Parameter estimation
Optimization

Lecture 2c

Parametric functions

- General ML model $Y \sim Distribution(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \dots)$
  - Some of them: $Y \sim f_{\boldsymbol{\theta}}(\boldsymbol{x}) + \epsilon, \epsilon \sim Distribution(\dots)$
  - Generalization of simple models can be done

- Example: logistic $Y \sim Bernoilli\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}\right)$
  - Generalization 1: (basis function expansion):
    $Y \sim Bernoilli\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T \phi(\boldsymbol{x})}}\right)$
  - Generalization 2: $Y \sim Bernoilli\left(\frac{1}{1+e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})}}\right)$
    - $f_{\boldsymbol{\theta}}(x) = \left||\mathbf{x} - \boldsymbol{\theta}|\right|^2$

Loss minimization

- Given training set T, we **want** to minimize

$$E_{new} = \int_{(\boldsymbol{x}_*, y_*)} E\big(y_*, \hat{y}(\boldsymbol{x}_*, \mathrm{T}, \boldsymbol{\theta})\big) p(\boldsymbol{x}_*, y_*) d\boldsymbol{x}_* dy_*$$

- We **can** minimize cost function

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L\big(y_i, \hat{y}(\boldsymbol{x}_i, \boldsymbol{\theta})\big)$$

$$E_{new} \approx J(\boldsymbol{\theta})?$$

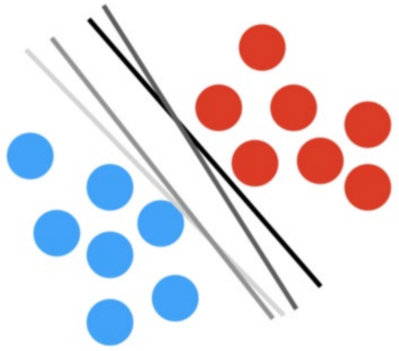
- Optimizing $J(\boldsymbol{\theta})$ does not lead to optimizing E_{new}
  - Overfitting

Loss minimization: comments

- Training a model with perfect accuracy unreasonable
  - Statistical noise for finite n

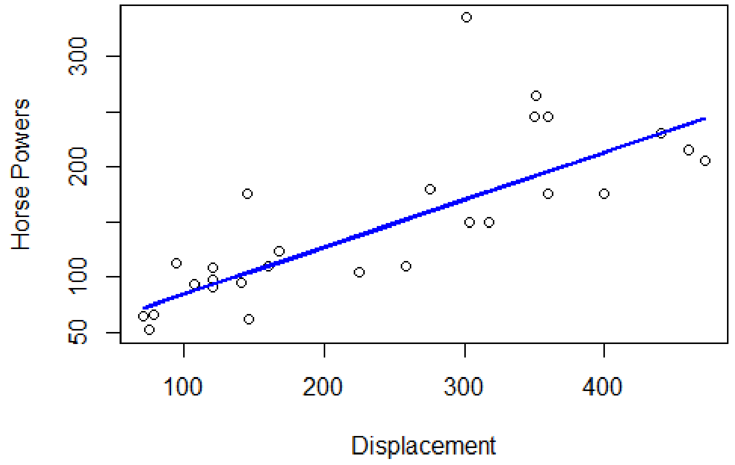
- Loss function can be different from error function

- Some loss functions are not good for training, for ex. misclass rate.



Loss functions

- Assuming a distribution, derive as minus log-likelihood:
- $y \sim Normal(f_\theta(x), \sigma^2) \rightarrow L(y, f_\theta(x)) = (y - f_\theta(x))^2$
- Heavy outliers $y \sim Laplace(f_\theta(x), \sigma^2) \rightarrow L(y, f_\theta(x)) = |y - f_\theta(x)|$



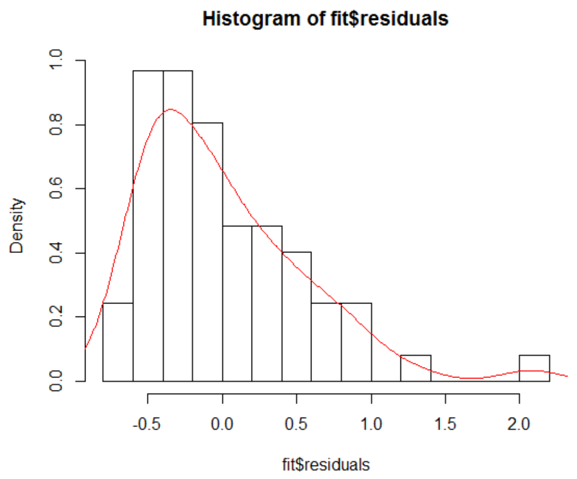
Loss functions

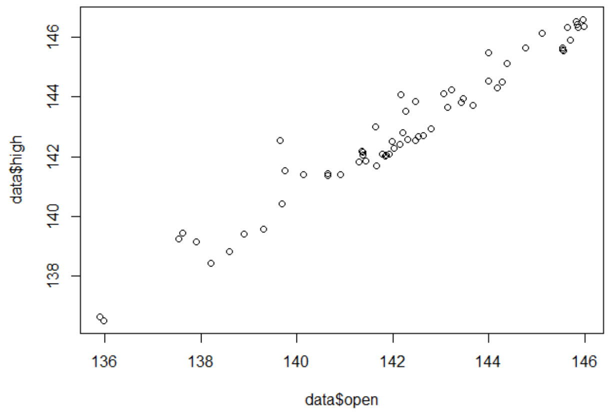
- Count data $y \sim Poisson(f_\theta(x))$

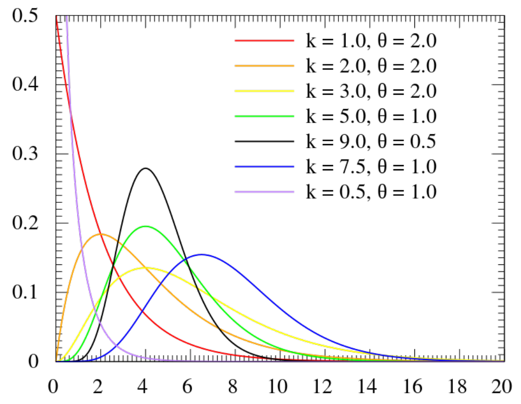
Example: Daily Stock prices NASDAQ

- Open
- High (within day)

1. Try to fit usual linear regression, study histogram of residuals







Loss functions

- If the distribution is difficult to assume / only some properties known→ **ad-hoc loss functions**

