Universidad Complutense de Madrid

Trabajo final de grado

Bioinformática: Inteligencia Artificial para profundizar en el conocimiento de trastornos metabólicos complejos.

Doble Grado en Ingeniería Informática y Matemáticas

Mateo Rodríguez Lavado

Dirigido por

Dra. María Guijarro Mata-García
Facultad de Informática
Departamento de Arquitectura de computadores y Automática

Dra. María Insenser Nieto
Investigadora del CIBERDEM (Centro de Investigación Biomédica en Red de Diabetes y Enfermedades Metabólicas asociadas)
Hospital Ramón y Cajal

Resumen

El síndrome de ovario poliquístico (SOP), es un desorden endocrino metabólico complejo de etiología heterogénea y poco conocida. Este síndrome definido como la combinación de disfunción ovulatoria, exceso de andrógenos y ovarios poliquísticos, es una causa importante de disfunción ovulatoria, de irregularidades menstruales, infertilidad, hiperandrogenismo clínico y disfunción metabólica en mujeres en edad fértil. Existen numerosas investigaciones relacionadas con este campo, entre las que destacan las de Ricardo Azziz. En este trabajo, se propone crear una aplicación en MATLAB que ayude a identificar y estudiar los factores más importantes implicados en su fisiopatología y su relación con la respuesta a los diferentes macronutrientes de la dieta (glucosa, lípidos y proteínas) a partir de los datos obtenidos de un proyecto de investigación. Este programa permitirá hacer un estudio detallado de cada una de las variables, centrándonos primero en un análisis descriptivo para más tarde, implementar distintos métodos de selección de variables, como son la regresión logística y el método Lasso. En nuestro caso, no hace falta buscar una muestra de individuos para realizar el trabajo ya que nos la proporciona el grupo de investigación del Centro de Investigación Biomédica en Red de Diabetes y Enfermedades Metabólicas Asociadas (CIBERDEM) del departamento de Endocrinología y Nutrición del hospital Ramón y Cajal quien ayudará con las conclusiones de la investigación.

Palabras clave: SOP, regresión logística, Lasso, MATLAB.

Abstract

The polycystic ovary syndrome (PCOS) is a not well know complex metabolic endocrine disorder of heterogeneous etiology. This syndrome, defined as the combination of ovulatory dysfunction, excess androgens and polycystic ovaries, is an important cause of ovulatory dysfunction, menstrual irregularities, infertility, clinical hyperandrogenism and metabolic dysfunction in women of childbearing age. There are several medical researches related to this field, most of then, published by Ricardo Azziz. In this project, we have developed an application in MATLAB that helps to identify and study the most important factors involved in its pathophysiology and its relationship with the response to the different macronutrients of the diet (glucose, lipids and proteins) through the data obtained from a research project. This application will allow an specific study of each variable, first focused on a descriptive analysis to later implement various methods of variable selection, such as logistic regression and the lasso method. In our case, the sample data needed to complete the study has been provided by CIBERDEM's research group of the department of Endocrinology and Nutrition of the Ramón y Cajal Hospital, which will help us with the conclusions of the research.

Keywords: PCOS, logistic regression, lasso, MATLAB.

Agradecimientos

En primer lugar, quiero dar las gracias a toda mi familia: a mis padres, a mis hermanos y a mi tío Mateo, por el apoyo incondicional que he recibido durante todos mis estudios. Especialmente a mi hermano Miguel por sus muchos consejos sobre cómo enfocar este trabajo y su ayudada a superar algunas de las dificultades que me ha planteado LaTeX.

A mi tutora, María Guijarro, por su paciencia y disponibilidad. Sin sus consejos, especialmente en lo que se refiere al enfoque estadístico, este trabajo me hubiera resultado mucho más complejo.

A la doctora María Inserser y el resto de profesionales del departamento de endocrinología y nutrición del hospital Ramón y Cajal de Madrid por haberme dado la oportunidad de descubrir las aplicaciones de la bioinformática.

A mi hermana María y mi amiga Ángela por sus explicaciones sobre cuestiones médicas.

Por último, no puedo olvidar a mis compañeros de Innova -la empresa en la que trabajo actualmente- por sus valiosas sugerencias sobre la comprensión de los algoritmos.

Índice general

Índice de figuras

Índice de tablas

Lista de variables

β	Parámetro de influencia de las variables predictoras
$\hat{Y}$	Predicción mediante un modelo
λ	Parámetro de ajuste del método Lasso
$\overline{d}$	Diferencia de medias
ρ	Coeficiente de correlación de Pearson
σ_i	Desviación típica de la i-esima variable de W
$\sigma_{X,Y}$	Covarianza de (X,Y)
θ	Parámetro de estimación
$E[\ ]$	Esperanza Matemática
H_0	Hipótesis nula
H_1	Hipótesis alternativa
$L(\ )$	Función de máxima verosimilitud
n	Número de observaciones de nuestro conjunto de datos
$P(\ )$	Probabilidad de una variable
s	Parámetro de restricción para el método Lasso
T	Estimador
u	Término de perturbación de una regresión
V_i	Cada uno de las variables de nuestro conjunto de variables sin normalizar
v_{i_j}	Cada uno de los valores que toma una variable no normalizada
V	Conjunto de todas las variables antes de normalizar
W_i	Cada uno de las variables de nuestro conjunto de variables normalizado

w_{i_j}	Cada uno de los valores que toma una variable normalizada
W	Conjunto de todas las variables después de normalizar
X_i	Cada una de las variables del conjunto de variables predictora
X	Conjunto de variables predictoras
Y	Variable dependiente

Capítulo 1

Introducción.

1.1. Introducción

Con el avance de la tecnología en las últimas décadas se han podido desarrollar métodos más complejos que permiten analizar la información de manera más exhaustiva obteniendo así mejores resultados. La importancia del big data y la inferencia estadística están actualmente reconocidas en la investigación. La medicina es uno de los campos en los que queda reflejado el uso de las nuevas tecnologías. Además, permiten conocer los mecanismos fisiopatológicos involucrados en las enfermedades. En este trabajo nos hemos centrado en el estudio del síndrome del ovario poliquístico (SOP).

Uno de los principales inconvenientes que en la actualidad presentan los estudios en medicina es la alta dimensionalidad de los conjuntos a analizar, debido a la gran cantidad de parámetros que se obtienen en distintas determinaciones analíticas. Esto mismo, es lo que le pasa al proyecto realizado en el hospital Ramón y Cajal llamado "respuesta hormonal, metabólica, inflamatoria y oxidativa a los diferentes macronutrientes de la dieta: influencia de los esteroides sexuales", con el que se pretende estudiar el efecto de los macronutrientes en mujeres con SOP, un síndrome relativamente común entre las mujeres y del que no se conoce mucho sobre su mecanismo fisiopatológico. En este trabajo de fin de grado se pretende dar solución a este problema a través del uso de métodos de selección de variables que permitirán dar una explicación a nuestros datos bajando el número de variables. Entre estos métodos se encuentra la regresión logística, nacida en el siglo XIX para estudiar el crecimiento de poblaciones. Su capacidad de modelar las probabilidades ha hecho que el modelo de regresión logística sea un método popular de análisis estadístico. El otro que se estudia en mayor profundidad es el método Lasso, introducido por Robert Tibshirani en 1996 y basado en el trabajo de Leo Breiman *'Non-negative Garrote'* ([1]). Se apoya en el método de mínimos cuadrados dándole una ligera modificación.

El trabajo se estructura en cuatro grandes secciones. La primera (capítulo 2) es una des-

cripción de los datos obtenidos del proyecto PI11/00357 del grupo de investigación Diabetes, obesidad y reproducción humana (DOHR) del hospital Ramón y Cajal. En la segunda parte (capítulo 3) nos centramos en todos los aspectos matemáticos que más tarde utilizamos en nuestra aplicación. También se indican los problemas de utilizar ciertos métodos y como evolucionamos hasta los que usamos. La tercera parte (capítulo 4) es una introducción al programa, en la que se explica las funciones más importantes utilizadas y se hace una guía de nuestra aplicación. En la última parte (capítulos 5 y 6) se encuentra el proceso que se ha seguido hasta obtener los resultados y las conclusiones proporcionadas por estos mismos.

1.2. Objetivos.

Como se ha especificado antes, este trabajo de fin de grado se desarrolla en colaboración con el servicio de Endocrinología y Nutrición del hospital Ramón y Cajal. El objetivo del proyecto de CIBERDEM es averiguar si la respuesta a los macronutrientes de la dieta (glucosa, lípidos y proteínas) es diferente en mujeres sanas, que consideramos de control, mujeres con el síndrome de ovario poliquístico y hombres, y cómo afecta la obesidad a esta respuesta. Para ello, se desarrollará una aplicación que permita analizar y tratar un conjunto de variables obtenidas por el hospital. Esta aplicación permitirá ver qué variables tienen mayor importancia al intentar clasificar nuestros datos en 6 grupos: hombre obesos y no obesos, mujeres sanas obesas y no obesas y mujeres con SOP obesas y no obesas. Para averiguar esto, utilizaremos diferentes técnicas estadísticas que se detallarán más adelante.

Capítulo 2

Especificaciones

En esta sección se especificarán qué herramientas se han utilizado para realizar este trabajo. También se explicará de dónde se han obtenido los datos y qué transformaciones se han realizado con ellos.

2.1. Marco de trabajo.

El trabajo se ha realizado en MATLAB, utilizando funciones integradas en este programa. Los datos fueron entregados en SPSS, se hizo un primer análisis en WEKA y, mas tarde se trataron en Excel para importarlos posteriormente a MATLAB.

2.2. Definiciones.

A continuación expondremos varias definiciones que nos van a resultar útiles más adelante.

- **Sesgo:** El sesgo de un estimador es la diferencia entre su esperanza matemática y el valor numérico del parámetro que estima ([2]). Dada una muestra $X_1, \ldots, X_p$ y un estimador $T(X_1, \ldots, X_p)$ del parámetro θ, el sesgo es:

$$sesgo(T) = E(T) - \theta \tag{2.1}$$

- **Error cuadrático medio (ECM):** El error cuadrático medio de un estimador T, para estimar θ, se define como ([2]):

$$ECM(T) = E[(T(X_1, \ldots, X_p) - \theta)^2] \tag{2.2}$$

Otra forma de expresarlo es:

$$\begin{aligned} E[(T-\theta)^2] &= E[((T - E[T]) + (E[T] - \theta))^2] \\ &= E[(T - E(T))^2 + (E[T] - \theta)^2 \\ &= V(T) + sesgo(T)^2 \end{aligned} \tag{2.3}$$

- **Contraste de hipótesis:** "Consiste en determinar si es aceptable, partiendo de datos muestrales, que la característica o el parámetro poblacional estudiado tome un determinado valor o esté dentro de unos determinados valores"([3]). Se consideran una hipótesis nula H_0 y una alternativa H_1, y se intenta averiguar cuál de las dos es la hipótesis verdadera. En nuestro caso, el contraste que vamos a hacer, como se verá mas adelante, es:

$$\begin{cases} H_0 : \beta_j = 0 \\ H_1 : \beta_j \neq 0 \end{cases}$$

- **Error de tipo I y de tipo II:** El error de tipo I es el error que se comete cuando no se acepta la hipótesis nula (H_0) siendo esta verdadera ([4]).

  El error de tipo II es el error que se comete cuando no se rechaza la hipótesis nula siendo esta falsa.

	H_0 verdadera	H_1 verdadera
Se acepta H_0	CORRECTO	Error tipo II
Se rechaza H_0	Error tipo I	CORRECTO

- **P-valor:** el p-valor de un contraste de hipótesis se define como la probabilidad de error en que incurriríamos en caso de rechazar la hipótesis nula. En otras palabras, la probabilidad de cometer un error de tipo I. A efectos prácticos, asumiremos que si el p-valor es inferior a 0,05 rechazamos H_0 y en caso contrario se acepta ([5]).

- **ODDS:** el odds es una medida estadística que se define como la probabilidad de que suceda un evento frente a que no suceda ([6]). Es decir, si definimos $P(X)$ como la probabilidad de que suceda X entonces:

$$ODDS = \frac{P(X)}{1 - P(X)} \tag{2.4}$$

2.3. Datos de entrada.

Para poder realizar todo el modelo, se nos han proporcionado una serie de datos de pacientes reclutados en un proyecto de investigación del hospital Ramón y Cajal. Los datos

comprenden un total de 53 pacientes, de los cuales 34 son mujeres y 19 son hombres. Aunque el SOP solo lo padecen mujeres en edad fértil, en este proyecto se incluye a los hombres porque se consideran como control de niveles de testosterona muy altos y para valorar si, como ocurre en otros casos, la respuesta a macronutrientes de las mujeres con SOP se parece más a la de los varones que a la de las mujeres sin SOP.

Estos sujetos están también clasificados en función de su índice de masa corporal (IMC) en obesos y no obesos (25 sujetos obesos y 28 no obesos) ya que la obesidad juega un papel importante en el desarrollo del SOP.

Nuestros datos estarán separados por sujetos con SOP y controles, de los cuales 17 están enfermos y 17 no. En esta clasificación no se incluyen los hombres.

A parte de estas tres variables (sexo, obesidad, enfermo o no enfermo) que forman los grupos a los que vamos a dirigir nuestro análisis, el hospital Ramón y Cajal les realizó varias determinaciones a los pacientes obteniendo así 371 variables, de las cuales, la mayoría son mediciones relacionadas con la ingesta de macronutrientes en los distintos tiempos, es decir que se medía una molécula, por ejemplo, el colesterol, después de que los sujetos ingirieran glucosa, o lípidos y proteínas. Posteriormente se repetía esta medida a los 30, 60, 90 120, 180 y 240 minutos.

Los datos fueron entregados en una tabla en SPSS, pero para una mayor facilidad a la hora de manejarlos se exportaron a una tabla Excel. En esta tabla Excel se hicieron varios cambios a las variables categóricas. Por ejemplo, existía una variable llamada Group que indicaba el sexo del sujeto y si estaba enfermo o no y se representaba con un 0 para mujeres no enfermas, un 3 para mujeres enfermas y 99 para hombres (los cuales no pueden estar enfermos). Como en nuestro proyecto vamos a analizar la importancia de las variables según el sexo y si tiene SOP o no, se decidió dividir la variable GROUP en dos variables binarias, sexo y SOP, y así facilitar el trabajo.

Debido a que el objetivo del trabajo de fin de grado es hacer una comparación entre variables para ver cuáles definen mejor nuestras variables objetivo, no se podía proceder al análisis sin antes realizar una normalización de los datos. Si definimos V como el conjunto de variables sin normalizar y W el conjunto de variables normalizadas entonces para una variable $v_i \in V$, el valor normalizado $w_{i,j}$ será:

$$w_{i,j} = \frac{v_{i,j} - min\, V_i}{max\, V_i - min\, V_i} \tag{2.5}$$

La idea inicial fue que el hospital nos proporcionase cuál es el valor máximo y mínimo posible de cada variable pero para algunas de las variables es realmente difícil definir un máximo o mínimo así que se escogió el máximo y mínimo del conjunto de datos.

Capítulo 3

Metodología.

En este capítulo se va a detallar el contenido matemático que se utiliza en este trabajo. Comenzando por los conceptos más sencillos hasta otros más complejos. Hay que tener en cuenta que los resultados de todos los cálculos que se hacen dependen de la variable objetivo que seleccionemos. En este caso, nuestras variables objetivo son el sexo, la obesidad, y tener o no SOP.

3.1. Diferencia de medias.

La diferencia de medias se basa en la hipótesis de que cuanto menor sean las medias más se parecen los datos de los dos grupos. En nuestro caso, tal y como se ha mencionado anteriormente, cogemos como grupos el sexo, la obesidad o si tienen la enfermedad. Se calculan las medias de ambos grupos para una variable y se restan. Si el resultado se acerca a cero suponemos que la variable en cuestión no diferencia entre un grupo y otro.

Si definimos $\overline{W_i} = \frac{1}{n}\sum_{k=1}^{n} w_{i_k}$ como la media de la variable $W_i \in W$ y $\overline{W_j} = \frac{1}{n}\sum_{k=1}^{n} w_{j_k}$ como la media de la variable W_j con $W_j \in W$ y $j \neq i$, entonces la diferencia de medias sería:

$$\overline{d} = |\overline{W_i} - \overline{W_j}|$$

3.2. Desviación típica.

La desviación típica o estándar es una medida de dispersión para variables de razón y de intervalo. Es una medida del grado de dispersión de los datos con respecto al valor promedio.

Esto, junto a la diferencia de medias nos sirve para ver cuánto de concentrados están los datos y si hay una clara diferenciación de los valores de una variable en los diferentes grupos.

Si definimos $W_i \in W$ como la variable de la que queremos calcular la desviación típica y $\overline{W_i}$ como la media de esa variable, calculamos la desviación típica de la siguiente forma:

$$\sigma_i = \sqrt{\frac{1}{n}\sum_{k=1}^{n} w_{i_k} - \overline{W_i})}$$

Pero existe un problema dentro de este análisis. Este aparece cuando los valores que tenemos están concentrados en un punto de la distribución excepto unos pocos. Estos valores difieren mucho de los valores "normales"que tenemos de esta variable y se denominan valores atípicos (en inglés outlier). Por lo tanto, como no es suficiente para sacar conclusiones, investigamos otras técnicas que se detallarán mas adelante.

3.3. Modelos de análisis de regresión.

El análisis de regresión lineal es una técnica estadística que estudia la relación entre varias variables. La regresión permite conocer que efecto causa una variable sobre otra. También puede predecir valores de una variable a partir de otra. Esto nos va a ser muy útil en nuestro trabajo. Para una definición más concreta, podemos utilizar la descrita por un trabajo de la universidad de Murcia: *"la regresión es el conjunto de técnicas usadas para explorar y cuantificar la relación de dependencia entre una variable cuantitativa llamada variable dependiente o respuesta y una o más variables independientes llamadas variables predictoras"* [7].

La forma más sencilla para expresar una regresión es a través de una ecuación lineal. Mostramos la variable dependiente en función de las variables predictoras de la forma $Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$, siendo Y la variable dependiente, X_i con $i \in 1, \ldots, p$ el conjunto de variables predictoras y β un parámetro que indica la influencia que tiene esa variable predictora en la dependiente.

El caso más simple es la recta $Y = mX + n$, donde $m, n \in \mathbb{R}$, y la llamaremos regresión lineal simple. Cuando $p > 1$ será una regresión múltiple.

3.3.1. Correlación

La correlación entre dos variables indica la fuerza y dirección de una relación lineal y de proporcionalidad entre estas. Dos variables están correlacionadas si cuando variamos los valores de una de las variables entonces varían de forma proporcional los valores de la otra. Sean dos variables X_1 y X_2, si X_1 está correlacionada con X_2 entonces X_2 también está correlacionada con X_1

La relación entre las variables finalmente se representa mediante la línea que mejor ajuste la nube de puntos que forman tales variables como se puede ver en la figura 3.1.

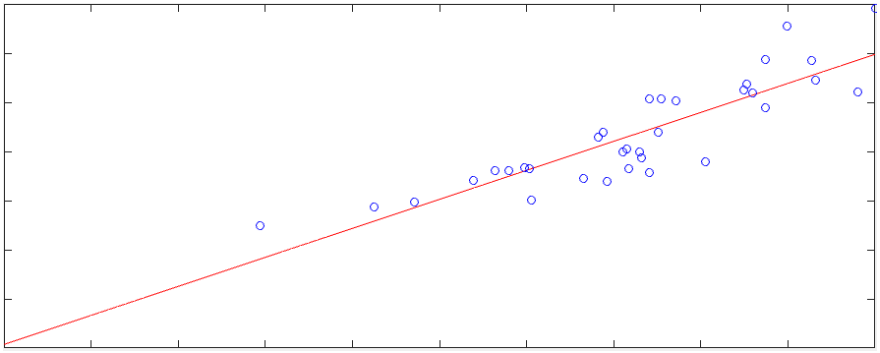
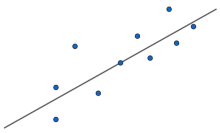
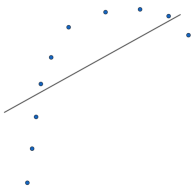


Figura 3.1: Recta de regresión

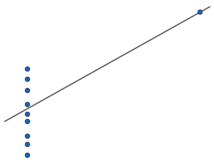
En la figura 3.2 podemos ver un ejemplo de cuatro gráficos de dispersión en los que se muestran relaciones diferentes.



