Statistical Computing (36-350)

Lecture 19: Optimization III: Constrained and Stochastic Optimization

Cosma Shalizi

30 October 2013

Agenda

- Constraints and penalties
- Stochastic optimization methods

READING (big picture): Francis Spufford, *Red Plenty*
OPTIONAL READING (big picture): Herbert Simon, *The Sciences of the Artificial*
OPTIONAL READING (close up): Bottou and Bosquet, “The Tradeoffs of Large Scale Learning”

Maximizing a multinomial likelihood

I roll dice n times; $n_1, \ldots n_6$ count the outcomes
Likelihood and log-likelihood:

$$
\begin{aligned}
L(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) &= \frac{n!}{n_1!n_2!n_3!n_4!n_5!n_6!}\prod_{i=1}^{6}\theta_i^{n_i} \\
\ell(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) &= \log\frac{n!}{n_1!n_2!n_3!n_4!n_5!n_6!} + \sum_{i=1}^{6} n_i \log\theta_i
\end{aligned}
$$

Optimize by taking the derivative and setting to zero:

$$
\begin{aligned}
\frac{\partial \ell}{\partial \theta_1} &= \frac{n_1}{\theta_1} = 0 \\
\therefore \theta_1 &= \infty
\end{aligned}
$$

or $n_1 = 0$

We forgot that $\sum_{i=1}^{6} \theta_i = 1$
We could use the constraint to eliminate one of the variables

$$\theta_6 = 1 - \sum_{i=1}^{5} \theta_i$$

Then solve the equations

$$\frac{\partial \ell}{\partial \theta_i} = \frac{n_1}{\theta_i} - \frac{n_6}{1 - \sum_{j=1}^{5} \theta_j} = 0$$

BUT eliminating a variable with the constraint is usually messy

Lagrange Multipliers

$$g(\theta) = c \Leftrightarrow g(\theta) - c = 0$$

Lagrangian:

$$\mathscr{L}(\theta, \lambda) = f(\theta) - \lambda(g(\theta) - c)$$

$= f$ when the constraint is satisfied

Lagrange Multipliers

$$g(\theta) = c \Leftrightarrow g(\theta) - c = 0$$

Lagrangian:

$$\mathcal{L}(\theta, \lambda) = f(\theta) - \lambda(g(\theta) - c)$$

$= f$ when the constraint is satisfied
Now do *unconstrained* minimization over θ and λ:

$$\begin{aligned} \nabla_\theta \mathcal{L}|_{\theta^*,\lambda^*} &= \nabla f(\theta^*) - \lambda^* \nabla g(\theta^*) = 0 \\ \left.\frac{\partial \mathcal{L}}{\partial \lambda}\right|_{\theta^*,\lambda^*} &= g(\theta^*) - c = 0 \end{aligned}$$

optimizing **Lagrange multiplier** λ enforces constraint
More constraints, more multipliers

Try the dice again:

$$\begin{aligned}
\mathcal{L} &= \log \frac{n!}{\prod_i n_i!} + \sum_{i=1}^{6} n_i \log(\theta_i) - \lambda\left(\sum_{i=1}^{6} \theta_i - 1\right) \\
\left.\frac{\partial \mathcal{L}}{\partial \theta_i}\right|_{\theta_i=\theta_i^*} &= \frac{n_i}{\theta_i^*} - \lambda^* = 0 \\
\frac{n_i}{\lambda^*} &= \theta_i^* \\
\sum_{i=1}^{6} \frac{n_i}{\lambda^*} &= \sum_{i=1}^{6} \theta_i^* = 1 \\
\lambda^* &= \sum_{i=1}^{6} n_i \Rightarrow \theta_i^* = \frac{n_i}{\sum_{i=1}^{6} n_i}
\end{aligned}$$

Thinking About the Lagrange Multipliers

Constrained minimum value is generally higher than the unconstrained

Changing the constraint level c changes $\theta^*, f(\theta^*)$

Thinking About the Lagrange Multipliers

Constrained minimum value is generally higher than the unconstrained

Changing the constraint level c changes $\theta^*, f(\theta^*)$

$$\begin{aligned}
\frac{\partial f(\theta^*)}{\partial c} &= \frac{\partial \mathscr{L}(\theta^*, \lambda^*)}{\partial c} \\
&= [\nabla f(\theta^*) - \lambda^* \nabla g(\theta^*)] \frac{\partial \theta^*}{\partial c} - [g(\theta^*) - c] \frac{\partial \lambda^*}{\partial c} + \lambda^* = \lambda^*
\end{aligned}$$

