A short course on optimization for distributed machine learning

Jong-Han Kim

EIC7024
Kyung Hee University

Course contents

- Machine learning problems
  - Regression
  - Classification
- Optimization for machine learning
  - Optimization algorithms
  - Distributed optimization and statistical learning

Predictors

Information flow

$$\begin{tikzcd}
u \arrow[r] & \boxed{\phi(\cdot)} \arrow[r, "x"] & \boxed{f(\cdot)} \arrow[r, "y"] & \boxed{\psi^{-1}(\cdot)} \arrow[r] & v
\end{tikzcd}$$

$x = \phi(u)$

$y = \psi(v)$

raw data	*feature*	*outcome*	*raw output*
student	hrs. sleep	exam score	grade
	hrs. study		
	hrs. facebook		
	alcohol freq.		
	$\vdots$		

Data fitting

- we think $y \in \mathbf{R}$ and $x \in \mathbf{R}^d$ are (approximately) related by

$$y \approx f(x)$$

- x is called the *independent variable* or *feature vector*
- y is called the *outcome* or *response* or *target* or *label* or *dependent variable*
- often y is something we want to predict
- we don't know the 'true' relationship between x and y

Features

often x is a vector of features:

- documents
  - x is word count histogram for a document
- patient data
  - x are patient attributes, test results, symptoms
- customers
  - x is purchase history and other attributes of a customer

Where features come from

- we use u to denote the raw input data, such as a vector, word or text, image, video, audio, . . .
- $x = \phi(u)$ is the corresponding *feature vector*
- the function ϕ is called the *embedding* or *feature function*
- ϕ might be very simple or quite complicated
- similarly, the raw output data v can be featurized as $y = \psi(v)$
- often we take $\phi(u)_1 = x_1 = 1$, the *constant feature*
- (much more on these ideas later)

Data and prior knowledge

- we are given data $x^1, \ldots, x^n \in \mathbf{R}^d$ and $y^1, \ldots, y^n \in \mathbf{R}$
- (x^i, y^i) is the ith *data pair* or *observation* or *example*
- we also (might) have *prior knowledge* about what f might look like
  - *e.g.*, f is smooth or continuous: $f(x) \approx f(\tilde{x})$ when x is near $\tilde{x}$
  - or we might know $y \geq 0$

Predictor

- we seek a *predictor* or *model* $g : \mathbf{R}^d \to \mathbf{R}$
- for feature vector x, our prediction (of y) is $\hat{y} = g(x)$
- predictor g is chosen based on both data and prior knowledge
- in terms of raw data, our predictor is

  $$\hat{v} = \psi^{-1}(g(\phi(u)))$$

  (with a slight variation when ψ is not invertible)

- $\hat{y}^i \approx y^i$ means our predictor does well on ith data pair
- *but our real goal is to have $\hat{y} \approx y$ for (x, y) pairs we have not seen*

Prediction methods

- fraud, psychic powers, telepathy, magic sticks, incantations, crystals, hunches, statistics, AI, machine learning, data science
- and many algorithms . . .
- example: nearest neighbor predictor
  - given x, find its nearest neighbor x^i among given data
  - then predict $\hat{y} = g(x) = y^i$

A learning algorithm is a recipe for producing a predictor given data

Linear predictors

Linear predictor

- predictors that are linear functions of x are widely used
- a linear predictor has the form

$$g(x) = \theta^{\mathsf{T}} x$$

  for some vector $\theta \in \mathbf{R}^d$, called the *predictor parameter vector*
- also called a *regression model*
- x_j is the jth feature, so the prediction is a linear combination of features

$$\hat{y} = g(x) = \theta_1 x_1 + \cdots + \theta_d x_d$$

- we get to choose the predictor parameter vector $\theta \in \mathbf{R}^d$
- sometimes we write $g_\theta(x)$ to emphasize the dependence on θ

Interpreting a linear predictor

$$\hat{y} = g(x) = \theta_1 x_1 + \cdots + \theta_d x_d$$

- θ_3 is the amount that prediction $\hat{y} = g(x)$ increases when x_3 increases by 1
  - particularly interpretable when x_3 is Boolean (only takes values 0 or 1)
- $\theta_7 = 0$ means that the prediction does not depend on x_7
- θ small means predictor is insensitive to changes in x:

$$|g(x) - g(\tilde{x})| = \left|\theta^\mathsf{T} x - \theta^\mathsf{T} \tilde{x}\right| = \left|\theta^\mathsf{T}(x - \tilde{x})\right| \le ||\theta||\, ||x - \tilde{x}||$$

Affine predictor

- suppose the first feature is constant, $x_1 = 1$
- the linear predictor g is then an *affine function* of $x_{2:d}$, *i.e.*, linear plus a constant
$$g(x) = \theta^{\mathsf{T}} x = \theta_1 + \theta_2 x_2 + \cdots + \theta_d x_d$$
- θ_1 is called the *offset* or *constant term* in the predictor
- θ_1 is the prediction when all features (except the constant) are zero

Empirical risk minimization

Loss function

a *loss* or *risk* function $\ell : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ quantifies how well (more accurately, how badly) $\hat{y}$ approximates y

- smaller values of $\ell(\hat{y}, y)$ indicate that $\hat{y}$ is a good approximation of y
- typically $\ell(y, y) = 0$ and $\ell(\hat{y}, y) \geq 0$ for all $\hat{y}$, y

examples

- *quadratic loss*: $\ell(\hat{y}, y) = (\hat{y} - y)^2$
- *absolute loss*: $\ell(\hat{y}, y) = |\hat{y} - y|$

Empirical risk

how well does the predictor g fit a data set (x^i, y^i), $i = 1, \ldots, n$, with loss ℓ?

- the *empirical risk* is the average loss over the data points,

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}^i, y^i) = \frac{1}{n}\sum_{i=1}^{n} \ell(g(x^i), y^i)$$

- if $\mathcal{L}$ is small, the predictor predicts the given data well

- when the predictor is parametrized by θ, we write

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(g_\theta(x^i), y^i)$$

  to show the dependence on the predictor parameter θ

Empirical risk minimization

- choosing the parameter θ in a parametrized predictor $g_\theta(x)$ is called *fitting the predictor* (to data)
- *empirical risk minimization (ERM)* is a general method for fitting a parametrized predictor
- ERM: *choose θ to minimize empirical risk $\mathcal{L}(\theta)$*
- thus, ERM chooses θ by attempting to match given data
- often there is no analytic solution to this minimization problem, so we use *numerical optimization* to find θ that minimizes $\mathcal{L}(\theta)$
  (more on this topic later)

Lease squares linear regression

Least squares linear regression

- linear predictor $\hat{y} = g_\theta(x) = \theta^\mathsf{T} x$
- $\theta \in \mathbf{R}^d$ is the model parameter
- we'll use square loss function $\ell(\hat{y}, y) = (\hat{y} - y)^2$
- empirical risk is MSE

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} (\theta^\mathsf{T} x^i - y^i)^2$$

- ERM: choose model parameter θ to minimize MSE
- called *linear least squares fitting* or *linear regression*

Least squares formulation

- express MSE in matrix notation as

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\theta^{\mathsf{T}}x^i - y^i)^2 = \frac{1}{n}\left\{(\theta^{\mathsf{T}}x^1 - y^1)^2 + \cdots (\theta^{\mathsf{T}}x^n - y^n)^2\right\}$$

$$= \frac{1}{n}\left\|\begin{bmatrix}(x^1)^{\mathsf{T}}\theta - y^1\\ \vdots \\ (x^n)^{\mathsf{T}}\theta - y^n\end{bmatrix}\right\|^2$$

$$= \frac{1}{n}\left\|\underbrace{\begin{bmatrix}(x^1)^{\mathsf{T}}\\ \vdots \\ (x^n)^{\mathsf{T}}\end{bmatrix}}_{X}\theta - \underbrace{\begin{bmatrix}y^1\\ \vdots \\ y^n\end{bmatrix}}_{y}\right\|^2 = \frac{1}{n}\|X\theta - y\|^2$$

- ERM is a *least squares problem*: choose θ to minimize $\|X\theta - y\|^2$ (factor $1/n$ doesn't affect choice of θ)

Least squares solution

- assuming X has linearly independent columns (which implies $n \geq d$), there is a unique optimal θ

$$\theta^\star = (X^\mathsf{T} X)^{-1} X^\mathsf{T} y = X^\dagger y$$

- standard algorithm:
  - compute QR factorization $X = QR$ (*e.g.*, Gram-Schmidt) (with orthogonal Q and invertible upper triangular R)
  - compute $Q^\mathsf{T} y$
  - solve $R\theta^\star = Q^\mathsf{T} y$ by back substitution
- in Julia: `theta_opt = X\y`
- complexity is $2d^2 n$ flops

Data matrix

- the $n \times d$ matrix
$$X = \begin{bmatrix} (x^1)^\mathsf{T} \\ \vdots \\ (x^n)^\mathsf{T} \end{bmatrix}$$
is called the *data matrix*
- ith row of X is ith feature vector, transposed
- jth column of X gives values of jth feature x_j across our data set
- X_{ij} is the value of jth feature for the ith data point

