Machine Learning Algorithms - Optimization algorithms

Caio Corro

Université Paris-Saclay, CNRS, Laboratoire Interdisciplinaire des Sciences du Numérique,
91400, Orsay, France

Linear models 1 / 3

Regression model

- Input: $\boldsymbol{x} \in \mathbb{R}^d$
- Output: $y \in \mathbb{R}$
- Scoring function: $s_{\boldsymbol{\theta}}(\boldsymbol{x}) = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$,
  parameters $\boldsymbol{\theta} = (\boldsymbol{a}, b)$ with $\boldsymbol{a} \in \mathbb{R}^d$ and $b \in \mathbb{R}$
- Prediction function: $\widehat{y}(w) = w$

Loss functions for regressions

- Square error (or quadratic error): $\ell(y, w) = \frac{1}{2}(y - w)^2$
- Absolute error: $\ell(y, w) = |y - w|$

Linear models 2 / 3

Binary classification model

- Input: $\boldsymbol{x} \in \mathbb{R}^d$
- Output: $y \in \{0, 1\}$ or $y \in [0, 1]$, for SVM is often easier to work with $y \in \{-1, 1\}$
- Scoring function: $s_{\boldsymbol{\theta}}(\boldsymbol{x}) = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$,
  parameters $\boldsymbol{\theta} = (\boldsymbol{a}, b)$ with $\boldsymbol{a} \in \mathbb{R}^d$ and $b \in \mathbb{R}$
- Prediction function:
  - Deterministic: $\widehat{y}(w) = \begin{cases} 1 & \text{if} w \geq 0, \\ 0 & \text{otherwise} \end{cases}$
  - Probabilistic: $\widehat{y}(w) = \sigma(w) = \frac{\exp(w)}{1+\exp(w)}$

Loss functions for binary classification

- Hinge loss: $\ell(y, w) = \max(0, 1 - (2y - 1)w)$
- Negative log-likelihood: $\ell(y, w) = -wy + \log(1 + \exp(w))$

Linear models 3 / 3

Multiclass classification model, k classes

- Input: $\boldsymbol{x} \in \mathbb{R}^d$
- Output: $\boldsymbol{y} \in E(k)$ or $\boldsymbol{y} \in \triangle(k)$
- Scoring function: $s_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$,
  parameters $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{b})$ with $\boldsymbol{A} \in \mathbb{R}^{k \times d}$ and $\boldsymbol{b} \in \mathbb{R}^k$
- Prediction function:
  - Deterministic: $\widehat{\boldsymbol{y}}(\boldsymbol{w}) = \arg\max_{\boldsymbol{y} \in E(k)} \langle \boldsymbol{w}, \boldsymbol{y} \rangle$
  - Probabilistic: $\widehat{\boldsymbol{y}}(\boldsymbol{w}) = \mathrm{softmax}(\boldsymbol{w})$

Loss functions for multiclass classification

- Hinge loss: $\ell(y, w) = \max(0, 1 - \langle \boldsymbol{w}, \boldsymbol{y} \rangle \max_{\boldsymbol{y}' \in E(y) \setminus \boldsymbol{y}} \langle \boldsymbol{w}, \boldsymbol{y}' \rangle)$
- Negative log-likelihood: $\ell(\boldsymbol{y}, \boldsymbol{w}) = -\langle \boldsymbol{w}, \boldsymbol{y} \rangle + \log \sum_i \exp(w_i)$

Linear model training 1 / 2

Data distribution

We denote $p(\mathbf{x}, \mathbf{y})$ the data distribution where:

- $\mathbf{x}$: random variables over inputs
- $\mathbf{y}$: random variables over outputs

Linear model training 1 / 2

Data distribution

We denote $p(\mathbf{x}, \mathbf{y})$ the data distribution where:

- $\mathbf{x}$: random variables over inputs
- $\mathbf{y}$: random variables over outputs

