Proyecto Final

Adrián Sánchez Cerrillo, Miguel Ángel López Robles

2 de junio de 2018

Presentación del problema a resolver

El problema seleccionado para el proyecto es la base de datos *Parkinson Telemonitoring*. Este conjunto de datos, elaborado por la Universidad de Oxford en colaboración con centros médicos de los Estados Unidos, se compone de una gama de mediciones biométricas sobre la voz de un total de 42 pacientes en una etapa temprana de la enfermedad de Parkinson, durante un periodo de séis meses. Por cada paciente existen unas 140 mediciones, resultando en un total de 5,875 mediciones.
El conjunto de datos se compone de un total de 22 atributos, dónde podemos encontrar información sobre el paciente, mediciones biomédicas sobre la voz, y escalas UPDRS (**Unified Parkinson's Disease Rating Scale**), siendo una escala de estratificación que se utiliza para seguir el avance de la enfermedad de Parkinson. El objetivo será el de predecir mediante regresión las escalas UPDRS a partir de las mediciones biomédicas de la voz de los distintos pacientes que componen este conjunto de datos.

Carga de ficheros

Primeramente, se cargarán las librerías necesarias para el desarrollo y ejecución de la práctica y, posteriormente, se cargará el memoria la base de datos correspondiente para su posterior análisis, mediante el uso de la función *read.csv()*.

Una vez hemos cargado las librerías, cargamos en memoria la base de datos.

```
set.seed(3)
datos = read.csv("./datos/parkinsons.csv")
```

Los datos se han cargado correctamente, procedemos a realizar la separación de datos y etiquetas.

Definición de los conjuntos de datos

En la descripción de la base de datos proporcionada, se indica que no existen valores nulos, por lo tanto no necesitaremos tener en cuenta esto antes de definir los conjuntos de datos.

Ahora vamos a separar la base de datos en datos y etiquetas, debemos tener en cuenta que para el estudio de nuestro problema solo son relevantes los atributos referentes a las mediciones biométricas de la voz, y como etiquetas deberemos establecer dos tipos, ya que el problema requiere realizar la predicción de dos atributos, siendo estos *total_UPDRS* y *motor_UPDRS*. Para la construcción de nuestros modelos usaremos principalmente la librería caret y por tanto para algunas funciones nos convendrá tener los datos separados de las etiquetas, vamos a extraer las dos etiquetas en distintos conjuntos.

```
datos = datos[,-c(1,2,3,4)]
UPDRSmotor = datos[,1]
UPDRStotal = datos[,2]
datos = datos[,-c(1,2)]
```

Una vez hemos definido los datos y las etiquetas, procedemos a preprocesar los datos y definir los conjuntos de entrenamiento y test.

Preprocesado de los datos

Para realizar la validación y entrenamiento de nuestro modelo, puesto que solo disponemos de un archivo con datos para hacer test, se realizará como sigue a continuación:

*** Train :** 80% Correspondiente al conjunto de datos servirá para realizar entrenamiento de nuestro modelo, particionando dicho subconjunto y validando el entrenamiento mediante la técnica de validación cruzada.

*** Test :** 20% Restante para validar nuestro modelo ya entrenado, obteniendo así un conjunto de datos para realizar test.

Separamos las etiquetas del conjunto de datos:

Primeramente, se va a comprobar si existe algúna columna de atributos cuya varianza sea cero, dichas variables serán irrelevantes para el estudio del problema.

```
zero <- nearZeroVar(datos)
zero
```

```
## integer(0)
```

Como podemos observar, no existe ningún atributo cuya varianza sea cero.
Ahora generamos aleatoriamente el conjunto de entrenamiento y de test, manteniendo la distribución mencionada anteriormente (80/20%).

```
indices.train = sample(nrow(datos),size = nrow(datos)*0.8)
datos.train = datos[indices.train,]
datos.test = datos[-indices.train,]

UPDRSmotor.train = UPDRSmotor[indices.train]
UPDRSmotor.test = UPDRSmotor[-indices.train]

UPDRStotal.train = UPDRStotal[indices.train]
UPDRStotal.test = UPDRStotal[-indices.train]
```