Constant fit

- the simplest feature vector is constant: $x = \phi(u) = 1$ (doesn't depend on u!)
- corresponding predictor is a constant function: $g(x) = \theta_1$
- data matrix is $X = \mathbf{1}_n$
- so $X^\dagger = (X^\top X)^{-1} X^\top = (1/n)\mathbf{1}^\top$ and
$$\theta^\star = X^\dagger y = \mathbf{1}^\top y/n = \mathrm{avg}(y)$$
- *the average of the outcome values is the best constant predictor* (for square loss)
- optimal RMSE is standard deviation of outcome values
$$\left(\frac{1}{n}\sum_{i=1}^{n}(\mathrm{avg}(y) - y^i)^2\right)^{1/2}$$

Regression

- with $u \in \mathbf{R}^{d-1}$: $x = \phi(u) = (1, u)$
- same as $x_1 = 1$ (the first feature is constant)
- predictor has form

$$\hat{y} = \theta^\mathsf{T} x = \theta_1 + \theta_{2:d}^\mathsf{T} u$$

  an affine function of u

Straight line fit

- with $u \in \mathbf{R}$, $x = (1, u) \in \mathbf{R}^2$
- model is $\hat{y} = g(x) = \theta_1 + \theta_2 u$
- this model is called *straight-line fit*
- when u is time, it's called the *trend line*
- when u is the whole market return, and y is an asset return, θ_2 is called 'β'

Constant versus straight-line fit models

- for the constant model, we choose θ_1 to minimize

$$\frac{1}{n}\sum_{i=1}^{n}(\theta_1 - y^i)^2$$

- for the straight-line model, we choose (θ_1, θ_2) to minimize

$$\frac{1}{n}\sum_{i=1}^{n}(\theta_1 + \theta_2 u^i - y^i)^2$$

- for optimal choices, this value is less than or equal to the one above (since we can take $\theta_2 = 0$ in the straight-line model)
- so the RMS error of the straight-line fit is no more than the standard deviation

Regularization

Sensitivity

- we have a linear predictor $\hat{y} = g(x) = \theta^\mathsf{T} x$
- if $|\theta_i|$ is large, then the prediction is very sensitive to x_i
  (*i.e.*, small changes in x_i lead to large changes in the prediction)
- large sensitivity can lead to overfit, poor generalization
  (which would turn up in validation)
- for $x_1 = 1$ (the constant feature), there is no sensitivity, since the feature does not change
- suggests that we would like θ (or $\theta_{2:d}$ if $x_1 = 1$) not too large

Regularizer

- we will measure the size of θ using a *regularizer* function $r : \mathbf{R}^d \rightarrow \mathbf{R}$
- $r(\theta)$ is a measure of the size of θ (or $\theta_{2:d}$)

- *quadratic regularizer* (a.k.a. ℓ_2 or sum-of-squares):
$$r(\theta) = ||\theta||^2 = \theta_1^2 + \cdots + \theta_d^2$$
- *absolute value regularizer* (a.k.a. ℓ_1):
$$r(\theta) = ||\theta||_1 = |\theta_1| + \cdots + |\theta_d|$$

Regularized empirical risk minimization

- predictor should fit the given data well, *i.e.*, we want empirical risk

  $$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(\theta^{\mathsf{T}} x^i, y^i)$$

  to be small

- predictor should not be too sensitive, *i.e.*, we want $r(\theta)$ small

- to trade off these two objectives, form *regularized empirical risk*

  $$\mathcal{L}(\theta) + \lambda r(\theta)$$

  where $\lambda \geq 0$ is the *regularization parameter* (or *hyper-parameter*)

- *regularized empirical risk minimization* (RERM): choose θ to minimize regularized empirical risk

- an optimization problem

Regularized empirical risk minimization

- for $\lambda = 0$, RERM reduces to ERM
- RERM produces a *family* of predictors, one for each value of λ
- in practice, we choose a few tens of values of λ, usually logarithmically spaced over a wide range
- use validation to choose among the candidate predictors
- we choose the largest value of λ that gives near minimum test error (*i.e.*, least sensitive predictor that generalizes well)

Ridge regression

- *ridge regression*: square loss and regularizer $r(\theta) = ||\theta||^2$ (or $||\theta_{2:d}||^2$ if $x_1 = 1$)
- also called *Tykhonov regularized least squares*
- regularized empirical risk is

$$\begin{aligned} \mathcal{L}(\theta) + \lambda r(\theta) &= ||X\theta - y||^2 + \lambda ||\theta||^2 \\ &= \left\| \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} \theta - \begin{bmatrix} y \\ 0 \end{bmatrix} \right\|^2 \end{aligned}$$

- so optimal θ is

$$\theta^\star = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix}^\dagger \begin{bmatrix} y \\ 0 \end{bmatrix} = (X^T X + \lambda I)^{-1} X^T y$$

- (how do you modify this to handle $r(\theta) = ||\theta_{2:d}||^2$?)

Penalty functions and error histograms

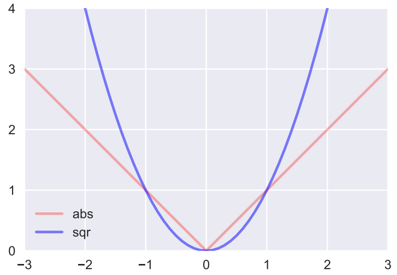
Loss and penalty functions

- empirical risk (or average loss) is $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(\theta^\mathsf{T} x^i, y^i)$
- the loss function $\ell(\hat{y}, y)$ penalizes deviation between the predicted value $\hat{y}$ and the observed value y
- common form for loss function: $\ell(\hat{y}, y) = p(\hat{y} - y)$
- p is the *penalty function*
- *e.g.*, the square penalty $p^{\mathsf{sqr}}(r) = r^2$
- $r = \hat{y} - y$ is the *prediction error* or *residual*

Penalty functions

- the penalty function tells us how much we object to different values of prediction error
- usually $p(0) = 0$ and $p(r) \geq 0$ for all r
- if p is *symmetric*, *i.e.*, $p(-r) = p(r)$, we care only about the magnitude (absolute value) of prediction error
- if p is *asymmetric*, *i.e.*, $p(-r) \neq p(r)$, it bothers us more to over- or underestimate

Square versus absolute value penalty



- for square penalty $p^{\mathsf{sqr}}(r) = r^2$
  - for small prediction errors, penalty is very small (small squared)
  - for large prediction errors, penalty is very large (large squared)
- for absolute penalty $p^{\mathsf{abs}}(r) = |r|$
  - for small prediction errors, penalty is large (compared to square)
  - for large prediction errors, penalty is small (compared to square)

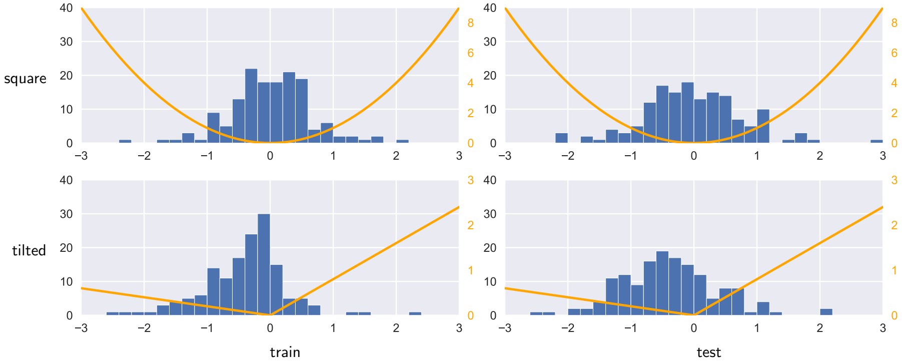
Predictors and choice of penalty function

- choice of penalty function depends on how you feel about large, small, positive, or negative prediction errors
- different choices of penalty function yield different predictor parameters
- choice of penalty function *shapes* the histogram of prediction errors, *i.e.*,

$$r^1, \dots, r^n$$

(usually divided into bins and displayed as bar graph distribution)

Histogram of residuals



- artificial data with $n = 300$ and $d = 30$, using 50/50 test/train split
- plots show histogram of residuals $r^1, \ldots, r^n$
- tilted loss results in distribution with most residuals $r^i < 0$, *i.e.*, predictor prefers $\hat{y}^i < y^i$

Robust fitting

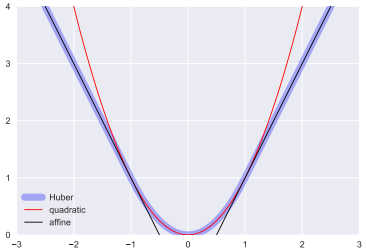
Outliers

- in some applications, a few data points are ‘way off’, or just ‘wrong’
- occurs due to transcription errors, error in decimal point position, *etc.*
- these points are called *outliers*
- even a few outliers in a data set can result in a poor predictor
- several standard methods are used to remove outliers, or reduce their impact
- one simple method:
  - create predictor from data set
  - flag data points with large prediction errors as outliers
  - remove them from the data set and repeat

