# Análisis exploratorio de datos

## Wizmir, dataset de regresión

### A.1 Descripción del tipo de datos de entrada (lista, dataframe, etc., número de filas, columnas, tipo de datos atómicos, etc.)

Tenemos una tabla de datos que representan datos relacionados de forma tabular, por lo que se almacenan en un dataframe usando los siguientes comandos:

library(readr)

wizmir <- read\_csv("wizmir/wizmir.dat", col\_names = FALSE, comment = "@")

Luego se leen los nombres de los atributos que están en los comentarios iniciales y se almacenan en una variable temporal con:  
names\_wizmir<-scan("wizmir/wizmir.dat",what=character(),sep=" ",nlines=11)

Se les asignan los nombres de variable a las columnas del dataframe con:  
names(wizmir)<- names\_wizmir [seq(4,32,by=3)]

En este caso todas las variables son de tipo cuantitativo continuo así que no es necesario cambiar el tipo.

Es una tabla compuesta de 10 columnas con 1461 observaciones, donde las primeras columnas son los datos de entrada y la décima, Mean\_temperature es la que se quiere predecir usando el regresor.

### A.2 Cálculo de media, desviación estándar, etc.

Usando summary(wizmir) se obtienen los datos estadísticos básicos, incluyendo cuartiles, mínimo, máximo, media, mediana. De estos datos llama la atención las observaciones de precipitación, ya que la mediana, y hasta el 3er cuartil está compuesto de 0.00, mientras que la máxima es de 7.6, indicando que está columna no debe aportar mucha información al dato que se quiere calcular ya que es básicamente consistente con solo algunos datos outliers.

También llama la atención en la columna de velocidad de viento máxima el hecho de que nuevamente la mayoría de los datos, la media y la mediana sean 34.28, casi que pareciera que este dato se usó como relleno para datos que se desconocían en la medición. El hecho de que no existan mediciones NA en los datos refuerza esta posibilidad.

Se calcula la desviación estándar usando sd() para cada variable. Se observa que solo 3 variables tienen desviación por debajo de 1, indicando una gran dispersión en la mayoría de las observaciones.

Luego se aplican los cálculos de skewness() y de kurtosis() a las distintas variables. La kurtosis da valores realmente altos, desde 1.7 hasta 164, tal y como se esperaba a partir de la desviación estándar, los cuartiles y los valores extremos. Mientras que skewness da valores aceptables para las variables excepto presión estándar y precipitación.

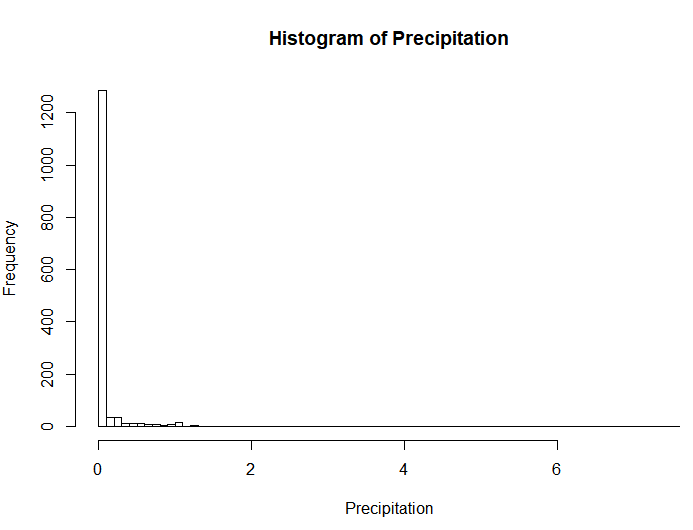
Para confirmar las sospechas se aplica el test de Shapiro-Wilk a las variables y se observa que ninguna sigue una distribución normal.

apply(wizmir,2,shapiro.test)

Finalmente calculo la correlación de los regresores con la variable que se quiere calcular:

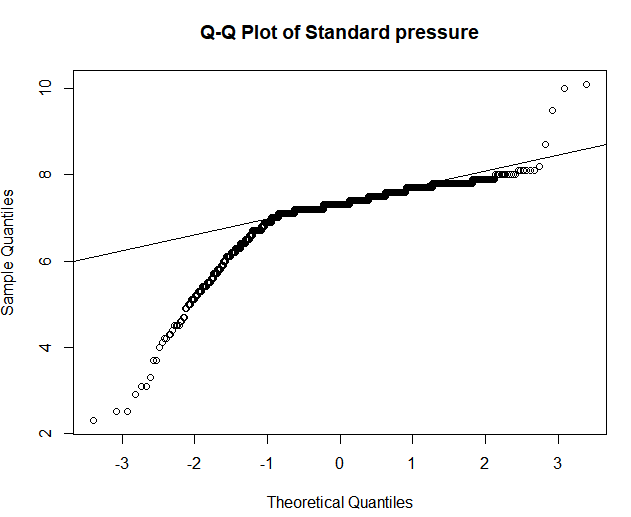
apply(wizmir[1:9],2,cor,wizmir[[10]])

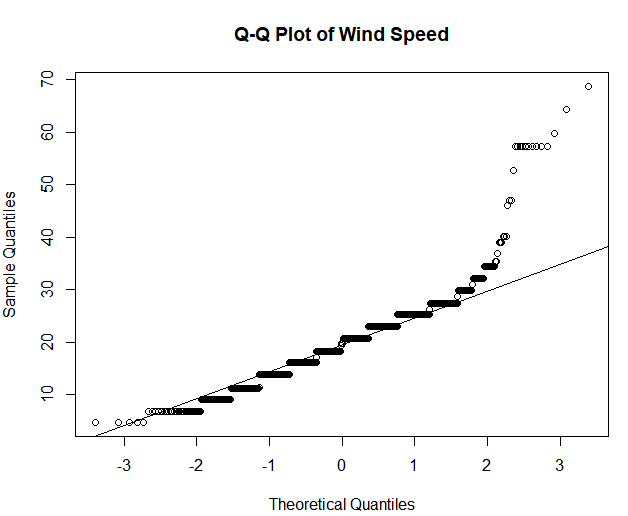
### A.3 Gráficos que permitan visualizar los datos adecuadamente.

Para obtener confirmación visual de los elementos extraídos con anterioridad se realiza un histograma sobre la variable de precipitación:  


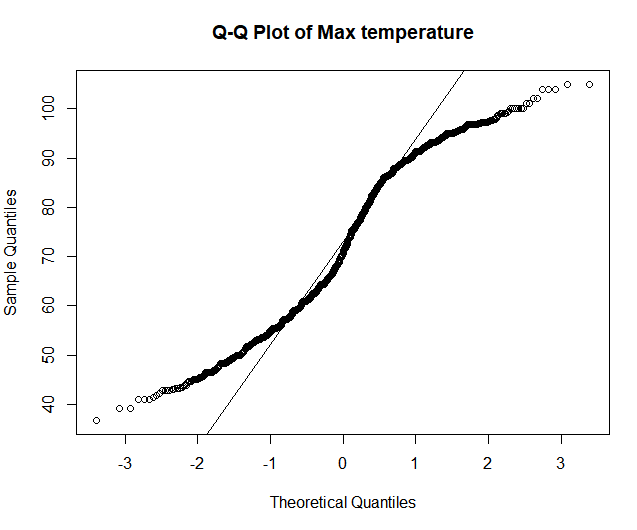
Se ve que la distribución está extremadamente concentrada, dando confirmación a lo anteriormente expuesto.

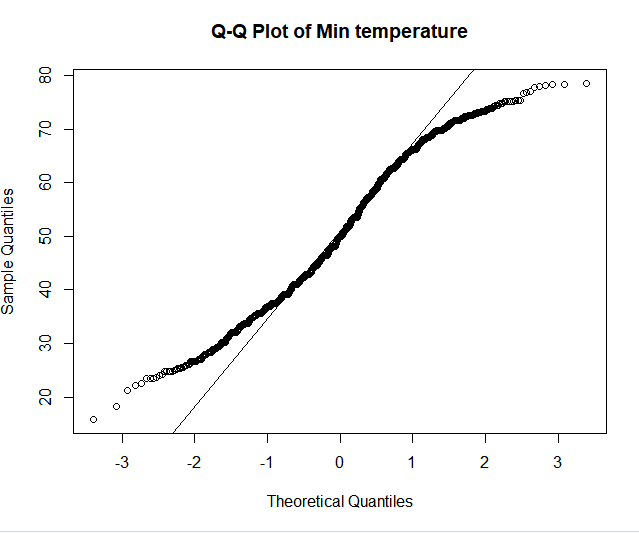
Luego se procede a realizar un Q-Q plot de algunas de las variables que tenían alto kurtosis y skewness, como Standard pressure y wind\_speed:



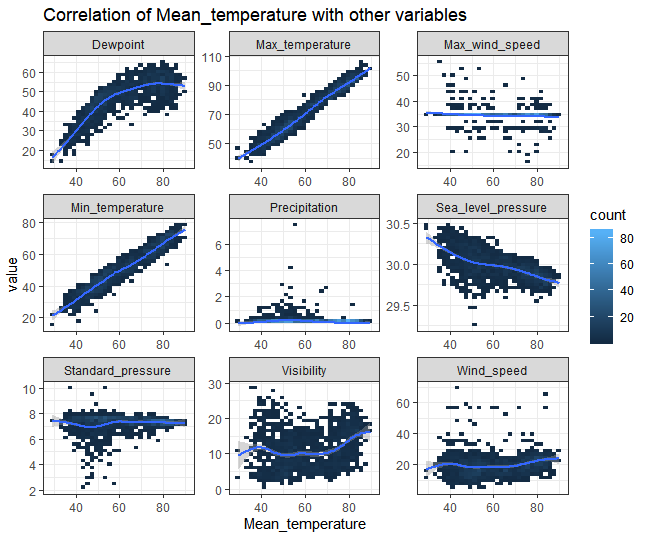


Se realiza unos Q-Q plot de las que indicaban tener una gran correlación, Max temperature y Min temperatura:





Al ser el objetivo una regresión de una variable numérica cuantitativa continua a partir de otras variables numéricas cuantitativas continuas se realiza una comparación de correlación de la misma con las otras variables:



### A.4 Descripción del conjunto de datos a partir de los puntos anteriores.

Los datos son muy heterogéneos y dispersos, hay muchas variables con alta kurtosis y todas fallan el test de Shapiro-Wilk. Esto significa que las variables no están normalizadas y que es probable que haya que realizar manipulación de los datos para poder aplicar algunos modelos de aprendizaje que asumen normalidad.