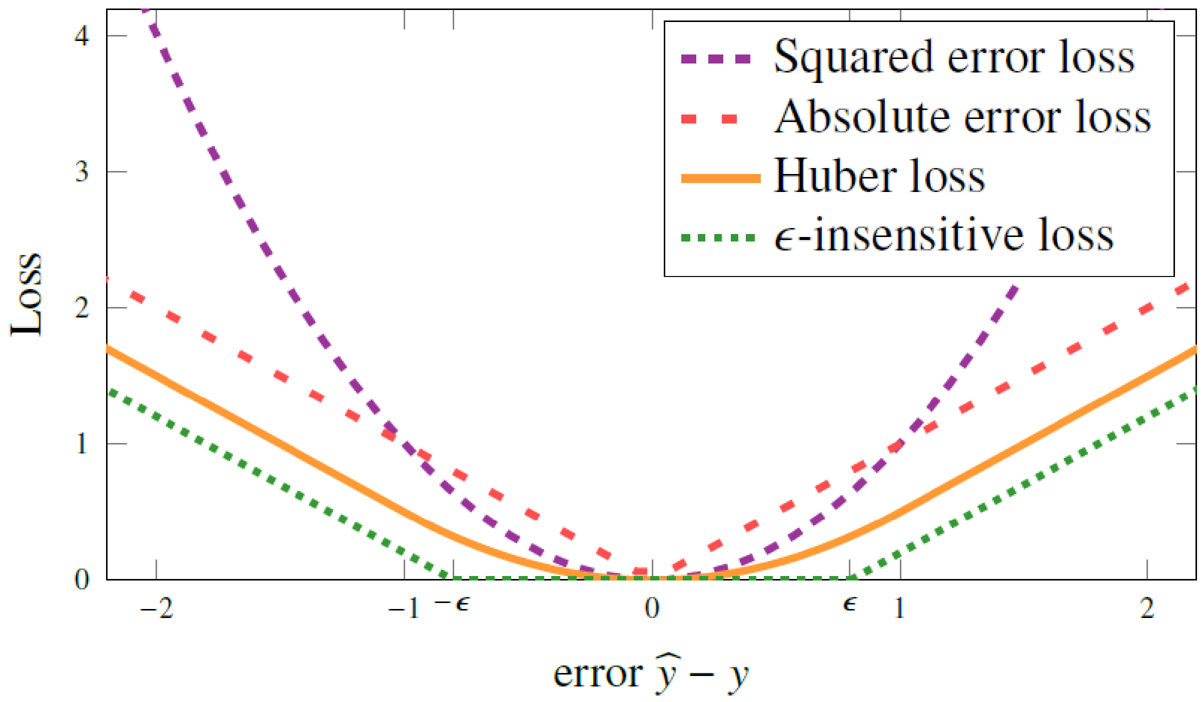
- **Huber loss**: similar to quadratic but robust to outliers

$$L(y, \widehat{y}) = \begin{cases} \frac{1}{2}(\widehat{y} - y)^2 & \text{if } |\widehat{y} - y| < 1, \\ |\widehat{y} - y| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

- **E-intensive loss**

$$L(y, \widehat{y}) = \begin{cases} 0 & \text{if } |\widehat{y} - y| < \epsilon, \\ |\widehat{y} - y| - \epsilon & \text{otherwise,} \end{cases}$$

Loss functions



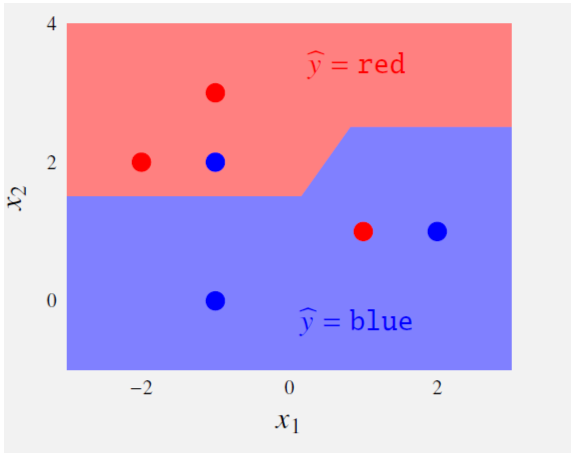
Loss functions: classification

- **Cross-entropy** corresponds to minus log-likelihood:

$$J(y, \hat{p}(y)) = -\sum_{i=1}^{n}\sum_{m=1}^{M} I(y_i = C_m) \log \hat{p}(y_i = C_m)$$

- Ad-hoc loss functions binary classification $C = \{-1,1\}$
  - Assume model returns $f(\boldsymbol{x})$: $\hat{y} = sign(f(\boldsymbol{x}))$
  - ***Example***: *logistic* $f(\boldsymbol{x}) = \frac{1}{1+e^{-\theta^T x}} - 0.5$

- **Note**: mistake when $yf(x) = -1$



Loss functions: classification

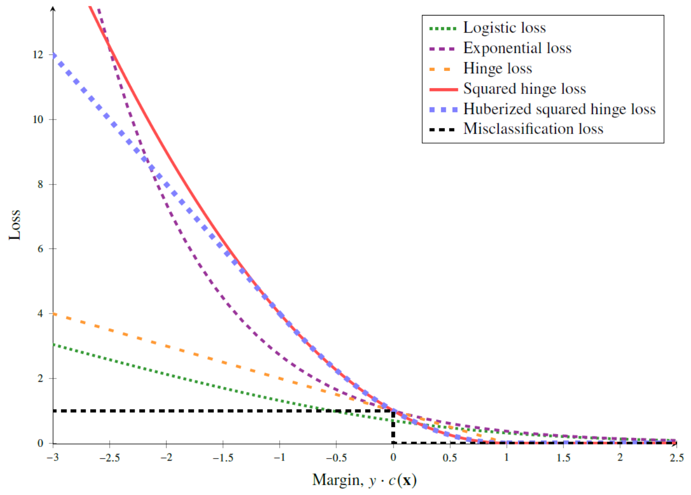
Ad-hoc loss functions binary classification

- **Exponential loss**

$$L(y \cdot f(\mathbf{x})) = \exp(-y \cdot f(\mathbf{x}))$$

- **Hinge loss**

$$L(y \cdot f(\mathbf{x})) = \begin{cases} 1 - y \cdot f(\mathbf{x}) & \text{for } y \cdot f(\mathbf{x}) \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$



Loss functions: classification

Binary to multiclass

- **One versus one**: class C_i vs class C_j+ majority voting from all classifiers
- **One versus rest**: class C_i vs not C_i+ highest probability class
- **Comparison**: OVO needs less data to train one model but more models.

Regularization

- $E_{new} \approx J(\theta)?$ – no
- Similar for (moderately) simple models, not similar for too complex model (overfitting).

- **Explicit regularization**: penalize complexity by changing cost function

- **Implicit regularization**: **early stopping**
  - If cost function optimized iteratively, don't let it decrease too much

Explicit regularization

- Penalize cost function

$$\min_{\theta} J(\theta) + \lambda R(\theta)$$

- $\lambda > 0$

- **L1 regularization**: $R(\boldsymbol{\theta}) = \lambda \|\boldsymbol{\theta}\|_1$
- **L2 regularization:** $R(\boldsymbol{\theta}) = \lambda \|\boldsymbol{\theta}\|_2$

- **Example: Ridge regression**

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{y}_i - \boldsymbol{\theta}^T \boldsymbol{x}_i)^2 + \lambda \sum_{j=1}^{p} \theta_j^2 , \qquad \lambda > 0$$

Explicit regularization: ridge regression

Equivalent form

$$\hat{\theta}^{ridge} = \text{argmin} \sum_{i=1}^{N} (y_i - \theta_0 - \theta_1 x_{1j} - \ldots - \theta_p x_{pj})^2$$

$$\text{subject to} \sum_{j=1}^{p} \theta_j^2 \leq s$$

Solution

$$\boxed{\boldsymbol{\theta}^{\boldsymbol{ridge}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-\mathbf{1}} \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}}$$

Ridge regression

Properties

- Extreme cases:
  - $\lambda = 0$ usual linear regression (no shrinkage)
  - $\lambda = +\infty$ fitting a constant ($\boldsymbol{\theta} = 0$ except of θ_0)
- Degrees of freedom decrease when λ increases
  - $\lambda = 0 \rightarrow d.f. = p$
- $p > n$ is doable
  - Compare with linear regression
- How to estimate λ?
  - cross-validation

Ridge regression

Example Computer Hardware Data Set : performance measured for various processors and also