Una vez hemos obtenido los conjuntos de entrenamiento y prueba, vamos a avanzar en el estudio de la varianza y de la muestra. Vamos a preprocesar nuestro conjunto de entrenamiento mediante el uso de la función **preProcess()**, mediante la aplicación de diferentes métodos para ajustar los valores de nuestros atributos. Usaremos *center* y *scale* de tal forma que extraeremos la media y dividiremos por la desviación estándar.
Para establecer una comparación, usaremos el preprocesamiento con y sin la aplicación de *PCA*, "Principal Component Analysis", el cual es un procedimiento estadístico que usa transformaciones ortogonales para reducir la dimensión del conjunto eliminando, si procede, aquellos atributos que considera correlacionados con otros, aunque solo sea ligeramente, y que por tanto, no considera relevantes para el desarrollo del modelo de aprendizaje.

```
ProcesamientoPCA <- preProcess(datos.train, method = c("center", "scale", "pca"),thres=0.95)
ProcesamientoPCA
```

```
## Created from 4700 samples and 16 variables
##
## Pre-processing:
##   - centered (16)
##   - ignored (0)
##   - principal component signal extraction (16)
##   - scaled (16)
##
## PCA needed 5 components to capture 95 percent of the variance
```

El preprocesamiento con PCA nos ha reducido el modelo a 5 variables, por lo tanto estaría descartando once de los atributos conjunto de datos.
Como podemos observar, PCA debería usarse cuando la dimensión del conjunto de datos es muy alta, no es completamente necesario reducir el número de atributos que disponemos, teniendo en cuenta además, que solo disponemos de 16 atributos. *PCA* trabaja creando combinaciones lineales de las variables y es generalmente usado cuando el número de atributos es suficientemente grande tal y como se ha mencionado con anterioridad. Puesto que solo disponemos de 16 atributos, hemos optado por no aplicarlo en el preprocesado.

```
ProcesamientoSinPCA <- preProcess(datos.train, method = c("center", "scale"))
ProcesamientoSinPCA
```

```
## Created from 4700 samples and 16 variables
##
## Pre-processing:
##   - centered (16)
##   - ignored (0)
##   - scaled (16)
```

Obtenemos el nuevo conjunto de datos tras haber aplicado el preprocesamiento anterior con la función **predict()**.
Actualizamos los conjuntos de datos:

```
datos.train = predict(ProcesamientoSinPCA,datos.train)
datos.test = predict(ProcesamientoSinPCA,datos.test)
```

Métrica a utilizar R-squared

Para validar y seleccionar un modelo final vamos a establecer comparaciones en torno al coeficiente de determinación R^2, el cual nos indica el porcentaje de varianza que nuestro modelo es capaz de explicar con respecto a los datos, por lo tanto, a la hora de determinar la calidad del modelo y predecir nuevos datos, este coeficiente debe ser lo más alto posible para garantizar que estamos ante buen modelo. También hay que tener en cuenta la posiblidad de sobreajuste en esta medida, ya que un coeficiente de determinación R^2 muy cercano a 1 también puede determinar que hemos ajustado nuestro modelo demasiado a los datos.
El aspecto positivo de esta métrica es que se encuentra normalizada, los valores posibles oscilarán entre cero y uno, a diferencia de la métrica por defecto RMSE, *"Root Mean Squared Error"*, que dependerá de la escala en la que se encuentren los datos y no nos proporcionará en una primera instancia, tanta información.