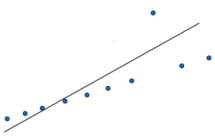
(a) Recta de regresión



(b) Recta de regresión



(c) Recta de regresión



(d) Recta de regresión

Figura 3.2: Ejemplo de distintas nubes de puntos con misma recta de regresión

Para todos los diagramas de la figura 3.2 la recta de regresión es la misma. Sin embargo, el único modelo lineal aceptable es el de la recta de la figura 3.2a. Así pues, se necesita de una

forma que permita medir el grado de asociación lineal entre dos variables. Lo que utilizamos para realizar esto son los llamados **coeficientes de correlación**.

3.3.2. Coeficientes de correlación

Para medir la relación lineal entre dos variables utilizamos el **coeficiente de correlación**. Existen distintos coeficientes de correlación. Destacamos en nuestro trabajo la correlación de Pearson que se ajusta bien si las variables son cuantitativas y están normalizadas.

Dado que hemos normalizado nuestros datos, podemos utilizar la correlación de Pearson que explicaremos a continuación. Dadas dos variables aleatorias X y Y sobre una población, el coeficiente de correlación de Pearson se define como ([8]):

$$\rho_{x,y} = \frac{\sigma_{X,Y}}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \tag{3.1}$$

Donde:

- $\sigma_{X,Y}$ es la covarianza de (X, Y).
- σ_X es la desviación típica de X.
- σ_Y es la desviación típica de Y.

El resultado se encuentra dentro de los valores del intervalo $[-1, 1]$, donde el signo indica el sentido de la relación. Según los distintos valores que tome ρ el coeficiente nos da diferentes relaciones:

- Si $\rho = 1$, existe una correlación positiva perfecta. La relación entre las dos variables es perfecta y la denominamos relación directa, es decir, cuando una de ellas aumenta, la otra también lo hace en una proporción constante.
- Si $0 < \rho < 1$, existe una correlación positiva.
- Si $\rho = 0$, no existe relación lineal. Esto no necesariamente implica que las variables sean totalmente independientes ya que pueden existir relaciones no lineales entre las dos variables.
- Si $-1 < \rho < 0$, existe una correlación negativa.
- Si $\rho = -1$, existe una correlación negativa perfecta. A este tipo de relación la denominamos relación inversa, es decir, cuando una de ellas aumenta, la otra disminuye en proporción constante.

Para hacer el análisis de correlación se utiliza el coeficiente de correlación de Pearson. Cuando se estime que la correlación entre dos variables es suficientemente alta se eliminará una de las dos ya que esto puede servir para mejorar los posteriores análisis. Esto es porque los diferentes algoritmos que vamos a utilizar predicen el funcionamiento de una variable a través de otra, dándole un peso de importancia a cada variable predictora. Si tenemos dos variables que explican lo mismo, el peso estará repartido entre ellas y por lo tanto sería menor que otras cuando esto no es cierto.

3.4. Regresión logística

El primero de los modelos que vamos a utilizar para seleccionar nuestras variables es el de regresión logística ([9], [10] y [11]).

Em primer lugar vamos a definir que es una variable binaria. Una variable binaria es aquella que sólo puede adquirir dos posibles valores, normalmente 0 o 1. Esto es importante ya que nuestra variable objetivo es binaria y los modelos que utilizaremos se basarán en esto.

Como antes se ha especificado, un modelo de regresión múltiple (que no tiene por qué ser lineal) nos permite explicar el comportamiento de una variable dependiente Y en función de una serie de variables independientes $X_1, X_2, \ldots, X_p$ y de lo que llamaremos un término de perturbación u, que indica la diferencia entre el valor real y la predicción:

$$Y = f(X_1, X_2, \ldots, X_p, u) \tag{3.2}$$

Si el modelo que tenemos es lineal, será de la forma:

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + u \tag{3.3}$$

El objetivo será ajustar nuestro modelo lo máximo posible a las observaciones que tenemos. Para ello habrá que estimar los parámetros β de los que depende nuestra regresión.

Si la variable Y es continua, se podrá utilizar un modelo de regresión lineal múltiple como el de la ecuación 3.3. Para calcular los parámetros β se utiliza el método de mínimos cuadrados (MCO). En cambio, si la variable dependiente es una variable binaria, como es nuestro caso, la regresión lineal múltiple presenta una serie de inconvenientes que imposibilitan la utilización de este método y por ello, recurriremos a la regresión logística que, como veremos más adelante, se ajustará mejor a nuestros datos.

3.4.1. Los problemas del modelo de regresión lineal.

Supongamos que tenemos una variable dependiente binaria, Y, a la que le aplicamos una regresión lineal múltiple y, por lo tanto, podemos explicarla a través de la ecuación 3.3. Como la variable es binaria, siempre se cumplirá que:

$$E[Y] = 0 \cdot P(Y = 0) + 1 \cdot P(Y = 1) = P(Y = 1) \tag{3.4}$$

donde $E[Y]$ indica la esperanza de la variable Y y P la función de probabilidad. Sabiendo que utilizamos el modelo de regresión lineal múltiple para explicar Y tenemos:

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + u \tag{3.5}$$

donde X_i con $i \in 1, \ldots, p$ es el conjunto de variables predictoras, β un parámetro que indica la influencia que tiene esa variable predictora en la dependiente y u el término de perturbación. Tomando u tal que $E[u] = 0$, y suponiendo que conocemos los valores de $X_1, X_2, \ldots, X_p$, tenemos que:

$$E[Y] = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p \tag{3.6}$$

E igualando las expresiones nos queda:

$$P(Y = 1) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p = Y - u \tag{3.7}$$

La expresión 3.7 indica que la variable binaria Y se puede expresar como la probabilidad de tener "éxito"más el término de perturbación u:

$$Y = P(Y = 1) + u = E[Y] + u \tag{3.8}$$

Sin embargo, este modelo presenta varios problemas para nuestro trabajo. Destacamos los siguientes:

- Como las variables que queremos predecir (SOP, sexo y obesidad) son binarias, es decir, sólo puede tomar valores cero y uno, al estimar los parámetros, estaremos ajustando con una recta la nube de puntos y el uso de esta recta para predecir los nuevos valores de estas variables a partir de los valores del resto de variables puede proporcionar valores

mayores que 1 o menores que 0, lo cual contradice la definición de probabilidad. Este caso se puede ver claramente en la figura 3.3.

- En una regresión lineal múltiple, la probabilidad de éxito es una combinación lineal de las variables X_i tal como se muestra en la ecuación 3.7. Si derivamos respecto X_i obtenemos:

$$\frac{\partial P(Y=1)}{\partial X_i} = \beta_i \ \forall i = 1, 2, \ldots, n$$

Luego si cambia alguna de las variables explicativas X_i, la variación que sufre $P(Y=1)$ es constante y, por tanto, independiente del valor de dicha variable explicativa, por lo que no podríamos saber qué variables explican Y que es el objetivo del trabajo.

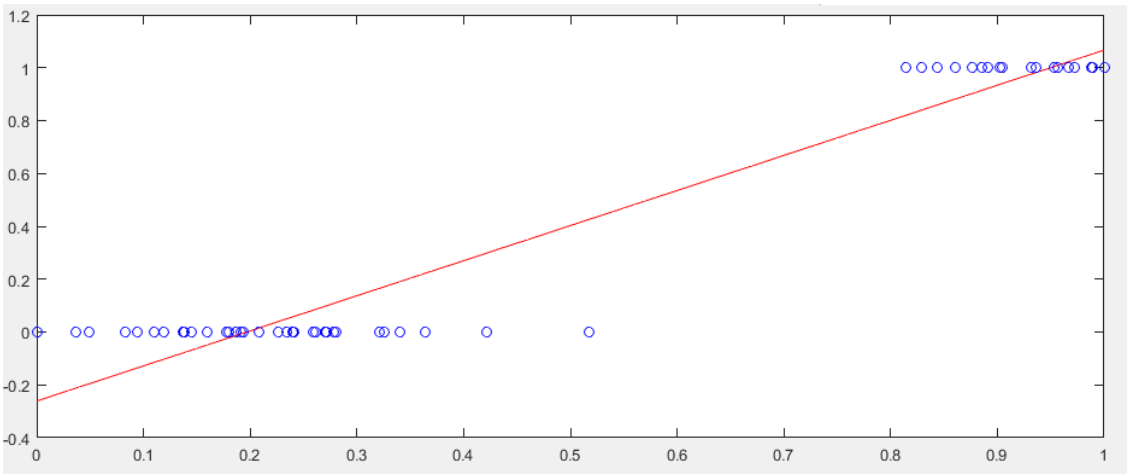


Figura 3.3: Recta de regresión para una variable dependiente binaria

3.4.2. Modelo logit.

Como se ha visto en el apartado anterior, para predecir una variable binaria Y se tienen serios problemas si se utiliza el modelo de regresión lineal. Para evitarlos, se pueden utilizar modelos no lineales como es el modelo logístico. La regresión logística se basa en modelar la probabilidad de que Y pertenezca a una categoría particular, en nuestro caso binaria. Es decir, se modela $p(X) = P(Y = 1|X)$

Para evitar los problemas anteriores debemos modelizar $p(X)$ usando una función que deba ser distinta de la función lineal (pues si no volveríamos a obtener el modelo lineal) y esté acotada por los valores cero y uno. Esta función es la función logística que viene dada por:

$$f(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}} \tag{3.9}$$

Por tanto, usando esta función, modelizaremos $p(X)$ como:

$$p(X) = \frac{e^{(\beta_0+\beta_1 X_1+\ldots+\beta_p X_p)}}{1+e^{-(\beta_0+\beta_1 X_1+\ldots+\beta_p X_p)}} \tag{3.10}$$

Si calculamos el odds ratio, descrito en el capítulo 3 en (2.4), tenemos que

$$\frac{p(X)}{1-p(X)} = e^{-(\beta_0+\beta_1 X_1+\ldots+\beta_p X_p)} \tag{3.11}$$

y aplicando logaritmos

$$ln\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p \tag{3.12}$$

La estimación de parámetros para el modelo de regresión logística es por el método de máxima verosimilitud, ya que la estimación por mínimos cuadrados no es capaz de producir estimadores insesgados de mínima varianza para los parámetros.

El método consiste en, como dice su nombre, maximizar la función de verosimilitud y encontrar los $(p+1)$ parámetros β para los cuales la probabilidad de los datos observados es la más alta.

Para hacer los cálculos más sencillos, vamos a suponer que solo tenemos dos parámetros desconocidos β_0 y β_1. Por lo tanto, bajo estas condiciones, nuestra función de máxima verosimilitud es:

$$L(\beta_0, \beta_1) = \prod_{j=1}^{n} p_j^{y_j}(1-p_j)^{1-y_j} \tag{3.13}$$

donde $p_j = \frac{e^{(\beta_0+\beta_1 x_j)}}{1+e^{-(\beta_0+\beta_1 x_j)}}$, $y_j \in [0,1]$ es el valor observado de Y en el j-esimo elemento de la muestra y x_j con $j = 1, \ldots, n$ son las observaciones de X. La estimación de los dos coeficientes requiere maximizar la función de verosimilitud, o equivalentemente, maximizar su logaritmo:

$$ln(L(\beta_0, \beta_1)) = \sum_{j=1}^{n}(y_j ln(p_j) + (1-y_j)ln(1-p_j)) \tag{3.14}$$

Derivando respecto de los dos parámetros (β_0, β_1) e igualando a cero obtenemos:

$$\sum_{j=1}^{n}(y_j - p_j) = 0 \tag{3.15}$$

$$\sum_{j=1}^{n} x_j(y_j - p_j) = 0 \tag{3.16}$$

Estas ecuaciones no son lineales y, por lo tanto, para resolverlas van a hacer falta métodos iterativos como el de Newton-Raphson (ver [12]).

3.5. Regresión LASSO.

En esta sección introduciremos las ideas del segundo método matemático que vamos a utilizar para escoger nuestras variables. Se hará un breve repaso de otras técnicas en las que se basa la regresión LASSO como es el método de mínimos cuadrados y se verán sus deficiencias con respecto al algoritmo LASSO.

3.5.1. Método de mínimos cuadrados.

El objetivo principal de un modelo de regresión es determinar los estimadores β_i a partir de una muestra dada. Mínimos cuadrados es un método que se suele utilizar para obtener estos estimadores. Se basa en intentar minimizar el error cuadrático medio ([13] y [14]).

Dado un vector de entrada $X^T = (X_1, \ldots, X_p)$ e $Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + u$, predecimos Y mediante el modelo:

$$\hat{Y} = \hat{\beta}_0 + \sum_{i=1}^{N} \hat{\beta}_i X_i \tag{3.17}$$

Para escribir la expresión anterior en forma vectorial hace falta incluir la variable constante 1 en X, e incluir también el coeficiente $\hat{\beta}_0$ en el vector de coeficientes $\hat{\beta}$ quedándonos así:

$$\hat{Y} = X^T \hat{\beta} \tag{3.18}$$

Previamente, enseñaremos dos criterios que también resuelven nuestro problema de calcular los estimadores mediante la diferencia entre el modelo poblacional y la predicción, pero que no son acertados. A la diferencia $y_i - \hat{y}_i$ la llamaremos residuo.

- EL primer método, y el más obvio, es intentar que la suma de los residuos sea lo más próxima posible a cero. Luego los β_i serían aquellos que minimizasen:

$$Min\ |\sum_{i=1}^{n} y_i - \hat{y}_i| \tag{3.19}$$

  El mayor problema de este criterio es que si tenemos residuos de igual valor pero distinto signo, se compensarían y podríamos obtener infinitas soluciones.

- El segundo criterio intenta evitar el problema de compensación del anterior y, por lo tanto, toma los valores absolutos de los residuos. En este caso se minimizaría la siguiente expresión:

$$Min\ \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{3.20}$$

  El problema de este segundo método es que el cálculo de los estimadores es bastante complicado.

Por lo tanto, para resolver estos problemas, utilizamos el método de mínimos cuadrados. Este método intenta minimizar la siguiente expresión:

$$MC(\beta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 \tag{3.21}$$

Esta expresión resuelve el problema de la compensación y, como veremos ahora, su cálculo es simple. Para hallar los estimadores de mínimos cuadrados debemos minimizar la suma de los cuadrados de los residuos. Para que resulte más sencillo, utilizaremos la notación vectorial. Luego podemos escribir

$$MC(\beta) = (y - X\beta)^T (y - X\beta) \tag{3.22}$$

donde X es una matriz $N \times p$ que tiene en cada fila un vector de entrada, e y un vector de dimensión N que contiene la salida de nuestro conjunto de entrenamiento. La expresión (3.22) se puede poner también como:

$$\begin{aligned} MC(\beta) &= (y - X\beta)^T(y - X\beta) = (y^T - \beta^T X^t)(y - X\beta) \\ &= y^T y - \beta^T X^T y - y^T X\beta + \beta X^T X\beta \\ &= y^T y - 2\beta^T X^T y + \beta^T X^T X\beta \end{aligned} \tag{3.23}$$

donde se han utilizado las propiedades de la traspuesta y el resultado $\beta^T X^T y = y^T X\beta$, que se cumple porque $\beta^T X^T y$ es un escalar 1×1 y es igual a su traspuesta.

Para obtener el mínimo respecto de β debemos igualar el vector de primeras derivadas al vector nulo.

$$\begin{aligned} \frac{\partial MC(\beta)}{\partial \beta} &= -2X^T y + X^T X\beta + \beta^T X^T X = -2X^T y + 2X^T X\beta \\ &= X^T(y - X\beta) = 0 \end{aligned} \tag{3.24}$$

Si $X^T X$ es invertible, la única solución viene dada por

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{3.25}$$

y el ajuste del i-ésimo valor X_i es $\hat{y}_i = \hat{y}_i(x_i) = x_i^T \hat{\beta}$

3.5.2. Los problemas del método de mínimos cuadrados

Esta sección se basa en un trabajo de fin de master de la universidad de la Coruña [15].

El método de mínimos cuadrados se utiliza comúnmente para estimar los parámetros β presentes en los modelos de regresión. Pero para este trabajo, en el que no todas las variables predictoras son relevantes, el método de mínimos cuadrados presenta varios problemas:

- Al estimar por mínimos cuadrados, vemos que tales estimaciones tienen bajo sesgo pero una gran varianza. Si alguno de los parámetros se ajustase a cero se podría mejorar la predicción.
- Si, como en nuestro caso, tenemos un gran numero de variables predictoras es normal que se quiera obtener un subconjunto que sea el que mejor defina nuestra variable dependiente. Para ello, necesitamos que algunas de las estimaciones sean cero, pero al estimar por mínimos cuadrados obtendremos valores próximos a cero aunque nunca exactamente cero.

- El problema de mayor importancia es el hecho de que en nuestro proyecto en particular, tenemos un mayor número de variables predictoras que de observaciones ($p > N$), y, por lo tanto, a la hora de despejar nuestros parámetros β nos encontramos con un sistema de ecuaciones indeterminado y no tendremos las ecuaciones necesarias para estimar los parámetros β.

3.5.3. Método Lasso

Para describir este método nos hemos apoyado en [1], [15], [16], [17]

El método Lasso (least absolute shrinkage and selection operator) introducido por Tibshirani en 1996 es un método que consigue ese ajuste de parámetros a cero que le pedíamos a mínimos cuadrados y, por tanto, nos permite seleccionar variables, imponiendo una restricción o una penalización sobre los coeficientes de regresión. Es un método pensado especialmente para resolver los problemas del tipo $p > N$. Este método se basa en el algoritmo de mínimos cuadrados pero añadiéndole esa penalización. Luego, la estimación de los parámetros se define como:

$$\hat{\beta}_\lambda = \underset{\beta}{argmin} \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i_j})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{3.26}$$

donde $y_i \in Y$, $x_{i_j} \in X_i$, β parámetro de influencia de las variables y λ parámetro de ajuste del método. Estadísticamente hablando, Lasso usa una penalización ℓ_1. La norma ℓ_1 de un vector de coeficientes β está dada por $||\beta||_1 = \sum|\beta_j|$. Como tratamos de minimizar, se puede observar que la penalización ℓ_1 tiene el efecto de obligar a que algunas de las estimaciones de coeficientes sean exactamente cero cuando el parámetro de ajuste λ es suficientemente grande lo que implica que seleccionar un buen valor de λ para el modelo Lasso es un problema crítico. Este problema lo trataremos más adelante cuando veamos la validación cruzada.

Si observamos la ecuación 3.26, cuando $\lambda = 0$, entonces el modelo Lasso simplemente da el ajuste de mínimos cuadrados, y cuando λ llega a ser suficientemente grande, el modelo Lasso da el modelo nulo en el cual todas las estimaciones de coeficientes son iguales a cero. Sin embargo, entre estos dos extremos, dependiendo del valor de λ, la regresión Lasso puede producir un modelo que implique cualquier número de variables.

Otra forma de expresar (3.26) es:

$$\underset{\beta}{minimizar}\left\{ \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i_j})^2 \right\} \quad sujeto\ a\ \sum_{j=1}^{p} |\beta_j| \leq s \tag{3.27}$$

donde s es el valor de la restricción. Es decir, para cada valor de λ hay algunos valores de s para los que la ecuación (3.27) dará los mismos coeficientes que (3.26). Cuando $p = 2$, entonces (3.27) indica que las estimaciones de los coeficientes del modelo Lasso tienen el error cuadrático medio más pequeño de todos los puntos que están dentro del rombo definido por $|\beta_1| + |\beta_2| \leq s$.

En la figura 3.4 se puede ver la representación gráfica de la estimación de los β donde el área azul sólida es la región de restricción $|\beta_1| + |\beta_2| \leq s$, mientras que las elipses rojas son los contornos de las soluciones a mínimos cuadrados.

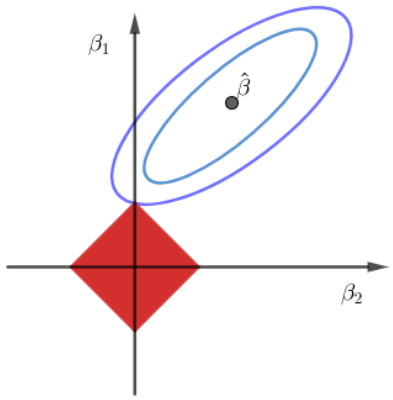


Figura 3.4: Representación gráfica de la estimación de los β para el método Lasso.

3.5.4. Validación cruzada de k iteraciones

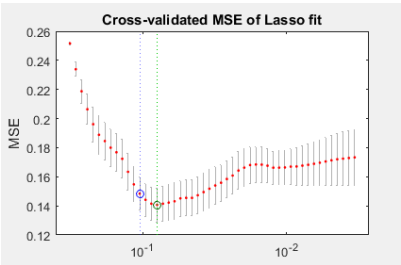
Como se ha especificado antes, se requiere de un método que permita seleccionar cuál el mejor valor para el parámetro de ajuste λ en (3.26), o equivalentemente, cuál es el mejor valor para la restricción s en (3.27).

