Solvers for dense and sparse quadratic problems

Alexandre Gramfort

alexandre.gramfort@telecom-paristech.fr

Telecom ParisTech

Master 2 Data Science, Univ. Paris Saclay
Optimisation for Data Science

Unconstrained optimization problem

Definition (Unconstrained optimization problem ($\mathcal{P}$))

$$\min_{x \in \mathbb{R}^n} f(x)$$

- where $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is the **objective function**

Quadratic functions

Definition (Quadratic form)

A quadratic form reads

$$f(x) = \frac{1}{2}x^\top Ax - b^\top x + c$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ and $c \in \mathbb{R}$.

$\rightarrow$ what equation verifies a stationary point? the point where the gradient is equal to zero

$\rightarrow$ what condition on A to have existence and unicity of x^*?

$\rightarrow$ show that minimizing f boils down to solving a linear system.

se reduire

grad f(x)=1/2 (A+A^t)x*-b

1/2 (A+A^t)x*=b

Taylor at order 2

Assuming f is twice differentiable, the Taylor expansion at order 2 of f at x reads:

$$\forall h \in \mathbb{R}^n,\ f(x+h) = f(x) + \nabla f(x)^\top h + \frac{1}{2} h^\top \nabla^2 f(x) h + o(\|h\|^2)$$

- $\nabla f(x) \in \mathbb{R}^n$ is the gradient.
- $\nabla^2 f(x) \in \mathbb{R}^{n\times n}$ the Hessian matrix.

Remark: It gives a local quadratic approximation

$\rightarrow$ Show that if $\nabla^2 f(x) = L\,\mathrm{I}$ then minimizing the quadratic approximation leads to gradient descent. With what step size?

h*=1/L grad f(x)

x+h*=x-(1/L)*gradf(x)

Ridge regression

We consider problems with n samples, observations, and p features, variables.

Definition (Ridge regression)

Let $y \in \mathbb{R}^n$ the n targets to predict and $(x^i)_i$ the n samples in $\mathbb{R}^p$. Ridge regression consists in solving the following problem

$$\min_{w,b} \frac{1}{2}\|y - Xw - b\|^2 + \frac{\lambda}{2}\|w\|^2 \, , \lambda > 0$$

where $w \in \mathbb{R}^p$ is called the weights vector, $b \in \mathbb{R}$ is the intercept (a.k.a. bias) and the ith row of X is x^i.

Remark: We have an optimization problem in dimension $p + 1$

Remark: Note that the intercept is not penalized with λ.

Taking care of the intercept

Exercise

Let

$$\hat{w}, \hat{b} = \arg\min_{w,b} \frac{1}{2}\|y - Xw - b\|^2 + \frac{\lambda}{2}\|w\|^2 , \lambda > 0$$

$\overline{y} \in \mathbb{R}$ *the mean of* y *and* $\overline{X} \in \mathbb{R}^p$ *the mean of each column of* X.

$\rightarrow$ *Show that* $\hat{b} = -\overline{X}^\top \hat{w} + \overline{y}$.

f(w,b)=1/2 ||y-Xw-b||^2 + lambda/2 ||w||^2
grad f(w,b) = (gradw f(w,b) , gradb f(w,b))^t

grad f(w*,b*)=0

grad(w) f(w,b)= -X^t (y-Xw-b) + lambda.w

grad(b) f(w,b)= -1|^t (y-Xw-b) = (-ÿ + ¨X^t. w + b) . n

Taking care of the intercept

Exercise

Let

$$\hat{w}, \hat{b} = \arg\min_{w,b} \frac{1}{2}\|y - Xw - b\|^2 + \frac{\lambda}{2}\|w\|^2 , \lambda > 0$$

$\overline{y} \in \mathbb{R}$ *the mean of* y *and* $\overline{X} \in \mathbb{R}^p$ *the mean of each column of* X.
$\rightarrow$ *Show that* $\hat{b} = -\overline{X}^\top \hat{w} + \overline{y}$.

