Ricardo Aler Mur

Práctica 2. Primero cargamos las librerias necesarias

```
library(knitr)
library(gbm)
library(randomForest)
library(kernlab)
library(lubridate) # install.packages("lubridate")
library(lattice)

opts_chunk$set(cache=FALSE)
options(stringsAsFactors=FALSE)
```

A continuacion leemos los datos de predicción meteorológica

```
windForecasts = read.csv("train_WF1.txt")
# Lo siguiente es para poner las fechas en formato fecha lubridate
windForecasts[, c("date", "dateB")] = lapply(windForecasts[, c("date", "dateB")], ymd_hms)
# Seleccionemos la salida relativa a la primera granja
windForecasts = subset(windForecasts, select=-c(wp2, wp3, wp4, wp5, wp6, wp7))
head(windForecasts)
```

```
##                  date hors    u     v   ws     wd      dateB   wp1 id
## 1 2009-07-01 01:00:00    1 2.34 -0.79 2.47 108.68 2009-07-01 0.085 -2
## 2 2009-07-01 02:00:00    2 2.18 -0.99 2.40 114.31 2009-07-01 0.020 -3
## 3 2009-07-01 03:00:00    3 2.20 -1.21 2.51 118.71 2009-07-01 0.060 -4
## 4 2009-07-01 04:00:00    4 2.35 -1.40 2.73 120.86 2009-07-01 0.045 -5
## 5 2009-07-01 05:00:00    5 2.53 -1.47 2.93 120.13 2009-07-01 0.035 -6
## 6 2009-07-01 06:00:00    6 2.66 -1.29 2.96 115.79 2009-07-01 0.005 -7
##   predecir
## 1       no
## 2       no
## 3       no
## 4       no
## 5       no
## 6       no
```

- 1) Primera Parte - exploración visual de los datos y determinación de atributos relevantes con modelos lineales
- 1.a) Separar 2/3 de los bloques de predicción de 48h para entrenamiento y el otro 1/3 para validación, tal y como se hace en el ejercicio 2 de las transparencias. Usad esta descomposición hasta el fin de la práctica, salvo que se diga lo contrario.

```
# Partimos los datos en bloques de 48h
lista = split(windForecasts, windForecasts$dateB)

set.seed(1) ## Para hacer el resultado replicable
paraEntrenamiento = sample(1:length(lista), round(2/3*length(lista)))

# Partimos en entrenamiento y validacion
entr = lista[paraEntrenamiento]
val =  lista[-paraEntrenamiento]
```

```
# Y lo juntamos todo en el mismo dataframe
entr = do.call(rbind, entr)
val =  do.call(rbind, val)

head(entr)
```

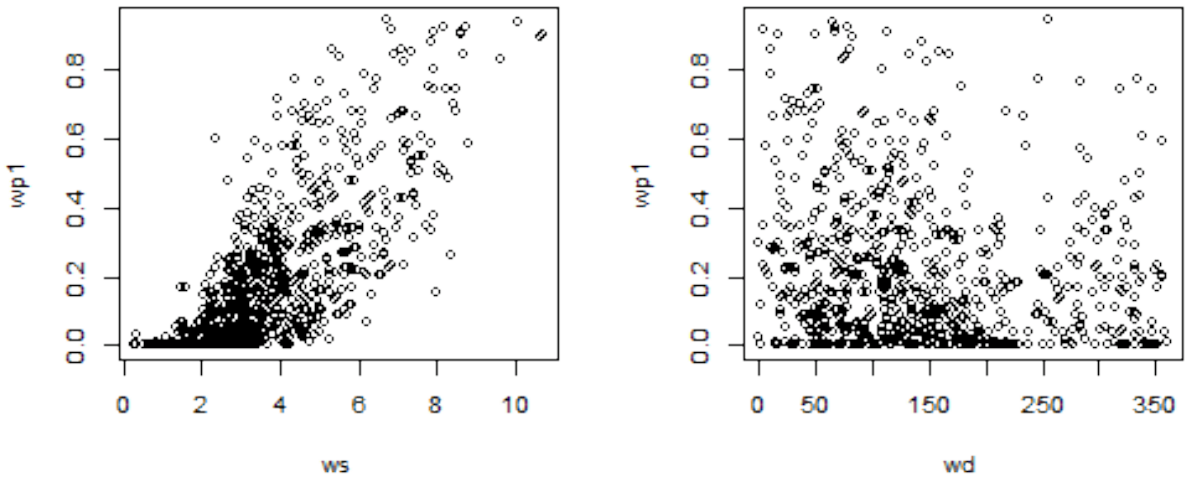
```
##                                          date hors     u    v   ws     wd
## 2009-11-23 12:00:00.13969 2009-11-23 13:00:00    1 -2.57 0.56 2.63 282.40
## 2009-11-23 12:00:00.13970 2009-11-23 14:00:00    2 -2.85 0.65 2.92 282.89
## 2009-11-23 12:00:00.13971 2009-11-23 15:00:00    3 -2.98 1.08 3.17 289.87
## 2009-11-23 12:00:00.13972 2009-11-23 16:00:00    4 -3.01 1.69 3.45 299.38
## 2009-11-23 12:00:00.13973 2009-11-23 17:00:00    5 -3.00 2.30 3.78 307.52
## 2009-11-23 12:00:00.13974 2009-11-23 18:00:00    6 -3.02 2.69 4.04 311.69
##                                            dateB   wp1    id predecir
## 2009-11-23 12:00:00.13969 2009-11-23 12:00:00 0.115 -3494       no
## 2009-11-23 12:00:00.13970 2009-11-23 12:00:00 0.050 -3495       no
## 2009-11-23 12:00:00.13971 2009-11-23 12:00:00 0.030 -3496       no
## 2009-11-23 12:00:00.13972 2009-11-23 12:00:00 0.020 -3497       no
## 2009-11-23 12:00:00.13973 2009-11-23 12:00:00 0.035 -3498       no
## 2009-11-23 12:00:00.13974 2009-11-23 12:00:00 0.090 -3499       no
```

