Analisi Colonial Broadcasting - Tesina

Luca Bajardi e Francesca Collini

31/07/2020

Carichiamo i dati leggendo il file csv e settiamo il seme del generatore pseudo-casuale così da avere i risultati sempre uguali.

```
rm(list =ls())
set.seed(1)
CBC = read.csv(file = "02 - Dati per caso Colonial Broadcasting.csv", header=T)
attach(CBC)
```

Il dataset che stiamo analizzando ha 88 osservazioni e 16 variabili:

```
dim(CBC)
```

```
## [1] 88 16
```

Le variabili si chiamano:

```
names(CBC)
```

```
##  [1] "network"     "fact"        "stars"       "month"       "day"
##  [6] "rating"      "prevratings" "competition" "bbs"         "abn"
## [11] "oct"         "dec"         "aprmay"      "mon"         "sun"
## [16] "march"
```

Vediamo l'inizio del dataset per vedere com'è fatto:

```
head(CBC)
```

```
##   network fact stars month day rating prevratings competition bbs abn oct
## 1     BBS    0     1     1   1   15.6        14.2        14.5   1   0   0
## 2     BBS    1     0     1   7   10.8        15.3        17.2   1   0   0
## 3     BBS    0     1     1   7   14.1        13.8        14.4   1   0   0
## 4     BBS    1     1     1   1   16.8        12.8        15.3   1   0   0
## 5     BBS    1     1     2   1   14.3        12.4        13.3   1   0   0
## 6     BBS    1     1     2   1   17.1        12.9        15.1   1   0   0
##   dec aprmay mon sun march
## 1   0      0   1   0     0
## 2   0      0   0   1     0
## 3   0      0   0   1     0
## 4   0      0   1   0     0
## 5   0      0   1   0     0
## 6   0      0   1   0     0
```

La variabile "Network" è categorica e può assumere tre diversi valori a seconda della rete sulla quale viene trasmesso il programma. Le variabili "Month" e "Day" indicano rispettivamente il mese ed il giorno della settimana in cui avviene la trasmissione e quindi sono entrambe categoriche. La variabile "Fact" è binaria e caratterizza i film, distinguendo quelli che sono basati su fatti reali da quelli di pura fantasia, mentre la variabile "Stars" è un valore quantitativo discreto che rappresenta il numero di attori che hanno ricevuto

un ingaggio superiore a $300,000. Infine le informazioni "Previous rating" e "Competition" sono varibili quantitative continue che si riferiscono il primo al valore di rating del programma immediatamente precedente mandato in onda sulla stessa rete, e l'altro è una media dei valori dei rating degli altri programmi mandati in onda contemporaneamente dalle altre due reti. Utilizzando questi valori vorremo prevedere il valore di rating del film che stiamo considerando, che è espresso appunto dal valore della variabile "Rating".

Vediamo inoltre che non ci sono elementi NA (Not Available), quindi possiamo usare tutte le osservazioni nel dataset:

```
sum(is.na(CBC$rating))
```

```
## [1] 0
```

Abbiamo a disposizione un dataset piccolo, che contiene solamente 88 osservazioni che vorremmo utilizzare sia per formulare il nostro modello di previsione (training set), sia per testarlo per capire quanto sono accurate le nostre previsioni. Per questo motivo, scegliamo arbitrariamente di considerare come test set il 30% del dataset totale e come training la parte rimanente.

```
train=sample(88,88*0.7)
```

Consideriamo il dataset CBC, ma per la formulazione del modello di regressione prendiamo solo il sottoinsieme train:

```
lm.fit=lm(rating~prevratings,data=CBC,subset=train)
```

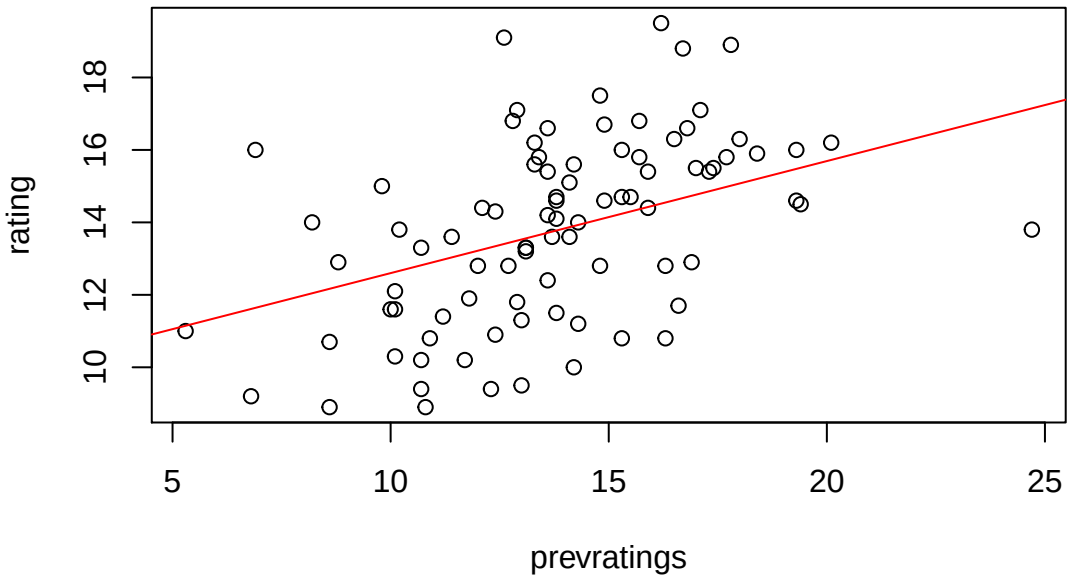
Dal summary vediamo che "prevratings" ha un coefficiente di regressione positivo. Questo fatto ha un senso logico perché possiamo suppore che quando il programma precedente ha un rating più alto, le persone rimangono sul quel canale e non lo cambino dopo la fine di quest'ultimo. Inoltre la variabile "prevratings", avendo un p-value sufficientemente piccolo, risulta significativa.

```
summary(lm.fit)
```

```
##
## Call:
## lm(formula = rating ~ prevratings, data = CBC, subset = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0270 -1.8332 -0.0239  1.7018  5.6967
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.50644    1.26880   7.492 3.96e-10 ***
## prevratings  0.30928    0.08843   3.498 0.000899 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.405 on 59 degrees of freedom
## Multiple R-squared:  0.1717, Adjusted R-squared:  0.1577
## F-statistic: 12.23 on 1 and 59 DF,  p-value: 0.0008991
```