Ways to deal with the intercept:

- Option 1 (dense case): Center the target y and each column feature and solve:

if feature of value is bigger than others, the descant'll be slow

$$\min_{w \in \mathbb{R}^p} \frac{1}{2}\|y - Xw\|^2 + \frac{\lambda}{2}\|w\|^2$$

- Option 2 (sparse case): Add a column of 1 to X and try not to penalize it (too much).

Ridge regression

prb is strongly convex because lamda>0
Lipschitz constant = ||X^t.X|| + lamda.Id

We consider:

$$\min_{w \in \mathbb{R}^p} \frac{1}{2} \|y - Xw\|^2 + \frac{\lambda}{2} \|w\|^2$$

Exercise

- *Show that ridge regression boils down to the minimization of a quadratic form.*
- *Propose a closed form solution.*
- *Show that the solution is obtained by solving a linear system.*
- *Is the objective function strongly convex?*
- *Assuming $n < p$ what is the value of the constant of strong convexity?*

Singular value decomposition (SVD)

- SVD is a factorization of a matrix (real here)
- $M = U\Sigma V^\top$ where $M \in \mathbb{R}^{n\times p}$, $U \in \mathbb{R}^{n\times n}$, $\Sigma \in \mathbb{R}^{n\times p}$, $V \in \mathbb{R}^{p\times p}$
- $U^\top U = UU^\top = I_n$ (orthogonal matrix)
- $V^\top V = VV^\top = I_p$ (orthogonal matrix)
- Σ diagonal matrix
- $\Sigma_{i,i}$ are called the singular values
- U are left-singular vectors
- V are right-singular vectors

Singular value decomposition (SVD)

- SVD is a factorization of a matrix (real here)
- U contains the eigenvectors of $MM^\top$ associated to the eigenvalues $\Sigma_{i,i}^2$ for $1 \leq i \leq n$.
- V contains the eigenvectors of $M^\top M$ associated to the eigenvalues $\Sigma_{i,i}^2$ for $1 \leq i \leq p$.
- we assume here $\Sigma_{i,i} = 0$ for $\min(n,p) \leq i \leq \max(n,p)$
- SVD is particularly useful to find the rank, null-space, image and pseudo-inverse of a matrix

Matrix inversion lemma

Proposition (Matrix inversion lemma)

also known as Sherman-Morrison-Woodbury formula states that:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1} VA^{-1},$$

where $A \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{n\times k}$, $C \in \mathbb{R}^{k\times k}$, $V \in \mathbb{R}^{k\times n}$.

Matrix inversion lemma

Proposition (Matrix inversion lemma)

also known as Sherman-Morrison-Woodbury formula states that:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1}VA^{-1},$$

where $A \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{n\times k}$, $C \in \mathbb{R}^{k\times k}$, $V \in \mathbb{R}^{k\times n}$.

Proof. Just check that (A+UCV) times the RHS of the Woodbury identity gives the identity matrix:

$$\begin{aligned}
& (A + UCV)\left[A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1}VA^{-1}\right] \\
= \quad & I + UCVA^{-1} - (U + UCVA^{-1}U)(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \\
= \quad & I + UCVA^{-1} - UC(C^{-1} + VA^{-1}U)(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \\
= \quad & I + UCVA^{-1} - UCVA^{-1} = I
\end{aligned}$$

Primal and dual implementation

We consider:

$$\min_{w \in \mathbb{R}^p} \frac{1}{2}\|y - Xw\|^2 + \frac{\lambda}{2}\|w\|^2$$

The solution is given by:

A = lambda . Id
X = U.E.V^t

$$\hat{w} = (X^\top X + \lambda I_p)^{-1} X^\top y$$

Using matrix inversion lemma show that:

utiliser pour calculer une matrice de dimension inférieure

$$\hat{w} = X^\top (XX^\top + \lambda I_n)^{-1} y$$

This is a dual formulation and the matrix to invert is in $\mathbb{R}^{n \times n}$.

$\rightarrow$ Using the SVD of X propose an implementation.