Modelos Lineales

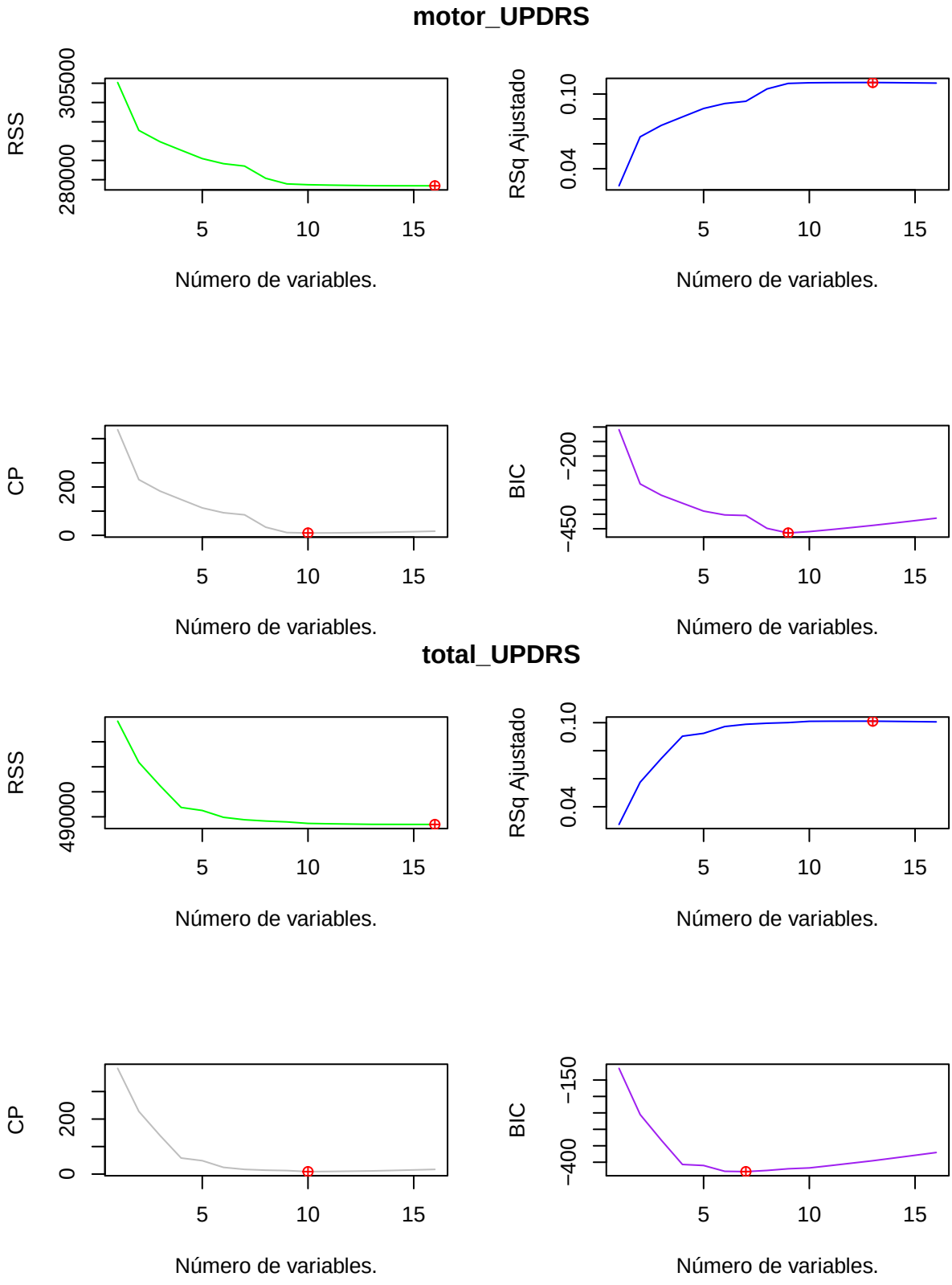
Para proceder con los modelos lineales, y el entrenamiento del modelo, seguiremos el procedimiento SRM, *"Structural Risk Minimization"*, empezaremos por la clase de funciones más simple, analizando los resultados que obtenemos, así como la métrica R^2, e incrementaremos la clase de funciones en caso de ser necesario.
Antes de comenzar con el procedimiento vamos a abordar el problema de elegir un subconjunto de los datos que resulte ser un buen predictor de las etiquetas a predecir, aunque el número de atributos que disponemos es relativamente bajo, vamos a analizar los resultados obtenidos por **regsubset()**, la cual realiza un análisis sobre como varían distintas métricas con el uso de los distintos atributos.

```
subsetMotor = regsubsets(x = datos.train, y = UPDRSmotor.train,
                         nvmax = ncol(datos.train), method = "forward")

subsetTotal = regsubsets(x = datos.train, y = UPDRStotal.train,
                         nvmax = ncol(datos.train), method = "forward")

summaryMotor = summary(subsetMotor)
summaryTotal = summary(subsetTotal)
```

Una vez se han realizado los cálculos pertinentes, se procederá a comparar y representar los resultados obtenidos :



Como se puede observar, para las dos variables que queremos predecir, tanto *motor_UPDRS* como *total_UPDRS*, se obtiene un error en la suma de error residual cuadrática (RSS) menor cuando utilizamos la totalidad de atributos que disponemos, de la misma manera, analizando la métrica R^2 ajustada vemos que no existe un gran cambio con el uso de 10 o 15 variables, por lo tanto consideramos que las dierencias para estas métricas no son significativas como para tener que usar un número de características menor, aludiendo además a lo mencionado en otros apartados, puesto que disponemos de un número de variables reducido.
Si observamos la métricas CP de *Mallow's*, es un coeficiente que queremos minimizar, puesto que un valor pequeño de esta métrica indica que estamos ante un modelo más preciso, pretende lograr un equilibrio en el número de predictores a incluir en el modelo, comparando la precisión y el sesgo con submodelos formados por combinaciones de las distintas variables. Como vemos, no existe diferencia significativa entre el mínimo encontrado para ambas etiquetas a predecir (10 variables), por lo tanto si analizamos esta métrica también podríamos llegar a la conclusión de utilizar el número total de atributos disponibles.
Por último nos encontramos con la métrica BIC, *"Bayesian Information Criterion"*, la cual es un criterio de selección de modelos muy restrictivo, introduciendo un término de penalización mayor que CP, tendiendo a elegir modelos con un menor número de predictores; por lo tanto y puesto que no varía demasiado, procederemos a realizar el análisis de nuestro modelo con todos los atributos.