Però il modello non spiega tutta la variabilità, infatti il valore R^2 è solo 0.1717. L'obiettivo è quello di massimizzare questo valore, in modo da poter predire il valore del rating con la maggiore accuratezza possibile. Possiamo vedere anche nel `plot`, che non tutta la variabilità è rappresentata.

```
plot(prevratings, rating)
abline(lm.fit, col="red")
```



Per capire la qualità della nostra previsione e per misurare l'errore, possiamo considerare il valore atteso del quadrato della deviazione, il MSE (Mean Square Error):

$$\mathbf{E}[(y-\widehat{y})^2] = \mathbf{E}^2[(f(x_0)-\widehat{f}(x_0)] + (\sigma_\epsilon)^2 + Var[\widehat{f}(x_0)]$$

dove $y = f(x_0) + \epsilon$ e $\widehat{y} = \widehat{f}(x_0)$. Il primo termine viene chiamato bias, il secondo è l'errore irriducibile dovuto al rumore interinseco presente nel sistema, mentre l'ultimo termine rappresenta la varianza.

Possiamo calcolare il MSE andando a validare questo modello sul sottoinsieme di test. L'errore viene definito come il rapporto tra la somma, su tutte le osservazioni presenti nel test set, del quadrato dell'errore sulle singole osservazioni, diviso per la cardinalità dell'insieme test.

$$MSE = \frac{\sum_{i \in test} (y_i - \widehat{y}_i)^2}{|test|}$$

```
mean((rating-predict(lm.fit,CBC))[-train]^2)
```

```
## [1] 3.489889
```

Il MSE di un solo modello non è una misura molto indicativa perchè non abbiamo termini di paragone, quindi dobbiamo calcolarlo anche per altri modelli e valutare quale sia il migliore andando a selezionare quello con l'errore più piccolo.

```
potenze = c(1,2,3,4,5,6,10,16)
ma<-matrix(nrow=3,ncol=length(potenze)+1)
ma[1,1]="potenze"
ma[2,1]="MSE"
ma[3,1]="R^2"
ma[1,2:(length(potenze)+1)]=potenze
for (i in 2:(length(potenze)+1)){
  pwr = potenze[i-1]
  lm.fit_n= lm(rating~poly(prevratings,pwr),data=CBC,subset=train)
  ma[2,i] = mean((rating-predict(lm.fit_n,CBC))[-train]^2)
  ma[3,i] = summary(lm.fit_n)$r.squared
}
ma
```

```
##      [,1]      [,2]                [,3]                [,4]
## [1,] "potenze" "1"                 "2"                 "3"
## [2,] "MSE"     "3.48988949215875"  "3.40803317223908"  "3.00023440403935"
## [3,] "R^2"     "0.171735197954496" "0.182204759148948" "0.221588843537537"
##      [,5]                [,6]                [,7]
## [1,] "4"                 "5"                 "6"
## [2,] "2.98907487503185"  "3.04344984117172"  "3.04871129507942"
## [3,] "0.221661737092959" "0.234124907653851" "0.24364159603347"
##      [,8]                [,9]
## [1,] "10"                "16"
## [2,] "3.29680579710106"  "2.94827384158756"
## [3,] "0.268680720924043" "0.356467229038818"
```

Possiamo vedere che, aumentando il grado dei polinomi in funzione di `prevratings`, non diminuisce il MSE, ma aumenta l'R^2, questo significa che stiamo facendo overfitting. Infatti con l'aumento della complessità del modello, R^2 nel sample, mal che vada rimane lo stesso perchè sto aggiungendo altri modi per spiegare la variabile che voglio prevedere, quando però quando testiamo il modello su un insieme esterno, questo non è necessariamente vero perchè contemporaneamente sta aumentando la varianza.

Poichè il nostro dataset è piuttosto piccolo per eseguire un test indipendente, la strategia della cross-validation può essere seguita. La tecnica consiste nel suddividere in K sottoinsiemi disgiunti i nostri dati, per ogni sottoinsime $k-esimo$ formulare il modello e calcolare il MSE_k utilizzando i dati di partenza andando però ad escludere il sottoinsieme in questione, testare poi questo modello su quest'ultimo. Così ogni sottoinsieme svolge $K-1$ volte il ruolo di training set e esattamente una volta il ruolo di test set. Il MSE complessivo del modello è:

$$CV(K) = \sum_{k=1}^{K} \frac{n_k}{n} MSE_k.$$

Nel caso limite in cui $K = n$, utilizziamo l'approccio Leave-One-Out Cross Validation. Utilizzando quindi $K = 88$, calcoliamo l'errore di ogni osservazione per la previsione fatta andando ad escludere quel dato e poi facendo semplicemente la media su tutti i valori.

```
sample.size = length(prevratings) #decido la dimensione del campione
cv.errors = numeric(sample.size) #prealloco un vettore di zeri
for (k in 1:sample.size){
  fitCV=lm(rating~prevratings,data=CBC[-k,]) #tolgo un'osservazione
  cv.errors[k]= ((rating-predict(fitCV,CBC))[k])^2
  #vado a prevedere solo per quell'osservazione
}
mean(cv.errors) #calcolo la media
```

```
## [1] 5.169679
```

Un metodo alternativo è quello di utilizzare la funzione "cv.glm" dove nel vettore Delta troviamo, nella prima posizione, lo stesso risultato precedente e nella seconda un termine che è leggermente migliorato perchè tiene conto dei possibili errori di approssimazione.

```
library(boot)
glm.fit=glm(rating~prevratings,data=CBC)
cv.err=cv.glm(CBC,glm.fit) #passa la struttura dati e il modello
names(cv.err)
```

```
## [1] "call"  "K"     "delta" "seed"
```

```
cv.err$K
```

```
## [1] 88
```

```r
cv.err$delta
```

```
## [1] 5.169679 5.168069
```

```r
#proviamo con polinomi con grado massimo diverso
cv.error=rep(0,8) #numeric(8)
j=1
for (i in potenze){ #LOOCV su 8 modelli
  glm.fit=glm(rating~poly(prevratings,i),data=CBC)
  cv.error[j]=cv.glm(CBC,glm.fit)$delta[1]
  j=j+1
}
cv.error
```

```
## [1] 5.169679e+00 5.436229e+00 5.102095e+00 6.600530e+00 1.584170e+01
## [6] 7.520611e+01 1.080856e+05 2.239373e+11
```

