Alternating direction method of multipliers (ADMM)

Again, this material is mostly pulled from [BPC+10]. I have uploaded the paper to T-square so you can download it if you are interested.

ADMM extends the method of multipliers in such away that we get back some of the decomposability (i.e. ability to parallelize) of standard dual ascent algorithms. It also gives us a flexible framework for incorporating many types of convex constraints, though we will again focus on linear equality constraints to start.

ADMM **splits** the optimization variable into two parts, $\boldsymbol{x}$ and $\boldsymbol{z}$, and solves programs of the form

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \;\; f_0(\boldsymbol{x}) + h(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c}.$$

The basic idea is to rotate through 3 steps:

1. Minimize the (augmented) Lagrangian over $\boldsymbol{x}$ with $\boldsymbol{z}$ and the Lagrange multipliers $\boldsymbol{\nu}$ fixed.
2. Minimize the (augmented) Lagrangian over $\boldsymbol{z}$ with $\boldsymbol{x}$ and $\boldsymbol{\nu}$ fixed.
3. Update the Lagrange multipliers using gradient ascent as before.

If the splitting is done in a careful manner, it can happen that each of the subproblems above can be easily computed and we can handle general convex constraints (more on this later).

To make the three steps above more explicit: the augmented La-

grangian is

$$L_\rho(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\nu}) = f_0(\boldsymbol{x})+h(\boldsymbol{z})+\boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c})+\frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c}\|_2^2,$$

and the general ADMM iteration is

$$\begin{aligned}
\boldsymbol{x}^{(k+1)} &= \arg\min_{\boldsymbol{x}} L_\rho(\boldsymbol{x},\boldsymbol{z}^{(k)},\boldsymbol{\nu}^{(k)}) \\
\boldsymbol{z}^{(k+1)} &= \arg\min_{\boldsymbol{z}} L_\rho(\boldsymbol{x}^{(k+1)},\boldsymbol{z},\boldsymbol{\nu}^{(k)}) \\
\boldsymbol{\nu}^{(k+1)} &= \boldsymbol{\nu}^{(k)} + \rho(\boldsymbol{A}\boldsymbol{x}^{(k+1)} + \boldsymbol{B}\boldsymbol{z}^{(k+1)} - \boldsymbol{c}).
\end{aligned}$$

The only real different between ADMM and MoM is the we are splitting the primal minimization into two parts instead of optimizing over $(\boldsymbol{x},\boldsymbol{z})$ jointly.

Scaled form.

We can write the ADMM iterations in a more convenient form by substituting

$$\boldsymbol{\mu} = \frac{1}{\rho}\boldsymbol{\nu}.$$

Then by "completing the square", you can check at home that

$$\boldsymbol{\nu}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c})+\frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c}\|_2^2 = \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c}+\boldsymbol{\mu}\|_2^2-\frac{\rho}{2}\|\boldsymbol{\mu}\|_2^2,$$

and so we can write:

ADMM:

$$\begin{aligned}
\boldsymbol{x}^{(k+1)} &= \arg\min_{\boldsymbol{x}} \left(f_0(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z}^{(k)} - \boldsymbol{c} + \boldsymbol{\mu}^{(k)}\|_2^2 \right) \\
\boldsymbol{z}^{(k+1)} &= \arg\min_{\boldsymbol{z}} \left(h(\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x}^{(k+1)} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c} + \boldsymbol{\mu}^{(k)}\|_2^2 \right) \\
\boldsymbol{\mu}^{(k+1)} &= \boldsymbol{\mu}^{(k)} + \boldsymbol{A}\boldsymbol{x}^{(k+1)} + \boldsymbol{B}\boldsymbol{z}^{(k+1)} - \boldsymbol{c}
\end{aligned}$$

Example: the LASSO

The following ℓ_1 regularized least-squares problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\tau\|\boldsymbol{x}\|_1$$

is called the LASSO; it is prevalent all across machine learning, model selection in statistics, and compressed sensing in signal processing. The $\tau>0$ above is a user-defined "smoothing parameter".

