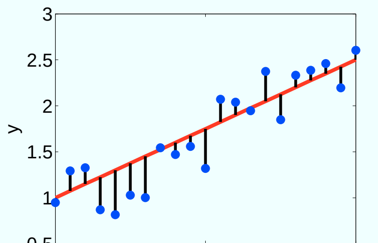
Linear regression - the model

We want to model

$$y = \beta_0 + \sum_{j=1}^{p} x_j \beta_j + \epsilon$$

where ϵ is Gaussian noise with mean μ and variance σ^2

We assume that y can be explained as a linear combination of p features in x, and this linear combination was corrupted by Gaussian noise.



Linear regression - likelihood function

We start by writing out a probability distribution

$$p(y|x, \beta_0, \beta, \sigma) = \frac{1}{\sqrt{(2\pi)\sigma^2}} \exp\left\{ -\frac{(y - \beta_0 - x'\beta)^2}{2\sigma^2} \right\}$$

Suppose we gathered n instances of x and y.

We denote the i^{th} instance as (x_i, y_i), then we can write down a likelihood function

$$L(\beta_0, \beta, \sigma) = \prod_{i=1}^{n} p(y_i|x_i, \beta_0, \beta, \sigma)$$

A likelihood function enables us to compare parameters in their suitability in explaining data. Always in the context of a model.

Log likelihood

Log likelihood is simpler to work with than plain likelihood

$$\begin{aligned}\log L(\beta_0, \beta, \sigma) &= \sum_i \log p(y_i | x_i, \beta_0, \beta, \sigma) \\ &= -n/2 \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_i (y_i - \beta_0 - x_i' \beta)^2\end{aligned}$$

Log is a monotone function; maximum of $\log(f(x))$ is achieved at the same point as maximum of $f(x)$.

Maximum likelihood estimate

Finding a maximum likelihood estimate amounts to fitting a probabilistic model.

In the case of linear regression there are a couple of ways of doing this. We are going with a super easy one: coordinate ascent.

Coordinate ascent

The idea is that we update each of our parameters, in turn, so as to maximize likelihood.

We stop when the log likelihood stops changing.

Deriving a coordinate ascent algorithm

Compute the partial derivative of log-likelihood with respect to each parameter and equate it to 0, solve for updates.

$$\frac{\partial \log L(\beta_0, \beta_1, \ldots, \beta_p)}{\partial \beta_i} = 0$$

In our case

$$\begin{aligned}
\beta_0 &= \frac{1}{n} \sum_i (y_i - x_i' \beta) \\
\beta_j &= \frac{1}{\sum_i x_{i,j}^2} \sum_i \underbrace{\left(y_i - \beta_0 - \sum_{k \neq j} x_{i,k} \beta_k\right)}_{y_i^{(-j)}} x_{i,j}
\end{aligned}$$

Coordinate ascent with normalized predictors

If we assume that each feature is normalized

$$\sum_{i} x_{i,k}^2 = 1 \quad \text{and} \quad \sum_{i} x_{i,k} = 0$$

then we can simplify[1]

$$\begin{aligned} \beta_0 &= \frac{1}{n}\sum_i y_i \\ \beta_j &= \sum_i y_i^{(-j)} x_{i,j} \end{aligned}$$

So we really only have to iterate through updating $\beta_1, \beta_2, \ldots \beta_k, \beta_1, \ldots$ until the likelihood no longer improves.

[1] reminder: $y_i^{(-j)} = y_i - \beta_0 - \sum_{k \neq j} x_{i,k}\beta_k$

More difficulties

Having more predictors than data instances ($p > n$) *guarantees* multicolinearity.

It is clear that we have to break ties between all of these solutions that have equal likelihood.

This is a common issue: an ill-posed problem

Regularization/prior

Standard solution to ill-posedness of problems is to use some sort of regularization.

Sum of squares of parameters - Tikhonov regularization

Change our objective

$$\underbrace{-n/2 \log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\sum_i (y_i - \beta_0 - x_i'\beta)^2}_{\text{log likelihood}} - \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{\textit{regularization}}$$

Regularization/prior

Feels like a trick: how would we come up with Tikhonov regularization?

Regularization/prior

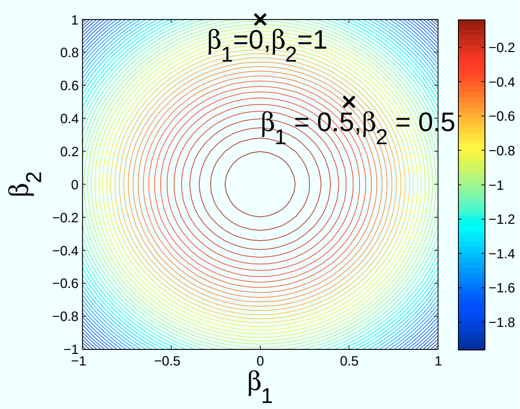
A second look at our model:

$$\begin{aligned} p(y|x, \beta_0, \beta, \sigma) &= \frac{1}{\sqrt{(2\pi)\sigma^2}} \exp\left\{-\frac{(y - \beta_0 - x'\beta)^2}{2\sigma^2}\right\} \\ p(\beta_j) &= \frac{1}{\sqrt{(2\pi)\psi^2}} \exp\left\{-\frac{\beta_j^2}{2\psi^2}\right\} \qquad (j > 0) \end{aligned}$$

We have added a prior distribution on feature weights β. A Gaussian distribution with mean 0 and variance ψ^2

Regularization/prior

What does this prior distribution do? Consider just 2 predictors, like the SNPs we had before. These are the level curves of our prior:



A priori, before we see any data, we are saying that we prefer an even split of weights over just choosing one of them.

Likelihood $\times$ prior $\propto$ posterior

Recall Bayes theorem:

$$p(\beta|\mathrm{Data}) = \frac{\mathrm{p}(\mathrm{Data}|\beta)\mathrm{p}(\beta)}{\mathrm{p}(\mathrm{Data})}$$

Product of likelihood function and a prior yield a posterior distribution

$$p(\beta|\mathrm{Data}) \propto \underbrace{p(Data|\beta)}_{\text{likelihood}}\underbrace{p(\beta)}_{\text{prior}}$$

Finding β that maximizes $p(\beta|\mathrm{Data})$ is called MAP (Maximum a posteriori) estimation.

For the purposes of MAP estimation of $p(\mathrm{Data})$ is a constant. It is very important in model selection. We will come back to it.

Likelihood $\times$ prior $\propto$ posterior

In our case

$$L(\beta_0, \beta, \sigma) = \prod_{i=1}^{n} p(y_i | x_i, \beta_0, \beta, \sigma)$$

and a prior

$$p(\beta) = \prod_j \frac{1}{\sqrt{(2\pi)\psi^2}} \exp\left\{ -\frac{\beta_j^2}{2\psi^2} \right\}$$

yield the posterior distribution over β

$$\begin{aligned} p(\beta|y) \quad \propto \quad & \prod_i \frac{1}{\sqrt{(2\pi)\sigma^2}} \exp\left\{ -\frac{(y_i - \beta_0 - x_i'\beta)^2}{2\sigma^2} \right\} \times \\ & \prod_j \frac{1}{\sqrt{(2\pi)\psi^2}} \exp\left\{ -\frac{\beta_j^2}{2\psi^2} \right\} \end{aligned}$$

Regularized log likelihood and log posterior

Regularized log likelihood after eliminating all terms that do not depend on β

$$-\frac{1}{2\sigma^2}\sum_i (y_i - \beta_0 - x_i'\beta)^2 - \lambda \sum_{j=1}^{p} \beta_j^2$$

Log of posterior $p(\beta|y)$ after eliminating all terms that do not depend on β

$$-\frac{1}{2\sigma^2}\sum_i (y_i - \beta_0 - x_i'\beta)^2 - \frac{1}{2\psi^2} \sum_{j=1}^{p} \beta_j^2$$

So if we set $\lambda = \frac{1}{2\psi^2}$ this the same objective with respect to β.

