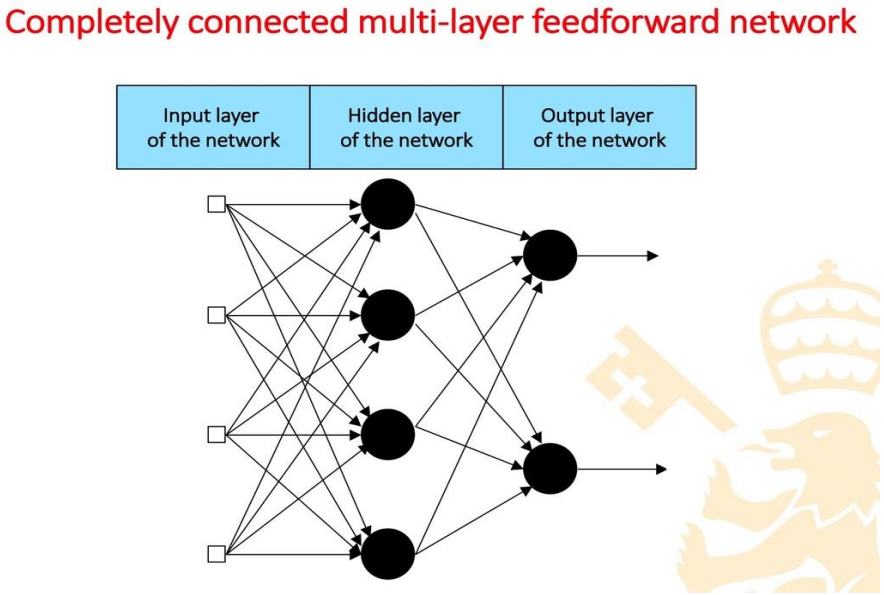
PRÁCTICA 4 MACHINE LEARNING

Jorge Candia & Alejandro Valbuena

1.- Aplicar algoritmo perceptrón multicapa al conjunto de datos usados en práctica de regresión lineal múltiple, analizar los resultados obtenidos, así como los parámetros usados. Comparar dichos resultados con los obtenidos en las prácticas 2 y 4.

El algoritmo de perceptrón multicapa (MLP) consiste en una red de perceptrones con n capas completamente conectadas unas a otras, como bien muestra la imagen:



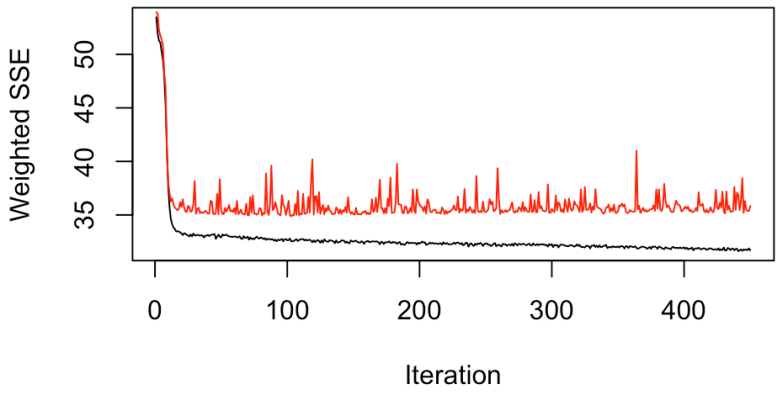
A medida que se van pasando ejemplos por la red, los pesos de las neuronas se actualizan mediante el algoritmo de backpropagation, lo que permite que al cabo de iterar numerosas veces el dataset, el modelo aprenda los patrones de este, incluso si son no lineales.

Se adjunta al final del documento el código empleado para elaborar una red MLP con el dataset de los informes 2 y 4, para su posterior análisis individual y comparativa con estas dos prácticas. El código ha sido comentado detalladamente para que sea más comprensible. A continuación, se procede a comentar los resultados de este.

Análisis del modelo:

El modelo elegido contiene 3 capas ocultas de 9, 5 y 3 neuronas. Esta elección ha sido bastante arbitraria ya que haciendo pruebas, al quitar o poner algún layer más y al aumentar o disminuir el número de neuronas de estos, el desempeño del modelo es aproximadamente el mismo, no cambia más de un 1% la R2.

Por otra parte, el modelo converge con muy pocas iteraciones ya que el error de test deja de bajar después de aproximadamente (a ojo en el plot) 25 epochs.