Taking

$$f_0(\boldsymbol{x})=\frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2 \quad \text{and} \quad h(\boldsymbol{z})=\tau\|\boldsymbol{z}\|_1,$$

we can rewrite this in ADMM form as

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \ f_0(\boldsymbol{x})+h(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x}-\boldsymbol{z}=\boldsymbol{0}.$$

The $\boldsymbol{x}$ update is

$$\boldsymbol{x}^{(k+1)}=\arg\min_{\boldsymbol{x}}\left(\frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\frac{\rho}{2}\|\boldsymbol{x}-\boldsymbol{z}^{(k)}+\boldsymbol{\mu}^{(k)}\|_2^2\right).$$

With both $\boldsymbol{z}^{(k)}$ and $\boldsymbol{\mu}^{(k)}$ fixed, this is equivalent to the least-squares problem:

$$\min_{\boldsymbol{x}} \ \left\|\begin{bmatrix}\boldsymbol{A}\\ \sqrt{\rho}\,\mathbf{I}\end{bmatrix}\boldsymbol{x}-\begin{bmatrix}\boldsymbol{b}\\ \sqrt{\rho}(\boldsymbol{z}^{(k)}-\boldsymbol{\mu}^{(k)})\end{bmatrix}\right\|_2^2.$$

This problem has a closed-form solution:

$$\begin{aligned}\boldsymbol{x}^{(k+1)}&=\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}+\rho\mathbf{I}\right)^{-1}\begin{bmatrix}\boldsymbol{A}^{\mathrm{T}} & \sqrt{\rho}\,\mathbf{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{b}\\ \sqrt{\rho}(\boldsymbol{z}^{(k)}-\boldsymbol{\mu}^{(k)})\end{bmatrix}\\ &=\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}+\rho\mathbf{I}\right)^{-1}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}+\rho(\boldsymbol{z}^{(k)}-\boldsymbol{\mu}^{(k)}))\end{aligned}$$

The $\boldsymbol{z}$ update update problem is:

$$\underset{\boldsymbol{z}}{\text{minimize}}\, \tau\|\boldsymbol{z}\|_1 + \frac{\rho}{2}\|\boldsymbol{z} - \boldsymbol{x}^{(k+1)} - \boldsymbol{\mu}^{(k)}\|_2^2.$$

Using the results from the Technical Details section below, we have a closed form for this as well:

$$\boldsymbol{z}^{(k+1)} = T_{\tau/\rho}(\boldsymbol{x}^{(k+1)} + \boldsymbol{\mu}^{(k)}),$$

where $T_\lambda(\cdot)$ is the term-by-tern soft-thresholding operator,

$$(T_\lambda(\boldsymbol{v}))[n] = \begin{cases} v[n] - \lambda, & v[n] > \lambda, \\ 0, & |v[n]| \leq \lambda, \\ v[n] + \lambda, & v[n] < -\lambda. \end{cases}$$

To summarize:

ADMM iterations for the LASSO

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \rho\mathbf{I}\right)^{-1}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} + \rho(\boldsymbol{z}^{(k)} - \boldsymbol{\mu}^{(k)})), \\ \boldsymbol{z}^{(k+1)} &= T_{\tau/\rho}(\boldsymbol{x}^{(k+1)} + \boldsymbol{\mu}^{(k)}), \\ \boldsymbol{\mu}^{(k+1)} &= \boldsymbol{\mu}^{(k)} + \boldsymbol{x}^{(k+1)} - \boldsymbol{z}^{(k+1)}. \end{aligned}$$

Convergence properties

We will state one convergence result. If the following two conditions hold:

1. f_0 and h are closed, proper, and convex (i.e. their epigraphs are nonempty closed convex sets),
2. strong duality holds,

then

- $\boldsymbol{A}\boldsymbol{x}^{(k)} + \boldsymbol{B}\boldsymbol{z}^{(k)} - \boldsymbol{c} \to \boldsymbol{0}$ as $k \to \infty$. That is, the primal iterates are asymptotically feasible.
- $f_0(\boldsymbol{x}^{(k)}) + h(\boldsymbol{z}^{(k)}) \to p^\star$ as $k \to \infty$. That is, the value of the objective function approaches the optimal value asymptotically.
- $\boldsymbol{\nu}^{(k)} \to \boldsymbol{\nu}^\star$ as $k \to \infty$, where $\boldsymbol{\nu}^\star$ is a dual optimal point.

Under additional assumptions, we can also have convergence to a primal optimal point, $(\boldsymbol{x}^{(k)}, \boldsymbol{z}^{(k)}) \to (\boldsymbol{x}^\star, \boldsymbol{z}^\star)$ as $k \to \infty$.