Ridge regression

The cost we introduced is referred to as ridge regression cost and the $\sum \beta_j^2$ ridge term.

We can state the ridge regression objective as

$$LL_{\mathrm{ridge}}(\beta) = -\frac{1}{2}\sum_i (y_i - \beta_0 - x_i'\beta)^2 - \frac{\alpha}{2}\sum_{j=1}^{p}\beta_j^2$$

You will frequently see it with a sign flip and optimization presented as a minimization. We'll stick with likelihood and maximization.

Back to optimization

Take derivatives of the ridge regression objective and set them to zero, this gives us[2]

$$\begin{aligned}\beta_0 &= \frac{1}{n}\sum_i (y_i - x_i'\beta) \\ \beta_j &= \frac{1}{1+\alpha}\sum_i y_i^{(-j)} x_{i,j}\end{aligned}$$

Stating the obvious: setting $\alpha = 0$ recovers the regression update.

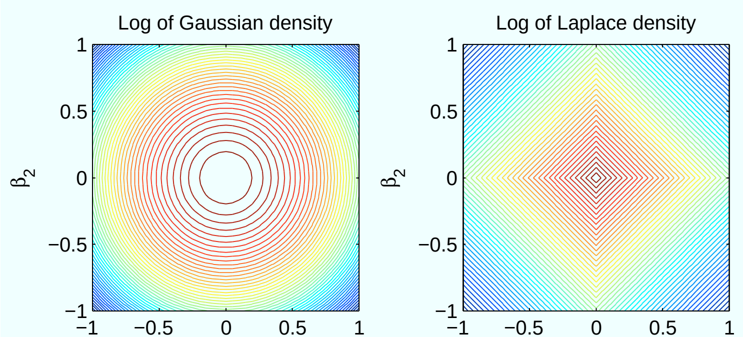
[2] reminder: $y_i^{(-j)} = y_i - \beta_0 - \sum_{k \neq j} x_{i,k}\beta_k$

More priors

Putting a Gaussian prior on β gave us an interesting and useful effect: we can deal with $p > n$ situations and weights get split across similar predictors.

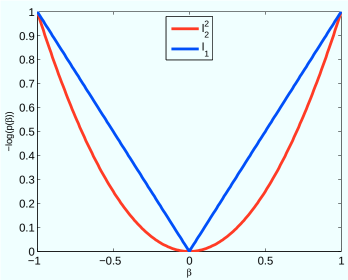
You can now play and put other priors on β, but one choice gets quite a bit of attention.

$$p(\beta_j) \propto \exp\left\{-\frac{1}{s}|\beta_j|\right\}$$



Encouraging sparsity - Penalty view

Ridge regression alleviates some of the problems of $p > n$ but does not drive feature weights to 0.



Red line corresponds to Gaussian prior and blue to Laplace prior.

Lasso = linear regression + ℓ_1 penalty

Or alternatively, linear regression with Laplace prior on β.

$$LL_{\text{lasso}}(\beta) = -1/2 \sum_i (y_i - \beta_0 - x_i'\beta)^2 - \lambda \sum_{j=1}^{p} |\beta_j|$$

Coordinate ascent for lasso

Not as easy as previous ones: $|\cdot|$ does not have a derivative at 0, so we can split cases

Suppose optimal $\beta_j > 0$ then we can write a partial derivative of objective down

$$\frac{\partial LL_{\text{lasso}}}{\partial \beta_j} = \sum_i (y - \beta_0 - x'\beta)(x_{i,j}) - \lambda$$

and equating this to 0 gives us[3]

$$\beta_j = \sum_i y_i^{(-j)} x_{i,j} - \lambda$$

as long as this does not invalidate our assumption that $\beta_j > 0$ we can accept this update.

[3] reminder: $y_i^{(-j)} = y_i - \beta_0 - \sum_{k \neq j} x_{i,k}\beta_k$

Coordinate ascent for lasso

Suppose optimal $\beta_j < 0$ then we can write a partial derivative of objective down

$$\frac{\partial LL_{\text{lasso}}}{\partial \beta_j} = \sum_i (y - \beta_0 - x'\beta)(x_{i,j}) + \lambda$$

and equating this to 0 gives us[4]

$$\beta_j = \sum_i y_i^{(-j)} x_{i,j} + \lambda$$

as long as this does not invalidate our assumption that $\beta_j < 0$ we can accept this update.

[4] reminder: $y_i^{(-j)} = y_i - \beta_0 - \sum_{k \neq j} x_{i,k}\beta_k$

Coordinate ascent for lasso

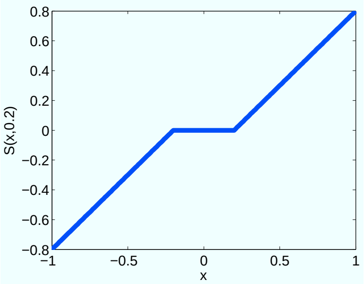
The only times we invalidate our assumptions are the ones where

$$\left|\sum_i y_i^{(-j)} x_{i,j}\right| < \lambda$$

Shrink and threshold

This reasoning leads to the definition of a shrinkage and threshold operator

$$S(x, \lambda) = \mathrm{sign}(x)\,\max(|x| - \lambda, 0)$$



Lasso updates

Now we can write down the lasso updates

$$\begin{aligned}\beta_0 &= \frac{1}{n}\sum_i (y_i - x_i'\beta) \\ \beta_j &= S\left(\sum_i y_i^{(-j)} x_{i,j}, \lambda\right)\end{aligned}$$

where $S(x,\lambda) = \mathrm{sign}(x)\max(|x| - \lambda, 0)$ and $y_i^{(-j)} = y_i - \beta_0 - \sum_{k\neq j} x_{i,k}\beta_k$

Objectives you can optimize now

Linear Regression using update

$$\beta_j = \sum_i y_i^{(-j)} x_{i,j}$$

Ridge Regression using update

$$\beta_j = \frac{1}{1+\alpha} \sum_i y_i^{(-j)} x_{i,j}$$

Lasso using update

$$\beta_j = S\left(\sum_i y_i^{(-j)} x_{i,j}, \lambda\right)$$

I have not told you how to pick λ,α etc. But you can think about cross validation and how you would use it.

Next time we will expand this to elastic net (Ridge + Lasso) talk about how to cross-validate and see some applications.