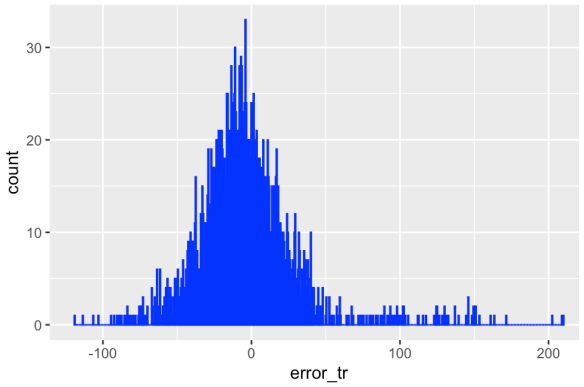
Análisis de resultados:

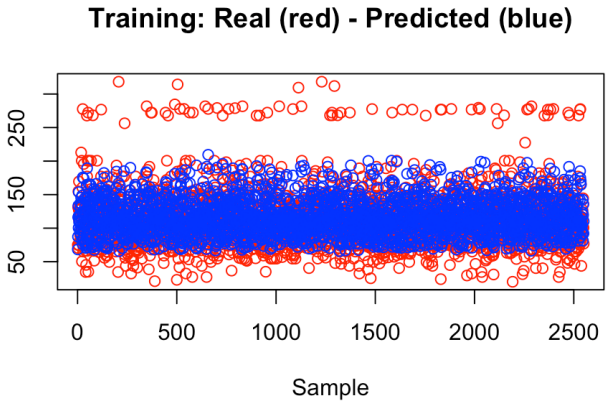
Los resultados del modelo tanto en training como en test son muy buenos: la R2 ajustada y el MSE en training son de 0.9215 y 1106 respectivamente, y de 0.9128 y 1250 en test, lo que además indica la ausencia de overfitting. Cabe remarcar, aunque se comentará en el análisis de los errores, la aparición de relativamente bastantes outliers, por lo que estos resultados podrían mejorarse si estos se eliminaran del dataset.

Análisis de los residuos:

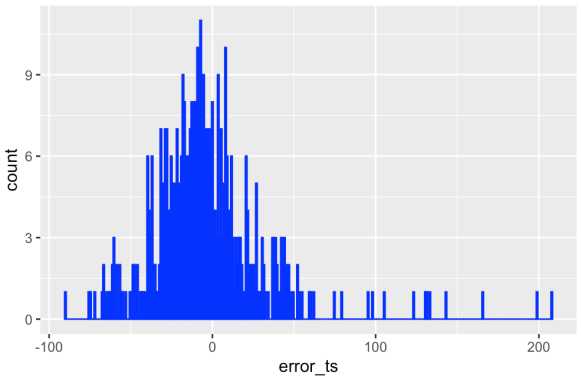
Adjuntamos las siguientes gráficas para ilustrar la distribución de los residuos tanto en training como en test. La de la izquierda es un histograma de los errores y la de la derecha una comparativa del valor real con el predicho. Se puede observar que las predicciones de training son bastante similares a las de test y los outliers comentados en el punto anterior.

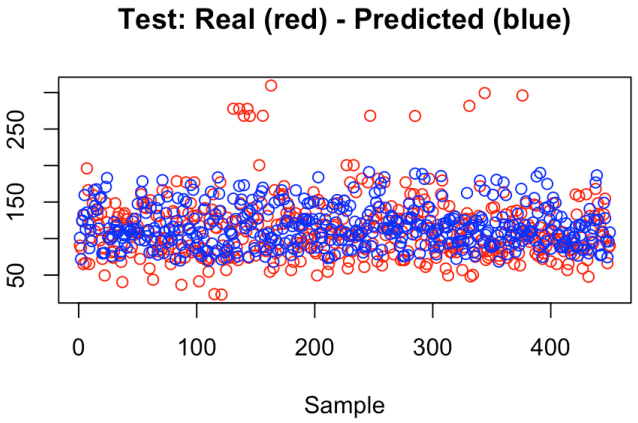
TRAINING:





TEST:





2.- Aplicar un algoritmo SVM de regresión al conjunto de datos usados en la práctica de regresión lineal múltiple, analizar los resultados obtenidos, así como los parámetros usados. Comparar dichos resultados con los obtenidos en las prácticas 2 y 4, así como con los del perceptrón multicapa.

Una Máquina de Soporte Vectorial es un algoritmo de machine learning que se basa en la selección de los llamados vectores soporte con el fin de maximizar los márgenes de unos separadores dados, lo cual se consigue a través de cálculos muy complicados de optimización matemática.