Thinking About the Lagrange Multipliers

Constrained minimum value is generally higher than the unconstrained
Changing the constraint level c changes $\theta^*, f(\theta^*)$

$$\begin{aligned}\frac{\partial f(\theta^*)}{\partial c} &= \frac{\partial \mathcal{L}(\theta^*, \lambda^*)}{\partial c} \\ &= \left[\nabla f(\theta^*) - \lambda^* \nabla g(\theta^*)\right]\frac{\partial \theta^*}{\partial c} - \left[g(\theta^*) - c\right]\frac{\partial \lambda^*}{\partial c} + \lambda^* = \lambda^*\end{aligned}$$

λ^* = Rate of change in optimal value as the constraint is relaxed
λ^* = "Shadow price": How much would you pay for minute change in the level of the constraint

Inequality Constraints

What about an *inequality* constraint?

$$h(\theta) \leq d \Leftrightarrow h(\theta) - d \leq 0$$

The region where the constraint is satisfied is the **feasible set**

Inequality Constraints

What about an *inequality* constraint?

$$h(\theta) \leq d \Leftrightarrow h(\theta) - d \leq 0$$

The region where the constraint is satisfied is the **feasible set**
Roughly two cases:

1. Unconstrained optimum is inside the feasible set $\Rightarrow$ constraint is **inactive**

Inequality Constraints

What about an *inequality* constraint?

$$h(\theta) \leq d \Leftrightarrow h(\theta) - d \leq 0$$

The region where the constraint is satisfied is the **feasible set**
Roughly two cases:

1. Unconstrained optimum is inside the feasible set $\Rightarrow$ constraint is **inactive**
2. Optimum is outside feasible set; constraint **is active, binds** or **bites**; *constrained* optimum is usually on the boundary

Inequality Constraints

What about an *inequality* constraint?

$$h(\theta) \leq d \Leftrightarrow h(\theta) - d \leq 0$$

The region where the constraint is satisfied is the **feasible set**
Roughly two cases:

1. Unconstrained optimum is inside the feasible set $\Rightarrow$ constraint is **inactive**
2. Optimum is outside feasible set; constraint **is active, binds** or **bites**; *constrained* optimum is usually on the boundary

Add a Lagrange multiplier; $\lambda \neq 0 \Leftrightarrow$ constraint binds

Mathematical Programming

Older than computer programming...

Optimize $f(\theta)$ subject to $g(\theta) = c$ and $h(\theta) \leq d$

"Give us the best deal on f, keeping in mind that we've only got d to spend, and the books have to balance"

Linear programming (Kantorovich, 1938)

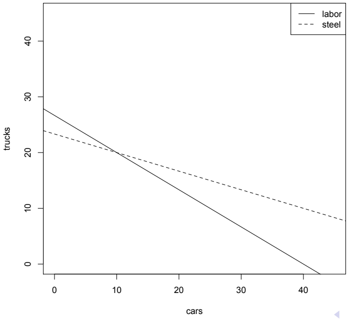
- f, h both linear in θ
- θ^* always at a corner of the feasible set

Back to the Factory

Constraints:

$$\begin{aligned} 40(\text{cars}) + 60(\text{trucks}) &\leq 1600 \\ 1(\text{cars}) + 3(\text{trucks}) &\leq 70 \end{aligned}$$

Revenue: \$13k/car, \$27k/truck
The feasible region:

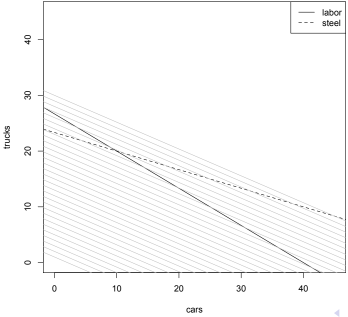


Back to the Factory

Constraints:

$$\begin{aligned} 40(\text{cars}) + 60(\text{trucks}) &\leq 1600 \\ 1(\text{cars}) + 3(\text{trucks}) &\leq 70 \end{aligned}$$

Revenue: \$13k/car, \$27k/truck
The feasible region:



Barrier Methods

(a.k.a. "interior point", "central path", etc.)

Barrier Methods

(a.k.a. "interior point", "central path", etc.)

Having constraints switch on and off abruptly is annoying

especially with gradient methods

Barrier Methods

(a.k.a. "interior point", "central path", etc.)