Proviamo inoltre ad utilizzare la K-fold Cross Validation. In questo caso specifico, dividiamo l'intero dataset in $K = 10$ sottogruppi ed ogni volta utilizziamo un sottoinsieme diverso che svolge la funzione di test set. Questo metodo risulta computazionalmente più veloce perchè andiamo a confrontare meno osservazioni.

```r
set.seed(20)
#divido in 10 gruppi che ovviamente non saranno della stessa cardinalità
cv.error.10=rep(0,8)
j=1
for (i in potenze){
  glm.fit=glm(rating~poly(prevratings,i),data=CBC)
  cv.error.10[j]=cv.glm(CBC,glm.fit,K=10)$delta[1]
  j=j+1
}
cv.error.10
```

```
## [1] 5.213349e+00 5.520776e+00 5.404177e+00 1.030911e+01 6.671234e+01
## [6] 6.876285e+02 9.527265e+05 2.486979e+12
```

```r
detach(CBC)
```

Questi diversi esperimenti, probabilmente ci suggeriscono che utilizzare solo la variabile "Prevratings" potrebbe non essere sufficiente per prevedere il rating. Con la funzione `regsubsets` possiamo, in modo automatico, selezionare quelle variabili che sono le migliori per predire la variabile risposta, quando viene fissato il numero totale di variabili da utilizzare nel modello. Il termine "migliore" si riferisce alle prestazioni, guardando la RSS. Utilizziamo il dataset eliminando la colonna che contiene la media dei rating di programmi mandati in onda dalle reti concorrenti. Infatti, questo valore è molto difficile da prevedere e potrebbe essere rischioso includerlo nel modello in quanto potrebbe portare a conclusioni sbagliate.

```r
library(leaps)
CBC$network=as.factor(CBC$network)
CBC$day=as.factor(CBC$day)
CBC$month=as.factor(CBC$month)
CBCr=CBC[, 1:7]
regfit.full=regsubsets(rating~.,CBCr,nvmax=15)
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(rating ~ ., CBCr, nvmax = 15)
## 15 Variables  (and intercept)
##             Forced in Forced out
## networkBBS      FALSE      FALSE
```

```
## networkCBC      FALSE      FALSE
## fact            FALSE      FALSE
## stars           FALSE      FALSE
## month2          FALSE      FALSE
## month3          FALSE      FALSE
## month4          FALSE      FALSE
## month5          FALSE      FALSE
## month9          FALSE      FALSE
## month10         FALSE      FALSE
## month11         FALSE      FALSE
## month12         FALSE      FALSE
## day2            FALSE      FALSE
## day7            FALSE      FALSE
## prevratings     FALSE      FALSE
## 1 subsets of each size up to 15
## Selection Algorithm: exhaustive
##           networkBBS networkCBC fact stars month2 month3 month4 month5
## 1  ( 1 )  " "        " "        " "  " "   " "    " "    " "    " "
## 2  ( 1 )  " "        " "        "*"  " "   " "    " "    " "    " "
## 3  ( 1 )  " "        " "        "*"  " "   " "    " "    " "    " "
## 4  ( 1 )  " "        " "        "*"  "*"   " "    " "    " "    " "
## 5  ( 1 )  " "        " "        "*"  "*"   " "    " "    " "    " "
## 6  ( 1 )  "*"        " "        "*"  "*"   " "    " "    " "    " "
## 7  ( 1 )  "*"        " "        "*"  "*"   " "    " "    " "    " "
## 8  ( 1 )  "*"        " "        "*"  "*"   " "    " "    "*"    " "
## 9  ( 1 )  "*"        "*"        "*"  "*"   " "    " "    "*"    " "
## 10  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    " "
## 11  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    "*"
## 12  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    "*"
## 13  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
## 14  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
## 15  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
##           month9 month10 month11 month12 day2 day7 prevratings
## 1  ( 1 )  " "    " "     " "     " "     " "  " "  "*"
## 2  ( 1 )  " "    " "     " "     " "     " "  " "  "*"
## 3  ( 1 )  " "    " "     " "     "*"     " "  " "  "*"
## 4  ( 1 )  " "    " "     " "     "*"     " "  " "  "*"
## 5  ( 1 )  " "    "*"     " "     "*"     " "  " "  "*"
## 6  ( 1 )  " "    "*"     " "     "*"     " "  " "  "*"
## 7  ( 1 )  " "    "*"     " "     "*"     "*"  " "  "*"
## 8  ( 1 )  " "    "*"     " "     "*"     "*"  " "  "*"
## 9  ( 1 )  " "    "*"     " "     "*"     "*"  " "  "*"
## 10  ( 1 ) " "    "*"     " "     "*"     "*"  " "  "*"
## 11  ( 1 ) " "    "*"     " "     "*"     "*"  " "  "*"
## 12  ( 1 ) " "    "*"     " "     "*"     "*"  "*"  "*"
## 13  ( 1 ) " "    "*"     " "     "*"     "*"  "*"  "*"
## 14  ( 1 ) "*"    "*"     " "     "*"     "*"  "*"  "*"
## 15  ( 1 ) "*"    "*"     "*"     "*"     "*"  "*"  "*"
```

```
reg.summary=summary(regfit.full)
```

Analizzando il risultato, osserviamo che quando fissiamo ad uno il numero di predittori, la variabile migliore che possiamo utilizzare è appunto "Prevratings", man mano che aumentiamo questo numero a questa si aggiungono tutte le altre, quando raggiungiamo il numero totale dei predittori che è uguale a 15.

```
reg.summary$rsq
```