Training problem

Find the model parameters that minimize the expected loss of the data distribution:

$$\min_{\boldsymbol{\theta}} \quad \mathbb{E}_{p(\mathbf{x},\mathbf{y})}[\ \ell(\mathbf{y}, s_{\boldsymbol{\theta}}(\mathbf{x}))\ ] \quad + \quad \alpha r(\theta)$$

- ℓ: loss function
- r: regularization function, usually not applied to all parameters in $\boldsymbol{\theta}$ (i.e. not applied to the bias/intercept term)
- $\alpha \geq 0$: regularization weight

Linear model training 2 / 2

Monte-Carlo estimation

We approximate the true expected loss using samples from the data distribution:

$$\mathbb{E}_{p(\mathbf{x},\mathbf{y})}[\ \ell(\mathbf{y}, s_\theta(\mathbf{x}))\ ] \quad \simeq \quad \frac{1}{|D|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in D} \ell(\boldsymbol{y}, s_\theta(\boldsymbol{x}))$$

where the training dataset D contains $|D|$ samples from the data distribution.

Linear model training 2 / 2

Monte-Carlo estimation

We approximate the true expected loss using samples from the data distribution:

$$\mathbb{E}_{p(\mathbf{x},\mathbf{y})}[\ \ell(\mathbf{y}, s_\theta(\mathbf{x}))\ ] \quad \simeq \quad \frac{1}{|D|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in D} \ell(\boldsymbol{y}, s_\theta(\boldsymbol{x}))$$

where the training dataset D contains $|D|$ samples from the data distribution.

Convexity

If

- the scoring function is linear
- the loss is convex
- the regularization function is convex

then the training problem object is convex.

$\Longrightarrow$ you have all the tools to prove this! (be very careful with the scoring function)

Generic optimization problem 1/2

Reweighting

Sometimes it is easier to absorb the $\frac{1}{|D|}$ factor in the regularization weight:

$$
\begin{aligned}
& \arg\min_{\boldsymbol{\theta}} \quad \frac{1}{|D|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in D} \ell(\boldsymbol{y}, s_{\boldsymbol{\theta}}(\boldsymbol{x})) \quad + \quad \alpha r(\theta) \\
= \quad & \arg\min_{\boldsymbol{\theta}} \quad \sum_{(\boldsymbol{x},\boldsymbol{y}) \in D} \ell(\boldsymbol{y}, s_{\boldsymbol{\theta}}(\boldsymbol{x})) \quad + \quad \underbrace{|D|\alpha}_{\substack{\text{new reg.} \\ \text{weight}}} \; r(\theta)
\end{aligned}
$$

Generic optimization problem 1/2

Reweighting

Sometimes it is easier to absord the $\frac{1}{|D|}$ factor in the regularization weight:

$$\begin{aligned} & \arg\min_{\boldsymbol{\theta}} \quad \frac{1}{|D|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in D} \ell(\boldsymbol{y}, s_{\boldsymbol{\theta}}(\boldsymbol{x})) \quad + \quad \alpha r(\theta) \\ = \quad & \arg\min_{\boldsymbol{\theta}} \quad \sum_{(\boldsymbol{x},\boldsymbol{y})\in D} \ell(\boldsymbol{y}, s_{\boldsymbol{\theta}}(\boldsymbol{x})) \quad + \quad \underbrace{|D|\alpha}_{\substack{\text{new reg.}\\ \text{weight}}} \; r(\theta) \end{aligned}$$

Generic problem

Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ and $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be two convex functions.

$$\min_{\boldsymbol{u}\in \mathbf{dom}\, f} f(\boldsymbol{u}) \qquad \text{or} \qquad \min_{\boldsymbol{u}\in \mathbf{dom}\, f \cap \mathbf{dom}\, h} f(\boldsymbol{u}) + h(\boldsymbol{u}) \qquad \text{or} \qquad \min_{\boldsymbol{u}\in \mathbf{dom}\, f \cap \mathbf{dom}\, h} f(\boldsymbol{M}\boldsymbol{u}) + h(\boldsymbol{u})$$

Generic optimization problem 2/2

Fenchel duality

Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ and $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be two convex functions, and consider the following optimization problem:

$$\min_{\boldsymbol{u} \in \mathbb{R}^n} f(\boldsymbol{M}\boldsymbol{u}) + h(\boldsymbol{u})$$

The Fenchel dual is defined as:

$$\geq \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} -f^*(\boldsymbol{\lambda}) - h^*(-\boldsymbol{M}^\top \boldsymbol{\lambda})$$

Primal-dual relationship

To recover primal variable values from the dual variables, use the stationarity KKT condition of the primal problem.