See [BPC+10, Section 3.2] for further discussion and references.

Convex constraints

We can write the general program

$$\underset{\boldsymbol{x}\in\mathcal{C}}{\text{minimize}} \;\; f_0(\boldsymbol{x}),$$

where $\mathcal{C}$ is a closed convex set, in ADMM form as

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f_0(\boldsymbol{x}) + h(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0},$$

where $h(\boldsymbol{z})$ is the indicator function for $\mathcal{C}$:

$$h(\boldsymbol{z}) = \begin{cases} 0, & \boldsymbol{z} \in \mathcal{C}, \\ \infty, & \boldsymbol{z} \notin \mathcal{C}. \end{cases}$$

Note that in this case, the $\boldsymbol{z}$ update is a closest-point-to-a-convex-set problem. For fixed $\boldsymbol{v} \in \mathbb{R}^N$,

$$\begin{aligned} \arg\min_{\boldsymbol{z}} h(\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{z} - \boldsymbol{v}\|_2^2 &= \arg\min_{\boldsymbol{z}\in\mathcal{C}} \|\boldsymbol{z} - \boldsymbol{v}\|_2 \\ &= P_{\mathcal{C}}(\boldsymbol{v}) \quad \text{(closest point in } \mathcal{C} \text{ to } \boldsymbol{v}). \end{aligned}$$

ADMM iteration for general convex constraints:

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \arg\min_{\boldsymbol{x}} \left(f_0(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{z}^{(k)} + \boldsymbol{\mu}^{(k)}\|_2^2 \right), \\ \boldsymbol{z}^{(k+1)} &= P_{\mathcal{C}}\left(\boldsymbol{x}^{(k+1)} + \boldsymbol{\mu}^{(k)} \right), \\ \boldsymbol{\mu}^{(k+1)} &= \boldsymbol{\mu}^{(k)} + \boldsymbol{x}^{(k+1)} - \boldsymbol{z}^{(k+1)}. \end{aligned}$$

Of course, this algorithm is most attractive when we have a fast method for computing $P_{\mathcal{C}}(\cdot)$.

Example: Basis Pursuit

A good proxy for finding the sparsest solution to an underdetermined system of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is to solve

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

To put this in ADMM form, we are solving

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \ f_0(\boldsymbol{x}) + h(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0},$$

with

$$f_0(\boldsymbol{x}) = \|\boldsymbol{x}\|_1, \quad \text{and} \quad h(\boldsymbol{z}) = \begin{cases} 0, & \boldsymbol{A}\boldsymbol{z} = \boldsymbol{b}, \\ \infty, & \text{otherwise.} \end{cases}$$

The projection onto $\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ can be given in closed form using the pseudo-inverse $\boldsymbol{A}^+$ of $\boldsymbol{A}$ as

$$\begin{aligned} P_{\mathcal{C}}(\boldsymbol{v}) &= \boldsymbol{A}^+(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{v}) + \boldsymbol{v} \\ &= (\mathbf{I} - \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{A})\boldsymbol{v} + \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b}, \end{aligned}$$

where the last equality comes from $\boldsymbol{A}^+ = \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}$ when $\boldsymbol{A}$ has full row rank.

The updates in this case are

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \arg\min_{\boldsymbol{x}} \left(\|\boldsymbol{x}\|_1 + \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{z}^{(k)} + \boldsymbol{\mu}^{(k)}\|_2^2 \right) \\ &= T_{1/\rho}(\boldsymbol{z}^{(k)} - \boldsymbol{\mu}^{(k)}) \\ \boldsymbol{z}^{(k+1)} &= (\mathbf{I} - \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{A})(\boldsymbol{x}^{(k+1)} + \boldsymbol{\mu}^{(k)}) + \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b} \\ \boldsymbol{\mu}^{(k+1)} &= \boldsymbol{\mu}^{(k)} + \boldsymbol{x}^{(k+1)} - \boldsymbol{z}^{(k+1)}. \end{aligned}$$

Example: Linear programming

Consider the general linear program

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \ \ \boldsymbol{x} \geq \boldsymbol{0},$$

where $\boldsymbol{A}$ is an $M \times N$ matrix with full row rank[1]. We can put this in ADMM form by first eliminating the equality constraints, then introducing the indication function for the non-negativity constraint.