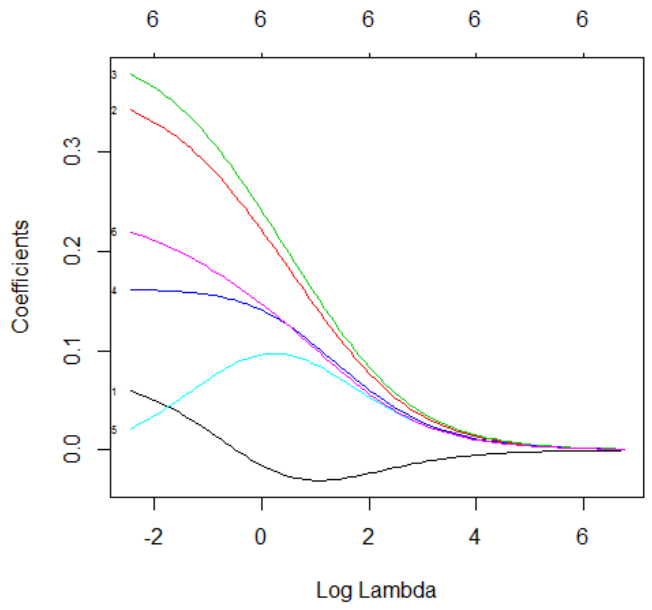
- Cycle time
- Memory
- Channels
- …

Build model predicting performance

Ridge regression

- R code: use package **glmnet** with alpha=0 (Ridge regression)
- Seeing how Ridge converges

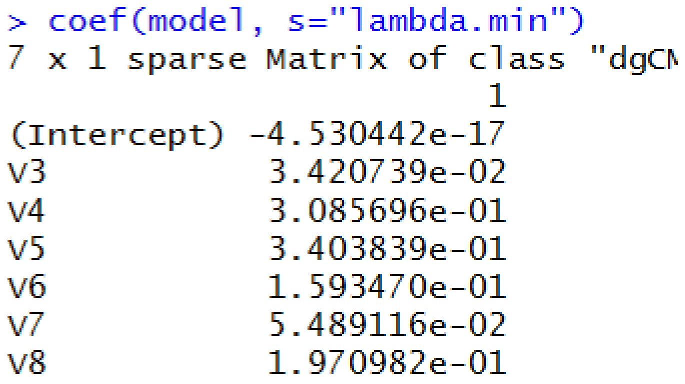
```
data=read.csv("machine.csv", header=F)
covariates=scale(data[,3:8])
response=scale(data[, 9])

model0=glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
plot(model0, xvar="lambda", label=TRUE)
```



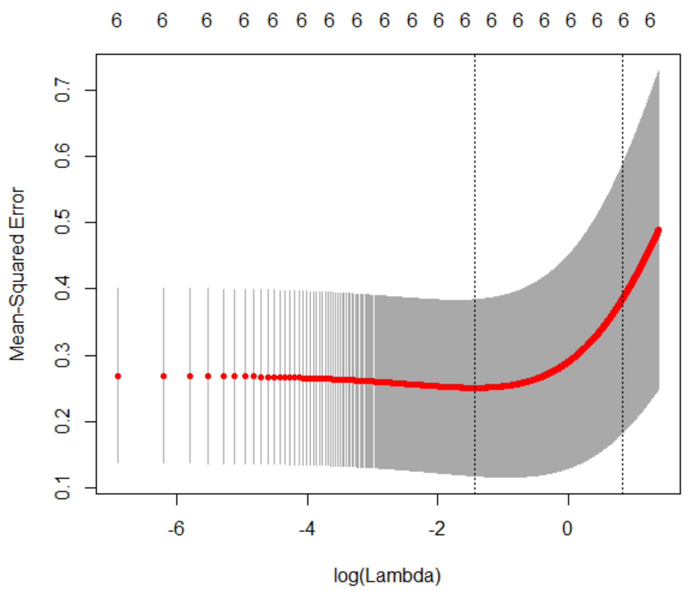
Ridge regression

- Choosing the best model by cross-validation:

```
model=cv.glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
model$lambda.min
plot(model)
coef(model, s="lambda.min")
```



```
> coef(model, s="lambda.min")
7 x 1 sparse Matrix of class "dgCM
                        1
(Intercept) -4.530442e-17
V3           3.420739e-02
V4           3.085696e-01
V5           3.403839e-01
V6           1.593470e-01
V7           5.489116e-02
V8           1.970982e-01
```



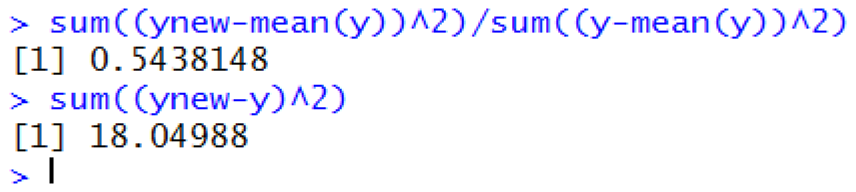
```
> model$lambda.min
 [1] 0.046
```

Ridge regression

- How good is this model in prediction?

```
ind=sample(209, floor(209*0.5))
data1=scale(data[,3:9])
train=data1[ind,]
test=data1[-ind,]

covariates=train[,1:6]
response=train[, 7]
model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",
lambda=seq(0,1,0.001))
y=test[,7]
ynew=predict(model, newx=as.matrix(test[, 1:6]), type="response")

#Coefficient of determination
sum((ynew-mean(y))^2)/sum((y-mean(y))^2)

sum((ynew-y)^2)
```

Note that data are so small so numbers change much for other train/test



```
> sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
[1] 0.5438148
> sum((ynew-y)^2)
[1] 18.04988
> 
```

LASSO

- Add **l_1 regularization term**

$$\hat{\theta}^{lasso} = \operatorname{argmin} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_{1j} - \ldots - \theta_p x_{pj})^2 + \lambda \sum_{j=1}^{p} |\theta_i| \right\}$$

- $\lambda > 0$ is **penalty factor**

- Equivalent formulation

$$\hat{\theta}^{lasso} = \operatorname{argmin} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_{1j} - \ldots - \theta_p x_{pj})^2$$

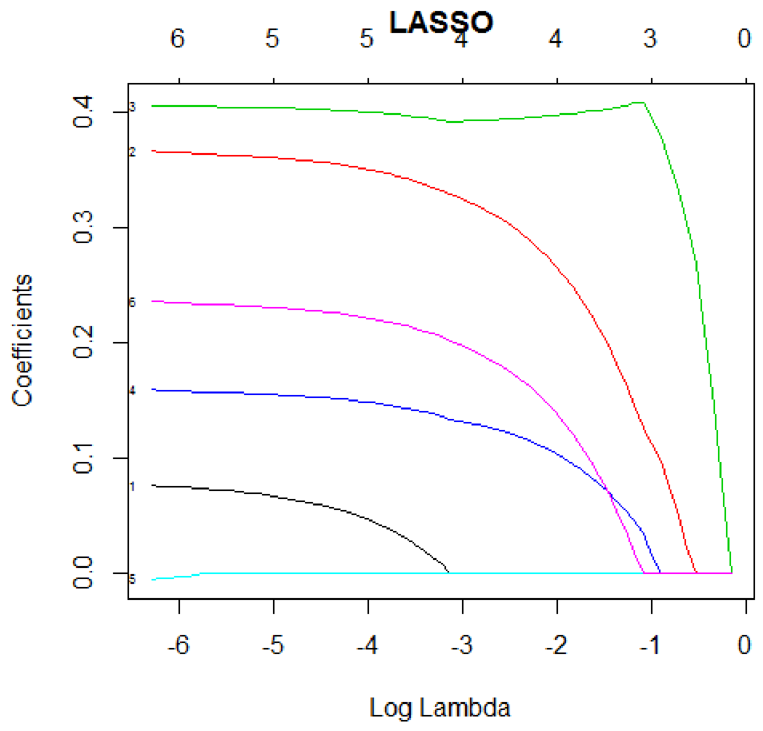
$$\text{subject to } \sum_{j=1}^{p} |\theta_i| \le s$$