Modelo lineal simple
Establecemos el método de control para validar el modelo con la función **trainControl()**, indicando la aplicación de validación cruzada con cinco particiones como método de validación.

```
control = trainControl(method = "cv", number = 5, allowParallel=TRUE)
```

Ahora usamos la función train para entrenar nuestro modelo e intentar predecir las etiquetas, establecemos el método de control al propuesto anteriormente e indicamos que la métrica que queremos tener en cuenta es R^2.

```
LinealSimpleMotor <- train(x = datos.train, y = UPDRSmotor.train, method = "lm", trControl = control, me
LinealSimpleTotal <- train(x = datos.train, y = UPDRStotal.train, method = "lm", trControl = control, me

LinealSimpleMotor$results
```

```
##   intercept     RMSE  Rsquared     MAE    RMSESD RsquaredSD     MAESD
## 1      TRUE 7.728039 0.1067338 6.53988 0.1856112 0.01757508 0.1725204
```

```
LinealSimpleTotal$results
```

```
##   intercept     RMSE   Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1      TRUE 10.21105 0.09999643 8.370195 0.2393763 0.02162909 0.1615522
```

Si analizamos los resultados podemos ver que el modelo generado no es muy preciso, puesto que la métrica R^2 nos indica que nuestro modelo es capaz de explicar en torno al 10% de la varianza de los datos que queremos predecir, lo cual es bastante bajo, por lo tanto vamos a proceder aumentando la clase de funciones a otras más complejas.

Aumento de la clase de funciones
Procederemos realizando un aumento de la clase de funciones de forma polinomial, comenzando con las cuadráticas hasta la obtención de un polinomio de quinto grado.
Primeramente cabe mencionar que entrenaremos los modelos lineales mediante el uso de la regresión *Ridge* y *Lasso*, ambas permiten regularización por hiperparámetros. Por un lado, la regresión *Ridge* incorpora un parámetro que pretende reducir los predictores a valores cercanos a cero, el efecto es provocar una reducción de la varianza del modelo, sin descartar en el proceso ningún atributo predictor del modelo; por otro lado, *Lasso* difiere en esto último, incluyendo la capacidad de excluir predictores del modelo final. El método *Lasso* al igual que la regresión *Ridge*, fuerza a que las estimaciones de los coefientes predictores tiendan a cero, consiguiendo que algunos de ellos sean cero directamente lo que, además de reducir la varianza, realiza selección de precitores.