Let $\boldsymbol{Q}$ be an $N \times (N - M)$ matrix whose columns span Null($\boldsymbol{A}$), and let $\boldsymbol{x}_0$ be any point such that $\boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{b}$. Then we can re-write the LP as

$$\underset{\boldsymbol{w}}{\text{minimize}} \ \boldsymbol{c}^{\mathrm{T}}(\boldsymbol{x}_0 + \boldsymbol{Q}\boldsymbol{w}) \quad \text{subject to} \quad \boldsymbol{x}_0 + \boldsymbol{Q}\boldsymbol{w} \geq \boldsymbol{0},$$

which we can write in ADMM form as

$$\underset{\boldsymbol{w}}{\text{minimize}} \, \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_0 + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{w} + h(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{Q}\boldsymbol{w} - \boldsymbol{z} = -\boldsymbol{x}_0,$$

where

$$h(\boldsymbol{z}) = \begin{cases} 0, & \boldsymbol{z} \geq \boldsymbol{0}, \\ \infty, & \text{otherwise.} \end{cases}$$

(We can drop the $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_0$ from the objective since it does not depend on either of the optimization variables.)

Notice that when $\boldsymbol{Q}$ has full column rank, the program

$$\underset{\boldsymbol{w}}{\text{minimize}} \ \boldsymbol{v}^{\mathrm{T}}\boldsymbol{w} + \frac{1}{2}\|\boldsymbol{Q}\boldsymbol{w} - \boldsymbol{y}\|_2^2,$$

[1] The full row rank assumption is not at all essential; I am just making it to keep things streamlined.

has the closed-form solution

$$\boldsymbol{w}^\star = (\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q})^{-1}(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{v}).$$

Also, the projection onto the non-negative orthant $\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{x} \geq \boldsymbol{0}\}$ is

$$P_{\mathcal{C}}(\boldsymbol{v}) = (\boldsymbol{v})_+, \quad \text{or} \quad (P_{\mathcal{C}}(\boldsymbol{v}))[n] = \begin{cases} v[n], & v[n] \geq 0, \\ 0, & v[n] < 0. \end{cases}$$

For the general linear program, then, the ADMM iterations are

$$\begin{aligned}
\boldsymbol{w}^{(k+1)} &= \arg\min_{\boldsymbol{w}} \left(\frac{1}{\rho}\boldsymbol{c}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{w} + \frac{1}{2}\|\boldsymbol{Q}\boldsymbol{w} - \boldsymbol{z}^{(k)} + \boldsymbol{x}_0 + \boldsymbol{\mu}^{(k)}\|_2^2 \right) \\
&= (\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q})^{-1} \left[\boldsymbol{Q}^{\mathrm{T}}(\boldsymbol{z}^{(k)} - \boldsymbol{x}_0 - \boldsymbol{\mu}^{(k)}) - \frac{1}{\rho}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{c} \right], \\
\boldsymbol{z}^{(k+1)} &= P_{\mathcal{C}}(\boldsymbol{Q}\boldsymbol{w}^{(k+1)} + \boldsymbol{x}_0 + \boldsymbol{\mu}^{(k)}) \\
&= \left(\boldsymbol{Q}\boldsymbol{w}^{(k+1)} + \boldsymbol{x}_0 + \boldsymbol{\mu}^{(k)} \right)_+ \\
\boldsymbol{\mu}^{(k+1)} &= \boldsymbol{\mu}^{(k)} + \boldsymbol{Q}\boldsymbol{w}^{(k+1)} - \boldsymbol{z}^{(k+1)} + \boldsymbol{x}_0.
\end{aligned}$$

Notice that especially when the columns of $\boldsymbol{Q}$ are orthogonal, $\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q} = \mathbf{I}$, all of these steps are very simple.

Distributed Recovery/Regression/Classification using ADMM

By being very crafty with how we do the splitting, we can use ADMM to solve certain kinds of optimization programs in a distributed manner.

We consider (this material comes from [BPC+10, Sec. 8]) the general problem of "fitting" a vector $\boldsymbol{x} \in \mathbb{R}^N$ to an observed vector $\boldsymbol{b} \in \mathbb{R}^M$ through an $M \times N$ matrix $\boldsymbol{A}$. We will encourage $\boldsymbol{x}$ to have certain structure using a regularizer. This type of problem is ubiquitous in signal processing and machine learning — the math stays the same, only the words change from area to area.