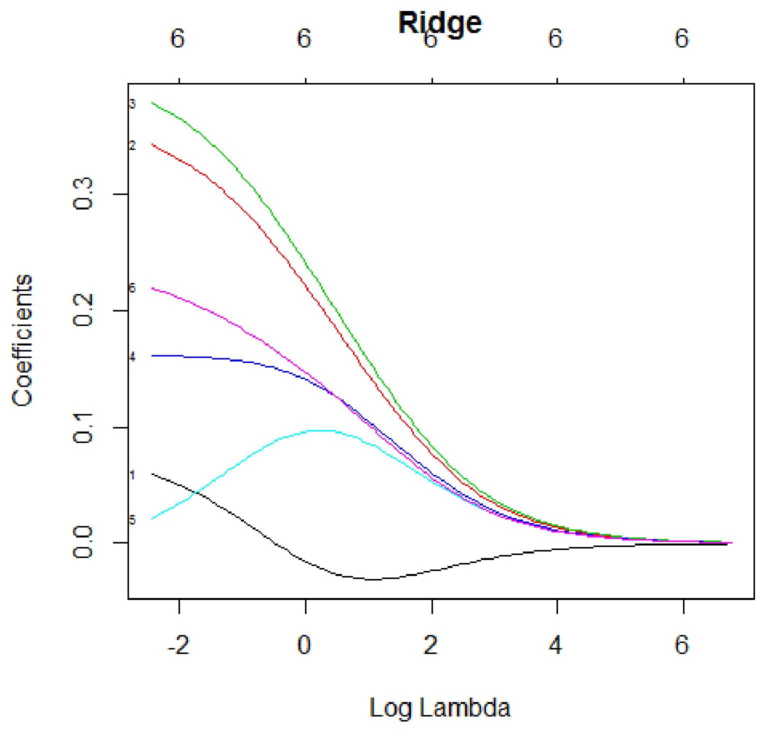
http://img-aws.ehowcdn.com

LASSO vs Ridge

- **LASSO yields sparse solutions!**

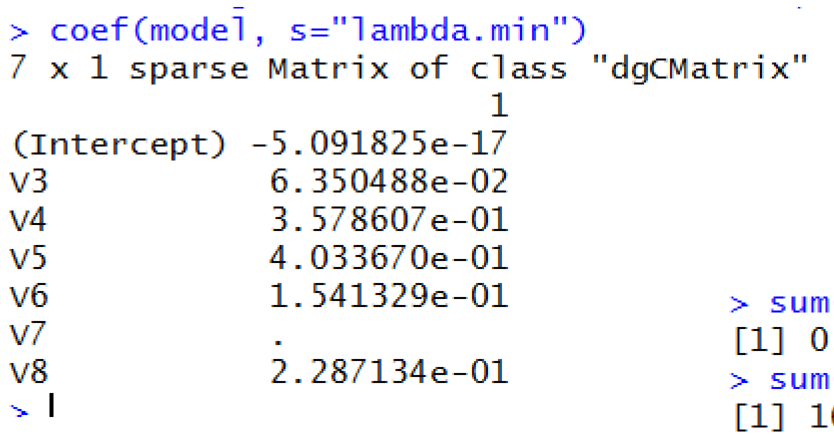
Example Computer hardware data



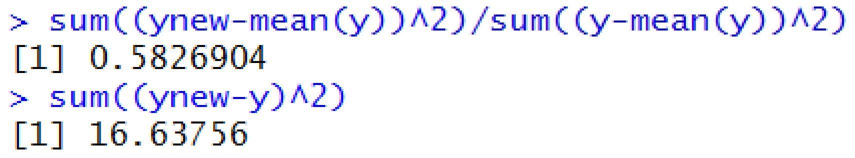


LASSO vs Ridge

- In R, use glmnet with **alpha=1**
- Only 5 variables selected by LASSO



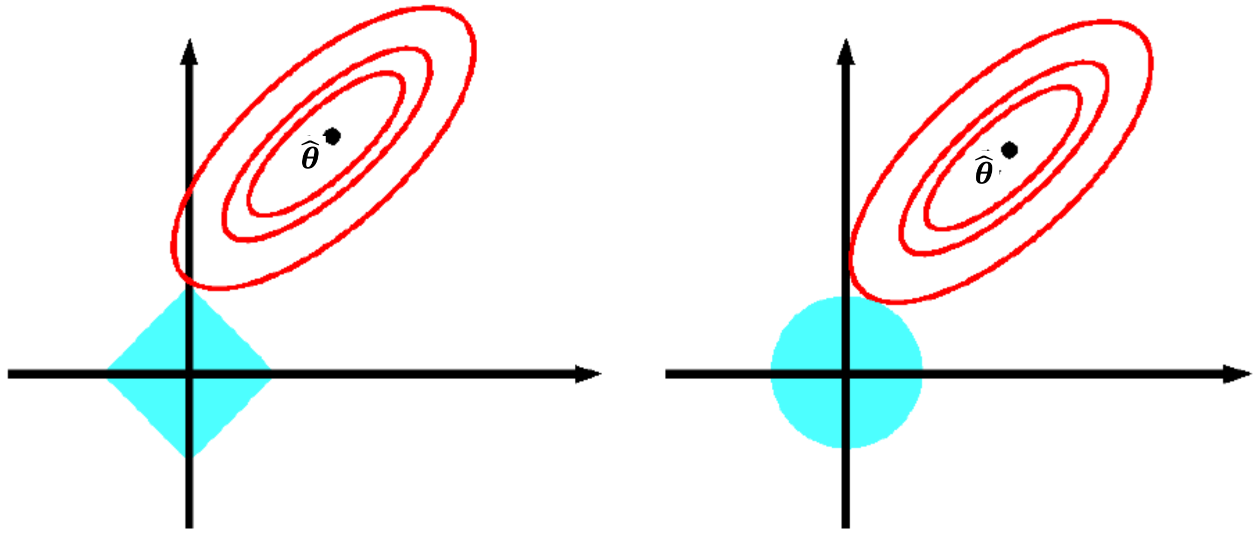
```
> coef(model, s="lambda.min")
7 x 1 sparse Matrix of class "dgCMatrix"
                        1
(Intercept) -5.091825e-17
V3           6.350488e-02
V4           3.578607e-01
V5           4.033670e-01
V6           1.541329e-01
V7           .
V8           2.287134e-01
> 
```



```
> sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
[1] 0.5826904
> sum((ynew-y)^2)
[1] 16.63756
```

LASSO vs Ridge

- Why Lasso leads to sparse solutions?
  - Feasible area for Ridge is a circle (2D)
  - Feasible area for LASSO is a polygon (2D)



LASSO properies

- **Lasso is widely used when $\boldsymbol{p \gg n}$**
  - Linear regression breaks down when $p > n$
  - Application: DNA sequence analysis, Text Prediction

- No explicit formula for $\hat{\theta}^{\boldsymbol{lasso}}$
  - Optimization algorithms used

