Pràctica CSI
Machine Learning

Pol Rodoreda i Nicolás Samus

15 de gener de 2017

Índex

1 Introducció

L'objectiu de la pràctica és desenvolupar un model de regressió amb la finalitat de resoldre el problema del iot.

1.1 *Dataset*

Les dades s'extreuen de la pàgina `https://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics`; són dades utilitzades per predir la resistència hidrodinàmica dels iots a partir de les dimensions i la velocitat. Poder predir la resistència en una fase inicial és de gran valor per avaluar el rendiment del vaixell i estimar-ne la potència motor necessària. Els *inputs* principals inclouen les dimensions del casc i la velocitat del vaixell.

El conjunt de les dades compren un total de 308 experiments a gran escala realitzats al laboratori d'hidrodinàmica de Delft. Els experiments inclouen un total de 22 formes de casc.

Els atributs que presenta el *dataset* són:

- Longitudinal position of the center of buoyancy
- Prismatic coefficient
- Length-displacement ratio
- Beam-draught ratio
- Length-beam ratio
- Froude number

La variable a mesurar és la resistència per unitat de pes de desplaçament:

- Residuary resistance per unit weight of displacement

2 Procediment

Seguidament es farà una explicació del procediment utilitzat, els mètodes i el resultat final.

El primer pas és la lectura de l'arxiu `yacht_hydrodynamics.data`, el qual conté les dades sobre les quals aplicarem els models. Es guarden les dades a la variable `data` i es posa noms a les columnes per treballar amb més comoditat:

```
data <- read.table(file = "yacht_hydrodynamics.data")
names(data) <- c('LonPos', 'PrismCoeff', 'LenDisRatio', '↩
   BeamDRatio', 'LenBRatio', 'FroudNum', 'Resistance')
```

La figura 1 mostra les 15 primeres línies del *dataset*.

	LonPos	PrismCoeff	LenDisRatio	BeamDRatio	LenBRatio	FroudNum	Resistance
1	-2.3	0.568	4.78	3.99	3.17	0.125	0.11
2	-2.3	0.568	4.78	3.99	3.17	0.150	0.27
3	-2.3	0.568	4.78	3.99	3.17	0.175	0.47
4	-2.3	0.568	4.78	3.99	3.17	0.200	0.78
5	-2.3	0.568	4.78	3.99	3.17	0.225	1.18
6	-2.3	0.568	4.78	3.99	3.17	0.250	1.82
7	-2.3	0.568	4.78	3.99	3.17	0.275	2.61
8	-2.3	0.568	4.78	3.99	3.17	0.300	3.76
9	-2.3	0.568	4.78	3.99	3.17	0.325	4.99
10	-2.3	0.568	4.78	3.99	3.17	0.350	7.16
11	-2.3	0.568	4.78	3.99	3.17	0.375	11.93
12	-2.3	0.568	4.78	3.99	3.17	0.400	20.11
13	-2.3	0.568	4.78	3.99	3.17	0.425	32.75
14	-2.3	0.568	4.78	3.99	3.17	0.450	49.49
15	-2.3	0.569	4.78	3.04	3.64	0.125	0.04

Figura 1: Vista del dataset

2.1 *Preprocessing*

Observem que el *dataset* sobre el qual es treballa no presenta *missings* a través de la comanda:

```
> which(is.na(data))
integer(0)
```

A més a més, com que les dades son generades en el laboratori a partir de un model, es considera que no hi han *outliers* que puguin afectar a l'estudi, per tant, no és necessari fer un pre-processat de les dades.

2.2 Visualitació del *dataset*

Primer de tot s'escalen les dades par tal de reduir la variància de les variables, de forma que queda una mitjana de 0 i una desviació de 1. Un problema a solventar és que la funció `scale()` transforma la variable **data** a *matrix* i per tant cal tornar a convertir-la a *data.frame*.

```
data <- scale(data)
data <- as.data.frame(data)
```

Seguidament es mostra el diagrama de dispersió (figura 2) de les dades que formen part del *dataset* per extreure'n una primera valoració.

