Grado en Matemática Computacional

Estancia en Prácticas y Proyecto Final de Grado

Estimaciones de densidad no paramétrica vía KERNEL

Autor:
Ximo Puchades Durà

Supervisor:
Pablo CAMACHO JIMÉNEZ
Tutor académico:
José Antonio LÓPEZ ORTÍ

Fecha de lectura: 15 de Noviembre de 2017
Curso académico 2016/2017

Resumen

Este documento trata sobre la estancia en prácticas en la empresa Openfinance y el trabajo final de grado de la asignatura MT1030-Prácticas Externas y Proyecto Fin de Grado del grado de Matemática Computacional de la Universidad Jaume I.

El proyecto de estancia en prácticas consiste en la simulación no paramétrica vía kernel de una serie de valores para estimar la duración de las mismas.

En el presente documento se explica la teoria de la simulación no paramétrica y se centra en la estimación vía núcleo. Además, se explica el proyecto realizado durante la estancia en prácticas y los resultados obtenidos.

Palabras clave

Estimación no paramétrica, núcleo, ancho de banda, densidad.

Keywords

Non-parametric estimation, kernel, bandwidth, density

Índice general

Capítulo 1

Introducción

En este documento se presenta las tareas realizadas durante mi estancia en prácticas en la empresa *Openfinance* y describe el Trabajo Fin de Grado que desarrolla la estimación de funciones de densidad no paramétrica vía Kernel.

En esta primera sección del TFG, que servirá como introducción para el resto del trabajo, presentaré la empresa, explicaré cuáles son las actividades que desarrollan y cuál es el mercado en el que trabajan ellos.

Posteriormente, explicaré brevemente cuál era mi función dentro de la empresa y qué esperaban de mi estancia en prácticas. Más adelante, en la sección 5 explicaré con más detalle el proyecto realizado en la estancia, expondré el trabajo realizado en cada quincena y los resultados obtenidos.

1.1. Objetivos, fines y actividades de la empresa

Openfinace es una empresa que se dedica al desarrollo, puesta en marcha y mantenimiento de soluciones tecnológicas para el asesoramiento financiero y gestión de carteras.
Su experiencia se basa en el desarrollo de soluciones integradas en entidades financieras que, apoyan a los clientes en todo el proceso de decisión desde la disposición de información financiera hasta las herramientas de trading en el mercado.

La compañía fue fundada en 2002 como consultora especializada en soluciones de inversión que aúna el emprendimiento con la experiencia tecnológica y financiera. Desde el 2011 pertenece al grupo

Bolsas y Mercados Españoles (BME) a través de su sociedad Infobolsa, lo que proporciona a Openfinance una solidez y confianza que lo posiciona como un referente en el sector financiero internacional. Su actividad principal se desarrolla en España, Colombia, Costa Rica, México y Chile.

Openfinance es una compañía experta en gestión del patrimonio y lleva 15 años creando e implantando soluciones de alto valor añadido en entidades financieras nacionales e internacionales. Sus clientes van desde la banca minorista, banca privada, asesores financieros independientes hasta sociedades y Agencias de Valores y compañías aseguradoras.

Openfinance se encarga de proporcionar la tecnología que facilita el proceso de captación, asesoramiento y fidelización de clientes adaptándose así a sus necesidades de negocio y al cumplimiento normativo. Ofrece soluciones globales y flexibles que se adaptan a la necesidad y el crecimiento de cada cliente en las diferentes etapas de su plan de negocio.

1.2. Idea y objetivo de mi trabajo en la empresa

La idea del proyecto a desarrollar durante la estancia en prácticas era desarrollar una aplicación en .NET que fuera capaz de simular el tiempo que se iba a necesitar para desarrollar una serie de tareas y dar una estimación al usuario de la duración de todas las mismas.

El objetivo final de el proyecto era que el usuario obtuviera un informe detallado de los datos que él mismo había introducido, de el proceso de simulación de estos y del resultado final obtenido.

Capítulo 2

Estimaciones no paramétricas

2.1. Introducción

La **regresión no paramétrica**, a diferencia de la regresión paramétrica (que supone que la función de regresión a estimar pertenece a alguna familia paramétrica de funciones), acepta que la función que se desea estimar no toma ninguna forma predefinida. La única condición que debe cumplir la función (m) es que sea *suave*[1] en términos de continuidad y diferenciabilidad.

Supongamos que la función de regresión desconocida es suave. Entonces, podemos esperar que las observaciones tomadas en puntos próximos a uno dado x puedan darnos información del vector de dicha función en x. (Eubank,1988)

Los métodos de regresión no paramétrica permiten mayor flexibilidad y se adaptan mucho mejor a situaciones reales que los métodos paramétricos. Por contraposición, al realizar pocas hipótesis sobre la estructura subyacente de los datos, las estimaciones resultantes pueden dar lugar a soluciones incoherentes, complejas y sobre todo, con un mayor coste computacional.

En el proceso de regresión no paramétrica es muy importante la elección de los tres parámetros que influyen en el desarrollo: *ancho de banda*, *grado de los ajustes polinomiales locales* y el núcleo (o *kernel*). Más adelante explicaremos qué es cada uno de estos parámetros, cómo influye a la hora de el proceso y la forma de elegir el más óptimo en alguno de los casos.

La idea de la regresión no paramétrica es definir unas bandas centradas en cada punto y de un ancho h (ancho de banda), y calcular la estimación en el punto utilizando tan sólo las observaciones que caen dentro de estas bandas. Para obtener la curva de regresión estimada, la banda definida por el parámetro h recorrerá todo el diagrama de dispersión de izquierda a derecha.

[1] Una función C es **suave** en un intervalo I si las primeras derivadas son continuas y no se anulan simultáneamente excepto posiblemente en los puntos terminales del intervalo I.

2.2. Métodos de regresión no paramétrica

2.2.1. Estimadores tipo núcleo

Estos estimadores realizan una media ponderada de las observaciones y_i. Es decir, el estimador de m en el punto x se define [2] como:

$$\hat{m}_h(x) = \sum_{i=1}^{n} \omega_{h_i}(x) y_i \tag{2.1}$$

donde $\{\omega_{h_i}\}_{i=1}^n$ es una sucesión de pesos que pueden depender del vector $\{x_i\}_{i=1}^n \cdot h$.

La elección del vector de pesos determina el comportamiento del estimador y se calcula:

$$\omega_{h_i} = \frac{K_h(x - x_i)}{\sum_{i=1}^n K_h(x - x_i)} \tag{2.2}$$

donde $k_h(\cdot) = \frac{1}{h} K(\frac{\cdot}{h})$ se denomina *función núcleo* y es, precisamente, la que determina el vector pesos (dependiendo de su distancia al punto x).

Finalmente, el estimador de m en el punto x se puede expresar como:

$$\hat{m}_h(x) = \frac{n^{-1} \sum_{i=1}^n K_h(x - x_i) y_i}{n^{-1} \sum_{i=1}^n K_h(x - x_i)} \tag{2.3}$$

2.2.2. Regresión polinomial local

La regresión polinomial local es una generalización de la estimación por núcleos. De hecho, la estimación por núcleos corresponde al ajuste de un polinomio de grado cero.

El objetivo general de la regresión polinomial local es ajustar un polinomio de grado p alrededor de un punto x_0 utilizando los datos de su entorno.