Robust penalty functions

- we say a penalty function is *robust* if it has low sensitivity to outliers
- robust penalty functions grow more slowly for large prediction error values than the square penalty
- and so 'allow' the predictor to have a few large prediction errors (presumably for the outliers)
- so they handle outliers more gracefully
- a *robust predictor* might fit, *e.g.*, 98% of the data very well

Huber loss



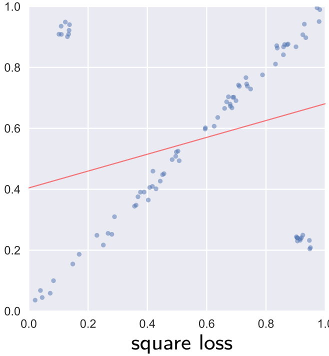
- the *Huber* penalty function is

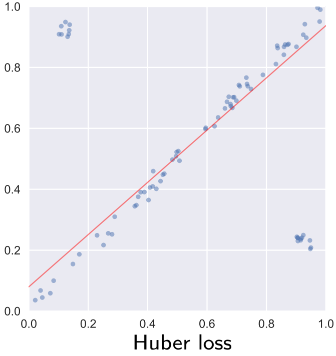
$$p^{\text{hub}}(r) = \begin{cases} r^2 & \text{if } |y| \le \alpha \\ \alpha(2|r| - \alpha) & \text{if } |r| > \alpha \end{cases}$$

- α is a parameter
- quadratic for small r, affine for large r

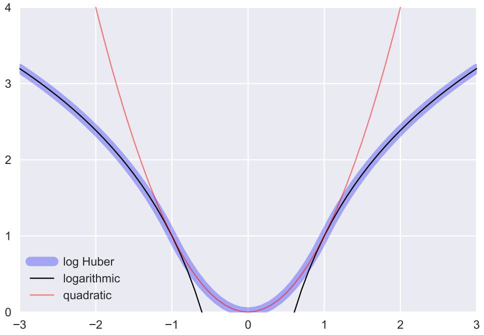
Huber loss

- linear growth for large r makes fit less sensitive to outliers
- ERM with Huber loss is called a *robust* prediction method





Log Huber

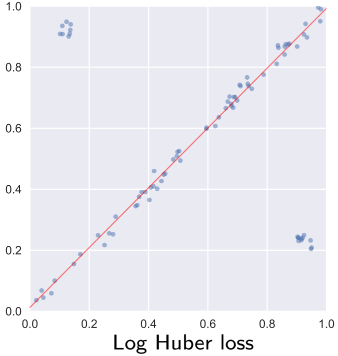


- quadratic for small y, logarithmic for large y

$$p^{\mathsf{dh}}(y) = \begin{cases} y^2 & \text{if } |y| \leq \alpha \\ \alpha^2(1 - 2\log(\alpha) + \log(y^2)) & \text{if } |y| > \alpha \end{cases}$$

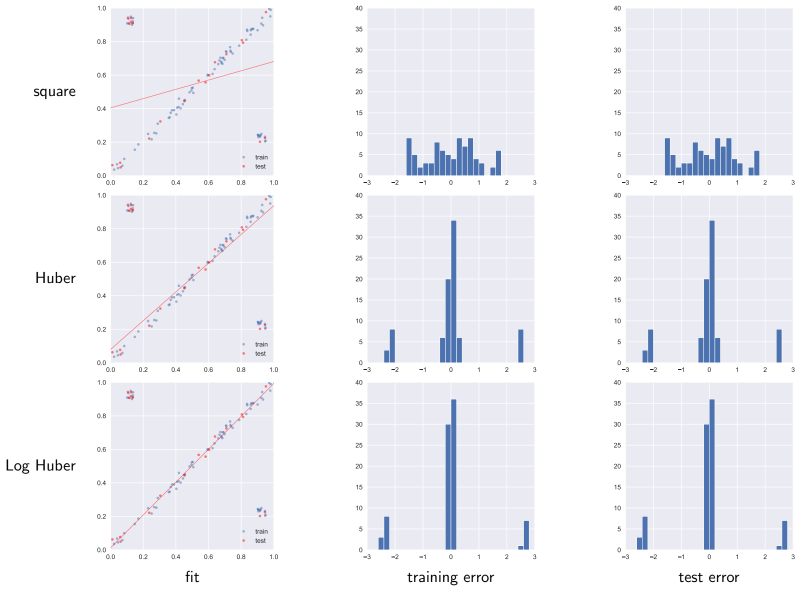
- diminishing incremental penalty at large y

Log Huber



- even less sensitive to outliers than Huber

Error distribution



Regularizers

Regularizers

- motivation:
  - large θ_i makes prediction $\theta^\mathsf{T} x$ sensitive to value of x_i
  - so we want θ (or $\theta_{2:d}$ if $x_1 = 1$) small
- regularizer $r : \mathbf{R}^d \to \mathbf{R}$ measures the size of θ
- usually regularizer is *separable*,

$$r(\theta) = q(\theta_1) + \cdots + q(\theta_d)$$

where $q : \mathbf{R} \to \mathbf{R}$ is a penalty function for the predictor coefficients

Sum squares regularizer

- *sum squares* regularizer uses square penalty $q^{\mathsf{sqr}}(a) = a^2$

$$r(\theta) = ||\theta||^2 = \theta_1^2 + \cdots + \theta_d^2$$

- also called *quadratic*, *Tychonov*, or ℓ_2 regularizer

ℓ_1 regularizer

- *sum absolute* or ℓ_1 regularizer uses absolute value penalty $q^{\text{abs}}(a) = |a|$

$$r(\theta) = ||\theta||_1 = |\theta_1| + \cdots + |\theta_d|$$

- $||\theta||_1$ is ℓ_1 *norm* of θ
- like the Euclidean or ℓ_2 norm $||\theta||$, it is a norm, *i.e.*, a measure of the size of the vector θ
- Euclidean norm is often written as $||\theta||_2$ to distinguish it from the ℓ_1 norm
- they are both members of the *p-norm family*, defined as

$$||\theta||_p = \left(|\theta_1|^p + \cdots + |\theta_d|^p\right)^{1/p}$$

  for $p \geq 1$

Lasso regression

- use square loss $\ell(\hat{y}, y) = (\hat{y} - y)^2$
- choosing θ to minimize $\mathcal{L}(\theta) + \lambda||\theta||_2^2$ is called *ridge regression*
- choosing θ to minimize $\mathcal{L}(\theta) + \lambda||\theta||_1$ is called *lasso regression*
- invented by (Stanford's) Rob Tibshirani, 1994
- widely used in advanced machine learning
- unlike ridge regression, there is no formula for the lasso parameter vector
- but we can efficiently compute it anyway (since it's convex)
- the lasso regression model has some interesting properties

Sparsifying regularizers

Sparse coefficient vector

- suppose θ is sparse, *i.e.*, many of its entries are zero
- prediction $\theta^{\mathsf{T}}x$ does not depend on features x_i for which $\theta_i = 0$
- this means we select *some* features to use (*i.e.*, those with $\theta_i \neq 0$)
- (possible) practical benefits of sparse θ:
  - can improve performance when many features are actually irrelevant
  - makes predictor *simpler to interpret*

Sparse coefficient vectors via ℓ_1 regularization

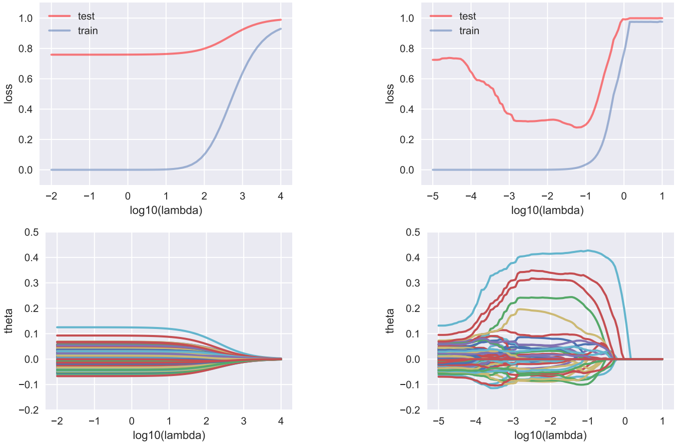
using ℓ_1 regularization leads to sparse coefficient vectors

$r(\theta) = ||\theta||_1$ is called a *sparsifying regularizer*

rough explanation:

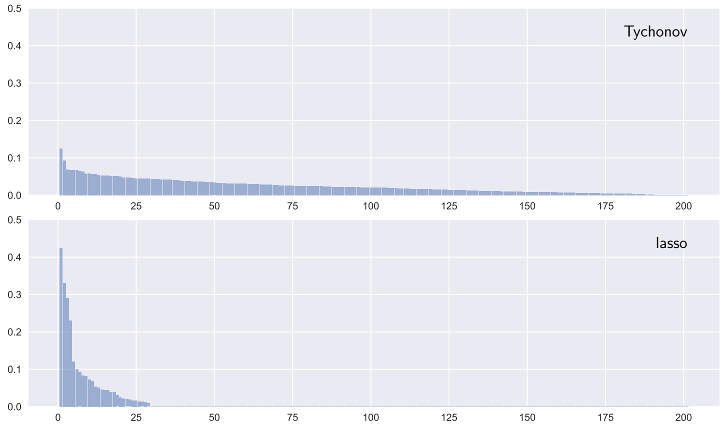
- for square penalty, once θ_i is small, θ_i^2 is very small
- so incentive for sum squares regularizer to make a coefficient smaller decreases once it is small
- for absolute penalty, incentive to make θ_i smaller keeps up all the way until it's zero

Example



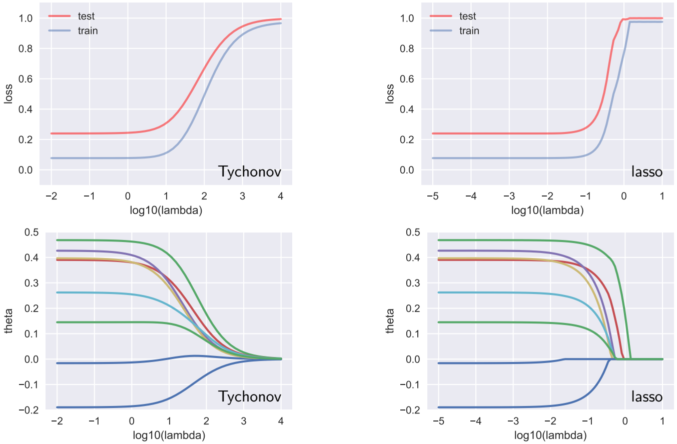
- artificially generated 50 data points, 200 features
- only a few features are relevant
- left hand plots use Tychonov, right hand use lasso

Example



- sorted $|\theta_i|$ at optimal λ
- lasso solution has only 35 non-zero components

Example



- choose λ based on regularization path with test data
- keep features corresponding to largest components of θ and *retrain*
- plots above use most important 7 features identified by lasso

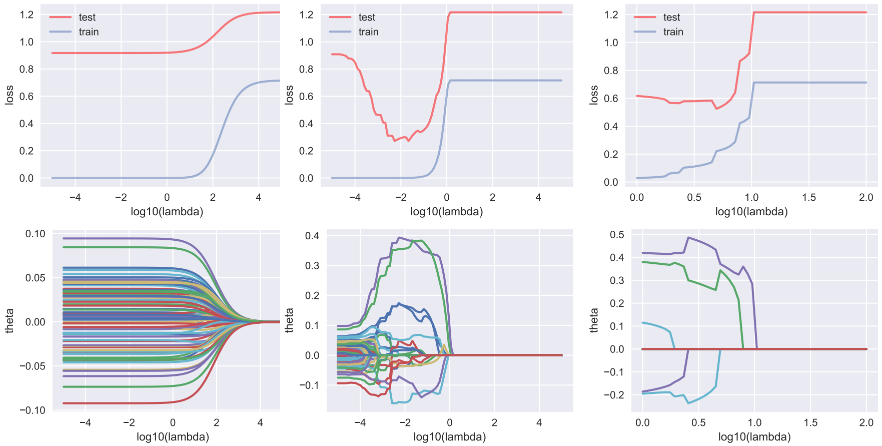
Even stronger sparsifiers

- $q(a) = |a|^{1/2}$
- called $\ell_{0.5}$ regularizer
- but you shouldn't use this term since

$$\left(|\theta_1|^{0.5} + \cdots + |\theta_d|^{0.5}\right)^2$$

  is not a norm (see VMLS)
- 'stronger' sparsifier than ℓ_1
- but not convex so computing θ is heuristic

Example



- ℓ_2, ℓ_1, and square root regularization

Nonnegative regularizer

Nonegative coefficients

- in some cases we know or require that $\theta_i \geq 0$
- this means that when x_i increases, so must our prediction
- we can think of this constraint as regularization with penalty function

$$q(a) = \begin{cases} 0 & a \geq 0 \\ \infty & a < 0 \end{cases}$$

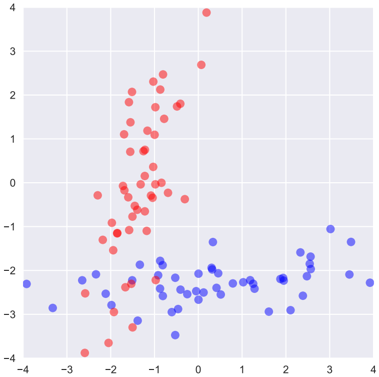
- example: y is lifespan, x_i measures healthy behavior i
- with quadratic loss, called *nonnegative least squares* (NNLS)
- common heuristic for nonnegative least squares: use $(\theta^{\text{ls}})_+$ (works poorly)

Classification

Boolean classification

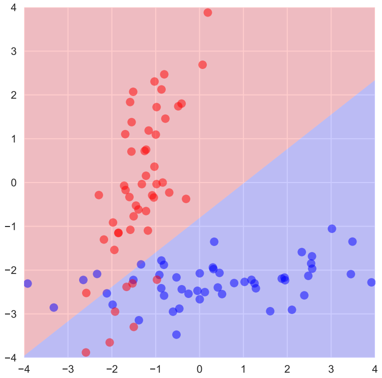
- supervised learning is called *boolean classification* when raw output variable v is a categorical that can take two possible values
- we denote these -1 and 1, and they often correspond to $\{\text{FALSE}, \text{TRUE}\}$ or $\{\text{NEGATIVE}, \text{POSITIVE}\}$
- for a data record u^i, v^i, the value $v^i \in \{-1, 1\}$ is called the *class* or *label*
- a *boolean classifier* predicts label $\hat{v}$ given raw input u

Classification



- here $u \in \mathbf{R}^2$
- red points have $v^i = -1$, blue points have $v^i = 1$
- we'd like a predictor that maps any $u \in \mathbf{R}^2$ into prediction $\hat{v} \in \{-1, 1\}$

Example: Least squares classifier



- embed $x = (1, u)$ and $y = v$, apply least squares regression
- gives $\hat{y} = \theta_1 + \theta_2 u_1 + \theta_3 u_2$
- predict using $\hat{v} = \text{sign}(\hat{y})$
- 11% of points misclassified at training

Confusion matrix

The two types of errors

- measure performance of a specific predictor on a set of n data records
- each data point i has $v^i \in \{-1, 1\}$
- and corresponding prediction $\hat{v}^i = g(v^i) \in \{-1, 1\}$
- only four possible values for the data pair $\hat{v}^i$, v^i:
    - *true positive* if $\hat{v}^i = 1$ and $v^i = 1$
    - *true negative* if $\hat{v}^i = -1$ and $v^i = -1$
    - *false negative* or *type II error* if $\hat{v}^i = -1$ and $v^i = 1$
    - *false positive* or *type I error* if $\hat{v}^i = 1$ and $v^i = -1$

Confusion matrix

- for a predictor and a data set define the *confusion matrix*

$$C = \begin{bmatrix} \#\text{ true negatives} & \#\text{ false negatives} \\ \#\text{ false positives} & \#\text{ true positives} \end{bmatrix} = \begin{bmatrix} C_{\mathsf{tn}} & C_{\mathsf{fn}} \\ C_{\mathsf{fp}} & C_{\mathsf{tp}} \end{bmatrix}$$

  (warning: some people use the transpose of C)

- $C_{\mathsf{tn}} + C_{\mathsf{fn}} + C_{\mathsf{fp}} + C_{\mathsf{tp}} = n$ (total number of examples)
- $N_{\mathsf{n}} = C_{\mathsf{tn}} + C_{\mathsf{fp}}$ is number of negative examples
- $N_{\mathsf{p}} = C_{\mathsf{fn}} + C_{\mathsf{tp}}$ is number of positive examples
- diagonal entries give numbers of correct predictions
- off-diagonal entries give numbers of incorrect predictions of the two types

Some boolean classification measures

- confusion matrix $\begin{bmatrix} C_{\mathsf{tn}} & C_{\mathsf{fn}} \\ C_{\mathsf{fp}} & C_{\mathsf{tp}} \end{bmatrix}$

- the basic error measures:
  - *false positive rate* is C_{fp}/n
  - *false negative rate* is C_{fn}/n
  - *error rate* is $(C_{\mathsf{fn}} + C_{\mathsf{fp}})/n$

- error measures some people use:
  - *true positive rate* or *sensitivity* or *recall* is $C_{\mathsf{tp}}/N_{\mathsf{p}}$
  - *false alarm rate* is $C_{\mathsf{fp}}/N_{\mathsf{n}}$
  - *specificity* or *true negative rate* is $C_{\mathsf{tn}}/N_{\mathsf{n}}$
  - *precision* is $C_{\mathsf{tp}}/(C_{\mathsf{tp}} + C_{\mathsf{fp}})$