UPDRS Motor

```r
train = datos.train

for (i in 2:5){

  train = cbind(train,I(datos.train)^i)

  fitLasso <- train(x = train, y = UPDRSmotor.train,
                    method = "lasso",
                    trControl = control,
                    metric = "Rsquared"
  )
  fitRidge<- train(x = train, y = UPDRSmotor.train,
                    method = "ridge",
                    trControl = control,
                    metric = "Rsquared"
  )

  cat("Polinomio Grado ", i, "para Lasso", "\n")
  print(fitLasso$results)


  cat("Polinomio Grado ", i,"para Ridge", "\n")
  print(fitRidge$results)
  cat("\n")
}
```

```
## Polinomio Grado  2 para Lasso
##   fraction     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1      0.1 7.608134 0.1342647 6.400262 0.1298393 0.01800405 0.1410560
## 2      0.5 7.609941 0.1339136 6.401004 0.1297928 0.01789078 0.1407517
## 3      0.9 7.612481 0.1333859 6.402292 0.1296222 0.01767906 0.1404082
## Polinomio Grado  2 para Ridge
##   lambda     RMSE  Rsquared      MAE     RMSESD RsquaredSD      MAESD
## 1  0e+00 7.617622 0.1333433 6.406523 0.11591473 0.02035752 0.08140622
## 2  1e-04 7.613598 0.1342871 6.404208 0.11962272 0.02103054 0.08192355
## 3  1e-01 7.691528 0.1153296 6.523120 0.08604809 0.01577471 0.06648141
##
## Polinomio Grado  3 para Lasso
##   fraction     RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1      0.1 7.535487 0.1501846 6.331813 0.1252056 0.01634882 0.09810032
## 2      0.5 7.586713 0.1418672 6.343440 0.1458476 0.02341274 0.08527775
## 3      0.9 7.589755 0.1413808 6.344507 0.1486119 0.02377898 0.08667823
## Polinomio Grado  3 para Ridge
##   lambda     RMSE  Rsquared      MAE     RMSESD RsquaredSD      MAESD
## 1  0e+00 7.567813 0.1455361 6.347577 0.10163798 0.02524812 0.10577319
## 2  1e-04 7.574973 0.1446251 6.348466 0.10463152 0.02601095 0.10923247
## 3  1e-01 7.647075 0.1269852 6.488031 0.09295583 0.02988936 0.08606519
##
## Polinomio Grado  4 para Lasso
##   fraction      RMSE   Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1      0.1 10.208795 0.09788192 6.455896 5.3596451 0.06346723 0.23713406
## 2      0.5  8.988792 0.09675002 6.415527 2.5714830 0.06107467 0.14418899
## 3      0.9  7.794112 0.12254990 6.349318 0.2896642 0.03160088 0.07619552
## Polinomio Grado  4 para Ridge
```

```
##   lambda     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1  0e+00 8.335522 0.1095008 6.405696 1.0770788 0.05599733 0.1347624
## 2  1e-04 7.934163 0.1195736 6.359109 0.6288575 0.04583957 0.1019250
## 3  1e-01 7.656694 0.1227323 6.492951 0.1609284 0.01935060 0.1299266
##
## Polinomio Grado  5 para Lasso
##   fraction     RMSE   Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1      0.1 17.50361 0.09594746 6.695035 20.18330 0.08152023 0.7905706
## 2      0.5 16.80434 0.09524198 6.676886 18.67154 0.08046786 0.7393399
## 3      0.9 17.64270 0.09239297 6.710719 20.65248 0.07766481 0.7990700
## Polinomio Grado  5 para Ridge
##   lambda     RMSE  Rsquared      MAE      RMSESD RsquaredSD      MAESD
## 1  0e+00 61.07117 0.1133068 8.823198 119.4028678 0.06952911 5.65913476
## 2  1e-04 16.97979 0.1319683 6.746070  21.2695141 0.07298753 1.11933037
## 3  1e-01  7.66423 0.1208410 6.493767   0.1026085 0.01294062 0.04782836
```

Como vemos no hemos obtenido una mejora significativa a la hora de predecir la etiqueta *motor_UPDRS* a pesar de haber incluido polinomios de grado quinto, aunque parece ser que los modelos lineales no van a ser la solución, vamos a seguir el mismo procedimiento con la etiqueta *total_UPDRS*.

