Principal component regression. Uncertainty estimation

Lecture 2d

Principle component regression

Step 1: Compute principal components $v_1 \dots v_M$

Step 2: Compute derived data as $z_i = Xv_i$

Step 3: Compute linear regression using data set Z with columns $z_1..z_M$, Y

The result
$$\hat{Y}_{pcr} = \sum_{m=1}^{M} \frac{\langle z_m, Y \rangle}{\langle z_m, z_m \rangle} z_m$$

Coefficients
$$\hat{\beta}_{pcr} = \sum_{m=1}^{M} \frac{\langle z_m, Y \rangle}{\langle z_m, z_m \rangle} v_m$$

PCR: comments

- Result depends on the scaling of features→ standardize the original data
- If M<p → reduced regression
- If M=p → The fitted response will be the same
- Ridge regression shrinks the coefficients along the principal components, principal components regression discards p-M smallest components

Partial Least Squares Regression (PLS)

- Idea with PLS is to find M<p orthogonal directions (as in PCR). The difference:

PCR

$$\max_\alpha \mathrm{Var}(\mathbf{X}\alpha)$$
$$\text{subject to } ||\alpha|| = 1,\ \alpha^T \mathbf{S} v_\ell = 0,\ \ell = 1, \ldots, m-1,$$

$$\max_\alpha \mathrm{Corr}^2(\mathbf{y}, \mathbf{X}\alpha)\mathrm{Var}(\mathbf{X}\alpha)$$
$$\text{subject to } ||\alpha|| = 1,\ \alpha^T \mathbf{S} \hat{\varphi}_\ell = 0,\ \ell = 1, \ldots, m-1.$$

PLS

Partial least squares regression (PLS)

Step 1: Standardize features to mean zero and variance one

Step 2: Compute the first derived feature by setting

$$\mathbf{z}_1 = \sum_{j=1}^{p} \varphi_{1j} \mathbf{x}_j$$

where the φ_{1j} is projection of **Y** on $\mathbf{x}_j$

Step 3: Orthogonalize $\mathbf{x}_1 \ldots \mathbf{x}_m$ with respect to z_1

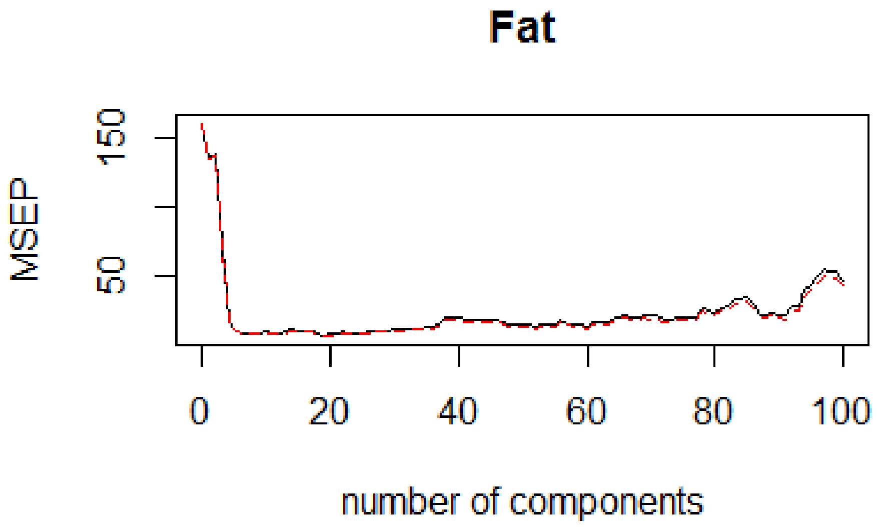
Step 4: repeat from step 2 and find $\mathbf{z}_2 .. \mathbf{z}_M$

Step 5: Compute regression of **Y** on $\mathbf{z}_1 .. \mathbf{z}_M$

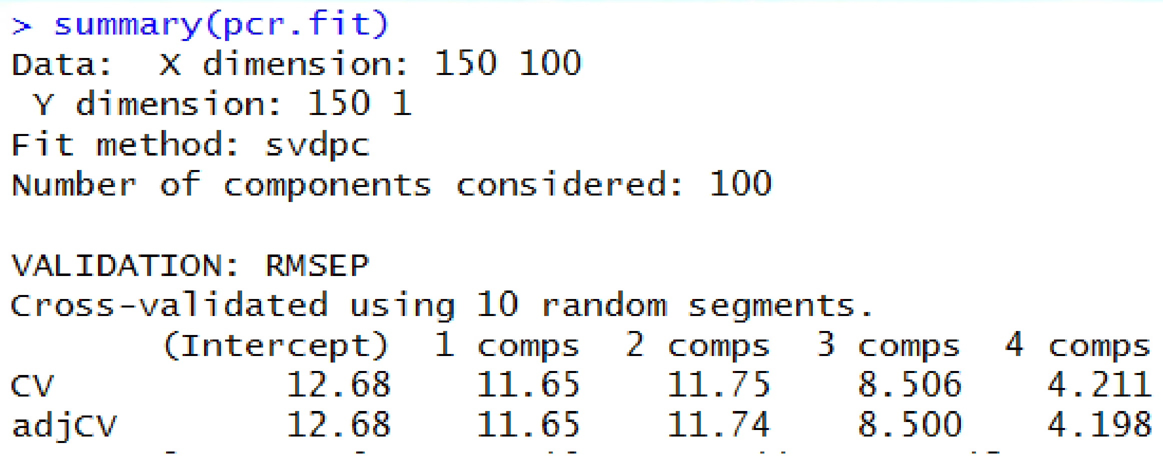
PCR and PLS: R

- Package **pls**
- PCR: pcr(formula, ncomp, data, scale = FALSE, validation = c("none", "CV", "LOO”...)
- PLS: plsr(...)