La validación cruzada (cross-validation) proporciona una forma sencilla de abordar este problema. Hay muchos tipos de validación cruzada, en nuestro caso explicaremos la validación cruzada de k iteraciones. Este método consiste en dividir nuestro conjunto de datos en k grupos de tal manera que el primer grupo se trata como un conjunto de validación, y el método se ajusta a lo $k - 1$ grupos restantes. Después se calcula el error cuadrático medio, ECM, sobre las observaciones de nuestro conjunto de validación. Este proceso se repite k veces, cada iteración escoge un conjunto diferente para validar. Obtendremos entonces k estimaciones del error $ECM_1, ECM_2, \ldots, ECM_k$. La estimación de la validación cruzada de k iteraciones se calcula promediando estos valores,

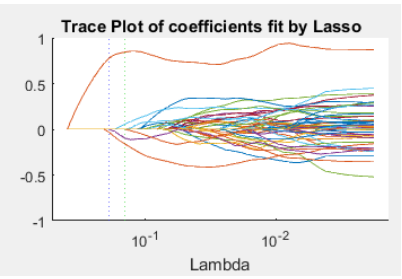
$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} ECM_i \tag{3.28}$$

Así pues, para calcular el mejor valor de λ se elegirá un conjunto de valores y se calculará el error de la validación cruzada para cada valor de λ, como se describe en (3.28). A continuación, seleccionamos el valor del parámetro de ajuste para el cual el error de la validación cruzada es menor.

En la figura 3.5a podemos observar el cálculo de los ECM según el valor de λ. Las líneas verticales punteadas verdes representan el ajuste de Lasso para el cual, el error de validación cruzada es menor. Con nuestros datos vemos que el mejor valor para λ es poco menor que 10^{-1}. En la figura 3.5b vemos como quedaría la selección de variables según el λ escogido. En este gráfico se observa perfectamente como, a medida que lambda es mayor, la cantidad de variables cuyo parámetro es distinto de cero es menor.



(a) Validación cruzada de diez iteraciones para Lasso



(b) Estimaciones correspondientes del coeficiente de Lasso

Figura 3.5: Ejemplo de solución del método Lasso con SOP como variable objetivo.

3.6. Otros métodos

Los siguientes métodos aquí expuestos se estudiaron como otra alternativa para realizar esta selección de variables pero fueron descartados al ver que o bien sus resultados eran difíciles de analizar o bien nuestros datos imposibilitaban la aplicación de los mismos. Como referencia a estos métodos hemos usado [18], [17] y [19].

3.6.1. Best First Search

Para explicar este algoritmo hace falta saber qué es una función heurística. Una función heurística es aquella que se utiliza para asignar un valor a cada uno de los estados de un problema permitiendo así "predecir" el mejor camino a tomar.

El algoritmo Best First Search, utilizado sobre todo en teoría de grafos, consiste en encontrar la variable mas prometedora sirviéndose de una función heurística para ello. Este algoritmo es una mezcla de la búsqueda en profundidad y la búsqueda en anchura siendo mas rápido que estos ya que no necesita expandir todo el árbol.

El algoritmo funciona de la siguiente manera:

- Expandir nodo inicial.
- Seleccionar el nodo con menor heurística. Si es final devolver el camino hasta llegar al nodo.
- Si no, volvemos a expandir el nodo y procedemos igual que el punto anterior.

3.6.2. Análisis de componentes principales

Se ha visto ya que el objetivo de este trabajo de fin de grado es reducir el número de variables y eso es precisamente lo que hace un análisis de componentes principales, mediante técnicas estadísticas reduce la dimensionalidad de nuestro conjunto de datos perdiendo la menor información posible. Esto lo consigue creando unas nuevas componentes que son una combinación lineal de algunas de las variables.

Para que se pueda realizar un correcto análisis de componentes principales se tiene que comprobar que las variables tienen correlación entre ellas ya que indica que la información está repetida. Cada componente principal esta formada por una serie de factores donde el primero debe recogerá la mayor información posible, el segundo recogerá la mayor información posible que el primer factor no haya recogido y así sucesivamente.

Luego cada componente quedaría expresada de la siguiente forma:

$$C_i = a_{1i} * W1 + a_{2i}W_2 + \ldots + a_{pi}Wp$$

donde $a \in [0, 1]$, $W_i \in W$ con $i = 1 \ldots p$ donde p representa el número de variables que hay en W.

3.6.3. Random forest

Random forest es un algoritmo que, con resultados bastante buenos, puede realizar clasificaciones o regresiones de un conjunto de datos. El algoritmo de random forest sigue los siguientes pasos:

- Se selecciona de forma aleatoria el 60 % de las variables aproximadamente y con ellas se crea un árbol de decisión.
- Este proceso se repite hasta obtener k árboles. Cada uno de estos árboles es completamente independiente de los demás.
- Finalmente, la selección de variables se hará promediando los resultados de cada uno de los árboles de decisión.

Este tipo de algoritmo es bastante utilizado cuando tenemos un gran conjunto de datos ya que permite paralelizar la creación de cada árbol por ser independientes los unos de los otros.

Capítulo 4

Desarrollo de la aplicación.

Dada la complicación de los métodos expuestos anteriormente sumado a la cantidad de datos que hay que procesar se ha visto necesario desarrollar una aplicación que permita realizar la selección de variables necesaria para resolver nuestro problema. Esta aplicación se ha desarrollado en MATLAB debido a la gran eficiencia que tiene a la hora de hacer cálculos con matrices. Otra de las ventajas de haber utilizado MATLAB es su facilidad a la hora de crear interfaces gráficas, con un diseño sencillo e intuitivo.

4.1. El lenguaje MATLAB

El nombre MATLAB proviene de *MATrix LABoratory* (Laboratorio de Matrices). MATLAB fue escrito originalmente para proporcionar un acceso sencillo al software matricial desarrollado por los proyectos LINPACK y EISPACK, que juntos representaban lo más avanzado en programas de cálculo matricial. MATLAB es un sistema interactivo cuyo elemento básico de datos es una matriz que no requiere dimensionamiento. Esto permite resolver muchos problemas numéricos en una fracción del tiempo que llevaría hacerlo en lenguajes como C, BASIC o FORTRAN. MATLAB ha evolucionado en los últimos años a partir de la colaboración de muchos usuarios. En entornos universitarios se ha convertido en la herramienta de enseñanza estándar para cursos de introducción en álgebra lineal aplicada, así como cursos avanzados en otras áreas. En la industria, MATLAB se utiliza para investigación y para resolver problemas prácticos de ingeniería y matemáticas, con un gran énfasis en aplicaciones de control y procesamiento de señales.

4.1.1. Tipos de datos.

Por defecto, MATLAB almacena todas las variables numéricas como valores de punto flotante de precisión doble. Los tipos de datos que utilizaremos en nuestro proyecto son:

- **Int8:** entero de 8 bits con el signo integrado.
- **Double:** número de doble precisión.
- **Char:** caracter alfanumérico. También se pueden hacer arrays de caracteres.
- **Cell:** es un tipo de dato que puede contener cualquier tipo de dato. Se utiliza sobre todo a la hora de hacer arrays. Lo más común son arrays de celdas de strings o combinar texto y números.
- **Handle:** es un tipo de puntero. Se suele utilizar para hacer referencia a un bloque de memoria u objeto.

Aunque no son propiamente un tipo de dato, también destacan:

- **NaN:** representa un dato que no es un número (not a number).
- **Matriz:** representa un conjunto de datos puestos como en una tabla de la dimensión que queramos. Este tipo conserva las propiedades de las matrices y, por lo tanto, todas las filas y columnas deben tener la misma dimensión escogida y cada dato que esté dentro de la matriz debe tener el mismo tipo de dato.

4.1.2. Funciones MATLAB utilizadas.

En esta sección se verán cuáles son las funciones MATLAB utilizadas para realizar nuestro proyecto. Sobre todo, nos centraremos en las funciones que nos permitirán hacer la selección de variables por distintos métodos. Suponemos que ya son conocidas las funciones básicas de los tipos, incluidas las de vectores y matrices.

- **xlsread:** este método se utiliza para importar una tabla excel desde MATLAB. Como parámetro hay que pasarle la dirección del excel que se quiera cargar. En nuestro caso, esta función nos devuelve una matriz con los valores de la matriz de excel y un vector de celdas con los nombres de las variables de la tabla excel.

- **glmfit:** escrito como *[b,dev,stats] = glmfit(X,y,distr)*, devuelve un vector (p + 1) de estimaciones de coeficientes, para una regresión lineal generalizada, de las respuestas en y sobre los predictores en X, usando la distribución distr. X es una matriz $n \times p$ de p predictores por cada una de las n observaciones. Distr puede ser cualquiera de las siguientes: *'binomial', 'gamma', 'inverse gaussian', 'normal'* (por defecto) y *'poisson'*. En la mayoría de los casos, y es un vector $n \times 1$ que contiene los valores del target. En nuestro caso, utilizaremos la distribución logit que es la que corresponde al modelo logístico. *Dev* es la desviación propia del ajuste. *Stats* es una estructura que contiene varios parámetros, aunque el que nos interesa es *stats.p*, que representa el p-valor.

- **glmval:** escrito como *yhat = glmval(b,X,link)*, calcula los valores predichos para el modelo lineal generalizado. Las distintas variables predictoras deben aparecer en diferentes columnas de X. b es un vector de estimaciones de coeficientes devuelto por la función glmfit. *Link* puede ser cualquiera de los vectores de caracteres o cualquier función personalizada que se usa en glmfit. En nuestro caso es *logit*.

- **polyfit:** escrito como p = *polyfit(x,y,n)*, devuelve los coeficientes para un polinomio $p(x)$ de grado n siendo el mejor ajuste por mínimos cuadrados para los valores de y. Los coeficientes en p están en potencias descendentes, y la longitud de p es $n + 1$

- **corrcoef:** escrito como R = *corrcoef(A)*, devuelve la matriz de correlación de A, siendo A una matriz en la que las columnas representan variables y las filas observaciones. Si pones un segundo argumento de la forma *corrcoef(A,B)* te calculará el coeficiente de correlación entre A y B siendo estos dos argumentos vectores con la información de cada variable.

- **lasso:** escrito como *[B,FitInfo] = lasso(X,Y,Name,Value)*, devuelve el conjunto de coeficientes ajustados por el método lasso según el lambda escogido. Se pueden incluir una serie de opciones a través del par de argumentos *Name-Value*. B es una matriz $p \times L$, donde p es el número de predictores de X y L es el número de valores de λ. *FitInfo* es una estructura que contiene información sobre el modelo, entre ellas, el valor de λ para el que se tiene el menor error.

Para más información sobre las funciones se pueden consultar en [20], donde nos hemos apoyado para realizar esta sección.

4.2. Interfaz de usuario guide.

En este apartado se explicará cómo se han realizado las interfaces en MATLAB y el funcionamiento que tienen cada una de ellas.

La aplicación se divide en varios scripts, el main principal y un script para cada cálculo distinto que queremos hacer: diferencia de medias, desviación típica, regresión lineal y corre-

lación, regresión logística y regresión lasso. A continuación, se explicará cada uno de estos scripts.

4.2.1. Main.

En esta parte se desarrolla la ventana principal de nuestra aplicación. Se compone de 4 partes como se puede observar en Figura 4.1

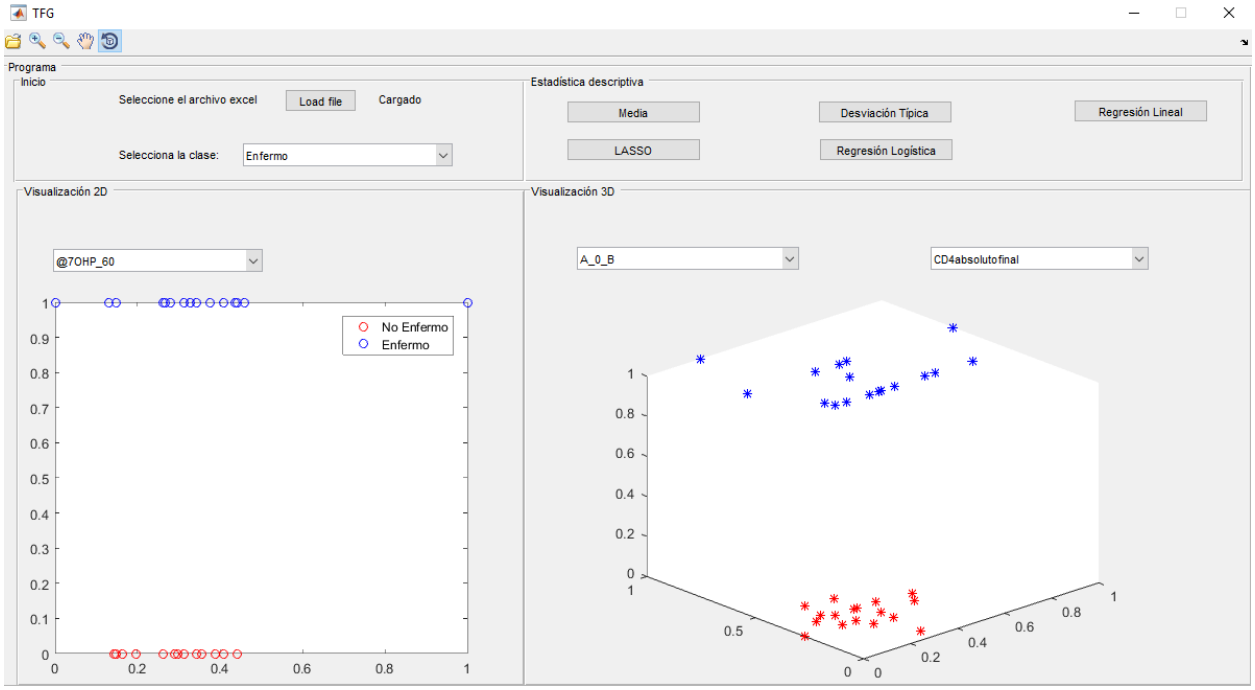


Figura 4.1: Programa principal

La primera parte es la carga de los datos a nuestro programa, situada en la esquina superior izquierda de nuestra aplicación (Figura 4.2). Se compone de un botón que se utiliza para seleccionar un archivo excel que contiene los datos. Al hacer clic se abre una ventana del explorador donde puedes buscar el excel.

Al seleccionar los datos se produce la imputación de los datos. Originalmente, los datos que se nos han proporcionado del Ramón y Cajal, contienen huecos para algunas variables de ciertos pacientes, bien porque no se realizó la medida de esa variable o bien porque los valores obtenidos estaban por encima o debajo del límite de detección del analito. Los algoritmos matemáticos que utilizamos para la selección de variables no se comportan de forma adecuada ante estos huecos, de hecho, la regresión Lasso elimina todas las observaciones que tienen huecos en alguna variable y, si no hiciésemos la imputación, todas las observaciones serían borradas. Esta imputación se realiza a través de medias agrupadas, es decir, agrupamos los datos por sexo, enfermedad y obesidad y calculamos la media de los valores no nulos de cada

variable. Si después de haber calculado las medias se encuentra un valor nulo en una variable específica, este valor se sustituye por la media calculada de dicha variable en la agrupación en la que se encuentre.

Mas abajo (Figura 4.2) se encuentra una lista desplegable que se utiliza para seleccionar la variable target. Como tenemos tres variables objetivo distintas, no tiene sentido que hagamos nuestros modelos con las tres variables, así que escogemos una y quitamos de nuestro conjunto de datos las otras dos.

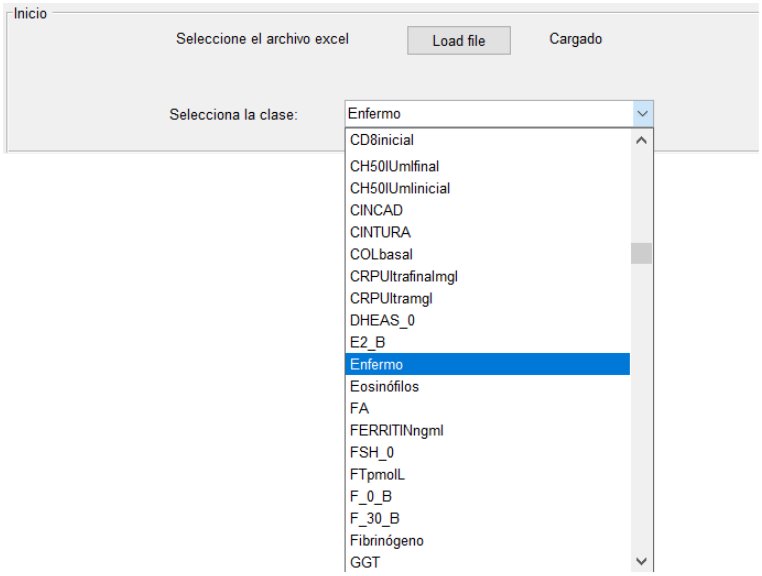


Figura 4.2: Carga de datos

Este paso es realmente importante para que la aplicación funcione ya que, si no se realiza, al no tener datos, se produce un error al intentar utilizar cualquiera de las funcionalidades de nuestro programa.

En la esquina superior derecha, nos encontramos un conjunto de botones (Figura 4.3) que nos llevarán a otras ventanas de la aplicación donde se realizarán todos los cálculos matemáticos explicados anteriormente.

Figura 4.3: Opciones de nuestro programa

- **Media:** se abrirá una ventana donde podremos ver el cálculo de las diferencias de medias respecto de la variable target escogida.
- **Desviación típica:** se abrirá una ventana donde podremos calcular la desviación típica de cada variable.

- **Regresión lineal:** se abrirá una ventana donde se mostrará la recta de regresión entre dos variables y su correlación
- **Regresión logística:** se abrirá una ventana que permitirá hacer una selección de variables a través de una regresión logística.
- **LASSO:** se abrirá una ventana que permitirá hacer una selección de variables a través de una regresión Lasso.

En la esquina inferior izquierda (Figura 4.4) se muestra una lista desplegable y una gráfica donde, al seleccionar una variable de la lista, se pintan los valores correspondientes de la variable en un eje de coordenadas, siendo el eje x la variable escogida y el eje y la variable target.

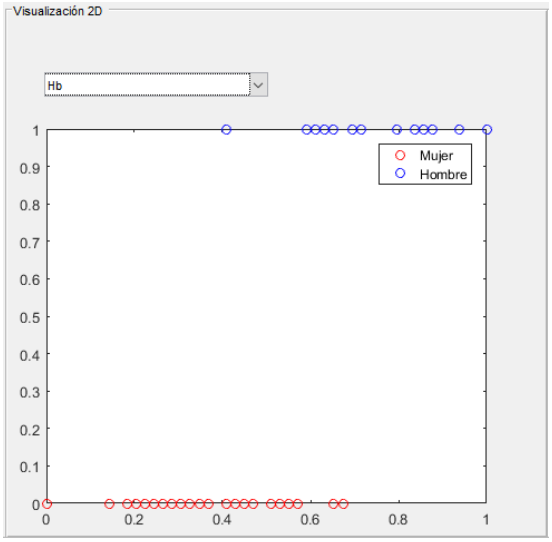


Figura 4.4: Gráfico 2D

Por último, en la esquina inferior derecha (Figura 4.5) se sitúa un eje de coordenadas donde se va a poder ver la nube de puntos formada por dos variables que se escogen en las listas desplegables y la variable target en 3D.

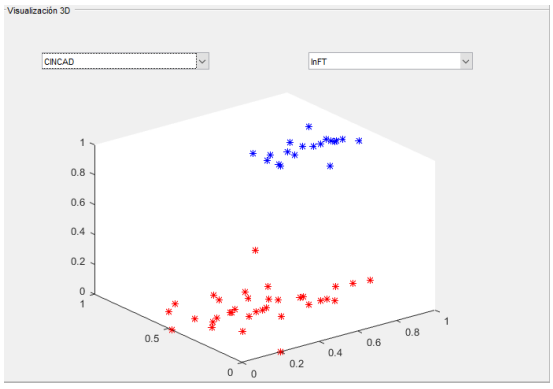


Figura 4.5: Gráfico 3D

4.2.2. Media.

A esta ventana se accede desde el main (Figura 4.3). Aquí se hacen todos los cálculos de diferencias de medias. La ventana consta de una gráfica, una lista y un espacio para introducir un número (Figura 4.6).