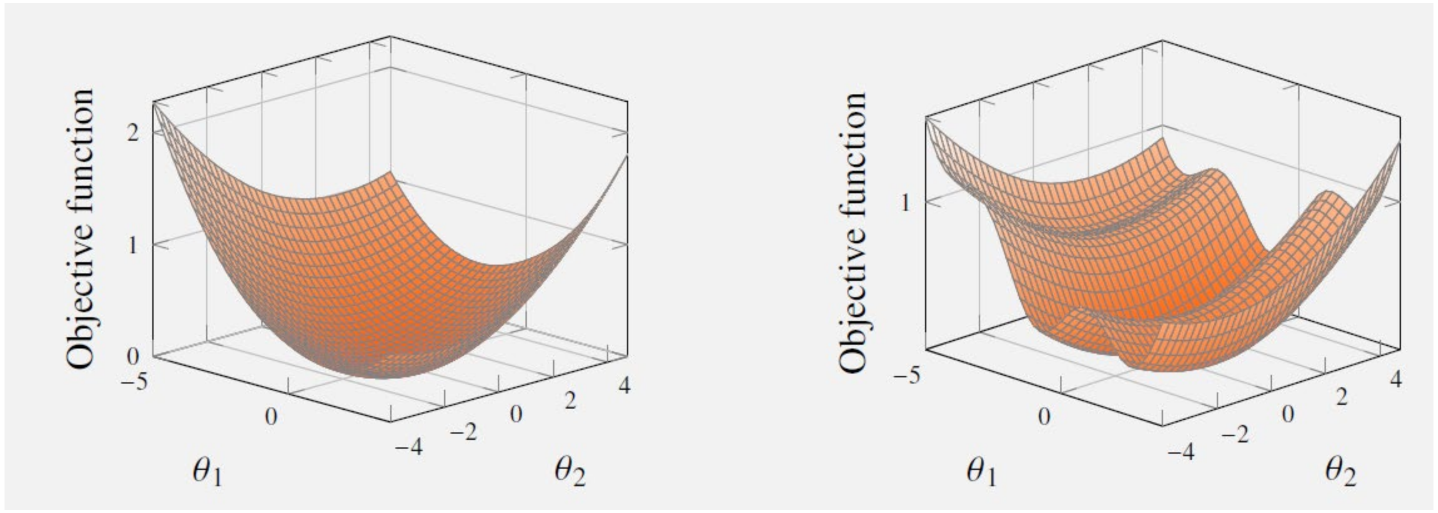
Optimization methods

- Numerical optimization often needed

$$\min_{\theta} J(\boldsymbol{\theta})$$

$$\min_{\lambda} E_{hold-out}(\lambda)$$

- If not convex objective, more than one local optimum

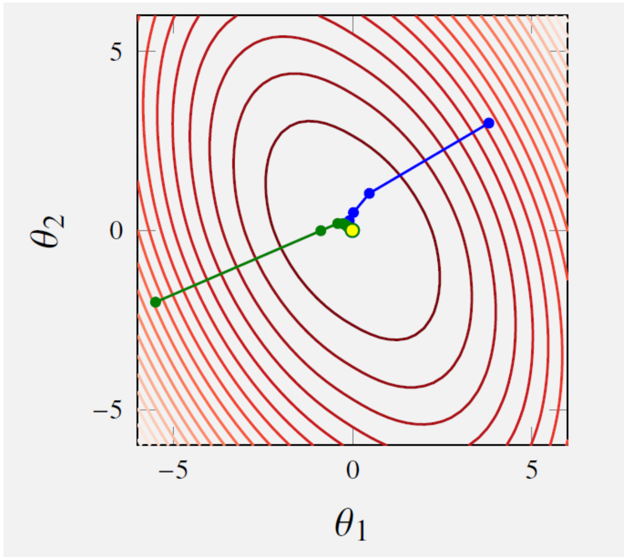


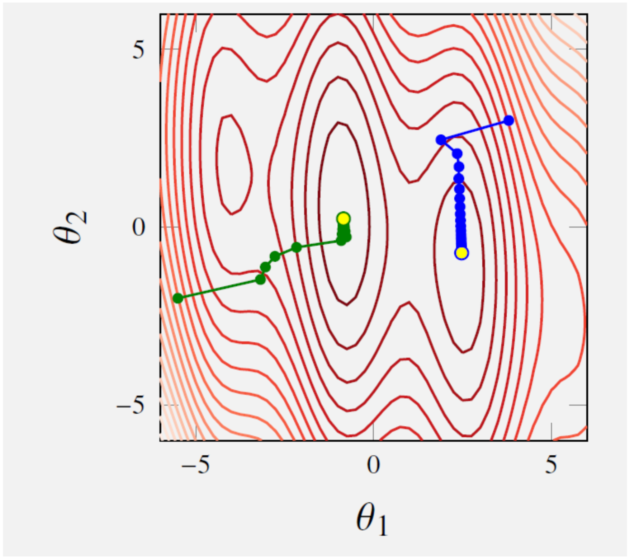
Optimization methods

- **Gradient descent method**

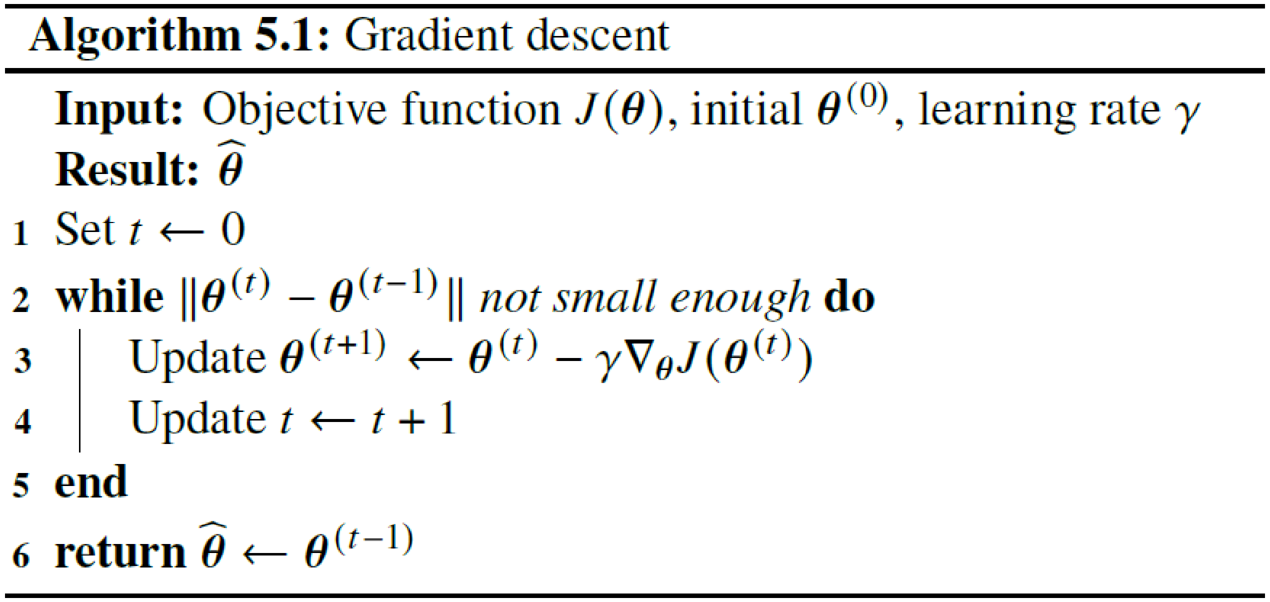
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- Basic idea:
  - Start from some point $\boldsymbol{\theta}_0$
  - Move to the next point along **descent direction** $-\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$