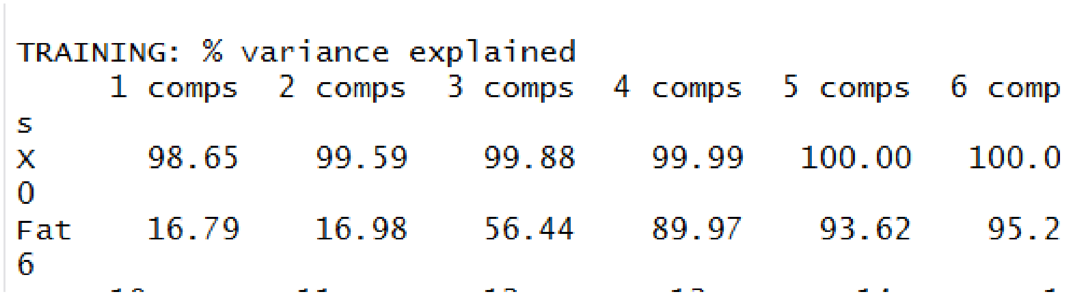
```
predictors=paste("Channel", 1:100,
sep="")
f=formula(paste("Fat ~ ",
paste(predictors, collapse=" + ")))

set.seed(12345)
pcr.fit=pcr(f, data=train,
validation="CV")
summary(pcr.fit)
validationplot(pcr.fit,val.type="MSEP")
```



PCR and PLS: R



```
> summary(pcr.fit)
Data:   X dimension: 150 100
 Y dimension: 150 1
Fit method: svdpc
Number of components considered: 100

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps
CV           12.68    11.65    11.75    8.506    4.211
adjCV        12.68    11.65    11.74    8.500    4.198
```



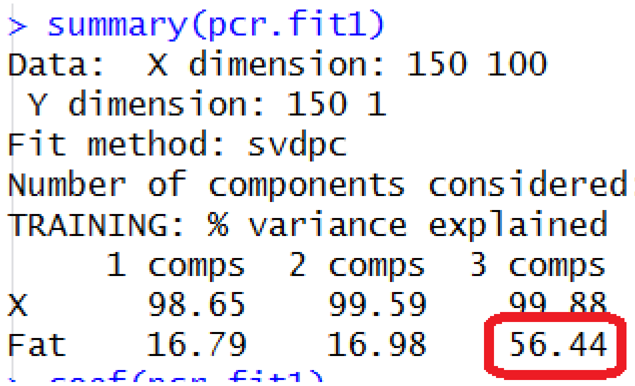
```
TRAINING: % variance explained
     1 comps  2 comps  3 comps  4 comps  5 comps  6 comp
s
X      98.65    99.59    99.88    99.99   100.00   100.0
0
Fat    16.79    16.98    56.44    89.97    93.62    95.2
6
```

PCR

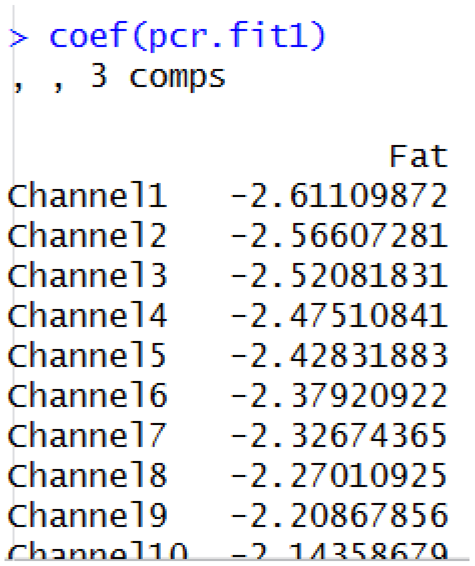
- Select 3 components

Coefficients in the original variables

```
pcr.fit1=pcr(f, 3,data=train, validation="none")
summary(pcr.fit1)
coef(pcr.fit1)
scores(pcr.fit1)
l=loadings(pcr.fit1)
print(l,cutoff=0)
Yloadings(pcr.fit1)
plot(pcr.fit1)
```

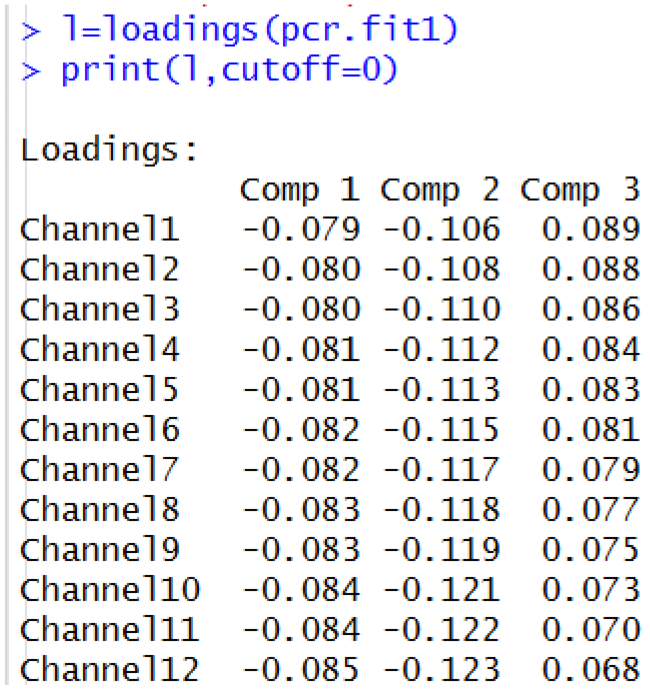


```
> summary(pcr.fit1)
Data:   X dimension: 150 100
 Y dimension: 150 1
Fit method: svdpc
Number of components considered: 3
TRAINING: % variance explained
     1 comps  2 comps  3 comps
X      98.65    99.59    99.88
Fat    16.79    16.98    56.44
> coef(pcr.fit1)
```