At a high level, we are interested in solving

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \text{Loss}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \ + \ \text{Regularizer}(\boldsymbol{x})$$

where the $M \times N$ matrix $\boldsymbol{A}$ and the M-vector $\boldsymbol{b}$ are given. Notice that

$$\text{Loss}(\cdot) : \mathbb{R}^M \to \mathbb{R}, \quad \text{and} \quad \text{Regularizer}(\cdot) : \mathbb{R}^N \to \mathbb{R}.$$

We will assume that one or both of these functions are separable, at least at the block level. This means we can write

$$\text{Loss}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \sum_{i=1}^{B} \ell_i(\boldsymbol{A}_i\boldsymbol{x} - \boldsymbol{b}_i),$$

where $\boldsymbol{A}_i$ are $M_i \times N$ matrices formed by partitioning the rows of $\boldsymbol{A}$, and $\boldsymbol{b}_i \in \mathbb{R}^{M_i}$ is the corresponding part of $\boldsymbol{b}$. For separable regularizers, we can write

$$\text{Regularizer}(\boldsymbol{x}) = \sum_{i=1}^{C} r_i(\boldsymbol{x}_i),$$

where the $\boldsymbol{x}_i \in \mathbb{R}^{N_i}$ partition the vector $\boldsymbol{x}$. These two types of separability will allow us to divide up the optimization in two different ways.

Example: Inverse Problems and Regression

Two popular methods for solving linear inverse problems and/or calculating regressors are solving

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \tau\|\boldsymbol{x}\|_2^2,$$

(*Tikhonov regularization* or *ridge regression*), and

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \tau\|\boldsymbol{x}\|_1,$$

(*basis pursuit denoising* or *the LASSO*).

These both clearly fit the separability criteria, as

$$\begin{aligned}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 &= \sum_{m=1}^{M}(\langle \boldsymbol{x}, \boldsymbol{a}_m\rangle - b[m])^2,\\ \|\boldsymbol{x}\|_2^2 &= \sum_{n=1}^{N}(x[n])^2\\ \|\boldsymbol{x}\|_1 &= \sum_{n=1}^{N}|x[n]|.\end{aligned}$$

where $\boldsymbol{a}_m^{\mathrm{T}}$ is the mth row of $\boldsymbol{A}$.

Example: Support Vector Machines

Previously, we saw how if we are given a set of M training examples $(\boldsymbol{x}_m, y_m)$, where $\boldsymbol{x}_m \in \mathbb{R}^N$ and $y_m \in \{-1, 1\}$, we can find a maximum margin linear classifier by solving

$$\min_{\boldsymbol{w},z} \ \|\boldsymbol{w}\|_2^2 \quad \text{subject to} \quad y_m(z - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1 \leq 0, \quad m = 1, \ldots, M.$$

With the classifier trained (optimal solution $\boldsymbol{w}^\star, z^\star$ computed), we can assign a label y' to a new point $\boldsymbol{x}'$ using

$$y' = \text{sign}(\langle \boldsymbol{x}', \boldsymbol{w}^\star \rangle + z^\star).$$

Instead of enforcing the constraints above strictly, we can allow some errors by penalizing mis-classifications on the training data appropriately. One reasonable way to do this is make the loss zero if $y_m(z - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1 \leq 0$, and then have it increase linearly as this quantity exceeds zero. That is, we solve

$$\min_{\boldsymbol{w},z} \sum_{m=1}^{M} \ell(y_m(z - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1) + \|\boldsymbol{w}\|_2^2,$$

where $\ell(\cdot)$ is the **hinge loss**

$$\ell(u) = (u)_+ = \begin{cases} 0, & u \leq 0, \\ u, & u > 0. \end{cases}$$

So "soft margin" SVM fits our model as what is inside the $\ell(\cdot)$ can be written as an affine function of the optimization variables:

$$y_m(z - \langle \boldsymbol{x}_m, \boldsymbol{w} \rangle) + 1 = \begin{bmatrix} -y_m \boldsymbol{x}_m & y_m \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ z \end{bmatrix} + 1.$$

Splitting across examples

This framework is useful when we have "many measurements of a small vector" or "large volumes of low-dimensional data".