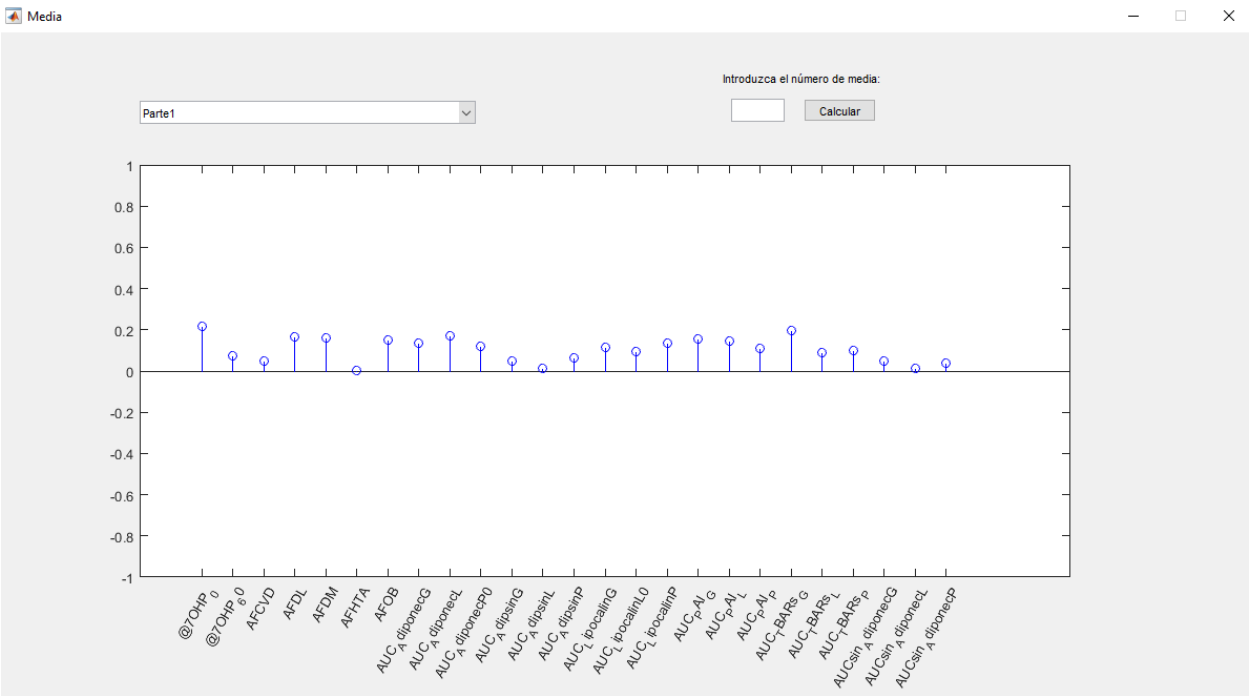


Figura 4.6: Diferencia de medias

En la gráfica se puede ver cuál es la diferencia de medias entre las clases, representado por un número en el eje y y el nombre de la variable en el eje x. El cálculo de la media se hace en base a la variable target escogida en el main. La lista desplegable se ha puesto para facilitar al usuario la visión del resultado ya que, al ser tantas variables, si las pusiéramos todas en una sola gráfica, no se podría distinguir ninguno de los valores. Por ello se ha dividido el conjunto en dieciséis partes con 24 variables cada parte, exceptuando la última que tiene siete.

En Figura 4.6 también se nos muestra un botón y un espacio para escribir un número. Este botón sirve para seleccionar el número de variables que el usuario quiere que aparezcan (respetando siempre la división que se ha explicado antes) pero ordenadas de mayor a menor. Es decir, si introduce un seis, le saldrán las seis variables que tengan una mayor diferencia en la media (Figura 4.7).

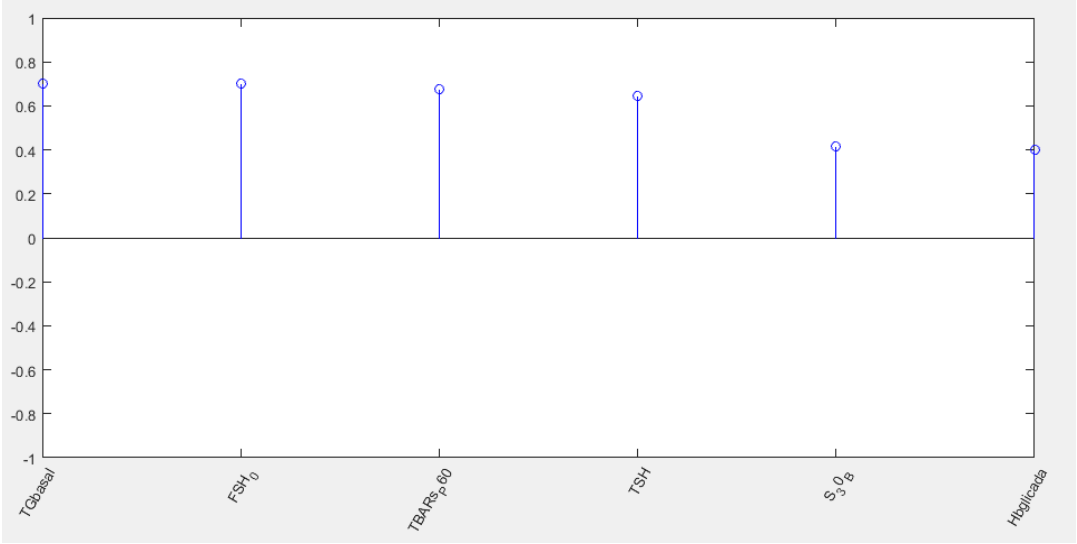


Figura 4.7: Cálculo de las seis variables con mayor diferencia de medias.

4.2.3. Desviación típica.

En esta ventana el usuario podrá calcular la desviación típica de cada una de las variables en cada grupo de su variable target. Esta ventana (Figura 4.8) se compone de un eje de coordenadas representado como en la media. La diferencia es que el usuario esta vez puede elegir de que clase quiere mirar la varianza de las variables con simplemente seleccionar esa clase en los botones situados en la esquina superior derecha. Igualmente, para facilitar la comprensión de los datos, se han dividido las variables en 16 partes de 24 variables cada una.

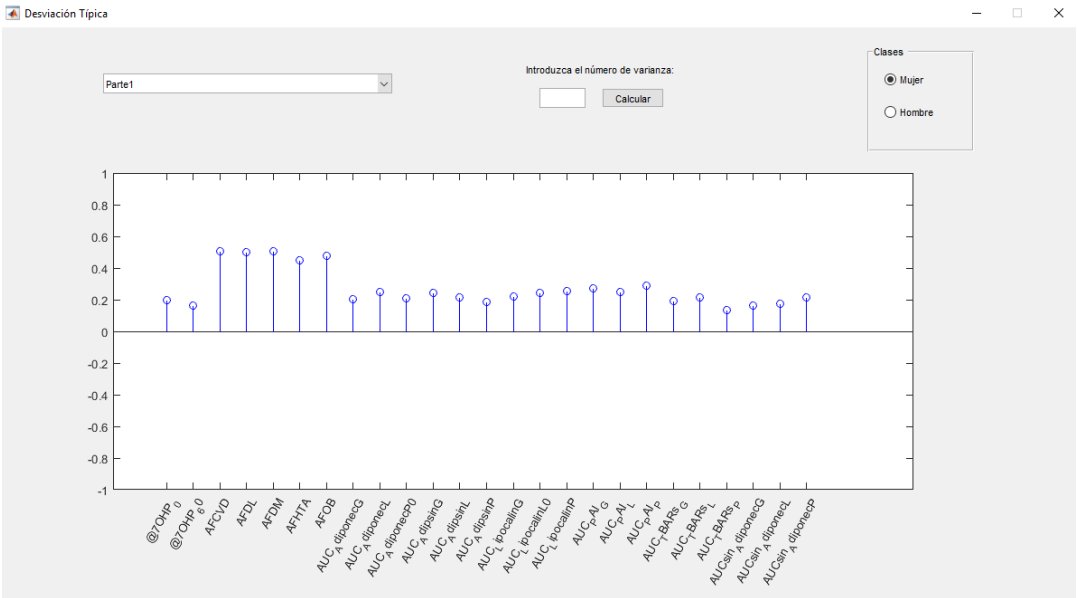


Figura 4.8: Desviación Típica

Al igual que con la media también podemos seleccionar el conjunto de variables que queramos que aparezcan, ordenadas de mayor a menor.

4.2.4. Regresión lineal y correlación.

En esta ventana se muestra todo lo referente a la regresión lineal y la correlación de variables explicado en el capítulo 4. La ventana (Figura 4.9) consta de tres partes. La primera son las listas desplegables que permiten escoger dos variables de nuestro conjunto para calcular su recta de regresión y su correlación. En esta parte están todas las variables, incluidas las variables target antes quitadas. Esto se hace así para que el usuario pueda ver cuál es la correlación de cualquier variable sin tener que volver atrás para cambiar el target. La segunda parte son los ejes de coordenadas, donde se sitúan en el eje x los valores de la variable seleccionada en la primera lista y en el eje y los de la segunda. La recta va cambiando a medida que se cambian los valores de la lista. Por último, la tercera parte se sitúa debajo de los ejes de coordenadas. Aquí se nos enseña el valor del coeficiente de correlación de Pearson, que como se ha explicado antes, indica la intensidad y sentido de la correlación

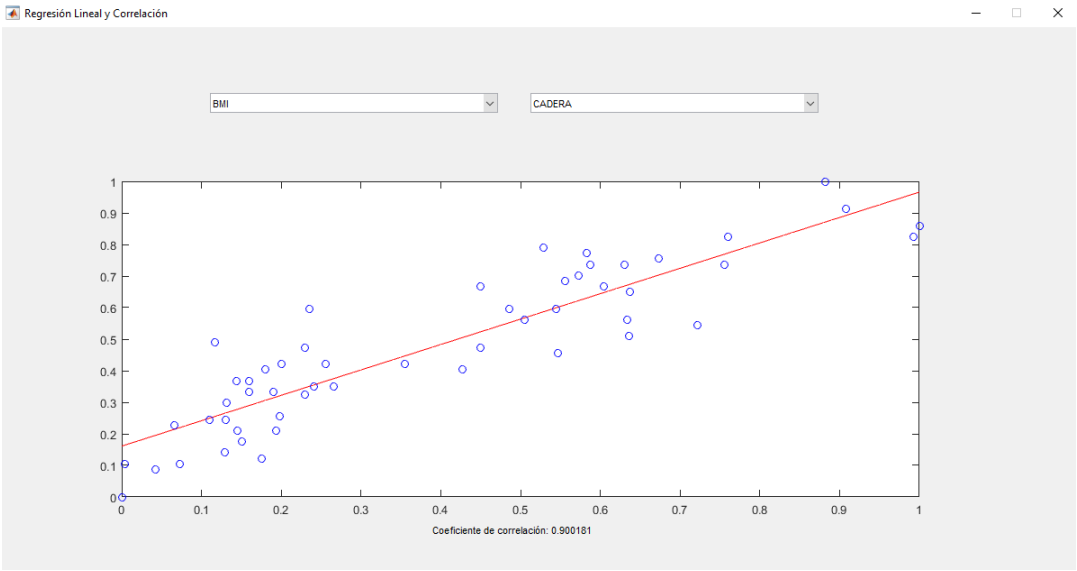


Figura 4.9: Correlación

4.2.5. Regresión logística.

En esta ventana es donde se hace el cálculo de selección de variables a través de una regresión logística (Figura 4.9). El proceso comienza cuando se pulsa el botón *Ejecutar*. Al principio el programa te hace un análisis de correlación de variables. Es decir, calcula la matriz de correlaciones y, recorriendo solo el triángulo superior de la matriz para hacerlo más eficiente, se quitan de nuestro conjunto de datos todas aquellas variables que tengan

una correlación mayor a 0,9.

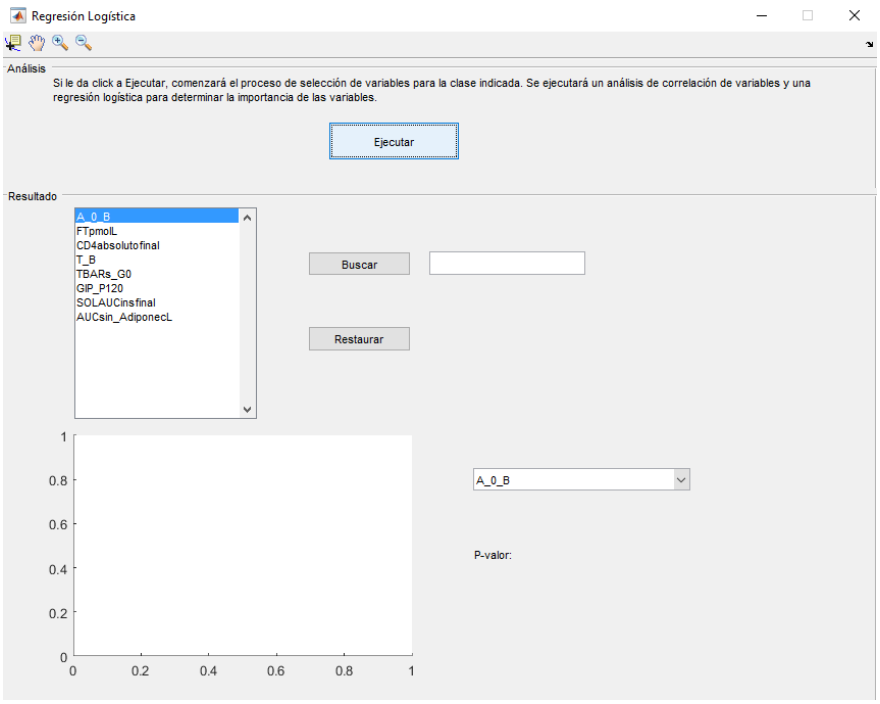


Figura 4.10: Regresión Logística

Después de la correlación, continuaríamos con lo que es la regresión logística, utilizando una de las funciones de MATLAB que, además, calcula los p-valores de las variables. Para seleccionar las variables que van a ajustar a nuestros modelos utilizamos el p-valor calculado, en concreto, todas las variables que tengan un p-valor menor que 0,05 se consideran significativas para explicar el modelo. Todas estas variables se muestran a través de una lista que se encuentra en el medio de la ventana. Junto a esta lista se han instalado dos botones:

- **Buscar:** a través del cuadro de texto situado a la derecha del botón permite filtrar las variables seleccionadas. Basta con que la cadena de caracteres que se introduzca este contenida en una variable para que dicha variable aparezca.

- **Restablecer:** vuelve a introducir todas las variables en la lista.

En la parte inferior de la ventana nos encontramos una lista y un sistema de coordenadas en el que se mostrará la función correspondiente a la regresión logística de la variable objetivo y la variable seleccionada en la lista. También se mostrará el p-valor de esa variable que hemos seleccionado.

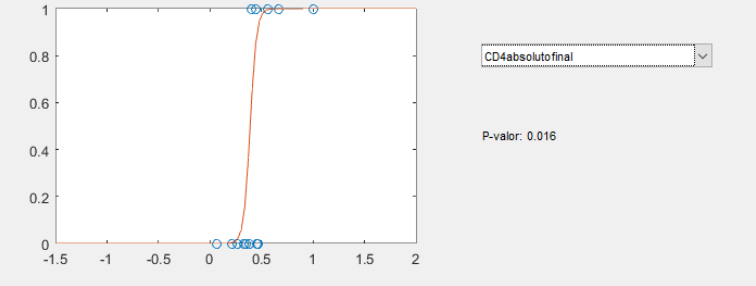


Figura 4.11: P-valor de variables seleccionadas

4.2.6. Lasso.

Otro método que tenemos para seleccionar las variables es el método Lasso. Esta ventana (Figura 4.12) es muy parecida a la de la regresión logística y consta de elementos iguales. Como se puede observar, existe también el botón *Ejecutar* que al pulsarlo comienza la regresión Lasso. Antes de ejecutar el algoritmo Lasso, como hicimos en la regresión logística, se hace un análisis de la correlación de las variables, así obtenemos el mismo número de variables que en el método de regresión logística. Además, si mantenemos las variables correlacionadas, el "peso"que tienen esas variables se reparte entre todas las variables correlacionadas y por tanto cada una podría tener menos "peso"del que se merece. Quitando estas variables este "peso"va todo a la misma variable y sube su importancia. Después se ejecuta el método Lasso para selección de variables que tiene por base lo explicado en el capítulo 4.

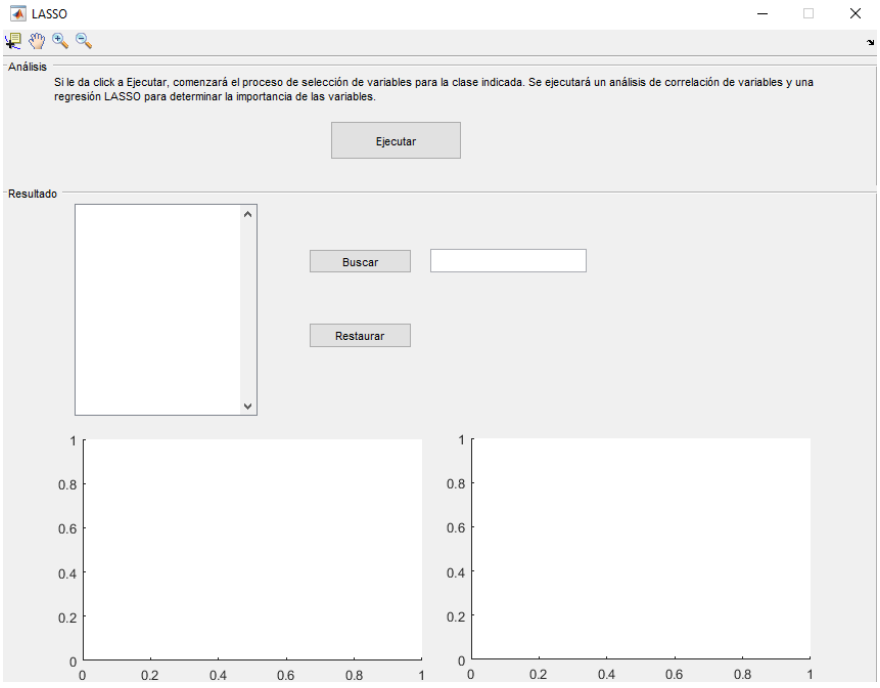


Figura 4.12: Método Lasso

Las variables que salen primero son las más significativas ya que tiene un β más alto. En esta ventana también disponemos de los botones *buscar* y *restablecer* para encontrar variables en nuestra lista. Junto con la selección de variables se muestran dos figuras (figura 4.13). La primera muestra las estimaciones correspondientes al coeficiente escogido. Como se puede observar, cumplen con lo explicado en el capítulo 4 para el método Lasso. Los coeficientes van tendiendo a cero a medida que aumentamos el λ. Gracias a las herramientas situadas en la parte superior de la ventana, podremos seleccionar cada línea para saber a qué variable se refiere. La línea que corta verticalmente al gráfico representa el valor de las variables para el λ escogido. La segunda de las figuras muestra el error que se obtiene al escoger un λ. En verde se muestra el λ con menor error de validación cruzada. De tal forma que una vez ejecutado el programa, la solución tendría este aspecto:

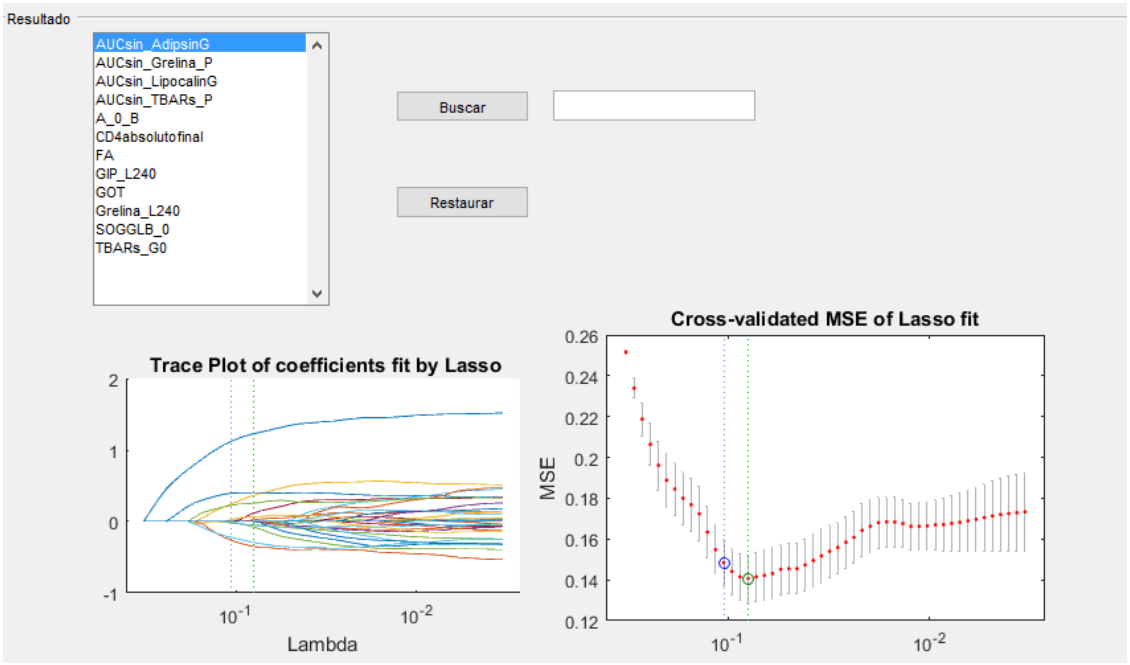


Figura 4.13: Resultados método Lasso

El círculo azul y la línea punteada localizan el punto con un error mínimo de validación cruzada más una desviación estándar. En nuestro caso, hemos escogido el valor de λ que representa la línea discontinua verde.