Dado que confiar en que se cree un perímetro que separe completamente dos clases puede causar overfitting en la gran mayoría de los casos reales, se introduce el concepto de *Soft Margin,* por el cual está permitido que se cuelen algunos errores dentro de nuestro margen en función del hiperparámetro *cost (C)*, que está dentro de la función a optimizar y se podría considerar como el parámetro que regula el nivel de overfitting (cuanto mayor C → menor margen → menos vectores soporte).

Dependiendo de la naturaleza del problema, la creación de la SVM se aborda de una manera u otra. Es decir, la función a optimizar que da como resultado los vectores soporte cambia en función de si nuestras variables se relacionan de forma lineal, polinómica potencial, sigmoidal, etc, ya que dentro de esta función se encuentra el llamado *Kernel*, que elige el analista según crea que es lo más adecuado.

El problema de forma generalizada es el siguiente:

Find $\alpha_1 \ldots \alpha_N$ such that
$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(x_i, x_j)$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

Cuya solución es:

$$f(x) = \Sigma\alpha_i y_i K(x_i, x_j) + b$$

Donde xi es un vector soporte. Ejemplos de *Kernel* son los siguientes:

- Linear: $K(\mathbf{x_i},\mathbf{x_j})= \mathbf{x_i}^T\mathbf{x_j}$
- Polynomial of power *p*: $K(\mathbf{x_i},\mathbf{x_j})= (1+ \mathbf{x_i}^T\mathbf{x_j})^p$
- Gaussian (radial-basis function network): $K(\mathbf{x_i},\mathbf{x_j}) = \exp(-\frac{\|\mathbf{x_i}-\mathbf{x_j}\|^2}{2\sigma^2})$
- Sigmoid: $K(\mathbf{x_i},\mathbf{x_j})= \tanh(\beta_0\mathbf{x_i}^T\mathbf{x_j} + \beta_1)$

La SVM localiza un plano separador en el *feature space* y clasifica los puntos en ese espacio. La función kernel juega el rol de lo que sería el producto escalar en el *feature space.*

Una vez resumidas las bases de las SVMs, vamos a explicar cómo se puede trasladar a problemas de predicción, que es lo que trataremos de aplicar en este estudio. Las llamadas *Support Vector Regression* (SVR) usan los mismos principios de los márgenes máximos de las SVMs con unas pequeñas diferencias.

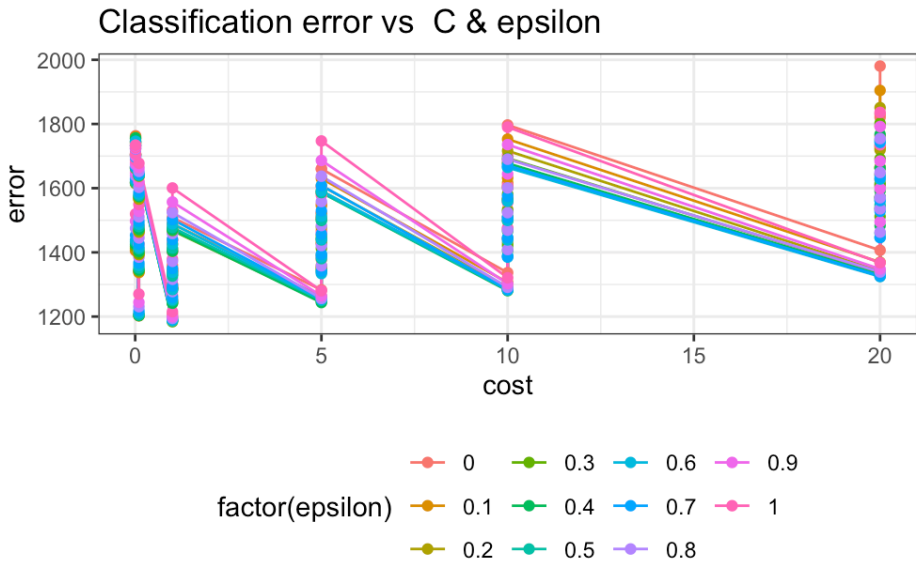
Lo primero es que la motivación del margen de tolerancia (ahora llamado épsilon en vez de coste) pasa a ser mayoritariamente relajar la complejidad del algoritmo, que ahora tiene una salida de infinitos posibles valores. Sin embargo, la idea es la misma: minimizar el error e individualizar el hiperplano que maximiza el margen, teniendo en cuenta que una parte del error es tolerado.