La variable de velocidad de viento máxima tiene el valor 34.8 repetido una cantidad extrema de veces, y como se nota en la gráfica esto hace que la relación con la variable que se desea medir sea muy baja. La variable de precipitación está extremadamente concentrada en 0, con algunos valores outliers que nuevamente no demuestran ninguna relación con la variable que se desea medir.

Sin embargo vemos que la temperatura máxima y la temperatura máxima tienen una relación lineal casi perfecta y son homoscedasticas, lo que indica que serían buenos regresores, además de contar con valores de correlación significativos.

La variable dewpoint tiene una relación heteroscedastica y se encuentra con una cola inicial, por lo que una previa conversión de los datos podría ayudar a ser interpretada mejor.

La variable de presión sobre el nivel del mar tiene una correlación inversa y posee algunos valores outliers notables pero podría ayudar en el modelo final.

Las variables restantes de presión estándar, visibilidad y velocidad del viento no muestran ninguna relación aparente con la variable que se desea calcular.

## Vowel, dataset de clasificación

### A.1 Descripción del tipo de datos de entrada (lista, data frame, etc., número de filas, columnas, tipo de datos atómicos, etc.)

Tenemos una tabla de datos que representan datos relacionados de forma tabular y con diferentes tipos, por lo que se almacenan en un data frame usando la siguiente función:

library(readr)

vowel <- read\_csv("vowel/vowel.dat", col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

Es importante notar como se redefine a la variable de clasificación como un factor, ya que esto es vital para los algoritmos de clasificación.

Luego se leen los nombres de los atributos que están en los comentarios iniciales y se almacenan en una variable temporal con:  
names\_vowel<-scan("vowel/vowel.dat",what=character(),sep=" ",nlines=15)

Se les asignan los nombres de variable a las columnas del dataframe con:  
names(vowel)<-names\_vowel[seq(4,69,by=5)]

Es una tabla compuesta de 14 columnas con 990 observaciones, donde las primeras columnas son los datos de entrada y la número 14, Class es la variable que se quiere predecir usando el clasificador.

### A.2 Cálculo de media, desviación estándar, etc.

Usando summary(vowel) obtenemos los datos estadísticos básicos, incluyendo cuartiles, mínimo, máximo, media, mediana.

De estos datos llama la atención el perfecto balanceo de las clases de ejemplo poseyendo 90 casos de cada una de las 11 clases, y el hecho de que varias variables parecen poseer distribuciones más o menos normales, aunque algunas tienen mínimos y máximos bastante lejos de la media y de los cuartiles.

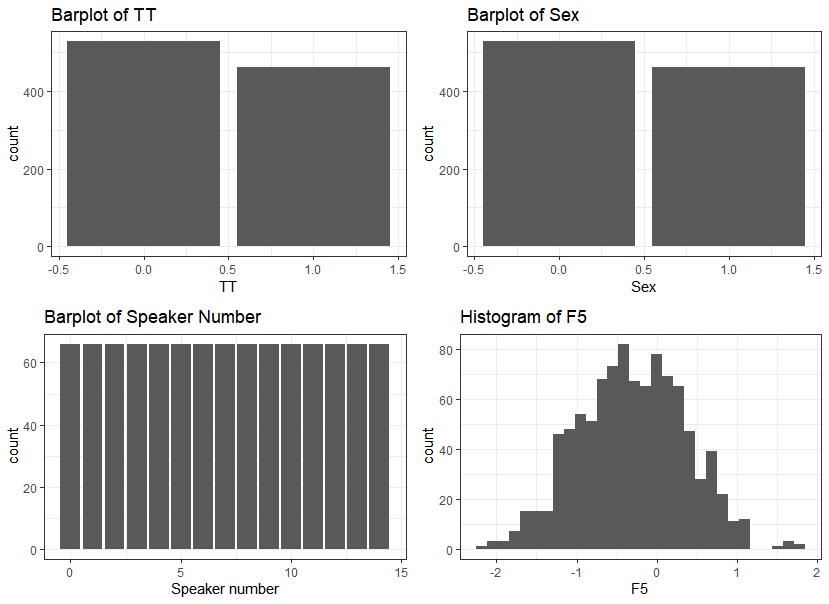
Calculo la desviación estándar usando sd() para cada variable. Se observa que todas las variables tienen una desviación inferior a 1, con excepción de SpeakerNumber que tiene 4.32 y de F1 que tiene 1.17, esto muestra que los datos no tienen mucha variación.

Luego se aplican los cálculos de skewness() y de kurtosis() a las distintas variables. Skewness muestra todos los valores por debajo de 0.5, indicando escasez de picos en los datos, mientras que la kurtosis para todas las variables F0 a F9 da valores por encima de 2, mostrando asimetría en los datos.

Para confirmar las sospechas se aplica el test de Shapiro-Wilk a las variables y se observa que la variable F4 tiene un valor de 0.07, mientras que el resto tiene valores muy por debajo de 0.05 por lo que no podemos asumir distribución normal.  
apply(vowel[1:13],2,shapiro.test)

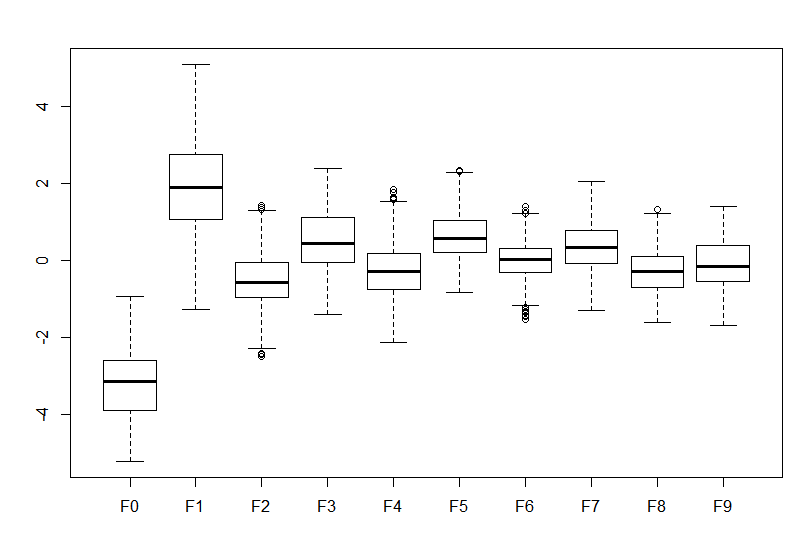
### A.3 Gráficos que permitan visualizar los datos adecuadamente.

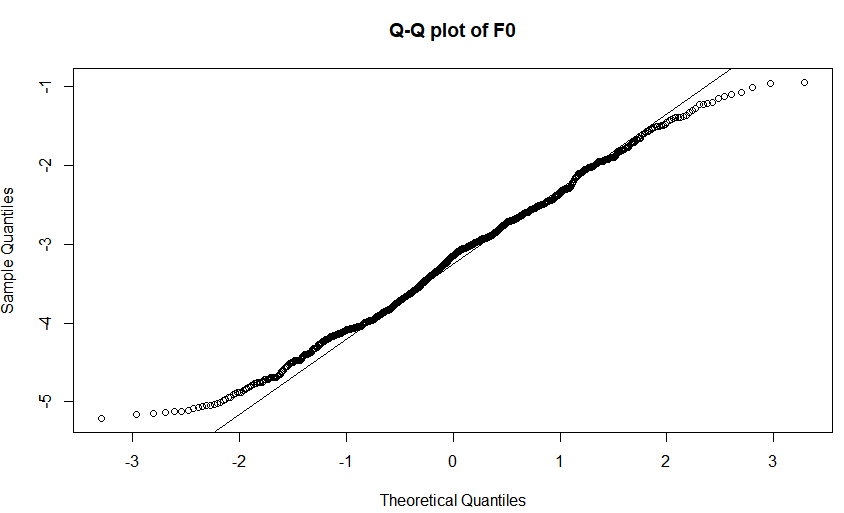
Se comienza con un grupo de bar plots de las variables categóricas TT, SpeakerNumber y Sex, además de un histograma de la variable F5 para confirmar su distribución visualmente



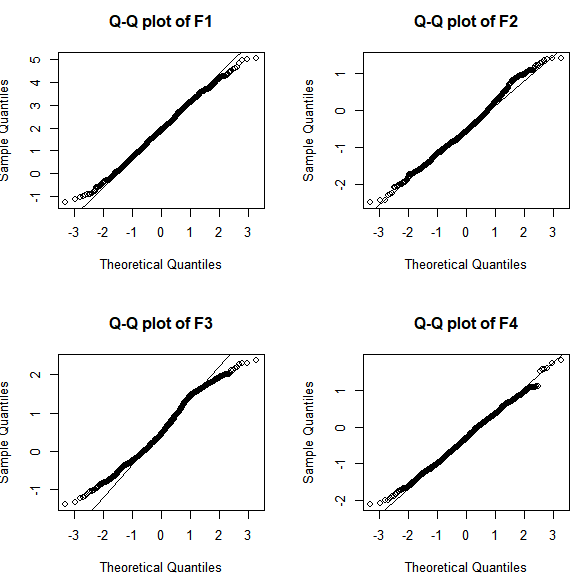
Se observa un muy buen balanceo de las variables categóricas y una concentración muy alta de los datos en F5 entre -1 y 1.