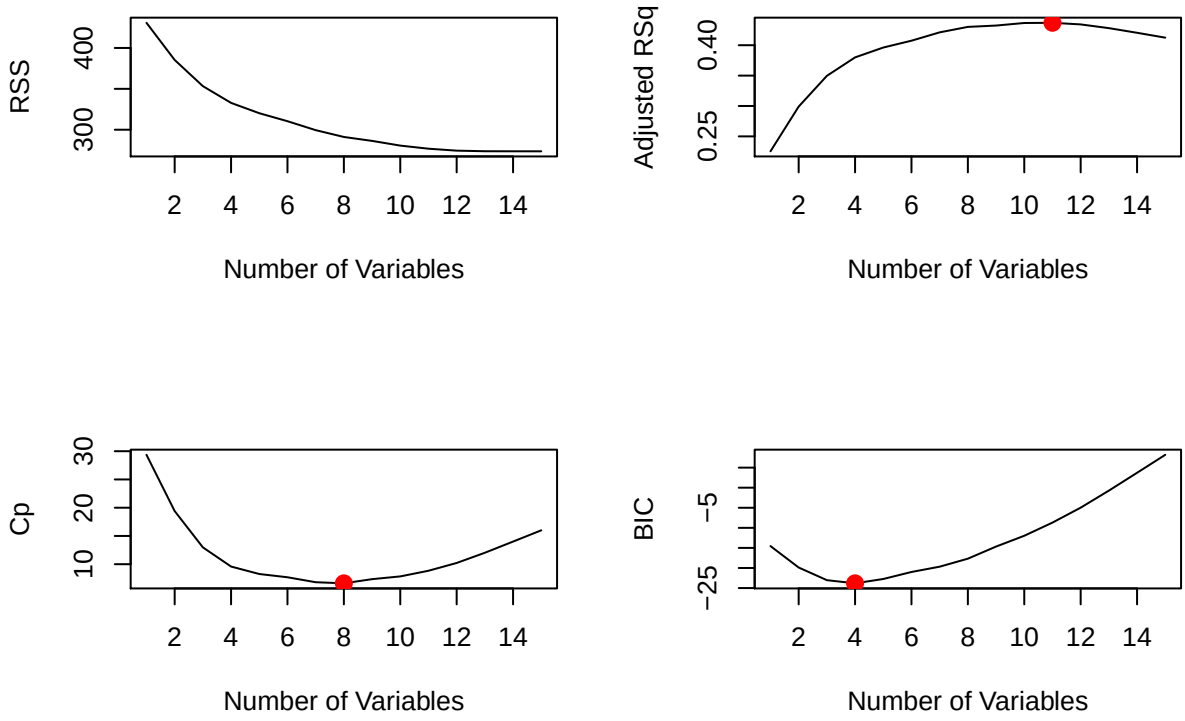
```
##  [1] 0.2343602 0.3151354 0.3720341 0.4084705 0.4308890 0.4482936 0.4677308
##  [8] 0.4825366 0.4910659 0.5013114 0.5080425 0.5122618 0.5135400 0.5137033
## [15] 0.5137192
```

Osserviamo che, come ci potremmo aspettare, l'R^2 aumenta in maniera monotona all'aumentare delle variabili, ma questo non necessariamente ci fa concludere che il modello in cui utilizziamo un maggior numero di variabili è il migliore. Per questo sarebbe opportuno considerare altri indici, al fine di capire qual è il giusto equilibrio tra numero di variabili e proporzione della variabilità spiegata.

Poichè stiamo confrontando modelli con un numero diverso di variabili, guardare solamente l'R^2, potrebbe non fornirci un'informazione utile o quanto meno potremmo avere in mano un'informazione piuttosto distorta. A questo proposito sarebbe utile utilizzare altri indici come l'R^2_{adj}, il C_p e il BIC:

- R^2_{adj} è definito dal seguente rapporto dove q rappresenta il numero di variabili ed n il numero di osservazioni:
$$R^2_{adj} = 1 - \frac{\frac{SSR}{(n-q-1)}}{\frac{TSS}{(n-1)}}.$$
Osserviamo che al crescere di q il denominatore diminuisce e se non scende abbastanza velocemente anche il numeratore, il modello potrebbe essere peggiore quando quelle variabili stanno spiegando del rumore. Quindi mentre quando aggiungiamo delle variabili l'R^2 non può diminuire, può succedere per l'R^2_{adj}.
- C_p, definendo $p = q + 1$ il numero totale di coefficienti dato dalla somma del numero di variabili e la costante. Definiamo inoltre σ^2 la stima dell'errore della varianza e n sempre il numero di osservazioni, il Mallow's C_p, si calcola come:
$$C_p = \frac{1}{n}(SSR + 2p\sigma^2)$$
- BIC (Bayesian Information Criterion) è definito da:
$$BIC = \frac{1}{n}(SSR + log(n)p\sigma^2)$$
In questo caso l'uso di più variabili viene penalizzato maggiormente rispetto all'indice C_p perchè in genere il logaritmo del numero delle variabili tende ad essere più grande di due. Per questo motivo quindi in linea di principio, utilizzando questo parametro, otterremo un modello più conservativo.

```
par(mfrow=c(2,2))
plot(reg.summary$rss,xlab="Number of Variables",ylab="RSS",type="l")
plot(reg.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
t=which.max(reg.summary$adjr2)
points(t,reg.summary$adjr2[t], col="red",cex=2,pch=20)
plot(reg.summary$cp,xlab="Number of Variables",ylab="Cp",type='l')
q=which.min(reg.summary$cp)
points(q,reg.summary$cp[q],col="red",cex=2,pch=20)
u=which.min(reg.summary$bic)
plot(reg.summary$bic,xlab="Number of Variables",ylab="BIC",type='l')
points(u,reg.summary$bic[u],col="red",cex=2,pch=20)
```



Analizzando i diversi grafici ed andando ad individuare i punti di ottimo (il massimo per l'R^2_{adj} e il minimo per gli altri due indici), osserviamo che secondo l'R^2_{adj} il numero ottimale da scegliere è 11, scegliendo il criterio del C_p, invece questo scende ad 8, ed si riduce infine a 4, andando a guardare il BIC. Per scegliere il migliore sottoinsieme possibile, possiamo utilizzare nuovamente la cross-validation.

```
# Choosing Among Models using the cross-validation approach
set.seed(1)
#selezioniamo a caso tra le osservazioni a nostra disposizione
train=sample(c(TRUE,FALSE), nrow(CBCr),rep=TRUE)
test=(!train) #complemento
regfit.best=regsubsets(rating~.,data=CBCr[train,],nvmax=15)
test.mat=model.matrix(rating~.,data=CBCr[test,])
#estrae dal dateset è la matrice X, potrebbe servire per i calcoli
val.errors=rep(NA,15)
for(i in 1:15){
  coefi=coef(regfit.best,id=i)
  pred=test.mat[,names(coefi)]%*%coefi
  #prodotto righe per colonne
  val.errors[i]=mean((CBCr$rating[test]-pred)^2)
}
val.errors
```

```
##  [1] 4.511783 6.196545 5.177542 5.789412 6.522034 5.724100 6.336633
##  [8] 5.653779 5.275389 5.086785 5.134484 6.021948 5.717762 6.132468
## [15] 6.196214
```

```
min=which.min(val.errors)
min
```

```
## [1] 1
```

```
coef(regfit.best,min)
```

```
## (Intercept) prevratings
##   8.6498631   0.3609965
```

```
#attributes(CBCr)
```