Gradient descent



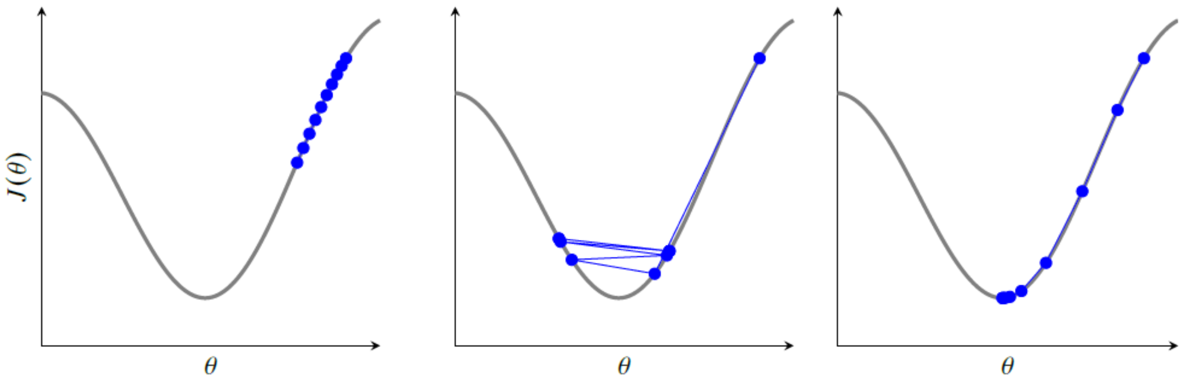
Algorithm 5.1: Gradient descent

```
Input: Objective function $J(\boldsymbol{\theta})$, initial $\boldsymbol{\theta}^{(0)}$, learning rate $\gamma$
Result: $\widehat{\boldsymbol{\theta}}$
Set $t \leftarrow 0$
while $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t-1)}\|$ not small enough do
    Update $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$
    Update $t \leftarrow t + 1$
end
return $\widehat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(t-1)}$
```

- Example: logistic regression

Gradient descent

- Influence of γ



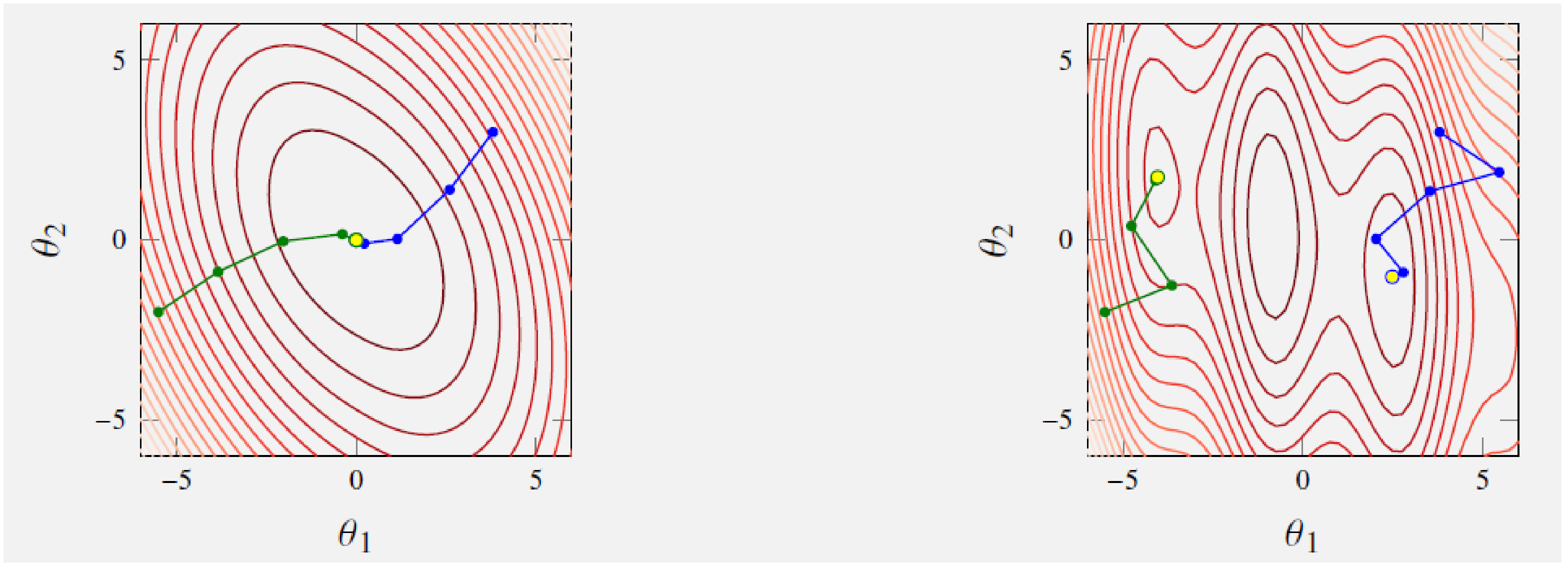
- Trace $J\left(\theta^{(t)}\right)$ vs t
  - High oscillation→decrease γ
  - Slow changes → increase γ

- Try with different $\theta^{(0)}$ if possible

Newton’s method

- Assume $J(\boldsymbol{\theta})$ is ”locally” quadratic
- Newton’s method: move along the best direction

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta[\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})]^{-1}[\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})]$$

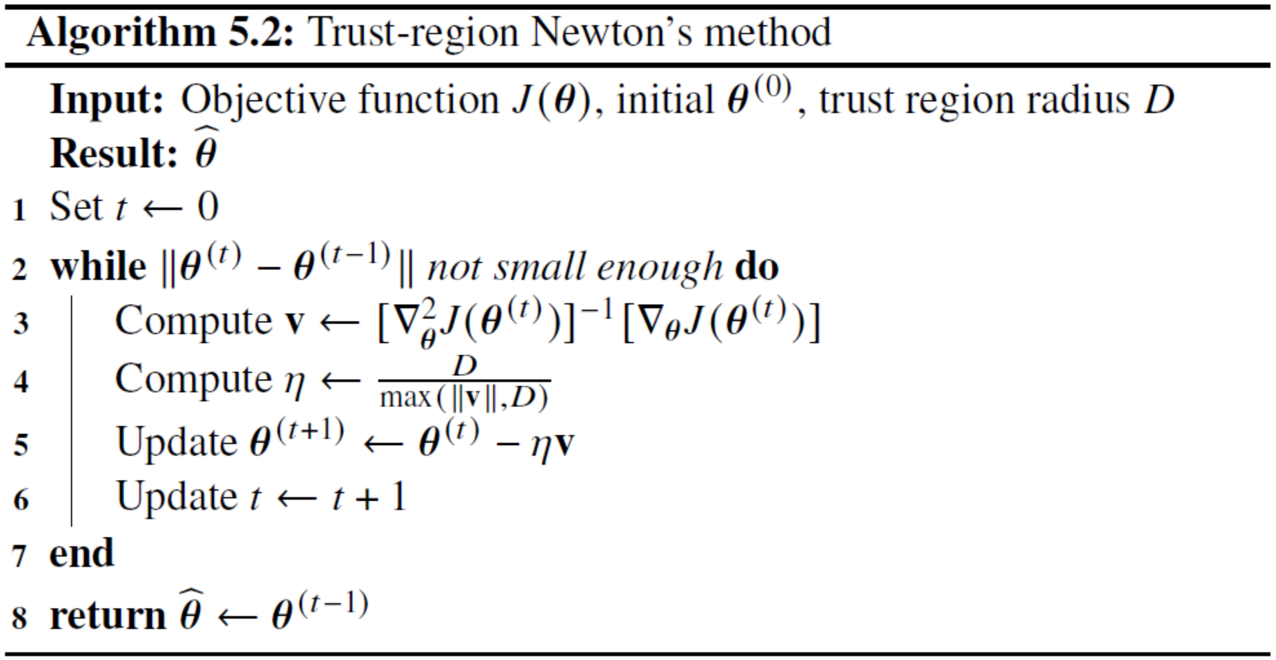


Newton's method

- Properties
  - No convergence guarantees
  - Advantage: if $J(\boldsymbol{\theta})$ is quadratic and $\eta = 1$ → convergence in one iteration
  - Disadvantage 1: Hessian must be invertable
  - Disadvantage 2: Hessian is computationally heavy