Supongamos que la función de regresión (m) tiene p derivadas en un punto x_0. Ahora, por el *teorema de Taylor* podemos escribir la función como:

$$m(x) \approx m(x_0) + m'(x_0)(x - x_0) + \frac{m''(x_0)}{2!}(x - x_0)^2 + ... + \frac{m^{(p)}(x_0)}{p!}(x - x_0)^p \tag{2.4}$$

Es decir, la función que buscamos se puede aproximar localmente por funciones polinómicas de

[2] (Nadaraya - Watson, 1964)

grado p:

$$m(x) \approx \sum_{j=0}^{p} \beta_j (x - x_0)^j \tag{2.5}$$

2.2.3. Suavizamiento por *Splines*

Un *spline* es una curva definida a trozos por polinomios sobre la que se imponen restricciones en los puntos de unión llamados *nodos*, que dividen el rango de x en regiones. Se utilizan para aproximar curvas con "formas complicadas". Aquí se utilizan los *splines cúbicos*[3]

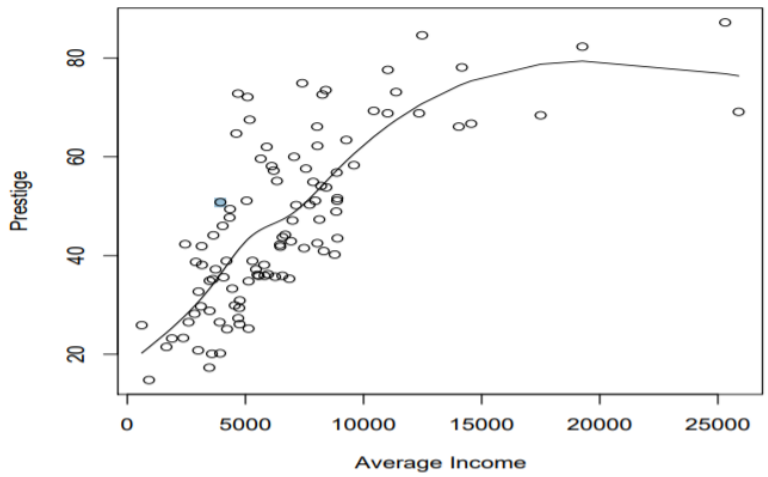


Figura 2.1: Spline cúbico con 4 nodos

Son estimadores que minimizan la función:

$$S_h(m) = \sum_{i=1}^{n} (y_i - m(x_i))^2 + h \int_0^1 m''(x)^2 dx \tag{2.6}$$

El primer término mide la proximidad a los datos. El segundo penaliza la curvatura de la función. h controla el balance del sesgo y la varianza de la curva ajustada:

- Si $h = 0$, la curva interpola los datos.
- Si $h \to \infty$ la segunda derivada se hace 0, por lo que tenemos un ajuste lineal por mínimos cuadrados.

[3]Curva construida a partir de trozos de polinomios de grado 3

2.3. Sesgo y consistencia

Definición 1. *Un estimador $\hat{f}$ de una función de densidad f es insesgado si:*

$$\forall x \epsilon \mathbb{R}^d, \quad E_f[\hat{f}(x)] = f(x)$$

Definición 2. *Un estimador $\hat{f}$ de una función de densidad f es débilmente consistente en forma puntual si $\hat{f}(x) \to f(x)$ para todo $x \epsilon \mathbb{R}$. Por el contrario, diremos que el estimador es fuertemente consistente en forma puntual si la convergencia es casi segura.*

Capítulo 3

Estimaciones via Kernel

3.1. Introducción

Este estimador fue propuesto por primera vez por Rosenblatt en el año 1956 en su publicación *Remarks on some nonparametric estimatees of a density function.*

Sea la muestra de n observaciones reales $X_1, ..., X_n$. Definimos la estimación tipo Núcleo como:

$$\hat{f}_n(x) = \frac{1}{n \cdot h_n} \sum_{i=1}^{n} K(\frac{x - X_i}{h}) \tag{3.1}$$

donde $K(x)$ es la función núcleo y h_n es el ancho de banda

3.2. Parámetros

3.2.1. Ancho de banda

El **ancho de banda** (h_n), también llamado *parámetro de suavización* es una secuencia de constantes positivas que determina la cantidad de suavización de la estimación. La elección del ancho de banda adecuado es un debate extenso, del cuál hablaremos en una sección más adelante. Si h es muy grande, la estimación resulta muy suave; si es muy pequeña, prácticamente se interpolan los datos.

Cabe destacar que, al ser un parámetro fijo a lo largo de toda la muestra, la estimación núcleo suele presentar distorsiones en las colas de la estimación, tal y como vemos en la siguiente figura.

3.2.2. Kernel

La **función núcleo** $(K(x))$, también conocida como *función peso*, es la que define la forma y la importancia de los pesos que se asocian a cada observación para el cálculo de la estimación.

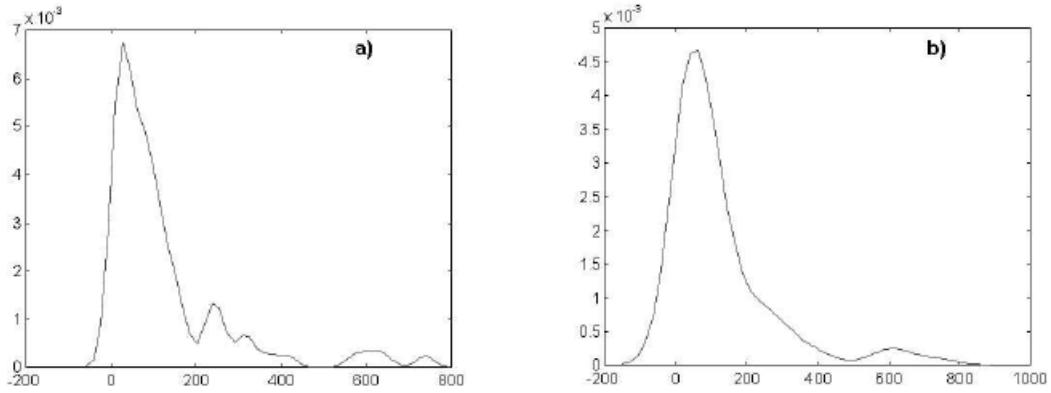


Figura 3.1: Estimación tipo núcleo con $h_n = 20$ (a) y $h_n = 60$ (b)

El estimador núcleo puede interpretarse como una suma de protuberancias (del inglés bump) situadas en las observaciones. (Antonio Miñarro en su libro "Estimacion no paramétrica de la función de densidad").