Capítulo 5

Pruebas realizadas.

En este capítulo se explicarán los pasos que se han seguido para obtener la aplicación y qué métodos se han desechado. Al inicio del proyecto se planteó la posibilidad de usar WEKA. WEKA es una plataforma de software para el aprendizaje automático y la minería de datos. Procedimos a realizar un método para observar cuáles eran las variables de las que dependía más el modelo, El algoritmo que se utilizó en Weka fue el *Best First Search*. Este algoritmo intenta encontrar la solución a través de una heurística, seleccionando las variables mas prometedoras. La heurística utilizada es una propia de WEKA llamada CfsSubsetEval que permite evaluar el valor de un subconjunto de variables al considerar la capacidad predictiva individual de cada característica junto con el grado de redundancia entre ellas ([21]). Los resultados obtenidos no nos convencieron ya que las variables seleccionadas no se adecuaban a lo que veíamos en las gráficas y se decidió profundizar en el estudio..

Por lo tanto, se procedió a hacer ciertas pruebas sobre estadística descriptiva en MATLAB para determinar la dependencia de las variables. Estas pruebas consistieron en crear programas que calculasen, por ejemplo, la diferencia de las medias por variable de cada una de las clases que tienen nuestras variables objetivo o targets (sexo, obesidad y SOP). Si la misma variable en las dos clases tiene la misma media quiere decir que sus valores son parecidos y que por tanto son candidatas a ser independientes de la variable objetivo. Si, por el contrario, sus medias son muy distintas, los valores de la variable dependen de la clase en la que se sitúe y, por lo tanto, se supone que explicará mejor la variable objetivo escogida que otras con medias parecidas. Como se ha especificado antes, esto no tiene por qué ser del todo cierto y por ello también se creó un programa para calcular la desviación típica por clase. Así, uniendo estos dos métodos, las conclusiones serían más fiables. Sin embargo, esto no es suficiente y observamos que, exceptuando un par de variables, la diferencia de medias era parecida con desviaciones típicas no muy altas y, por lo tanto, no se explicaba bien la variable.

Así surgió la idea de reducir primero el número de variables que teníamos. Variables que explicasen lo mismo las agruparíamos o las eliminaríamos. Además, esto nos serviría para poder realizar otros métodos más avanzados ya que si reducíamos mucho el número de

variables, podríamos llegar a tener un conjunto de datos en el que el número de observaciones fuese mayor que el de variables. Las dos soluciones que se nos ocurrieron fueron hacer un análisis de correlación y realizar un análisis de componentes principales.

Para el análisis de correlación se realizó un programa que permite calcular el coeficiente de correlación de Pearson para hallar la intensidad de la relación entre dos variables y, además, para ayudar a visualizar las correlaciones, el programa pinta la recta de regresión. Esto demostró que al menos se podían quitar unas 100 variables que tenían una correlación con otras de más de 0.9 y por lo tanto reducíamos así mucho la cantidad de variables sobre las que actuar. Aun así, seguían siendo demasiadas variables y por ello se pasó a investigar sobre el análisis de componentes principales.

El análisis de componentes principales (PCA) es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos. Supongamos que deseamos visualizar n observaciones con mediciones sobre un conjunto de p características, $X_1, X_2, \ldots, Xp$, como parte de un análisis exploratorio de datos. Podríamos hacerlo examinando los diagramas de dispersión bidimensionales de los datos, cada uno de los cuales contiene las mediciones de n observaciones en dos de las características. Sin embargo, existen $\binom{p}{2} = p(p-1)/2$ diagramas de dispersión; por ejemplo, con $p = 10$ hay 45 diagramas. Si p es grande, entonces no será posible mirar todos ellos; por otra parte, muy probablemente ninguno de ellos será determinante ya que cada uno contiene sólo una pequeña fracción de la información total presente en el conjunto de datos. Nos gustaría encontrar una representación de baja dimensión de los datos que capture tanta información como sea posible. Por ejemplo, si podemos obtener una representación bidimensional de los datos que capturan la mayor parte de la información, entonces podemos trazar las observaciones en este espacio de baja dimensión. PCA proporciona una herramienta para hacer esto. Encuentra una representación de baja dimensión de un conjunto de datos que contiene tanta información como sea posible. La idea es que cada una de las n observaciones vive en el espacio p-dimensional, pero no todas estas dimensiones son igualmente interesantes. PCA busca un pequeño número de dimensiones que son tan interesantes como sea posible, donde el concepto de interesante se mide por cuánto varían las observaciones a lo largo de cada dimensión. Cada una de las dimensiones encontradas por PCA es una combinación lineal de las características de p.

El problema que se vio con PCA fue en el momento de la interpretación de los datos. Cada componente contiene una combinación lineal de todas las variables y nuestro modelo se explica a través de varias de esas combinaciones lineales. Si queríamos ver relaciones entre variables, la importancia de las variables en nuestros targets, con este método, se hacía muy difícil interpretar y por ello se desechó.

En este momento empezamos a buscar métodos de selección de variables. Observamos que el método de regresión lineal funcionaba muy mal con nuestras variables objetivo así que se investigó sobre otros métodos de regresión que aceptasen targets binarios. De ahí surgió la regresión logística descrita en el capítulo 4. En la regresión logística tomamos como buenas aquellas variables que tuviesen un p-valor por debajo del 0,05.

El siguiente paso fue intentar hacer una selección de variables a través de algoritmos más complejos como el random forest. Random forest es un algoritmo para clasificación y regresión de amplio uso en la comunidad que tiene un rendimiento bueno para datos de alta dimensionalidad. El algoritmo se basa en construir un conjunto de árboles de decisión con la peculiaridad de que en cada uno de ellos se selecciona una muestra aleatoria de m predictores como candidatos, divididos del conjunto completo de p predictores. Después se promedian todas las predicciones obtenidas con cada uno de los árboles. El hecho de no coger todas las predictoras es útil ya que, si hay una variable que pesa mucho y otras que son importantes, pero con menor peso, en varios de los árboles, la principal quedará fuera y dejará sitio a todas esas variables con menos peso. Si cogiésemos todos los predictores al final todos nuestros árboles estarían correlacionados.

El random forest es, además, un algoritmo eficiente incluso con una cantidad alta de predictores. Pero nuestro problema al utilizarlo fue la cantidad de observaciones n que teníamos. Al haber tan pocas observaciones nuestro árbol final tenía muy poca profundidad y seleccionaba una o dos variables como mucho.

Volvía a salir el problema $p >> n$ así que se investigó sobre posibles métodos que funcionasen bien en estos casos. Uno de los que mejor se comportan es el método lasso tratado en el capítulo 4. Este método, definido como una variación de mínimos cuadrados, selecciona variables dependiendo de una penalización.

Otro de los grandes problemas que se han tenido con el hecho de que p es mucho mayor que n es el testeo de nuestros modelos. Para testear un modelo es necesario separar un conjunto de datos y no entrenar nuestros modelos con ellos. Así después, una vez obtenido el modelo, se puede comprobar el porcentaje de acierto que se ha tenido y la bondad del modelo. Pero al tener únicamente 53 pacientes, si separábamos unos pocos, o bien nos quedábamos con pocas observaciones para entrenar nuestros modelos o bien no había suficientes para que el testeo fuese fiable. Por ello, la solución de los dos modelos es considerada, aunque tiene mayor peso el modelo lasso debido al buen comportamiento que tiene en general con nuestro tipo de problemas.

Capítulo 6

Resultados.

6.1. Variables seleccionadas por los modelos.

Una vez realizados los modelos hemos obtenido los siguientes resultados para los diferentes targets:

- Enfermedad:

Tabla 6.1: Lasso Enfermedad

Variable	β
CD4absolutofinal	1,2278
A_0_B	0,4054
AUCsin_LipocalinG	0,3667
AUCsin_Grelina_P	-0,3508
TBARs_G0	-0,2940
Grelina_L240	0,2566
AUCsin_TBARs_P	0,1145
AUCsin_AdipsinG	-0,0773
GOT	0,0646
SOGGLB_0	-0,0522
GIP_L240	-0,0481
FA	0,0220

Tabla 6.2: Regresión Logística Enfermedad

Variable	$P - valor$
A_0_B	0,0095
FTpmolL	0,0156
CD4absolutofinal	0,0166
T_B	0,0197
TBARs_G0	0,0252
GIP_P120	0,0279
SOLAUCinsfinal	0,0334
AUCsin_AdiponecL	0,0399

- Sexo:

Tabla 6.3: Lasso Sexo

Variable	β
Grasa	-0,5511
TALLA	0,482
Hb	0,4184
CINCAD	0,2937
Fibrinógeno	-0,2565
HDLbasal	-0,2380
AUCsin_LipocalinP	-0,204
FERRITINngml	0,1756
SOLAUCglutotal	0,1703
GOT	0,1447
Grelina_L240	0,1221
S_30_B	0,113
AUCsin_Grelina_L	0,1117
PRL	-0,0845
A_0_B	-0,0822
SOPINS120	-0,064
Grelina_P60	-0,0566
SOGHOMABETA	-0,051
AUCsin_AdipsinP	-0,0496
AUC_TBARs_G	0,0482
PAI_P120	-0,0428
TAD	-0,034
TSH	-0,005

Tabla 6.4: Regresión Logística Sexo

Variable	$P-valor$
Grasa	0,0002
TALLA	0,0005
CD4absolutofinal	0,0009
SHBG_B	0,001
Hb	0,0011
E2_B	0,0017
CD4final	0,0022
@7OHP_0	0,0023
FSH_0	0,0029
HDLbasal	0,0036
SOPGL120	0,0044
SOLGL240	0,0047
CINCAD	0,0049
A_30_B	0,005
FERRITINngml	0,0054
SOLAUCglutotal	0,006
GOT	0,0064
Fibrinógeno	0,0078
GPT	0,0093
S_30_B	0,0097
SOGGL30	0,0108
AUC_TBARs_G	0,0118
SOPAUCglutotal	0,0147
TBARs_G60	0,0247
SOGGL120	0,0311
Grelina_G120	0,0369
SOGGLB_0	0,0423
AUCsin_Grelina_L	0,0452
TBARs_G0	0,0476
TAS	0,0566

- Obesidad:

Tabla 6.5: Lasso Obesidad

Variable	β
CINTURA	0,8453
Grelina_G60	-0,163
SOGISIcomp	-0,0988
Grasa	0,0589

Tabla 6.6: Regresión Logística Obesidad

Variable	$P-valor$
Grelina_G60	0,0002
SOPISIceder	0,0003
C4mgdlfinal	0,0004
SOGHOMAIR	0,0005
SOGISIcomp	0,0005
SOLHOMAIR	0,0005
C3mgdlfinal	0,0006
Grasa	0,0006
SOLISIceder	0,0006
SOPHOMAIR	0,0006
SOLAUCinstotal	0,0007
CINTURA	0,0008
C3mgdlinicial	0,0008
SOGGLUINS0	0,0009
CINCAD	0,0009
SOLINS240	0,0011
CRPUltrafinalmgl	0,0011
SOGISIceder	0,0011
LGSOGTG120	0,0013
AUC_PAI_P	0,0017
S_3_B	0,0018
Grelina_P0	0,0023
SOGGL120	0,0023
SOPINS120	0,0024
A_30_B	0,0026
Fibrinógeno	0,0031
SOPGLUINS0	0,0031
AFOB	0,0035
AUC_PAI_G	0,0035
AUCsin_Grelina_L	0,0036
C4mgdlinicial	0,0038
PAI_L0	0,004
SOGINS120	0,0046
TAD	0,0048
Grelina_P120	0,0056
PAI_P120	0,0056
PAI_G0	0,0059
SOPGLB_0	0,006
SOLGLUINS0	0,0063
AUC_TBARs_G	0,0065

Las variables se muestran ordenadas según su importancia, de mayor a menor, y aquellas que están de color verde son las que coinciden en los dos modelos y, por lo tanto, se presupone que van a explicar mejor nuestro conjunto de datos.

Podemos observar cómo nuestros resultados tienen sentido. Por ejemplo, si nos centramos en los resultados de le enfermedad (tablas 6.1 y 6.2) vemos que destacan sobre todo la variable CD4, que son células efectoras del sistema inmunológico, y la variable A_0_B, hormona que se usa para diagnosticar enfermedades endocrinas. Si nos vamos a las tablas 6.3 y 6.4, vemos como destacan las variables de grasa, talla y nivel de hemoglobina (Hb) que tienen valores específicos dependiendo del sexo. También cabe destacar variables como FERRITINngml, importante para distinguir entre enfermedades metabólicas según seas hombre o mujer, o el fibrinógeno, que es una molécula indicativa de inflamación e importante para el estudio de indicadores que marcan diferencias en el sexo. Pasando a los resultados de la obesidad (tablas 6.5 y 6.6), es obvio que la cantidad de grasa o la circunferencia de la cintura están muy relacionados con la obesidad pero también son importantes la grelina, hormona que aumenta el apetito, o la variable SOGISIcomp que sirve para medir la insulina de un paciente después de haberle dado una sobrecarga oral de glucosa. Es normal su aparición en nuestro estudio ya que las personas obesas tienen una mayor resistencia a la insulina, demostrada con niveles superiores de glucosa tras la insulina.

6.2. Conclusiones.

Tras analizar los resultados obtenidos, se llega a la conclusión de que, aunque no podamos testear nuestros modelos, el más fiable es el modelo lasso, que es el que en general acota más el número de variables, además de que está demostrada su efectividad ante el tipo de problemas que ofrecen nuestros datos.

En cuestión de tiempo de ejecución, los dos algoritmos pueden servir ya que, con la cantidad de datos que tenemos, tardan aproximadamente lo mismo (0,9 segundos). Para comprobar cuál es el más eficiente deberíamos probar con una cantidad de datos mayor.

En relación a las variables analizadas que parecen tener más importancia con el desarrollo de SOP y su respuesta a los diferentes macronutrientes y como conclusión de este proyecto, cabe destacar que el modelo selecciona variables relacionadas con el componente metabólico (Grelina, lipocalina, adipsina, etc.) o de activación de sistema inmunológico (CD4) frente a los determinantes hormonales androgénicos (androstendiona (A) en el modelo Lasso). Es decir, que aunque existan variables que representen a hormonas sexuales, los datos muestran que también son importantes aquellas relacionadas con el metabolismo y las hormonas gastrointestinales que varían en respuesta a glucosa y proteínas como se observa en las variables del área bajo la curva (AUC): AUCsin_lipocalina y AUCsin_grelina. Esto es importante para confirmar que el SOP no es únicamente una alteración hormonal, sino también metabólica y que a su vez modifica parámetros de estrés oxidativo como el TBARS que mide la peroxidación lipídica.

Por otro lado, hay que remarcar el papel de la obesidad como parte de la clínica del SOP, no solo por su mayor prevalencia sino por el agravamiento de los factores metabólicos adversos que hacen que estas mujeres tengan un mayor riesgo de enfermedades cardiovasculares. En el modelo por obesidad se aprecia un papel en la concentración de insulina, glucosa y en el valor del HOMAIR (índice de resistencia a insulina).

En el caso del modelo que diferencia varones y mujeres también es importante resaltar que existen muchos parámetros diferenciales (dimorfismo sexual) relacionados tanto con hormonas como con el metabolismo de glucosa y lípidos. Esto es importante a la hora de tratar las enfermedades metabólicas, pues probablemente sea diferente el curso de la enfermedad en hombres y mujeres, así como las respuestas a los tratamientos en función de la concentración de hormonas. Actualmente no existen guías clínicas que diferencien entre hombres y mujeres, pero se sabe que diferentes enfermedades tienen una prevalencia e impacto diferente en función del sexo.

Para posibles futuras investigaciones, se recomienda que se aumente el número de personas a observar para poder mejorar los modelos.

Por último, podemos señalar la utilidad de estos modelos para elucidar las variables más importantes involucradas en la fisiopatología del SOP y así poder dirigir futuras investigaciones centrando nuestra atención en estas variables a fin de conocer su mecanismo de acción sobre el SOP más en profundidad y poder aplicarlo a la práctica clínica de estos pacientes.