We partition the rows of $\boldsymbol{A}$ and entries of $\boldsymbol{b}$:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \vdots \\ \boldsymbol{A}_B \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_B \end{bmatrix}.$$

If the loss function is separable over this partition, our optimization problem is

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \sum_{i=1}^{B} \ell_i(\boldsymbol{A}_i\boldsymbol{x} - \boldsymbol{b}_i) + r(\boldsymbol{x}),$$

where $r(\cdot)$ is the regularizer. We start by splitting the optimization variables in the loss function and those in the regularizer, arriving at the equivalent program

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \sum_{i=1}^{B} \ell_i(\boldsymbol{A}_i\boldsymbol{x} - \boldsymbol{b}_i) + r(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0}.$$

This does not make the Lagrangian for the primal update separable, as the $\boldsymbol{A}_i$ are still tying together all of the entries in $\boldsymbol{x}$. The trick is to introduce B different $\boldsymbol{x}_i \in \mathbb{R}^N$, one for each block, and then use the constraints to make them all agree. This is done with

$$\underset{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_B}{\text{minimize}} \ \sum_{i=1}^{B} \ell_i(\boldsymbol{A}_i\boldsymbol{x}_i - \boldsymbol{b}_i) + r(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{x}_i - \boldsymbol{z} = \boldsymbol{0}, \ \ i = 1, \ldots, B.$$

The augmented Lagrangian for this last problem is

$$L_\rho(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_B,\boldsymbol{z},\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_B)=\sum_{i=1}^{B}\ell_i(\boldsymbol{A}_i\boldsymbol{x}_i-\boldsymbol{b}_i)+\frac{\rho}{2}\sum_{i=1}^{B}\|\boldsymbol{x}_i-\boldsymbol{z}+\boldsymbol{\mu}_i\|_2^2+r(\boldsymbol{z}),$$

where $\boldsymbol{\mu}_i$ are the (rescaled) Lagrange multipliers for the constraint $\boldsymbol{x}_i-\boldsymbol{z}=\boldsymbol{0}$.

As the Lagrangian is separable over the B blocks, each of the primal updates for the $\boldsymbol{x}_i$ can be performed independently. This makes the ADMM iteration

$$\begin{aligned}\boldsymbol{x}_i^{(k+1)}&=\arg\min_{\boldsymbol{x}_i}\left(\ell_i(\boldsymbol{A}_i\boldsymbol{x}_i-\boldsymbol{b}_i)+\frac{\rho}{2}\|\boldsymbol{x}_i-\boldsymbol{z}^{(k)}+\boldsymbol{\mu}_i^{(k)}\|_2^2\right)\\&\qquad i=1,\ldots,B\end{aligned}$$

$$\boldsymbol{z}^{(k+1)}=\arg\min_{\boldsymbol{z}}\left(r(\boldsymbol{z})+\frac{\rho}{2}\sum_{i=1}^{B}\|\boldsymbol{z}-\boldsymbol{x}_i^{(k+1)}-\boldsymbol{\mu}_i^{(k)}\|_2^2\right)$$

$$\begin{aligned}\boldsymbol{\mu}_i^{(k+1)}&=\boldsymbol{\mu}_i^{(k)}+\boldsymbol{x}_i^{(k+1)}-\boldsymbol{z}^{(k+1)}\\&\qquad i=1,\ldots,B.\end{aligned}$$

The $\boldsymbol{z}$ update can be written in terms of the average of the $\boldsymbol{x}_i^{(k+1)}$. To see this, first note that

$$\begin{aligned}\sum_{i=1}^{B}\|\boldsymbol{z}-\boldsymbol{v}_i\|_2^2&=B\|\boldsymbol{z}\|_2^2-2\left\langle\boldsymbol{z},\sum_{i=1}^{B}\boldsymbol{v}_i\right\rangle+\sum_{i=1}^{N}\|\boldsymbol{v}_i\|_2^2\\&=B\|\boldsymbol{z}\|_2^2-2B\left\langle\boldsymbol{z},\bar{\boldsymbol{v}}\right\rangle+B\|\bar{\boldsymbol{v}}\|_2^2+\left(-B\|\bar{\boldsymbol{v}}\|_2^2+\sum_{i=1}^{N}\|\boldsymbol{v}_i\|_2^2\right)\\&=B\|\boldsymbol{z}-\bar{\boldsymbol{v}}\|_2^2+\left(-B\|\bar{\boldsymbol{v}}\|_2^2+\sum_{i=1}^{N}\|\boldsymbol{v}_i\|_2^2\right).\end{aligned}$$