A continuación presentamos las funciones kernel más utilizadas, así como el rango del parámetro en el cuál la función trabaja correctamente:

1. Epanechnikov: para las observaciones que están a distancia entre 0 y h, asigna pesos de entre $\frac{3}{4}$ y 0 con decrecimiento cuadrático. Por el contrario, las observaciones que están fuera de este rango se les asigna un 0.

$$K(t) = \tfrac{3}{4}(1 - t^2) \quad \text{con} \quad |t| < 1$$

A continuación vemos cuál es su gráfica:

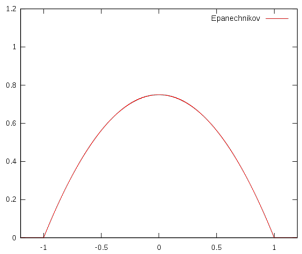


Figura 3.2: Kernel Epanechnikov

2. Gauss: asigna los pesos siguiendo la distribución normal estándar. Es decir, las observaciones cuya distancia oscile entre 0 y 1 tendrán un peso de entre 0,4 y 0,2. Las que estén a distancia 3 tendrán un peso de 0,0039. Y las observaciones cuya distancia sea superior a 3 recibirán un peso muy cercano a 0.

$$K(t) = \frac{1}{\sqrt{2\pi}} e^{-(1/2)t^2} \quad \text{con} \quad |t| < \infty$$

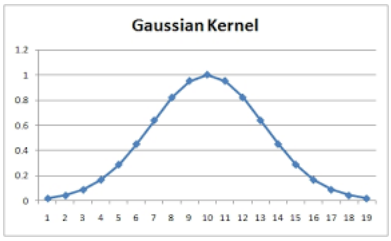


Figura 3.3: Kernel Gaussiano

3. Triangular: asigna pesos de $\frac{1}{h}$ a las observaciones coincidentes. A las observaciones cuya distancia sea igual o superior a h, el peso que se les asigna decrece.

$$K(t) = 1 - |t| \quad \text{con} \quad |t| < 1$$

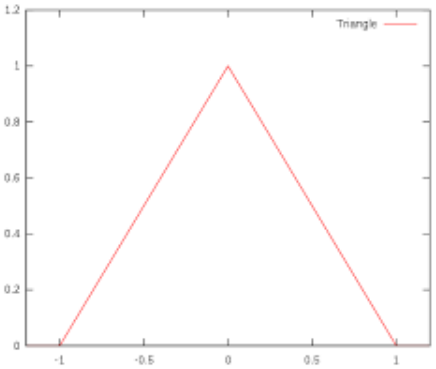


Figura 3.4: Kernel Triangular

4. Rectangular: Asigna el mismo peso a todas las observaciones.

$$K(t) = \frac{1}{2} \quad \text{con} \quad |t| < 1$$

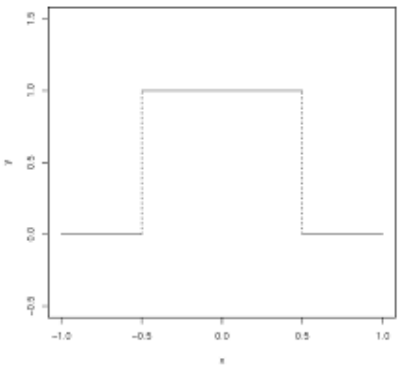


Figura 3.5: Kernel Rectangular

5. Biweight: para observaciones cercanas a las muestras, el peso es un valor cercano a 0,93, que a su vez va decreciendo para observaciones más alejadas.

$$K(t) = \frac{15}{16}(1 - t^2)^2 \quad \text{con} \quad |t| < 1$$

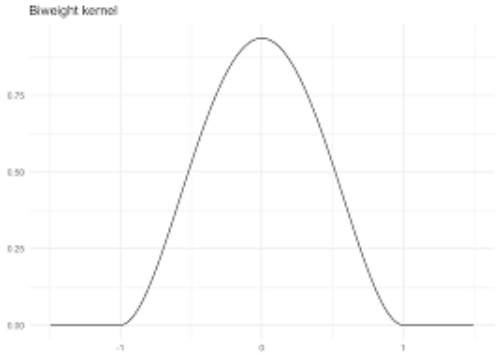


Figura 3.6: Estimador Biweight

6. Triweight:

$$K(t) = \frac{35}{32}(1-t^2)^3 \quad \text{con} \quad |t| < 1$$

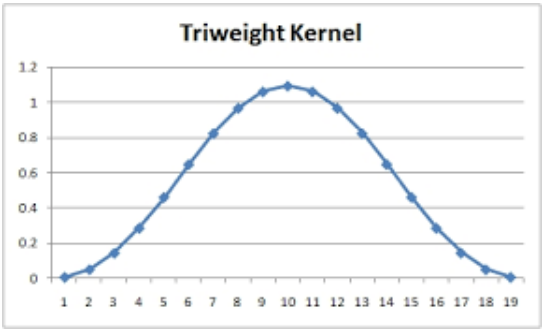


Figura 3.7: Estimador Triweight

Estas funciones son simétricas [1] y acotadas [2] y, además, verifican que:

- $\int_{-\infty}^{\infty} |K(x)|dx < \infty$
- $\lim_{x \to \infty} |x \cdot K(x)| = 0$
- $\int_{-\infty}^{\infty} K(x)dx = 1$

Complementariamente, *Nadaraya* enumeró una serie de condiciones que debían de cumplir las funciones núcleo:

1. $K(x) = K(-x)$ (Condición de simetría anteriormente mencionada)
2. $\int_{-\infty}^{\infty} K(x)dx = 1$
3. $\sup_{-\infty<x<\infty} |K(x)| < \infty$

[1] $\forall x \in \mathbb{R}, \quad f(-x) = f(x)$
[2] $\forall x \in \mathbb{R}, \quad K' < f(x) < K \;\; con \;\; K' < K$

4. $\int_{-\infty}^{\infty} x^i \cdot K(x)dx = 0 \quad i = 1, \ldots, s-1$

5. $\int_{-\infty}^{\infty} x^s \cdot K(x)dx = k_s \neq 0$

6. $\int_{-\infty}^{\infty} x^s |K(x)|dx < \infty$

3.3. Criterio de errores

Debido a la complejidad y variedad de estimadores, necesitamos un parámetro que nos indique 'qué estimación es mejor' o incluso, 'qué estimador es mejor'. Pero, hasta el presente, no se ha llegado a un consenso para establecer un criterio único de error.

Para clasificar la estimación existen dos vertientes dentro de el cálculo de error:

- Cuando trabajemos con estimadores sesgados, utilizaremos el criterio de minimizar el **error cuadrático medio** (*Mean Squared Error*) (MSE)
- Cuando utilicemos estimaciones de las funciones de densidad, el MSE se calculará como la suma de la varianza más el cuadrado del sesgo.

$$MSE\{\hat{f}(x)\} = E[\hat{f}(x) - f(x)]^2 = Var\{\hat{f}(x)\} + Sesgo^2\{\hat{f}(x)\}^3 \tag{3.2}$$

A pesar de esto, y como el objetivo de la estimación no paramétrica es obtener una representación de la densidad completa, necesitamos recurrir a criterios de error globales. A continuación, presentamos los más utilizados:

3.3.1. L_∞

Norma L_∞

$$\sup_x |\hat{f}(x) - f(x)| \tag{3.3}$$

3.3.2. IAE

Norma L_1, conocido como **error absoluto integrado** (*Integrated Absolute Error*):

$$IAE\{\hat{f}(x)\} = \int |\hat{f}(x) - f(x)|dx \tag{3.4}$$

Este criterio da más importancia a las colas de densidad. Además, vemos que:

$$0 \leq L_1 = \int |\hat{f}(x) - f(x)|dx \leq \int |\hat{f}(x)|dx + \int |f(x)|dx \leq 2 \tag{3.5}$$