$\rightarrow$ Can you use the SVD to confirm the primal-dual link?

$\rightarrow$ What if X is sparse, n is 1e5 and p is 1e6? No, we use conjugate gradient

Conjugate gradient method: Solve $Ax = b$

The conjugate gradient method is an iterative method to solve linear systems with positive definite matrices ($A \succ 0$). It only needs to know how to compute Ax (operation can be implicit).

Principle:

- Iterate: $x^{k+1} = x^k - \beta_k d^k$
- The direction d^k depends on all the gradients at previous iterates $(\nabla f(x^1), \ldots, \nabla f(x^k))$.
- $p^k = \beta_k d^k$ is chosen as the vector in $\mathrm{span}(\nabla f(x^1), \ldots, \nabla f(x^k))$ which minimizes $f(x^k - p^k)$

Conjugate gradient method: Solve $Ax = b$

Theorem (Convergence in n iterations)

The conjugate gradient algorithm finds the minimum of positive definite quadratic form f, in at most n iterations.

Conjugate gradient method: Solve $Ax = b$

- Property
  - $\forall l < k,\ Ap^k \perp p^l$
  - i.e., vectors p^k and p^l are *conjugate* w.r.t. A
- Computation of the direction:
  - $d^k = g^k + \alpha_k d^{k-1}$ where $g^k = \nabla f(x^k)$ (we correct the gradient with a term that depends on previous iterations),
  - $$\alpha_k = -\frac{\langle g^k, Ad^{k-1}\rangle}{\langle Ad^{k-1}, d^{k-1}\rangle}$$
- Computation of optimal step size:
  - $$\beta_k = \frac{\langle g^k, d^k\rangle}{\langle Ad^k, d^k\rangle}$$

Conjugate gradient: Solve $Ax = b$

Require: $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$

$x^0 \in \mathbb{R}^n$, $g^0 = Ax^0 - b$
for $k = 0$ to n **do**
    if $g^k = 0$ **then**
        break
    end if
    if $k = 0$ **then**
        $d^k = g^0$
    else
        $\alpha_k = -\frac{\langle g^k, Ad^{k-1}\rangle}{\langle d^{k-1}, Ad^{k-1}\rangle}$
        $d^k = g^k + \alpha_k d^{k-1}$
    end if
    $\beta_k = \frac{\langle g^k, d^k\rangle}{\langle d^k, Ad^k\rangle}$
    $x^{k+1} = x^k - \beta_k d^k$
    $g^{k+1} = Ax^{k+1} - b$
end for
return x^{k+1}

Proof of Conjugate gradient

If $g^k = 0$, then $x^k = x^*$ is solution of the linear system $Ax = b$.
For $k = 1$, we have $d^0 = g^0$, so:

$$\begin{aligned} &\langle g^1, d^0\rangle \\ =&\langle Ax^1 - b, d^0\rangle \\ =&\langle Ax^0 - b, d^0\rangle - \beta_0\langle Ad^0, d^0\rangle \\ =&\langle g^0, d^0\rangle - \beta_0\langle Ad^0, d^0\rangle \\ =&0 \end{aligned} \tag{1}$$

by definition of β_0. This leads to

$$\langle g^1, g^0\rangle = \langle g^1, d^0\rangle = 0$$

and

$$\langle d^1, Ad^0\rangle = \langle g^1, Ad^0\rangle + \alpha_0\langle d^0, Ad^0\rangle = 0$$

by definition of α_0.

Proof of Conjugate gradient

One can prove the result by recurrence assuming that:

$$\langle g^k, g^j \rangle = 0 \text{ for } 0 \le j < k$$
$$\langle g^k, d^j \rangle = 0 \text{ for } 0 \le j < k$$
$$\langle d^k, Ad^j \rangle = 0 \text{ for } 0 \le j < k$$

If $g^k \neq 0$, the algorithm computes x^{k+1}, g^{k+1} and d^{k+1}.