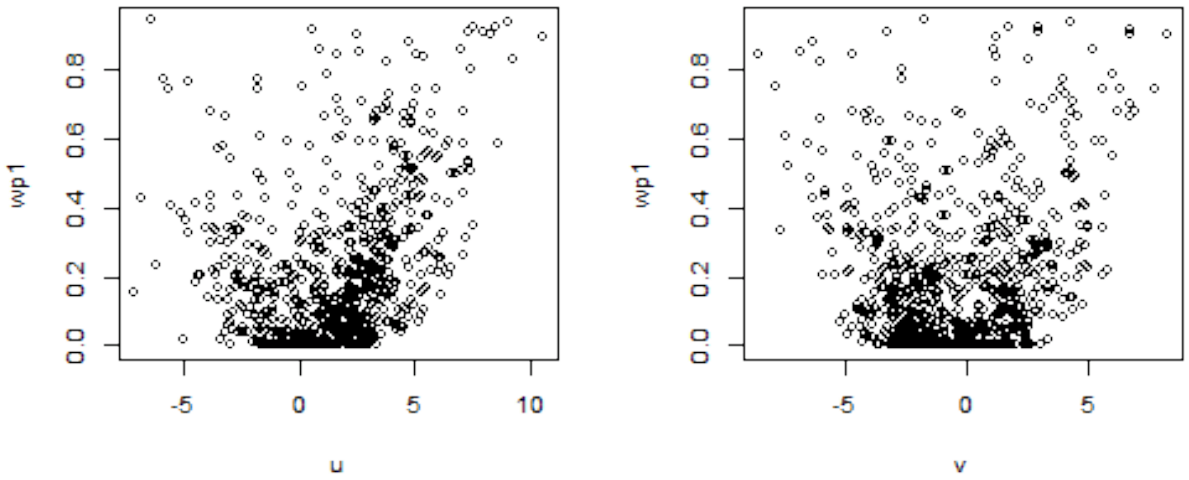
```
head(val)
```

```
##                                       date hors    u     v   ws     wd
## 2009-07-01 00:00:00.1 2009-07-01 01:00:00    1 2.34 -0.79 2.47 108.68
## 2009-07-01 00:00:00.2 2009-07-01 02:00:00    2 2.18 -0.99 2.40 114.31
## 2009-07-01 00:00:00.3 2009-07-01 03:00:00    3 2.20 -1.21 2.51 118.71
## 2009-07-01 00:00:00.4 2009-07-01 04:00:00    4 2.35 -1.40 2.73 120.86
## 2009-07-01 00:00:00.5 2009-07-01 05:00:00    5 2.53 -1.47 2.93 120.13
## 2009-07-01 00:00:00.6 2009-07-01 06:00:00    6 2.66 -1.29 2.96 115.79
##                                 dateB   wp1 id predecir
## 2009-07-01 00:00:00.1 2009-07-01 0.085 -2       no
## 2009-07-01 00:00:00.2 2009-07-01 0.020 -3       no
## 2009-07-01 00:00:00.3 2009-07-01 0.060 -4       no
## 2009-07-01 00:00:00.4 2009-07-01 0.045 -5       no
## 2009-07-01 00:00:00.5 2009-07-01 0.035 -6       no
## 2009-07-01 00:00:00.6 2009-07-01 0.005 -7       no
```