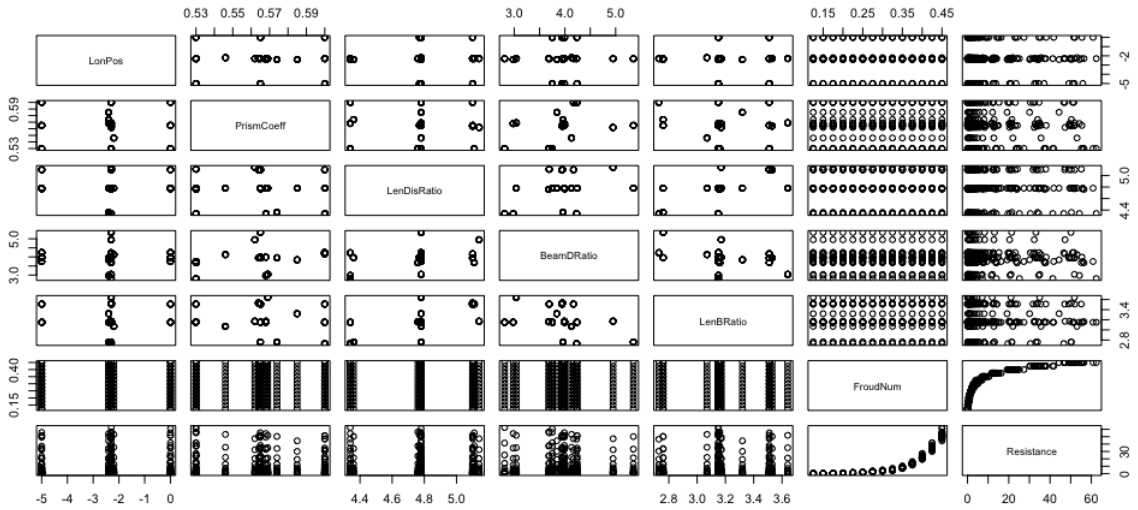

Figura 2: Diagrama de dispersió

Observant el diagrama podem determinar una relació exponencial entre el *Froud Number* i la variable objectiu, la resistència. Pel que fa a la resta de variables no s'observa cap relació determinant amb la variable objectiu.

2.3 Models

Per tal d'entrenar i validar els models, és necessari separar el *dataset*; dues terceres parts es destinaran a l'entrenament (*learn*) i una tercera part a la validació (*test*).

```
set.seed(1)
learn <- sample(1:N, round(2*N/3))
nlearn <- length(learn)
ntest <- N - nlearn
data.learn <- data[learn,]
data.test <- data[-learn,]
```

Els nous conjunts de dates queden distribuïts de la següent manera:

- **data.learn** → 205 files de dades
- **data.test** → 103 files de dades

2.3.1 *Ridge Regression*

El primer model que s'implementa és *Ridge Regression*. La finalitat d'aquest model és solucionar el problema que genera el mètode de mínims quadrats alhora de calcular el

paràmetre β afegint un hiperparàmetre λ. Quan el paràmetre lambda és diferent de 0 ens trobem davant un estimador esbiaixat de beta, el qual contrau les betes cap a 0 sense arribar-hi. Amb aquest procés afegim biaix però reduïm variància.

La importància és trobar el valor adequat de λ, per això s'usa un criteri generalitzat de validació creuada (*generalized cross-validation*). El procés fixa un rang de possibles valors i calcula la validació creuada. El lambda òptim és aquell que minimitza CV.

```
library(car)

model.ridge <- lm.ridge(Resistance ~ ., data=data.learn, ↩
   lambda = seq(0,10,0.1))

plot(seq(0,10,0.1), model.ridge$GCV, main="GCV of Ridge ↩
   Regression", type="l",
     xlab=expression(lambda), ylab="GCV")
```

El resultat obtingut és el representat a la figura 3, on observem un valor òptim de lambda entre 4 i 6.