Proof of Conjugate gradient

- By construction one has $\langle g^{k+1}, d^k\rangle = 0$ (cf. (1)).
- For $j < k$:

$$\begin{aligned}
&\langle g^{k+1}, d^j\rangle \\
=&\langle g^{k+1}, d^j\rangle - \langle g^k, d^j\rangle \\
=&\langle g^{k+1} - g^k, d^j\rangle \\
=&-\beta_k\langle Ad^k, d^j\rangle \\
=&0 \text{ (recurrence hypothesis)}
\end{aligned}$$

- For $j \leq k$:

$$\langle g^{k+1}, g^j\rangle = \langle g^{k+1}, d^j\rangle - \alpha_j\langle g^{k+1}, d^{j-1}\rangle = 0 \quad ,$$

since $g^j = d^j - \alpha_j d^{j-1}$.

Proof of Conjugate gradient

- Now: $d^{k+1} = g^{k+1} + \alpha_{k+1} d^k$. For $j < k$

$$\begin{aligned} &\langle d^{k+1}, Ad^j \rangle \\ =&\langle g^{k+1}, Ad^j \rangle + \alpha_{k+1} \langle d^k, Ad^j \rangle \\ =&\langle g^{k+1}, Ad^j \rangle \quad . \end{aligned}$$

  As $g^{j+1} = g^j - \beta_j Ad^j$, one obtains

$$\langle g^{k+1}, Ad^j \rangle = \frac{1}{\beta_j} \langle g^{k+1}, g^j - g^{j+1} \rangle = 0 \text{ if } \beta_j \neq 0.$$

  This implies that if $\beta_j \neq 0$, $\langle d^{k+1}, Ad^j \rangle = 0$ for $j < k$.
- Furthermore one has $\langle d^{k+1}, Ad^k \rangle = 0$.
- So $\langle d^{k+1}, Ad^j \rangle = 0$ for $j < k + 1$.

Proof of Conjugate gradient

- This completes the proof for $\beta_j \neq 0$ and $g^j \neq 0$.
- However one has that

$$\langle g^k, d^k \rangle = \langle g^k, g^k \rangle + \alpha_k \langle g^k, d^{k-1} \rangle = \|g^k\|^2 \quad ,$$

and $\beta_k = \frac{\langle g^k, d^k \rangle}{\langle A d^k, d^k \rangle}$.

- So β_k can only be 0 if $g^k = 0$, which would imply that $x^k = x^*$.
- Furthermore

$$\|d^k\|^2 = \|g^k\|^2 + \alpha_k^2 \|d^{k-1}\|^2 \quad .$$

So if $g^k \neq 0$ then $d^k \neq 0$.

Proof of Conjugate gradient

- Consequently, if the vectors g^0, g^1, $\ldots$, g^k are all non-zero, the vectors d^0, d^1, $\ldots$, d^k are also non-zero.
- These vectors are an orthogonal basis for the dot product $\langle \cdot, \cdot \rangle_A$ and the $k+1$ directions
- g^0, g^1, $\ldots$, g^k are an orthogonal basis for the dot product $\langle \cdot, \cdot \rangle$.
- These directions are therefore independent. As a consequence, if g^0, g^1, $\ldots$, g^{n-1} are all non-zero, one has that $d^n = g^n = 0$.
- So it converges after n iterations at the most.

Warm starts and paths

In machine learning it is common to try to solve a problem that is very similar to a previous one. remplace Id by lambda

- You train a model every day and you need just to "update" the model
- You look for the best hyperparmater and evaluate the parameter on a grid of values. For example on a grid of λ when doing cross-validation.

More

Note: Conjugate gradient for sparse linear systems is implemented in `scipy.sparse.linalg.cg` and in `scipy.optimize.fmin_cg`

Note: `sklearn.linear_model.Ridge` has many solvers. In v0.18 you have *'svd'*, *'cholesky'*, *'lsqr'*, *'sparse_cg'*, *'sag'* and and *'auto'* mode.

→ more in the lecture notes.
→ cf. notebook