[3] $Sesgo^2\{\hat{f}(x)\} = E[\hat{f}(x)] - f(x)$

3.3.3. ISE

Norma L_2, conocido como **error cuadrático integrado** (*Integrated Squared Error*):

$$ISE\{\hat{f}(x)\} = \int [\hat{f}(x) - f(x)]^2 dx \tag{3.6}$$

Este criterio sin embargo, al elevar al cuadrado la diferencia, resta importancia a los valores pequeños. También vemos que, al contrario que el criterio anterior, este no está acotado:

$$0 \leq L_2 = \int [\hat{f}(x) - f(x)]^2 dx \ \ \leq \infty \tag{3.7}$$

3.3.4. MISE

Puesto que el *ISE* es una función de una realización particular de n puntos, otro criterio de error más adecuado es considerar el promedio de éste. Este criterio es conocido como **error cuadrático medio integrado** (*Mean Integrated Squared Error*):

$$MISE\{\hat{f}(x)\} = E[ISE\{\hat{f}(x)\}] = \int E[\hat{f}(x) - f(x)]^2 dx \tag{3.8}$$

3.3.5. MISE Ponderado

Existe otra forma de calificar la estimación que consiste en introducir un **vector de pesos** a la hora de calcular el MISE para, por ejemplo, dar más importancia a un intervalo determinado si se desea:

$$MISE_w\{\hat{f}(x)\} = \int E[\hat{f}(x) - f(x)]^2 w(x) dx \tag{3.9}$$

siendo w(x) el vector de pesos.

3.3.6. Otros criterios de error

- El criterio de Kullback-Leibler. La divergencia de *Kullback-Leibler* entre dos funciones $f(x)$ y $g(x)$ se define como:

$$I(f;g) = \int f(x) \log(\frac{f(x)}{g(x)}) \tag{3.10}$$

  Por tanto, utilizando esto, comparamos la función de densidad con la estimación y obtenemos:

$$\int \hat{f} \log(\hat{f}/f) \tag{3.11}$$

- La distancia de Hellinger

$$[\int (\hat{f}^{1/p} - f^{1/p})^p]^{1/p} \tag{3.12}$$

- La variación total

$$TV(\hat{f}, f) = \sup_A |\int_A \hat{f} - \int_A f| \tag{3.13}$$

- Normas L_p

$$L_p = \left[\int |\hat{f} - f|^p\right]^{1/p} \qquad 0 < p < \infty \tag{3.14}$$

3.4. Selección del ancho de banda

La idea que vamos a utilizar en esta sección va a ser la de realizar un promedio local de las observaciones en un entorno de x. Es decir, en cada punto de estimación, se hará un promedio local de las observaciones Y_i incluidas en una banda vertical con intervalo centrado en dicho punto. Obviamente, la amplitud de ésta dependerá del valor escogido en el parámetro h (ancho de banda o amplitud).

- Si el ancho de banda que escogemos es grande, necesitaremos de muchas observaciones para poder calcular la estimación de m en cada punto x. Como consecuencia de esto, obtendremos un aumento del sesgo de la estimación (anteriormente explicado) y un estimador con una excesiva suavidad.

- Si el ancho de banda que escogemos es pequeño, las observaciones que precisaremos reducirán considerablemente. Por tanto, obtendremos un aumento de la variabilidad en la estimación y un estimador con poca suavidad.

Por tanto, para buscar el ancho de banda óptimo, deberemos buscar un equilibrio entre la varianza y el sesgo del estimador. Es decir, encontrar un ancho de banda que haga posible un ajuste razonable de los datos sin que aumente excesivamente la variabilidad del estimador.

Una primera posible solución sería dejar al investigador la elección de este parámetro. Sería una elección subjetiva del ancho de banda: la persona encargada de realizar la estimación, después de observar un diagrama de dispersión de los datos, escogería un parámetro de suavizado que él considerara oportuno y adecuado para dichos datos en concreto. Este método de elección de h hace que la estimación no pueda ser automatizada. Además, este método se empieza a complicar a medida que aumentamos las dimensiones con las que trabajamos.

Por el contrario, si encontramos un método de selección dirigido por los datos que nos devuelva el ancho de banda óptima de forma automática, evitaríamos la subjetividad de la estimación y conseguiríamos la automatización del método.

3.4.1. Selectores basados en la suma de cuadrados de los residuos

Una forma de encontrar el ancho de banda óptimo podría ser realizar una estimación de alguno de los criterios de error comentados en la sección 3.3 y minimizar esta estimación del error con respecto a h.

La suma de los cuadrados de los residuos se expresa como:

$$n^{-1}\sum_{i=1}^{n}\tilde{w}(x_i)[y_i-\hat{m}_h(x_i)]^2 = n^{-1}(y-\hat{m}(h))^T\tilde{W}(y-\hat{m}(h)) \tag{3.15}$$

A partir de esta expresión, la función de validación cruzada (CV) está basada en el estimador $\hat{m}_{h,i}(x_i)$ construido a partir de una submuestra de tamaño $n-1$ tomada a partir de los datos. Dicho estimador es la estimación $\hat{m}_h(x_i)$ que no utiliza la observación i-ésima y_i. El criterio se expresa como:

$$CV(h) = n^{-1}\sum_{i=1}^{n}w(x_i)(y_i-\hat{m}_{h,i}(x_i))^2 \tag{3.16}$$

Por tanto, si denotamos con $\hat{h}_{CV}$ al ancho de banda que hace mínima la función de validación cruzada:

$$\hat{h}_{CV} = \arg\min_{h\epsilon H_n} CV(h) \tag{3.17}$$

Ahora bien, la calidad de la estimación dependerá obviamente del criterio de error que elijamos. Y fue esto precisamente lo que motivó la introducción del *Criterio de Validación Cruzada Generalizada* (GCV), que se define como:

$$GCV(h) = n\sum_{i=1}^{n}\tilde{w}(x_i)\left(\frac{y_i-\hat{m}_h(x_i)}{tr(\tilde{W}-\tilde{W}S(h))}\right)^2 \tag{3.18}$$

Capítulo 4

Simulación de muestras

4.1. Introducción

El programa R es un entorno de análisis y programación para la informática estadística que forma parte del proyecto de software libre GNU (*General Public License*).

Fueron los estadísticos *Ross Ihaka* y *Robert Gentleman* (junto con su grupo de investigación de la universidad de Auckland), quiénes basándose en el lenguaje de programación S [1], desarrollaron este nuevo lenguaje de programación más sencillo, eficiente y sobretodo intuitivo.

A diferencia de otros lenguajes de programación, R es un lenguaje interpretado [2]. R-base permite realizar tareas de estadística sencillas, básicas, habituales. Pero la gran virtud que tiene este programa es la facilidad de ampliar la funcionalidad mediante librerías y extensiones. Gracias a los usuarios que utilizan R, se han ido desarrollando nuevas librerías con todo tipo de utilidades que dotan a este programa de un amplio abanico de posibilidades dentro de el campo de la estadística.

RStudio es una herramienta gráfica y abierta para R que ofrece una base flexible y poderosa para la informática estadística. La aparición de este software hizo que R fuera abierto para todos los usuarios, que a su vez ayudaron a ampliar las funcionalidades de R y

[1] S es un lenguaje de programación estadístico desarrollado en los años 80 y que está orientado a objetos de alto nivel. Las dos implementaciones modernas de S son R y S-PLUS.

[2] Un lenguaje interpretado es aquel para el que la mayoría de sus implementaciones ejecuta las instrucciones directamente, sin compilar a instrucciones en lenguaje máquina.