- Solution: quasi-Newton methods (ex. **BFGS**)
  - Choose some $H^{(0)}$
  - Approximate hessian $H^{(t)} = \phi(H^{(t-1)}, \nabla J(\boldsymbol{\theta}^{(t-1)}), \nabla J(\boldsymbol{\theta}^{(t)}))$

Newton’s method



Algorithm 5.2: Trust-region Newton’s method

```
    Input: Objective function $J(\boldsymbol{\theta})$, initial $\boldsymbol{\theta}^{(0)}$, trust region radius $D$
    Result: $\widehat{\boldsymbol{\theta}}$
1   Set $t \leftarrow 0$
2   while $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t-1)}\|$ not small enough do
3       Compute $\mathbf{v} \leftarrow [\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})]^{-1} [\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})]$
4       Compute $\eta \leftarrow \frac{D}{\max(\|\mathbf{v}\|, D)}$
5       Update $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta\mathbf{v}$
6       Update $t \leftarrow t + 1$
7   end
8   return $\widehat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(t-1)}$
```

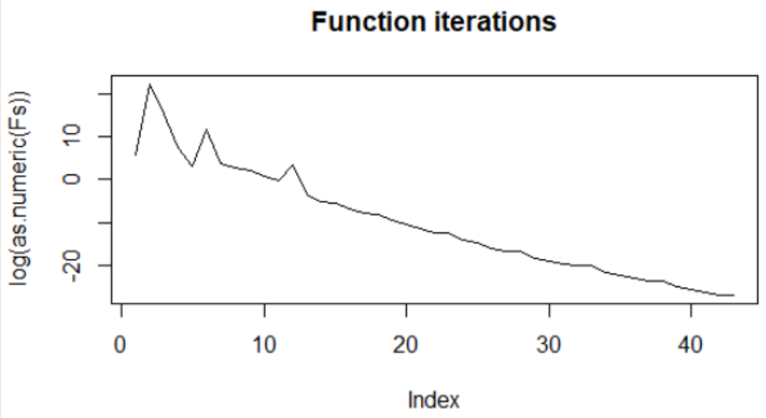
Optimization methods in R

- In R, use optim(par, fn, gr, method,…)
  - par: initial parameter vector
  - fn: function to optimize
  - gr: gradient function
  - method

Example: trace plot for $y = (x_1 - 2)^4 + (x_2 - 4)^4$

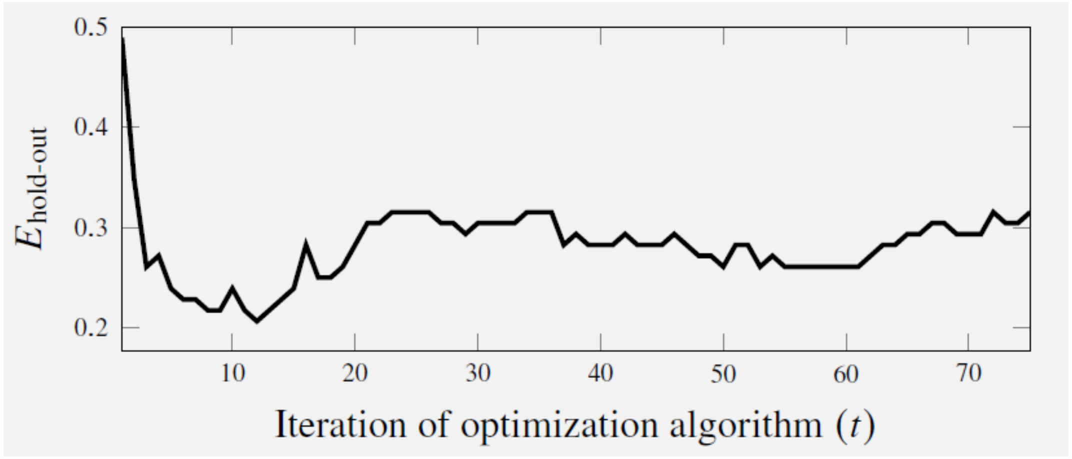
Optimization methods in R

```
#Workaround: optim does not return iterations

Fs=list()
Params=list()
k=0

myf<- function(x){
  f=(x[1]-2)^4+(x[2]-4)^4
  .GlobalEnv$k= .GlobalEnv$k+1
  .GlobalEnv$Fs[[k]]=f
  .GlobalEnv$Params[[k]]=x
  return(f)
}
myGrad <-function(x) c(4*(x[1]-2)^3, 4*(x[2]-4)^3)

res<-optim(c(0,0), fn=myf, gr=myGrad, method="BFGS")

plot(log(as.numeric(Fs)), type="l", main="Function
iterations")
```

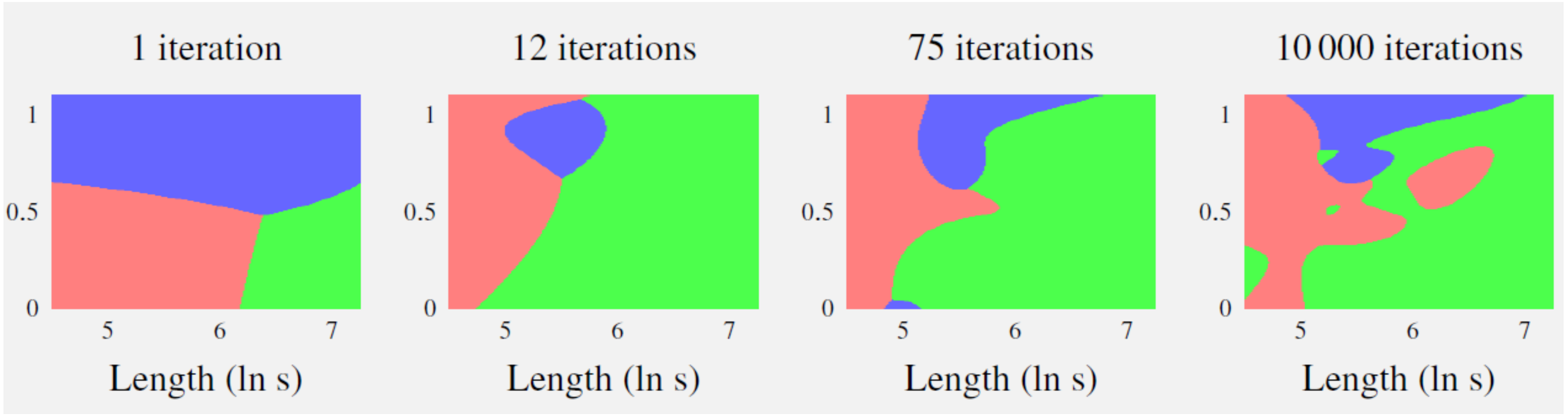


Implicit regularization

- **Early stopping**
  - For complex models, accurate model optimization may lead to overfitting
  - Start from some parameter set (probably not optimal, large E_{train} and E_{new})
  - Trace the validation error (and training error?) for each t
  - Choose model with the smallest validation error