Gradient descent

Generic optimization problem

Let's consider the following optimization problem:

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

where $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is a proper, closed and convex function.

Gradient descent algorithm

Assume f is differentiable everywhere in its domain. The gradient descent algorithm is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$\boldsymbol{u}^{(t+1)} = \boldsymbol{u}^{(t)} - \epsilon^{(t)} \nabla f(\boldsymbol{u}^{(t)})$$

- $\epsilon^{(t)}$ is the stepsize at time step t
- initial point $\boldsymbol{x}^{(0)} \in \mathbf{dom}\, f$ can be chosen randomly

Why does it work?

Theorem: Descent direction

Let $\boldsymbol{u}$ be a non optimal point, i.e. $\nabla f(\boldsymbol{u}) \neq 0$.
Then, there exist ϵ such that:

$$f(\boldsymbol{u} - \epsilon \nabla f(\boldsymbol{u})) < f(\boldsymbol{u})$$

We say that $-\nabla f(\boldsymbol{u})$ is a descent direction.
Proof: See [Boyd et al., 2004, Sections 9.2 and 9.3] and [Beck, Lemma 5.7]

Why does it work?

Theorem: Descent direction

Let $\boldsymbol{u}$ be a non optimal point, i.e. $\nabla f(\boldsymbol{u}) \neq 0$.
Then, there exist ϵ such that:

$$f(\boldsymbol{u} - \epsilon \nabla f(\boldsymbol{u})) < f(\boldsymbol{u})$$

We say that $-\nabla f(\boldsymbol{u})$ is a descent direction.
Proof: See [Boyd et al., 2004, Sections 9.2 and 9.3] and [Beck, Lemma 5.7]

Stepsize

How to choose the stepsize?

- line search: (approximately) search for the best stepsize, i.e. solve $\epsilon^{(t)} = \arg\min_{\epsilon > 0} f(\boldsymbol{x}^{(t)} - \epsilon \nabla f(\boldsymbol{x}^{(t)}))$
- constant stepsize
- diminishing stepsize: start with a given stepsize and decrease its value each t steps or according to the function evaluation / dev data evaluation

Stochastic gradient descent 1 / 3

Let's consider the following optimization problem:

$$\min_{\boldsymbol{u}\in\mathbb{R}^n} \quad \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{u})$$

where $\forall i \in \{1...n\}, f_i : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is a set of proper closed convex functions, we assume the intersection of their domain is a non-empty convex set.

In stochastic gradient descent, at each step the gradient is approximated using a subset of the functions f_i:

$$\boldsymbol{u}^{(t+1)} = \boldsymbol{u}^{(t)} - \frac{\epsilon^{(t)}}{|\mathrm{I}(t)|}\sum_{i\in\mathrm{I}(t)} \nabla f_i(\boldsymbol{u})$$

where $\mathrm{I}(t) \subseteq \{1...n\}$ is the subset of indices used at step t.
$\implies$ the subset of should consist of uniformly sampled indices!