Notation

There is a standard way to write optimization problems

$$\begin{array}{rl} \underset{\theta \in \mathbf{R}^n}{\text{minimize}} & f(\theta) \\ \text{subject to} & g(\theta) = 0 \\ & h(\theta) \leq 0 \end{array}$$

and also a standard way to denote minimizers (optimal values for θ)

$$\underset{\theta \in \mathbf{R}^n, g(\theta)=0, h(\theta)\leq 0}{\text{argmin}} \quad f(\theta)$$

Notation – Lasso optimization problem

Recall the objective, a likelihood function restricted to terms involving β

$$-1/2 \sum_{i=1}^{n} (y_i - \beta_0 - x_i'\beta)^2 - \lambda \sum_{j=1}^{p} |\beta_j|$$

In the standard form for optimization problems we write

$$\underset{\beta_0 \in \mathbf{R}, \beta \in \mathbf{R}^p}{\text{minimize}} \quad \tfrac{1}{2\sigma^2} \textstyle\sum_{i=1}^{n} (y_i - \beta_0 - x_i'\beta)^2 + \lambda \textstyle\sum_{j=1}^{p} |\beta_j|$$

Note the sign flip: we maximize likelihoods/probabilities but we minimize costs/losses.

Elastic Net

Ridge regression as a means to deal with correlated predictors.

- Problem: too many predictors are non-zero

Lasso as means to deal with large number of predictors.

- Problem: arbitrary decisions between highly correlated predictors

How to combine sparsity of lasso and “equal” weighting of ridge?

Elastic Net

Interpolate between the two costs/priors

$$-1/2\sum_{i=1}^{n}(y_i-\beta_0-x_i'\beta)^2-\alpha\frac{\lambda}{2}\sum_{j=1}^{p}\beta_j^2-(1-\alpha)\lambda\sum_{j=1}^{p}|\beta_j|$$

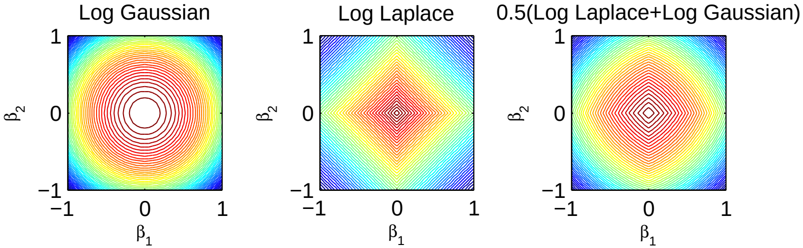
Average in log domain = geometric average in real domain.

In the extremes of α:

- $\alpha = 0$ recovers lasso
- $\alpha = 1$ recovers ridge

In principle, you should not do worse than lasso or ridge.

Elastic Net Level Curves[1]



$$-1/2\sum_{i=1}^{n}(y_i - \beta_0 - x_i'\beta)^2 - \alpha\frac{\lambda}{2}\sum_{j=1}^{p}\beta_j^2 - (1-\alpha)\lambda\sum_{j=1}^{p}|\beta_j|$$

[1] make your own: use `meshgrid` and `contour` in Matlab

Elastic Net Optimization: Coordinate Ascent

We are still in the world of coordinate ascent and assuming normalized predictors.
Taking derivatives of the objective, setting them to zero, and reasoning by cases, we get:

$$\begin{aligned}
\beta_0 &= \frac{1}{n}\sum_i y_i \\
\beta_j &= S\left(\frac{1}{1+(\alpha\lambda)}\sum_i y_i^{(-j)} x_{i,j}\ ,\ (1-\alpha)\lambda\right),
\end{aligned}$$

where

$$\begin{aligned}
S(x,\lambda) &\equiv \mathrm{sign}(x)\max(|x|-\lambda,0) \\
y_i^{(-j)} &\equiv y_i - \beta_0 - \sum_{k\neq j} x_{i,k}\beta_k
\end{aligned}$$

Choosing λ, α

Given data $\{(x_i, y_i) : i = 1..n\}$, λ, α we know how to produce β_0, β.

It is fair to say that β_0, β are functions of data, λ, α

Data is given but λ, α are chosen.

Choosing λ, α

Assume the training dataset is drawn from $p(x, y)$

Ideally we want to choose λ and α such that the resulting $\hat{\beta}_0$ and $\hat{\beta}$ give small error on future data.

$$E_p[(y - \hat{\beta}_0 - x'\hat{\beta})^2] = \int_x \int_y p(x, y)(y - \hat{\beta}_0 - x'\hat{\beta})^2 dxdy$$

We don't have the capability to compute this, since the true distribution $p(x, y)$ is hidden from us.

But, we can emulate future draws.

Choosing λ, α

Randomly split data into train and test (50/50,80/20).

Let Test denote indices of data that are in the test set and Train indices of data that are in the train set. $\mathrm{Test} \bigcap \mathrm{Train} = \emptyset$

Learn $\hat{\beta}_0, \hat{\beta}$ on $\{(x_i, y_i) : i \in \mathrm{Train}\}$ using some λ, α

In a limit of test set size

$$\mathrm{Err}_{\mathrm{Test}}(\hat{\beta}_0, \hat{\beta}) = \frac{1}{|Test|} \sum_{i \in Test} [(y_i - \hat{\beta}_0 - x_i' \hat{\beta})^2]$$

tends to

$$E_p[(y - \hat{\beta}_0 - x' \hat{\beta})^2]$$

Choosing λ, α

All this to say that we can use a test set error to evaluate our performance on unseen data.

We can choose α,λ so as to minimize this proxy (test error).

Cross-validation

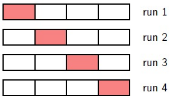
K-fold cross-validation: divide dataset into k subsets, call them $\mathrm{Set}_1, \ldots \mathrm{Set}_k$

Foreach i=1:k

1. Set $\mathrm{Test} = \mathrm{Set}_i$ and $\mathrm{Train} = \bigcup_{k \neq i} \mathrm{Set}_k$
2. Learn β_0 and β on Train using α, λ
3. $\mathrm{CVErr}_i(\alpha, \lambda) = \frac{1}{|Test|} \sum_{i \in Test} [(y_i - \hat{\beta}_0 - x_i' \hat{\beta})^2]$

$\mathrm{CVErr}(\alpha, \lambda) = \frac{1}{k} \sum_k \mathrm{CVErr}_k(\alpha, \lambda)$

4-fold illustration



Extreme $k = n$, size of dataset, then we obtain Leave-One-Out (LOO) scheme. And resulting estimate of error is LOO CV estimate.

CVErr above is mean across all folds; report standard deviations as well.

Choosing λ, α

So we can score a pair λ, α in a number of ways

How do we do search?

A universal approach: grid of values

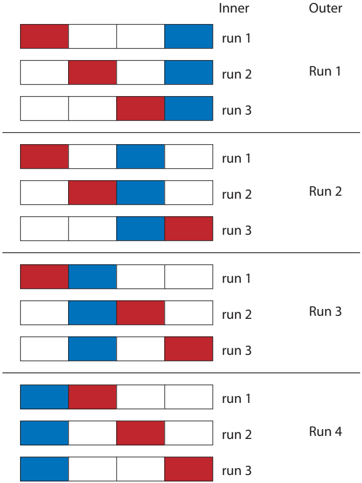
Question for you

Observation: The elastic net fitting method consists of fitting β_0, β *and* choosing α, λ via cross validation.

How do we estimate the performance of a method that involves cross validation as a part of training?