Andando ad osservare tutti i possibili modelli, quello migliore si ottiene utilizzando solamente la variabile "Prevratings" perchè probabilmente le altre variabili non aggiungono troppi miglioramenti alle informazioni che abbiamo già ricavato. Metodi alternativi per selezionare il miglior sottoinsieme di variabili, sono quelli della selezione greedy. Possiamo procedere o con la selezione "Forward", dove partiamo dal modello nullo ed aggiungiamo una variabile per volta, scegliendo quella che porta al più alto miglioramento, oppure con la selezione "backward", in cui seguiamo la strada al contrario, quindi partiamo dal modello completo andando ad eliminare una variabile alla volta.

```
regfit.fwd=regsubsets(rating~.,data=CBCr,nvmax=15,method="forward")
summary(regfit.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(rating ~ ., data = CBCr, nvmax = 15, method = "forward")
## 15 Variables  (and intercept)
##             Forced in Forced out
## networkBBS      FALSE      FALSE
## networkCBC      FALSE      FALSE
## fact            FALSE      FALSE
## stars           FALSE      FALSE
## month2          FALSE      FALSE
## month3          FALSE      FALSE
## month4          FALSE      FALSE
## month5          FALSE      FALSE
## month9          FALSE      FALSE
## month10         FALSE      FALSE
## month11         FALSE      FALSE
## month12         FALSE      FALSE
## day2            FALSE      FALSE
## day7            FALSE      FALSE
## prevratings     FALSE      FALSE
## 1 subsets of each size up to 15
## Selection Algorithm: forward
##           networkBBS networkCBC fact stars month2 month3 month4 month5
## 1  ( 1 )  " "        " "        " "  " "   " "    " "    " "    " "
## 2  ( 1 )  " "        " "        "*"  " "   " "    " "    " "    " "
## 3  ( 1 )  " "        " "        "*"  " "   " "    " "    " "    " "
## 4  ( 1 )  " "        " "        "*"  "*"   " "    " "    " "    " "
## 5  ( 1 )  " "        " "        "*"  "*"   " "    " "    " "    " "
## 6  ( 1 )  "*"        " "        "*"  "*"   " "    " "    " "    " "
## 7  ( 1 )  "*"        " "        "*"  "*"   " "    " "    " "    " "
## 8  ( 1 )  "*"        " "        "*"  "*"   " "    " "    "*"    " "
## 9  ( 1 )  "*"        "*"        "*"  "*"   " "    " "    "*"    " "
## 10  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    " "
## 11  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    "*"
## 12  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    "*"
## 13  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
## 14  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
## 15  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
##           month9 month10 month11 month12 day2 day7 prevratings
## 1  ( 1 )  " "    " "     " "     " "     " "  " "  "*"
```

```
## 2  ( 1 )  " "     " "      " "      " "      " "   " "   "*"
## 3  ( 1 )  " "     " "      " "      "*"      " "   " "   "*"
## 4  ( 1 )  " "     " "      " "      "*"      " "   " "   "*"
## 5  ( 1 )  " "     "*"      " "      "*"      " "   " "   "*"
## 6  ( 1 )  " "     "*"      " "      "*"      " "   " "   "*"
## 7  ( 1 )  " "     "*"      " "      "*"      "*"   " "   "*"
## 8  ( 1 )  " "     "*"      " "      "*"      "*"   " "   "*"
## 9  ( 1 )  " "     "*"      " "      "*"      "*"   " "   "*"
## 10  ( 1 ) " "     "*"      " "      "*"      "*"   " "   "*"
## 11  ( 1 ) " "     "*"      " "      "*"      "*"   " "   "*"
## 12  ( 1 ) " "     "*"      " "      "*"      "*"   "*"   "*"
## 13  ( 1 ) " "     "*"      " "      "*"      "*"   "*"   "*"
## 14  ( 1 ) "*"     "*"      " "      "*"      "*"   "*"   "*"
## 15  ( 1 ) "*"     "*"      "*"      "*"      "*"   "*"   "*"
```

```
regfit.bwd=regsubsets(rating~.,data=CBCr,nvmax=15,method="backward")
summary(regfit.bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(rating ~ ., data = CBCr, nvmax = 15, method = "backward")
## 15 Variables  (and intercept)
##             Forced in Forced out
## networkBBS      FALSE      FALSE
## networkCBC      FALSE      FALSE
## fact            FALSE      FALSE
## stars           FALSE      FALSE
## month2          FALSE      FALSE
## month3          FALSE      FALSE
## month4          FALSE      FALSE
## month5          FALSE      FALSE
## month9          FALSE      FALSE
## month10         FALSE      FALSE
## month11         FALSE      FALSE
## month12         FALSE      FALSE
## day2            FALSE      FALSE
## day7            FALSE      FALSE
## prevratings     FALSE      FALSE
## 1 subsets of each size up to 15
## Selection Algorithm: backward
##          networkBBS networkCBC fact stars month2 month3 month4 month5
## 1  ( 1 )  " "        " "        " "  " "   " "    " "    " "    " "
## 2  ( 1 )  " "        " "        "*"  " "   " "    " "    " "    " "
## 3  ( 1 )  " "        " "        "*"  " "   " "    " "    " "    " "
## 4  ( 1 )  " "        " "        "*"  "*"   " "    " "    " "    " "
## 5  ( 1 )  " "        " "        "*"  "*"   " "    " "    " "    " "
## 6  ( 1 )  "*"        " "        "*"  "*"   " "    " "    " "    " "
## 7  ( 1 )  "*"        " "        "*"  "*"   " "    " "    " "    " "
## 8  ( 1 )  "*"        " "        "*"  "*"   " "    " "    "*"    " "
## 9  ( 1 )  "*"        "*"        "*"  "*"   " "    " "    "*"    " "
## 10  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    " "
## 11  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    "*"
## 12  ( 1 ) "*"        "*"        "*"  "*"   "*"    " "    "*"    "*"
## 13  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
## 14  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
## 15  ( 1 ) "*"        "*"        "*"  "*"   "*"    "*"    "*"    "*"
```

```
##           month9 month10 month11 month12 day2 day7 prevratings
## 1  ( 1 )  " "    " "     " "     " "     " "  " "  "*"
## 2  ( 1 )  " "    " "     " "     " "     " "  " "  "*"
## 3  ( 1 )  " "    " "     " "     "*"     " "  " "  "*"
## 4  ( 1 )  " "    " "     " "     "*"     " "  " "  "*"
## 5  ( 1 )  " "    "*"     " "     "*"     " "  " "  "*"
## 6  ( 1 )  " "    "*"     " "     "*"     " "  " "  "*"
## 7  ( 1 )  " "    "*"     " "     "*"     "*"  " "  "*"
## 8  ( 1 )  " "    "*"     " "     "*"     "*"  " "  "*"
## 9  ( 1 )  " "    "*"     " "     "*"     "*"  " "  "*"
## 10  ( 1 ) " "    "*"     " "     "*"     "*"  " "  "*"
## 11  ( 1 ) " "    "*"     " "     "*"     "*"  " "  "*"
## 12  ( 1 ) " "    "*"     " "     "*"     "*"  "*"  "*"
## 13  ( 1 ) " "    "*"     " "     "*"     "*"  "*"  "*"
## 14  ( 1 ) "*"    "*"     " "     "*"     "*"  "*"  "*"
## 15  ( 1 ) "*"    "*"     "*"     "*"     "*"  "*"  "*"
```