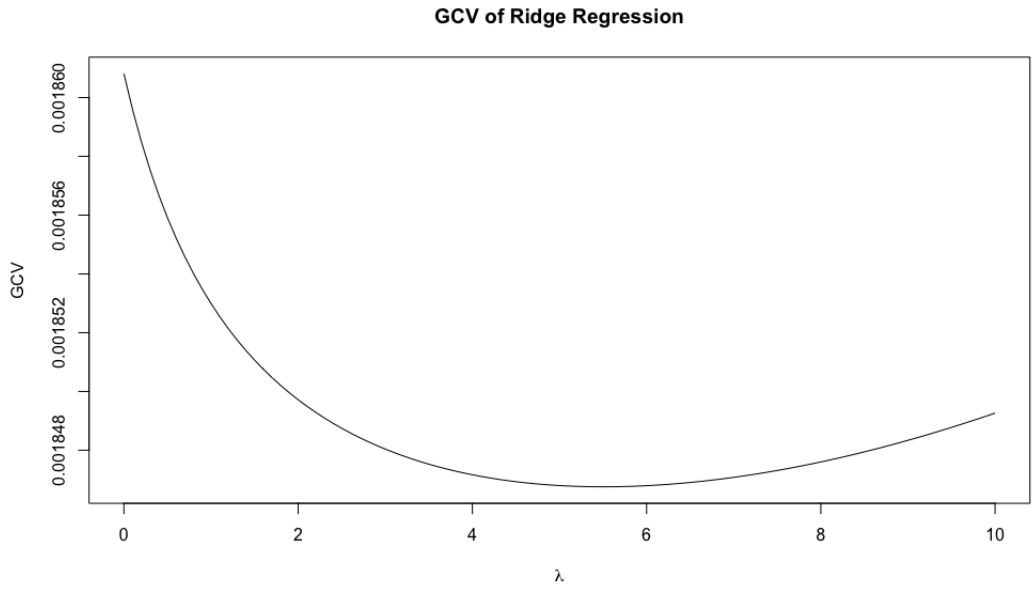

Figura 3: GCV de ridge regression

El resultat exacte l'obtenim a través de la comanda:

```
> lambda.ridge <- seq(0,10,0.1)[which.min(model.ridge$GCV)]
5.5
```

La figura 4 mostra com es modifiquen els paràmetres en funció del valor de λ. Observem la contracció citada anteriorment.

Finalment es torna a entrenar el model, però aquesta vegada utilitzant el valor més òptim de lambda. Els valors de beta resultants es mostren a la figura **??**; aquests valors són

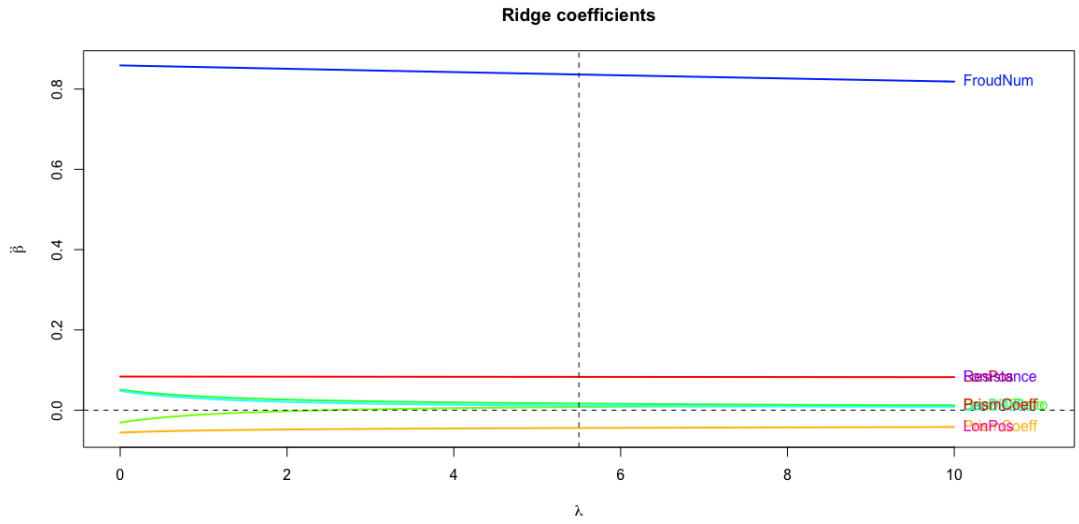

Figura 4: Modificació dels paràmetres en funció de λ

útils alhora de valorar la importància de les diferents variables en relació amb la variable objectiu. En aquest cas, observem que el ***Froud Number*** és la variable més relacionada.