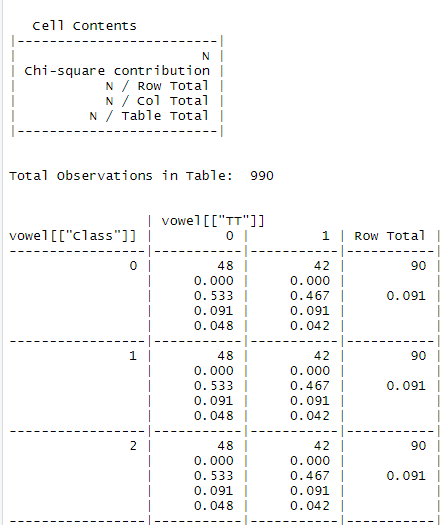
Un boxplot de todas las variables F0 a F9 muestra datos con una distribución muy similar y con pocos valores extremos



Todas las variables F0~F9 siguen aproximadamente la misma distribución 

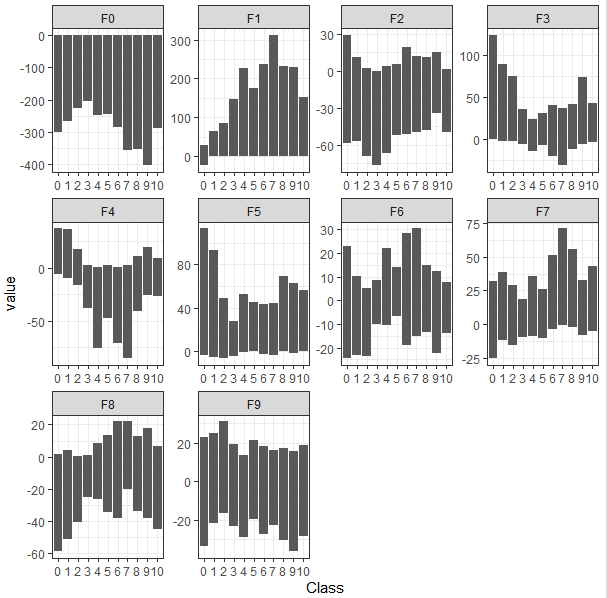
Es una distribución bastante normal, con colas pequeñas.

Usando una tabla de contingencia se observa que la contribución de las variables TT, SpeakerNumber y Sex es 0 con respecto a la variable Class, por lo que no existe relación aparente entre estas y pueden ser descartadas por el clasificador.



(Segmento de la tabla de contingencia)

Se realiza un barplot de las diferentes clases y su relación en las distintas F0~F9



A simple vista no se aprecia ninguna relación directa entre las variables numéricas y la variable categórica que se desea estimar. Algunas funciones tienen una influencia un poco mayor que otras, como se observa en F3, F4 y F5, pero en general el clasificador probablemente tendrá que usar todas y asignar pesos variables para determinar la clase final.

### A.4 Descripción del conjunto de datos a partir de los puntos anteriores.

El conjunto de datos tiene 13 variables de entrada y una de salida. Las 3 variables categóricas TT, Sex y SpeakerNumber tienen una distribución y balanceo casi perfecto entre las clases, de forma tal que la tabla de contingencia no muestra ninguna relación entre la variable que se quiere clasificar y estas.

Entre las 990 observaciones se tienen 90 de cada una de las clases que se quiere clasificar, dando un perfecto equilibrio para el clasificador, esto es muy importante ya que podemos evitar un sobre ajustamiento (overfitting).

Las variables F0 ~ F9 son muy parecidas, con poca variación en los datos y sin picos muy notables, aunque con datos asimétricos un poco concentrados.

También será necesario normalizar las variables F0~F9 para poder aplicar algoritmos que asumen normalidad.

# Regresión

### R.1 Utilizar el algoritmo de regresión lineal simple sobre cada regresor (variable de entrada) para obtener los modelos correspondientes. Si el dataset R asignado incluye más de 5 regresores, seleccione a su criterio los 5 que considere más relevantes. Una vez obtenidos los modelos, elegir el que considere más adecuado para su conjunto de datos según las medidas de calidad conocidas.

Basado en la exploración realizada previamente se puede observar que los dos regresores que tienen una correlación lineal son Max\_temperature y Min\_temperature, mientras que Dewpoint y Sea\_level\_pressure podrían usarse en cierta medida pero cuentan con mucha dispersión y muchos valores outlier.

Usando el siguiente segmento de código para los regresores mencionados anteriormente podemos obtener el modelo lineal y su RMSE, que no es más que una métrica para medir el error basándose en la separación de los puntos originales y los propuestos por el modelo:

fit1=lm(Mean\_temperature~Max\_temperature)

sqrt(sum(abs(Mean\_temperature -fit1$fitted.values)^2)/length(fit1$fitted.values))

También se debe usar la función summary(fit1) para obtener el p-value, adjusted R-squared y los grados de libertad para establecer una comparación completa.

En el caso de estudio se determina que el regresor Max\_temperature es el mejor regresor, con un adjusted R-squared de 0.9576 y un RMSE de 2.957531, además de un p-value inferior a 0.05.

### R.2 Utilizar el algoritmo para regresión lineal múltiple. Justificar adecuadamente si el modelo obtenido aporta mejoras respecto al modelo elegido en el paso anterior.

Para realizar regresión lineal sobre todas las variables se usa el siguiente segmento de código:

fit7=lm(Mean\_temperature~.,data=wizmir)

En este caso la regresión lineal múltiple da un modelo bastante preciso, con un adjusted R-squared de 0.9924, pero es un modelo que usa todas las variables y es probable que esté sobreajustando, revisando con summary(fit7) podemos ir eliminando variables que aporten poco al modelo para disminuir el número de variables dependientes, hasta que llegamos a:

fit9=lm(Mean\_temperature~.-Precipitation -Wind\_speed -Standard\_pressure - Max\_wind\_speed,data=wizmir)

Este modelo tiene un adjusted R-squared de 0.9923 pero tiene 4 menos variables, por lo que es más interpretable.

Se experimenta con varias combinaciones de las 5 variables restantes a ver si se puede obtener un modelo más preciso:

fit11=lm(Mean\_temperature~Min\_temperature+Max\_temperature+Dewpoint+I(Dewpoint^2)+I(log(Sea\_level\_pressure+Visibility)),data=wizmir)

Pero lamentablemente ningún modelo encontrado es capaz de superar la marca de 0.9924 o de mejorar la interpretabilidad.

Sin embargo, se puede intentar obtener un modelo simple usando las dos variables que mejor correlación tienen según el análisis exploratorio de datos: Max\_temperature y Min\_temperature, este modelo podría no ser extremadamente preciso pero al depender de solo dos variables con una relación lineal sería muy fácil de interpretar

fit6=lm(Mean\_temperature~Min\_temperature+Max\_temperature)

Este modelo nos da un adjusted R-squared de 0.9915 y un RMSE de 1.320711 lo cual está muy bien para un modelo lineal que depende solo de dos variables.

### R.3 Aplicar el algoritmo k -NN para regresión no paramétrica.

Uno de los métodos para aplicar K-NN usando K-fold cross-validation con 5 particiones es el siguiente:

library(kknn)

nombre<-"wizmir/wizmir"

run\_knn\_fold <- function( i , x, tt = "test") {

file<- paste(x,"-5-",i,"tra.dat",sep ="")

x\_tra<- read.csv(file, comment.char="@")

file<- paste(x,"-5-",i,"tst.dat",sep ="")

x\_tst<- read.csv(file, comment.char="@")

In<- length(names( x\_tra))- 1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X", 1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

if(tt == "train") {

test<-x\_tra

}

else{

test<-x\_tst

}

fitMulti=kknn(Y~.,x\_tra,test)

yprime=fitMulti$fitted.values

sum(abs(test$Y-yprime)^2)/length(yprime) ##MSE

}

knnMSEtrain<-mean(sapply (1:5,run\_knn\_fold,nombre ,"train"))

knnMSEtest<-mean(sapply (1:5,run\_knn\_fold,nombre ,"test"))

En las variables de knnMSEtrain y knnMSEtest se almacenan los valores de la media de error obtenido en las diferentes particiones, 2.5383 sobre el conjunto de entrenamiento y 6.060107 sobre el conjunto de prueba. Realizando el mismo proceso de K-fold cross-validation sobre las mismas 5 particiones para el mejor modelo de regresión lineal obtenemos 1.564527 sobre las particiones de entrenamiento y 1.604908 sobre particiones de prueba. Para el caso de un modelo lineal simple con dos variables se tiene un MSE de 1.744869 sobre el conjunto de entrenamiento y 1.751218 sobre el conjunto de test.

Basado en estos resultados se puede concluir que K-nn no supera a la regresión lineal para este problema de regresión.