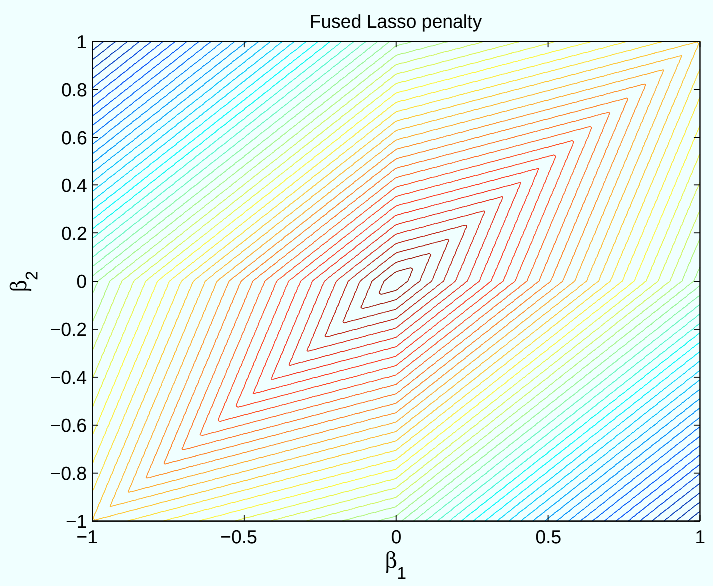
Question for you

Wrap it in another cross validation: $\underbrace{\overbrace{\text{Train},\text{Validate}}^{\text{inner}},\ \text{Test}}_{\text{outter}}$



	Inner	Outer
🟥⬜⬜🟦	run 1	
⬜🟥⬜🟦	run 2	Run 1
⬜⬜🟥🟦	run 3	
🟥⬜🟦⬜	run 1	
⬜🟥🟦⬜	run 2	Run 2
⬜⬜🟦🟥	run 3	
🟥🟦⬜⬜	run 1	
⬜🟦🟥⬜	run 2	Run 3
⬜🟦⬜🟥	run 3	
🟦🟥⬜⬜	run 1	
🟦⬜🟥⬜	run 2	Run 4
🟦⬜⬜🟥	run 3	

Coordinate ascent does not always work



$$-|\beta_1| - |\beta_2| - 2|\beta_1 - \beta_2|$$

Optimization of challenging objectives

Break coupling across non-smooth objectives by adding more variables.

Challenging

$$\underset{\beta \in \mathbf{R}^2}{\text{minimize}} \quad |\beta_1| + |\beta_2| + \kappa|\beta_1 - \beta_2|$$

Easier after adding an auxiliary variable

$$\begin{aligned} \underset{\beta \in \mathbf{R}^2, \delta \in \mathbf{R}}{\text{minimize}} \quad & |\beta_1| + |\beta_2| + \kappa|\delta| \\ \text{subject to} \quad & \quad \delta = \beta_1 - \beta_2 \end{aligned}$$

Making friends with Lagrangians

An example of a separable objective tied through a constraint

$$\begin{aligned} \underset{x,y}{\text{minimize}} \quad & f(x) + g(y) \\ \text{subject to} \quad & x = y \end{aligned}$$

and its Lagrangian

$$L(x, y, \lambda) = f(x) + g(y) + \lambda'(x - y)$$

Augmented Lagrangian

$$AL(x, y, \lambda) = f(x) + g(y) + \sum_i \lambda_i (x_i - y_i) + \rho/2 \sum_i (x_i - y_i)^2$$

Alternating Direction Method of Multipliers

Iterate updates

$$\begin{aligned} x &= \operatorname*{argmin}_x AL(x, y, \lambda) \\ y &= \operatorname*{argmin}_y AL(x, y, \lambda) \\ \lambda &= \lambda^{\text{prev}} + \rho(x_i - y_i), \end{aligned}$$

where $\rho > 0$

ADMM for fused lasso on 2 variables

The fused lasso reformulated problem

$$\begin{array}{rl}\underset{\beta\in\mathbf{R}^2,\delta\in\mathbf{R}}{\text{minimize}} & |\beta_1| + |\beta_2| + \kappa|\delta| \\ \text{subject to} & \delta - (\beta_1 - \beta_2) = 0\end{array}$$

and its Augmented Lagrangian

$$\begin{aligned} AL(\beta_1, \beta_2, \delta, \lambda) &= |\beta_1| + |\beta_2| + \kappa|\delta| + \\ &\quad \lambda(\delta - (\beta_1 - \beta_2)) + \rho/2(\delta - (\beta_1 - \beta_2))^2 \end{aligned}$$

ADMM for fused lasso on 2 variables

We willl to note that the update for β_1 (similarly β_2 and δ_1):

$$\begin{aligned}
\beta_1 &= \operatorname*{argmin}_{\beta_1} AL(\beta_1, \beta_2, \delta_1, \lambda) \\
&= \operatorname*{argmin}_{\beta_1} |\beta_1| - \lambda\beta_1 + \rho/2(\delta - (\beta_1 - \beta_2))^2
\end{aligned}$$

can be cast into a form that we already know how to solve.

Complete squares

$$\operatorname*{argmin}_{\beta_1} |\beta_1| - \lambda\beta_1 + \rho/2(\delta - (\beta_1 - \beta_2))^2$$

is equal to

$$\operatorname*{argmin}_{\beta_1} |\beta_1| + \rho/2(\beta_1 - (\delta + \beta_2 + \lambda/\rho))^2$$

Recognize the form?

Solution here is: $S(\delta + \beta_2 + \lambda/\rho, 1.0)$

Key steps in optimizing coupled penalties

1. Reformulate problem to remove sharing of variables between non-smooth parts of objective
2. Write down Augmented Lagrangian for your reformulated problem
3. Iterate the ADMM scheme

Suggestion: Use square completion trick to derive ADMM updates for fused lasso on 2 variables.

Implement this, check that updates are correct, play with ρ.

Logistic regression

We dealt with modeling a continuous variable – linear regression.

Frequently we care about modeling a discrete variable (label) with 2,3,4 … different states

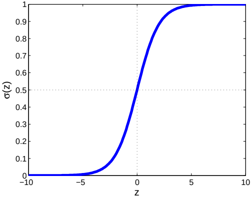
Today we deal with modeling a binary target variable:

- healthy vs. cancer
- responsive vs. nonresponsive
- binding vs. nonbinding peptide

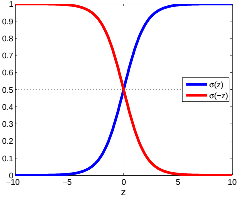
Sometimes a binary variable is the result of a discretization (high/low).

Sigmoid: Transforming your scalars into probabilities since...