Las SVRs se basan en la optimización de la siguiente función de pérdida, la cual ignora el error si este es menor que el valor que se le ha otorgado al hiperparámetro épsilon:

$$||y - f(x)||_\varepsilon = \max\{0, ||y - f(x)|| - \varepsilon\}\}$$

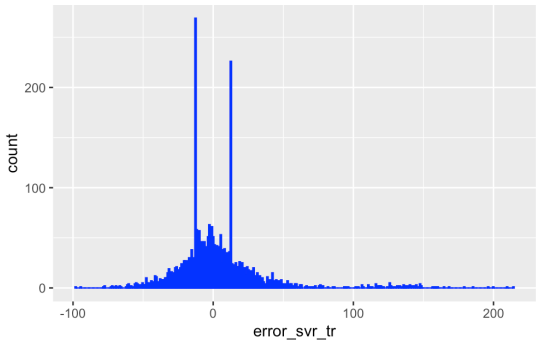
A continuación se procederá a comentar los resultados obtenidos aplicando este algoritmo al problema propuesto. De nuevo, el código está comentado con explicaciones paso a paso sobre cómo aplicar todas estas explicaciones en R.

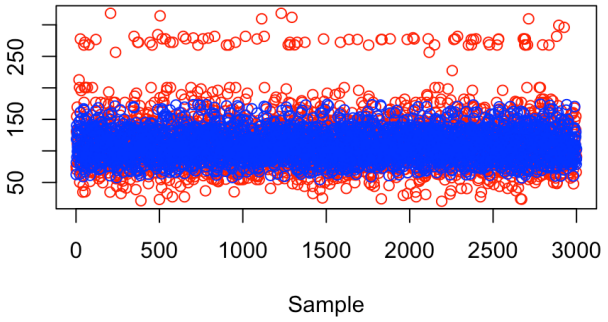
Análisis del modelo: Se hace un bucle para elegir el mejor modelo entre varios valores para los hiperparámetros *epsilon, cost* y *gamma.* Finalmente, los elegidos para el mejor modelo son *epsilon = 0.3, cost = 1* y *gamma = 0.1.* El número de vectores soporte del modelo es de 590 y el kernel escogido fue el radial. El error VS coste de cada modelo es el siguiente:



Análisis de resultados:
Se obtiene un MSE de 995 y una R2 de 0.4284. Los resultados son bastante peores que los obtenidos con el perceptrón multicapa, y de nuevo, aparecen bastantes outliers.

Análisis de los residuos:
La distribución de los errores es la de las imágenes que se muestran a continuación, la de la izquierda es un histograma, mientras que la de la derecha es una comparativa de su valor real con el predicho. Como se ha comentado anteriormente, existen unos cuantos outliers.



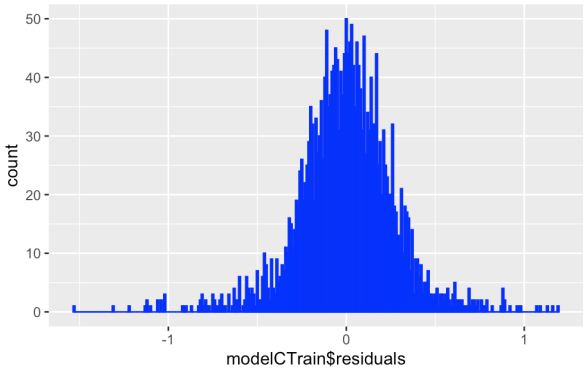


3.- Comparación con MLR y MLR regularizada

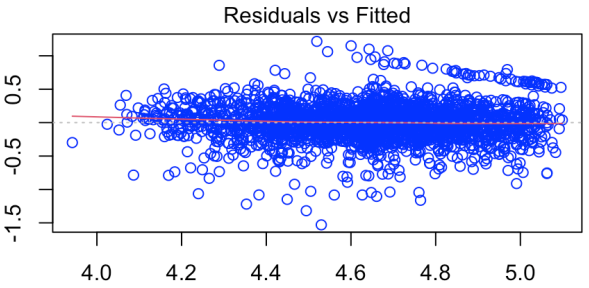
El desempeño de MLR con regularización fue el mismo que el de MLR a secas, por lo que solo se comparará con este. La R2 es de 0.3812, mucho menor comparada con la de la red neuronal, pero bastante similar a la de la SVR.