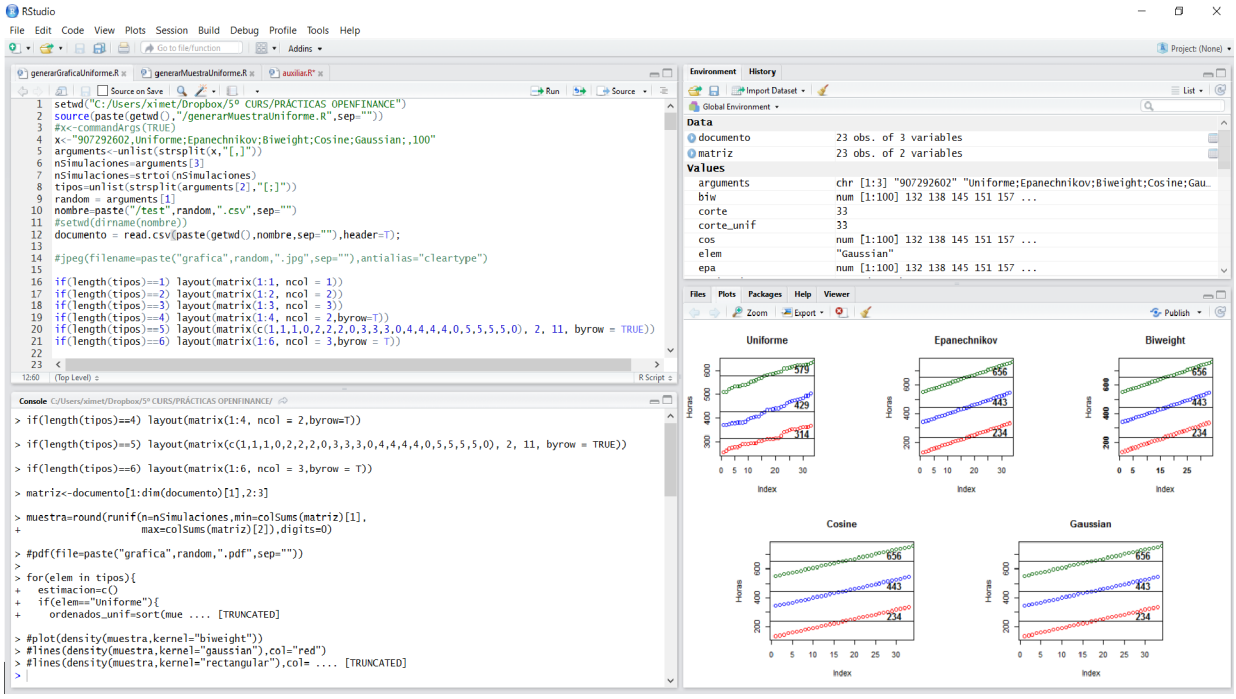


Figura 4.1: Captura de Rstudio durante el desarrollo del proyecto de prácticas

que éstas fuesen compartidas para todo el público.

4.2. Función *density* de R

La función density(x, bw, kernel,...) de R realiza una estimación de la función de densidad vía kernel. Esta función, la cuál viene instalada por defecto en el software (es decir, no es necesario cargar ningún paquete adicional desarrollado por algún usuario), nos permite, a partir de una muestra de los datos, realizar una simulación de la función de densidad vía núcleo. A continuación presentamos los principales argumentos que admite esta función [3]:

- **x**: Los datos a partir de los cuáles se calculará la estimación.
- **bw**: El ancho de banda.
- **adjust**: El ancho de banda utilizado en realidad es $adjust * bw$

[3]Documentación recogida del propio software R

- **kernel**: El kernel que se va a utilizar. Puede ser un valor de entre ('gaussian', 'rectangular', 'triangular', 'epanechnikov', 'biweight', 'cosine' o 'optcosine')

- **weights**: El vector numérico de pesos de observación no negativos (de la misma longitud que x).

- **give.Rkern**: Se trata de un valor lógico. Si es *TRUE*, no se estima la densidad y se devuelve el ancho de banda canónico del kernel escogido.

- **n**: el número de puntos igualmente espaciados en los que se va a estimar la densidad. Cuando $n > 512$, se redondea a una potencia de 2 durante los cálculos y el resultado final es interpolado por approx. Así que casi siempre tiene sentido especificar n como un potencia de dos.

- **from,to**: los puntos izquierdo y derecho de la rejilla a los que se debe estimar la densidad.

- **na.rm**: Se trata de un valor lógico. Si es *TRUE*, elimina los valores que faltan. Si es *FALSE*, dichos valores causan error.

Ahora, vamos a explicar los diferentes valores que devuelve la función:

- **x**: Las coordenadas de los puntos donde se estima la función.

- **y**: Los valores de densidad estimados. Son no negativos, aunque pueden ser cero.

- **bw**: El ancho de banda utilizado, especificado anteriormente en la llamada a la función.

- **n**: El tamaño de la muestra después de la eliminación de los valores nulos.

- **call**: La llamada que produjo el resultado.

- **data.name**: El nombre del argumento x.

- **has.na**: Es un valor lógico para la compatibilidad.

4.3. Función *bw.nr* de R

Para solucionar el problema que hemos estudiado en la sección 3.4, donde intentabamos encontrar el ancho de banda óptimo, R dispone de la función *bw.nr*. Esta función es llamada en el argumento de la función *density bw* e indica el método que se utiliza para calcular el ancho de banda óptimo. A continuación presentamos los posibles

métodos que se implementan:

- **bw.nrd0**: esta función implementa una regla empírica para elegir el ancho de banda. El valor predeterminado es 0.9 veces el mínimo de la desviación estándar y el rango intercuartílico dividio entre 1.34 veces el tamaño de la muestra y la potencia negativa de un quinto (regla de Silverman).

- **bw.nrd**: utiliza la variación de Scott usando el factor 1.06 [4]

- **bw.ucv y bw.bcv**: Implementa la validación cruzada parcial y sesgada respectivamente.

- **bw.SJ**: Implementa los métodos de *Sheather and Jones* usando la estimación piloto de derivados [5]. Este método propone estimar la curvatura utilizando un ancho de banda distinto al que se usa para estimar la densidad.

4.4. Ejemplo de uso de la función *density*

Con el fin de visualizar todo lo explicado anteriormente, vamos a ver un ejemplo donde veremos como utilizar la función density y su comportamiento.

En el siguiente ejemplo, utilizamos la muestra *airquality* proporcionada por el propio R en la cuál muestra las mediciones diarias de la calidad del aire en Nueva York tomadas desde mayo a septiembre de 1973. Como bien indica en la documentación, estos datos se obtuvieron del Departamento de Conservación del Estado de Nueva York y del Servicio Meteorológico Nacional. Se miden distintos valores:

- Ozono: ozono medio medido en ppb (partes por billón).
- Solar.R: radiación solar medida en Langleys.
- Viento: velocidad media del viento medida en mph (millas por hora).
- Temperatura: temperatura máxima diaria en grados Fahrenheit.
- Mes
- Dia

De todas estas medidas, vamos a estudiar el caso del ozono. Como este vector tiene datos faltantes (nulos), eliminamos estos (porque carecen de interés) y pasamos de tener un vector con 153 muestras a 116.

[4] Scott, D. W. (1992) Multivariate Density Estimation: Theory, Practice, and Visualization. Wiley.