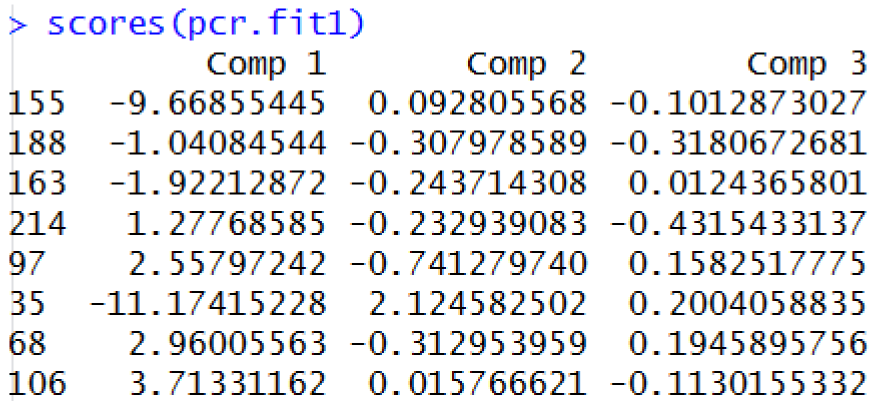
```
> coef(pcr.fit1)
, , 3 comps

                   Fat
Channel1   -2.61109872
Channel2   -2.56607281
Channel3   -2.52081831
Channel4   -2.47510841
Channel5   -2.42831883
Channel6   -2.37920922
Channel7   -2.32674365
Channel8   -2.27010925
Channel9   -2.20867856
Channel10  -2.14358679
```

PCR



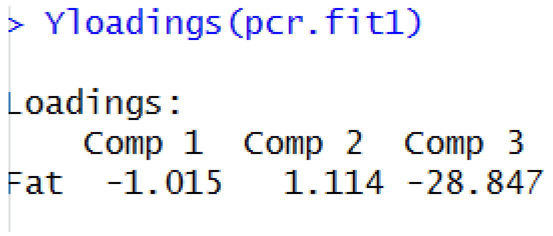
```
> l=loadings(pcr.fit1)
> print(l,cutoff=0)

Loadings:
          Comp 1 Comp 2 Comp 3
Channel1  -0.079 -0.106  0.089
Channel2  -0.080 -0.108  0.088
Channel3  -0.080 -0.110  0.086
Channel4  -0.081 -0.112  0.084
Channel5  -0.081 -0.113  0.083
Channel6  -0.082 -0.115  0.081
Channel7  -0.082 -0.117  0.079
Channel8  -0.083 -0.118  0.077
Channel9  -0.083 -0.119  0.075
Channel10 -0.084 -0.121  0.073
Channel11 -0.084 -0.122  0.070
Channel12 -0.085 -0.123  0.068
```

Scores matrix (new coordinates)



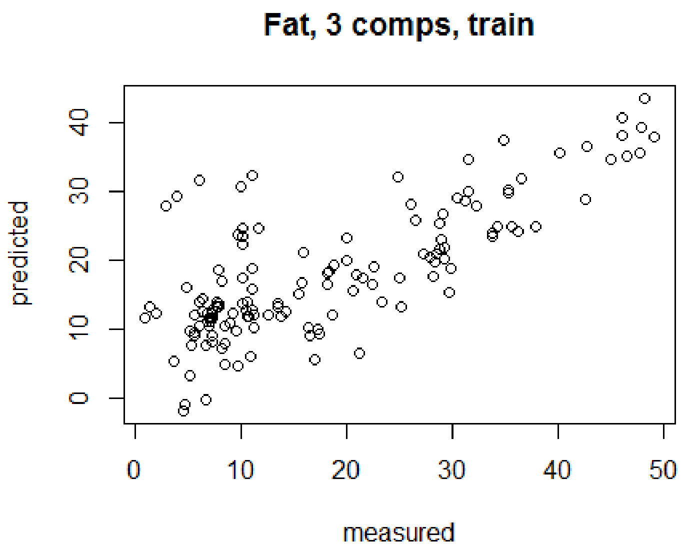
```
> scores(pcr.fit1)
         Comp 1       Comp 2        Comp 3
155  -9.66855445  0.092805568 -0.1012873027
188  -1.04084544 -0.307978589 -0.3180672681
163  -1.92212872 -0.243714308  0.0124365801
214   1.27768585 -0.232939083 -0.4315433137
97    2.55797242 -0.741279740  0.1582517775
35  -11.17415228  2.124582502  0.2004058835
68    2.96005563 -0.312953959  0.1945895756
106   3.71331162  0.015766621 -0.1130155332
```

PCR



```
> Yloadings(pcr.fit1)

Loadings:
    Comp 1  Comp 2  Comp 3
Fat  -1.015   1.114 -28.847
```