Having constraints switch on and off abruptly is annoying

especially with gradient methods

Fix $\mu > 0$ and try minimizing

$$f(\theta) - \mu \log(d - h(\theta))$$

"pushes away" from the barrier — more and more weakly as $\mu \rightarrow 0$

Barrier Methods

(a.k.a. "interior point", "central path", etc.)

Having constraints switch on and off abruptly is annoying

especially with gradient methods

Fix $\mu > 0$ and try minimizing

$$f(\theta) - \mu \log(d - h(\theta))$$

"pushes away" from the barrier — more and more weakly as $\mu \to 0$

1. Initial θ in feasible set, initial μ
2. While ((not too tired) and (making adequate progress))
    1. Minimize $f(\theta) - \mu \log(d - h(\theta))$
    2. Reduce μ
3. Return final θ

Constraints vs. Penalties

$$\operatorname*{argmin}_{\theta : h(\theta) \leq d} f(\theta) \Leftrightarrow \operatorname*{argmin}_{\theta, \lambda} f(\theta) - \lambda(h(\theta) - d)$$

d doesn't matter for doing the second minimization over θ

Constraints vs. Penalties

$$\operatorname*{argmin}_{\theta : h(\theta) \leq d} f(\theta) \Leftrightarrow \operatorname*{argmin}_{\theta, \lambda} f(\theta) - \lambda(h(\theta) - d)$$

d doesn't matter for doing the second minimization over θ

Constrained optimization	$\Leftrightarrow$	Penalized optimization
Constraint level d	$\Leftrightarrow$	Penalty factor λ

Statistical Applications of Penalization

Minimize MSE of linear function $\beta \cdot x$: ordinary least squares regression

Statistical Applications of Penalization

Minimize MSE of linear function $\beta \cdot x$: ordinary least squares regression

penalty on length of coefficient vector, $\sum \beta_j^2$: ridge regression

stabilizes estimate when data are collinear, $p > n$, or just noisy

Statistical Applications of Penalization

Minimize MSE of linear function $\beta \cdot x$: ordinary least squares regression

penalty on length of coefficient vector, $\sum \beta_j^2$: ridge regression

stabilizes estimate when data are collinear, $p > n$, or just noisy

penalty on sum of coefficients, $\sum |\beta_j|$: lasso

stability + drive small coefficients to 0 (“sparsity”)

Statistical Applications of Penalization

Minimize MSE of linear function $\beta \cdot x$: ordinary least squares regression

penalty on length of coefficient vector, $\sum \beta_j^2$: ridge regression

stabilizes estimate when data are collinear, $p > n$, or just noisy

penalty on sum of coefficients, $\sum |\beta_j|$: lasso

stability + drive small coefficients to 0 ("sparsity")

Minimize MSE of function + penalty on curvature : spline

fit smooth regressions w/o assuming specific form

Statistical Applications of Penalization

Minimize MSE of linear function $\beta \cdot x$: ordinary least squares regression

penalty on length of coefficient vector, $\sum \beta_j^2$: ridge regression

stabilizes estimate when data are collinear, $p > n$, or just noisy

penalty on sum of coefficients, $\sum |\beta_j|$: lasso

stability + drive small coefficients to 0 ("sparsity")

Minimize MSE of function + penalty on curvature : spline

fit smooth regressions w/o assuming specific form

Smoothing over time, space, other relations

e.g., social or genetic ties

Statistical Applications of Penalization

Minimize MSE of linear function $\beta \cdot x$: ordinary least squares regression

penalty on length of coefficient vector, $\sum \beta_j^2$: ridge regression

stabilizes estimate when data are collinear, $p > n$, or just noisy

penalty on sum of coefficients, $\sum |\beta_j|$: lasso

stability + drive small coefficients to 0 ("sparsity")

Minimize MSE of function + penalty on curvature : spline

fit smooth regressions w/o assuming specific form

Smoothing over time, space, other relations

e.g., social or genetic ties

Usually decide on penalty factor/constraint level by trying to predict out of sample

R implementation

In penalty form, just chose λ and modify your objective function

R implementation

In penalty form, just chose λ and modify your objective function
`constrOptim` implements the barrier method
Try this:

```
factory <- matrix(c(40,1,60,3),nrow=2,
  dimnames=list(c("labor","steel"),c("car","truck")))
available <- c(1600,70); names(available) <- rownames(factory)
prices <- c(car=13,truck=27)
revenue <- function(output) { return(-output %*% prices) }
plan <- constrOptim(theta=c(5,5),f=revenue,grad=NULL,
  ui=-factory,ci=-available,method="Nelder-Mead")
plan$par
```

only works with constraints like $\mathbf{u}\theta \geq c$, so minus signs

Problems with Big Data

Typical statistical objective function, mean-squared error:

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - m(x_i, \theta))^2$$

Problems with Big Data

Typical statistical objective function, mean-squared error:

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - m(x_i, \theta))^2$$

Getting a value of f is $O(n)$, ∇f is $O(np)$, $\mathbf{H}$ is $O(np^2)$

worse still if m slows down with n

Problems with Big Data

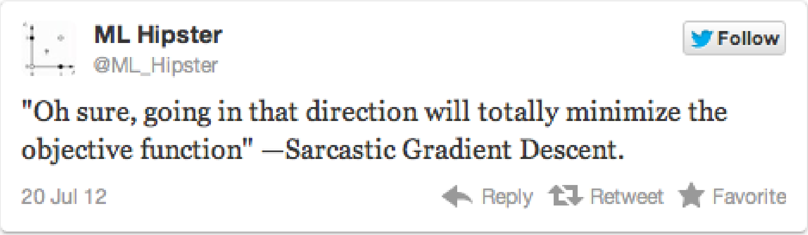
Typical statistical objective function, mean-squared error:

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - m(x_i, \theta))^2$$

Getting a value of f is $O(n)$, ∇f is $O(np)$, $\mathbf{H}$ is $O(np^2)$

worse still if m slows down with n

Not bad when $n = 100$ or even $n = 10^4$, but if $n = 10^9$ or $n = 10^{12}$ we don't even know which way to move


ML Hipster
@ML_Hipster
Follow
"Oh sure, going in that direction will totally minimize the objective function" —Sarcastic Gradient Descent.
20 Jul 12
Reply
Retweet
Favorite

Sampling, the Alternative to Sarcastic Gradient Descent

Pick *one* data point I at random (uniform on $1:n$)
Loss there, $(y_I - m(x_I, \theta))^2$, is random, but

$$\mathbb{E}\left[(y_I - m(x_I, \theta))^2\right] = f(\theta)$$

Sampling, the Alternative to Sarcastic Gradient Descent

Pick *one* data point I at random (uniform on $1:n$)
Loss there, $(y_I - m(x_I, \theta))^2$, is random, but

$$\mathbb{E}\left[(y_I - m(x_I, \theta))^2\right] = f(\theta)$$

Generally, if $f(\theta) = n^{-1} \sum_{i=1}^{n} f_i(\theta)$ and f_i are well-behaved,

$$\begin{aligned} \mathbb{E}\left[f_I(\theta)\right] &= f(\theta) \\ \mathbb{E}\left[\nabla f_I(\theta)\right] &= \nabla f(\theta) \\ \mathbb{E}\left[\nabla^2 f_I(\theta)\right] &= \mathbf{H}(\theta) \end{aligned}$$

Sampling, the Alternative to Sarcastic Gradient Descent

Pick *one* data point I at random (uniform on $1:n$)
Loss there, $(y_I - m(x_I, \theta))^2$, is random, but

$$\mathbb{E}\left[(y_I - m(x_I, \theta))^2\right] = f(\theta)$$

Generally, if $f(\theta) = n^{-1}\sum_{i=1}^{n} f_i(\theta)$ and f_i are well-behaved,

$$\begin{aligned}
\mathbb{E}\left[f_I(\theta)\right] &= f(\theta) \\
\mathbb{E}\left[\nabla f_I(\theta)\right] &= \nabla f(\theta) \\
\mathbb{E}\left[\nabla^2 f_I(\theta)\right] &= \mathbf{H}(\theta)
\end{aligned}$$

$\therefore$ Don't optimize with all the data, optimize with random samples

Stochastic Gradient Descent

Draw lots of one-point samples, let their noise cancel out:

1. Start with initial guess θ, learning rate η
2. While ((not too tired) and (making adequate progress))
    1. At t^{th} iteration, pick random I uniformly
    2. Set $\theta \leftarrow \theta - t^{-1}\eta\nabla f_I(\theta)$
3. Return final θ

Stochastic Gradient Descent

Draw lots of one-point samples, let their noise cancel out:

1. Start with initial guess θ, learning rate η
2. While ((not too tired) and (making adequate progress))
    1. At t^{th} iteration, pick random I uniformly
    2. Set $\theta \leftarrow \theta - t^{-1}\eta\nabla f_I(\theta)$
3. Return final θ