### R.4 Comparar los resultados de los dos algoritmos de regresión múltiple

Se comienza cargando los datos de las tablas de resultado con:

resultados <- read.csv("regr\_train\_alumnos.csv")

tablatra <- cbind(resultados[,2:dim( resultados)[2]])

colnames(tablatra ) <- names(resultados)[2:dim(resultados)[2]]

rownames (tablatra ) <- resultados [,1]

Ya con estos datos se puede realizar la normalización necesaria para realizar una comparación usando el método de Wilcoxon sobre algoritmos de regresión mediante el siguiente comando, el cual utiliza como base de comparación a los resultados del modelo de regresión lineal y a K-NN como algoritmo con el que comparar:

difs <- (tablatra[,2] - tablatra[,1]) / tablatra[,2]

Y preparamos entonces la tabla de comparación para realizar la comparación mediante el siguiente comando:

wilc\_1\_2 <- cbind(ifelse(difs <0, abs( difs)+0.1, 0+0.1), ifelse(difs>0, abs(difs)+0.1, 0+0.1))

Aquí hay que notar que Wilcoxon no funciona con valores igual a 0, por lo que se añade 0.1 a todos los resultados. La primera columna va a contener al algoritmo de comparación y va a tener el valor absoluto de la diferencia si esta es menor a 0, o 0.1 de lo contrario, mientras que la segunda columna tendría al algoritmo base de la comparación y contendría el valor absoluto de la diferencia si este es mayor a 0 o 0.1 de lo contrario.

Ya con esta separación se puede realizar el test de Wilcoxon:

LMvsKNNtst <-wilcox.test(wilc\_1\_2[,1], wilc\_1\_2[,2], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic

pvalue <- LMvsKNNtst$p.value

LMvsKNNtst <- wilcox.test(wilc\_1\_2[,2], wilc\_1\_2[,1], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic

Rmas #168

Rmenos #3

pvalue #3.814697e-05

El Rmas es el valor del modelo lineal mientras que el Rmenos es el de K-nn y 1-pvalue nos daría el grado de confianza en la diferencia entre ambos algoritmos. En este caso la confianza en que son diferentes es enorme y muy cercana a 100%.

No obstante, aquí se usa los datos en las tablas de entrenamiento, si usamos las tablas que contienen los valores de precisión de los algoritmos en los test entonces se observarían resultados muy diferentes y más precisos.

Al repetir el proceso se obtiene un Rmas de 78 para K-nn, un Rmenos de 93 para regresión lineal y un pvalue de 0.7660294. Al realizar la operación de 1-0.7660294\*100% obtenemos aproximadamente un valor de 23.4% de confianza en que los algoritmos son similares.

Se prosiguen realizando comparaciones, esta vez usando el método de Friedman para ambas tablas:

test\_friedman <- friedman.test(as.matrix(tablatra))

test\_friedman #chi-squared = 20.333, p-value=3.843e-05

Aquí obtenemos un p-value tan pequeño que se puede asegurar con total confianza que hay al menos un algoritmo muy diferente a los demás, al aplicar la comparación sobre la tabla de test (la más importante) se obtiene un p-value de de 0.01467 que es igualmente bastante pequeño.

Para determinar cuál es el algoritmo tan significativamente diferente se usa el método de comparación de Holm en ambas tablas:

#Holm training data algorithm comparison

#Prepare the groups

tam <- dim(tablatra)

groups <- rep(1:tam[2], each=tam[1])

#Do the test

pairwise.wilcox.test(as.matrix(tablatra), groups, p.adjust = "holm", paired = TRUE)

Aquí la tabla resultante da valores muy similares y pequeños entre todos los algoritmos, por lo que no se observa un claro ganador, sin embargo al realizar la comparación sobre la tabla de test se obtiene:

1 2

2 0.580 -

3 0.081 0.108

Donde 3 representa el algoritmo M5 y tiene una clara diferencia y superioridad sobre ambos K-nn y Lm, mientras que K-nn y Lm no muestran gran diferencia entre ellos

# Clasificación

### C.1 Utilizar el algoritmo k-NN probando con diferentes valores de k. Elegir el que considere más adecuado para su conjunto de datos. Analice que ocurre en los valores de precisión en training y test con los diferentes valores de k.

Para usar K-NN es necesario operar con solo valores numéricos y un factor como el objetivo de clasificación, esta operación se realiza durante la entrada de datos:

read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

No es necesario crear dummies para las 3 primeras variables categóricas ya que solo toman valor de 0 y de 1.

Para medir la precisión en training usando los 10 segmentos preparados con anterioridad para hacer cross validation y a la vez medir la precisión con varios valores de K, para K-nn, se puede usar la siguiente función:

#Do knn with only training data

run\_knn\_fold\_train <- function(i,x,k) {

file<- paste(x,"-10-",i,"tra.dat",sep ="")

x\_tra<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

In<- length(names( x\_tra))-1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

#Normalize data and drop the three categorical variables with no contribution for knn algorithm

x\_tra\_n <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

#Do knn only on training data

knn\_test\_pred <- knn(train = x\_tra\_n, test = x\_tra\_n, cl = x\_tra$Y, k=k)

postResample(pred = knn\_test\_pred, obs = x\_tra$Y)[1] #Accuracy column

}

#helper function to simply doing 10 cv with 10 different k values for knn

run\_knn\_fold\_train\_different\_k <- function(k){

knnMSEtrain<-mean(sapply(1:10,run\_knn\_fold\_train,nombre,k))

knnMSEtrain

}

train\_values<-sapply(1:10,run\_knn\_fold\_train\_different\_k)

train\_values #Knn values with different K using CV, most accurate is K=1, but it's likely to be overfitting

#[1] 1.0000000 0.9950617 0.9930415 0.9810325 0.9792368 0.9652076 0.9626263

#[8] 0.9424242 0.9380471 0.9139169

Nos da como resultado que mientras menor la K, mayor la precisión, con un 100% máximo, mientras que con k=10 se obtiene una precisión de 91.3%. En este caso vemos que si hacemos las pruebas con el mismo conjunto de entrenamiento, mientras menor la K, mayor es la probabilidad de estar entrenando el modelo por encima de su límite.

Es importante notar que en el segmento de código:

x\_tra\_n <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

Eliminamos las otras variables categóricas que no contribuyen a la clase y tienen poca relación, mejorando así los resultados obtenidos por K-nn.

Para realizar el cross validation de forma correcta usando los conjuntos de test y probando con diferentes K se puede usar el siguiente segmento de código:

#Proper Knn with both train and test data

run\_knn\_fold\_test <- function(i,x,k) {

file<- paste(x,"-10-",i,"tra.dat",sep ="")

x\_tra<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

file<- paste(x,"-10-",i,"tst.dat",sep ="")

x\_tst<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

In<- length(names( x\_tra))-1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X",1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

#Normalize data and drop the three categorical variables with no contribution for knn algorithm

x\_tra\_n <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

x\_tst\_n <- as.data.frame(lapply(x\_tst[,4:13], scale, center = TRUE, scale = TRUE))

#Do Knn and then compare with test data

knn\_test\_pred <- knn(train = x\_tra\_n, test = x\_tst\_n, cl = x\_tra$Y, k=k)

postResample(pred = knn\_test\_pred, obs = x\_tst$Y)[1] #Accuracy column

}

#Helper function

run\_knn\_fold\_test\_different\_k <- function(k,type){

knnMSEtest<-sapply(1:10,run\_knn\_fold\_test,nombre,k)

knnMSEtest

}

#Obtaining all KNN models accuracy for K between 1 and 10 for each partition

test\_values<-sapply(1:10,run\_knn\_fold\_test\_different\_k)

#Obtain the mean and standard deviation for each tested K

knn\_means<-apply(test\_values,2,mean)

knn\_sd<-apply(test\_values,2,sd)

knn\_means

knn\_sd

#Save 5 KNN models with K=1,3,5,7,10 for comparison with other models for each test

tests\_knn<-test\_values[,c(1,3,5,7,10)]

tests\_knn

#For each K in knn from 1 to 10 we get the following accuracy

#0.9909091 0.9717172 0.9696970 0.9414141

#0.9373737 0.9272727 0.9131313 0.8888889

#0.8777778 0.8363636

#And the following standard deviation

#0.01004474 0.01831848 0.02233417

#0.02064607 0.03222488 0.02418624

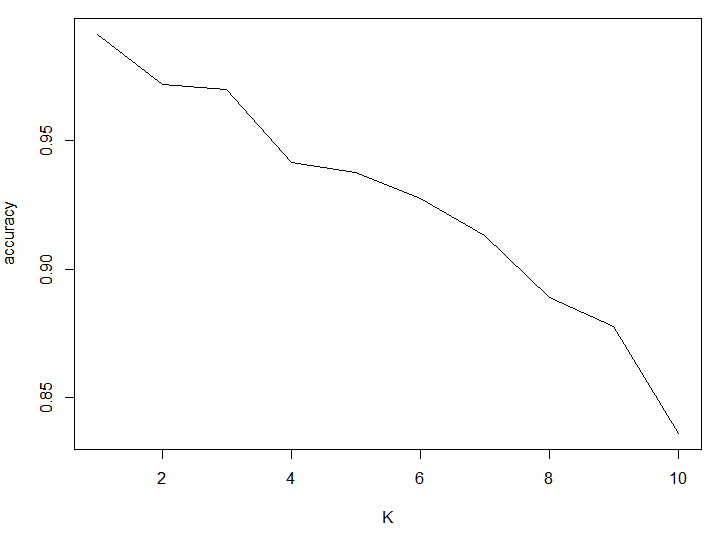
#0.03305842 0.02735366 0.05225910

#0.05233497

Esta vez se obtiene una precisión de 99.1% en el test para K=1 y de 83.6% para K=10 con una desviación del 1% para K=1 y 5% para K=10.

En el apéndice se incluye además el código para realizar K-nn mediante la librería caret, usando valores de K desde 5 hasta 23, donde el mejor modelo es 5 con una precisión de 94.2%, valor que es bastante cercano al obtenido realizando K-nn de forma manual para K=5.