Stochastic gradient descent 2 / 3

Machine learning application

We call $\mathrm{I}(t)$ a mini-batch and it consists of a subset of the training data.

$$\min_{\boldsymbol{\theta}} \quad \underbrace{\frac{1}{|D|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in D} \ell(\boldsymbol{y}, s_{\boldsymbol{\theta}}(\boldsymbol{x}))}_{\substack{\text{Approximate this term} \\ \text{using a subset of datapoints}}} \quad + \quad \alpha r(\theta)$$

Two approaches

- Sampling with replacement: at each step, randomly choose a subset of datapoints
- Sampling without replacement: optimization is based on a sequence of epochs
  - randomly choose of subset of datapoints that you did not see in the current epoch yet
  - an epoch is over when you saw all datapoints

$\Longrightarrow$ Sampling without replacement is standard in ML

Stochastic gradient descent 3 / 3

```
# Loop over epoch
for epoch in range(num_epochs):
    random.shuffle(training_data)

    # Loop over minibatches
    for i in range(0, len(training_data), minibatch_size):
        minibatch = training_data[i : i + minibatch_size]

        optimization_step(minibatch)

    # Evaluate on dev data
    evaluate_on_dev()
```

Other tricks:

- Save the model that obtain the best results on dev
- Control stepsize thanks to dev results

Subgradient descent

Subgradient descent 1 / 3

Non-differentiable functions

- Hinge loss: $\ell(y, w) = \max(0, 1 - (2y - 1)w)$
- L1 regularization: $r(\boldsymbol{a}) = \sum_i |a_i|$

Subgradient descent 1 / 3

Non-differentiable functions

- Hinge loss: $\ell(y, w) = \max(0, 1 - (2y - 1)w)$
- L1 regularization: $r(\boldsymbol{a}) = \sum_i |a_i|$

Subgradient descent algorithm

Assume $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is proper, closed and convex, but non-differentiable.

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

The **SUB**gradient descent algorithm is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$\boldsymbol{u}^{(t+1)} = \boldsymbol{u}^{(t)} - \epsilon^{(t)} \boldsymbol{g}^{(t)}$$

where $\boldsymbol{g}^{(t)} \in \partial f(\boldsymbol{u}^{(t)})$ is a subgradient of f at $\boldsymbol{u}^{(t)}$.

Subgradient descent 2 / 3

The subgradient **is not** a descent direction.

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be the function $f(\boldsymbol{u}) = |u_1| + 2|u_2|$, $\epsilon > 0$ and

$$\boldsymbol{u} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \boldsymbol{g} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \in \partial f(\boldsymbol{u})$$

Subgradient descent 2 / 3

The subgradient **is not** a descent direction.

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be the function $f(\boldsymbol{u}) = |u_1| + 2|u_2|$, $\epsilon > 0$ and

$$\boldsymbol{u} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \boldsymbol{g} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \in \partial f(\boldsymbol{u})$$

$$\begin{aligned} f(\boldsymbol{u}) &= |u_1| + 2|u_2| \\ &= |1| + 2|0| \\ &= 1 \end{aligned}$$

Subgradient descent 2 / 3

The subgradient **is not** a descent direction.

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be the function $f(\boldsymbol{u}) = |u_1| + 2|u_2|$, $\epsilon > 0$ and

$$\boldsymbol{u} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \boldsymbol{g} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \in \partial f(\boldsymbol{u})$$

$$\begin{aligned} f(\boldsymbol{u}) &= |u_1| + 2|u_2| \\ &= |1| + 2|0| \\ &= 1 \end{aligned}$$

$$\begin{aligned} f(\boldsymbol{u} - \epsilon \boldsymbol{g}) &= |u_1 - \epsilon g_1| + 2|u_2 - \epsilon g_2| \\ &= |1 - \epsilon 1| + 2|0 - \epsilon 2| = |1 - \epsilon| + |4\epsilon| \end{aligned}$$

Note that $|a + b| \leq |a| + |b|$, therefore:

$$\geq |1 - \epsilon + 4\epsilon| = |1 + 3\epsilon|$$

Therefore we have $\forall \epsilon > 0 : f(\boldsymbol{u} - \epsilon \boldsymbol{g}) \geq |1 + 3\epsilon| > f(\boldsymbol{u})$.

Subgradient descent 2 / 3

Is this an issue?

- Subgradient is not a descent direction, but we can show that we still can get closer to optimal solutions
- Work in many setting
- Not so good for L1 regularization, may (will?) “zig-zag” around solutions with null parameters

Proximal method

Proximal method

Let

- $f : \mathbb{R}^n \to \mathbb{R}$ be a proper, closed, convex and **differentiable** function,
- $g : \mathbb{R}^n \to \mathbb{R}$ be a proper, closed, convex and **non-differentiable** function.