where $\bar{\boldsymbol{v}} = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{v}_i$. Thus

$$\begin{aligned}
&\arg\min_{\boldsymbol{z}} \left(r(\boldsymbol{z}) + \frac{\rho}{2} \sum_{i=1}^{B} \|\boldsymbol{z} - \boldsymbol{x}_i^{(k+1)} - \boldsymbol{\mu}_i^{(k)}\|_2^2 \right) \\
&= \arg\min_{\boldsymbol{z}} \left(r(\boldsymbol{z}) + \frac{B\rho}{2} \|\boldsymbol{z} - \bar{\boldsymbol{x}}^{(k+1)} - \bar{\boldsymbol{\mu}}^{(k)}\|_2^2 \right)
\end{aligned}$$

Distributed ADMM (dividing rows of $\boldsymbol{A}$)

$$\boldsymbol{x}_i^{(k+1)} = \arg\min_{\boldsymbol{x}_i} \left(\ell_i(\boldsymbol{A}_i\boldsymbol{x}_i - \boldsymbol{b}_i) + \frac{\rho}{2} \|\boldsymbol{x}_i - \boldsymbol{z}^{(k)} + \boldsymbol{\mu}_i^{(k)}\|_2^2 \right)$$
$$i = 1, \ldots, B$$

$$\boldsymbol{z}^{(k+1)} = \arg\min_{\boldsymbol{z}} \left(r(\boldsymbol{z}) + \frac{B\rho}{2} \|\boldsymbol{z} - \bar{\boldsymbol{x}}^{(k+1)} - \bar{\boldsymbol{\mu}}^{(k)}\|_2^2 \right)$$

$$\boldsymbol{\mu}_i^{(k+1)} = \boldsymbol{\mu}_i^{(k)} + \boldsymbol{x}_i^{(k+1)} - \boldsymbol{z}^{(k+1)}$$
$$i = 1, \ldots, B.$$

where

$$\bar{\boldsymbol{x}}^{(k+1)} = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{x}_i^{(k+1)}, \quad , \bar{\boldsymbol{\mu}}^{(k)} = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{\mu}_i^{(k)}.$$

The high-level architecture is that B separate units solve independent optimization programs for the B $\boldsymbol{x}_i$ updates. These are collected and averaged, and a single optimization program is solved to get the $\boldsymbol{x}$ update. The new $\boldsymbol{z}$ is then communicated back to each of the B units. The Largrange multiplier update can easily be com-

puted both centrally and at the B units, so these do not have to be communicated.

Example: the LASSO

With $\ell_i(\boldsymbol{A}_i\boldsymbol{x}_i - \boldsymbol{b}_i) = \|\boldsymbol{A}_i\boldsymbol{x}_i - \boldsymbol{b}_i\|_2^2$ and $r(\boldsymbol{x}) = \tau\|\boldsymbol{x}\|_1$, the ADMM iteration becomes

$$\boldsymbol{x}_i^{(k+1)} = \arg\min_{\boldsymbol{x}_i} \left(\|\boldsymbol{A}_i\boldsymbol{x}_i - \boldsymbol{b}_i\|_2^2 + \frac{\rho}{2}\|\boldsymbol{x}_i - \boldsymbol{z}^{(k)} + \boldsymbol{\mu}_i^{(k)}\|_2^2 \right)$$
$$i = 1, \ldots, B$$

$$\boldsymbol{z}^{(k+1)} = T_{\tau/(B\rho)}\left(\bar{\boldsymbol{x}}^{(k+1)} + \bar{\boldsymbol{\mu}}^{(k)} \right)$$

$$\boldsymbol{\mu}_i^{(k+1)} = \boldsymbol{\mu}_i^{(k)} + \boldsymbol{x}_i^{(k+1)} - \boldsymbol{z}^{(k+1)}$$
$$i = 1, \ldots, B.$$

The $\boldsymbol{x}_i$ updates are all small unconstrained least-squares problems whose solutions can be computed independently; the $\boldsymbol{z}$ update is a simple soft thresholding, and the $\boldsymbol{\mu}_i$ updates are computed simply by adding vectors.