Neyman-Pearson error

- *Neyman-Pearson error* over a data set is $\kappa C_{\mathsf{fn}}/n + C_{\mathsf{fp}}/n$
- a scalarization of our two objectives, false positive and false negative rates
- κ is how much more false negatives irritate us than false positives
- when $\kappa = 1$, the Neyman-Pearson error is the *error rate*

- we'll use the Neyman-Pearson error as our scalarized measure

ERM for classification tasks

Embedding

- we embed raw input and output records as $x = \phi(u)$ and $y = \psi(v)$
- ϕ is the feature map
- ψ is *the identity map*, $\psi(v) = v$
- un-embed by $\hat{v} = \mathrm{sign}(\hat{y})$
- equivalent to $\hat{v} = \underset{v \in \{-1,1\}}{\mathrm{argmin}} \, |\hat{y} - \psi(v)|$
- *i.e.*, choose the nearest boolean value to the (real) prediction $\hat{y}$

ERM

- given loss function $\ell(\hat{y}, y)$, *empirical risk* on a data set is

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \ell(\hat{y}^i, y^i)$$

- for linear model $\hat{y} = \theta^\mathsf{T} x$, with $\theta \in \mathbf{R}^d$,

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(\theta^\mathsf{T} x^i, y^i)$$

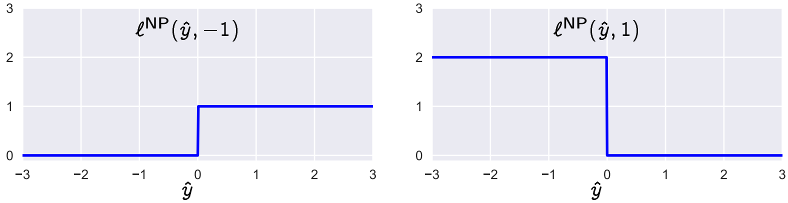
- ERM: choose θ to minimize $\mathcal{L}(\theta)$
- regularized ERM: choose θ to minimize $\mathcal{L}(\theta) + \lambda r(\theta)$, with $\lambda > 0$

Loss functions for boolean classification

- to apply ERM, we need a loss function on embedded variables $\ell(\hat{y}, y)$
- y can only take values -1 or 1
- but $\hat{y} = \theta^\mathsf{T} x \in \mathbf{R}$ can be any real number
- to specify ℓ, we only need to give two functions (of a scalar $\hat{y}$):
  - $\ell(\hat{y}, -1)$ is how much $\hat{y}$ irritates us when $y = -1$
  - $\ell(\hat{y}, 1)$ is how much $\hat{y}$ irritates us when $y = 1$
- we can take $\ell(\hat{y}, 1) = \kappa \ell(-\hat{y}, -1)$, to reflect that false negatives irritate us a factor κ more than false positives

Neyman-Pearson loss

- Neyman-Pearson loss is
  - $\ell^{\mathrm{NP}}(\hat{y}, -1) = \begin{cases} 1 & \hat{y} \geq 0 \\ 0 & \hat{y} < 0 \end{cases}$
  - $\ell^{\mathrm{NP}}(\hat{y}, 1) = \kappa l^{\mathrm{NP}}(\hat{y}, -1) = \begin{cases} \kappa & \hat{y} < 0 \\ 0 & \hat{y} \geq 0 \end{cases}$
- empirical Neyman-Pearson risk $\mathcal{L}^{\mathrm{NP}}$ is the Neyman-Pearson error



The problem with Neyman-Pearson loss

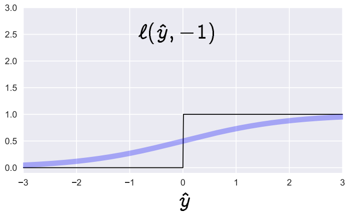
- empirical Neyman-Pearson risk $\mathcal{L}^{\mathtt{NP}}(\theta)$ is not differentiable, or even continuous (and certainly not convex)
- worse, its gradient $\nabla\mathcal{L}^{\mathtt{NP}}(\theta)$ is either zero or undefined
- so an optimizer does not know how to improve the predictor

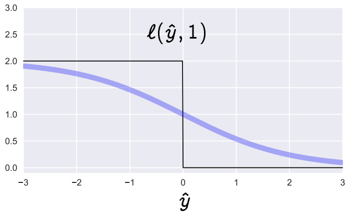
Idea of proxy loss

- we get better results using a *proxy loss* that
  - approximates, or at least captures the flavor of, the Neyman-Pearson loss
  - is more easily optimized (*e.g.*, is convex or has nonzero derivative)

- we want a proxy loss function
  - with $\ell(\hat{y}, -1)$ small when $\hat{y} < 0$, and larger when $\hat{y} > 0$
  - with $\ell(\hat{y}, +1)$ small when $\hat{y} > 0$, and larger when $\hat{y} < 0$
  - which has other nice characteristics, *e.g.*, differentiable or convex

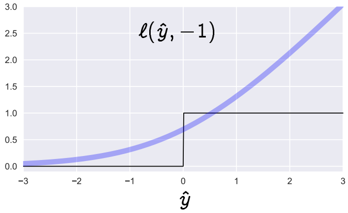
Sigmoid loss

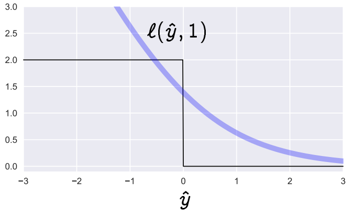




- $\ell(\hat{y}, -1) = \dfrac{1}{1 + e^{-\hat{y}}}, \quad \ell(\hat{y}, 1) = \kappa \ell(-\hat{y}, -1) = \dfrac{\kappa}{1 + e^{\hat{y}}}$
- differentiable approximation of Neyman-Pearson loss
- but not convex

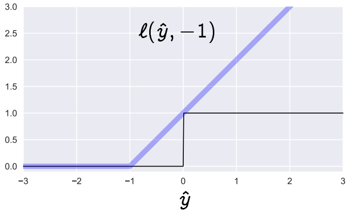
Logistic loss

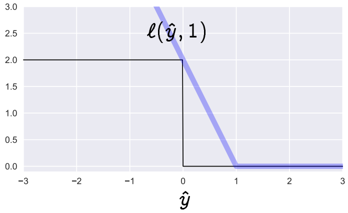




- $\ell(\hat{y}, -1) = \log(1 + e^{\hat{y}}), \quad \ell(\hat{y}, 1) = \kappa \ell(-\hat{y}, -1) = \kappa \log(1 + e^{-\hat{y}})$
- differentiable and convex approximation of Neyman-Pearson loss

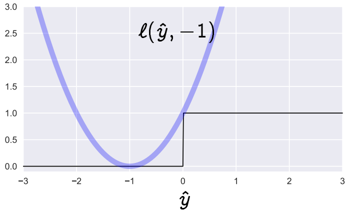
Hinge loss

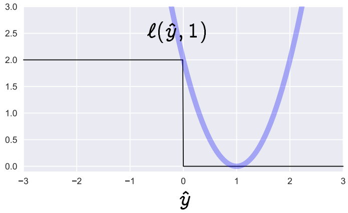




- $\ell(\hat{y}, -1) = (1 + \hat{y})_+, \quad \ell(\hat{y}, 1) = \kappa \ell(-\hat{y}, -1) = \kappa(1 - \hat{y})_+$
- nondifferentiable but convex approximation of Neyman-Pearson loss

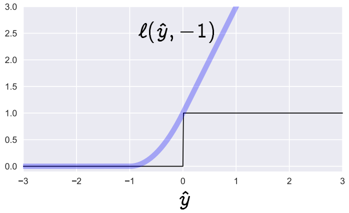
Square loss

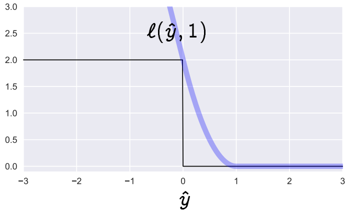




- $\ell(\hat{y}, -1) = (1+\hat{y})^2, \quad \ell(\hat{y}, 1) = \kappa \ell(-\hat{y}, -1) = \kappa(1-\hat{y})^2$
- ERM is least squares problem

Hubristic loss





- define the *hubristic loss* (huber + logistic) as

$$\ell(\hat{y}, -1) = \begin{cases} 0 & \hat{y} < -1 \\ (\hat{y}+1)^2 & -1 \leq \hat{y} \leq 0 \\ 1 + 2\hat{y} & \hat{y} > 0 \end{cases}$$