- 1.b) Ahora, coged los datos de entrenamiento y seleccionad aquellos relativos al horizonte temporal de una hora (hors==1) y ved mediante plots las relaciones entre u, v, ws y wd y la producción eléctrica wp. ¿Son las relaciones entre las entradas y la salida wp lineales o no lineales? ¿Hay mucho ruido en la salida?

```
# Seleccionamos los datos relativos al horizonte temporal de 1h
entr1 = subset(entr, hors==1)
val1 = subset(val, hors==1)


par(mfrow=c(2,2))
plot(wp1 ~ ws, data=entr1)
plot(wp1 ~ wd, data=entr1)
plot(wp1 ~ u, data=entr1)
plot(wp1 ~ v, data=entr1)
```

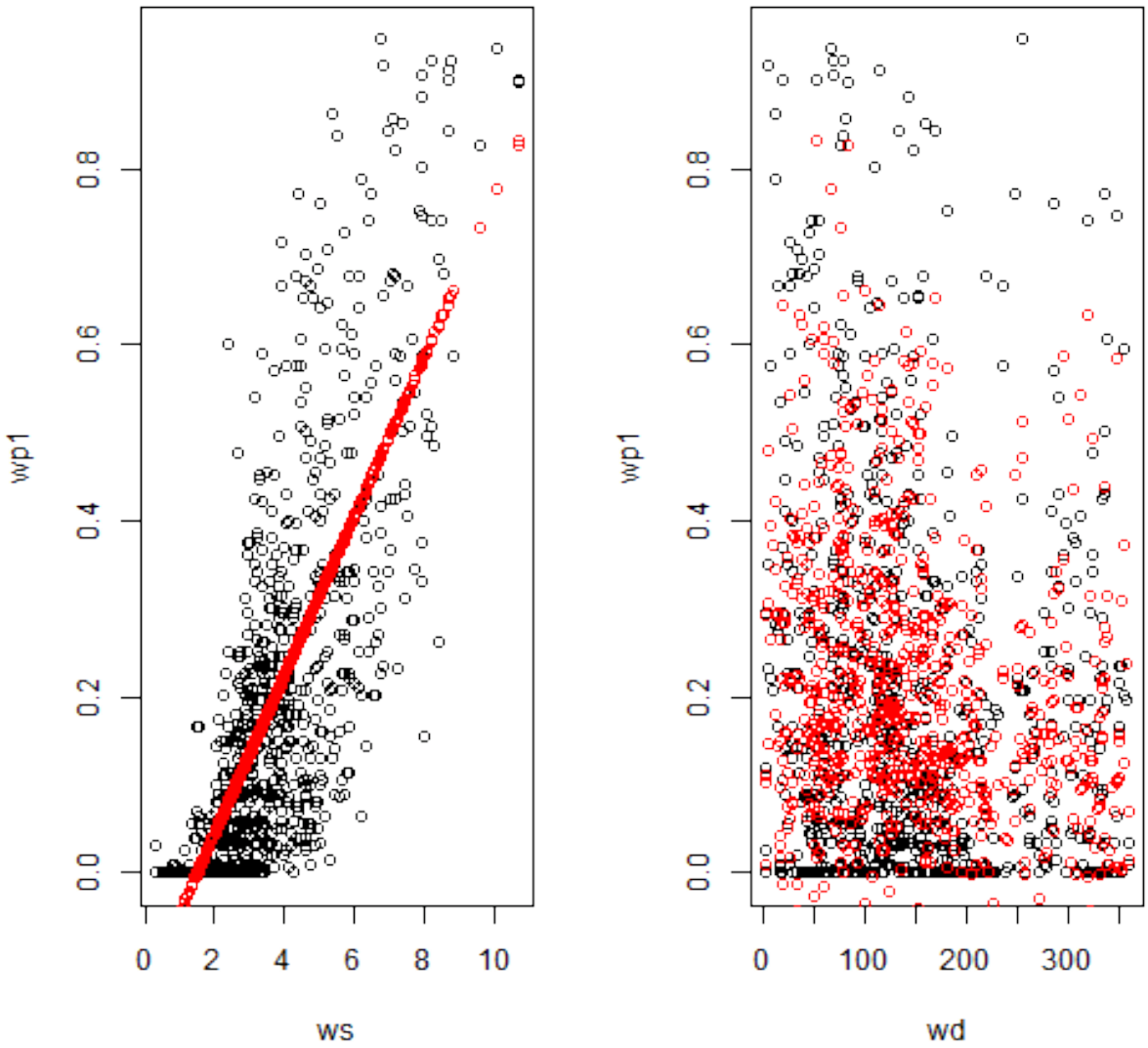





```
set.seed(1)
m = lm(wp1 ~ ws+wd, data=entr1)
p = predict(m, data=entr1)
```