A mayor K, menor precisión:



### C.2 Utilizar el algoritmo LDA para clasificar. No olvide comprobar las asunciones.

El algoritmo LDA tiene varias asunciones para que funcione correctamente, y si estas no se cumplen el algoritmo tendrá un desempeño inferior. Estas asunciones son:

1. **Normalidad en los clasificadores**
2. **Varianza común**
3. **Elementos de los datos de prueba aleatorios**
4. **De 5 a 10 más ejemplos que predictores.**

Para determinar la normalidad se puede usar el test de Shapiro mediante el siguiente comando:

apply(vowel[4:13],2,shapiro.test)

El cual da un p-value inferior a 0.05 en casi todas las variables, indicando una necesidad de normalización antes de aplicar LDA.

Para la varianza se usa el comando de:

var(vowel\_n)

El cual nos da suficiente varianza entre las variables.

Se asume que las observaciones en los datos son aleatorias.

Con el comando:

dim(vowel\_n)

Podemos observar que hay 990 observaciones para 10 predictores, lo cual es superior al mínimo de 5 veces más sugerido por LDA.

Basado en estos datos se puede proceder a realizar LDA usando cross validation en 10 segmentos como se muestra en el siguiente segmento de código:

run\_lda\_fold <- function(i,x) {

#Load train and test data of the current fold

file<- paste(x,"-10-",i,"tra.dat",sep ="")

x\_tra<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

file<- paste(x,"-10-",i,"tst.dat",sep ="")

x\_tst<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

In<- length(names( x\_tra))-1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X",1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

#Normalize data

x\_tra[4:13] <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

x\_tst[4:13] <- as.data.frame(lapply(x\_tst[,4:13], scale, center = TRUE, scale = TRUE))

#drop predictors with 0 variance and near 0 correlation

x\_tra<-x\_tra[-c(1,2,3)]

#Do the algorithm calculations

lda.fit <- lda(Y~.-Y,data=x\_tra)

#Predicted values against test data

ld.pred<-predict(lda.fit,x\_tst)

#Return the mean of how many predicted values are correct as an accuracy metric

mean(ld.pred$class==x\_tst$Y)

}

lda\_cv<-sapply(1:10,run\_lda\_fold,nombre) #Obtain the mean accuracy on each fold

mean(lda\_cv) #average them to obtain the final accuracy of the algorithm

sd(lda\_cv) #standard deviaton to determine error

Lo cual da un resultado final de 0.6171717, es decir una precisión aproximada de 61.8% por lo que se puede concluir que LDA no es un buen modelo para este problema. La desviación típica es de 4.1%.

En el apéndice se incluye el código para ejecutar LDA usando la librería Caret generando los segmentos usando la misma librería y que da una precisión promedio de 61%.

### C.3 Utilizar el algoritmo QDA para clasificar. No olvide comprobar las asunciones.

QDA es similar a LDA en su forma de programación e implementación pero tiene asunciones ligeramente diferentes, en QDA:

1. El número de predictores debe ser inferior al número de casos dentro de cada clase.
2. Dentro de cada clase, los predictores no pueden tener valores extremos de colinearidad.

En este caso contamos con 11 clases y 10 predictores, con 99 casos dentro de cada clase. Se cumple la primera condición ya que tenemos más casos en cada clase que el número de predictores.

Ya que son 11 clases y 10 predictores tendríamos que determinar la varianza de 110 casos, por lo que usamos la media de varianza dentro de cada clase para los predictores:

var\_l<-vector("numeric",11)

for(x in 1:11){

v\_0<-sapply(vowel[vowel$Class == as.character(x-1),][4:13], var)

var\_l[x]<-mean(v\_0)

}

var\_l

Aquí obtenemos varianzas en el rango de 0.18 a 0.57 lo cual no es extremo.

Similar a LDA se realiza un cross validation en 10 segmentos como se muestra en el siguiente segmento de código:

run\_qda\_fold <- function(i,x) {

#Load train and test data of the current fold

file<- paste(x,"-10-",i,"tra.dat",sep ="")

x\_tra<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

file<- paste(x,"-10-",i,"tst.dat",sep ="")

x\_tst<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

In<- length(names( x\_tra))-1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X",1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

#normalize them

x\_tra[4:13] <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

x\_tst[4:13] <- as.data.frame(lapply(x\_tst[,4:13], scale, center = TRUE, scale = TRUE))

#Do the algorithm calculations

qda.fit <-qda(Y~.-Y, data=x\_tra)

#Test

qda.pred <- predict(qda.fit,x\_tst)

#Return the mean of the test as an accuracy metric

mean(qda.pred$class==x\_tst$Y)

}

qda\_cv<-sapply(1:10,run\_qda\_fold,nombre) #Obtain the mean accuracy on each fold

mean(qda\_cv) #average them to obtain the final accuracy of the algorithm

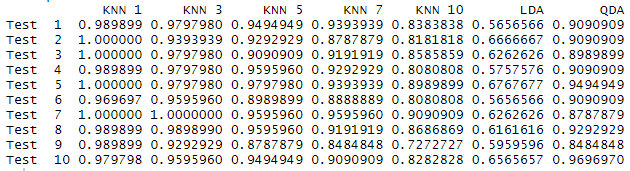
sd(qda\_cv) #standard deviaton to determine error

Lo cual da un resultado final de 0.9111111, es decir una precisión aproximada de 91.1% por lo que se puede concluir que QDA es un modelo relativamente aceptable para este problema. La desviación típica es de 3.4%, lo cual es un poco elevado.

En el apéndice se incluye el código para ejecutar QDA usando la librería Caret generando los segmentos usando la misma librería y que da una precisión promedio de 91.5%.

### C.4 Comparar los resultados de los tres algoritmos.

De los 3 algoritmos Knn con K=1 tiene la mejor precisión con la desviación típica más baja, lo cual nos indica que realmente es el mejor de todos con una precisión del 99.1% y una desviación de solo un 1%. No obstante se procede a utilizar tests estadísticos de comparación múltiple para realizar una comparación entre los diferentes algoritmos y sus resultados en los diferentes tests usando la tabla:



Y los siguientes segmentos de código:

#Wilcoxon test

#No need to scale the error in classification as all belong in the range 0~1.

#Best KNN with K=1 against LDA

LMvsKNNtst <-wilcox.test(comparison\_table[,1], comparison\_table[,6], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic #This is the value for Linear regression model

pvalue <- LMvsKNNtst$p.value #This is the p-value of the comparison

LMvsKNNtst <- wilcox.test(comparison\_table[,6], comparison\_table[,1], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic #This is the value for K-nn model

Rmas #55

Rmenos #0

pvalue #0.005857099

#We can assess with total confidence that they are indeed different

#Best KNN with K=1 against QDA

LMvsKNNtst <-wilcox.test(comparison\_table[,1], comparison\_table[,7], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic #This is the value for Linear regression model

pvalue <- LMvsKNNtst$p.value #This is the p-value of the comparison

LMvsKNNtst <- wilcox.test(comparison\_table[,7], comparison\_table[,1], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic #This is the value for K-nn model

Rmas #55

Rmenos #0

pvalue #0.005857099

#We can assess with total confidence that they are indeed different, it's surprising that the numbers are identical,

#but just goes to show how much Knn with K=1 dominates the comparison

#Best KNN with LDA against QDA

LMvsKNNtst <-wilcox.test(comparison\_table[,6], comparison\_table[,7], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic #This is the value for Linear regression model

pvalue <- LMvsKNNtst$p.value #This is the p-value of the comparison

LMvsKNNtst <- wilcox.test(comparison\_table[,7], comparison\_table[,6], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic #This is the value for K-nn model

Rmas #0

Rmenos #55

pvalue #0.005825024

#Similar to the cases above, we are very confident that they perform differently.

#Friedman training data algorithm comparison

test\_friedman <- friedman.test(as.matrix(comparison\_table))

test\_friedman #chi-squared = 55.211, p-value=4.203e-10, significantly inferior to 0.05 which shows at least one algorithm

#is very different from another

#Holm algorithm comparison

#Prepare the groups

tam <- dim(comparison\_table)

groups <- rep(1:tam[2], each=tam[1])

#Do the test

pairwise.wilcox.test(as.matrix(comparison\_table), groups, p.adjust = "holm", paired = TRUE)

# 1 2 3 4 5 6

# 2 0.104 - - - - -

# 3 0.104 0.104 - - - -

# 4 0.104 0.104 0.104 - - -

# 5 0.041 0.104 0.104 0.104 - -

# 6 0.104 0.104 0.041 0.041 0.104 -

# 7 0.104 0.104 0.104 0.905 0.104 0.104

# 1-5 3-6 4-6 below 0.05, so K-nn with K=1 and K=10, and LDA with Knn k=5 and K = 7 are different.

apply(comparison\_table,2,mean)

Aquí se observa que K-nn con K=1 domina completamente el espacio de prueba y que todos los demás algoritmos son significamente inferiores a los resultados obtenidos por K-NN con K= 1. Esto no es común y es un caso particular de este dataset, en la mayoría de los casos K-NN con K =1 hace mucho sobreajustamiento (overfitting) y no generaliza bien.