Implicit regularization



Optimization for large data

Stochastic gradient descent

Idea: use gradient descent + approximation to expected value

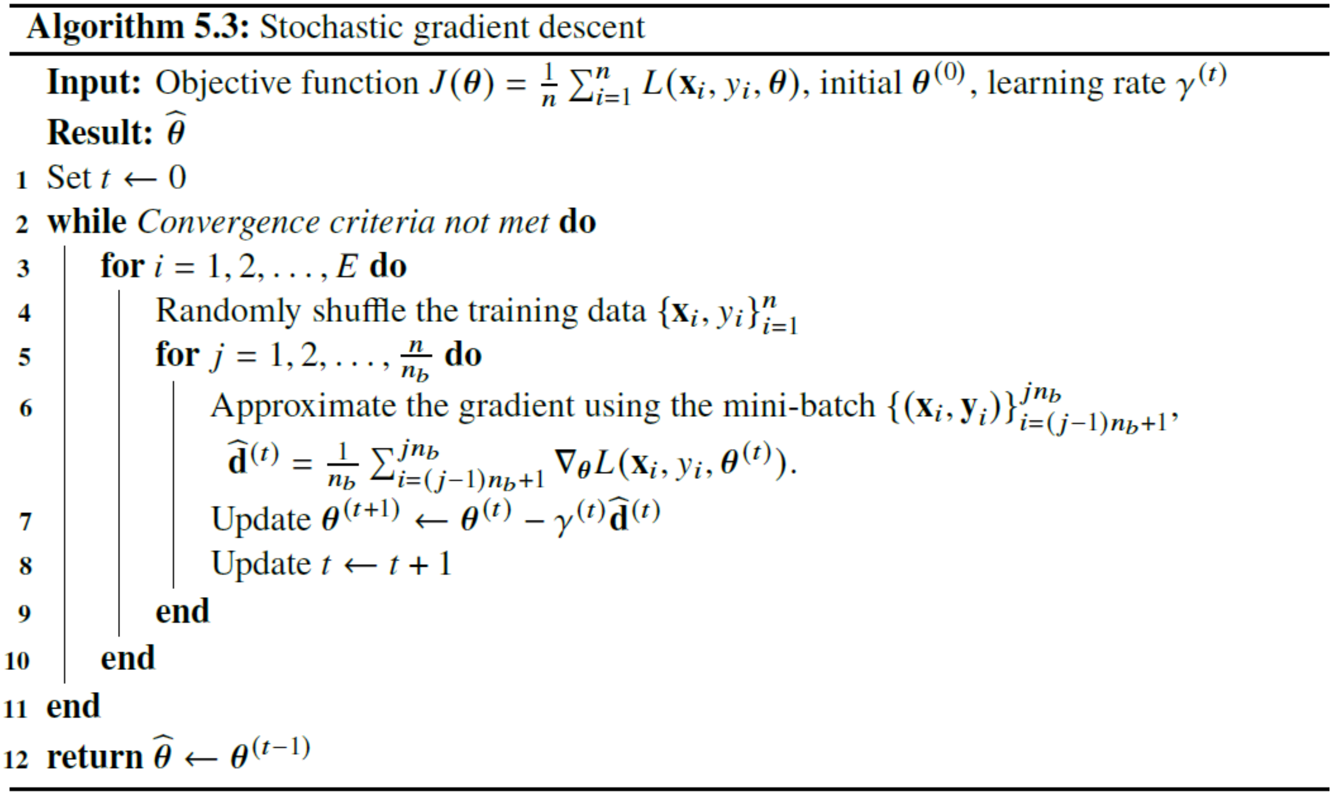
- For **random** sample of size n_b from sample of size n

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \approx \frac{1}{n_b}\sum_{i=1}^{n_b} x_i$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{n_b} \sum_{(\boldsymbol{x}_i, y_i) \in sample} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}_i, y_i, \boldsymbol{\theta})$$

1. One **epoch**:
   1. Permute data and divide into batches of size n_b
   2. In each optimization iteration, use one batch
2. Repeat step 1

Stochastic gradient descent



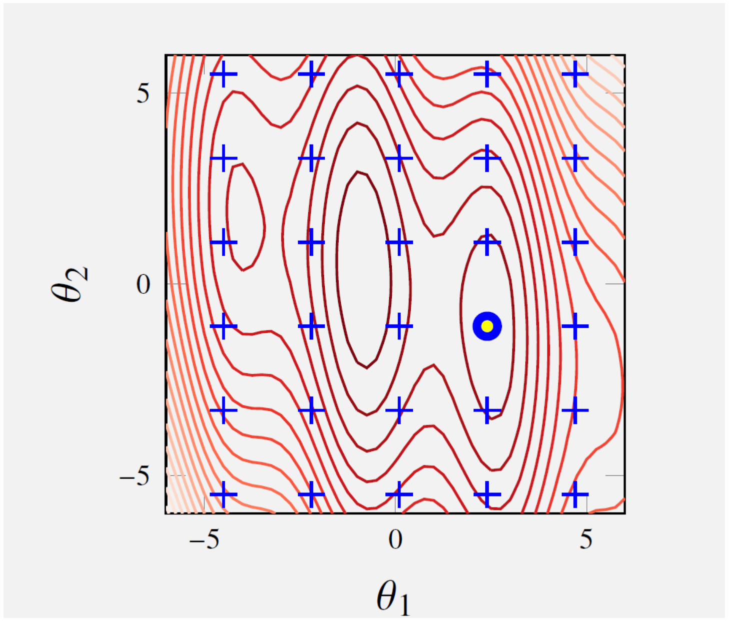
Algorithm 5.3: Stochastic gradient descent

```
Input: Objective function $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \theta)$, initial $\theta^{(0)}$, learning rate $\gamma^{(t)}$
Result: $\widehat{\theta}$
Set $t \leftarrow 0$
while Convergence criteria not met do
    for $i = 1, 2, \ldots, E$ do
        Randomly shuffle the training data $\{\mathbf{x}_i, y_i\}_{i=1}^{n}$
        for $j = 1, 2, \ldots, \frac{n}{n_b}$ do
            Approximate the gradient using the mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=(j-1)n_b+1}^{jn_b}$,
            $\widehat{\mathbf{d}}^{(t)} = \frac{1}{n_b}\sum_{i=(j-1)n_b+1}^{jn_b} \nabla_\theta L(\mathbf{x}_i, y_i, \theta^{(t)})$.
            Update $\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)}\widehat{\mathbf{d}}^{(t)}$
            Update $t \leftarrow t + 1$
        end
    end
end
return $\widehat{\theta} \leftarrow \theta^{(t-1)}$
```

- Different choices for γ_t, for ex $\gamma^{(t)} = \frac{1}{t^{\alpha}}, \alpha \in (0.5, 1]$

Hyperparameter optimization

- $E_{hold-out}$ costly to compute→ usual optimization very hard
  - Note: for each λ first we need to optimize θ…+ gradients of $E_{hold-out}$
- Grid search (can also be costly)
  - Alternative: Bayesian optimization


θ_2
θ_1