UPDRS Total

```
train = datos.train

for (i in 2:5){

  train = cbind(train,I(datos.train)^i)

  fitLasso <- train(x = train, y = UPDRStotal.train,
                    method = "lasso",
                    trControl = control,
                    metric = "Rsquared"
  )
  fitRidge<- train(x = train, y = UPDRStotal.train,
                    method = "ridge",
                    trControl = control,
                    metric = "Rsquared"
  )

  cat("Polinomio Grado ", i, "para Lasso", "\n")
  print(fitLasso$results)


  cat("Polinomio Grado ", i,"para Ridge", "\n")
  print(fitRidge$results)
  cat("\n")
}
```

```
## Polinomio Grado  2 para Lasso
##   fraction     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1      0.1 10.07065 0.1241653 8.175950 0.2071286 0.02212098 0.1685306
## 2      0.5 10.07819 0.1230987 8.179229 0.2102833 0.02236264 0.1686444
## 3      0.9 10.08293 0.1224694 8.179551 0.2136287 0.02258209 0.1683603
## Polinomio Grado  2 para Ridge
##   lambda     RMSE  Rsquared      MAE     RMSESD RsquaredSD     MAESD
## 1  0e+00 10.08593 0.1214715 8.182190 0.09255263 0.01732277 0.1234855
```

```
## 2  1e-04 10.08011 0.1222929 8.180746 0.09773602 0.01838217 0.1225891
## 3  1e-01 10.10191 0.1178958 8.230724 0.13917200 0.01817229 0.1257402
##
## Polinomio Grado  3 para Lasso
##   fraction     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1      0.1 10.09282 0.1316670 8.121971 0.2767436 0.03372792 0.1667299
## 2      0.5 10.01497 0.1368553 8.113352 0.1385951 0.02633143 0.1527873
## 3      0.9 10.00653 0.1376920 8.112889 0.1299129 0.02556364 0.1518160
## Polinomio Grado  3 para Ridge
##   lambda      RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1  0e+00  9.990294 0.1423710 8.090110 0.2203632 0.03254354 0.1845547
## 2  1e-04 10.051502 0.1390247 8.097696 0.3304379 0.03618397 0.1800650
## 3  1e-01 10.032097 0.1312343 8.184363 0.1526180 0.02829136 0.1531400
##
## Polinomio Grado  4 para Lasso
##   fraction     RMSE  Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1      0.1 11.12522 0.1282098 8.135588 2.795623 0.06913859 0.3508467
## 2      0.5 11.11553 0.1286896 8.139924 2.784702 0.06940500 0.3560783
## 3      0.9 10.65756 0.1320470 8.118033 1.783611 0.06494240 0.3107401
## Polinomio Grado  4 para Ridge
##   lambda     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1  0e+00 10.34056 0.1257459 8.118092 0.7139177 0.06153411 0.2408664
## 2  1e-04 10.28364 0.1299456 8.082592 0.6270057 0.05632121 0.2177796
## 3  1e-01 10.03533 0.1309903 8.197943 0.1425937 0.02602575 0.1530422
##
## Polinomio Grado  5 para Lasso
##   fraction     RMSE   Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1      0.1 27.40313 0.06960670 8.711206 34.94877 0.06914980 1.2883000
## 2      0.5 23.92054 0.06845303 8.598370 27.33169 0.06747275 1.0383641
## 3      0.9 17.50523 0.06753330 8.388203 12.73432 0.06673384 0.5781541
## Polinomio Grado  5 para Ridge
##   lambda      RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1  0e+00 12.347146 0.1120591 8.240501 4.8525630 0.07342852 0.5468361
## 2  1e-04  9.928261 0.1521634 8.003886 0.2117116 0.03486722 0.1542307
## 3  1e-01 10.040930 0.1286242 8.197062 0.1041761 0.01884632 0.1085549
```

Se confirman las hipótesis realizadas anteriormente, puesto que no hemos conseguido mejorar vamos a intentar usar modelos no lineales con la intención de obtener un mejor ajuste para nuestro modelo.

Modelos No Lineales

Random Forest

Para llevar a cabo el estudio del problema mediante *Random Forest*, hemos realizado diversas pruebas para definir la configuración del algoritmo; primeramente cabe destacar la elección del criterio de selección de predictores, tras varias pruebas analizando los resultados entre $m = p$, $m = p/2$ y $m = sqrt(p)$, hemos optado por esta última puesto que era la que mejores resultados nos proporcionaba para este caso.
Con el objetivo de que la ejecución de la práctica sea más eficiente no vamos a incluir la parte de las pruebas en este fichero, externamente se ha experimentado en el ajuste de hiperparámetros en amplios rangos de valores y hemos optado por usar un número total de 500 árboles. La elección de un número mayor de árboles reporta una mejora, aunque no muy significativa; además también hemos tenido en cuenta la eficiencia computacional, por lo que hemos optado por dicha cantidad.

```
fitrfMotor<- train(x = datos.train, y = UPDRSmotor.train,
                   method = "rf",
```

```
                  trControl = control,
                  tuneGrid = expand.grid(.mtry=sqrt(ncol(datos.train)) ),
                  ntree = c(500),
                  metric = "Rsquared"
)
fitrfTotal<- train(x = datos.train, y = UPDRStotal.train,
                   method = "rf",
                   trControl = control,
                   tuneGrid = expand.grid(.mtry=sqrt(ncol(datos.train)) ),
                   ntree = c(500),
                   metric = "Rsquared"
)

fitrfMotor$result
```

```
##   mtry     RMSE  Rsquared      MAE    RMSESD RsquaredSD      MAESD
## 1    4 6.643837 0.3623765 5.403504 0.1040251 0.02664209 0.09233488
```

```
fitrfTotal$result
```

```
##   mtry     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
## 1    4 8.654638 0.3738907 6.835176 0.1358851 0.02767221 0.1329472
```

Como se puede observar, hemos obtenido mejora con respecto a los modelos lineales, sin embargo, estamos todavía lejos de obtener un modelo de calidad que sea un buen predictor de las etiquetas. La métrica R^2 no nos aporta un buen valor con el que podamos asegurarnos que nuestro predictor es de calidad.