Por otra parte, los errores son los siguientes:

Training:



Test:



PERCEPTRÓN MULTICAPA CON RSNNS:

```
setwd("/Users/jorgecandia/UNIVERSIDAD/TERCERO/Machine Learning/Prácticas R/Práctica 7 (NN - SVM)")

##### PREPROCESADO DE LOS DATOS #####
library(ISLR)
library(tidyverse)      #data manipulation and visualization
library("psych")  # For multi.hist
library("corrplot") #For corrplot
library("modelr")  # For add_predictions
library(varhandle)
library(dummies)

#data(package = 'ISLR') #Inspecciono datasets de ISLR e importo Wage
wages <- as.data.frame(Wage)


#Preprocesado. Todas las variables discretas vienen ya en factor
any(is.na(wages)) #No hay NAs en el dataframe
summary(wages)
str(wages)
wages <- wages[,-c(6,10)] #Sólo hay una región y quito tb logwage (explico wage)
wages <- dummy.data.frame(wages) #Convierto factores a dummies


#Partición de los datos para training y test
set.seed(100)

inputs<-wages[,-c(23)]
target<-wages[,c(23)]

#Nomalizo variable 1, 2; La explicada (9) ya está en target sóla
variables_normalizadas<-normalizeData(inputs[,c(1,2)], type="0_1")
inputs_norm<-inputs

inputs_norm<-normalizeData(inputs, type="0_1")
target_norm<-normalizeData(target, type="0_1")

sets<-splitForTrainingAndTest(inputs_norm,target_norm,ratio=0.15) #Función mazo útil de
```

```r
RSNNS


##### SE REALIZARÁ EL PERCEPTRÓN MULTICAPA (MLP) CON LA LIBRERÍA RSNNS #####
library(RSNNS)      # for MLP training
library(ROCR)       # for ROC curves


### Creación del modelo de MLP
mlp_model<-mlp(sets$inputsTrain, #Separo variables explicativas de la explicada
               sets$targetsTrain,
               size=c(9,5,3), #number of units in the hidden layer(s)
               initFunc="Randomize_Weights", #Pesos aleatorios al principio
               initFuncParams=c(-0.3, 0.3), # the parameters for the initialization
function
               learnFunc="Std_Backpropagation", #Método de aprendizaje -> backpropagation
               learnFuncParams=c(0.2, 0.0), # the parameters for the learning function
               maxit = 450, #n batches?
               updateFunc="Topological_Order", # ????
               hiddenActFunc="Act_Logistic", # Función de activacion -> sigmoide/logistic
               linOut=TRUE, #sets the activation function of the output units to linear
or logistic
               inputsTest = sets$inputsTest, #Meto datos de test ya desde aquí
               targetsTest = sets$targetsTest)



pred_ts_norm <- predict(mlp_model, sets$inputsTest) #Predicciones de test
plotIterativeError(mlp_model) #Error de training (rojo) VS error de test (negro)
#Plotting error for test
plotRegressionError(sets$targetsTest,pred_ts_norm,pch=3)
plotROC(pred_ts_norm, sets$inputsTest)


# Weights and other information of the mlp
weightMatrix(mlp_model)
extractNetInfo(mlp_model)
summary(mlp_model)
print(mlp_model)

##########################################
###Predicciones de TRAINIG -> MSE y R2 ;Denormalización y cálculo de errores
pred_tr_norm <- predict(mlp_model, sets$inputsTrain)

pred_tr_denorm <-denormalizeData(pred_tr_norm, getNormParameters(target_norm))
target_tr_denorm<- denormalizeData(sets$targetsTrain, getNormParameters(target_norm)) ##
Lo mismo q target

error_tr<-target_tr_denorm - pred_tr_denorm
```

```
#### Exploración de errores

#Histogram of training error
ggplot(data=as.data.frame(error_tr), mapping= aes(x=error_tr))+
  geom_histogram(binwidth=0.5, col=c('blue'))

# Valores reales VS predichos TRAINING
plot(target_tr_denorm,type = "p",col = "red", xlab = "Sample",
     ylab = "medv Value",
     main = "Training: Real (red) - Predicted (blue)") #Valor real

lines(pred_tr_denorm, type = "p", col = "blue") #Valor predicho

# Compare predictions vs real values (cuanto más cerca de la recta mejor)
plot(target_tr_denorm,pred_tr_denorm,col='red',
     main='Training: Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)


###MSE y R2

# Calculating MSE for training
MSE.nn <- sum((error_tr)^2)/nrow(error_tr)

#Estimation of R^2 for training
num<-sum((error_tr)^2)
den<-sum((target_tr_denorm-mean(sets$targetsTrain))^2)

R2<-1-(num/den)
R2_adjust<-(1-(1-R2)*(nrow(target_tr_denorm)-1)/(nrow(target_tr_denorm)-ncol(target_tr_de
norm)-1))



#Plotting error for training
plotRegressionError(sets$targetsTrain,pred_tr_norm,pch=3)
plotROC(pred_tr_norm, sets$targetsTrain)

#################################################################
###Predicciones de TEST -> MSE y R2 ;Denormalización y cálculo de errores
pred_ts_norm <- predict(mlp_model, sets$inputsTest) #Mismo cálculo que antes

pred_ts_denorm <-denormalizeData(pred_ts_norm, getNormParameters(target_norm))
target_ts_denorm<- denormalizeData(sets$targetsTest, getNormParameters(target_norm))

error_ts<-target_ts_denorm-pred_ts_denorm


#### Exploración de errores

#Histogram of residuals
```

```r
ggplot(data=as.data.frame(error_ts), mapping= aes(x=error_ts))+
  geom_histogram(binwidth=0.4, col=c('blue'))

# Valores reales VS predichos TEST
plot(target_ts_denorm,type = "p",col = "red", xlab = "Sample",
     ylab = "medv Value",
     main = "Test: Real (red) - Predicted (blue)")
lines(pred_ts_denorm, type = "p", col = "blue")

# Compare predictions vs real values (cuanto más cerca de la recta mejor)
plot(target_ts_denorm,pred_ts_denorm,col='red',
     main='Test Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)


###MSE y R2

# Calculating MSE for the test data set
MSE.nn_ts <- sum((error_ts)^2)/nrow(error_ts)

#Estimation of R^2 for training
num<-sum((error_ts)^2)
den<-sum((target_ts_denorm-mean(sets$targetsTest))^2)

R2_ts<-1-(num/den)
R2_adjust_ts<-(1-(1-R2_ts)*(nrow(target_ts_denorm)-1)/(nrow(target_ts_denorm)-ncol(target
_ts_denorm)-1))
```