# Apéndice: Código R de Wizmir completo

#Data entry

library(readr)

wizmir <- read\_csv("wizmir/wizmir.dat", col\_names = FALSE, comment = "@")

names\_wizmir<-scan("wizmir/wizmir.dat",what=character(),sep=" ",nlines=11)

names(wizmir)<-names\_wizmir[seq(4,32,by=3)]

attach(wizmir)

#EDA

summary(wizmir) #cuartile, min, max, mean, median

apply(wizmir,2,sd) #Standard deviation of the regressors

library(moments) #library necessary for skewness and kurtosis

apply(wizmir,2,skewness) #skewness of the data, basically how condensed and with peaks it is

apply(wizmir,2,kurtosis) #how assymetrical the data is

apply(wizmir,2,shapiro.test) #normalization test

apply(wizmir[1:9],2,cor,wizmir[[10]]) #correlation between regressors and the target variable

#Visual analysis

hist(wizmir[["Precipitation"]],main = "Histogram of Precipitation",xlab="Precipitation",breaks=100) #histogram to check distribution

#Multipe Q-Q plots to visualize the normalization of the data

qqnorm(as.matrix(wizmir["Standard\_pressure"]),main="Q-Q Plot of Standard pressure")

qqline(as.matrix(wizmir["Standard\_pressure"]))

qqnorm(as.matrix(wizmir["Wind\_speed"]),main="Q-Q Plot of Wind Speed")

qqline(as.matrix(wizmir["Wind\_speed"]))

qqnorm(as.matrix(wizmir["Max\_temperature"]),main="Q-Q Plot of Max temperature")

qqline(as.matrix(wizmir["Max\_temperature"]))

qqnorm(as.matrix(wizmir["Min\_temperature"]),main="Q-Q Plot of Min temperature")

qqline(as.matrix(wizmir["Min\_temperature"]))

#Visual correlation of the regressors with the target variable.

library(ggplot2)

library(tidyr)

wizmir %>%

gather(-Mean\_temperature, key = "var", value = "value") %>%

ggplot(aes(x = Mean\_temperature, y = value)) +

geom\_bin2d() +

stat\_smooth() +

facet\_wrap(~ var, scales = "free") +

labs(title="Correlation of Mean\_temperature with other variables") +

theme\_bw()

#R.1

#Utilizar el algoritmo de regresión lineal simple sobre cada regresor para obtener los modelos.

#Si el dataset contiene más de 5, usar las 5 más relevantes.

#Después escoger el más adecuado según las medidas de calidad.

#Max\_temperature has the best correlation and is a pretty normalized regressor

fit1=lm(Mean\_temperature~Max\_temperature)

summary(fit1)

plot(Mean\_temperature~Max\_temperature)

abline(fit1,col="red")

sqrt(sum(abs(Mean\_temperature -fit1$fitted.values)^2)/length(fit1$fitted.values))

#0.9576 Adjusted R squared

#2.957531 RMSE

#P-value below 0.05 and a small adjusted R squared of 0.9576 alongside 2.957531 RMSE shows this is a good model using only one regressor

fit2=lm(Mean\_temperature~Min\_temperature)

summary(fit2)

plot(Mean\_temperature~Min\_temperature)

abline(fit2,col="red")

sqrt(sum(abs(Mean\_temperature -fit2$fitted.values)^2)/length(fit2$fitted.values))

#0.919 Adjusted R squared

#4.08958 RMSE

fit3=lm(Mean\_temperature~Dewpoint)

summary(fit3)

plot(Mean\_temperature~Dewpoint)

abline(fit3,col="red")

sqrt(sum(abs(Mean\_temperature -fit3$fitted.values)^2)/length(fit3$fitted.values))

#0.6152 Adjusted R squared

#8.911483 RMSE

fit4=lm(Mean\_temperature~Sea\_level\_pressure)

summary(fit4)

plot(Mean\_temperature~Sea\_level\_pressure)

abline(fit4,col="red")

#0.3394 Adjusted R squared

#No need to further calculate when previous models are superior

fit5=lm(Mean\_temperature~Visibility)

summary(fit5)

plot(Mean\_temperature~Visibility)

abline(fit5,col="red")

#0.05042 Adjusted R squared

#R.2

#Utilizar el algoritmo de regresión lineal múltiple. Justificar si el modelo es mejor que el anterior.

#Tener en cuenta posibles interacciones y no linealidad.

#Classic combination of two linear regressors with high correlation to the target variable

fit6=lm(Mean\_temperature~Min\_temperature+Max\_temperature)

summary(fit6)

sqrt(sum(abs(Mean\_temperature -fit6$fitted.values)^2)/length(fit6$fitted.values))

#0.9915

#1.320711 RMSE

#P-value below 0.05 and a small adjusted R squared of 0.9915 alongside 1.320711 RMSE shows this is a good model using only two regressors

#All inclusive model

fit7=lm(Mean\_temperature~.,data=wizmir)

summary(fit7)

sqrt(sum(abs(Mean\_temperature -fit7$fitted.values)^2)/length(fit7$fitted.values))

#0.9924

#1.251833 RMSE

#Lowest error but uses all variables, could lead to overfitting or an hard to understand model

#Removing variables with low contribution

fit8=lm(Mean\_temperature~.-Precipitation -Wind\_speed,data=wizmir)

summary(fit8)

#0.9924

#Further removing

fit9=lm(Mean\_temperature~.-Precipitation -Wind\_speed -Standard\_pressure - Max\_wind\_speed,data=wizmir)

summary(fit9)

sqrt(sum(abs(Mean\_temperature -fit9$fitted.values)^2)/length(fit9$fitted.values))

#0.9923

#1.257494 RMSE

#Best model found in this search, very low error and includes 4 less regressors than the multiple model.

#Further removing, increase in error

fit10=lm(Mean\_temperature~.-Dewpoint - Sea\_level\_pressure -Precipitation -Wind\_speed -Standard\_pressure - Max\_wind\_speed,data=wizmir)

summary(fit10)

#0.9917

#Testing with non lineal interactions

fit11=lm(Mean\_temperature~Min\_temperature+Max\_temperature+Dewpoint+I(Dewpoint^2)+I(log(Sea\_level\_pressure+Visibility)),data=wizmir)

summary(fit11)

sqrt(sum(abs(Mean\_temperature -fit11$fitted.values)^2)/length(fit11$fitted.values))

#0.9923

#1.271545 RMSE

#Further testing

fit12=lm(Mean\_temperature~Min\_temperature+Max\_temperature+log(Dewpoint)+Sea\_level\_pressure+Visibility,data=wizmir)

summary(fit12)

sqrt(sum(abs(Mean\_temperature -fit11$fitted.values)^2)/length(fit11$fitted.values))

#0.9923

#1.271545 RMSE

#Other tests deleted in order to avoid bloating code

#R.3

#Aplicar el algoritmo k-NN para regresión

#Load k-nn library and setup location of the folder containing the folds for cross validation

library(kknn)

nombre<-"wizmir/wizmir"

#Function to do K-nn manually over the folds

run\_knn\_fold <- function( i , x, tt = "test") {

file<- paste(x,"-5-",i,"tra.dat",sep ="")

x\_tra<- read.csv(file, comment.char="@")

file<- paste(x,"-5-",i,"tst.dat",sep ="")

x\_tst<- read.csv(file, comment.char="@")

In<- length(names( x\_tra))- 1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X", 1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

if(tt == "train") {

test<-x\_tra

}

else{

test<-x\_tst

}

fitMulti=kknn(Y~.,x\_tra,test)

yprime=fitMulti$fitted.values

sum(abs(test$Y-yprime)^2)/length(yprime) ##MSE

}

knnMSEtrain<-mean(sapply (1:5,run\_knn\_fold,nombre ,"train")) #train accuracy average over the 5 folds

knnMSEtest<-mean( sapply (1:5,run\_knn\_fold,nombre ,"test")) #test accuracy average over the 5 folds

knnMSEtrain #2.5383

knnMSEtest #6.060107

#Function to do linear regression manually over the folds

run\_lm\_fold <- function( i , x, tt = "test", formula) {

file<- paste(x,"-5-",i,"tra.dat",sep ="")

x\_tra<- read.csv(file, comment.char="@")

file<- paste(x,"-5-",i,"tst.dat",sep ="")

x\_tst<- read.csv(file, comment.char="@")

In<- length(names( x\_tra))- 1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X", 1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

if(tt == "train") {

test<-x\_tra

}

else{

test<-x\_tst

}

fitMulti =lm(formula,x\_tra)

yprime=predict( fitMulti,test )

sum(abs(test$Y -yprime)^2)/length(yprime) ##MSE

}

lmMSEtrain<-mean(sapply (1:5,run\_lm\_fold,nombre ,"train",Y~.)) #train accuracy average over the 5 folds for best model

lmMSEtest <-mean(sapply (1:5,run\_lm\_fold,nombre,"test",Y~.)) #test accuracy average over the 5 folds for best model

lmMSEtrain #1.564527

lmMSEtest #1.604908

lmMSEtrain<-mean(sapply (1:5,run\_lm\_fold,nombre ,"train",Y~X1+X2)) #train accuracy average over the 5 folds for simple lineal model

lmMSEtest <-mean(sapply (1:5,run\_lm\_fold,nombre,"test",Y~X1+X2)) #test accuracy average over the 5 folds for simple lineal

lmMSEtrain #1.744869

lmMSEtest #1.751218

lmMSEtrain<-mean(sapply (1:5,run\_lm\_fold,nombre ,"train",Y~. -X4 -X6 -X8 -X9)) #train accuracy average over the 5 folds for 5 regressors model

lmMSEtest <-mean(sapply (1:5,run\_lm\_fold,nombre,"test",Y~.-X4 -X6 -X8 -X9)) #test accuracy average over the 5 folds for 5 regressors model

lmMSEtrain #1.579781

lmMSEtest #1.609519

#R.4

#Comparar los resultados de los dos algoritmos de regresión múltiple entre sí.

#Comparar con los resultados del M5 que están en la tabla de resultados disponibles.

#Remplazar en el .csv con los valores propios y usar Wilcoxon, Friedman y Holms.