And consider a problem of the form:

$$\min_{\boldsymbol{u} \in \mathbb{R}^n} \quad f(\boldsymbol{u}) \quad + \quad h(\boldsymbol{u})$$

Proximal method

The proximal method is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$\boldsymbol{u}^{(t+1)} = \mathbf{prox}_h \left(\ \boldsymbol{u}^{(t)} - \epsilon^{(t)} \nabla f(\boldsymbol{u}^{(t)}) \ \right)$$

where $\mathbf{prox}_h$ is the proximal operator of h defined as:

$$\mathbf{prox}_h(\boldsymbol{u}) = \underset{\boldsymbol{u}' \in \mathbb{R}^d}{\arg\min} \frac{1}{2} \|\boldsymbol{u} - \boldsymbol{u}'\|_2^2 + \ h(\boldsymbol{u}')$$

Properties of the proximal operator 1/2

Additively separable functions

Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a proper closed convex function defined as follows:

$$f(\boldsymbol{u}) = \sum_i f_i(u_i).$$

Then, the proximal operator of f is defined as:

$$\mathbf{prox}_f(\boldsymbol{u}) = \begin{bmatrix} \mathbf{prox}_{f_1}(u_1) \\ \mathbf{prox}_{f_2}(u_1) \\ \dots \\ \mathbf{prox}_{f_n}(u_n) \end{bmatrix}$$

Properties of the proximal operator 2/2

Scaling

Let $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a proper closed convex function and f a function defined as:

$$f(\boldsymbol{u}) = \lambda h(\lambda^{-1}\boldsymbol{u})$$

with $\lambda > 0$. Then, the proximal operator of f is defined as:

$$\mathbf{prox}_f(\boldsymbol{u}) = \lambda\, \mathbf{prox}_{\lambda^{-1}g}(\lambda^{-1}\boldsymbol{u})$$

Application

Linear regression with L1 regularization:

$$\min_{\boldsymbol{a},b} \quad \underbrace{\frac{1}{|D|} \sum_{(\boldsymbol{x},y)\in D} \frac{1}{2}(y - \langle \boldsymbol{a}, \boldsymbol{x} \rangle - b)^2}_{\text{differentiable}} \quad + \quad \underbrace{\alpha \sum_i |a_i|}_{\substack{\text{non-differentiable} \\ \text{but separable!}}}$$

$\implies$ can be minimized via proximal method!

Application

Linear regression with L1 regularization:

$$\min_{\boldsymbol{a},b} \quad \underbrace{\frac{1}{|D|} \sum_{(\boldsymbol{x},y)\in D} \frac{1}{2}(y - \langle \boldsymbol{a}, \boldsymbol{x} \rangle - b)^2}_{\text{differentiable}} \quad + \quad \underbrace{\alpha \sum_i |a_i|}_{\substack{\text{non-differentiable} \\ \text{but separable!}}}$$

$\implies$ can be minimized via proximal method!

The soft-thresholding operator

Let $f : \mathbb{R} \to \mathbb{R}$ be the absolute value function, i.e. $f(u) = |u|$.
The proximal operator of αf with $\alpha > 0$ is defined as:

$$\mathbf{prox}_{\lambda f(u)} = \begin{cases} u - \alpha & \text{if } u > \alpha, \\ 0 & \text{if } u \in [-\alpha, \alpha], \\ u + \alpha & \text{if } u < \alpha \end{cases} \quad = \max(0, |u| - \alpha) \times \text{sign}(u)$$

Projected gradient descent 1 / 2

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function, S a convex set and consider the following optimization problem:

$$\min_{\boldsymbol{u} \in \mathbb{R}^d} \quad f(\boldsymbol{u}) \quad + \quad \delta_S(\boldsymbol{u}) \qquad \text{where} \qquad \delta_S(\boldsymbol{u}) = \begin{cases} 0 & \text{if } \boldsymbol{u} \in S, \\ +\infty & \text{otherwise.} \end{cases}$$