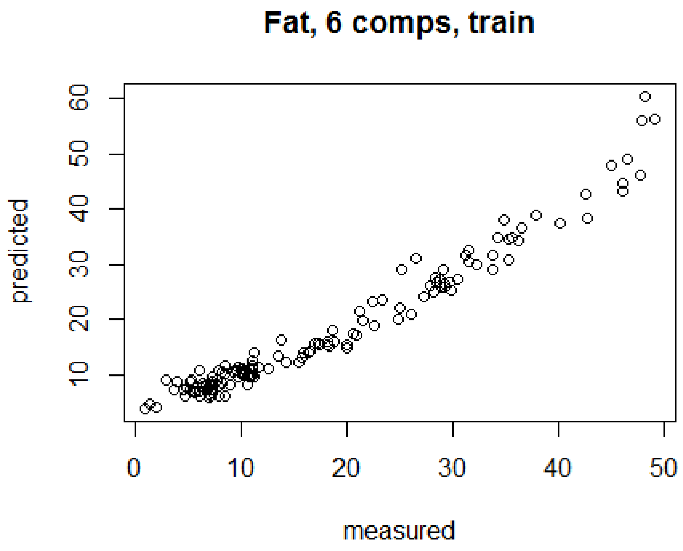
What is the regression equation
in the new coordinates?



Is fit OK?

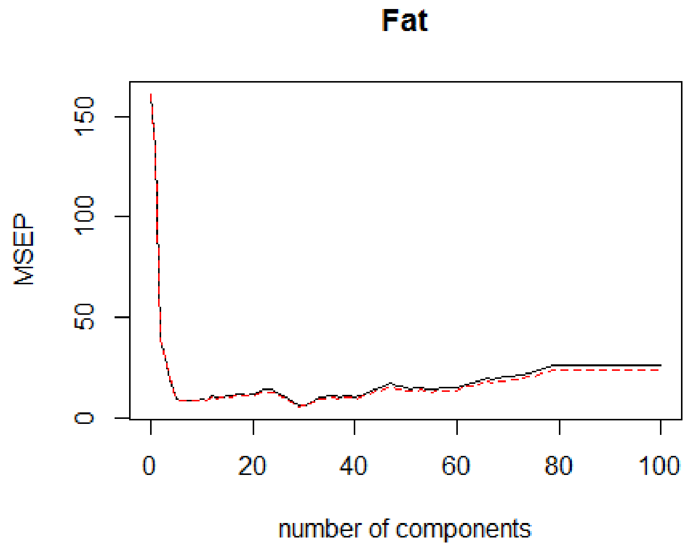
PCR

- Now 6 components



```
> summary(pcr.fit2)
Data:   X dimension: 150 100
 Y dimension: 150 1
Fit method: svdpc
Number of components considered: 6
TRAINING: % variance explained
     1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
X      98.65    99.59    99.88    99.99   100.00   100.00
Fat    16.79    16.98    56.44    89.97    93.62    95.29
```

PLS



```
> summary(pls.fit2)
Data:   X dimension: 150 100
 Y dimension: 150 1
Fit method: kernelpls
Number of components considered: 6
TRAINING: % variance explained
     1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
X      98.65    98.95    99.75    99.99   100.00   100.00
Fat    17.10    77.67    83.32    90.58    94.93    95.46
```

Probabilistic models

- Why it is beneficial to assume a **probabilistic** model?
- A common approach to modelling in CS and engineering:

$$y = f(x, w)$$

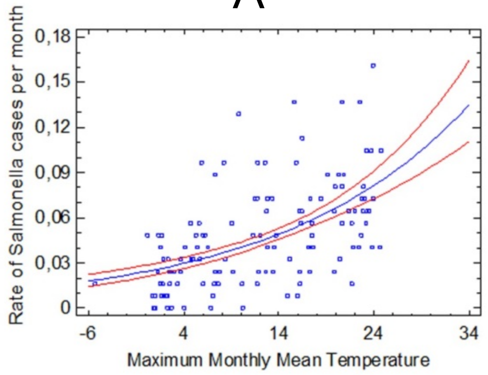
- f is known, w is unknown
- Fit model to data with least squares, optimization or ad hoc→ find w

Probabilistic models

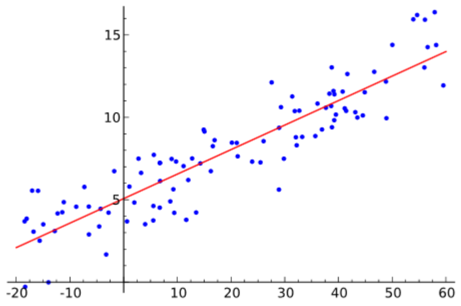
Arguments against deterministic models:

- The model does not really describe actual data (error is not explained)
  - No difference between modelling data A (Poisson) and B (Normal)
  - Estimation strategy for A is not good for B
- The model typically gives a **deterministic answer**, no information about uncertainty
  - ”…The exchange rate tomorrow will be 8.22 …”

A



B



Probabilistic models

Probabilistic model

$$Y \sim Distribution(f(x,w),\theta)$$

- Data is fully explained (error as well)
- Automatic principle for finding parameters: MLE , MAP or Bayes theorem
- Automatic principle for finding uncertainty (conf. limits)
  - **Bootstrap**
  - Posterior probability
- Possibility to generate new data of the same type
  - Further testing of the model