Shrinking step-size by $1/t$ ensures noise in each gradient dies down

Stochastic Gradient Descent

Draw lots of one-point samples, let their noise cancel out:

1. Start with initial guess θ, learning rate η
2. While ((not too tired) and (making adequate progress))
    1. At t^{th} iteration, pick random I uniformly
    2. Set $\theta \leftarrow \theta - t^{-1}\eta\nabla f_I(\theta)$
3. Return final θ

Shrinking step-size by $1/t$ ensures noise in each gradient dies down

(Variants: put points in some random order, only check progress after going over each point once, adjust $1/t$ rate, average a couple of random data points, etc.)

```
stoch.grad.descent <- function(f,theta,df,max.iter=1e6,rate=1e-6) {
  stopifnot(require(numDeriv))
  for (t in 1:max.iter) {
    g <- stoch.grad(f,theta,df)
    theta <- theta - (rate/t)*g
  }
  return(x)
}

stoch.grad <- function(f,theta,df) {
  i <- sample(1:nrow(df),size=1)
  noisy.f <- function(theta) { return(f(theta, data=df[i,])) }
  stoch.grad <- grad(noisy.f,theta)
  return(stoch.grad)
}
```

Stochastic Newton's Method

a.k.a. 2nd order stochastic gradient descent

1. Start with initial guess θ
2. While ((not too tired) and (making adequate progress))
    1. At t^{th} iteration, pick uniformly-random I
    2. $\theta \leftarrow \theta - t^{-1}\mathbf{H}_I^{-1}(\theta)\nabla f_I(\theta)$
3. Return final θ

+ all the Newton-ish tricks to avoid having to recompute the Hessian

Stochastic Gradient Methods

Pros:

- Each iteration is fast (and constant in n)
- Never need to hold all data in memory
- Does converge eventually

Stochastic Gradient Methods

Pros:

- Each iteration is fast (and constant in n)
- Never need to hold all data in memory
- Does converge eventually

Cons:

- Noise *does* reduce precision — more iterations to get within ϵ of optimum than non-stochastic GD or Newton

Stochastic Gradient Methods

Pros:

- Each iteration is fast (and constant in n)
- Never need to hold all data in memory
- Does converge eventually

Cons:

- Noise *does* reduce precision — more iterations to get within ϵ of optimum than non-stochastic GD or Newton

Often low computational cost to get within *statistical* error of the optimum

Simulated Annealing

Use Metropolis to sample from a density $\propto e^{-f(\theta)/T}$
Samples will tend to be near small values of f
Keep lowering T as we go along ("cooling", "annealing")

Simulated Annealing

Use Metropolis to sample from a density $\propto e^{-f(\theta)/T}$
Samples will tend to be near small values of f
Keep lowering T as we go along ("cooling", "annealing")

1. Set initial θ, $T > 0$
2. While ((not too tired) and (making adequate progress))
    1. Proposal: $Z \leftarrow r(\cdot|\theta)$ (e.g., Gaussian noise)
    2. Draw $U \sim \mathrm{Unif}(0, 1)$
    3. Acceptance: If $U < e^{-\frac{f(Z)-f(\theta)}{T}}$ then $\theta \leftarrow Z$
    4. Reduce T a little
3. Return final θ

Always moves to lower values of f, sometimes moves to higher

Simulated Annealing

Use Metropolis to sample from a density $\propto e^{-f(\theta)/T}$
Samples will tend to be near small values of f
Keep lowering T as we go along ("cooling", "annealing")

1. Set initial θ, $T > 0$
2. While ((not too tired) and (making adequate progress))
    1. Proposal: $Z \leftarrow r(\cdot|\theta)$ (e.g., Gaussian noise)
    2. Draw $U \sim \mathrm{Unif}(0, 1)$
    3. Acceptance: If $U < e^{-\frac{f(Z)-f(\theta)}{T}}$ then $\theta \leftarrow Z$
    4. Reduce T a little
3. Return final θ

Always moves to lower values of f, sometimes moves to higher
No derivatives, works for discrete problems, few guarantees

Summary

- Constraints are usually part of optimization
    - Constrained optimum generally at the boundary of feasible set
    - Lagrange multipliers turn constrained problems into unconstrained ones
    - Multipliers are prices: trade-off between tightening constraint and worsening optimal value
- Stochastic optimization methods use probability in the search
    - Stochastic gradient descent samples the data; gives up precision for speed
    - Simulated annealing randomly moves against the objective function; escapes local minima