[5] Sheather, S. J. and Jones, M. C. (1991) A reliable data-based bandwidth selection method for kernel density estimation. Journal of the Royal Statistical Society series

Ahora, vamos a simular una segunda muestra de datos de una ciudad que llamaremos *Rozhok*. Además, vamos a suponer que en este caso, el ozono de esta zona sigue una distribución Normal con media y varianza igual a la media y varianza del vector *ozono*. Para realizar la simulación, vamos a utilizar la función proporcionada por R *rnorm*.

Por tanto, y realizando la estimación de la función de densidad de Rozhok vía kernel, obtenemos la siguiente comparación:

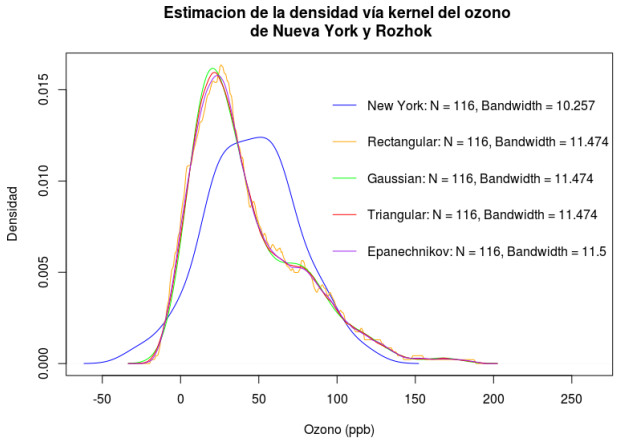


Figura 4.2: Estimacion utilizando la función *bw.nrd0*

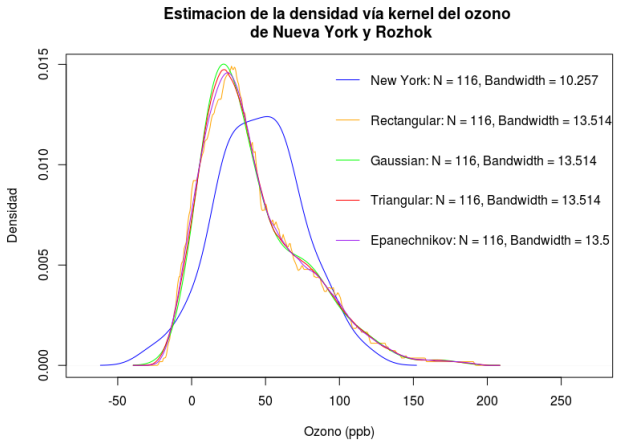


Figura 4.3: Estimacion utilizando la función *bw.nrd*

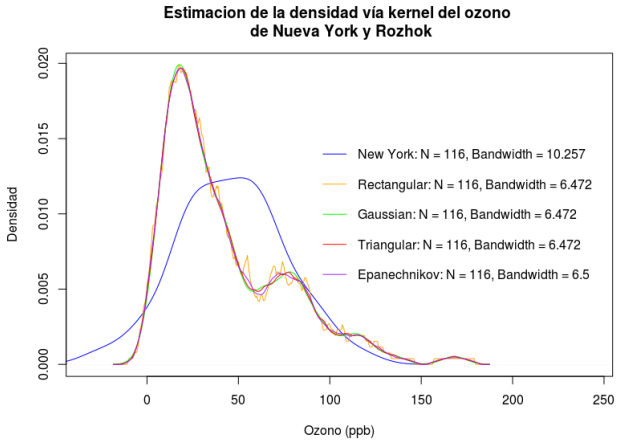


Figura 4.4: Estimacion utilizando la función *ucv*

Aquí vemos que la simulación que más se aproxima a la densidad original resulta de utilizar la función *nrd*, que corresponde con la variación de Scott. Y en cuánto a los núcleos utilizados, todos consiguen un resultado bastante similiar entre ellos. Con esto, podemos sacar la conclusión que no es tan importante la elección del kernel cómo lo es la elección de un buen ancho de banda.

Capítulo 5

Desarollo de las prácticas

5.1. Explicación detallada del proyecto realizado en la empresa

La idea del proyecto que debía realizar en la empresa era básicamente la de estimar una serie de valores con el fin de prever cuanto tiempo iba a tardar un determinado equipo de trabajo en realizar una serie de tareas. Es decir, los valores que se deseaban estimar representaban las horas que un determinado grupo de trabajadores iban a emplear en un futuro para desarrollar una tarea concreta.

Como la idea del proyecto era integrarlo dentro de la aplicación web de la empresa (aunque iba a ser únicamente de uso interno), debía de desarrollar una aplicación en .NET (puesto que es el lenguaje de programación que ellos utilizan) que implementara esta función. Más adelante explicaré cual fue mi proceso de aprendizaje de este lenguaje de programación y daré detalles de unos pequeños proyectos que realicé por recomendación de mi supervisor en la empresa (Pablo Camacho) para acelerar el proceso..

A partir de este estudio estadístico, debía desarrollar una aplicación que pudiese integrarse dentro de la propia web, que permitiera a los trabajadores (más concretamente a los responsables de grupo) introducir una serie de valores e hiciera una estimación completa de los mismos, devolviendo como resultado un informe con una explicación detallada de los resultados obtenidos.

5.1.1. Aplicación Web

La primera parte que realicé del proyecto (aunque como explicaré posteriormente en la sección 5.2, antes de desarrollar el proyecto, tuve que realizar unos pequeños proyectos de prueba para conocer, saber y entender cómo funciona .NET) fue la parte frontal de la aplicación: la interfaz del usuario. Antes de iniciar el proyecto y empezar con la parte de la programación, tuve que valorar cuáles eran los datos que debían mostrarse en la página

principal. Es decir, definir cuál iba a ser la forma de la aplicación web. Después de proponer diversas opciones y formatos, decidí junto con mi supervisor cuál iba a ser el formato del *front*:

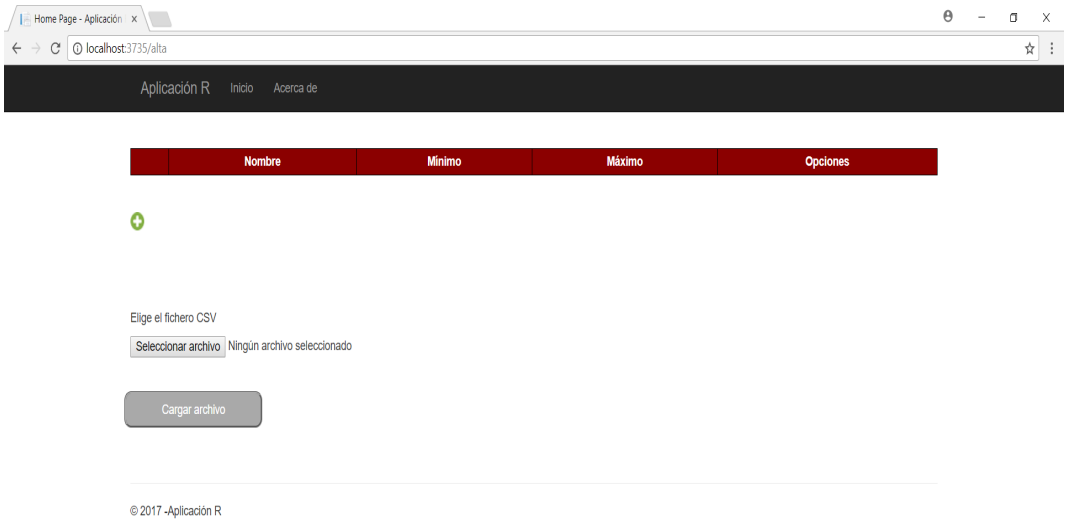


Figura 5.1: Front de la aplicación

Decidimos conjuntamente que lo mejor era que el método para introducir los datos era mediante una tabla vacía. Para rellenarla, el usuario tendría dos métodos: la primera era añadir una a una cada una de las tareas a simular; otra era cargar un fichero excel (.csv) y generar la tabla a partir del mismo.