Las relaciones no son lineales, aunque la relación entre ws y wp1 se acerca

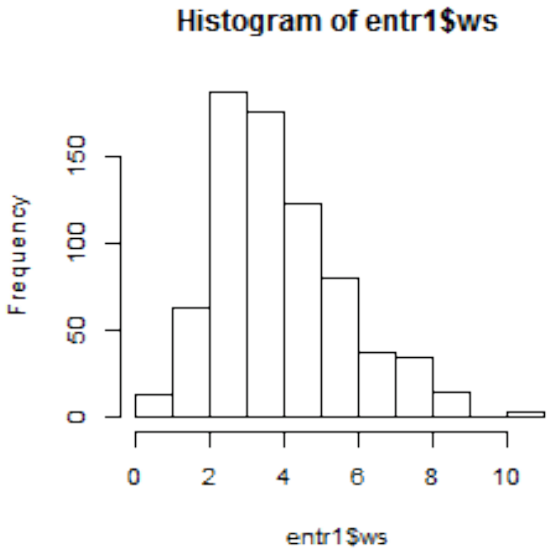
```
par(mfrow=c(1,2))
plot(wp1 ~ ws, data=entr1)
points(p ~ ws, data=entr1, col="red")

plot(wp1 ~ wd, data=entr1)
points(p ~ wd, data=entr1, col="red")
```

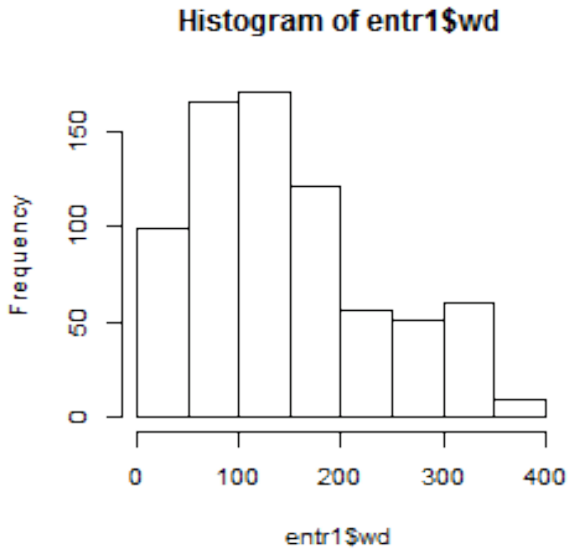


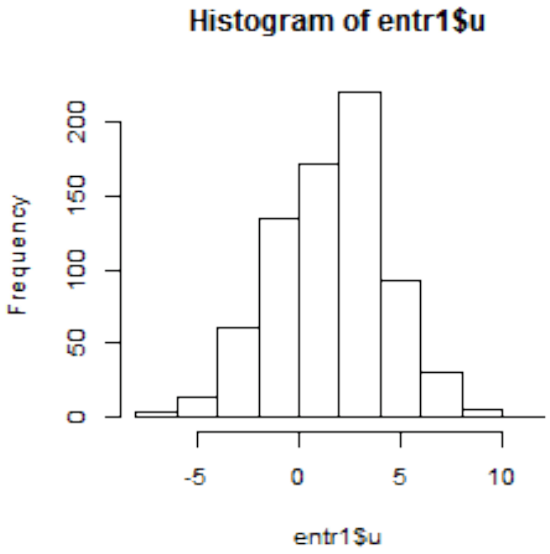

```
par(mfrow=c(1,1))
```

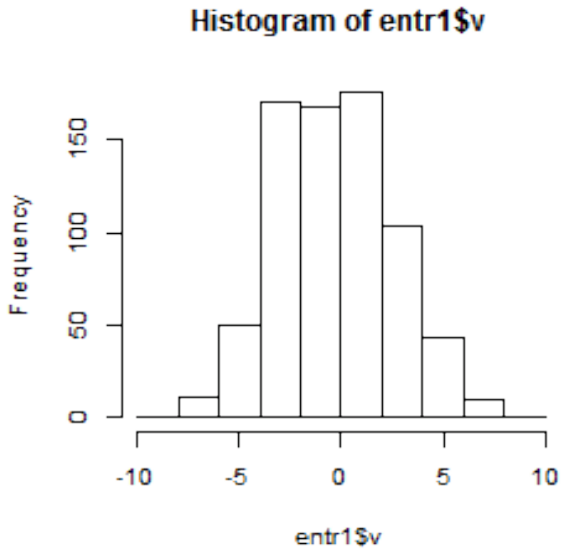
- 1.c) Responder de manera gráfica: ¿todas las velocidades de viento se producen con la misma frecuencia? ¿todas las direcciones de viento se producen con la misma frecuencia?

```
par(mfrow=c(2,2))
hist( entr1$ws)
hist( entr1$wd)
hist( entr1$u)
hist( entr1$v)
```










```
par(mfrow=c(1,1))
```