Projected gradient descent 1 / 2

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function, S a convex set and consider the following optimization problem:

$$\min_{\boldsymbol{u} \in \mathbb{R}^d} \quad f(\boldsymbol{u}) \quad + \quad \delta_S(\boldsymbol{u}) \qquad \text{where} \qquad \delta_S(\boldsymbol{u}) = \begin{cases} 0 & \text{if } \boldsymbol{u} \in S, \\ +\infty & \text{otherwise.} \end{cases}$$

Note that the proximal operator of the indicator function of S is the **projection function**:

$$\mathbf{proj}_{\delta_S}(\boldsymbol{u}) \quad = \quad \arg\min_{\boldsymbol{u}' \in \mathbb{R}^d} \|\boldsymbol{u} - \boldsymbol{u}'\| + \delta_S(\boldsymbol{u}') \quad = \quad \arg\min_{\boldsymbol{u}' \in S} \|\boldsymbol{u} - \boldsymbol{u}'\| \quad = \quad \mathbf{proj}_j(\boldsymbol{u})$$

Projected gradient descent 2 / 2

Let

- $f : \mathbb{R}^n \to \mathbb{R}$ be a proper, closed, convex and **differentiable** function,
- S be a convex set

And consider a problem of the form: $\quad \min_{\boldsymbol{u} \in S} \quad f(\boldsymbol{u})$

Projected gradient descent 2 / 2

Let

- $f : \mathbb{R}^n \to \mathbb{R}$ be a proper, closed, convex and **differentiable** function,
- S be a convex set

And consider a problem of the form: $\min_{\boldsymbol{u} \in S} \quad f(\boldsymbol{u})$

Projected gradient descent

The projected gradient descent is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$\boldsymbol{u}^{(t+1)} = \mathbf{proj}_S \left(\, \boldsymbol{u}^{(t)} - \epsilon^{(t)} \nabla f(\boldsymbol{u}^{(t)}) \, \right)$$

It is a special case of the proximal method.

Non-differentiable objective

If f is non-differentiable, a similar approach is called the projected **SUB**gradient descent algorithm.

Coordinate descent

Coordinate descent

Motivations

All these algorithms require a stepsize, which may be difficult to tune.
Is there any method that does not depend on a stepsize?

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a proper, closed, convex and differentiable function. Assume a problem of the form:

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

The coordinate descent algorithm is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$u_1^{(t+1)} \quad \in \quad \arg\min_{u_1 \in \mathbb{R}} f(\ [\ u_1, u_2^{(t)}, u_3^{(t)}, ..., u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ )$$

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a proper, closed, convex and differentiable function. Assume a problem of the form:

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

The coordinate descent algorithm is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$\begin{aligned}
u_1^{(t+1)} &\in \arg\min_{u_1 \in \mathbb{R}} f(\ [\ u_1, u_2^{(t)}, u_3^{(t)}, ..., u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ ) \\
u_2^{(t+1)} &\in \arg\min_{u_2 \in \mathbb{R}} f(\ [\ u_1^{(t+1)}, u_2, u_3^{(t)}, ..., u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ )
\end{aligned}$$

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a proper, closed, convex and differentiable function. Assume a problem of the form:

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

The coordinate descent algorithm is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$\begin{aligned}
u_1^{(t+1)} &\in \operatorname*{arg\,min}_{u_1 \in \mathbb{R}} f(\ [\ u_1, u_2^{(t)}, u_3^{(t)}, \dots, u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ ) \\
u_2^{(t+1)} &\in \operatorname*{arg\,min}_{u_2 \in \mathbb{R}} f(\ [\ u_1^{(t+1)}, u_2, u_3^{(t)}, \dots, u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ ) \\
u_3^{(t+1)} &\in \operatorname*{arg\,min}_{u_3 \in \mathbb{R}} f(\ [\ u_1^{(t+1)}, u_2^{(t+1)}, u_3, \dots, u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ )
\end{aligned}$$