Supported Vector Machine

Al igual que con el método anterior, en SVM se han realizado diversas pruebas con un amplio rango de valores para obtener parámetros adecuados con los que realizar la ejecución, por lo que hemos obtenido que la mejor configuración es la siguiente, usando el kernel RBF-Gaussiano o radial, siendo este kernel el compuesto por todas las funciones que satisfacen $\Phi(r) = \Phi(||x||)$, conocidas comúnmente como funciones radiales.

```
fitsvmRMotor = train(x = datos.train, y = UPDRSmotor.train,
                method = "svmRadial",
                trControl = control,
                tuneGrid = expand.grid(.sigma = 0.41, .C = 4),
                metric = "Rsquared"

)

fitsvmRTotal = train(x = datos.train, y = UPDRStotal.train,
                 method = "svmRadial",
                 trControl = control,
                 tuneGrid = expand.grid(.sigma = 0.34, .C = 4.5),
                 metric = "Rsquared"

)

fitsvmRMotor$result
```

```
##   sigma C     RMSE  Rsquared     MAE    RMSESD RsquaredSD     MAESD
## 1  0.41 4 6.429534 0.3872507 4.88109 0.1722378 0.02075803 0.1402946
```

```
fitsvmRTotal$result
```

```
##   sigma   C     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
```

```
## 1  0.34 4.5 8.301951 0.4077979 6.260943 0.2815451 0.04310667 0.1949825
```

Como vemos, se ha obtenido una mejora, con respecto a los métodos lineales y Random Forest, aunque poco significativa con respecto a estos últimos, sin embargo estos detalles se analizarán como conclusión final.

Boosting

Por último vamos a analizar y probar el método de Boosting, el cual nos ha reportado los mejores resultados utilizando para ello una configuración formada por 51 árboles de profundidad 8, para ello hemos analizado parcialmente las posibles combinaciones entre estos dos hiperparámetros.

```
fitboostMotor <- train(x = datos.train, y = UPDRSmotor.train,
                   method = "blackboost",
                   trControl = control,
                   metric = "Rsquared",
                   tuneGrid = expand.grid(.mstop = 51, .maxdepth = 8)

)

fitboostTotal <- train(x = datos.train, y = UPDRStotal.train,
                       method = "blackboost",
                       trControl = control,
                       metric = "Rsquared",
                       tuneGrid = expand.grid(.mstop = 55, .maxdepth = 9)

)

fitboostMotor
```

```
## Boosted Tree
##
## 4700 samples
##   16 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3761, 3761, 3759, 3760, 3759
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.345939  0.2150943  6.167098
##
## Tuning parameter 'mstop' was held constant at a value of 51
##
## Tuning parameter 'maxdepth' was held constant at a value of 8
```

```
fitboostTotal
```

```
## Boosted Tree
##
## 4700 samples
##   16 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3760, 3760, 3762, 3760, 3758
## Resampling results:
##
```

```
##   RMSE      Rsquared   MAE
##   9.618679  0.2261413  7.799113
##
## Tuning parameter 'mstop' was held constant at a value of 55
##
## Tuning parameter 'maxdepth' was held constant at a value of 9
```

Vemos que no hemos mejorado con respecto a SVM, por lo tanto vamos a analizar a continuación el error obtenido fuera de la muestra.

Selección del modelo final

Ahora vamos a probar los 3 modelos no lineales fuera de la muesra, en el set que habiamos reservado para realizar la prueba.

```
yhat = predict(fitrfMotor,datos.test)
randomforest = postResample(yhat,UPDRSmotor.test)

yhat = predict(fitboostMotor,datos.test)
boosting = postResample(yhat,UPDRSmotor.test)

yhat = predict(fitsvmRMotor,datos.test)
svmR = postResample(yhat,UPDRSmotor.test)

Emotor = rbind(randomforest,boosting,svmR)
#resultados finales para UPDRStotal
yhat = predict(fitrfTotal,datos.test)
randomforest = postResample(yhat,UPDRStotal.test)

yhat = predict(fitboostTotal,datos.test)
boosting = postResample(yhat,UPDRStotal.test)

yhat = predict(fitsvmRTotal,datos.test)
svmR = postResample(yhat,UPDRStotal.test)

Etotal = rbind(randomforest,boosting,svmR)
print(Emotor)
```

```
##                  RMSE  Rsquared      MAE
## randomforest 6.496855 0.3535424 5.232359
## boosting     7.069347 0.2433868 5.873745
## svmR         6.325670 0.3786563 4.859415
```

```
print(Etotal)
```

```
##                  RMSE  Rsquared      MAE
## randomforest 8.446654 0.3690550 6.580315
## boosting     9.258921 0.2534054 7.423456
## svmR         8.127070 0.4018044 6.155997
```

