tangible risk of great errors. Considering a linear system $Ax = b$, when b slightly change we get $A(x + \Delta x) = b + \Delta b$. If now we wanted to compare Δx with Δb, specifically their relative values $||\frac{\Delta x}{x}||$ and $||\frac{\Delta b}{b}||$.

1.4 Linear least squares

Considering an overdetermined system, finding the solution is impossible (b does not lie on the subspace spanned by the columns of A), therefore we're looking for approximate solutions, specifically the best one. We then need some tools.

1.4.1 Orthogonality and projections

Since, when working in machine learning, visualizing data in a useful way can be pretty difficult, projections come in help. These compress the data while minimizing the loss of useful informations, therefore making data more *densely informative*.

2 Optimization

2.1 Descent methods

This is an iterative method for the solution of the minimization problem. These are the most widely used methods, that given the starting vector x_0, we compute the sequence of vectors with this formula:

$$x_{k+1} = x_k + \alpha_k p_k$$

where p_k is a descent direction, and α is a step size. What happens is we add a step towards the direction.

2.2 Gradient descent

The gradient descent is a first order optimization method where the descent direction is set as

$$p_k = -\nabla f(x_k)$$

The convergence speed is linear, therefore this method is kind of slow.

2.2.1 Newton method

It uses second order information of f, i.e. the Hessian. The direction is set as $\mathbf{p}_k = -\mathrm{H}_f^{-1}\left(\mathbf{x}_k\right) \nabla f\left(\mathbf{x}_k\right)$ The convergence properties are related to the Hessian, and it is faster. At every iteration the speed increases. The computational cost is greater since we have to solve a linear system. So, the cost per iteration is composed of the computing of the gradient PLUS the computing of the linear system solution. So, even though iterations are more complex, less of them are required. So there's a tradeoff between the complexity and the number of iterations. There's not a best method, it depends on the cases. There are some modifications in the Newton methods we could apply. The very difficult condition to hold is that H has to be positive definite in every point we compute, so there are some relaxations to this condition. In some cases, H is approximated:

$$\mathrm{H}_f\left(\mathbf{x}_k\right) \simeq \mathrm{B}\left(\mathbf{x}_k\right)$$

or we could only use the first derivatives. A way of reducing the computational cost we can approximately solve the system: we compute the direction p_k with iterative methods.

2.3 Stochastic optimization

When we're working with milions of data, the objective function (the function to minimize) is related to probabilistic events, having the form:

$$G(\mathbf{x}) = \sum_{i=1}^{n} G_i(\mathbf{x})$$

Each G_i is the so-called *loss function.* For example, in a supervised classification problem we minimize the empirical risk function. In a standard gradient descent,

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k - \alpha_k \nabla G\left(\mathbf{x}_k\right) \\ &= \mathbf{x}_k - \alpha_k \sum_{i=1}^{n} \nabla G_i\left(\mathbf{x}_k\right) \end{aligned}$$

When working with lots of data, the gradient computation is really expensive. In stochastic gradient descent, the true gradient is approximated at each iteration by the gradient of a single observation, randomly picked. The iteration step becomes:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla G_i\left(\mathbf{x}_k\right)$$

We can compromise by computing the gradient on a small number of observations (*mini-batch*).

2.4 Least square problem

Now, we have a non-linearly square problem. We have previously analyzed linear problems only. In this case, we can suppose that our function is a general function $\mathbf{r} : \mathbb{R}^n \to \mathbb{R}^m$, where n is the dimension of the space of the data and m is the dimension of the space of the unknowns. So, we have to minimize the sum of the squares of the $r_i(x)$.

$$\min ||\mathbf{r}(\mathbf{x})||^2 = \min \sum_{i=1}^{m} r_i^2(\mathbf{x})$$

The Jacobian (J_r) is a matrix that contains the partial derivatives of r, with respect to the components. The formulas to compute the hessian or the gradient are very expensive to compute. The methods used to compute these are:

- Gradient methods
- Newton-like methods
- Gauss-Newton method, where p_k is computed as $\left(\mathrm{J}_r^T \cdot \mathrm{J}_r\right)(\mathbf{x}_k) \cdot \mathbf{p}_k = -\mathrm{J}_r^T(\mathbf{x}_k) \cdot \mathbf{r}(\mathbf{x}_k)$
- Levenberg, Marquardt method, adds regularization because $(J_r^T \cdot J_r)$ can be ill-conditioned. p_k is computed as
$$\left(\mathrm{J}_r^T(\mathbf{x}_k) \cdot \mathrm{J}_r(\mathbf{x}_k) + \lambda \mathrm{I}\right) \cdot \mathbf{p}_k = -\mathrm{J}_r^T(\mathbf{x}_k) \cdot \mathbf{r}(\mathbf{x}_k)$$
as you can see, a term is added: λI. The matrix J could be ill-conditioned, and this identity matrix multiplied by a constant λ, often used with tentative values, only changes the diagonal of $J_r^T \cdot J_r$ and has the effect of moving the spectrum of the matrix. If $A \cdot A^T$ is symmetric and positive definite, the norm of A is the square root of its largest eigenvalue. Why? If A is symmetric, $A \cdot A^T$ is A^2. If $u_1, ..., u_n$ are eigenvalues of A, $u_1^2, ..., u_n^2$ are the eigenvalues of A^2. Hence, the norm-2 of $||A||_2$ is $\sqrt{\mu_1^2} = \mu_1$. So: the 2-norm is the highest eigenvalue. The 2-norm of A^{-1} is the highest eigenvalue of $A^{-1T} \cdot A^{-1}$ is therefore the smallest eigenvalue of A, $\sqrt{\frac{1}{\mu_n^2}}$. So, the condition number of A $K(A) = ||A||_2||A^{-1}||_2 = \frac{\mu_1}{\mu_n}$. The minimum eigenvalue is small, adding a constant lammba reduces the spectrum. Regularization is often used in ML algorithms to prevent **overfitting**.

2.5 Constrained optimization

We now consider the problem

$$\begin{gathered}\min_x G(\mathbf{x}) \\ \text{s.t. } h_i(\mathbf{x}) \leq 0 i = 1, \dots m\end{gathered}$$

To solve the second one, we can introduce the Lagrange multipliers λ_i, where the Lagrangian function associated to the problem is

$$L(\mathbf{x}, \lambda) = G(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i h_i(\mathbf{x})$$

Many numerical methods also use the concept of duality. The minimization is moved in another set of variables; given a minimization problem, there is not a unique way of getting duality. We now consider the so called *Lagrangian duality*.

3 Vector calculus

When we're working with multi-value functions, we have the optimization algorithms at the basis for many ML algorithms.

3.1 Chain rule

Let's consider a function $f : \mathbb{R}^2 \to \mathbb{R}$ of two variables x_1, x_2. Furthermore, $x_1(t)$ and $x_2(t)$ are themselves functions of t. To compute the gradient of f with respect to t, we need to apply the chain rule for multivariate functions as:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1(t)}{\partial t} \\ \frac{\partial x(t)}{\partial t} \end{bmatrix} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}$$

where d denoted the gradient.