###Se proviamo ad osservare i coefficienti per uno specifico numero di variabili, vediamo che non coincidono esattamente però i valori sono comunque molto vicini.

```
coef(regfit.full,6)
```

```
## (Intercept)  networkBBS        fact       stars     month10     month12
##   8.0275543  -0.8739989   2.1263890   1.0278787  -1.2362843   1.8301880
## prevratings
##   0.3383274
```

```
coef(regfit.fwd,6)
```

```
## (Intercept)  networkBBS        fact       stars     month10     month12
##   8.0275543  -0.8739989   2.1263890   1.0278787  -1.2362843   1.8301880
## prevratings
##   0.3383274
```

```
coef(regfit.bwd,6)
```

```
## (Intercept)  networkBBS        fact       stars     month10     month12
##   8.0275543  -0.8739989   2.1263890   1.0278787  -1.2362843   1.8301880
## prevratings
##   0.3383274
```

A questo punto andiamo a selezionare il modello migliore, andando a trovare quali e quante sono le variabili che ci permettono di prevedere meglio la variabile target. Per questo scopo, andiamo ad utilizzare di nuovo la k-folds cross validation con K=10.

```
k=10
 predict.regsubsets=function(object,newdata,id,...){
  form=as.formula(object$call[[2]])
  mat=model.matrix(form,newdata)
  coefi=coef(object,id=id)
  xvars=names(coefi)
  mat[,xvars]%*%coefi
}
set.seed(1)
folds=sample(1:k,nrow(CBCr),replace=TRUE)
cv.errors=matrix(NA,k,15, dimnames=list(NULL, paste(1:15)))
for(j in 1:k){
  best.fit=regsubsets(rating~.,data=CBCr[folds!=j,],nvmax=15)
```

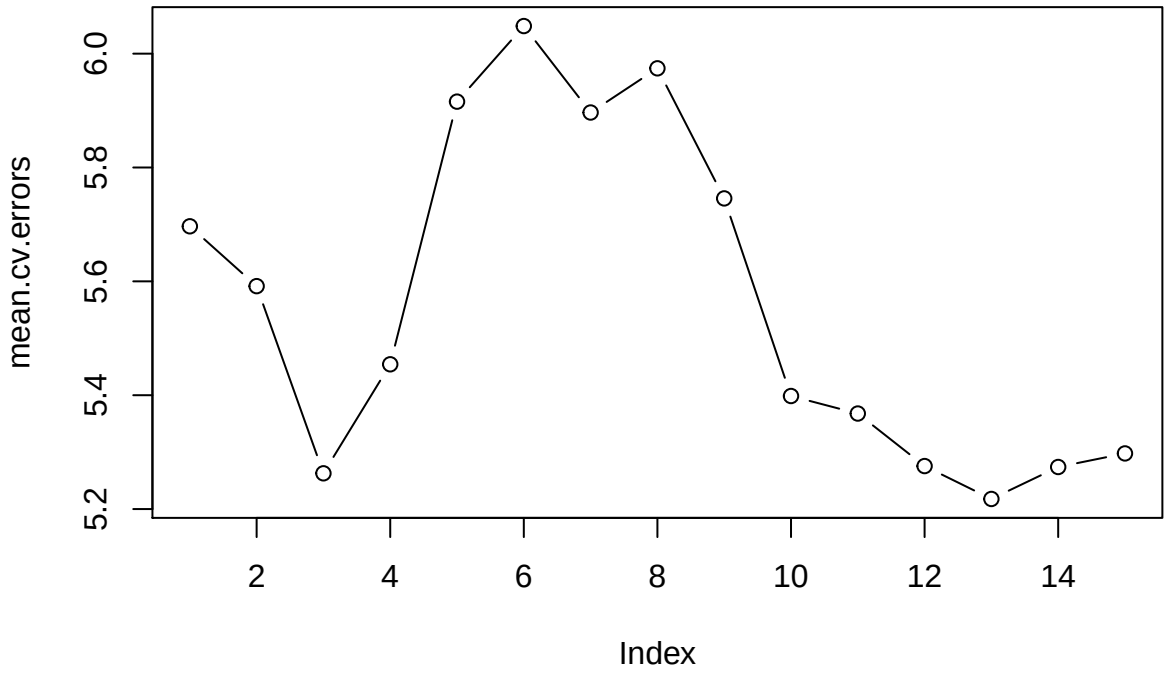
```
  for(i in 1:15){
    pred=predict(best.fit,CBCr[folds==j,],id=i)
    cv.errors[j,i]=mean( (CBC$rating[folds==j]-pred)^2)
  }
}
mean.cv.errors=apply(cv.errors,2,mean)
mean.cv.errors
```

```
##        1        2        3        4        5        6        7        8
## 5.696734 5.591594 5.262762 5.454305 5.915720 6.048468 5.896608 5.974218
##        9       10       11       12       13       14       15
## 5.745726 5.398565 5.367793 5.275533 5.217724 5.273894 5.297743
```

```
min=which.min(mean.cv.errors)
min
```

```
## 13
## 13
```

```
par(mfrow=c(1,1))
plot(mean.cv.errors,type='b')
```




```
reg.best=regsubsets(rating~.,data=CBCr, nvmax=15)
```

```
coef(reg.best,min)
```

```
## (Intercept)  networkBBS  networkCBC        fact       stars      month2
##  12.2962318  -2.3403938  -1.7883086   1.7244258   0.5244193   0.9707485
##      month3      month4      month5     month10     month12        day2
##   0.3001762  -1.0854441  -0.9618590  -1.1316915   1.7740989  -2.2338032
##        day7 prevratings
##  -0.4875536   0.1779626
```

```
reg.best=regsubsets(rating~network+fact+stars*month*day*prevratings,data=CBCr, nvmax=num_var_max, metho
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 38 linear dependencies found
```

```
## Reordering variables and trying again:
```

—>

Regressione Ridge e Lasso

Nel metodo dei minimi quadrati standard, la funzione obiettivo da minimizzare è:

$$\sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{q}\beta_j X_j^{(k)}\right)^2.$$