Como se puede observar, los resultados que hemos obtenido finalmente no son realmente buenos, a pesar de haber analizado distintos casos mediante el uso de amplios rangos de valores para los distinos métodos no lineales, no hemos conseguido encontrar un modelo que realmente explique nuestros datos.
En nuestro caso, de tener que seleccionar un modelo de los obtenidos sería el SVM con nucleo RBF-Gaussiano ya que es el que nos proporciona mejores resultados ,en cuanto a la métrica R^2 se refiere, de los métodos no lineales y muy por encima de los lineales. Tal y como se impartió en clase de teoría, nuestro objetivo

será minimizar el error fuera de la muestra, pero una forma de conseguirlo y poder garantizarlo en cierta medida será también buscando un modelo que nos proporcione un buen error dentro de la muestra, tal y como dicta la desigualdad de Hoeffding respecto al tamaño de la muestra y con una probabilidad dada $1 - \delta$: $E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{1}{2N} log \frac{2}{\delta}}$, teniendo cuidado con el sobreajuste. En nuestro caso podemos asegurar que no se ha producido sobreajuste, ya que no hemos alcanzado un valor cercano a 1 teniendo en cuenta la métrica R^2. Por esto seleccionamos este modelo como solución a nuestro problema.

Conclusiones finales

Los resultados obtenidos, no han sido los esperados, especialmente en términos de error. En primer lugar, pensamos que esto podría deberse al propio problema en sí, ya que se trata de calcular la escala evaluación de la enfermedad de Parkinsons que posee un paciente a través de mediciones biométricas de su voz. Realmente nos pareció algo difícil de conseguir, y realmente podría ser que no existiera correlación entre nuestros datos y el objetivo.

A medida que trabajamos con el conjunto de datos nos surgió la idea de que el verdadero problema podría ser el data set y sus limitaciones. ¿Podría deberse la imposibilidad de obtener un modelo adecuado debido a ruido en la medición de los datos? Nuestro optimismo nos hacía pensar que siempre debemos partir sobre la hipótesis de la existencia ruido en la muestra, sin embargo, si se estaba intentado llevar a cabo un estudio válido, se deberían usar medidores de calidad que minimizaran el ruido lo máximo posible.

Puesto que considerábamos que nuestra forma de proceder se había realizado correctamente, no veíamos claro de donde podían venir estos problemas. A pesar de que el objetivo de esta práctica no es el de ajustar lo mejor posible el modelo ni realizar investigaciones, nos hemos basado en un estudio realizado por el departamento de Administración de Ciencia e Ingeniería de la Universidad de Stanford *[2]* para resolver nuestras dudas.
En este estudio nos encontramos con resultados similares a los obtenidos. Los investigadores de la Universidad de Stanford revelaron que tras una gran investicación sobre los datos encontraron casos muy llamativos, tales como inconstencias en los datos para medidas de un mismo paciente, con grandes cambios en la escala UPDRS, aludiendo a la dificultad de extraer un modelo preciso y con grandes errores en medición. Encontraron la causa del error tras ser notificados por la empresa que realizó el muestreo de los pacientes, de la baja calidad con la que se recolectó el audio y el dispositivo inadecuado que se utilizó para ello, limitando por tanto, las mediciones biométricas como predictoras de la escala.

En definitiva se confirmaba nuestra sospecha de que los datos no son confiables o no son consistentes y por tanto influyen negativamente en la construcción de modelos.
Como conclusión final podemos decir que este conjunto de datos tiene diversas limitaciones y sus mediciones son de baja calidad por lo que los modelos que se podían construir a partir de él no son demasiado buenos. Este proyecto nos muestra como no siempre se llegan a buenos resultados y uno de los factores que influyen en esto son los datos que poseemos. No solo importa el tamaño de la muestra, si no que además, ésta sea de calidad y que posea un ruido lo más bajo posible.

Referencias

[1] A Tsanas, MA Little, PE McSharry, LO Ramig (2009)
'Accurate telemonitoring of Parkinson's disease progression by non-invasive speech tests',
IEEE Transactions on Biomedical Engineering.

[2] Genain, N., Huberth, M., Vidyashankar, R. Predicting Parkinson's Disease Severity from Patient Voice Features. *University of Stanford*. Recuperado de http://roshanvid.com/stuff/parkinsons.pdf