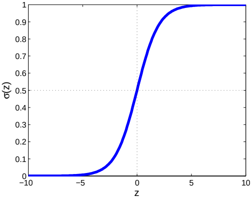
$$\sigma(z) = \frac{1}{1+\exp\{-z\}}$$



$$1 - \sigma(z) = \sigma(-z)$$

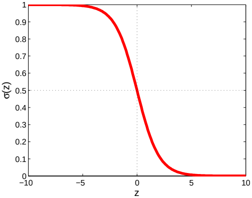


Candidate formulation



$$p(y = +1|x, \beta_0, \beta) = \sigma(+(\beta_0 + x'\beta)) = \frac{1}{1 + \exp\{-(\beta_0 + x'\beta)\}}$$

Candidate formulation



$$p(y = -1 | x, \beta_0, \beta) = \sigma(-(\beta_0 + x'\beta)) = \frac{1}{1 + \exp\{+(\beta_0 + x'\beta)\}}$$

Candidate formulation

Since we have

$$p(y = +1|x, \beta_0, \beta) = \sigma(+(\beta_0 + x'\beta)) = \frac{1}{1 + \exp\left\{-(\beta_0 + x'\beta)\right\}}$$

and

$$p(y = -1|x, \beta_0, \beta) = \sigma(-(\beta_0 + x'\beta)) = \frac{1}{1 + \exp\left\{+(\beta_0 + x'\beta)\right\}}$$

we can merge these two statements into

$$p(y|x, \beta_0, \beta) = \sigma(y(\beta_0 + x'\beta)) = \frac{1}{1 + \exp\left\{-y(\beta_0 + x'\beta)\right\}}$$

Logistic regression

Probability of y

$$p(y|x, \beta_0, \beta) = \frac{1}{1 + \exp\{-y(\beta_0 + x'\beta)\}}$$

Likelihood function

$$L(\beta_0, \beta) = \prod_i p(y_i|x_i, \beta_0, \beta) = \prod_i \frac{1}{1 + \exp\{-y_i(\beta_0 + x_i'\beta)\}}$$

Log-Likelihood function

$$\mathrm{LL}(\beta_0, \beta) = -\sum_i \log\{1 + \exp\{-y_i(\beta_0 + x_i'\beta)\}\}$$

Maximum likelihood

We are again pursuing maximum likelihood estimate for β_0, β this time for logistic regression.

We need to ascend the maximum likelihood surface to find an optimal β_0, β.

We will again use partial derivatives to accomplish this.

$$\nabla \mathrm{LL}(\beta_0, \beta) = \begin{bmatrix} \frac{\partial LL(\beta_0,\beta)}{\partial \beta_0} \\ \frac{\partial LL(\beta_0,\beta)}{\partial \beta_1} \\ \dots \\ \frac{\partial LL(\beta_0,\beta)}{\partial \beta_p} \end{bmatrix}$$

Optimization

$$\mathrm{LL}(\beta_0, \beta) = -\sum_i \log\left\{1 + \exp\left\{-y_i(\beta_0 + x_i'\beta)\right\}\right\}$$

We can take partial derivatives

$$\begin{aligned}
\frac{\partial \mathrm{LL}(\beta_0, \beta)}{\partial \beta_j} &= \sum_i \frac{\exp\left\{-y_i(\beta_0 + x_i'\beta)\right\}}{1 + \exp\left\{-y_i(\beta_0 + x_i'\beta)\right\}} y_i x_{i,j} \\
&= \sum_i \left(1 - \frac{1}{1 + \exp\left\{-y_i(\beta_0 + x_i'\beta)\right\}}\right) y_i x_{i,j} \\
&= \sum_i \left(1 - p(y_i|x_i, \beta_0, \beta)\right) y_i x_{i,j}
\end{aligned}$$

Optimization: Cannot get closed form coordinate ascent

We have partial derivatives

$$\frac{\partial \mathrm{LL}(\beta_0, \beta)}{\partial \beta_j} = \sum_i \left(1 - p(y_i | x_i, \beta_0, \beta)\right) y_i x_{i,j}$$

but we cannot solve for optimal β_j by setting the partial derivatives to 0.

Optimization

We cannot do coordinate ascent so we will use gradient a little differently

$$\begin{aligned} \nabla \mathrm{LL}(\beta_0, \beta) &= \left[\begin{array}{l} \sum_i (1 - p(y_i | x_i, \beta_0, \beta)) y_i \\ \sum_i (1 - p(y_i | x_i, \beta_0, \beta)) y_i x_{i,1} \\ \dots \\ \sum_i (1 - p(y_i | x_i, \beta_0, \beta)) y_i x_{i,p} \end{array} \right] \begin{array}{l} (\beta_0) \\ (\beta_1) \\ \\ (\beta_p) \end{array} \\ &= \sum_i (1 - p(y_i | x_i, \beta_0, \beta)) y_i \left[\begin{array}{c} 1 \\ x_i \end{array} \right] \end{aligned}$$

Check your gradients

Remember finite differences

$$\frac{\mathrm{LL}(\beta_0, \beta_1, ..., \beta_j + h, ...\beta_n) - \mathrm{LL}(\beta_0, \beta_1, ..., \beta_j, ...\beta_n)}{h}$$

approximate

$$\frac{\partial \mathrm{LL}}{\partial \beta_j}(\beta_0, \beta)$$

for small h (1e-10)

Easy optimization: gradient ascent

$$
\begin{aligned}
\beta_0^{\mathrm{new}} &= \beta_0 + s \sum_i (1 - p(y_i|x_i, \beta_0, \beta)) y_i \\
\beta^{\mathrm{new}} &= \beta + s \sum_i (1 - p(y_i|x_i, \beta_0, \beta)) y_i x_i
\end{aligned}
$$

s is a small step size (learning rate) chosen in an adhoc manner (1e-1,1e-2,1e-3).

Too large a step results in worse likelihood.
Too small a step results in slow algorithm.

Instead of a fixed step size, we can use *backtracking*.

Backtracking idea

We are given objective $f(x)$, a current set of parameters x and a direction g (e.g. $g = \nabla f(x)$).

$$x^{\text{new}} = x + sg$$

but we do not know a good step size s

You can try a schedule of step sizes, for example

$$1, 0.95, 0.95^2, \ldots 0.95^n$$

until you find one for which $f(x + sg)$ is better than $f(x)$.

Backtracking because it looks like you are backtracking from your first ambitious step.

Gradient ascent method with backtracking

- Start at some β_0, β
- While LL is sufficienty changing repeat
  1. compute $\nabla \mathrm{LL}(\beta_0, \beta)$
  2. use backtracking along direction $\nabla \mathrm{LL}(\beta_0, \beta)$ to get s
  3. $\left[\begin{array}{c} \beta_0 \\ \beta \end{array} \right] = \left[\begin{array}{c} \beta_0 \\ \beta \end{array} \right] + s \nabla \mathrm{LL}(\beta_0, \beta)$

Newton's method

Newton's method finds zeros of functions by using their first gradient.

A point where all of the partial derivatives are zero is a critical point (maximum for a concave function).

$$\nabla \mathrm{LL}(\beta_0, \beta) \equiv 0$$

To use Newton's method we need Hessian ($\nabla^2 \mathrm{LL}(\beta_0, \beta)$)

Newton's method

If we have the gradient ($\nabla \mathrm{LL}$) and Hessian ($\nabla^2 \mathrm{LL}$) then a single step of Newton's method is

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \ldots \\ \beta_n \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \ldots \\ \beta_n \end{bmatrix} - s \nabla^2 \mathrm{LL}(\beta_0, \beta)^{-1} \nabla \mathrm{LL}(\beta_0, \beta)$$

The costly part is Hessian matrix inversion (solution of a linear system).

There is still step size to worry about.