Ni la velocidad del viento ni su direccion tienen una distribucion uniforme. Existen menos datos con poco y mucho viento que con cantidades de datos intermedias

- 1.d) Con los datos de entrenamiento de c), construid cuatro modelos lineales con lm: lm(wp ~ u), lm(wp ~ v), lm(wp ~ ws), lm(wp ~ wd) y calculad el error cuadrático medio con los datos de validación. ¿Cuál de los atributos parece ser más relevante?

```
errorCuadratico = function(x,y) sqrt(sum((x-y)^2/length(x)))
am = function(x) {
  set.seed(1)
  m1 = lm(as.formula(paste0("wp1 ~ ", x)), data=entr1)
```

```
  p1 = predict(m1, val1)
  errorCuadratico(p1,val1$wp1)
}

valores = c(ws=am("ws"), wd=am("wd"), u=am("u"), v=am("v"))
print(sort(valores))
```

```
##        ws         v        wd         u
## 0.1626617 0.2353768 0.2398306 0.2438073
```

La velocidad del viento es claramente el atributo más predictivo

- 1.e) ¿Hay alguna mejora (con un modelo lineal) si usamos ws + wd? ¿Y si usamos ws + wd mas un tercer atributo? ¿Y si usamos los cuatro atributos? Desde el punto de vista de los modelos lineales, ¿cuántos atributos merecería la pena utilizar?

```
valores2= c(ws_wd=am("ws+wd"), ws_wd_u=am("ws+wd+u"), ws_wd_u_v=am("ws+wd+u+v"))
valores2 = c(valores, valores2)
print(sort(valores2))
```

```
## ws_wd_u_v   ws_wd_u     ws_wd        ws         v        wd         u
## 0.1601961 0.1610981 0.1626475 0.1626617 0.2353768 0.2398306 0.2438073
```

Lo mejor es usar los cuatro atributos, aunque la mejora que aportan los otros tres sobre usar sólo ws es pequeña * 2) Segunda parte - Determinación del mejor tipo de modelo * 2.a) Con los datos de 1.c) entrenad gbm y svm y validadlos con los datos de validación con sus parámetros por omisión. ¿Se consigue mejorar a los modelos lineales?

```
set.seed(1)
mo=lm(wp1 ~ ws+wd+u+v, data=entr1)
errores = c(lm = errorCuadratico(predict(mo, val1), val1$wp1))

set.seed(1)
mo=gbm(wp1 ~ ws+wd+u+v, data=entr1)
```

```
## Distribution not specified, assuming gaussian ...
```

```
errores = c(errores, gbm = errorCuadratico(predict(mo, val1, n.trees=100), val1$wp1))

set.seed(1)
mo=ksvm(wp1 ~ ws+wd+u+v, data=entr1)
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
errores = c(errores, svm = errorCuadratico(predict(mo, val1), val1$wp1))

set.seed(1)
mo=ksvm(wp1 ~ ws+wd+u+v, data=entr1, kernel="vanilladot")
```

```
##  Setting default kernel parameters
```

```
errores = c(errores, svmLineal = errorCuadratico(predict(mo, val1), val1$wp1))

print(sort(errores))
```

```
##        lm svmLineal       svm       gbm
## 0.1601961 0.1631840 0.1728444 0.2313854
```

Con los parámetros por omision, las mejores técnicas son las lineales

- 2.b) Probad también random forest con parámetros por omisión y determinad las variables más relevantes que saca random forest con la función importance. ¿Coincide con lo que salía con los modelos lineales construidos con los atributos individuales?

```
set.seed(1)
mo=randomForest(wp1 ~ ws+wd+u+v, data=entr1)
errores = c(errores, RF = errorCuadratico(predict(mo, val1), val1$wp1))
print(sort(errores))
```

```
##        lm svmLineal        RF       svm       gbm
## 0.1601961 0.1631840 0.1678734 0.1728444 0.2313854
```

Random Forests obtiene resultados similares a los modelos lineales, aunque peores a ellos, pero mejores a gbm y svm

```
importance(mo)[order(importance(mo), decreasing = TRUE),]
```

```
##        ws         u         v        wd
## 12.616239  8.461528  7.125877  4.619914
```