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a proper, closed, convex and differentiable function. Assume a problem of the form:

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

The coordinate descent algorithm is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$\begin{aligned}
u_1^{(t+1)} &\in \arg\min_{u_1 \in \mathbb{R}} f(\ [\ u_1, u_2^{(t)}, u_3^{(t)}, ..., u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ ) \\
u_2^{(t+1)} &\in \arg\min_{u_2 \in \mathbb{R}} f(\ [\ u_1^{(t+1)}, u_2, u_3^{(t)}, ..., u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ ) \\
u_3^{(t+1)} &\in \arg\min_{u_3 \in \mathbb{R}} f(\ [\ u_1^{(t+1)}, u_2^{(t+1)}, u_3, ..., u_{n-1}^{(t)}, u_n^{(t)}\ ]^\top\ ) \\
&\dots \\
u_{n-1}^{(t+1)} &\in \arg\min_{u_{n-1} \in \mathbb{R}} f(\ [\ u_1^{(t+1)}, u_2^{(t+1)}, u_3^{(t+1)}, ..., u_{n-1}, u_n^{(t)}\ ]^\top\ ) \\
u_n^{(t+1)} &\in \arg\min_{u_n \in \mathbb{R}} f(\ [\ u_1^{(t+1)}, u_2^{(t+1)}, u_3^{(t+1)}, ..., u_{n-1}^{(t+1)}, u_n\ ]^\top\ )
\end{aligned}$$

Or any other order, as long as you directly use the new value for the next coordinate.

Coordinate descent for non smooth objective

Let $f : \mathbb{R}^d \to \mathbb{R} \cup \{\infty\}$ and $h : \mathbb{R}^d \to \mathbb{R} \cup \{\infty\}$ be proper, closed and convex functions such that:

- f is differentiable
- h is not differentiable but additively separable,
  i.e. can be writted as $h(\boldsymbol{u}) = \sum_i h_i(u_i)$

Then, the following problem:

$$\min_{\boldsymbol{u} \in \mathbb{R}^d} \quad f(\boldsymbol{u}) \quad + \quad h(\boldsymbol{u})$$

can be solved via coordinate descent.

WARNING: the condition the h is additively separable is very important!

SVM dual optimization

SVM dual objective

From last week course, remember that the SVM dual objective is devined as

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & -\sum_{i=1}^{n} \boldsymbol{\lambda}_i - \frac{1}{2} \boldsymbol{\lambda}^\top \boldsymbol{Y} \boldsymbol{X} \boldsymbol{X}^\top \boldsymbol{Y} \boldsymbol{\lambda} \\ \text{s.t.} \quad & -1 \leq \boldsymbol{\lambda}_i \leq 0 \quad \forall 1 \leq i \leq n \end{aligned}$$

where

- $\boldsymbol{X} \in \mathbb{R}^{n \times d}$: matrix where each row consists of a training point, i.e. $X_{i,j} = x_j^{(i)}$
- $\boldsymbol{Y} \in R^{n \times n}$: diagonal matrix containing labels, i.e. $Y_{i,i} = y^{(i)}$ and $\forall i \neq j : Y_{i,j} = 0$.

Primal-dual relationship

To get back the primal variable values from the dual variables:

$$\boldsymbol{a} = -\boldsymbol{X}^\top \boldsymbol{Y} \boldsymbol{\lambda}$$

Project gradient ascent 1/2

Objective function:

$$f(\lambda) = -\sum_{i=1}^{n} \lambda_i - \frac{1}{2}\lambda^\top \boldsymbol{YXX}^\top \boldsymbol{Y}\lambda$$

Project gradient ascent step

$$\lambda^{(t+1)} = \mathbf{proj}_{[-1,0]^n}\left[\lambda^{(t)} + \epsilon \nabla f(\lambda^{(t)})\right]$$

where **proj** is the projection operator.