- $\ell(\hat{y}, 1) = \kappa \ell(-\hat{y}, -1)$

Boolean classifiers

Least squares classifier

- use empirical risk with square loss

$$\mathcal{L}(\theta) = \frac{1}{n}\left(\sum_{i:y^i=-1} (1+\hat{y}^i)^2 \; + \; \kappa \sum_{i:y^i=1} (1-\hat{y}^i)^2 \right)$$

  and your choice of regularizer

- with sum squares regularizer, this is *least squares classifier*

- we can minimize $\mathcal{L}(\theta) + \lambda r(\theta)$ using, *e.g.*, QR factorization

Logistic regression

- use empirical risk with logistic loss

$$\mathcal{L}(\theta) = \frac{1}{n}\left(\sum_{i:y^i=-1} \log(1+e^{\hat{y}^i}) \ + \ \kappa \sum_{i:y^i=1} \log(1+e^{-\hat{y}^i})\right)$$

  and your choice of regularizer

- can minimize $\mathcal{L}(\theta)+\lambda r(\theta)$ using prox-gradient method
- we will find an actual minimizer if r is convex

Support vector machine

(usually abbreviated as *SVM*)

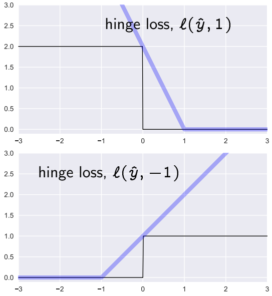
- use empirical risk with hinge loss

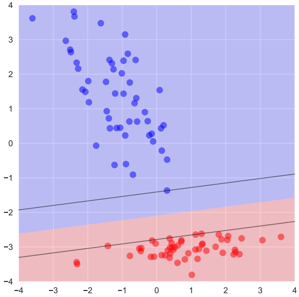
$$\mathcal{L}(\theta) = \frac{1}{n}\left(\sum_{i:y^i=-1} (1+\hat{y}^i)_+ \; + \; \kappa \sum_{i:y^i=1} (1-\hat{y}^i)_+ \right)$$

  and sum squares regularizer

- $\mathcal{L}(\theta) + \lambda r(\theta)$ is convex
- it can be minimized by various methods (but not prox-gradient)

Support vector machine





- decision boundary is $\theta^{\mathsf{T}}x = 0$
- black lines show points where $\theta^{\mathsf{T}}x = \pm 1$
- what is the training risk here?

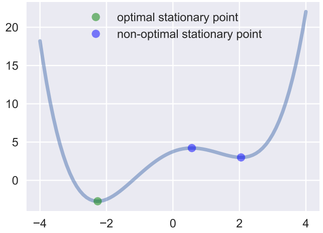
Optimization problems and algorithms

Optimization problem

$$\text{minimize} \quad f(\theta)$$

- $\theta \in \mathbf{R}^d$ is the *variable* or *decision variable*
- $f : \mathbf{R}^d \to \mathbf{R}$ is the *objective function*
- goal is to choose θ to minimize f
- $\theta^\star$ is *optimal* means that for all θ, $f(\theta) \geq f(\theta^\star)$
- $f^\star = f(\theta^\star)$ is the *optimal value* of the problem
- optimization problems arise in many fields and applications, including machine learning

Optimality condition



- let's assume that f is *differentiable*, *i.e.*, partial derivatives $\frac{\partial f(\theta)}{\partial \theta_i}$ exist
- if $\theta^\star$ is optimal, then $\nabla f(\theta^\star) = 0$
- $\nabla f(\theta) = 0$ is called the *optimality condition* for the problem
- there can be points that satisfy $\nabla f(\theta) = 0$ but are not optimal
- we call points that satisfy $\nabla f(\theta) = 0$ *stationary points*
- not all stationary points are optimal

Solving optimization problems

- in some cases, we can solve the problem analytically
- *e.g.*, least squares: minimize $f(\theta) = \|X\theta - y\|^2$
  - optimality condition is $\nabla f(\theta) = 2X^T(X\theta - y) = 0$
  - this has (unique) solution $\theta^\star = (X^TX)^{-1}X^Ty = X^\dagger y$
    (when columns of X are linearly independent)
- in other cases, we resort to an *iterative algorithm* that computes a sequence $\theta^1, \theta^2, \ldots$ with, hopefully, $f(\theta^k) \to f^\star$ as $k \to \infty$

Iterative algorithms

- *iterative algorithm* computes a sequence $\theta^1, \theta^2, \ldots$
- θ^k is called the kth *iterate*
- θ^1 is called the *starting point*
- many iterative algorithms are *descent methods*, which means

  $$f(\theta^{k+1}) < f(\theta^k), \quad k = 1, 2, \ldots$$

  i.e., each iterate is better than the previous one
- this means that $f(\theta^k)$ converges, but not necessarily to $f^\star$

Stopping criterion

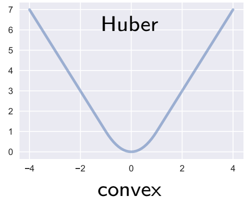
- in practice, we stop after a finite number K of steps
- typical stopping criterion: stop if $\|\nabla f(\theta^k)\| \leq \epsilon$ or $k = k^{\mathsf{max}}$
- ϵ is a small positive number, the *stopping tolerance*
- k^{max} is the maximum number of iterations
- in words: we stop when θ^k is almost a stationary point
- we hope that $f(\theta^K)$ is not too much bigger than $f^\star$
- or more realistically, that θ^K is at least useful for our application

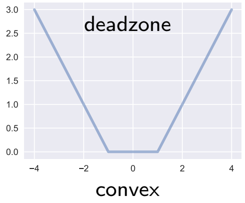
Non-heuristic and heuristic algorithms

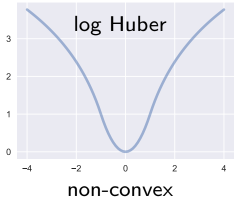
- in some cases we *know* that $f(\theta^k) \to f^\star$, for any θ^1
- in words: *we'll get to a solution if we keep iterating*
- called *non-heuristic*

- other algorithms do not guarantee that $f(\theta^k) \to f^\star$
- we can hope that even if $f(\theta^k) \not\to f^\star$, θ^k is still useful for our application
- called *heuristic*

Convex functions







- a function $f : \mathbf{R}^d \to \mathbf{R}$ is *convex* if for any θ, $\tilde{\theta}$, and α with $0 \leq \alpha \leq 1$,

$$f(\alpha\theta + (1-\alpha)\tilde{\theta}) \leq \alpha f(\theta) + (1-\alpha) f(\tilde{\theta})$$

- roughly speaking, f has 'upward curvature'
- for $d = 1$, same as $f''(\theta) \geq 0$ for all θ

Convex optimization

- optimization problem

$$\text{minimize} \quad f(\theta)$$

  is called *convex* if the objective function f is convex

- for convex optimization problem, $\nabla f(\theta) = 0$ only for θ optimal, *i.e.*, *all stationary points are optimal*

- algorithms for convex optimization are non-heuristic

- *i.e.*, *we can solve convex optimization problems* (exactly, in principle)

Convex ERM problems

- regularized empirical risk function $f(\theta) = \mathcal{L}(\theta) + \lambda r(\theta)$, with $\lambda \geq 0$,

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} p(\theta^\mathsf{T} x^i - y^i), \qquad r(\theta) = q(\theta_1) + \cdots + q(\theta_d)$$

- f is convex if loss penalty p and parameter penalty q functions are convex

- convex penalties: square, absolute, tilted absolute, Huber
- non-convex penalties: log Huber, squareroot

Gradient method

Gradient method

- assume f is differentiable
- at iteration θ^k, create affine (Taylor) approximation of f valid near θ^k
$$\hat{f}(\theta; \theta^k) = f(\theta^k) + \nabla f(\theta^k)^T(\theta - \theta^k)$$
- $\hat{f}(\theta; \theta^k) \approx f(\theta)$ for θ near θ^k
- choose θ^{k+1} to make $\hat{f}(\theta^{k+1}; \theta^k)$ small, but with $||\theta^{k+1} - \theta^k||$ not too large
- choose θ^{k+1} to minimize $\hat{f}(\theta; \theta^k) + \frac{1}{2h^k}||\theta - \theta^k||^2$
- $h^k > 0$ is a *trust parameter* or *step length* or *learning rate*
- solution is $\theta^{k+1} = \theta^k - h^k \nabla f(\theta^k)$
- roughly: take step in direction of negative gradient

Gradient method update

- choose θ^{k+1} to as minimizer of

$$f(\theta^k) + \nabla f(\theta^k)^T(\theta - \theta^k) + \frac{1}{2h^k}||\theta - \theta^k||^2$$

- rewrite as

$$f(\theta^k) + \frac{1}{2h^k}||(\theta - \theta^k) + h^k \nabla f(\theta^k)||^2 - \frac{h^k}{2}||\nabla f(\theta^k)||^2$$

- first and third terms don't depend on θ
- middle term is minimized (made zero!) by choice

$$\theta = \theta^k - h^k \nabla f(\theta^k)$$

How to choose step length

- if h^k is too large, we can have $f(\theta^{k+1}) > f(\theta^k)$
- if h^k is too small, we have $f(\theta^{k+1}) < f(\theta^k)$ but progress is slow