	LonPos	PrismCoeff	LenDisRatio	BeamDRatio	LenBRatio	FroudNum
0.009994	0.083009	-0.044277	0.007961	0.016480	0.011326	0.836230

L'últim pas a realitzar és el càlcul dels errors a la fase d'entrenament i validació:

```
(pred.ridgereg <- sum((data.learn$Resistance - beta.ridgereg↩
   .FINAL[1] - as.matrix(data.learn[,1:6])%*%beta.ridgereg.↩
   FINAL[2:7])^2)/nlearn)*100

(pred.ridgereg <- sum((data.test$Resistance - beta.ridgereg.↩
   FINAL[1] - as.matrix(data.test[,1:6])%*%beta.ridgereg.↩
   FINAL[2:7])^2)/ntest)*100
```

L'error obtingut a l'entrenament és del 36% i l'error obtingut en la validació és del 32%. Es detecta un petit *overfitting* que junt amb un error molt gran, fa que es qualifiqui el model com a incorrecte.

2.3.2 *Lasso Regression*

El següent model a provar és *Lasso Regression*. Aquest model penalitza els coeficients amb la norma L1 i contrau els que influeixen poc sobre la variable objectiu, fent que molts d'ells convergeixin a zero. És per això que podem fer-lo servir per eliminar variables.

```
library(lars)

model.lasso <- lars(as.matrix(data.test[,1:6]), as.numeric(↩
   data.test$Resistance), type="lasso")
```

```
lambda.lasso <- c(model.lasso$lambda, 0)

beta.lasso <- coef(model.lasso)
```

Els coeficients de beta són:

```
LonPos    PrismCoeff  LenDisRatio BeamDRatio LenBRatio  FroudNum
-0.054336 0.000000    -0.004034   0.000000   0.000000   0.612610
```

La figura 5 mostra els valors de beta respecta el valor lambda. Es marca verticalment el valor de lambda amb el qual s'aconsegueixen els valors de beta citats.

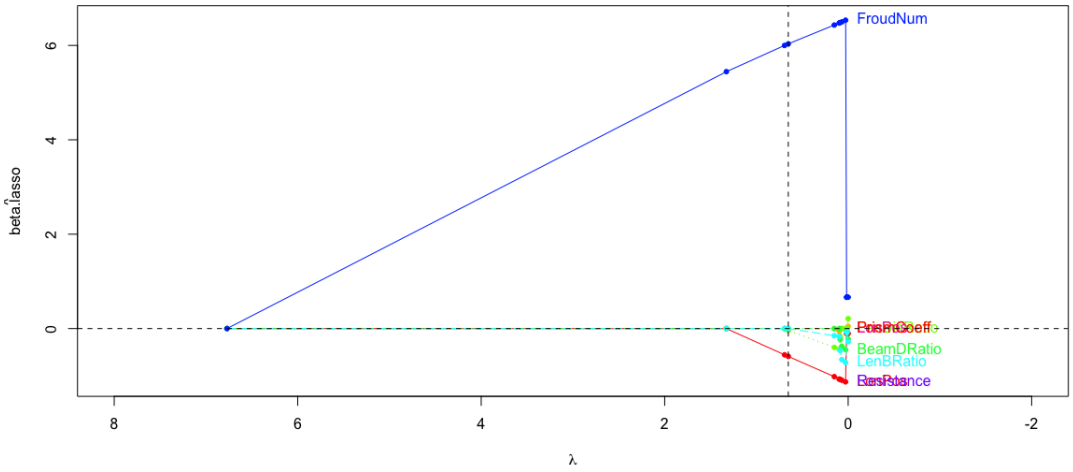

Figura 5: Lasso β

Amb aquesta configuració, el model presenta un error d'entrenament del 45% i un error de validació del 26%. Tornem a detectar *overfitting* en el model i un error massa gran.