Quando però vogliamo diminuire il valore della varianza e quindi penalizzare i coefficienti di regressione troppo grandi per cercare di evitare l'overfitting, possiamo introdurre dei termini aggiuntivi alla funzione obiettivo:

- Un metodo è quello che viene chiamato **Ridge Regression**. In questo caso, la funzione obiettivo diventa:
$$\sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{q}\beta_j X_j^{(k)}\right)^2 + \lambda\sum_{j=1}^{q}\beta_j^2.$$
Andando ad aumentare il coefficiente di penalità λ, facciamo tendere al limite i coefficienti a zero, facendo crescere il bias, ma eventualmente riducendo il MSE a causa della riduzione della varianza. Con questa norma sui coefficienti andiamo a penalizzare di più i coefficienti grandi.
- Un metodo alternativo è quello della **Lasso Regression**. In questo caso anzichè essere penalizzati con la norma L_2 come nel caso precedente, i coefficienti vengono penalizzati con la norma L_1, quindi la formula diventa:
$$\sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{q}\beta_j X_j^{(k)}\right)^2 + \lambda\sum_{j=1}^{q}|\beta_j|.$$
In questo caso penalizziamo maggiornemnte i β più piccoli, infatti il comportamento di questa regressione è molto simile alla precedente, però adesso riusciamo direttamente a fare la selezione di variabili perchè alcuni coefficienti vanno a zero anche quando la penalità λ assume un valore finito.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 3.0-2
```

```
CBC = read.csv(file = "02 - Dati per caso Colonial Broadcasting.csv", header=T)
x=model.matrix(rating~.,CBCr)[,-1] #tolgo l'intercetta
#perchè Beta_0 non va penalizzato
y=CBC$rating
grid=10^seq(10,-2,length=100)
```

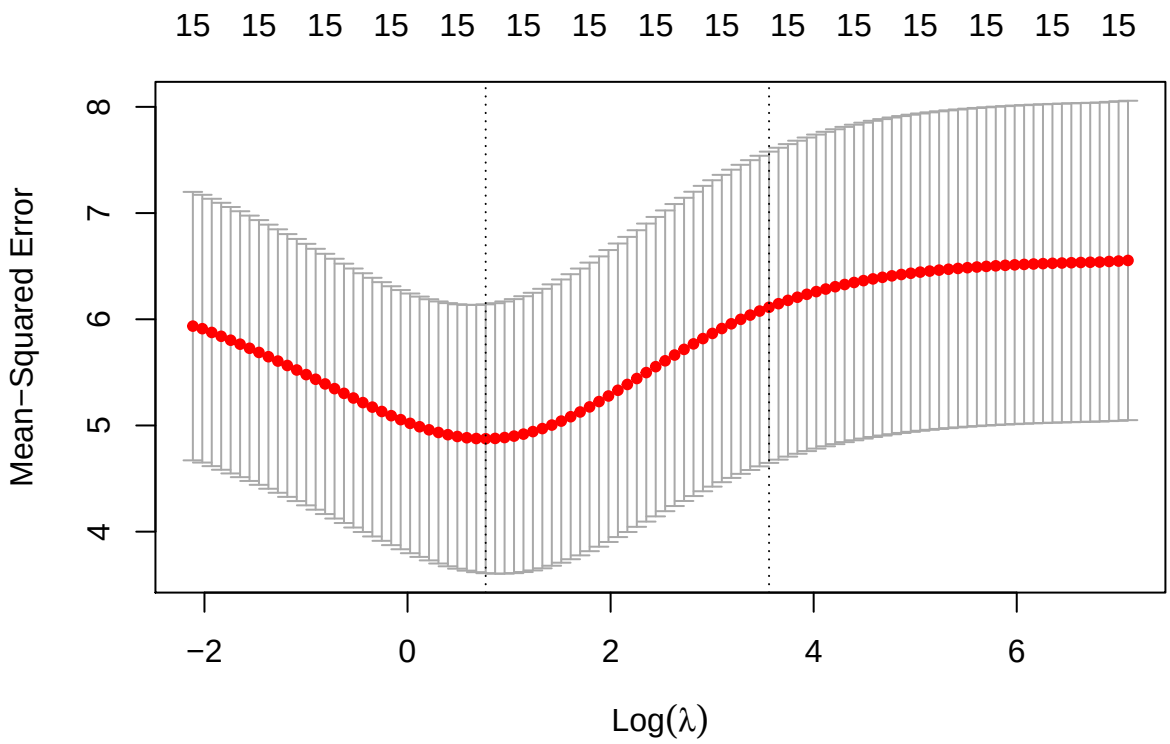
La funzione `glmnet` implementa sia i due casi estremi di regressione ridge e lasso, sia regressioni intermedie a seconda del valore di α che fissiamo.

```
ridge.mod=glmnet(x,y,alpha=0,lambda=grid)
dim(coef(ridge.mod))
```

```
## [1]  16 100
```

In questo modo abbiamo ottenuto 100 diversi modelli di regressione, uno per ogni valore di λ. Invece di utilizzare tutti i dati per la costruzione del modello, potremmo dividerli in un insieme di training e uno di test e utilizzare la cross-validation per scegliere il miglior valore di λ

```
set.seed(1)
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)
y.test=y[test]
cv.out=cv.glmnet(x[train,],y[train],alpha=0)
plot(cv.out)
```




```
#MSE in funzione di lamda per trovare quello migliore
#c'è una certa variabilità nella stima
bestlam=cv.out$lambda.min
bestlam
```

```
## [1] 2.158988
```

```
ridge.pred=predict(ridge.mod,s=bestlam,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