Optimization: Hessian of logistic regression likelihood

Gradient

$$\begin{aligned}\nabla \mathrm{LL}(\beta_0, \beta) &= \sum_i (1 - p(y_i | x_i, \beta_0, \beta)) y_i \left[\begin{array}{c} 1 \\ x_i \end{array} \right] \\ &= \sum_i \sigma(-y_i(\beta_0 + \beta' x)) y_i \left[\begin{array}{c} 1 \\ x_i \end{array} \right]\end{aligned}$$

We will be taking derivatives of $\sigma(y_i(\beta_0 + \beta' x))$

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)\sigma(-z)dz$$

Optimization: Hessian of logistic regression likelihood

With the gradient (partial derivatives) and the equality for the derivative of the sigmoid we can can compute Hessian ($\nabla^2 \mathrm{LL}$)

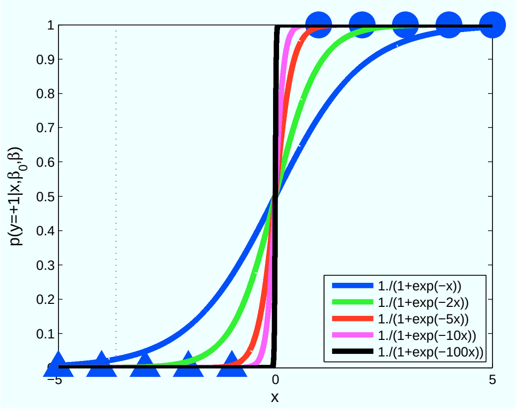
$$\begin{aligned}\frac{\partial^2 \mathrm{LL}(\beta_0, \beta)}{\partial \beta_j \partial \beta_k} &= \sum_i \tilde{x}_{i,j} \tilde{x}_{i,k} \sigma(y_i(\beta_0 + \beta' x_i)) \sigma(-y_i(\beta_0 + \beta' x_i)) \\ &= - \sum_i \tilde{x}_{i,j} \tilde{x}_{i,k} p(y_i | x_i, \beta_0, \beta)(1 - p(y_i | x_i, \beta_0, \beta))\end{aligned}$$

where $\tilde{x}_{i,0} = 1$ and $\tilde{x}_{i,j} = x_{i,j}$ for $j > 0$
Gradient ($\nabla \mathrm{LL}$) in this notation

$$\frac{\partial L(\beta_0, \beta)}{\partial \beta_j} = \sum_i (1 - p(y_i | x_i, \beta_0, \beta)) y_i \tilde{x}_{i,j}$$

Logistic regression in a well separated case

There is nothing stopping the fitting procedure from scaling βs up if data is separable



Ridge ... again

$$\mathrm{LL}(\beta_0, \beta) = -\sum_i \log\left\{1 + \exp\left\{-y_i(\beta_0 + x_i'\beta)\right\}\right\} - \frac{\lambda}{2}\sum_j \beta_j^2$$

which is same as putting a Gaussian prior on the feature weights (β)

$$\begin{aligned} p(y|x, \beta_0, \beta) &= \frac{1}{1 + \exp\{-y(\beta_0 + x'\beta)\}} \\ p(\beta_j) &= \frac{1}{\sqrt{(2\pi)\psi^2}} \exp\left\{-\frac{\beta_j^2}{2\psi^2}\right\} \qquad (j > 0) \end{aligned}$$

Changes for optimization

Very little work involved in changing gradients (can proceed on to gradient ascent)

$$\frac{\partial \mathrm{LL}(\beta_0, \beta)}{\partial \beta_j} = \sum_i (1 - p(y_i | x_i, \beta_0, \beta)) y_i \tilde{x}_{i,j} - \lambda \beta_j$$

and even less in Hessian – only diagonal changes

$$\begin{aligned}
\frac{\partial^2 \mathrm{LL}(\beta_0, \beta)}{\partial^2 \beta_j} &= -\sum_i \tilde{x}_{i,j}^2 p(y_i | x_i, \beta_0, \beta)(1 - p(y_i | x_i, \beta_0, \beta)) - \lambda \\
\frac{\partial^2 \mathrm{LL}(\beta_0, \beta)}{\partial^2 \beta_0} &= -\sum_i \tilde{x}_{i,j}^2 p(y_i | x_i, \beta_0, \beta)(1 - p(y_i | x_i, \beta_0, \beta))
\end{aligned}$$

where $\tilde{x}_{i,0} = 1$ and $\tilde{x}_{i,j} = x_{i,j}$ for $j > 0$

Taylor – single variable

Given a function f and its derivatives $f', f'', f^{(3)}...$ evaluated at x_0 we can approximate $f(x_0 + d)$ with

$$f(x_0) + f'(x_0)d + \frac{1}{2!}f''(x_0)d^2 + \frac{1}{3!}f^{(3)}(x_0)d^3 + ... \qquad (1)$$

Frame of mind: the expresion (1) is a polynomial in d

If d is small then $d^2, d^3, ...$ drop off quickly, further $\frac{1}{n!}$ also drops off quickly so the contribution from the higher order derivatives is scaled down.

Taylor – single variable

$$f(x_0) + f'(x_0)d + \frac{1}{2!}f''(x_0)d^2 + \underbrace{\frac{1}{3!}f^{(3)}(x_0)d^3 + \dots}_{\text{higher order terms}} \qquad (2)$$

In general you can find some interval $[-d_{\max}, d_{\max}]$ such that the contribution from higher order terms is smaller in absolute value than some preset ϵ.

$$\left| \frac{1}{3!}f^{(3)}(x_0)d^3 + \frac{1}{4!}f^{(4)}(x_0)d^4 + \dots \right| \leq \epsilon$$

Taylor – single variable

So for a given x_0 and a $d \in [-d_{\max}, d_{\max}]$ you can define a quadratic polynomial in d

$$q(d) = f(x_0) + f'(x_0)d + \frac{1}{2!}f''(x_0)d^2$$

and guarantee that

$$|f(x_0 + d) - q(d)| < \epsilon$$

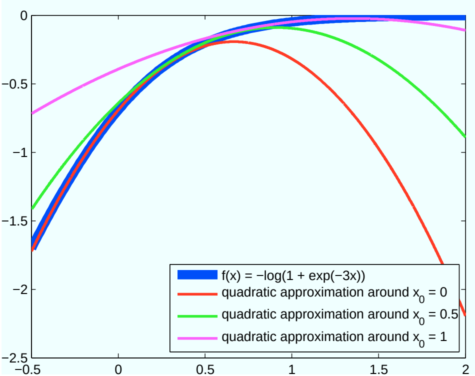
Upshot: If we are willing to accept the ϵ error and restrict d we can work with the quadratic polynomial $q(d)$ instead of the possibly hairy $f(x_0 + d)$

Fun with quadratics

A quadratic approximation of $f(x)$ around x_0

$$q_{x_0}(d) = f(x_0) + f'(x_0)d + \frac{1}{2!}f''(x_0)d^2$$

I am using notation q_{x_0} to emphasise that this is a quadratic approximation around x_0, because ...



Fun with quadratics

We know how to construct quadratic approximations of functions around a given x_0.

We can use them as proxies for original function in some region $[x_0 - d_{\max}, x_0 + d_{\max}]$

Putting quadratic approximation to use

Well then, let's find its maximizer. This is the point where $q'_{x_0}(d) = 0$ and since

$$q'_{x_0}(d) = f'(x_0) + f''(x_0)d$$

we can set

$$d^* = \operatorname*{argmax}_d q_{x_0}(d) = -\frac{f'(x_0)}{f''(x_0)}$$

This would take us to the optimum of our quadratic approximation.