Random Forests coincide que ws es la variable mas importante, pero las ordena de manera diferente a un modelo lineal * 2.c) Ajustad los parámetros n.trees, interaction.depth y shrinkage de gbm. Validad siempre con los datos de validación. ¿Es posible superar lo que salía con los modelos lineales?

```
paraGBM = data.frame(sh=c(), id=c(), nt=c(), tr=c(), va=c())

arboles = c(100, 1000, 5000, 10000)

for (shrinkage in c(0.001, 0.005, 0.01, 0.05, 0.1, 0.5)){
  for (interaction.depth in seq(2,8,by=2)){
    set.seed(1)
    res=gbm(wp1 ~ ws+wd+u+v, data=entr1, n.trees=max(arboles), shrinkage=shrinkage, interaction.depth=i

    for(indice in 1:length(arboles)){
      errorEntrenamiento = errorCuadratico(predict(res, entr1, n.trees=arboles[indice]), entr1$wp1)
      errorValidacion = errorCuadratico(predict(res, val1, n.trees=arboles[indice]), val1$wp1)
      paraGBM =rbind(paraGBM,
                     data.frame(shrinkage, interaction.depth,arboles[indice], errorEntrenamiento, error
      )
    }

  }}
```

Distribution not specified, assuming gaussian … Distribution not specified, assuming gaussian …

```r
names(paraGBM) = c("shrinkage","interaction.depth","n.trees","errorTrain","errorVal")
paraGBM = paraGBM[order(paraGBM$errorVal),]
```

El mejor resultados es para los siguientes parámetros:

```r
kable(head(paraGBM,1))
```

	shrinkage	interaction.depth	n.trees	errorTrain	errorVal
22	0.005	4	1000	0.1328772	0.1598272

Los primeros mejores resultados son:

```r
kable(head(paraGBM))
```

	shrinkage	interaction.depth	n.trees	errorTrain	errorVal
22	0.005	4	1000	0.1328772	0.1598272
3	0.001	2	5000	0.1405320	0.1599720
7	0.001	4	5000	0.1327833	0.1599777
18	0.005	2	1000	0.1405587	0.1600125
4	0.001	2	10000	0.1349050	0.1601313
34	0.010	2	1000	0.1350172	0.1601375

Los resultados de todos los modelos

```r
errores = c(errores, mejorGBM=paraGBM$errorVal[1])
print(sort(errores))
```

```
##  mejorGBM        lm svmLineal        RF       svm       gbm
## 0.1598272 0.1601961 0.1631840 0.1678734 0.1728444 0.2313854
```

- 2.d) Probad a hacer lo mismo con la mejor combinación de atributos encontrada en la primera parte (se trataría de construir el modelo con 1, 2, o 3 atributos, lo que mejor funcionara en la primera parte. ¿Es mejor que lo que salía en los resultados correspondientes de la primera parte?

Como en la primera parte la mejor solucion era con los cuatro atributos, no es necesario hacer nada mas en esta parte. * 3) Tercera parte - Añadir atributos que representen los 1, 2, o 3 instantes previos de la serie

temporal * 3.a) Construid un modelo lineal y un modelo con gbm que use los datos que se venían usando hasta ahora (los de 1c) con hors==1) pero añadiendo uno, dos o más instantes de la serie temporal previa a la hora de predicción de wp. ¿Añadir estos valores mejora los resultados obtenidos hasta ahora? ¿Cuántos valores habría que añadir?