Anexos

Apéndice A

Código Matlab

A.1. Main

```
function varargout = TFG(varargin)
% TFG MATLAB code for TFG.fig
%      TFG, by itself, creates a new TFG or raises the existing
%      singleton*.
%
%      H = TFG returns the handle to a new TFG or the handle to
%      the existing singleton*.
%
%      TFG('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TFG.M with the given input arguments.
%
%      TFG('Property','Value',...) creates a new TFG or raises the
%      existing singleton*.  Starting from the left, property value pairs
%      are applied to the GUI before TFG_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
%      application stop.  All inputs are passed to TFG_OpeningFcn via
%      varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
   one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help TFG

% Last Modified by GUIDE v2.5 18-Aug-2017 19:49:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @TFG_OpeningFcn, ...
```

```
                   'gui_OutputFcn',  @TFG_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before TFG is made visible.
function TFG_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to TFG (see VARARGIN)

% Choose default command line output for TFG
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes TFG wait for user response (see UIRESUME)
% uiwait(handles.main);


% --- Outputs from this function are returned to the command line.
function varargout = TFG_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in load.
function load_Callback(hObject, eventdata, handles)
    [FileName,PathName]=uigetfile('*.xlsx');
    handles.load=fullfile(PathName,FileName);

    if FileName == 0
        return;
    end;
    set(handles.textCargado,'string','');
    [data,names]=xlsread(handles.load);
    data=imputar(data);
    m=size(data,1);
```

```
89      cellNoSort=[names;num2cell(data)];
90      cellSort=transpose(sortrows(transpose(cellNoSort),1));
91      handles.data=cell2mat(cellSort(2:m+1,:));
92      handles.names=cellSort(1,:);
93      set(handles.textCargado,'string','Cargado');
94
95
96
97
98      set(handles.selectClass,'string',handles.names);
99      set(handles.popupVarVarX,'string',handles.names);
100     set(handles.popupVarVarY,'string',handles.names);
101     handles.selectedVarX = 1;
102     handles.selectedVarY = 1;
103     setappdata(0,'data',handles.data);
104     setappdata(0,'names',handles.names);
105     guidata(hObject, handles);
106
107 % hObject    handle to load (see GCBO)
108 % eventdata  reserved - to be defined in a future version of MATLAB
109 % handles    structure with handles and user data (see GUIDATA)
110
111
112 function data=imputar(data)
113
114 [m,n] = size(data);
115
116 obesidad=[1,10,18,27,35,45,54];
117 for j=4:n
118     for i=1:6
119         if any(isnan(data(obesidad(i):obesidad(i+1)-1,j)))
120             media=mean(data(obesidad(i):obesidad(i+1)-1,j),'omitnan');
121             for k=obesidad(i):obesidad(i+1)-1
122                 if isnan(data(k,j))
123                     data(k,j) = media;
124                 end
125             end
126         end
127     end
128 end
129
130
131
132 function texto_Callback(hObject, eventdata, handles)
133 % hObject    handle to texto (see GCBO)
134 % eventdata  reserved - to be defined in a future version of MATLAB
135 % handles    structure with handles and user data (see GUIDATA)
136
137 % Hints: get(hObject,'String') returns contents of texto as text
138 %        str2double(get(hObject,'String')) returns contents of texto as a
139 %        double
140
141
142 % --- Executes during object creation, after setting all properties.
143 function texto_CreateFcn(hObject, eventdata, handles)
144 % hObject    handle to texto (see GCBO)
```

```
145 % eventdata  reserved - to be defined in a future version of MATLAB
146 % handles    empty - handles not created until after all CreateFcns called
147
148 % Hint: edit controls usually have a white background on Windows.
149 %       See ISPC and COMPUTER.
150 if ispc && isequal(get(hObject,'BackgroundColor'),...
151                    get(0,'defaultUicontrolBackgroundColor'))
152     set(hObject,'BackgroundColor','white');
153 end
154
155
156 % --- Executes on button press in media.
157 function media_Callback(hObject, eventdata, handles)
158 % hObject    handle to media (see GCBO)
159 % eventdata  reserved - to be defined in a future version of MATLAB
160 % handles    structure with handles and user data (see GUIDATA)
161 Media;
162
163 % --- Executes on button press in desvTipica.
164 function desvTipica_Callback(hObject, eventdata, handles)
165 % hObject    handle to desvTipica (see GCBO)
166 % eventdata  reserved - to be defined in a future version of MATLAB
167 % handles    structure with handles and user data (see GUIDATA)
168 Desviacion_Tipica;
169
170 % --- Executes on button press in regresionLineal.
171 function regresionLineal_Callback(hObject, eventdata, handles)
172 % hObject    handle to regresionLineal (see GCBO)
173 % eventdata  reserved - to be defined in a future version of MATLAB
174 % handles    structure with handles and user data (see GUIDATA)
175 Regresion_lineal;
176
177 % --- Executes on selection change in popupVarClass.
178 function popupVarClass_Callback(hObject, eventdata, handles)
179 % hObject    handle to popupVarClass (see GCBO)
180 % eventdata  reserved - to be defined in a future version of MATLAB
181 % handles    structure with handles and user data (see GUIDATA)
182
183 selectedVar = get(hObject,'Value');
184
185 axes(handles.axesVarClass);
186 m1=size(handles.dataClass1,1);
187 m2=size(handles.dataClass2,1);
188 y=zeros(m1,1);
189 plot(handles.dataClass1(:,selectedVar),y,'ro');
190 hold on;
191 y=ones(m2,1);
192 plot(handles.dataClass2(:,selectedVar),y,'bo');
193 hold off;
194 legend(handles.textClass0,handles.textClass1);
195
196 % Hints: contents = cellstr(get(hObject,'String')) returns popupVarClass
197 %                   contents as cell array
198 %        contents{get(hObject,'Value')} returns selected item from
199 %                   popupVarClass
200
```

```
201
202 % --- Executes during object creation, after setting all properties.
203 function popupVarClass_CreateFcn(hObject, eventdata, handles)
204 % hObject    handle to popupVarClass (see GCBO)
205 % eventdata  reserved - to be defined in a future version of MATLAB
206 % handles    empty - handles not created until after all CreateFcns called
207
208 % Hint: popupmenu controls usually have a white background on Windows.
209 %       See ISPC and COMPUTER.
210 if ispc && isequal(get(hObject,'BackgroundColor'),...
211                    get(0,'defaultUicontrolBackgroundColor'))
212     set(hObject,'BackgroundColor','white');
213 end
214
215
216 % --- Executes on selection change in popupVarVarX.
217 function popupVarVarX_Callback(hObject, eventdata, handles)
218 % hObject    handle to popupVarVarX (see GCBO)
219 % eventdata  reserved - to be defined in a future version of MATLAB
220 % handles    structure with handles and user data (see GUIDATA)
221
222 % Hints: contents = cellstr(get(hObject,'String')) returns popupVarVarX
223 %        contents as cell array
224 %        contents{get(hObject,'Value')} returns selected item from
225 %        popupVarVarX
226
227 handles.selectedVarX = get(hObject,'Value');
228
229 m1=size(handles.dataClass1,1);
230 m2=size(handles.dataClass2,1);
231 y=zeros(m1,1);
232 axes(handles.axesVarVar);
233 axis([-1 2 0 1]);
234 plot3(handles.dataClass1(:,handles.selectedVarX),...
235       handles.dataClass1(:,handles.selectedVarY),y,'r*');
236 hold on;
237 y=ones(m2,1);
238 plot3(handles.dataClass2(:,handles.selectedVarX),...
239       handles.dataClass2(:,handles.selectedVarY),y,'b*');
240 hold off;
241 % end;
242 guidata(hObject, handles);
243
244 % --- Executes during object creation, after setting all properties.
245 function popupVarVarX_CreateFcn(hObject, eventdata, handles)
246 % hObject    handle to popupVarVarX (see GCBO)
247 % eventdata  reserved - to be defined in a future version of MATLAB
248 % handles    empty - handles not created until after all CreateFcns called
249
250 % Hint: popupmenu controls usually have a white background on Windows.
251 %       See ISPC and COMPUTER.
252 if ispc && isequal(get(hObject,'BackgroundColor'),...
253                    get(0,'defaultUicontrolBackgroundColor'))
254     set(hObject,'BackgroundColor','white');
255 end
256
```

```
257
258 % --- Executes on selection change in popupVarVarY.
259 function popupVarVarY_Callback(hObject, eventdata, handles)
260 % hObject    handle to popupVarVarY (see GCBO)
261 % eventdata  reserved - to be defined in a future version of MATLAB
262 % handles    structure with handles and user data (see GUIDATA)
263
264 % Hints: contents = cellstr(get(hObject,'String')) returns popupVarVarY
265 %        contents as cell array
266 %        contents{get(hObject,'Value')} returns selected item from
267 %        popupVarVarY
268 handles.selectedVarY = get(hObject,'Value');
269
270 m1=size(handles.dataClass1,1);
271 m2=size(handles.dataClass2,1);
272 y=zeros(m1,1);
273 axes(handles.axesVarVar);
274 axis([-1 2 0 1]);
275 plot3(handles.dataClass1(:,handles.selectedVarX),...
276       handles.dataClass1(:,handles.selectedVarY),y,'r*');
277 hold on;
278 y=ones(m2,1);
279 plot3(handles.dataClass2(:,handles.selectedVarX),...
280       handles.dataClass2(:,handles.selectedVarY),y,'b*');
281 hold off;
282
283 guidata(hObject, handles);
284
285 % --- Executes during object creation, after setting all properties.
286 function popupVarVarY_CreateFcn(hObject, eventdata, handles)
287 % hObject    handle to popupVarVarY (see GCBO)
288 % eventdata  reserved - to be defined in a future version of MATLAB
289 % handles    empty - handles not created until after all CreateFcns called
290
291 % Hint: popupmenu controls usually have a white background on Windows.
292 %       See ISPC and COMPUTER.
293 if ispc && isequal(get(hObject,'BackgroundColor'),...
294                    get(0,'defaultUicontrolBackgroundColor'))
295     set(hObject,'BackgroundColor','white');
296 end
297
298
299 % --- Executes on button press in seleccion.
300 function seleccion_Callback(hObject, eventdata, handles)
301 % hObject    handle to seleccion (see GCBO)
302 % eventdata  reserved - to be defined in a future version of MATLAB
303 % handles    structure with handles and user data (see GUIDATA)
304 seleccionarVars;
305
306
307 % --- Executes on selection change in selectClass.
308 function selectClass_Callback(hObject, eventdata, handles)
309 % hObject    handle to selectClass (see GCBO)
310 % eventdata  reserved - to be defined in a future version of MATLAB
311 % handles    structure with handles and user data (see GUIDATA)
312
```

```
313 % Hints: contents = cellstr(get(hObject,'String')) returns selectClass
314 %                    contents as cell array
315 %        contents{get(hObject,'Value')} returns selected item from
316 %                    selectClass
317
318 selectedClass = get(hObject,'Value');
319 namesVar = get(hObject,'String');
320 nameVar = namesVar(selectedClass);
321 handles.textClass0='NoClass';
322 handles.textClass1='NoClass';
323
324 handles.selectedClass = selectedClass;
325 m=size(handles.data,1);
326
327 handles.dataClass1=[];
328 handles.dataClass2=[];
329
330
331 [m,n]=size(handles.data);
332
333 data=handles.data;
334 names = handles.names;
335 if strcmp(nameVar,'Enfermo') == 1
336     m = 34;
337     pos = find(strcmp(names,'Sexo'));
338     data(:,pos) = [];
339     names(pos) = [];
340     pos = find(strcmp(names,'OBESE'));
341     data(:,pos) = [];
342     names(pos) = [];
343
344     handles.selectedClass = find(strcmp(names,'Enfermo'));
345     handles.textClass0 = 'No Enfermo';
346     handles.textClass1 = 'Enfermo';
347 elseif strcmp(nameVar,'Sexo') == 1
348     pos = find(strcmp(names,'Enfermo'));
349     data(:,pos) = [];
350     names(pos) = [];
351     pos = find(strcmp(names,'OBESE'));
352     data(:,pos) = [];
353     names(pos) = [];
354
355     handles.selectedClass = find(strcmp(names,'Sexo'));
356     handles.textClass0 = 'Mujer';
357     handles.textClass1 = 'Hombre';
358 elseif strcmp(nameVar,'OBESE') == 1
359     pos = find(strcmp(names,'Sexo'));
360     data(:,pos) = [];
361     names(pos) = [];
362     pos = find(strcmp(names,'Enfermo'));
363     data(:,pos) = [];
364     names(pos) = [];
365
366     handles.selectedClass = find(strcmp(names,'OBESE'));
367     handles.textClass0 = 'No Obeso';
368     handles.textClass1 = 'Obeso';
```

```
369 end
370
371
372
373 n = size(data,2);
374 if handles.selectedClass == 1
375     handles.dataNoTarget=data(1:m,2:n);
376     handles.namesNoTarget=names(2:n);
377 elseif handles.selectedClass == n
378     handles.dataNoTarget=data(1:m,1:n-1);
379     handles.namesNoTarget=names(1:n-1);
380 else
381     handles.dataNoTarget=[data(1:m,1:handles.selectedClass-1) ...
382         data(1:m,handles.selectedClass+1:n)];
383     handles.namesNoTarget=[names(1:handles.selectedClass-1) ...
384         names(handles.selectedClass+1:n)];
385 end
386
387 handles.target=data(1:m,handles.selectedClass);
388
389 for i=1:m
390     if handles.target(i)==0
391         handles.dataClass1=[handles.dataClass1;handles.dataNoTarget(i,:)];
392     elseif handles.target(i)==1
393         handles.dataClass2=[handles.dataClass2;handles.dataNoTarget(i,:)];
394     end
395 end
396
397
398 set(handles.popupVarClass,'string',handles.namesNoTarget);
399
400 setappdata(0,'dataClass1',handles.dataClass1);
401 setappdata(0,'dataClass2',handles.dataClass2);
402 setappdata(0,'selectedClass',handles.selectedClass);
403 setappdata(0,'dataNoTarget',handles.dataNoTarget);
404 setappdata(0,'namesNoTarget',handles.namesNoTarget);
405 setappdata(0,'target',handles.target);
406 setappdata(0,'textClass0',handles.textClass0);
407 setappdata(0,'textClass1',handles.textClass1);
408 guidata(hObject, handles);
409
410
411 % --- Executes during object creation, after setting all properties.
412 function selectClass_CreateFcn(hObject, eventdata, handles)
413 % hObject    handle to selectClass (see GCBO)
414 % eventdata  reserved - to be defined in a future version of MATLAB
415 % handles    empty - handles not created until after all CreateFcns called
416
417 % Hint: popupmenu controls usually have a white background on Windows.
418 %       See ISPC and COMPUTER.
419 if ispc && isequal(get(hObject,'BackgroundColor'),...
420                    get(0,'defaultUicontrolBackgroundColor'))
421     set(hObject,'BackgroundColor','white');
422 end
423
424
```

```
425 % --- Executes on button press in selectVarsLogit.
426 function selectVarsLogit_Callback(hObject, eventdata, handles)
427 % hObject    handle to selectVarsLogit (see GCBO)
428 % eventdata  reserved - to be defined in a future version of MATLAB
429 % handles    structure with handles and user data (see GUIDATA)
430 seleccionarVarsLogit;
```

A.2. Media

```
1  function varargout = Media(varargin)
2  % MEDIA MATLAB code for Media.fig
3  %      MEDIA, by itself, creates a new MEDIA or raises the existing
4  %      singleton*.
5  %
6  %      H = MEDIA returns the handle to a new MEDIA or the handle to
7  %      the existing singleton*.
8  %
9  %      MEDIA('CALLBACK',hObject,eventData,handles,...) calls the local
10 %      function named CALLBACK in MEDIA.M with the given input arguments.
11 %
12 %      MEDIA('Property','Value',...) creates a new MEDIA or raises the
13 %      existing singleton*.  Starting from the left, property value pairs
   are
14 %      applied to the GUI before Media_OpeningFcn gets called.  An
15 %      unrecognized property name or invalid value makes property
   application
16 %      stop.  All inputs are passed to Media_OpeningFcn via varargin.
17 %
18 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
   one
19 %      instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help Media
24
25 % Last Modified by GUIDE v2.5 04-May-2017 21:26:07
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
30                    'gui_Singleton',  gui_Singleton, ...
31                    'gui_OpeningFcn', @Media_OpeningFcn, ...
32                    'gui_OutputFcn',  @Media_OutputFcn, ...
33                    'gui_LayoutFcn',  [] , ...
34                    'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
```

```
43 end
44 % End initialization code - DO NOT EDIT
45
46
47 % --- Executes just before Media is made visible.
48 function Media_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to Media (see VARARGIN)
54
55
56 h = findobj('Tag','main'); %h tiene el property inspector
57 if ~isempty(h)
58     dataClass1 = getappdata(0,'dataClass1');
59     dataClass2 = getappdata(0,'dataClass2');
60     names = getappdata(0,'names');
61 end
62
63 [m,n] = size(dataClass1);
64 [mH,nH] = size(dataClass2);
65 handles.names=names;
66
67  woman_mean = zeros(1,n);
68  man_mean = zeros(1,nH);
69  handles.total_mean = zeros(1,n);
70
71 for j = 2:n
72     woman_mean(j) = mean(dataClass1(:,j),'omitnan');
73     man_mean(j) = mean(dataClass2(:,j), 'omitnan');
74     handles.total_mean(j) = abs(woman_mean(j) - man_mean(j));
75 end
76
77 set(handles.popupmenu,'string',{'Parte1 ';'Parte2 ';'Parte3 ';'Parte4 ';'
    Parte5 ';'Parte6 ';'Parte7 ';'Parte8 ';
78     'Parte9 ';'Parte10';'Parte11';'Parte12';'Parte13';'Parte14';'Parte15';
       'Parte16'});
79
80 handles.listaUt = handles.total_mean;
81 handles.namesUt = handles.names;
82 axes(handles.axes1);
83 incremento = 24;
84 stem(2:(2+incremento), handles.listaUt(2:(2+incremento)), 'b');
85 set(gca, 'XTick',2:(2+incremento), 'XTickLabel',names(2:(2+incremento)));
86 set(gca, 'XTickLabelRotation', 60);
87 ylim([-1 1]);
88
89 % Choose default command line output for Media
90 handles.output = hObject;
91
92 % Update handles structure
93 guidata(hObject, handles);
94
95 % UIWAIT makes Media wait for user response (see UIRESUME)
96 % uiwait(handles.figure1);
```

```
97
98
99  % --- Outputs from this function are returned to the command line.
100 function varargout = Media_OutputFcn(hObject, eventdata, handles)
101 % varargout  cell array for returning output args (see VARARGOUT);
102 % hObject    handle to figure
103 % eventdata  reserved - to be defined in a future version of MATLAB
104 % handles    structure with handles and user data (see GUIDATA)
105
106 % Get default command line output from handles structure
107 varargout{1} = handles.output;
108
109
110 % --- Executes on selection change in popupmenu.
111 function popupmenu_Callback(hObject, eventdata, handles)
112 % hObject    handle to popupmenu (see GCBO)
113 % eventdata  reserved - to be defined in a future version of MATLAB
114 % handles    structure with handles and user data (see GUIDATA)
115
116 % Hints: contents = cellstr(get(hObject,'String')) returns popupmenu
       contents as cell array
117 %        contents{get(hObject,'Value')} returns selected item from
       popupmenu
118
119 incremento = 24;
120 indice=0;
121 [m,n] = size(handles.listaUt);
122
123 val = get(hObject,'Value');
124 switch val
125     case 1
126         indice=1;
127     case 2
128         indice=2;
129     case 3
130         indice=3;
131     case 4
132         indice=4;
133     case 5
134         indice=5;
135     case 6
136         indice=6;
137     case 7
138         indice=7;
139     case 8
140         indice=8;
141     case 9
142         indice=9;
143     case 10
144         indice=10;
145     case 11
146         indice=11;
147     case 12
148         indice=12;
149     case 13
150         indice=13;
```

```
151     case 14
152         indice=14;
153     case 15
154         indice=15;
155     case 16
156         indice=16;
157 end
158 
159     stem(((indice-1)*incremento + 1):min((((indice-1)*incremento + 1)+
           incremento),n), handles.listaUt(((indice-1)*incremento + 1):min((((
           indice-1)*incremento + 1)+incremento),n)), 'b');
160     set(gca, 'XTick',((indice-1)*incremento + 1):min((((indice-1)*
           incremento + 1)+incremento),n), 'XTickLabel',handles.namesUt(((
           indice-1)*incremento + 1):min(((indice-1)*incremento + 1)+
           incremento,n)));
161     set(gca, 'XTickLabelRotation', 60);
162     ylim([-1 1]);
163 
164 guidata(hObject,handles);
165 
166 % --- Executes during object creation, after setting all properties.
167 function popupmenu_CreateFcn(hObject, eventdata, handles)
168 % hObject    handle to popupmenu (see GCBO)
169 % eventdata  reserved - to be defined in a future version of MATLAB
170 % handles    empty - handles not created until after all CreateFcns called
171 
172 % Hint: popupmenu controls usually have a white background on Windows.
173 %       See ISPC and COMPUTER.
174 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
       defaultUicontrolBackgroundColor'))
175     set(hObject,'BackgroundColor','white');
176 end
177 
178 
179 
180 function numText_Callback(hObject, eventdata, handles)
181 % hObject    handle to numText (see GCBO)
182 % eventdata  reserved - to be defined in a future version of MATLAB
183 % handles    structure with handles and user data (see GUIDATA)
184 
185 % Hints: get(hObject,'String') returns contents of numText as text
186 %        str2double(get(hObject,'String')) returns contents of numText as
       a double
187 
188 
189 % --- Executes during object creation, after setting all properties.
190 function numText_CreateFcn(hObject, eventdata, handles)
191 % hObject    handle to numText (see GCBO)
192 % eventdata  reserved - to be defined in a future version of MATLAB
193 % handles    empty - handles not created until after all CreateFcns called
194 
195 % Hint: edit controls usually have a white background on Windows.
196 %       See ISPC and COMPUTER.
197 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
       defaultUicontrolBackgroundColor'))
198     set(hObject,'BackgroundColor','white');
```

```
199 end
200
201
202 % --- Executes on button press in numMedias.
203 function numMedias_Callback(hObject, eventdata, handles)
204 % hObject    handle to numMedias (see GCBO)
205 % eventdata  reserved - to be defined in a future version of MATLAB
206 % handles    structure with handles and user data (see GUIDATA)
207 str=get(handles.numText, 'string');
208 num=str2num(str);
209 if num <= 0 || num > 367
210     num=367;
211 end
212 [listaOrd,nombresOrd]=ordenacion_maximos(num, handles);
213 particiones = floor(num/24)+1;
214 listaPopUp=cell(1,particiones);
215 for i = 1:particiones
216     listaPopUp{i}=strcat('Parte',int2str(i));
217 end
218 set(handles.popupmenu,'Value',1);
219 set(handles.popupmenu,'string',listaPopUp);
220
221 indice=1;
222 incremento=24;
223 stem(((indice-1)*incremento + 1):min((((indice-1)*incremento + 1)+
    incremento),num), listaOrd(((indice-1)*incremento + 1):min((((indice-1)
    *incremento + 1)+incremento),num)), 'b');
224 set(gca, 'XTick',((indice-1)*incremento + 1):min((((indice-1)*incremento +
     1)+incremento),num), 'XTickLabel',nombresOrd(((indice-1)*incremento +
    1):min(((indice-1)*incremento + 1)+incremento,num)));
225 set(gca, 'XTickLabelRotation', 60);
226 ylim([-1 1]);
227
228 handles.listaUt = listaOrd(1:num);
229 handles.namesUt = nombresOrd(1:num);
230 guidata(hObject, handles);
231
232
233 function [lista,nombresOrd] = ordenacion_maximos(num, handles)
234     lista = handles.total_mean;
235     nombresOrd = handles.names;
236     n = size(lista,2);
237     for i=1:num
238         maxIndex = i;
239         maxValue = lista(i);
240         for j=i+1:n
241             if lista(j) > maxValue;
242                 maxIndex = j;
243                 maxValue = lista(j);
244             end
245         end
246         aux= lista(i);
247         lista(i)=lista(maxIndex);
248         lista(maxIndex)=aux;
249         aux=nombresOrd(i);
250         nombresOrd(i)=nombresOrd(maxIndex);
```

```
251         nombresOrd(maxIndex)=aux;
252     end
```