#load data from the .csv with the different accuracy results from the algorithms applied over different datasets

#Training data accuracy

resultados <- read.csv("regr\_train\_alumnos.csv")

tablatra <- cbind(resultados[,2:dim( resultados)[2]])

colnames(tablatra ) <- names(resultados)[2:dim(resultados)[2]]

rownames (tablatra ) <- resultados [,1]

#Test data accuracy

resultados <- read.csv ("regr\_test\_alumnos.csv")

tablatst <- cbind(resultados [,2:dim(resultados)[2]])

colnames(tablatst) <- names(resultados)[2:dim(resultados)[2]]

rownames (tablatst) <- resultados [,1]

#Wilcoxon test

#Need to normalize the error in regression, here we normalize for comparison between Linear regression and K-NN

difs <- (tablatra[,2] - tablatra[,1]) / tablatra[,2]

#We make a table containing two columns depending on the normalized difference above. If the difference is negative, we assign the absolute value

#of the difference to the comparison algorithm +0.1, in this case K-NN, or 0+0.1 if it's positive. Then we do the reverse for the algorithm we took as

#base for the comparison, if it's positive we take the difference+0.1 and 0+0.1 otherwise. The 0.1 is due to wilcoxon calculations below

wilc\_1\_2 <- cbind(ifelse(difs <0, abs( difs)+0.1, 0+0.1), ifelse(difs>0, abs(difs)+0.1, 0+0.1))

colnames(wilc\_1\_2) <- c( colnames(tablatra)[2], colnames(tablatra)[1])

wilc\_1\_2

#The actual wilcoxon test, where we define what algorithms we want to compare

LMvsKNNtst <-wilcox.test(wilc\_1\_2[,1], wilc\_1\_2[,2], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic #This is the value for Linear regression model

pvalue <- LMvsKNNtst$p.value #This is the p-value of the comparison

LMvsKNNtst <- wilcox.test(wilc\_1\_2[,2], wilc\_1\_2[,1], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic #This is the value for K-nn model

Rmas #168

Rmenos #3

pvalue #3.814697e-05

#1-pvalue gives us an extremely high confidence value that they perform differently, not to mention the distance between R+ and R-

#Same Wilcoxon test but applied over test data instead of train data and this time with K-nn as main comparison algorithm

difs <- (tablatst[,1] - tablatst[,2]) / tablatst[,1]

wilc\_1\_2 <- cbind(ifelse (difs <0, abs( difs)+0.1, 0+0.1), ifelse (difs >0, abs( difs)+0.1, 0+0.1))

colnames(wilc\_1\_2) <- c( colnames(tablatst)[1], colnames(tablatst)[2])

wilc\_1\_2

LMvsKNNtst <-wilcox.test(wilc\_1\_2[,1], wilc\_1\_2[,2], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic

pvalue <- LMvsKNNtst$p.value

LMvsKNNtst <- wilcox.test(wilc\_1\_2[,2], wilc\_1\_2[,1], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic

Rmas #78

Rmenos #93

pvalue #0.7660294

#This time we get a 0.234\*100%= 23.4% confidence that they are different, so we can conclude there isn't any major difference

#Friedman training data algorithm comparison

test\_friedman <- friedman.test(as.matrix(tablatra))

test\_friedman #chi-squared = 20.333, p-value=3.843e-05, significantly inferior to 0.05 which shows at least one algorithm

#is very different from another

#Friedman test data algorithm comparison

test\_friedman <- friedman.test(as.matrix(tablatst))

test\_friedman #chi-squared = 8.4444, p-value=0.01467, inferior to 0.05 which shows at least one algorithm is very different from another

#Holm training data algorithm comparison

#Prepare the groups

tam <- dim(tablatra)

groups <- rep(1:tam[2], each=tam[1])

#Do the test

pairwise.wilcox.test(as.matrix(tablatra), groups, p.adjust = "holm", paired = TRUE)

# 1 2

# 2 0.0031 -

# 3 0.0032 0.0032

# All are very similarly different, no noticeable advantages

#Holm test data algorithm comparison

#Prepare the groups

tam <- dim( tablatst)

groups <- rep(1:tam[2], each=tam[1])

#Do the test

pairwise.wilcox.test(as.matrix(tablatst), groups, p.adjust = "holm", paired = TRUE)

# 1 2

#2 0.580 -

#3 0.081 0.108

#M5 has a distinct advantage over both K-nn and Lm with ~90% confidence, meanwhile K-nn and Lm are very similar

# Apéndice: Código R de Vowel completo

#Data entry

library(readr)

vowel <- read\_csv("vowel/vowel.dat", col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

vowel <- as.data.frame(vowel)

names\_vowel<-scan("vowel/vowel.dat",what=character(),sep=" ",nlines=15)

names(vowel)<-names\_vowel[seq(4,69,by=5)]

attach(vowel)

#EDA

summary(vowel) #cuartile, min, max, mean, median

apply(vowel[1:13],2,sd) #Standard deviation of the regressors

library(moments) #library necessary for skewness and kurtosis

apply(vowel[1:13],2,skewness) #skewness of the data, basically how condensed and with peaks it is

apply(vowel[1:13],2,kurtosis) #how assymetrical the data is

apply(vowel[1:13],2,shapiro.test) #normalization test

#Visual analysis

hist(vowel[[1]],breaks=2,main="Histogram of TT",xlab="TT") #histogram to check distribution

#Function to draw multiple plots at the same time using the grid library

multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {

library(grid)

plots <- c(list(...), plotlist)

numPlots = length(plots)

if (is.null(layout)) {

layout <- matrix(seq(1, cols \* ceiling(numPlots/cols)),

ncol = cols, nrow = ceiling(numPlots/cols))

}

if (numPlots==1) {

print(plots[[1]])

} else {

grid.newpage()

pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

for (i in 1:numPlots) {

matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,

layout.pos.col = matchidx$col))

}

}

}

#Multiple bar plots of the categorical variables

library(ggplot2)

p1 <- ggplot(vowel, aes(x=vowel[[1]])) +

geom\_bar() +

labs(title="Barplot of TT",x="TT") +

theme\_bw()

p2 <- ggplot(vowel, aes(x=vowel[[2]])) +

geom\_bar() +

labs(title="Barplot of Speaker Number",x="Speaker number")+

theme\_bw()

p3 <- ggplot(vowel, aes(x=vowel[[3]])) +

geom\_bar() +

labs(title="Barplot of Sex",x="Sex") +

theme\_bw()

p4 <- ggplot(vowel, aes(x=vowel[[8]])) +

geom\_histogram() +

labs(title="Histogram of F5",x="F5") +

theme\_bw()

multiplot(p1,p2,p3,p4, cols=2)

#Box plot of the numerical continuous variables

boxplot(vowel[4:13])

#Normal distribution plots of half of the continuous variables

qqnorm(vowel[["F0"]],main="Q-Q plot of F0")

qqline(vowel[["F0"]])

qqnorm(vowel[["F1"]],main="Q-Q plot of F1")

qqline(vowel[["F1"]])

qqnorm(vowel[["F2"]],main="Q-Q plot of F2")

qqline(vowel[["F2"]])

qqnorm(vowel[["F3"]],main="Q-Q plot of F3")

qqline(vowel[["F3"]])

qqnorm(vowel[["F4"]],main="Q-Q plot of F4")

qqline(vowel[["F4"]])

#Cross tabulation for contingency table for relation between categorical variables

library(gmodels)

CrossTable(vowel[["Class"]], vowel[["TT"]], prop.t=TRUE, prop.r=TRUE,prop.c=TRUE)

CrossTable(vowel[["Class"]], vowel[["SpeakerNumber"]], prop.t=TRUE, prop.r=TRUE,prop.c=TRUE)

CrossTable(vowel[["Class"]], vowel[["Sex"]], prop.t=TRUE, prop.r=TRUE,prop.c=TRUE)

#Bar plot of all numerical continuous variables against Class to determine distribution and correlation visually

library(tidyr)

vowel %>%

gather(-Class, -Sex, -TT, -SpeakerNumber, key = "var", value = "value") %>%

ggplot(aes(x = Class, y = value)) +

geom\_col() +

stat\_smooth() +

facet\_wrap(~ var, scales = "free") +

theme\_bw()

# C.1

# Utilizar el algoritmo k-NN probando con diferentes valores de k.

# Elegir el que considere más adecuado para su conjunto de datos.

# Analice que ocurre en los valores de precisión en training

# y test con los diferentes valores de k.

#Preprocessing needed to scale the continuous variables

table(vowel$Class)

vowel\_n <- as.data.frame(lapply(vowel[,4:13], scale, center = TRUE, scale = TRUE))

boxplot(vowel\_n) #checking if it worked as intended

plot(vowel[[14]],vowel\_n[[1]])

#Knn

#load libraries and prepare file location

library(class)

library(caret)

nombre<- "vowel/vowel"

#Do knn with only training data

run\_knn\_fold\_train <- function(i,x,k) {

file<- paste(x,"-10-",i,"tra.dat",sep ="")

x\_tra<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

In<- length(names( x\_tra))-1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

#Normalize data and drop the three categorical variables with no contribution for knn algorithm

x\_tra\_n <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

#Do knn only on training data

knn\_test\_pred <- knn(train = x\_tra\_n, test = x\_tra\_n, cl = x\_tra$Y, k=k)

postResample(pred = knn\_test\_pred, obs = x\_tra$Y)[1] #Accuracy column

}

#helper function to simply doing 10 cv with 10 different k values for knn

run\_knn\_fold\_train\_different\_k <- function(k){

knnMSEtrain<-mean(sapply(1:10,run\_knn\_fold\_train,nombre,k))

knnMSEtrain

}

train\_values<-sapply(1:10,run\_knn\_fold\_train\_different\_k)

train\_values #Knn values with different K using CV, most accurate is K=1, but it's likely to be overfitting

#[1] 1.0000000 0.9950617 0.9930415 0.9810325 0.9792368 0.9652076 0.9626263

#[8] 0.9424242 0.9380471 0.9139169

#Proper Knn with both train and test data

run\_knn\_fold\_test <- function(i,x,k) {

file<- paste(x,"-10-",i,"tra.dat",sep ="")

x\_tra<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

file<- paste(x,"-10-",i,"tst.dat",sep ="")

x\_tst<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

In<- length(names( x\_tra))-1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X",1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