Putting quadratic approximation to use

$$d^* = \operatorname*{argmax}_d q_{x_0}(d) = -\frac{f'(x_0)}{f''(x_0)}$$

If $d^* \in [-d_{\max}, d_{\max}]$ we can suggest a reasonable guess of the maximizer of f

$$x_0 + d^*$$

We can say that $f(x_0 + d^*)$ is within ϵ of local maximum of $f(x)$.

Putting quadratic approximation to use

We can also take any d between 0 and d^* and guarantee that $q(d) \geq q(0)$. This is a step towards the optimum.

Further, we can compute the improvement $q(d) - q(0)$ under our quadratic approximation and compare to $f(x_0 + d) - f(x_0)$

Remember, as we shrink d these two improvements will converge.

You should be reminded of the backtracking: We shrink d until our approximation is satisfactory.

Generalization to higher dimensional problems

We can still use Taylor expansion:

$$f(\mathbf{x}_0 + \mathbf{d}) = f(\mathbf{x}_0) + \mathbf{d}'\nabla f(\mathbf{x}_0) + \frac{1}{2}\mathbf{d}'\nabla^2 f(\mathbf{x}_0)\mathbf{d} + \ldots$$

where gradient

$$\nabla f(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}_1} f(\mathbf{x}_0) \\ \ldots \\ \frac{\partial}{\partial \mathbf{x}_n} f(\mathbf{x}_0) \end{bmatrix}$$

and Hessian

$$\nabla\nabla f(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial^2}{\partial^2 \mathbf{x}_1} f(\mathbf{x}_0) & \frac{\partial^2}{\partial \mathbf{x}_1 \partial \mathbf{x}_2} f(\mathbf{x}_0) & \ldots & \frac{\partial^2}{\partial \mathbf{x}_1 \partial \mathbf{x}_n} f(\mathbf{x}_0) \\ \frac{\partial^2}{\partial \mathbf{x}_2 \partial \mathbf{x}_1} f(\mathbf{x}_0) & \frac{\partial^2}{\partial^2 \mathbf{x}_2} f(\mathbf{x}_0) & \ldots & \frac{\partial^2}{\partial \mathbf{x}_2 \partial \mathbf{x}_n} f(\mathbf{x}_0) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial \mathbf{x}_n \partial \mathbf{x}_1} f(\mathbf{x}_0) & \frac{\partial^2}{\partial \mathbf{x}_n \partial \mathbf{x}_2} f(\mathbf{x}_0) & \ldots & \frac{\partial^2}{\partial^2 \mathbf{x}_n} f(\mathbf{x}_0) \end{bmatrix}$$

Hessian bowls

One interesting classification of matrices revolves around their definiteness.

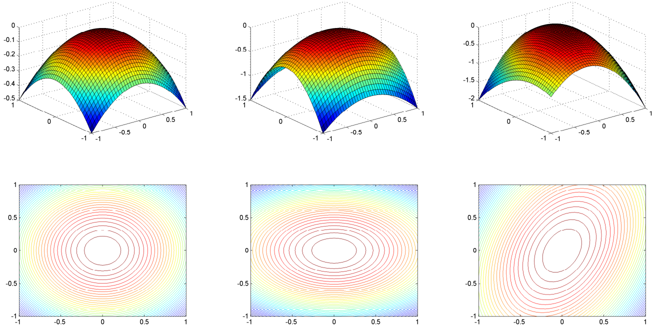
A matrix $A \in \mathbf{R}^{n \times n}$ is

- positive semi definite (psd) if $x'Ax \geq 0$ for all $x \in \mathbf{R}^n$
- negative semi definite (nsd) if $x'Ax \leq 0$ for all $x \in \mathbf{R}^n$

This is relevant to us in the context of Hessian matrices because positive (negative) definite Hessian matrices guarantee convexity (concavity) of objective.

Examples of functions with negative semi definite Hessian

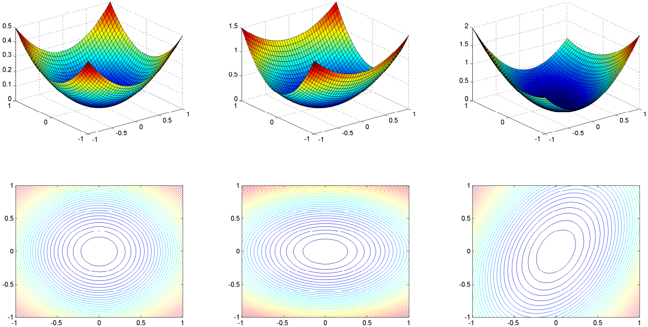
In all examples below $f(x) = (1/2)x'Hx$



Hessian $\begin{bmatrix} -0.5 & 0 \\ 0 & -0.5 \end{bmatrix}$ $\begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$ $\begin{bmatrix} -2 & 0.5 \\ 0.5 & -1 \end{bmatrix}$

Examples of functions with positive semi definite Hessian

In all examples below $f(x) = (1/2)x'Hx$



Hessian $\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ $\begin{bmatrix} 2 & -0.5 \\ -0.5 & 1 \end{bmatrix}$

Question

We know how to numerically check if our analytically derived gradients are right – finite differences.

How do we check an analytically derived Hessian?

Back to Taylor and higher dimensional problems

As we did in case of 1D optimization, we can adopt a quadratic approximation of f around some $\mathbf{x}_0$

$$q_{\mathbf{x}_0}(\mathbf{d}) = f(\mathbf{x}_0) + \mathbf{d}'\nabla f(\mathbf{x}_0) + \frac{1}{2}\mathbf{d}'\nabla^2 f(\mathbf{x}_0)\mathbf{d}$$

and optimize this function instead of f

Newton again

Optimum of $q_{\mathbf{x}_0}(d)$ is achieved at

$$\mathbf{d}^* = \operatorname*{argmax}_{\mathbf{d}} q_{x_0}(\mathbf{d}) = -(\nabla^2 f(\mathbf{x}_0))^{-1} \nabla f(\mathbf{x}_0)$$

and you can compare this to the 1D version

$$d^* = \operatorname*{argmax}_{d} q_{x_0}(d) = -\frac{f'(x_0)}{f''(x_0)}$$

Step sizes

Again, as in the 1D case, quadratic function is only a reasonable approximation locally.

Usually we cannot take a full step ($s = 1$) as we may not trust the quadratic approximation that far away

$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} - s[\nabla^2 f(\mathbf{x}^{\text{old}})]^{-1}\nabla f(\mathbf{x}^{\text{old}})$$

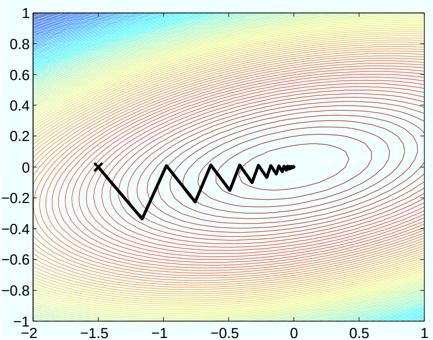
so we have to employ some sort of step-size search algorithm[1], usually a backtracking variant.