Primero cargamos los datos, donde cada uno tiene los instantes t12 hasta t1 (t12 es el instante anterior, t11 es el instante siguiente, etc)

```
windForecastsConSerie = read.csv("trainConSerie_WF1.txt")
# Lo siguiente es para poner las fechas en formato fecha lubridate
windForecastsConSerie[, c("date", "dateB")] = lapply(windForecastsConSerie[, c("date", "dateB")], ymd_h
names(windForecastsConSerie)
```

```
##  [1] "dateB"    "date"     "hors"     "u"        "v"        "ws"
##  [7] "wd"       "wp1"      "wp2"      "wp3"      "wp4"      "wp5"
## [13] "wp6"      "wp7"      "id"       "predecir" "t1"       "t2"
## [19] "t3"       "t4"       "t5"       "t6"       "t7"       "t8"
## [25] "t9"       "t10"      "t11"      "t12"
```

```
head(windForecastsConSerie)
```

```
##                 dateB                date hors    u     v   ws     wd
## 1 2009-07-01 12:00:00 2009-07-01 13:00:00    1 2.77 -0.65 2.85 103.17
## 2 2009-07-01 12:00:00 2009-07-01 14:00:00    2 3.12 -0.74 3.20 103.36
## 3 2009-07-01 12:00:00 2009-07-01 15:00:00    3 3.29 -0.62 3.35 100.63
## 4 2009-07-01 12:00:00 2009-07-01 16:00:00    4 3.31 -0.37 3.33  96.42
## 5 2009-07-01 12:00:00 2009-07-01 17:00:00    5 3.17 -0.13 3.17  92.38
## 6 2009-07-01 12:00:00 2009-07-01 18:00:00    6 2.88 -0.04 2.88  90.73
##     wp1   wp2 wp3 wp4   wp5 wp6 wp7  id predecir    t1   t2   t3    t4
## 1 0.000 0.026   0   0 0.111   0   0 -14       no 0.085 0.02 0.06 0.045
## 2 0.010 0.021   0   0 0.137   0   0 -15       no 0.085 0.02 0.06 0.045
## 3 0.000 0.016   0   0 0.106   0   0 -16       no 0.085 0.02 0.06 0.045
## 4 0.000 0.021   0   0 0.086   0   0 -17       no 0.085 0.02 0.06 0.045
## 5 0.000 0.016   0   0 0.056   0   0 -18       no 0.085 0.02 0.06 0.045
## 6 0.005 0.032   0   0 0.030   0   0 -19       no 0.085 0.02 0.06 0.045
##      t5    t6 t7 t8   t9   t10  t11  t12
## 1 0.035 0.005  0  0 0.01 0.025 0.03 0.01
## 2 0.035 0.005  0  0 0.01 0.025 0.03 0.01
## 3 0.035 0.005  0  0 0.01 0.025 0.03 0.01
## 4 0.035 0.005  0  0 0.01 0.025 0.03 0.01
## 5 0.035 0.005  0  0 0.01 0.025 0.03 0.01
## 6 0.035 0.005  0  0 0.01 0.025 0.03 0.01
```

```
lista = split(windForecastsConSerie, windForecasts$dateB)
```

```
## Warning in split.default(x = seq_len(nrow(x)), f = f, drop = drop, ...):
## data length is not a multiple of split variable
```

```
set.seed(1)
paraEntrenamiento = sample(1:length(lista), round(2/3*length(lista)))

entr = lista[paraEntrenamiento]
```

```
val =  lista[-paraEntrenamiento]

entr = do.call(rbind, entr)
val =  do.call(rbind, val)
```

```
conSerie = function(horizonte){

  erroresInstantesGBM=c()
  erroresInstantesLM=c()

  atributos = c("ws","wd", "u", "v", "wp1")
  extra = paste0("t", 12:1)

  for(cuantos in 0:3){
    rango = if(cuantos==0) 0 else 1:cuantos
    extras = extra[rango]
    entr1 = subset(entr, hors==horizonte, select=c(atributos, extras))
    val1 = subset(val, hors==horizonte, select=c(atributos, extras))

    erroresInstantesGBM[as.character(length(extras))] = with(paraGBM[1,], {
      set.seed(1)
      res=gbm(wp1 ~ ., data=entr1, n.trees=n.trees, shrinkage=shrinkage, interaction.depth=interaction.
      errorCuadratico(val1$wp1, predict(res, val1, n.trees=n.trees))})

      set.seed(1)
      res=lm(wp1 ~ ., data=entr1)
      erroresInstantesLM[as.character(length(extras))] = errorCuadratico(val1$wp1, predict(res, val1))

  }

  list(GBM=erroresInstantesGBM, LM=erroresInstantesLM)

}
```

```
resultados = conSerie(1)
```

```
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
```

Metiendo de cero a tres instantes con horizonte de 1h

```
print(sort(resultados$GBM))
```

```
##          3          2          1          0
## 0.06988108 0.07007762 0.07091879 0.16424132
```

```
print(sort(resultados$LM))
```

```
##          3          2          1          0
## 0.06844318 0.06852259 0.07033259 0.16295014
```

Se puede ver

```
resultados = conSerie(12)
```

```
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
```

Metiendo de cero a tres instantes con horizonte de 12h

```
print(sort(resultados$GBM))
```

```
##         1         2         3         0
## 0.1510310 0.1516198 0.1521589 0.1667229
```

```
print(sort(resultados$LM))
```

```
##         2         1         3         0
## 0.1527471 0.1527699 0.1539561 0.1674158
```

- 3.b) ¿Ocurre lo mismo si hacemos la predicción a 24 horas (hors==24) o a 48 horas (hors==48)?

Metiendo de cero a tres instantes y horizonte temporal a 24 horas

```
resultados = conSerie(24)
```

```
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
```

Metiendo de cero a tres instantes con horizonte de 24h

```
print(sort(resultados$GBM))
```

```
##         3         2         1         0
## 0.1557953 0.1558863 0.1566203 0.1618795
```

```
print(sort(resultados$LM))
```

```
##         3         2         1         0
## 0.1545211 0.1550408 0.1551343 0.1614594
```

```
resultados = conSerie(48)
```

```
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
## Distribution not specified, assuming gaussian ...
```

Metiendo de cero a tres instantes y horizonte temporal a 48 horas

```
print(sort(resultados$GBM))
```

```
##         1         2         3         0
## 0.1710593 0.1711494 0.1714899 0.1723101
```

```
print(sort(resultados$LM))
```

```
##         2         3         1         0
## 0.1693003 0.1694245 0.1695088 0.1714060
```

Parece que la conclusion es que los instantes anteriores proporcionan mucha influencia en horizontes temporales pequeños (1h), pero esa ventaja se va diluyendo al incrementar el horizonte temporal y para 48h prácticamente no hay mejora. El primer instante de la serie temporal parece ser el más relevante * 4) Cuarta parte - Apartado libre. Basándose en el análisis anterior, intentad mejorar los resultados o probad algo diferente. Podéis hacerme sugerencias y lo comentamos.