#Normalize data and drop the three categorical variables with no contribution for knn algorithm

x\_tra\_n <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

x\_tst\_n <- as.data.frame(lapply(x\_tst[,4:13], scale, center = TRUE, scale = TRUE))

#Do Knn and then compare with test data

knn\_test\_pred <- knn(train = x\_tra\_n, test = x\_tst\_n, cl = x\_tra$Y, k=k)

postResample(pred = knn\_test\_pred, obs = x\_tst$Y)[1] #Accuracy column

}

#Helper function

run\_knn\_fold\_test\_different\_k <- function(k,type){

knnMSEtest<-sapply(1:10,run\_knn\_fold\_test,nombre,k)

knnMSEtest

}

#Obtaining all KNN models accuracy for K between 1 and 10 for each partition

test\_values<-sapply(1:10,run\_knn\_fold\_test\_different\_k)

#Obtain the mean and standard deviation for each tested K

knn\_means<-apply(test\_values,2,mean)

knn\_sd<-apply(test\_values,2,sd)

knn\_means

knn\_sd

#Save 5 KNN models with K=1,3,5,7,10 for comparison with other models for each test

tests\_knn<-test\_values[,c(1,3,5,7,10)]

tests\_knn

#For each K in knn from 1 to 10 we get the following accuracy

#0.9909091 0.9717172 0.9696970 0.9414141

#0.9373737 0.9272727 0.9131313 0.8888889

#0.8777778 0.8363636

#And the following standard deviation

#0.01004474 0.01831848 0.02233417

#0.02064607 0.03222488 0.02418624

#0.03305842 0.02735366 0.05225910

#0.05233497

plot(knn\_means, type="l", xlab="K", ylab="accuracy")

#Knn using Caret for double confirmation

knnFit <- train(vowel[,4:13], vowel[,14],

method = "knn",

preProcess = c("center", "scale"),

tuneLength = 10,

trControl = trainControl(method = "cv")

)

knnFit

#K tested from 5 to 23, best model is K=5 with an approximate accuracy of 94.2%

# C.2

# Utilizar el algoritmo LDA para clasificar. No olvide comprobar las asunciones.

#LDA

library(MASS)

#LDA assumptions, normality, randomness, samples 5 to 10x more than predictors, common variance

# normality

apply(vowel[4:13],2,shapiro.test) #fail, need to normalize

# Assuming observations are random and uncorrelated...

# 5 to 10 more samples than predictors

dim(vowel\_n) #99 times more, 10 times more after being separated into folds

# Predictors have a common variance

var(vowel\_n)

# Linear Discriminant Analysis by hand using 10 fold cross validation

run\_lda\_fold <- function(i,x) {

#Load train and test data of the current fold

file<- paste(x,"-10-",i,"tra.dat",sep ="")

x\_tra<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

file<- paste(x,"-10-",i,"tst.dat",sep ="")

x\_tst<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

In<- length(names( x\_tra))-1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X",1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

#Normalize data

x\_tra[4:13] <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

x\_tst[4:13] <- as.data.frame(lapply(x\_tst[,4:13], scale, center = TRUE, scale = TRUE))

#drop predictors with 0 variance and near 0 correlation

x\_tra<-x\_tra[-c(1,2,3)]

#Do the algorithm calculations

lda.fit <- lda(Y~.-Y,data=x\_tra)

#Predicted values against test data

ld.pred<-predict(lda.fit,x\_tst)

#Return the mean of how many predicted values are correct as an accuracy metric

mean(ld.pred$class==x\_tst$Y)

}

lda\_cv<-sapply(1:10,run\_lda\_fold,nombre) #Obtain the mean accuracy on each fold

mean(lda\_cv) #average them to obtain the final accuracy of the algorithm

#0.6171717

sd(lda\_cv) #standard deviaton to determine error

#0.04108572

# Linear Discriminant Analysis with caret for double confirmation

ldaFit <- train(vowel[,4:13], vowel[,14],

method = "lda",

preProcess = c("center", "scale"),

tuneLength = 10,

trControl = trainControl(method = "cv"))

ldaFit

#0.5959596

# C.3

# Utilizar el algoritmo QDA para clasificar. No olvide comprobar las asunciones.

#QDA

# Check same variance but this time for each class

var\_l<-vector("numeric",11)

for(x in 1:11){

v\_0<-sapply(vowel[vowel$Class == as.character(x-1),][4:13], var)

var\_l[x]<-mean(v\_0)

}

var\_l

# Quadratic Discriminant Analysis by hand using 10 fold cross validation

run\_qda\_fold <- function(i,x) {

#Load train and test data of the current fold

file<- paste(x,"-10-",i,"tra.dat",sep ="")

x\_tra<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

file<- paste(x,"-10-",i,"tst.dat",sep ="")

x\_tst<-read\_csv(file, col\_names = FALSE, col\_types = cols(X1 = col\_integer(), X14 = col\_factor(levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")), X2 = col\_integer(), X3 = col\_integer()), skip = 18)

In<- length(names( x\_tra))-1

names(x\_tra)[1:In] <- paste("X",1:In,sep ="")

names(x\_tra)[In+1] <- "Y"

names(x\_tst)[1:In] <- paste("X",1:In, sep ="")

names(x\_tst)[In+1] <- "Y"

#normalize them

x\_tra[4:13] <- as.data.frame(lapply(x\_tra[,4:13], scale, center = TRUE, scale = TRUE))

x\_tst[4:13] <- as.data.frame(lapply(x\_tst[,4:13], scale, center = TRUE, scale = TRUE))

#Do the algorithm calculations

qda.fit <-qda(Y~.-Y, data=x\_tra)

#Test

qda.pred <- predict(qda.fit,x\_tst)

#Return the mean of the test as an accuracy metric

mean(qda.pred$class==x\_tst$Y)

}

qda\_cv<-sapply(1:10,run\_qda\_fold,nombre) #Obtain the mean accuracy on each fold

mean(qda\_cv) #average them to obtain the final accuracy of the algorithm

#0.9111111

sd(qda\_cv) #standard deviaton to determine error

#0.03393833

#Quadratic Discriminant Analysis with caret for double confirmation

qdaFit <- train(vowel[,1:13], vowel$Class,

method = "qda",

preProcess = c("center", "scale"),

tuneLength = 10,

trControl = trainControl(method = "cv"))

qdaFit

#0.9171717

# C.4

# Comparar los resultados de los tres algoritmos.

tests\_knn

lda\_cv

qda\_cv

comparison\_table<-cbind(tests\_knn,lda\_cv,qda\_cv)

colnames(comparison\_table)<-c("KNN 1", "KNN 3", "KNN 5", "KNN 7", "KNN 10","LDA","QDA")

rownames(comparison\_table)<-paste("Test ",1:10)

comparison\_table

#Wilcoxon test

#No need to scale the error in classification as all belong in the range 0~1.

#Best KNN with K=1 against LDA

LMvsKNNtst <-wilcox.test(comparison\_table[,1], comparison\_table[,6], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic #This is the value for Linear regression model

pvalue <- LMvsKNNtst$p.value #This is the p-value of the comparison

LMvsKNNtst <- wilcox.test(comparison\_table[,6], comparison\_table[,1], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic #This is the value for K-nn model

Rmas #55

Rmenos #0

pvalue #0.005857099

#We can assess with total confidence that they are indeed different

#Best KNN with K=1 against QDA

LMvsKNNtst <-wilcox.test(comparison\_table[,1], comparison\_table[,7], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic #This is the value for Linear regression model

pvalue <- LMvsKNNtst$p.value #This is the p-value of the comparison

LMvsKNNtst <- wilcox.test(comparison\_table[,7], comparison\_table[,1], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic #This is the value for K-nn model

Rmas #55

Rmenos #0

pvalue #0.005857099

#We can assess with total confidence that they are indeed different, it's surprising that the numbers are identical,

#but just goes to show how much Knn with K=1 dominates the comparison

#Best KNN with LDA against QDA

LMvsKNNtst <-wilcox.test(comparison\_table[,6], comparison\_table[,7], alternative = "two.sided", paired=TRUE)

Rmas <- LMvsKNNtst$statistic #This is the value for Linear regression model

pvalue <- LMvsKNNtst$p.value #This is the p-value of the comparison

LMvsKNNtst <- wilcox.test(comparison\_table[,7], comparison\_table[,6], alternative = "two.sided", paired=TRUE)

Rmenos <- LMvsKNNtst$statistic #This is the value for K-nn model

Rmas #0

Rmenos #55

pvalue #0.005825024

#Similar to the cases above, we are very confident that they perform differently.

#Friedman training data algorithm comparison

test\_friedman <- friedman.test(as.matrix(comparison\_table))

test\_friedman #chi-squared = 55.211, p-value=4.203e-10, significantly inferior to 0.05 which shows at least one algorithm

#is very different from another

#Holm algorithm comparison

#Prepare the groups

tam <- dim(comparison\_table)

groups <- rep(1:tam[2], each=tam[1])

#Do the test

pairwise.wilcox.test(as.matrix(comparison\_table), groups, p.adjust = "holm", paired = TRUE)

# 1 2 3 4 5 6

# 2 0.104 - - - - -

# 3 0.104 0.104 - - - -

# 4 0.104 0.104 0.104 - - -

# 5 0.041 0.104 0.104 0.104 - -

# 6 0.104 0.104 0.041 0.041 0.104 -

# 7 0.104 0.104 0.104 0.905 0.104 0.104

# 1-5 3-6 4-6, so K-nn with K=1 and K=10, and LDA with Knn k=5 and K = 7 are different.

apply(comparison\_table,2,mean)