[1] also called line-search since we search along direction $(\nabla^2 f(\mathbf{x}_0))^{-1}\nabla f(\mathbf{x}_0)$

Obligatory steepest descent/ascent slide

Hessians seem like a bother, can't we just use gradients?

$$f(\mathbf{x}) = (1/2)\mathbf{x}' \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & 4 \end{bmatrix} \mathbf{x}$$



With Newton's method this problem is solved in a single step.

Standard optimization techniqus

- Steepest descent: cheap steps, but too many of them.
- Newton: expensive step, but smaller number of them

Alternatives

- BFGS (BroydenFletcherGoldfarbShanno) approximates $[\nabla^2 f]^{-1}$ (inverse of Hessian)
- Conjugate gradients avoid zig-zagging of steepest descent by avoiding undoing previous iterations

Solid implementations for both available. A nice tutorial on conjugate gradients:
`http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`

Also look at:
`http://www.di.ens.fr/~mschmidt/Software/minFunc.html`

Things we talked about so far

- Gradient descent
- Newton
- and in passing L-BFGS,conjugate gradients

In the context of logistic regression, it turns out that there is a fast algorithm that takes advantage of quadratic approximation but avoids explicitly inverting Hessians.

Coordinate ascent strikes again – coordinate ascent on quadratic *not* the logistic objective.

Completing squares

Saw it before, let me repeat it

$$\operatorname*{argmax}_{\beta} - \sum_i \sum_j a_i \beta_j x_{i,j} - (1/2) \sum_i \sum_j \sum_k \beta_j \beta_k x_{i,j} x_{i,k}$$

has the same optimal β as

$$\operatorname*{argmax}_{\beta} -(1/2) \sum_i (a_i - \sum_j \beta_j x_{i,j})^2$$

because the objectives differ by a term that does not depend on β – a constant for purposes of this particular optimization.

The particular constant here is $(1/2) \sum_i a_i^2$.

Using least squares to optimize logistic regression

The log likelihood function for logistic regression

$$\mathrm{LL}(\beta_0, \beta) = -\sum_i \log\left\{1 + \exp\left\{-y_i(\beta_0 + x_i'\beta)\right\}\right\}$$

Quadratic approximation around $\hat{\beta}_0, \hat{\beta}$

$$q_{\hat{\beta}_0,\hat{\beta}}(d) = \mathrm{LL}(\hat{\beta}_0, \hat{\beta}) + d'\nabla\mathrm{LL}(\hat{\beta}_0, \hat{\beta}) + (1/2)d'\nabla^2\mathrm{LL}(\hat{\beta}_0, \hat{\beta})d$$

where

$$\nabla\mathrm{LL}(\hat{\beta}_0, \hat{\beta})_j = \sum_i (1 - p(y_i|x_i, \hat{\beta}_0, \hat{\beta}))y_i\tilde{x}_{i,j}$$

and

$$[\nabla^2\mathrm{LL}(\hat{\beta}_0, \hat{\beta})]_{j,k} = -\sum_i \tilde{x}_{i,j}\tilde{x}_{i,k}p(y_i|x_i, \hat{\beta}_0, \hat{\beta})(1 - p(y_i|x_i, \hat{\beta}_0, \hat{\beta}))$$

with $\tilde{x}_{i,0} = 1$ and $\tilde{x}_{i,j} = x_{i,j}$

Using least squares to optimize logistic regression

After some massaging[2] we can obtain a problem of familiar form

$$\operatorname*{argmax}_{\beta_0,\beta} -(1/2)\sum_i (1-t_i)t_i\left(z_i-\beta_0-x_i'\beta\right)^2$$

where $t_i = p(y_i|x_i,\hat{\beta}_0,\hat{\beta})$ and $z_i = \frac{y_i}{t_i}+\hat{\beta}_0+x_i'\hat{\beta}$.

This is a weighted least squares problem since we have different weights associated with each instance.

Coordinate ascent is an option again. Note that the coordinate ascent is done on an approximate objective (our quadratic)!

[2] complete squares while working with d; do change of variables $d=\beta-\hat{\beta}$

Coordinate ascent updates

Take the derivative, set it to 0, solve for update

$$\begin{aligned}\beta_0^* &= \frac{\sum_i (1-t_i)t_i(z_i - x_i'\beta)}{\sum_i (1-t_i)t_i} \\ \beta_j^* &= \frac{\sum_i (1-t_i)t_i(z_i - \beta_0 - \sum_{k\neq j} x_{i,k}\beta_k)x_{i,j}}{\sum_i (1-t_i)t_i x_{i,j}^2}\end{aligned}$$

Note that β_0^*, β^* yield an improvement for the quadratic approximation, but not necessarily for the original objective.

We still have to do a line search and find $\beta^{\mathrm{new}} \in [\beta, \beta^*]$ and $\beta_0^{\mathrm{new}} \in [\beta_0, \beta_0^*]$ that also yield an improvement in the original objective

$$\begin{aligned}\beta_0^{\mathrm{new}} &= \tau\beta_0 + (1-\tau)\beta_0^* \\ \beta^{\mathrm{new}} &= \tau\beta + (1-\tau)\beta^*\end{aligned}$$

where $\tau \in [0,1]$

Iterating updates

Note that $t_i = p(y_i | x_i, \beta_0, \beta)$ and this will change as you update your βs.

A new linear regression problem is constructed on the fly to better approximate the underlying logistic regression problem.

The payoff – after taking all of those derivatives

A week or so ago we labored to get updates for *linear regression* with various penalties/priors.

It turns out that we can convert logistic regression cost into a sequence of weighted linear regression problems[3].

Then we can convert penalized logistic regression into penalized weighted linear regression.

$$\begin{aligned}
\mathrm{LL}(\beta_0, \beta) &\approx q_{\hat\beta_0,\hat\beta}(\beta_0 - \hat\beta_0, \beta - \hat\beta) \\
\mathrm{LL}(\beta_0, \beta) - \lambda \sum_j \beta_j^2 &\approx q_{\hat\beta_0,\hat\beta}(\beta_0 - \hat\beta_0, \beta - \hat\beta) - \lambda \sum_j \beta_j^2 \\
\mathrm{LL}(\beta_0, \beta) - \lambda \sum_j |\beta_j| &\approx q_{\hat\beta_0,\hat\beta}(\beta_0 - \hat\beta_0, \beta - \hat\beta) - \lambda \sum_j |\beta_j| \\
&\cdots
\end{aligned}$$

[3] Iteratively Reweighted Least Squares (IRLS)

Logistic regression with elastic net

Since elastic net subsumes lasso and ridge we will procede directly to that update using $w_i = (1 - t_i)t_i$

$$\begin{aligned}
\beta_0^* &= \frac{\sum_i w_i(z_i - x_i'\beta)}{\sum_i w_i} \\
\beta_j^* &= \frac{1}{\sum_i w_i x_{i,j}^2 + \alpha\lambda} S\left(\sum_i w_i(z_i - \beta_0 - \sum_{k\neq j} x_{i,k}\beta_k)x_{i,j}, (1-\alpha)\lambda\right),
\end{aligned}$$

and we can write updates

$$\begin{aligned}
\beta_0^{\text{new}} &= \tau\beta_0 + (1-\tau)\beta_0^* \\
\beta^{\text{new}} &= \tau\beta + (1-\tau)\beta^*,
\end{aligned}$$

that still require a search over τ.