Como parte opcional, en lugar de construir un modelo distinto para cada horizonte temporal, vamos a construir un único modelo con todos los horizontes temporales, pero añadiendo una variable de entrada mas que sea el horizonte temporal

```
atributos = c("ws","wd", "u", "v", "wp1", "t10", "t11", "t12")
entr1 = subset(entr, select=c(atributos, "hors"))
set.seed(1)
resLM=lm(wp1 ~ ., data=entr1)

set.seed(1)
resGBM=with(paraGBM[1,], {
  gbm(wp1 ~ ., data=entr1, n.trees=n.trees, shrinkage=shrinkage, interaction.depth=interaction.depth)
})
```

```
## Distribution not specified, assuming gaussian ...
```

```
for(horizonte in c(1,12, 24, 48)){
  val1 = subset(val, hors==horizonte, select=c(atributos, "hors"))

  erroresInstantesModeloCompleto = errorCuadratico(val1$wp1, predict(resLM, val1))
  cat(paste0("LM: Error modelo completo, con el test para horizonte de ", horizonte, " es: ", erroresIns

  erroresInstantesModeloCompleto = with(paraGBM[1,], {
     errorCuadratico(val1$wp1, predict(resGBM, val1, n.trees=n.trees))
  })
  cat(paste0("GBM: Error modelo completo, con el test para horizonte de ", horizonte, " es: ", erroresIn
}
```

```
## LM: Error modelo completo, con el test para horizonte de 1 es: 0.139505820458602
## GBM: Error modelo completo, con el test para horizonte de 1 es: 0.126613398112969
## LM: Error modelo completo, con el test para horizonte de 12 es: 0.15578514067376
## GBM: Error modelo completo, con el test para horizonte de 12 es: 0.149223719268169
## LM: Error modelo completo, con el test para horizonte de 24 es: 0.154092768710202
## GBM: Error modelo completo, con el test para horizonte de 24 es: 0.153299198983858
## LM: Error modelo completo, con el test para horizonte de 48 es: 0.167836201423223
## GBM: Error modelo completo, con el test para horizonte de 48 es: 0.165388366321024
```

Parece haber algo de mejora para los horizontes grandes, pero ya no se observa la gran mejora en los horizontes pequeños (de 1h)

VAmos a aprender un modelo con los 3 últimos instantes de tiempo, otro con las variables meteorológicas y otro no lineal con svm que pondere

```
datosInstantes = subset(entr1, select=c("t12", "t11", "t10", "wp1"))
modeloInstantes = lm(wp1 ~ ., data=datosInstantes)

datosMeteo = subset(entr1, select=c(-t12, -t11, -t10, -hors))
modeloMeteo = lm(wp1 ~ ., data=datosMeteo)

datosMezcla = data.frame(instantes = predict(modeloInstantes, datosInstantes),
                         meteo = predict(modeloMeteo, datosMeteo),
                         hors=entr1$hors,
                         wp1 = entr1$wp1
                         )
datosMezclaVal = data.frame(instantes = predict(modeloInstantes, val),
                            meteo = predict(modeloMeteo, val),
                            hors=val$hors,
                            wp1 = val$wp1
)

modeloCompleto = ksvm(wp1 ~ ., datosMezcla)
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
for(horizonte in c(1,12, 24, 48)){
  datosMezclaVal1 = subset(datosMezclaVal, hors==horizonte)

  erroresInstantesModeloCompleto = errorCuadratico(datosMezclaVal1$wp1, predict(modeloCompleto, datosMez
  cat(paste0("Error modelo completo, con el test para horizonte de ", horizonte, " es: ", erroresInstan

}
```

```
## Error modelo completo, con el test para horizonte de 1 es: 0.079904144876644
## Error modelo completo, con el test para horizonte de 12 es: 0.15506975322083
## Error modelo completo, con el test para horizonte de 24 es: 0.159715175297841
## Error modelo completo, con el test para horizonte de 48 es: 0.169387837557627
```

Este modelo que sigue un esquema Stacking, se acerca bastante mas a lo que obteníamos al aprender 1 modelo para cada horizonte temporal