- a simple scheme:
  - if $f(\theta^{k+1}) > f(\theta^k)$, set $h^{k+1} = h^k/2$, $\theta^{k+1} = \theta^k$ (a *rejected step*)
  - if $f(\theta^{k+1}) \leq f(\theta^k)$, set $h^{k+1} = 1.2h^k$ (an *accepted step*)
- reduce step length by half if it's too long; increase it 20% otherwise

Gradient method summary

choose an initial $\theta^1 \in \mathbf{R}^d$ and $h^1 > 0$ (*e.g.*, $\theta^1 = 0$, $h^1 = 1$)

```
for $k = 1, 2, \ldots, k^{\mathrm{max}}$
    1. compute $\nabla f(\theta^k)$; quit if $\|\nabla f(\theta^k)\|$ is small enough
    2. form tentative update $\theta^{\mathrm{tent}} = \theta^k - h^k \nabla f(\theta^k)$
    3. if $f(\theta^{\mathrm{tent}}) \leq f(\theta^k)$, set $\theta^{k+1} = \theta^{\mathrm{tent}}$, $h^{k+1} = 1.2h^k$
    4. else set $h^k := 0.5h^k$ and go to step 2
```

Gradient method convergence

- (assuming some technical conditions hold) we have

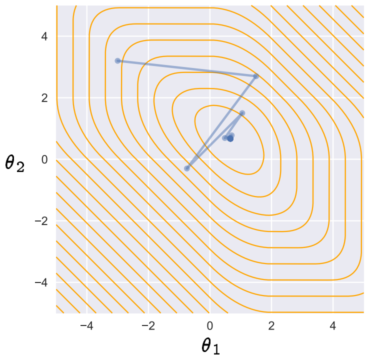
$$\|\nabla f(\theta^k)\| \to 0 \text{ as } k \to \infty$$

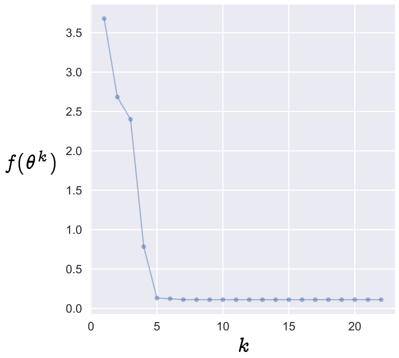
- *i.e.*, the gradient method always finds a stationary point

- for *convex problems*
  - gradient method is *non-heuristic*
  - for any starting point θ^1, $f(\theta^k) \to f^\star$ as $k \to \infty$

- for *non-convex problems*
  - gradient method is *heuristic*
  - we can (and often do) have $f(\theta^k) \not\to f^\star$

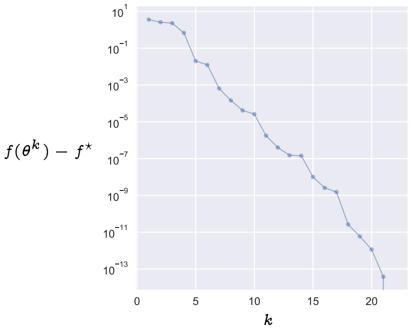
Example: Convex objective

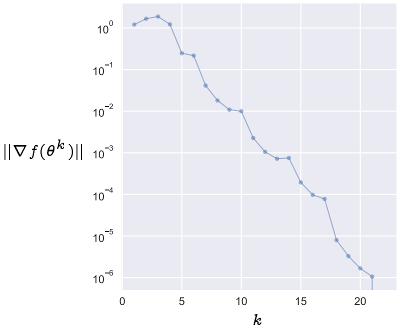




- $f(\theta) = \frac{1}{3}\left(p^{\text{hub}}(\theta_1 - 1) + p^{\text{hub}}(\theta_2 - 1) + p^{\text{hub}}(\theta_1 + \theta_2 - 1)\right)$
- f is convex
- optimal point is $\theta^\star = (2/3, 2/3)$, with $f^\star = 1/9$

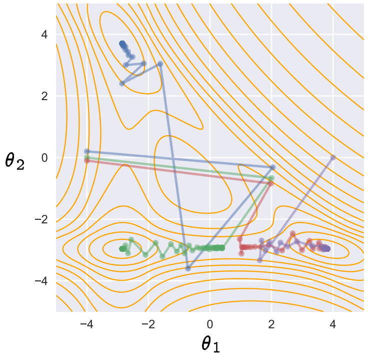
Example: Convex objective

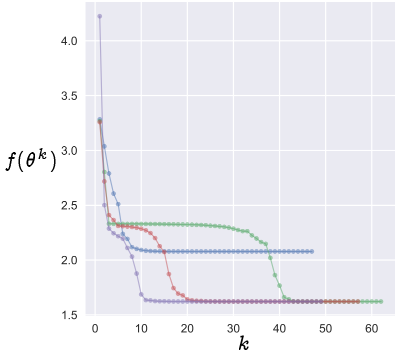




- $f(\theta^k)$ is a decreasing function of k, (roughly) exponentially
- $\|\nabla f(\theta^k)\| \to 0$ as $k \to \infty$

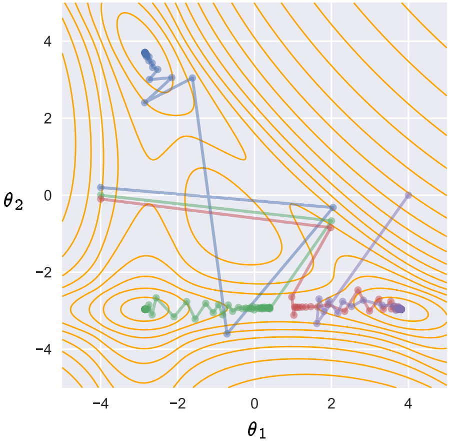
Example: Non-convex objective

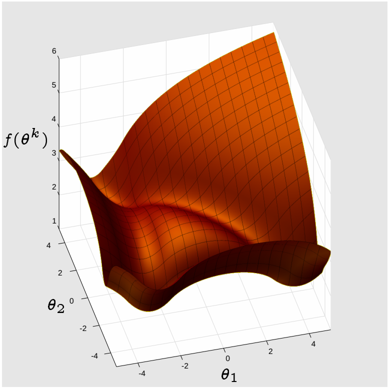




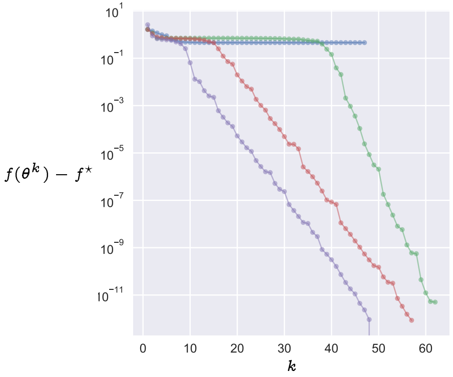
- $f(\theta) = \frac{1}{3}\left(p^{\mathrm{lh}}(\theta_1 + 3) + p^{\mathrm{lh}}(2\theta_2 + 6) + p^{\mathrm{lh}}(\theta_1 + \theta_2 - 1)\right)$
- f is sum of log-Huber functions, so not convex
- gradient algorithm converges, but limit depends on initial guess

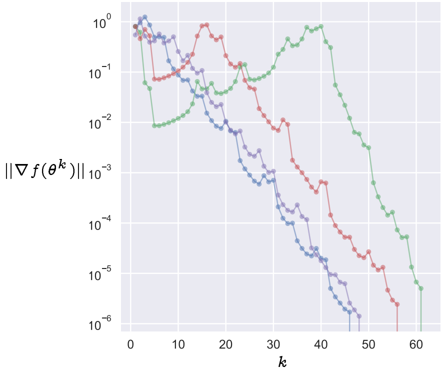
Example: Non-convex objective





Example: Non-convex objective





Gradient method for ERM

Gradient of empirical risk function

- empirical risk is sum of terms for each data point

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}^i, y^i) = \frac{1}{n}\sum_{i=1}^{n} \ell(\theta^T x^i, y^i)$$

- convex if loss function ℓ is convex in first argument
- gradient is sum of terms for each data point

$$\nabla\mathcal{L}(\theta) = \nabla\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell'(\theta^T x^i, y^i) x^i$$

where $\ell'(\hat{y}, y)$ is derivative of ℓ with respect to its first argument $\hat{y}$

Evaluating gradient of empirical risk function

- compute n-vector $\hat{y}^k = X\theta^k$
- compute n-vector z^k, with entries $z_i^k = \ell'(\hat{y}_i^k, y^i)$
- compute d-vector $\nabla\mathcal{L}(\theta^k) = (1/n)X^T z^k$

- first and third steps are matrix-vector multiplication, each costing $2nd$ flops
- second step costs order n flops (dominated by other two)
- total is $4nd$ flops