A.3. Desviación Típica

```
1  function varargout = Desviacion_Tipica(varargin)
2  % VARIANZA MATLAB code for Varianza.fig
3  %      VARIANZA, by itself, creates a new VARIANZA or raises the existing
4  %      singleton*.
5  %
6  %      H = VARIANZA returns the handle to a new VARIANZA or the handle to
7  %      the existing singleton*.
8  %
9  %      VARIANZA('CALLBACK',hObject,eventData,handles,...) calls the local
10 %      function named CALLBACK in VARIANZA.M with the given input
   arguments.
11 %
12 %      VARIANZA('Property','Value',...) creates a new VARIANZA or raises
   the
13 %      existing singleton*.  Starting from the left, property value pairs
   are
14 %      applied to the GUI before Varianza_OpeningFcn gets called.  An
15 %      unrecognized property name or invalid value makes property
   application
16 %      stop.  All inputs are passed to Varianza_OpeningFcn via varargin.
17 %
18 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
   one
19 %      instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help Varianza
24
25 % Last Modified by GUIDE v2.5 10-May-2017 18:12:13
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
30                    'gui_Singleton',  gui_Singleton, ...
31                    'gui_OpeningFcn', @Desviacion_Tipica_OpeningFcn, ...
32                    'gui_OutputFcn',  @Desviacion_Tipica_OutputFcn, ...
33                    'gui_LayoutFcn',  [] , ...
34                    'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
43 end
44 % End initialization code - DO NOT EDIT
```

```
45
46
47  % --- Executes just before Varianza is made visible.
48  function Desviacion_Tipica_OpeningFcn(hObject, eventdata, handles,
        varargin)
49  % This function has no output args, see OutputFcn.
50  % hObject    handle to figure
51  % eventdata  reserved - to be defined in a future version of MATLAB
52  % handles    structure with handles and user data (see GUIDATA)
53  % varargin   command line arguments to Varianza (see VARARGIN)
54
55  h = findobj('Tag','main'); %h tiene el property inspector
56  if ~isempty(h)
57      dataClass1 = getappdata(0,'dataClass1');
58      dataClass2 = getappdata(0,'dataClass2');
59      names = getappdata(0,'names');
60      textClass0 = getappdata(0,'textClass0');
61      textClass1 = getappdata(0,'textClass1');
62  end
63
64  [m,n] = size(dataClass1);
65  [mH,nH] = size(dataClass2);
66
67  handles.deviation_Class1 = zeros(1,n-1);
68  handles.deviation_Class2 = zeros(1,nH-1);
69  handles.names = names;
70
71  [m,n] = size(dataClass1);
72  [mH,nH] = size(dataClass2);
73
74
75  for j = 2:n
76      handles.deviation_Class1(j) = std(dataClass1(:,j),'omitnan');
77      handles.deviation_Class2(j) = std(dataClass2(:,j),'omitnan');
78  end
79
80  set(handles.popupmenu,'string',['Parte1 ';'Parte2 ';'Parte3 ';'Parte4 ';'
        Parte5 ';'Parte6 ';'Parte7 ';'Parte8 ';
81                                  'Parte9 ';'Parte10';'Parte11';'Parte12';'
                                       Parte13';'Parte14';'Parte15';'Parte16'
                                       ]);
82  set(handles.buttonClass1,'Value',1)
83  handles.listaUt = handles.deviation_Class1;
84  handles.namesUt = names;
85
86  axes(handles.axes1);
87  incremento = 24;
88  stem(2:(2+incremento), handles.deviation_Class1(2:(2+incremento)), 'b');
89  set(gca, 'XTick',2:(2+incremento), 'XTickLabel',names(2:(2+incremento)));
90  set(gca, 'XTickLabelRotation', 60);
91  ylim([-1 1]);
92
93  set(handles.buttonClass1,'string',textClass0);
94  set(handles.buttonClass2,'string',textClass1);
95
96  % Choose default command line output for Varianza
```

```
97  handles.output = hObject;
98
99  % Update handles structure
100 guidata(hObject, handles);
101
102 % UIWAIT makes Varianza wait for user response (see UIRESUME)
103 % uiwait(handles.figure1);
104
105
106 % --- Outputs from this function are returned to the command line.
107 function varargout = Desviacion_Tipica_OutputFcn(hObject, eventdata,
        handles)
108 % varargout  cell array for returning output args (see VARARGOUT);
109 % hObject    handle to figure
110 % eventdata  reserved - to be defined in a future version of MATLAB
111 % handles    structure with handles and user data (see GUIDATA)
112
113 % Get default command line output from handles structure
114 varargout{1} = handles.output;
115
116
117 % --- Executes on selection change in popupmenu.
118 function popupmenu_Callback(hObject, eventdata, handles)
119 % hObject    handle to popupmenu (see GCBO)
120 % eventdata  reserved - to be defined in a future version of MATLAB
121 % handles    structure with handles and user data (see GUIDATA)
122
123 incremento = 24;
124 clase = get(handles.buttonClass1,'Value');
125 indice=0;
126 n = size(handles.listaUt,2);
127
128 val = get(hObject,'Value');
129 switch val
130     case 1
131         indice=1;
132     case 2
133         indice=2;
134     case 3
135         indice=3;
136     case 4
137         indice=4;
138     case 5
139         indice=5;
140     case 6
141         indice=6;
142     case 7
143         indice=7;
144     case 8
145         indice=8;
146     case 9
147         indice=9;
148     case 10
149         indice=10;
150     case 11
151         indice=11;
```

```
152     case 12
153         indice=12;
154     case 13
155         indice=13;
156     case 14
157         indice=14;
158     case 15
159         indice=15;
160     case 16
161         indice=16;
162 end
163
164 % if ((indice-1)*incremento + 2)+incremento > n
165 %    incremento = n-(indice-1)*incremento + 2;
166 % end
167
168 %if clase == 1
169     stem(((indice-1)*incremento + 2):min((((indice-1)*incremento + 2)+
            incremento),n), handles.listaUt(((indice-1)*incremento + 2):min((((
            indice-1)*incremento + 2)+incremento),n)), 'b');
170     set(gca, 'XTick',((indice-1)*incremento + 2):min((((indice-1)*
            incremento + 2)+incremento),n), 'XTickLabel',handles.namesUt(((
            indice-1)*incremento + 2):min(((indice-1)*incremento + 2)+
            incremento,n)));
171     set(gca, 'XTickLabelRotation', 60);
172     ylim([-1 1]);
173 % else
174 %     stem(((indice-1)*incremento + 2):min((((indice-1)*incremento + 2)+
    incremento),n), handles.deviation_Class2(((indice-1)*incremento + 2):
    min((((indice-1)*incremento + 2)+incremento),n)), 'b');
175 %     set(gca, 'XTick',((indice-1)*incremento + 2):min((((indice-1)*
    incremento + 2)+incremento),n), 'XTickLabel',handles.names(((indice-1)*
    incremento + 2):min(((indice-1)*incremento + 2)+incremento,n)));
176 %     set(gca, 'XTickLabelRotation', 60);
177 %     ylim([-1 1]);
178 % end
179
180 %
181 % axes(handles.axes1);
182 % incremento = 24;
183 % handles.partes = [incremento,2*incremento,3*incremento,4*incremento];
184 % stem(2:(2+handles.partes(1)), handles.deviation_Class1(2:(2+incremento))
    , 'b');
185 % set(gca, 'XTick',2:(2+incremento), 'XTickLabel',names(2:(2+incremento)))
    ;
186 % set(gca, 'XTickLabelRotation', 60);
187 % ylim([-1 1]);
188
189 guidata(hObject,handles);
190
191 % Hints: contents = cellstr(get(hObject,'String')) returns popupmenu
    contents as cell array
192 %        contents{get(hObject,'Value')} returns selected item from
    popupmenu
193
194
```

```
195 % --- Executes during object creation, after setting all properties.
196 function popupmenu_CreateFcn(hObject, eventdata, handles)
197 % hObject    handle to popupmenu (see GCBO)
198 % eventdata  reserved - to be defined in a future version of MATLAB
199 % handles    empty - handles not created until after all CreateFcns called
200
201 % Hint: popupmenu controls usually have a white background on Windows.
202 %       See ISPC and COMPUTER.
203 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
204     set(hObject,'BackgroundColor','white');
205 end
206
207
208
209 function numVar_Callback(hObject, eventdata, handles)
210 % hObject    handle to numVar (see GCBO)
211 % eventdata  reserved - to be defined in a future version of MATLAB
212 % handles    structure with handles and user data (see GUIDATA)
213
214 % Hints: get(hObject,'String') returns contents of numVar as text
215 %        str2double(get(hObject,'String')) returns contents of numVar as a
    double
216
217
218 % --- Executes during object creation, after setting all properties.
219 function numVar_CreateFcn(hObject, eventdata, handles)
220 % hObject    handle to numVar (see GCBO)
221 % eventdata  reserved - to be defined in a future version of MATLAB
222 % handles    empty - handles not created until after all CreateFcns called
223
224 % Hint: edit controls usually have a white background on Windows.
225 %       See ISPC and COMPUTER.
226 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
227     set(hObject,'BackgroundColor','white');
228 end
229
230
231 % --- Executes on button press in numVarButton.
232 function numVarButton_Callback(hObject, eventdata, handles)
233 % hObject    handle to numVarButton (see GCBO)
234 % eventdata  reserved - to be defined in a future version of MATLAB
235 % handles    structure with handles and user data (see GUIDATA)
236 str=get(handles.numVar, 'string');
237 num=str2num(str);
238 if num <= 0 || num > 367
239     num=367;
240 end
241 [listaOrd,nombresOrd]=ordenacion_maximos(num, handles);
242 particiones = floor(num/24) + 1;
243 listaPopUp=cell(1,particiones);
244 for i = 1:particiones
245     listaPopUp{i}=strcat('Parte',int2str(i));
246 end
247 set(handles.popupmenu,'string',listaPopUp);
```

```
248 set(handles.popupmenu,'Value',1);
249 indice=1;
250 incremento=24;
251 stem(((indice-1)*incremento + 1):min((((indice-1)*incremento + 1)+
    incremento),num), listaOrd(((indice-1)*incremento + 1):min((((indice-1)
    *incremento + 1)+incremento),num)), 'b');
252 set(gca, 'XTick',((indice-1)*incremento + 1):min((((indice-1)*incremento +
     1)+incremento),num), 'XTickLabel',nombresOrd(((indice-1)*incremento +
    1):min(((indice-1)*incremento + 1)+incremento,num)));
253 set(gca, 'XTickLabelRotation', 60);
254 ylim([-1 1]);
255
256 handles.listaUt = listaOrd(1:num);
257 handles.namesUt = nombresOrd(1:num);
258 guidata(hObject, handles);
259
260
261 function [lista,nombresOrd] = ordenacion_maximos(num, handles)
262     clase = get(handles.buttonClass1,'Value');
263     if clase == 1
264         lista=handles.deviation_Class1;
265     else
266         lista=handles.deviation_Class2;
267     end
268     nombresOrd = handles.names;
269     n = size(lista,2);
270     for i=1:num
271         maxIndex = i;
272         maxValue = lista(i);
273         for j=i+1:n
274             if lista(j) > maxValue;
275                 maxIndex = j;
276                 maxValue = lista(j);
277             end
278         end
279         aux= lista(i);
280         lista(i)=lista(maxIndex);
281         lista(maxIndex)=aux;
282         aux=nombresOrd(i);
283         nombresOrd(i)=nombresOrd(maxIndex);
284         nombresOrd(maxIndex)=aux;
285     end
```

A.4. Regresión Lineal y Correlación

```
1 function varargout = Regresion_lineal(varargin)
2 % COVARIANZA MATLAB code for Covarianza.fig
3 %      COVARIANZA, by itself, creates a new COVARIANZA or raises the
    existing
4 %      singleton*.
5 %
6 %      H = COVARIANZA returns the handle to a new COVARIANZA or the handle
     to
7 %      the existing singleton*.
```

```
8  %
9  %      COVARIANZA('CALLBACK',hObject,eventData,handles,...) calls the
       local
10 %      function named CALLBACK in COVARIANZA.M with the given input
       arguments.
11 %
12 %      COVARIANZA('Property','Value',...) creates a new COVARIANZA or
       raises the
13 %      existing singleton*.  Starting from the left, property value pairs
       are
14 %      applied to the GUI before Covarianza_OpeningFcn gets called.  An
15 %      unrecognized property name or invalid value makes property
       application
16 %      stop.  All inputs are passed to Covarianza_OpeningFcn via varargin.
17 %
18 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
       one
19 %      instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help Covarianza
24
25 % Last Modified by GUIDE v2.5 05-Aug-2017 12:09:44
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
30                    'gui_Singleton',  gui_Singleton, ...
31                    'gui_OpeningFcn', @Regresion_lineal_OpeningFcn, ...
32                    'gui_OutputFcn',  @Regresion_lineal_OutputFcn, ...
33                    'gui_LayoutFcn',  [] , ...
34                    'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
43 end
44 % End initialization code - DO NOT EDIT
45
46
47 % --- Executes just before Covarianza is made visible.
48 function Regresion_lineal_OpeningFcn(hObject, eventdata, handles, varargin
       )
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to Covarianza (see VARARGIN)
54
55 h = findobj('Tag','main'); %h tiene el property inspector
56 if ~isempty(h)
```

```
57      dataClass1 = getappdata(0,'dataClass1');
58      dataClass2 = getappdata(0,'dataClass2');
59      data = getappdata(0,'data');
60      names = getappdata(0,'names');
61      textClass0 = getappdata(0,'textClass0');
62      textClass1 = getappdata(0,'textClass1');
63  end
64
65  [m,n] = size(data);
66  handles.names=names;
67  handles.data=data;
68  handles.dataClass1 = dataClass1;
69  handles.dataClass2 = dataClass2;
70  handles.varx = 0;
71  handles.vary = 0;
72
73  set(handles.Var1,'string',names);
74  set(handles.Var2,'string',names);
75
76  handles.selectedVar1 = 1;
77  handles.selectedVar2 = 1;
78
79  A=data(:,1)';
80  B=data(:,1)';
81  Aisnan = ~isnan(A);
82  Bisnan = ~isnan(B);
83  validos = Aisnan & Bisnan;
84  A=A(validos);
85  B=B(validos);
86  C=polyfit(A,B,1);
87  axes(handles.axes1);
88  x=0:0.01:1;
89  y=C(1)*x + C(2);
90  plot(x,y,'r');
91  hold on;
92  plot(A,B,'bo');
93  hold off;
94  coef = corrcoef(A,B);
95  set(handles.coefText,'String',coef(2,1));
96
97  % Choose default command line output for Covarianza
98  handles.output = hObject;
99
100 % Update handles structure
101 guidata(hObject, handles);
102
103 % UIWAIT makes Covarianza wait for user response (see UIRESUME)
104 % uiwait(handles.figure1);
105
106
107 % --- Outputs from this function are returned to the command line.
108 function varargout = Regresion_lineal_OutputFcn(hObject, eventdata,
        handles)
109 % varargout  cell array for returning output args (see VARARGOUT);
110 % hObject    handle to figure
111 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
112 % handles    structure with handles and user data (see GUIDATA)
113
114 % Get default command line output from handles structure
115 varargout{1} = handles.output;
116
117
118 % --- Executes on selection change in Var1.
119 function Var1_Callback(hObject, eventdata, handles)
120 % hObject    handle to Var1 (see GCBO)
121 % eventdata  reserved - to be defined in a future version of MATLAB
122 % handles    structure with handles and user data (see GUIDATA)
123
124 handles.selectedVar1 = get(hObject,'Value');
125 A=handles.data(:,handles.selectedVar1)';
126 B=handles.data(:,handles.selectedVar2)';
127 Aisnan = ~isnan(A);
128 Bisnan = ~isnan(B);
129 validos = Aisnan & Bisnan;
130 A=A(validos);
131 B=B(validos);
132 C=polyfit(A,B,1);
133 axes(handles.axes1);
134 x=0:0.01:1;
135 y=C(1)*x + C(2);
136 plot(x,y,'r');
137 hold on;
138 plot(A,B,'bo');
139 hold off;
140 coef = corrcoef(A,B);
141 set(handles.coefText,'String',coef(2,1));
142
143 guidata(hObject, handles);
144
145 % Hints: contents = cellstr(get(hObject,'String')) returns Var1 contents
      as cell array
146 %        contents{get(hObject,'Value')} returns selected item from Var1
147
148
149 % --- Executes during object creation, after setting all properties.
150 function Var1_CreateFcn(hObject, eventdata, handles)
151 % hObject    handle to Var1 (see GCBO)
152 % eventdata  reserved - to be defined in a future version of MATLAB
153 % handles    empty - handles not created until after all CreateFcns called
154
155 % Hint: popupmenu controls usually have a white background on Windows.
156 %       See ISPC and COMPUTER.
157 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
      defaultUicontrolBackgroundColor'))
158     set(hObject,'BackgroundColor','white');
159 end
160
161
162 % --- Executes on selection change in Var2.
163 function Var2_Callback(hObject, eventdata, handles)
164 % hObject    handle to Var2 (see GCBO)
165 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
166 % handles    structure with handles and user data (see GUIDATA)
167
168 handles.selectedVar2 = get(hObject,'Value');
169 A=handles.data(:,handles.selectedVar1)';
170 B=handles.data(:,handles.selectedVar2)';
171 Aisnan = ~isnan(A);
172 Bisnan = ~isnan(B);
173 validos = Aisnan & Bisnan;
174 A=A(validos);
175 B=B(validos);
176 C=polyfit(A,B,1);
177 axes(handles.axes1);
178 x=0:0.01:1;
179 y=C(1)*x + C(2);
180 plot(x,y,'r');
181 hold on;
182 plot(A,B,'bo');
183 hold off;
184 coef = corrcoef(A,B);
185 set(handles.coefText,'String',coef(2,1));
186
187 guidata(hObject, handles);
188
189 % Hints: contents = cellstr(get(hObject,'String')) returns Var2 contents
       as cell array
190 %        contents{get(hObject,'Value')} returns selected item from Var2
191
192
193 % --- Executes during object creation, after setting all properties.
194 function Var2_CreateFcn(hObject, eventdata, handles)
195 % hObject    handle to Var2 (see GCBO)
196 % eventdata  reserved - to be defined in a future version of MATLAB
197 % handles    empty - handles not created until after all CreateFcns called
198
199 % Hint: popupmenu controls usually have a white background on Windows.
200 %       See ISPC and COMPUTER.
201 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
       defaultUicontrolBackgroundColor'))
202     set(hObject,'BackgroundColor','white');
203 end
```

A.5. Regresión Logística

```
1  function varargout = seleccionarVarsLogit(varargin)
2  % SELECCIONARVARSLOGIT MATLAB code for seleccionarVarsLogit.fig
3  %      SELECCIONARVARSLOGIT, by itself, creates a new SELECCIONARVARSLOGIT
4  %      or raises the existing singleton*.
5  %
6  %      H = SELECCIONARVARSLOGIT returns the handle to a new
7  %      SELECCIONARVARSLOGIT or the handle to the existing singleton*.
8  %
9  %      SELECCIONARVARSLOGIT('CALLBACK',hObject,eventData,handles,...)
       calls
10 %      the local function named CALLBACK in SELECCIONARVARSLOGIT.M with
```

```
    the
%       given input arguments.
%
%       SELECCIONARVARSLOGIT('Property','Value',...) creates a new
%       SELECCIONARVARSLOGIT or raises the existing singleton*.  Starting
%       from the left, property value pairs are applied to the GUI before
%       seleccionarVarsLogit_OpeningFcn gets called.  An unrecognized
%       property name or invalid value makes property application stop. All
%       inputs are passed to seleccionarVarsLogit_OpeningFcn via varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
    one
%       instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above word to modify the response to help seleccionarVarsLogit

% Last Modified by GUIDE v2.5 18-Aug-2017 20:00:12

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @seleccionarVarsLogit_OpeningFcn, ...
                   'gui_OutputFcn',  @seleccionarVarsLogit_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before seleccionarVarsLogit is made visible.
function seleccionarVarsLogit_OpeningFcn(hObject, eventdata, handles,...
                                         varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to seleccionarVarsLogit (see VARARGIN)

h = findobj('Tag','main'); %h tiene el property inspector
if ~isempty(h)
    dataNoTarget = getappdata(0,'dataNoTarget');
    namesNoTarget = getappdata(0,'namesNoTarget');
    target = getappdata(0,'target');
end