2.3.3 *Decision Tree*

Els arbres de classificació i regresió (CART) ens permeten treballar fàcilment amb variables numèriques i categòriques. Entre els seus avantatges podem destacar sobretot la robustesa davant *outliers* i la facilitat d'interpretació. La metodologia consisteix en tres passos:

- Construcció de l'arbre saturat
- Elecció de la mida correcte de l'arbre
- Classificació de noves dades a través de l'arbre construït

La diferència entre els arbres de classificació i regressió és el criteri de divisió dels nodes i el criteri de cost-complexitat per podar-lo.

El mètode que es fa servir en el cas de regressió és `anova`:

```
library(rpart)
```

```
tree <- rpart(data.learn$Resistance ~ ., data=data.learn, ↩
   method = "anova")
```

L'arbre generat es mostra a la figura 6.

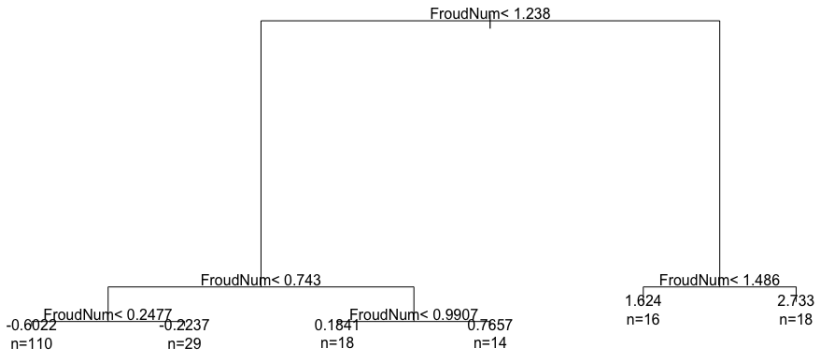

Figura 6: Arbre de decisió

L'error d'entrenament és del 1.8% i el de validació del 1.04%, el que ens indica un model molt bo. Per tant, de moment descartem els dos primers models i provem finalment en crear un *Random Forest*.

2.3.4 *Random Forest*

El model *Random Forest* és un algoritme de classificació i regressió especialment bo per a dades d'altes dimensions. Es pot considerar com la integració de les tècniques: *Decision Tree*, *Bagging* i *Random subspace*.

L'esquema de l'algoritme és el següent:

1. Fent servir reemplaç es crea de forma aleatòria el conjunt d'entrenament[1].
2. A ser una selecció aleatòria, hi ha paràmetres que queden fora de la llista i passen a formar part del conjunt de validació o *Out of bag* (OOB).
3. A cada divisió de l'arbre, es busca la millor variable per dividir les dades sobre un subconjunt m.
4. Fent ús de l'índex de Gini es busca la millor divisió de les dades d'entrenament.
5. Una vegada s'ha entrenat l'arbre, l'avaluació de cada nova entrada es realitza amb el conjunt dels arbres; en el cas de la regressió amb el valor mitja dels resultats.

Les dades OOB es fan servir per determinar la impuresa en els nodes terminals. La suma de les impureses determina la impuresa general de l'arbre.

[1]Igual mida que l'original

```
library(randomForest)

rf <- randomForest(Resistance ~ ., data = data.learn, ntree ↩
   = 500, proximity = FALSE)
```

Importància de les variables

Seguidament es presenta una gràfica que mostra la importància de les diferents variables usades en el model, ordenades de més a menys segons la posició en la gràfica (descendent). Novament observem que la variable **Froud Number** és molt més important que les altres.