El primer método fue muy fácil de implementar, puesto que simplemente era añadir un botón con la función de añadir una fila a la tabla.

El segundo método fue un poco más complicado: el primer problema que nos surgió fue la codificación del fichero .csv. Debido a que hay diversas codificaciones con las que se puede guardar un fichero csv, decidimos por convenio que los ficheros que se iban a leer en la aplicación iban a ser con codificación *UTF-8*. Obviamente, si los ficheros que se leían no

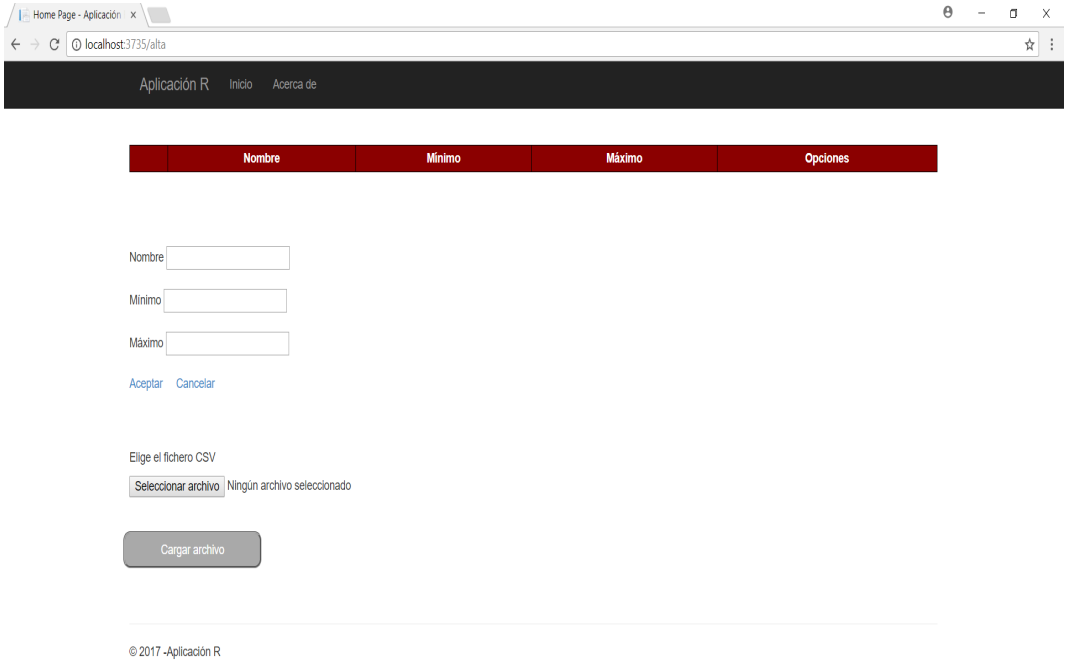


Figura 5.2: Primer método de introducir las tareas en la tabla

estaban guardados en esta codificación, no daría ningún error de apertura; simplemente, habría algunas palabras que no las leería bien (sobretodo palabras con acentos o caracteres extraños).

Además, decidimos también que los ficheros debían tener como separador de columnas un punto y coma (*;*) y como separador de tareas un retorno de carro. Por último, el documento debía finalizar con una línea indicando que era el final del documento con la palabra "*Total*" y el total de las horas mínimo y el total de las horas máximo [1].

Para visualizar mejor todo lo que acabo de comentar, en la Figura 5.3 adjunto una imagen con el fichero inicial que utilicé como documento de prueba de la aplicación.

Hay que recalcar que si los ficheros no tenían este formato, la aplicación no sabría leer el fichero y en esta ocasión si que saltaría un error de apertura del fichero.

Además, para poder modificar la tabla (independientemente de cómo hayamos introducido los datos), añadimos dos botones al final de cada línea que permitían al usuario modificar la tarea o eliminarla directamente.

[1] Hay que destacar que este convenio lo decidí junto con mi supervisor puesto que él sabía qué formato, en general, iban a tener todos los ficheros con los que los usuarios iban a trabajar. Por este motivo me ayudó a definir todos estos estándares.

```
Base;24;64
Login;8;32
Contraseña;8;16
Información legal;4;16
Advertencia legal;4;16
Mi cartera/Cuentas;8;16
Informe posiciones y comparado;16;40
Listado movimientos;4;12
Vista producto/Datos;4;16
Vista producto/Evolución;32;56
Vista producto/Distribuciones;16;44
Vista producto/RentabilidadRiesgo;16;36
Resumen y resumen comparado;6;16
Distribución cartera;16;44
Detalle distribución;8;32
Rentabilidad cartera;32;56
Riesgo;8;24
Perfil riesgo;8;24
Datos personales;4;16
Datos banquero;4;16
Correspondencia;12;32
Firmar documentos;4;8
Cambiar contraseña;4;8
Total;250;640
```

Figura 5.3: Formato fichero .csv

Una vez introducidas las tareas que se desean simular (con los tres datos requeridos: nombre, mínimo y máximo), y antes de llamar a la *Api* que debe de realizar la simulación (que más adelante hablaremos de ella), el usuario debía de indicar cuáles iban a ser los kernels que quería que se utilizasen en la simulación. Para ello, disponía de una lista de kernels y podía seleccionar cuáles deseaba utilizar. Añadí una opción de simulación que era utilizar la distribución Uniforme, puesto que para comparar con el resto de simulaciones me parecía la más acertada. Por tanto, la lista de las opciones era la siguiente:

- Uniforme
- Epanechnikov
- Biweight
- Cosine
- Gaussian
- Rectangular

Por último, el usuario podía seleccionar el número de simulaciones que quería que se realizasen por núcleo.