SUPPORT VECTOR REGRESSION CON MSVM-3:

```
#FALTA PARTIR WAGES Y HACER TEST
################ SE REALIZARÁ LA SUPPORT VECTOR REGRESSION (SVR) CON LA LIBRERÍA
e1071 ################
library(e1071)
#Hago un bucle para ver los mejores hiperparámetros // CUIDADO TARDA MUCHO
svr_train<-tune("svm", wage~.,
                data=wages,
                kernel="radial",
                ranges=list(
                  epsilon=seq(0,1,0.1),
                  cost=c(0.01, 0.1, 1, 5, 10, 20),
                  gamma=c(0.1, 0.5, 1, 2, 5, 10)))


summary(svr_train)

#Guardo el mejor modelo
best_model <- svr_train$best.model
summary(best_model)

# Plotting error versus cost
ggplot(data=svr_train$performances, aes(x=cost, y=error,
color=factor(epsilon)))+
  geom_line()+
  geom_point()+
  labs(title="Classification error vs  C & epsilon")+
  theme_bw()+
  theme(legend.position ="bottom")




#####################
###Predicciones de TRAINIG -> MSE y R2, cálculo y exloración de errores
pred_svr_tr <- predict(best_model, wages)
error_svr_tr<- wages$wage - pred_svr_tr


#### Exploración de errores

#Histogram of training error
ggplot(data=as.data.frame(error_svr_tr), mapping= aes(x=error_svr_tr))+
  geom_histogram(binwidth=0.9, col=c('blue'))

# Valores reales VS predichos TRAINING
plot(wages$wage,type = "p",col = "red", xlab = "Sample",
     ylab = "medv Value",
     main = "Training: Real (red) - Predicted (blue)") #Valor real
```

```r
lines(pred_svr_tr, type = "p", col = "blue") #Valor predicho

# Compare predictions vs real values (cuanto más cerca de la recta mejor)
plot(wages$wage,pred_svr_tr,col='red',
     main='Training: Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)


###MSE y R2

# Calculating MSE for training
MSE.nn <- sum((error_svr_tr)^2)/length(error_svr_tr)

#Estimation of R^2 for training
num<-sum((error_svr_tr)^2)
den<-sum((wages$wage-mean(wages$wage))^2)

R2<-1-(num/den)
```