Prox-gradient method

Minimizing composite functions

- want to minimize $F(\theta) = f(\theta) + g(\theta)$ (called *composite function*)
- f is differentiable, but g need not be
- example: minimize $\mathcal{L}(\theta) + \lambda r(\theta)$, with $r(\theta) = ||\theta||_1$
- we'll see idea of gradient method extends directly to composite functions

Selective linearization

- at iteration k, linearize f *but not g*

$$\hat{F}(\theta;\theta^k) = f(\theta^k) + \nabla f(\theta^k)^T(\theta - \theta^k) + g(\theta)$$

- want $\hat{F}(\theta;\theta^k)$ small, but with θ near θ^k
- choose θ^{k+1} to minimize $\hat{F}(\theta;\theta^k) + \frac{1}{2h^k}||\theta - \theta^k||^2$, with $h^k > 0$
- same as minimizing

$$g(\theta) + \frac{1}{2h^k}||\theta - (\theta^k - h^k\nabla f(\theta^k))||^2$$

- for many 'simple' functions g, this minimization can be done analytically
- this iteration from θ^k to θ^{k+1} is called *prox-gradient step*

Prox-gradient iteration

- prox-gradient iteration has two parts:
  1. *gradient step*: $\theta^{k+1/2} = \theta^k - h^k \nabla f(\theta^k)$
  2. *prox step*: choose θ^{k+1} to minimize $g(\theta) + \frac{1}{2h^k}||\theta - \theta^{k+1/2}||^2$

  ($\theta^{k+1/2}$ is an intermediate iterate, in between θ^k and θ^{k+1})

- step 1 handles differentiable part of objective, *i.e.*, f
- step 2 handles second part of objective, *i.e.*, g

Proximal operator

- given function $q : \mathbf{R}^d \to \mathbf{R}$, and $\kappa > 0$,

$$\mathbf{prox}_{q,\kappa}(v) = \operatorname*{argmin}_{\theta} \left(q(\theta) + \frac{1}{2\kappa} ||\theta - v||^2 \right)$$

  is called the *proximal operator* of q at v, with parameter κ

- the prox-gradient step can be expressed as

$$\theta^{k+1} = \mathbf{prox}_{g,h^k}(\theta^{k+1/2}) = \mathbf{prox}_{g,h^k}(\theta^k - h^k \nabla f(\theta^k))$$

- hence the name prox-gradient iteration

How to choose step length

- same as for gradient, but using $F(\theta) = f(\theta) + g(\theta)$

- a simple scheme:
  - if $F(\theta^{k+1}) > F(\theta^k)$, set $h^{k+1} = h^k/2$, $\theta^{k+1} = \theta^k$ (a *rejected step*)
  - if $F(\theta^{k+1}) \le F(\theta^k)$, set $h^{k+1} = 1.2h^k$ (an *accepted step*)
- reduce step length by half if it's too long; increase it 20% otherwise

Stopping criterion

- stopping condition for prox-gradient method:

$$\left\| \nabla f(\theta^{k+1}) - \frac{1}{h^k}(\theta^{k+1} - \theta^{k+1/2}) \right\| \leq \epsilon$$

- analog of $\|\nabla f(\theta^{k+1})\| \leq \epsilon$ for gradient method
- second term $-\frac{1}{h^k}(\theta^{k+1} - \theta^{k+1/2})$ serves the purpose of a gradient for g (which need not be differentiable)

Prox-gradient method summary

choose an initial $\theta^1 \in \mathbf{R}^d$ and $h^1 > 0$ (*e.g.*, $\theta^1 = 0$, $h^1 = 1$)

```
for $k = 1, 2, \ldots, k^{\text{max}}$
    1. gradient step. $\theta^{k+1/2} = \theta^k - h^k \nabla f(\theta^k)$
    2. prox step. $\theta^{\text{tent}} = \operatorname{argmin}_\theta \left( g(\theta) + \frac{1}{2h^k} ||\theta - \theta^{k+1/2}||^2 \right)$
    3. if $F(\theta^{\text{tent}}) \leq F(\theta^k)$,
        (a) set $\theta^{k+1} = \theta^{\text{tent}}$, $h^{k+1} = 1.2h^k$
        (b) quit if $\left\| \nabla f(\theta^{k+1}) - \frac{1}{h^k}(\theta^{k+1} - \theta^{k+1/2}) \right\| \leq \epsilon$
    4. else set $h^k := 0.5h^k$ and go to step 1
```

Prox-gradient method convergence

- prox-gradient method always finds a stationary point
  - suitably defined for non-differentiable functions
  - assuming some technical conditions hold

- for *convex problems*
  - prox-gradient method is *non-heuristic*
  - for any starting point θ^1, $F(\theta^k) \to F^\star$ as $k \to \infty$

- for *non-convex problems*
  - prox-gradient method is *heuristic*
  - we can (and often do) have $F(\theta^k) \not\to F^\star$

Prox-gradient for regularized ERM

Prox-gradient for sum squares regularizer

- let's apply prox-gradient method to $F(\theta) = \mathcal{L}(\theta) + \lambda||\theta||_2^2$
  - $f(\theta) = \mathcal{L}(\theta)$
  - $g(\theta) = \lambda||\theta||_2^2 = \lambda\theta_1^2 + \cdots + \lambda\theta_d^2$
- in prox step, we need to minimize $\lambda\theta_i^2 + \frac{1}{2h^k}(\theta_i - \theta_i^{k+1/2})^2$ over θ_i
- solution is $\theta_i = \frac{1}{1+2\lambda h^k}\theta_i^{k+1/2}$
- so prox step just shrinks the gradient step $\theta^{k+1/2}$ by the factor $\frac{1}{1+2\lambda h^k}$

- prox-gradient iteration:
  1. gradient step: $\theta^{k+1/2} = \theta^k - h^k \nabla\mathcal{L}(\theta^k)$
  2. prox step: $\theta^{k+1} = \frac{1}{1+2\lambda h^k}\theta^{k+1/2}$

Prox-gradient for ℓ_1 regularizer

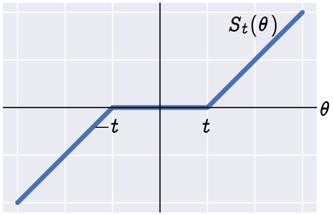
- let's apply prox-gradient method to $F(\theta) = \mathcal{L}(\theta) + \lambda||\theta||_1$
  - $f(\theta) = \mathcal{L}(\theta)$
  - $g(\theta) = \lambda||\theta||_1 = \lambda|\theta_1| + \cdots + \lambda|\theta_d|$
- in prox step, we need to minimize $\lambda|\theta_i| + \frac{1}{2h^k}(\theta_i - \theta_i^{k+1/2})^2$ over θ_i
- solution is

$$\theta_i^{k+1} = \begin{cases} \theta_i^{k+1/2} - \lambda h^k & \theta_i^{k+1/2} > \lambda h^k \\ 0 & |\theta_i^{k+1/2}| \leq \lambda h^k \\ \theta_i^{k+1/2} + \lambda h^k & \theta_i^{k+1/2} < -\lambda h^k \end{cases}$$

- called *soft threshold function*
- sometimes written as

$$\begin{aligned} \theta_i^{k+1} = S_{\lambda h^k}(\theta_i^{k+1/2}) &= \text{sign}(\theta_i^{k+1/2})(|\theta_i^{k+1/2}| - \lambda h^k)_+ \\ &= (\theta_i^{k+1/2} - \lambda h^k)_+ - (-\theta_i^{k+1/2} - \lambda h^k)_+ \end{aligned}$$

Soft threshold function



- prox-gradient iteration for regularized ERM with ℓ_1 regularization:
  1. gradient step: $\theta^{k+1/2} = \theta^k - h^k \nabla \mathcal{L}(\theta^k)$
  2. prox step: $\theta_i^{k+1} = S_{\lambda h^k}(\theta_i^{k+1/2}) \quad$ for $i = 1, \ldots, d$.
- the soft threshold step shrinks all coefficients
- and sets the small ones to zero

Prox-gradient step for nonnegative regularizer

- let's apply prox-gradient method to $F(\theta) = \mathcal{L}(\theta) + r(\theta)$, where $r(\theta) = 0$ for $\theta \geq 0$, ∞ otherwise
  - $f(\theta) = \mathcal{L}(\theta)$
  - $g(\theta) = q(\theta_1) + \cdots + q(\theta_d)$
- in prox step, we need to minimize $q(\theta_i) + \frac{1}{2h^k}(\theta_i - \theta_i^{k+1/2})^2$ over θ_i
- solution is $\theta_i = \left(\theta_i^{k+1/2}\right)_+$
- so prox step just replaces the gradient step $\theta_i^{k+1/2}$ with its positive part

- prox gradient iteration:
  1. gradient step: $\theta^{k+1/2} = \theta^k - h^k \nabla \mathcal{L}(\theta^k)$
  2. prox step: $\theta^{k+1} = \left(\theta^{k+1/2}\right)_+$