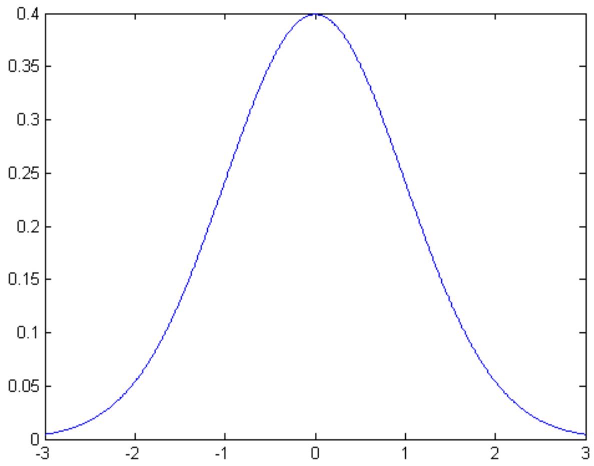
Uncertainty estimation

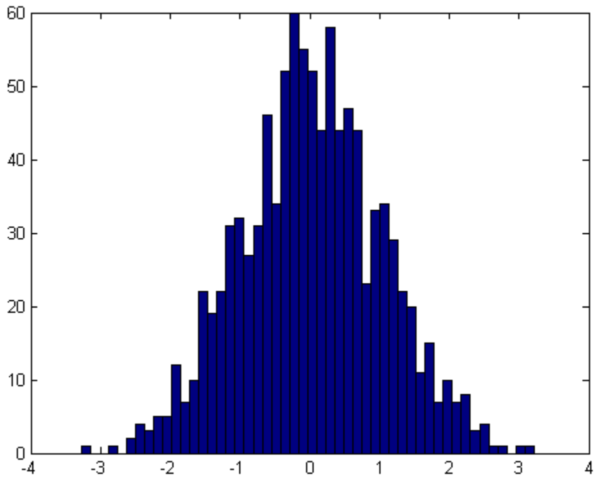
- Given estimator $\hat{f} = \hat{f}(x, D)$ (or $\widehat{\boldsymbol{\alpha}} = \delta(D)$), how to estimate the uncertainty?

- Answer 1: if the distribution for data *D* is given, compute analytically the distribution for the estimator→ derive confidence limits
  - Often difficult
  - Example: In simple linear regression, $\widehat{\alpha}$ follows *t* distribution

- Answer 2: Use **bootstrap**

The bootstrap: general principle



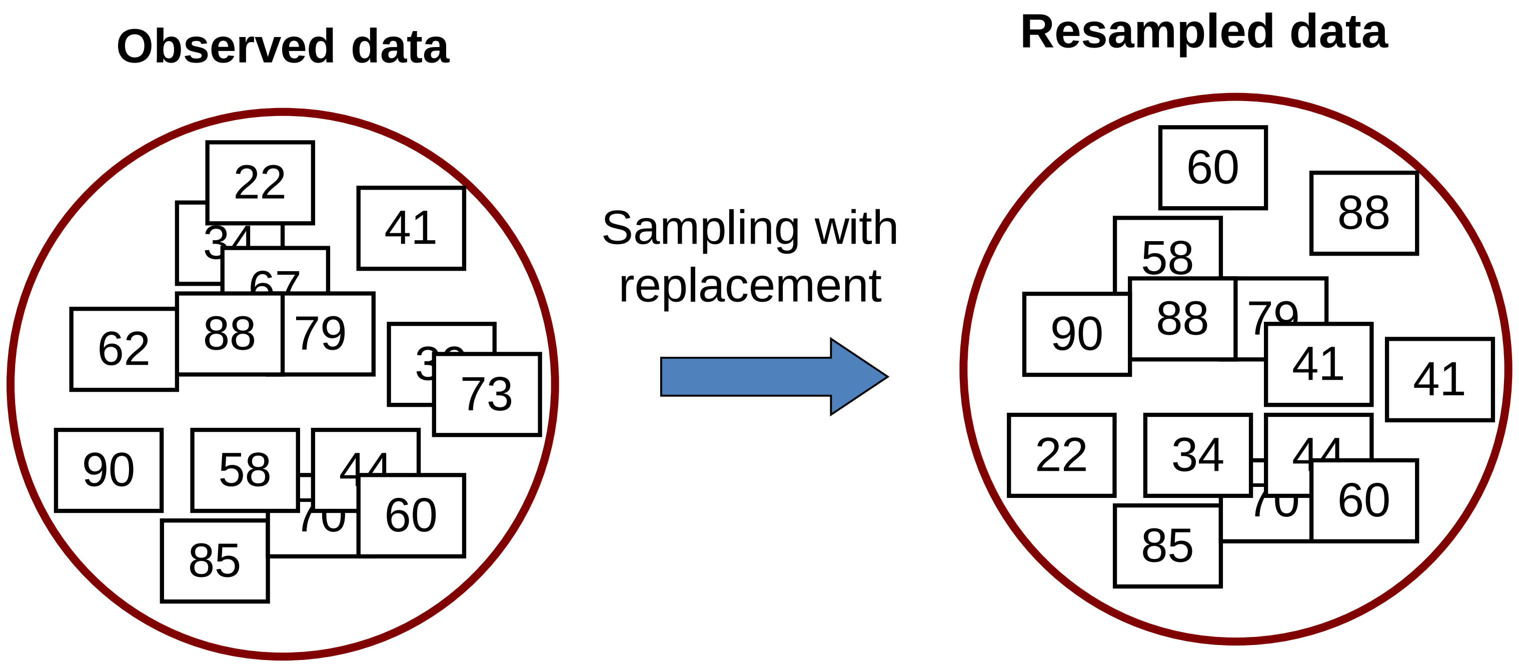


We want to determine uncertainty of $\hat{f}(D, X)$

1. Generate many different D_i from their distribution
2. Use histogram of $\hat{f}(D_i, X)$ to determine confidence limits→ unfortunately can not be done (distr of *D is often unknown)*