```

```
65 handles.dataNoTarget = dataNoTarget;
66 handles.namesNoTarget = namesNoTarget;
67 handles.target = target;
68
69 % Choose default command line output for seleccionarVarsLogit
70 handles.output = hObject;
71
72 % Update handles structure
73 guidata(hObject, handles);
74
75 % UIWAIT makes seleccionarVarsLogit wait for user response (see UIRESUME)
76 % uiwait(handles.figure1);
77
78
79 % --- Outputs from this function are returned to the command line.
80 function varargout = seleccionarVarsLogit_OutputFcn(hObject, eventdata,...
81                                                     handles)
82 % varargout  cell array for returning output args (see VARARGOUT);
83 % hObject    handle to figure
84 % eventdata  reserved - to be defined in a future version of MATLAB
85 % handles    structure with handles and user data (see GUIDATA)
86
87 % Get default command line output from handles structure
88 varargout{1} = handles.output;
89
90
91 % --- Executes on selection change in varList.
92 function varList_Callback(hObject, eventdata, handles)
93 % hObject    handle to varList (see GCBO)
94 % eventdata  reserved - to be defined in a future version of MATLAB
95 % handles    structure with handles and user data (see GUIDATA)
96
97 % Hints: contents = cellstr(get(hObject,'String')) returns varList
      contents
98 %        as cell array
99 %        contents{get(hObject,'Value')} returns selected item from varList
100
101
102 % --- Executes during object creation, after setting all properties.
103 function varList_CreateFcn(hObject, eventdata, handles)
104 % hObject    handle to varList (see GCBO)
105 % eventdata  reserved - to be defined in a future version of MATLAB
106 % handles    empty - handles not created until after all CreateFcns called
107
108 % Hint: listbox controls usually have a white background on Windows.
109 %       See ISPC and COMPUTER.
110 if ispc && isequal(get(hObject,'BackgroundColor'),...
111                    get(0,'defaultUicontrolBackgroundColor'))
112     set(hObject,'BackgroundColor','white');
113 end
114
115
116
117 function word_Callback(hObject, eventdata, handles)
118 % hObject    handle to word (see GCBO)
119 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
120 % handles    structure with handles and user data (see GUIDATA)
121
122 % Hints: get(hObject,'String') returns contents of word as word
123 %        str2double(get(hObject,'String')) returns contents of word as a
    double
124
125
126 % --- Executes during object creation, after setting all properties.
127 function word_CreateFcn(hObject, eventdata, handles)
128 % hObject    handle to word (see GCBO)
129 % eventdata  reserved - to be defined in a future version of MATLAB
130 % handles    empty - handles not created until after all CreateFcns called
131
132 % Hint: edit controls usually have a white background on Windows.
133 %       See ISPC and COMPUTER.
134 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
135     set(hObject,'BackgroundColor','white');
136 end
137
138
139 % --- Executes on button press in buscar.
140 function buscar_Callback(hObject, eventdata, handles)
141 % hObject    handle to buscar (see GCBO)
142 % eventdata  reserved - to be defined in a future version of MATLAB
143 % handles    structure with handles and user data (see GUIDATA)
144 str=get(handles.word, 'string');
145 names = get(handles.varList, 'string');
146 index = strfind(names,str);
147 pos=find(arrayfun(@(x) isequal(x,{[]}),index)==1);
148 names(pos) = [];
149 set(handles.varList,'String',names);
150
151 % --- Executes on button press in restaurar.
152 function restaurar_Callback(hObject, eventdata, handles)
153 % hObject    handle to restaurar (see GCBO)
154 % eventdata  reserved - to be defined in a future version of MATLAB
155 % handles    structure with handles and user data (see GUIDATA)
156 set(handles.varList,'String',handles.dataNamesL);
157 set(handles.word, 'string','');
158
159 % --- Executes on button press in ejecutar.
160 function ejecutar_Callback(hObject, eventdata, handles)
161
162 [m,n] = size(handles.dataNoTarget);
163 RandStream.setGlobalStream(RandStream('mt19937ar','seed',1));
164 data = [handles.dataNoTarget handles.target];
165 matrizCorr = corrcoef(handles.dataNoTarget);
166 pos = [];
167 for i=1:n-1
168     result = find(matrizCorr(i,(i+1):n) > 0.9);
169     if ~isempty(result)
170        pos = [pos result+i];
171     end
172 end
173
```

```
174 handles.dataNoTarget(:,pos) = [];
175 handles.namesNoTarget(pos) = [];
176
177 pos=[];
178 for i=1:size(handles.dataNoTarget,2)
179     if corrcoef(handles.dataNoTarget(:,i),handles.target) > 0.89
180         pos = [pos i];
181     end
182 end
183
184 handles.dataNoTarget(:,pos) = [];
185 handles.namesNoTarget(pos) = [];
186
187 [m,n] = size(handles.dataNoTarget);
188 handles.dataL=[];
189 handles.dataNamesL=[];
190 xx = linspace(-1.5,2);
191 handles.yfit = [];
192 handles.pvalor=[];
193 %[b,dev,stats] = glmfit(handles.dataNoTarget,handles.target,'binomial','
       link','logit');
194 tic
195 for i=1:n
196     [b,dev,stats] = glmfit(handles.dataNoTarget(:,i),handles.target,'
          binomial','link','logit');
197     if stats.p <= 0.05
198         handles.dataL=[handles.dataL handles.dataNoTarget(:,i)];
199         handles.dataNamesL = [handles.dataNamesL handles.namesNoTarget(i)
              ];
200         handles.yfit=[handles.yfit glmval(b,xx,'logit')];
201         handles.pvalor=[handles.pvalor stats.p(2)];
202     end
203 end
204 toc
205 aux=[handles.dataNamesL;num2cell(handles.pvalor)];
206 cellSort=transpose(sortrows(transpose(aux),2));
207 handles.dataNamesL = cellSort(1,:);
208 handles.pvalor = cell2mat(cellSort(2,:));
209
210 set(handles.varList,'String',handles.dataNamesL);
211 set(handles.selectedVars,'string',handles.dataNamesL);
212
213 guidata(hObject, handles);
214
215 % hObject    handle to ejecutar (see GCBO)
216 % eventdata  reserved - to be defined in a future version of MATLAB
217 % handles    structure with handles and user data (see GUIDATA)
218
219
220 % --- Executes on selection change in selectedVars.
221 function selectedVars_Callback(hObject, eventdata, handles)
222 % hObject    handle to selectedVars (see GCBO)
223 % eventdata  reserved - to be defined in a future version of MATLAB
224 % handles    structure with handles and user data (see GUIDATA)
225
226 % Hints: contents = cellstr(get(hObject,'String')) returns selectedVars
```

```
    contents as cell array
227 %        contents{get(hObject,'Value')} returns selected item from
    selectedVars
228 selectedVar = get(hObject,'Value');
229 axes(handles.axes1);
230 xx = linspace(-1.5,2);
231 plot(handles.dataL(:,selectedVar),handles.target,'o',xx,handles.yfit(:,
    selectedVar),'-');
232 set(handles.pvalueText,'String',handles.pvalor(selectedVar));
233
234 % --- Executes during object creation, after setting all properties.
235 function selectedVars_CreateFcn(hObject, eventdata, handles)
236 % hObject    handle to selectedVars (see GCBO)
237 % eventdata  reserved - to be defined in a future version of MATLAB
238 % handles    empty - handles not created until after all CreateFcns called
239
240 % Hint: popupmenu controls usually have a white background on Windows.
241 %       See ISPC and COMPUTER.
242 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
243     set(hObject,'BackgroundColor','white');
244 end
```

A.6. Lasso

```
1  function varargout = seleccionarVars(varargin)
2  % SELECCIONARVARS MATLAB code for seleccionarVars.fig
3  %      SELECCIONARVARS, by itself, creates a new SELECCIONARVARS or raises
       the existing
4  %      singleton*.
5  %
6  %      H = SELECCIONARVARS returns the handle to a new SELECCIONARVARS or
       the handle to
7  %      the existing singleton*.
8  %
9  %      SELECCIONARVARS('CALLBACK',hObject,eventData,handles,...) calls the
       local
10 %      function named CALLBACK in SELECCIONARVARS.M with the given input
       arguments.
11 %
12 %      SELECCIONARVARS('Property','Value',...) creates a new
       SELECCIONARVARS or raises the
13 %      existing singleton*.  Starting from the left, property value pairs
       are
14 %      applied to the GUI before seleccionarVars_OpeningFcn gets called.
       An
15 %      unrecognized property name or invalid value makes property
       application
16 %      stop.  All inputs are passed to seleccionarVars_OpeningFcn via
       varargin.
17 %
18 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
       one
19 %      instance to run (singleton)".
```

```
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help seleccionarVars
24
25 % Last Modified by GUIDE v2.5 03-Aug-2017 19:28:18
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
30                    'gui_Singleton',  gui_Singleton, ...
31                    'gui_OpeningFcn', @seleccionarVars_OpeningFcn, ...
32                    'gui_OutputFcn',  @seleccionarVars_OutputFcn, ...
33                    'gui_LayoutFcn',  [] , ...
34                    'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
43 end
44 % End initialization code - DO NOT EDIT
45
46
47 % --- Executes just before seleccionarVars is made visible.
48 function seleccionarVars_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to seleccionarVars (see VARARGIN)
54
55 % Choose default command line output for seleccionarVars
56
57 h = findobj('Tag','main'); %h tiene el property inspector
58 if ~isempty(h)
59     dataNoTarget = getappdata(0,'dataNoTarget');
60     namesNoTarget = getappdata(0,'namesNoTarget');
61     target = getappdata(0,'target');
62 end
63
64 handles.dataNoTarget = dataNoTarget;
65 handles.namesNoTarget = namesNoTarget;
66 handles.target = target;
67
68 handles.output = hObject;
69 % Update handles structure
70 guidata(hObject, handles);
71
72 % UIWAIT makes seleccionarVars wait for user response (see UIRESUME)
73 % uiwait(handles.figure1);
74
75
```

```
76 % --- Outputs from this function are returned to the command line.
77 function varargout = seleccionarVars_OutputFcn(hObject, eventdata, handles
       )
78 % varargout  cell array for returning output args (see VARARGOUT);
79 % hObject    handle to figure
80 % eventdata  reserved - to be defined in a future version of MATLAB
81 % handles    structure with handles and user data (see GUIDATA)
82
83 % Get default command line output from handles structure
84 varargout{1} = handles.output;
85
86
87 % --- Executes on button press in ejecutar.
88 function ejecutar_Callback(hObject, eventdata, handles)
89 % hObject    handle to ejecutar (see GCBO)
90 % eventdata  reserved - to be defined in a future version of MATLAB
91 % handles    structure with handles and user data (see GUIDATA)
92 [m,n] = size(handles.dataNoTarget);
93 RandStream.setGlobalStream(RandStream('mt19937ar','seed',1));
94 data = [handles.dataNoTarget handles.target];
95 matrizCorr = corrcoef(handles.dataNoTarget);
96 pos = [];
97 for i=1:n-1
98     result = find(matrizCorr(i,(i+1):n) > 0.9);
99     if ~isempty(result)
100        pos = [pos result+i];
101    end
102 end
103
104 handles.dataNoTarget(:,pos) = [];
105 handles.namesNoTarget(pos) = [];
106
107 pos=[];
108 for i=1:size(handles.dataNoTarget,2)
109     if corrcoef(handles.dataNoTarget(:,i),handles.target) > 0.89
110         pos = [pos i];
111     end
112 end
113
114 handles.dataNoTarget(:,pos) = [];
115 handles.namesNoTarget(pos) = [];
116
117 CV0 = cvpartition(handles.target,'KFold',7);
118 [B,FitInfo]=lasso(handles.dataNoTarget,handles.target,'CV',CV0,'
       PredictorNames',handles.namesNoTarget);
119 axes(handles.axes4)
120 lassoPlot(B,FitInfo,'PlotType','Lambda','XScale','log','PredictorNames',
       handles.namesNoTarget,'Parent',handles.axes4);
121 axes(handles.axes3)
122 lassoPlot(B,FitInfo,'PlotType','CV','Parent',handles.axes3);
123
124 pos=find(B(:,FitInfo.IndexMinMSE)==0);
125 handles.dataNoTarget(:,pos) = [];
126 handles.namesNoTarget(pos) = [];
127 posNo0=find(B(:,FitInfo.IndexMinMSE)~=0);
128 aux=[handles.namesNoTarget;num2cell(B(posNo0,FitInfo.IndexMinMSE)');
```

```
    num2cell(abs(B(posNo0,FitInfo.IndexMinMSE))')];
cellSort=transpose(sortrows(transpose(aux),3));
cellSort=fliplr(cellSort);
handles.namesNoTarget = cellSort(1,:);

set(handles.vars,'String',handles.namesNoTarget);


guidata(hObject, handles);

% --- Executes on selection change in vars.
function vars_Callback(hObject, eventdata, handles)
% hObject    handle to vars (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns vars contents
    as cell array
%        contents{get(hObject,'Value')} returns selected item from vars


% --- Executes during object creation, after setting all properties.
function vars_CreateFcn(hObject, eventdata, handles)
% hObject    handle to vars (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function word_Callback(hObject, eventdata, handles)
% hObject    handle to word (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of word as text
%        str2double(get(hObject,'String')) returns contents of word as a
    double


% --- Executes during object creation, after setting all properties.
function word_CreateFcn(hObject, eventdata, handles)
% hObject    handle to word (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
```

```
180     set(hObject,'BackgroundColor','white');
181 end
182
183
184 % --- Executes on button press in buscar.
185 function buscar_Callback(hObject, eventdata, handles)
186 % hObject    handle to buscar (see GCBO)
187 % eventdata  reserved - to be defined in a future version of MATLAB
188 % handles    structure with handles and user data (see GUIDATA)
189 str=get(handles.word, 'string');
190 names = get(handles.vars, 'string');
191 index = strfind(names,str);
192 pos=find(arrayfun(@(x) isequal(x,{[]}),index)==1);
193 names(pos) = [];
194 set(handles.vars,'String',names);
195
196 % --- Executes on button press in restaurar.
197 function restaurar_Callback(hObject, eventdata, handles)
198 % hObject    handle to restaurar (see GCBO)
199 % eventdata  reserved - to be defined in a future version of MATLAB
200 % handles    structure with handles and user data (see GUIDATA)
201
202 set(handles.vars,'String',handles.namesNoTarget);
203 set(handles.word, 'string','');
204
205
206 % --- Executes on selection change in finalVars.
207 function finalVars_Callback(hObject, eventdata, handles)
208 % hObject    handle to finalVars (see GCBO)
209 % eventdata  reserved - to be defined in a future version of MATLAB
210 % handles    structure with handles and user data (see GUIDATA)
211
212 % Hints: contents = cellstr(get(hObject,'String')) returns finalVars
        contents as cell array
213 %        contents{get(hObject,'Value')} returns selected item from
        finalVars
214
215
216 % --- Executes during object creation, after setting all properties.
217 function finalVars_CreateFcn(hObject, eventdata, handles)
218 % hObject    handle to finalVars (see GCBO)
219 % eventdata  reserved - to be defined in a future version of MATLAB
220 % handles    empty - handles not created until after all CreateFcns called
221
222 % Hint: listbox controls usually have a white background on Windows.
223 %       See ISPC and COMPUTER.
224 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
225     set(hObject,'BackgroundColor','white');
226 end
```

Bibliografía

[1] Tibshirani R. Regression Shrinkage and Selection Via the Lasso. Journal of the Royal Statistical Society, Series B. 1996;58:267–288.

[2] Universidad de Valladolid. *Inferencia estadística. Estimación puntual*;. [Online]. Disponible en: `http://www5.uva.es/estadmed/inferen/estima_punt/t9prop_ins.htm`.

[3] Universidad de Granada y Miguel Ángel Montero Alonso. *Inferencia, estimación y contraste de hipótesis*;. [Online]. Disponible en: `http://www.ugr.es/~eues/webgrupo/Docencia/MonteroAlonso/estadisticaII/tema4.pdf`.

[4] Universidad Carlos III de Madrid, Departamento de Estadística. *Contraste de hipótesis en una población*;. [Online]. Disponible en: `http://www.est.uc3m.es/esp/nueva_docencia/getafe/economia/estadistica_ii/documentacion_transp_archivos/tema2esp.pdf`.

[5] Universidad Cardenal Herrera; Botella-Rocamora, P ; Alacreu-García, M y Martínez-Beneito, M A . *Inferencia estadística (intervalos de confianza y p-valor).*;. [Online]. Disponible en: `https://www.uv.es/~mamtnez/IECRC.pdf`.

[6] Hospital Ramón y Cajal. *Odds Ratio*;. [Online]. Disponible en: `http://www.hrc.es/bioest/Medidas_frecuencia_63.html`.

[7] Universidad de Murcia y María Elvira Ferre Jaén. *Modelos de Regresión*;. [Online]. Disponible en: `http://www.um.es/ae/FEIR/40/`.

[8] Wikipedia. *Coeficiente de correlación de Pearson*;. [Online]. Disponible en: `https://es.wikipedia.org/wiki/Coeficiente_de_correlacion_de_Pearson`.

[9] Universidad Abierta de Cataluña, Ángel Alejandro Juan Pérez, Renatas Kizys y Luis María Manzanedo Del Hoyo. *Regresión logística binaria*;. [Online]. Disponible en: `https://www.uoc.edu/in3/emath/docs/RegLogistica.pdf`.

[10] Universidad Carlos III de Madrid. *Introducción a la regresión logística*;. [Online]. Disponible en: `http://halweb.uc3m.es/esp/Personal/personas/amalonso/esp/bstat-tema9.pdf`.

[11] Universidad de Granada y Manuel Salas Velasco. *La regresión logística. Una aplicación a la demanda de estudios universitarios.* Estadística Española. 1996;38:193–217.

[12] Universidad de Granada y Tania Iglesias Cabo. *Métodos de Bondad de Ajuste en Regresón Logística*;. [Online]. Disponible en: http://masteres.ugr.es/moea/pages/tfm-1213/TFM_IglesiasCabo_Tania.

[13] Universidad de Valencia y Ezequiel Uriel. *El modelo de regresión simple: estimación y propiedades*;. [Online]. Disponible en: http://www.uv.es/uriel/2%20El%20modelo%20de%20regresion%20lineal%20simple%20estimacion%20y%20propiedades.pdf.

[14] Universidad de Valencia. *Modelo de regresión lineal simple*;. [Online]. Disponible en: https://www.uv.es/uriel/material/Morelisi.pdf.

[15] Universidade da Coruña y Ana González Vidal. *Selección de variables: Una revisión de métodos existentes*;. [Online]. Disponible en: http://eio.usc.es/pub/mte/descargas/ProyectosFinMaster/Proyecto_1263.pdf.

[16] Friedman J, Hastie T, Tibshirani R. The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd ed. Springer; 2008.

[17] James G, Witten D, Hastie T, Tibshirani R. An Introduction to Statistical Learning with Applications in R. 2nd ed. Springer; 2013.

[18] INAOE y Jesús Antonio González Bernal. *Búsquedas basadas en Heurísticas*;. [Online]. Disponible en: https://ccc.inaoep.mx/~jagonzalez/AI/Sesion5_Busquedas2.pdf.

[19] Universidad Abierta de Cataluña y Manuel Terrádez Gurrea. *Análisis de componentes principales*;. [Online]. Disponible en: https://www.uoc.edu/in3/emath/docs/Componentes_principales.pdf.

[20] Mathworks;. [Online]. Disponible en: https://es.mathworks.com/help/stats/.

[21] WEKA. *Class CfsSubsetEval*;. [Online]. Disponible en: http://weka.sourceforge.net/doc.stable/weka/attributeSelection/CfsSubsetEval.html.

[22] Ryan TP. Modern reggression methods. 2nd ed. Wiley; 2008.

[23] Universidad de Castilla La Mancha y Raul Martín Martín. *Correlaciones con SPSS*;. [Online]. Disponible en:https://previa.uclm.es/profesorado/raulmmartin/Estadistica/PracticasSPSS/CORRELACION_CON_SPSS.pdf.

[24] Instituto IEA. *Diferencias entre correlación y regresión lineal*;. [Online]. Disponible en: http://blog.elinsignia.com/2016/11/08/diferencias-correlacion-regresion-lineal/.