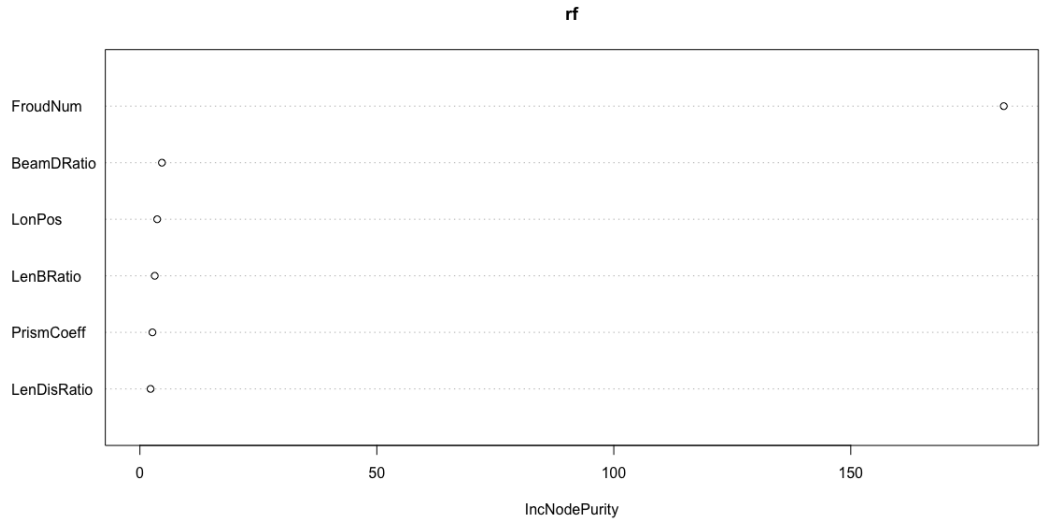

Figura 7: Importància de es variables

Comportament en funció del número d'arbres

La figura 8 il·lustra el comportament generalitzat de l'algoritme en funció del número d'arbres que incorpora. Inicialment, un increment del número d'arbres convergeix en una disminució de l'error, però arriba un punt que la variació s'estanca. Per tant, es torna a generar el model amb un número d'arbres igual a 100 (paràmetre *ntree*).

```
rf <- randomForest(Resistance ~ ., data = data.learn, ntree ↩
   = 100, proximity = FALSE)
```

L'error obtingut a l'entrenament és del 8.7% i l'error obtingut en la validació és del 4.5%. Es detecta *overfitting*, per tant posem en dubte la qualitat del model, tot i que no és un error tan greu com el que hem observat .

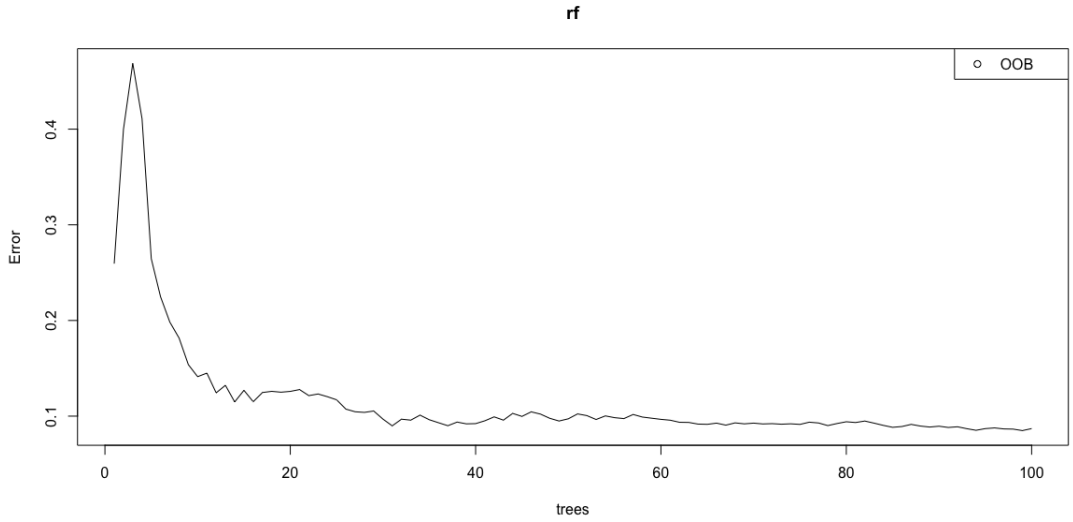

Figura 8: Comportament segons el número d'arbres

3 Conclusions

La taula 1 mostra una comparació dels errors d'entrenament i validació dels diferents models presentats amb la finalitat d'escollir quin és el millor:

Error	Ridge regression	Lasso regression	Decision tree	Random forest
Entrenament	36%	45%	1.8%	8.7%
Validació	32%	26%	1.04%	4.5%

Taula 1: Comparació d'errors

Durant el desenvolupament de l'informe s'ha anat especificant els avantatges i problemes de cada model. Observant l'error el millor model és l'arbre de decisió, però no tan sols el triem per això, sinó perquè ofereix una fàcil interpretació.