Instead: Generate many different D_i^* from the empirical distribution (histogram)

Nonparametric bootstrap



$\bar{x}$

$\bar{x}_1^*, \bar{x}_2^*, \ldots, \bar{x}_N^*$

Nonparametric bootstrap

Given estimator $\widehat{w} = \hat{f}(D)$

Assume $X \sim F(X, w)$, F and w are unknown

1. Estimate $\widehat{w}$ from data **D=**$(X_1,\ldots X_n)$
2. Generate $\mathbf{D_1}$ **=**$(X^*_1,\ldots X^*_n)$ by sampling with replacement
3. Repeat step 2 B times
4. The distribution of w is given by $\hat{f}(D_1), \ldots \hat{f}(D_B)$

Nonparametric bootstrap can be applied to any deterministic estimator, distribution-free

Parametric bootstrap

Given estimator $\widehat{w} = \hat{f}(D)$

Assume $X \sim F(X, w)$, F is known and w is unknown

1. Estimate $\widehat{w}$ from data **D=**$(X_1, \dots X_n)$
2. Generate $\mathbf{D_1}$ **=**$(X^*_1, \dots X^*_n)$ by generating from $F(X, \widehat{w})$
3. Repeat step 2 B times
4. The distribution of w is given by $\hat{f}(D_1), \dots \hat{f}(D_B)$

Parametric bootstrap is **more** precise if the distribution form is correct

Uncertainty estimation

1. Get D_1 , … D_B by bootstrap
2. Use $\hat{f}(D_1)$, … $\hat{f}(D_B)$ to estimate the uncertainty
   – Boostrap percentile
   – Bootstrap Bca
   – …

- Bootstrap works for all distribution types
- Can be bad accuracy for small data sets $n < 40$ (empirical is far from true)
- Parametric bootstrap works even for small samples

Bootstrap confidence intervals

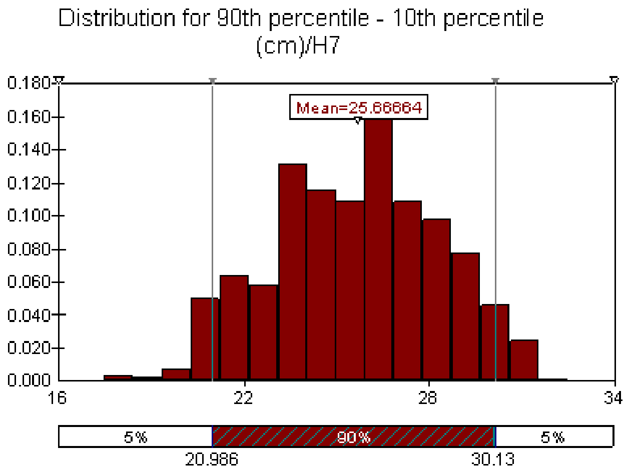
- To estimate 100(1-α) confidence interval for w

Bootstrap percentile method

1. Using bootstrap, compute $\hat{f}(D_1)$, … $\hat{f}(D_B)$, sort in ascending order, get $w_1 \dots w_B$
2. Define A_1=ceil(B α/2), A_2=floor(B-B α/2)
3. Confidence interval is given by

$$\left(w_{A_1}, w_{A_2}\right)$$

Look at the plot…



Bootstrap: regression context

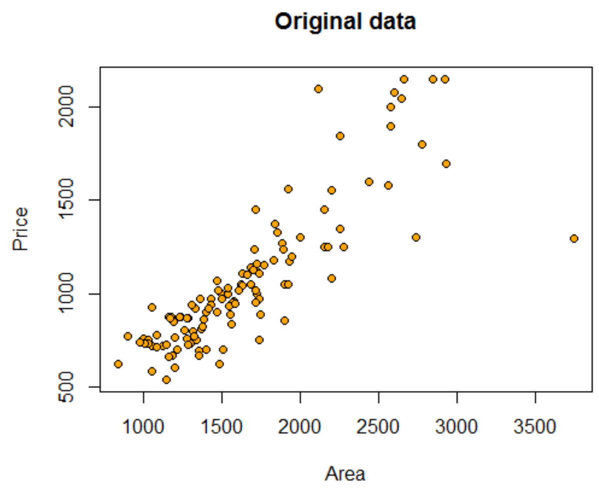
- Model $Y \sim F(X, w)$
- Data $\mathrm{D} = \{(Y_i, X_i), i = 1, \dots, n\}$
- Idea: produce several bootstrap sets that are similar to D