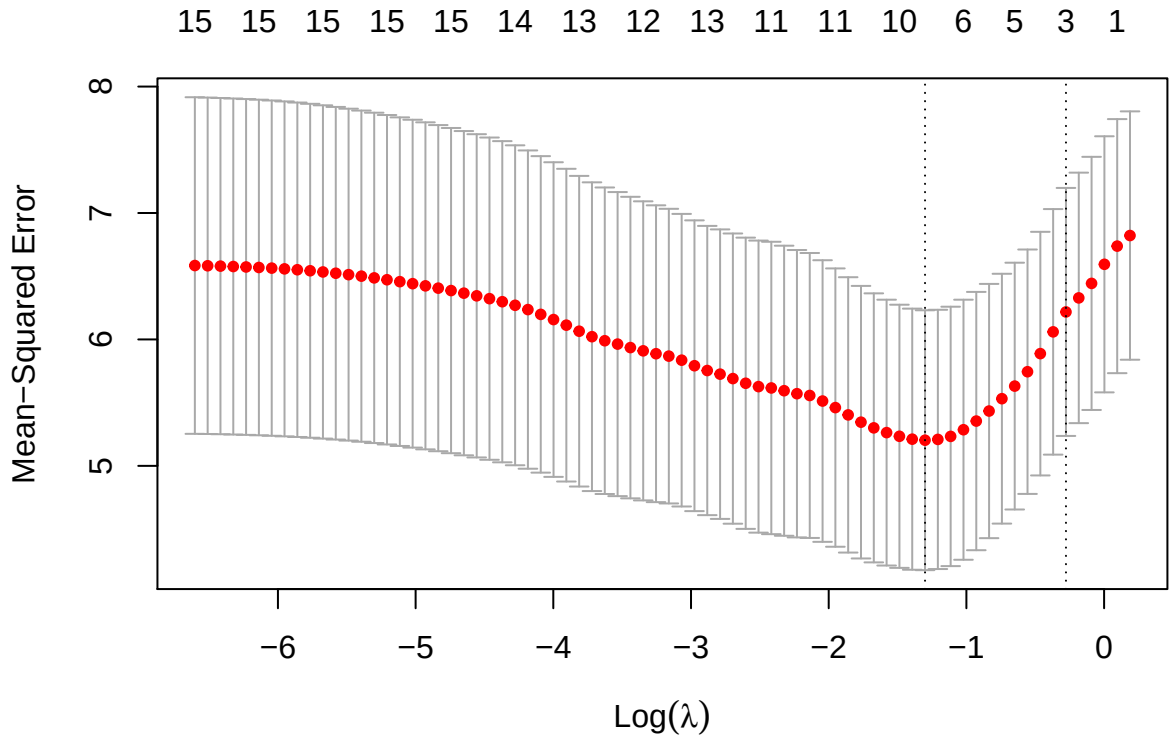
```
## [1] 3.660313
```

```
#dopo che ho capito che questo è il modello migliore, vado a
#prevederlo con tutti i dati
out=glmnet(x,y,alpha=0)
predict(out,type="coefficients",s=bestlam)[1:16,]
```

```
## (Intercept)  networkBBS  networkCBC        fact       stars      month2
##  11.3325066  -0.7228168  -0.3036844   0.8264924   0.3279908   0.7077498
##      month3      month4      month5      month9     month10     month11
##   0.3545764  -0.7483114  -0.4719883  -0.1273826  -0.5779363   0.1329403
##     month12        day2        day7 prevratings
```

```
##   0.9475162  -0.4318434   0.1298016   0.1667785
```

```
lasso.mod=glmnet(x[train,],y[train],alpha=1,lambda=grid)
set.seed(1)
cv.out=cv.glmnet(x[train,],y[train],alpha=1)
plot(cv.out)
```




```
bestlam=cv.out$lambda.min
bestlam
```

```
## [1] 0.2724334
```

```
lasso.pred=predict(lasso.mod,s=bestlam,newx=x[test,])
mean((lasso.pred-y.test)^2)
```

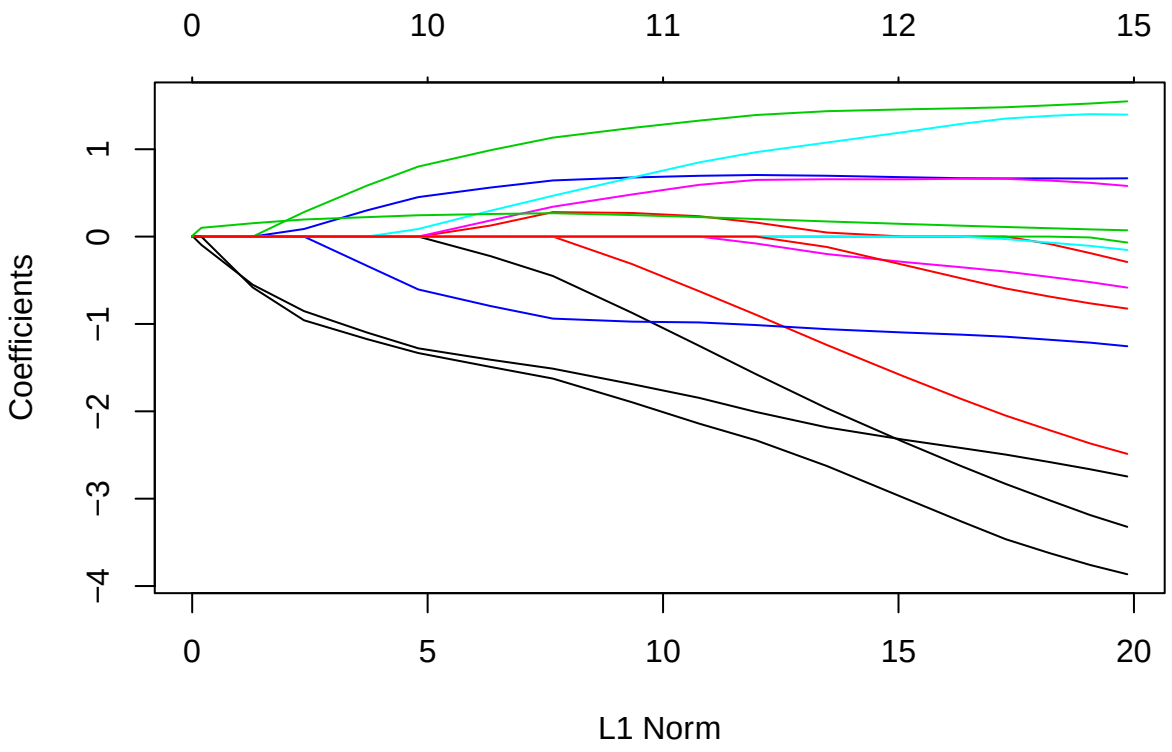
```
## [1] 5.084048
```

```
out=glmnet(x,y,alpha=1,lambda=grid)
lasso.coef=predict(out,type="coefficients",s=bestlam)[1:16,]
lasso.coef
```

```
##  (Intercept)   networkBBS   networkCBC         fact        stars
##  9.465159331 -0.221262200  0.000000000  0.962335003  0.183646785
##       month2       month3       month4       month5       month9
##  0.529251453  0.003537758 -0.362448269  0.000000000  0.000000000
##      month10      month11      month12         day2         day7
## -0.342675200  0.000000000  0.896153418 -0.038003733  0.000000000
##  prevratings
##  0.282785990
```

```
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to
## unique 'x' values
```




```
plot(ridge.mod)
```