Projection

Project into the convex set $[-1,0]^n$ => clip each coordinate to $[-1,0]$, i.e.:

$$\mathrm{Clip}_{[0,1]}(w) = \begin{cases} -1 & \text{if } w \leq -1 \\ 0 & \text{if } w \geq 0 \\ w & \text{otherwise} \end{cases}$$

Project gradient ascent 2/2

Objective function:

$$f(\lambda) = -\sum_{i=1}^{n} \lambda_i - \frac{1}{2}\lambda^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\lambda$$

Gradient

$$\begin{aligned}\nabla f(\lambda^{(t)}) &= -1 - \frac{1}{2}\left(\boldsymbol{Y}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y} + \left(\boldsymbol{Y}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\right)^\top\right)\lambda^{(t)} \\ &= -1 - \boldsymbol{Y}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\lambda^{(t)}\end{aligned}$$

See Equation 97 in the Matrix Cookbook.

Quadratic program

Box constrained quadratic problem

$$\begin{aligned} \max_{\lambda} \quad & \frac{1}{2}\lambda^\top \boldsymbol{Q}\, \lambda + \boldsymbol{b}^\top \lambda \\ \text{s.t.} \quad & l \le \lambda \le u \end{aligned}$$

- $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ symmetric matrix of weights associated with quadratic term
- $\boldsymbol{b} \in \mathbb{R}^n$ weights associated with linear term

SVM dual problem

- $\boldsymbol{Q} = -\boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}$
- $\boldsymbol{b} = -1$
- $l = -1$ and $u = 0$.

Coordinate ascent

$$
\begin{aligned}
\max_{\lambda} \quad & \frac{1}{2}\lambda^\top \boldsymbol{Q}\,\lambda + \boldsymbol{b}^\top \lambda \\
\text{s.t.} \quad & l \leq \lambda \leq b
\end{aligned}
$$

Main idea

Iteratively solve the problem wrt to one single element (coordinate) of λ only:

- For a given index k, solve $\frac{\partial}{\partial \lambda_k} f(\lambda) = 0$
- If the solution does not satisfy the constraints, clip it.

Notes

- when you solve for one coordinate, you immediately use the new coordinate solution for the next coordinate!
- coordinates may be visited in any order.

Coordinate ascent step 1/2

Rewrite the objective as:

$$\begin{aligned} f(\lambda) &= \frac{1}{2}\lambda^\top \boldsymbol{Q}\lambda + \boldsymbol{b}^\top \lambda \\ &= \frac{1}{2}\sum_{i=1}^{n}\lambda_i \sum_{j=1}^{n}\lambda_j Q_{j,i} + \sum_{i=1}^{n} b_i\lambda_i \\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j Q_{j,i} + \sum_{i=1}^{n} b_i\lambda_i \\ &= \frac{1}{2}\sum_{i\neq j}\lambda_i\lambda_j Q_{j,i} + \frac{1}{2}\sum_{i}\lambda_i^2 Q_{i,i} + \sum_{i=1}^{n} b_i\lambda_i \end{aligned}$$

whose partial derivative is:

$$\frac{\partial}{\partial\lambda_k} f(\lambda) = \sum_{i\neq k}\lambda_i Q_{k,i} + \lambda_k Q_{k,k} + b_k$$

Coordinate ascent step 2/2

Partial derivative of the objective:

$$\frac{\partial}{\partial \lambda_k} f(\lambda) = \sum_{i \neq k} \lambda_i Q_{k,i} + \lambda_k Q_{k,k} + b_k$$

Solving for the derivate equals to zero gives:

$$\sum_{i \neq k} \lambda_i Q_{k,i} + \lambda_k Q_{k,k} + b_k = 0$$

$$\lambda_k = \frac{-b_k - \sum_{i \neq k} \lambda_i Q_{k,i}}{Q_{k,k}}$$

Therefore, solving for coordinate k is simply setting:

$$\lambda_k = \mathrm{Clip}_{[l,b]} \left[\frac{-b_k - \sum_{i \neq k} \lambda_i Q_{k,i}}{Q_{k,k}} \right]$$