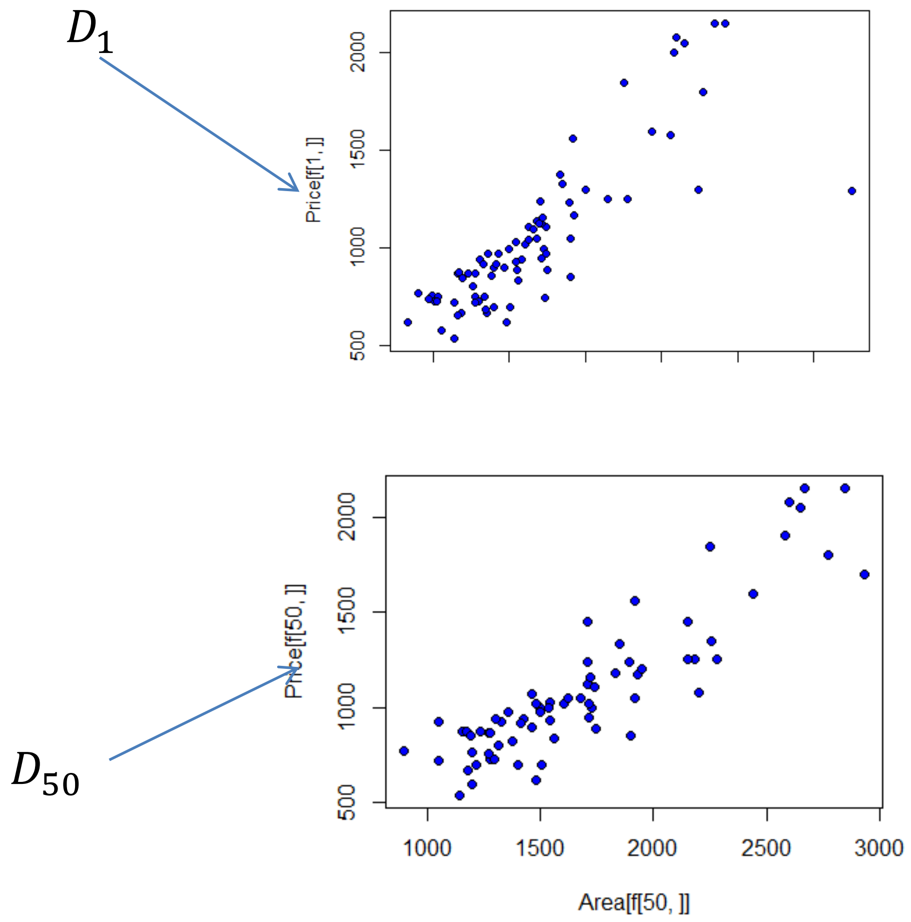
Nonparametric bootstrap:

1. Using observation set **D**, sample **pairs** (X_i, Y_i) with replacement and get bootstrap sample **D$_1$**
2. Repeat step 1 B times→ get **D$_{1,\dots}$ D$_B$**

Uncertainty estimation

Example: Albuquerque dataset:
Y=Price of House
X=Area (sqft)





We sample data index, from {1…N}

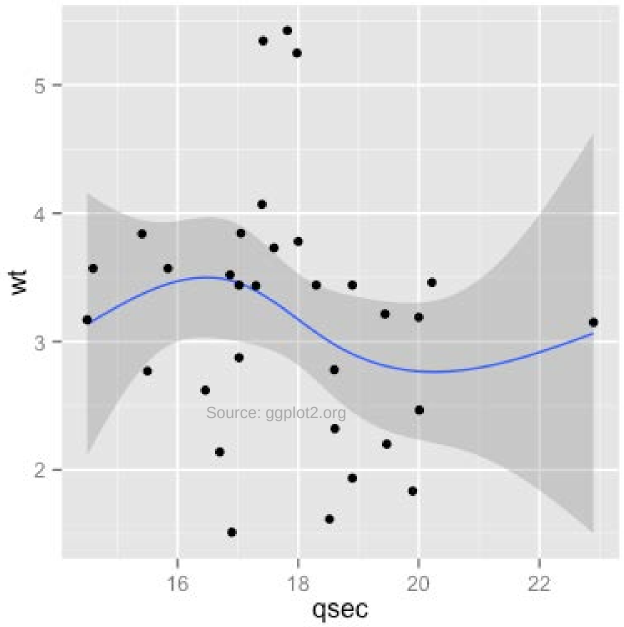
Bootstrap: regression context

Parametric bootstrap

1. Fit a model to D → get $\hat{w}(D)$.
2. Set $X_i^* = X_i$, generate $Y_i^* \sim F(X_i, \hat{w})$.
3. $D_i = \{(X_i^*, Y_i^*), i = 1, \dots, n\}$
4. Repeat step 2 B times

Confindence intervals in regression

- Given $Y \sim Distribution(y|x,w), EY|X = \mu|x = f(x,w)$
  - Example: $Y \sim N(w^T x, \sigma^2), \mu|x = f(x,w) = w^T x$
- Estimate intervals for $\mu|x = f(x,w)$ for many X, combine in a **confidence band**



- What is estimator?
  - $\mu|x = f(x,w)$

Confindence intervals in regression

Estimation

1. Compute $D_1, \ldots D_B$ using a bootstrap
2. Fit model to $D_1, \ldots D_B$ →estimate $\widehat{w}_1, \ldots \widehat{w}_B$
3. For a given X, compute $f(X, \widehat{w}_1), \ldots f(X, \widehat{w}_B)$ and estimate confidence interval by (percentile method)
4. Combine confidence intervals in a band

Bootstrap: R

- Package **boot**
  - **Functions:**
    - boot()
    - boot.ci() – 1 parameter
    - envelope() – many parameters
- Random random generation for parametic bootstrap:
  - Rnorm()
  - Runif()
  - …

boot(data, statistic, R, sim = "ordinary", ran.gen = function(d, p) d, mle = NULL,…)

Nonparametric bootstrap:

- Write a function *statistic* that depends on *dataframe* and *index* and returns the estimator

```
library(boot)
data2=data[order(data$Area),]#reordering data according
to Area

# computing bootstrap samples
f=function(data, ind){
  data1=data[ind,]# extract bootstrap sample
  res=lm(Price~Area, data=data1) #fit linear model
  #predict values for all Area values from the original
data
  priceP=predict(res,newdata=data2)
  return(priceP)
}
res=boot(data2, f, R=1000) #make bootstrap
```