Example: SVM

For the SVM, we collect the weights and the offset into a single optimization vector

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{w} \\ z \end{bmatrix} \in \mathbb{R}^{N+1}$$

and use

$$\boldsymbol{A}_i = \begin{bmatrix} -y_1\boldsymbol{x}_1 & y_1 \\ \vdots & \vdots \\ -y_{N_1}\boldsymbol{x}_{N_1} & y_{N_1}. \end{bmatrix}$$

Note that the regularization does not include the last term in $\boldsymbol{v}$:

$$r(\boldsymbol{v}) = \sum_{n=1}^{N} |v[n]|^2.$$

This makes the ADMM iteration

$$\boldsymbol{v}_i^{(k+1)} = \arg\min_{\boldsymbol{v}_i} \left(\mathbf{1}^{\mathrm{T}} (\boldsymbol{A}_i \boldsymbol{v}_i + \mathbf{1})_+ + \frac{\rho}{2} \|\boldsymbol{v}_i - \boldsymbol{z}^{(k)} + \boldsymbol{\mu}_i^{(k)}\|_2^2 \right)$$

$$\boldsymbol{z}_{1:N}^{(k+1)} = \frac{\rho}{1 + N\rho} \left(\bar{\boldsymbol{v}}_{1:N}^{(k+1)} + \bar{\boldsymbol{\mu}}_{1:N}^{(k)} \right)$$

$$z^{(k+1)}[N+1] = \bar{\boldsymbol{v}}^{(k+1)}[N+1] + \bar{\boldsymbol{\mu}}^{(k)}[N+1]$$

$$\boldsymbol{\mu}_i^{(k+1)} = \boldsymbol{\mu}_i^{(k)} + \boldsymbol{v}_i^{(k+1)} - \boldsymbol{z}^{(k+1)}.$$

where $\boldsymbol{x}_{1:N}$ is the first N entries of the vector $\boldsymbol{x}$, and $x[N+1]$ is the last entry.

Splitting across features

Similarly, we can divide up the *columns* of $\boldsymbol{A}$. This is described in [BPC+10, Section 8.3].

Technical Details: Decoupled ℓ_1 minimization

Consider the optimization problem

$$\underset{\boldsymbol{z}}{\text{minimize}} \ \lambda\|\boldsymbol{z}\|_1 + \frac{1}{2}\|\boldsymbol{z}-\boldsymbol{v}\|_2^2, \tag{1}$$

where $\boldsymbol{v}$ is a fixed vector. This program is separable:

$$\begin{aligned}
&\underset{\boldsymbol{z}}{\text{minimize}} \ \lambda\sum_{n=1}^{N}|z[n]| + \frac{1}{2}\sum_{n=1}^{N}(z[n]-v[n])^2 \\
&= \underset{\boldsymbol{z}}{\text{minimize}} \ \sum_{n=1}^{N}\left(\lambda|z[n]| + \frac{1}{2}(z[n]-v[n])^2\right)
\end{aligned}$$

and so we can solve each 1 dimensional problem individually.

For fixed $v \in \mathbb{R}$, we can compute the minimizer of

$$\underset{z\in\mathbb{R}}{\text{minimize}} \ \lambda|z| + \frac{1}{2}(z-v)^2$$

explicitly. This function is convex, and is differentiable everywhere except at $z=0$. Away from zero, the derivative is

$$\frac{\mathrm{d}f}{\mathrm{d}z} = \begin{cases} \lambda + z - v, & z > 0 \\ -\lambda + z - v, & z < 0. \end{cases}$$

For the optimal value $z^\star$ to be positive, we need $\lambda + z^\star - v = 0$; this can only hold for $z^\star > 0$ if $v > \lambda$. Similarly, for $z^\star$ to be negative, we need $-\lambda + z^\star - v = 0$; this can only hold for $z^\star < 0$ if $v < -\lambda$. If neither of these conditions hold, we must have $z^\star = 0$. Thus

$$z^\star = \begin{cases} v-\lambda, & v > \lambda \\ 0, & |v| \le \lambda \\ v+\lambda, & v < -\lambda. \end{cases}$$

We use $T_\lambda(\cdot)$ to denote the nonlinear mapping above, and so

$$z^\star = T_\lambda(v).$$

T_λ is called a *soft thresholding* or *shrinkage* operator.

The solution to (1) just applies the shrinkage operator term by term:

$$\boldsymbol{z}^\star = T_\lambda(\boldsymbol{v}), \quad \text{or} \quad z^\star[n] = T_\lambda(v[n]).$$

References

[BPC+10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2010.