Bootstrap: R

Parametric bootstrap:

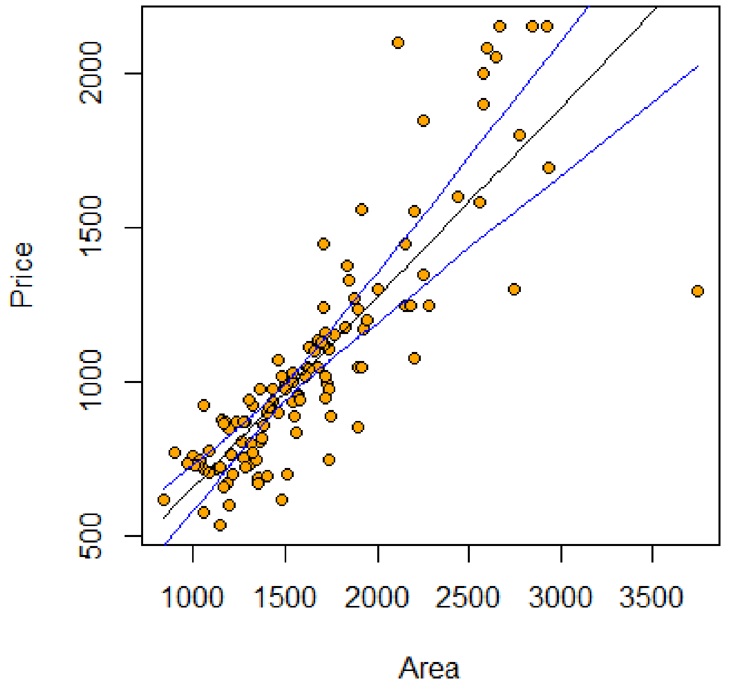
- Compute value *mle* that estimates model parameters from the data
- Write function *ran.gen* that depends on *data* and *mle* and which generates new data
- Write function *statistic* that depend on *data* which will be generated by *ran.gen* and should return the estimator

```
mle=lm(Price~Area, data=data2)

rng=function(data, mle) {
  data1=data.frame(Price=data$Price,
Area=data$Area)
  n=length(data$Price)
#generate new Price
  data1$Price=rnorm(n,predict(mle,
newdata=data1),sd(mle$residuals))
  return(data1)
}

f1=function(data1){
  res=lm(Price~Area, data=data1) #fit linear
model
  #predict values for all Area values from
the original data
  priceP=predict(res,newdata=data2)
  return(priceP)
}

res=boot(data2, statistic=f1, R=1000,
mle=mle,ran.gen=rng, sim="parametric")
```

Uncertainty estimation: R

- Bootstrap cofidence bands for linear model

```
e=envelope(res) #compute confidence bands

fit=lm(Price~Area, data=data2)
priceP=predict(fit)

plot(Area, Price, pch=21, bg="orange")
points(data2$Area,priceP,type="l") #plot fitted line

#plot cofidence bands
points(data2$Area,e$point[2,], type="l", col="blue")
points(data2$Area,e$point[1,], type="l", col="blue")
```



Prediction bands

- Confidence interval for Y|X= interval for mean $EY|X$
- Prediction interval for Y|X= interval for $Y|X$

$$Y{\sim}Distribution(x,w)$$

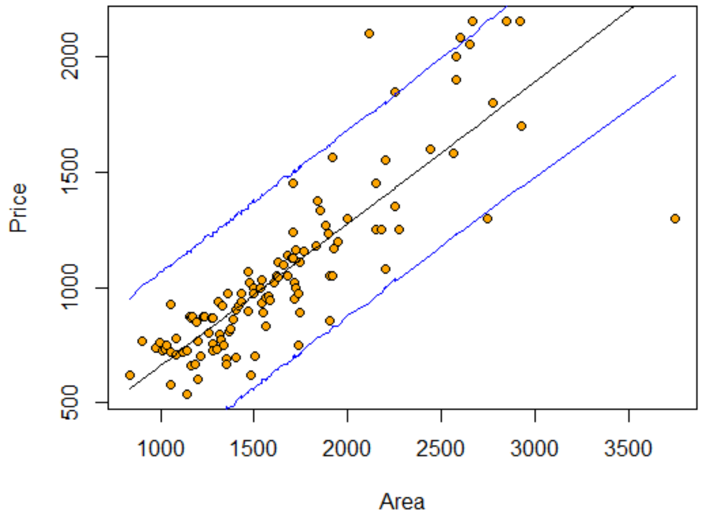
Prediction band for parametric bootstrap

1. Run parametric bootstrap and get $D_1, \ldots D_B$
2. Fit the model to the data and get $\hat{w}(D_1), \ldots \hat{w}(D_B)$
3. For each X, generate from $Distribution\left(X, \hat{w}(D_1)\right), \ldots$ $Distribution\left(X, \hat{w}(D_B)\right)$ and apply percentile method
4. Connect the intervals→get the band

Estimation of the model quality

Example: parametric bootstrap

```
mle=lm(Price~Area, data=data2)

f1=function(data1){
  res=lm(Price~Area, data=data1) #fit linear
model
  #predict values for all Area values from the
original data
  priceP=predict(res,newdata=data2)
  n=length(data2$Price)
  predictedP=rnorm(n,priceP, sd(mle$residuals))
  return(predictedP)
}
res=boot(data2, statistic=f1, R=10000,
mle=mle,ran.gen=rng, sim="parametric")
```



Why wider band?