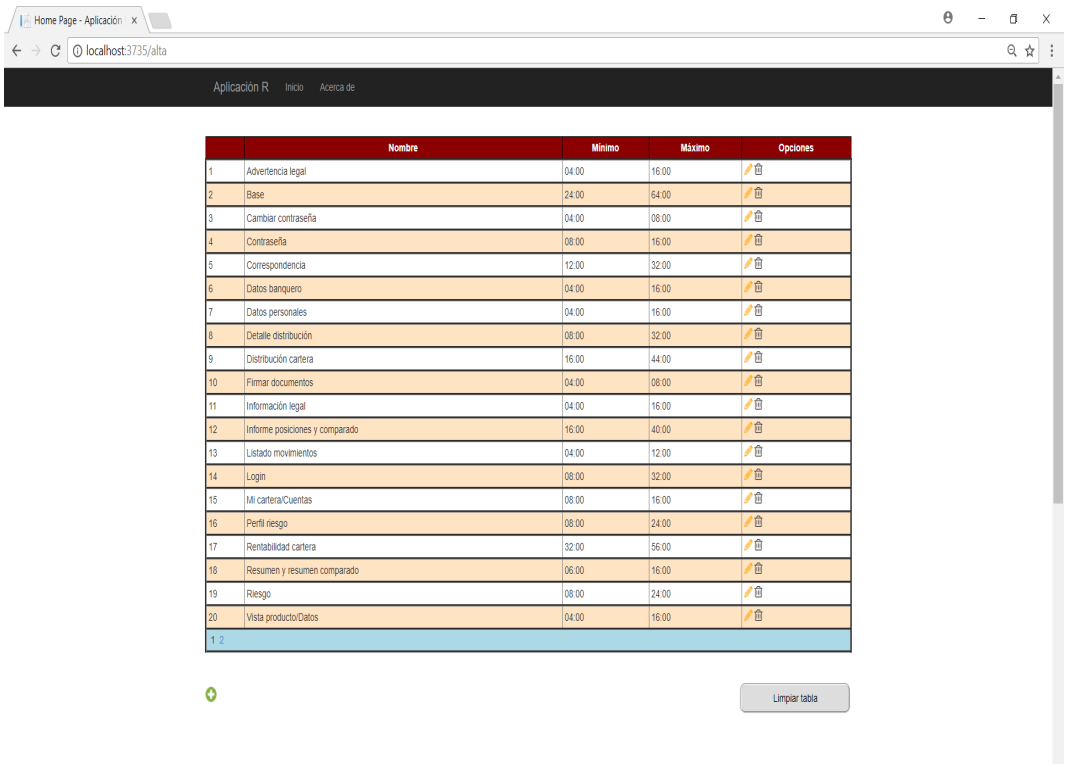


Figura 5.4: Tabla con todas las tareas

El paso siguiente que debía de hacer la aplicación era *serializar* todos estos datos que el usuario había introducido (tanto la tabla con las tareas, como los núcleos seleccionados y el número de simulaciones introducido) para poder enviárselo al servidor que contenía la Api y que supiera interpretar los datos.

Serializar todos los datos y enviarlos como un único objeto en una petición *Request* al servidor no fue tarea sencilla, puesto que el serializador que tiene por defecto .NET solo trabaja con objetos de tipo primitivo. Por tanto, y junto con la ayuda de mi supervisor, tuve que elaborar un serializador propio para que convirtiera la tabla con las tareas en un objeto *json*. A partir de esto, construía un objeto, otra vez de tipo json, que contenía la tabla serializada, una lista con los núcleos seleccionados y el número de simulaciones introducido.

Una vez construido el objeto json con todos los datos, se realizaba la petición al servidor y se dejaba en espera de la respuesta.

5.1.2. Api

El trabajo de la Api era *deserializar* los datos enviados en la petición, enviárselos a R para

que realice la simulación, y recoger el resultado para devolvérselo al cliente.

La primera tarea que era deserializar los datos fue sencilla, puesto que pude reutilizar el serializador de la aplicación. Por tanto, volvía a tener las tareas en forma de tabla, los núcleos en forma de lista y el número de simulaciones en forma de entero.

La lista con los núcleos y el número de simulaciones no causó ningún problema a la hora de pasar los datos a R, puesto que el programa R se iba a ejecutar por consola y se podían pasar estos datos como argumentos. Con la tabla nos vimos obligados a tomar una solución que no teníamos previsto en un principio: tuvimos que volver a generar un archivo .csv y guardarlo en la carpeta local. Esta solución no nos gustaba puesto que suponía un consumo de memoria innecesario [2]. Además, suponía guardar un archivo que luego el propio R iba a volver a tener que abrir. Pero como no encontramos otra solución más eficiente, consideramos esta solución como apta a pesar de los inconvenientes explicados.

5.1.3. Simulación con R

La programación de la simulación con R fue bastante sencilla puesto que era bastante repetitiva. En la opción "$Uniforme$", simplemente realizaba una simulación con el comando *runif* indicando el mínimo y el máximo. En el resto de opciones, realizaba la simulación con el comando *density* que hemos explicado anteriormente indicando el núcleo seleccionado.

Para hacer más práctica la simulación, y que el usuario supiera interpretar mejor los resultados devueltos, añadimos una funcionalidad más: dividir la simulación en tres tipos de resultados, según lo arriesgado que quería ser el usuario a la hora de prever las horas que necesitaría.

Con todo se generaba un gráfico que se guardaba en la carpeta local reutilizando el número aleatorio que se pasaba como argumento para evitar, otra vez, la colisión de peticiones.

5.1.4. Lectura del gráfico

La última tarea que realizaba la Api era leer el gráfico generado por R y pasarlo en el cuerpo de la respuesta al cliente para mostrarlo en la aplicación. La imagen la pasaba como un

[2]Para evitar una colisión de peticiones, para cada petición se generaba un número aleatorio que se adhería al nombre del fichero .csv (Por ejemplo, un fichero podría llamarse test126241.csv). Este número también se pasaba como argumento de consola para que R supiera cuál de todos los ficheros debía abrir

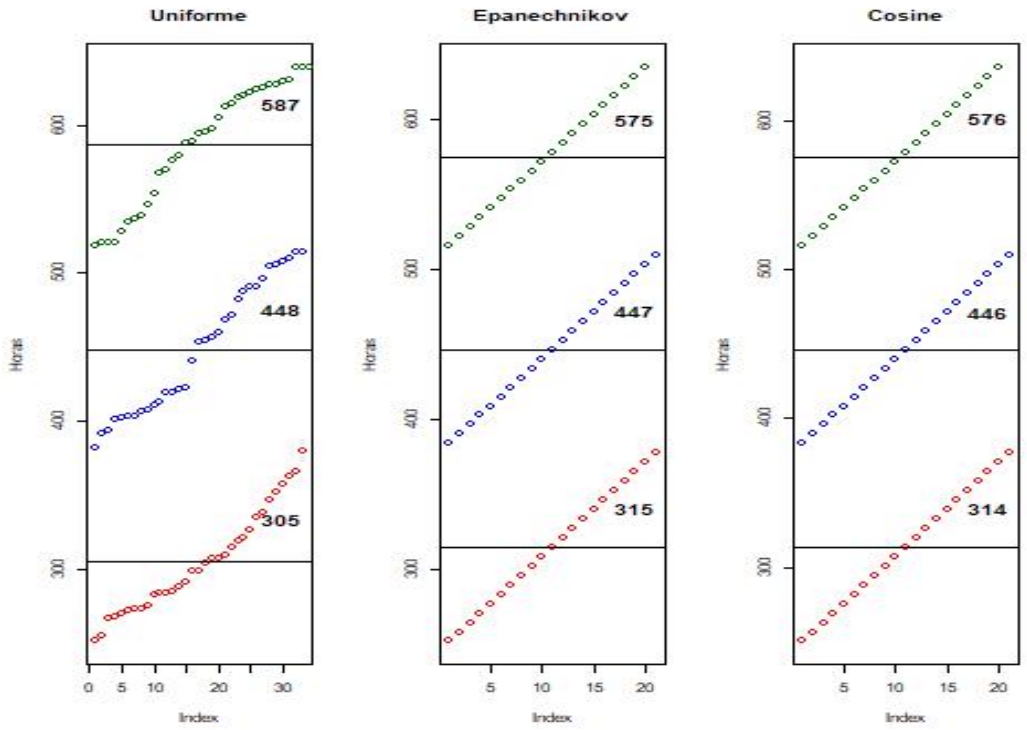


Figura 5.5: Ejemplo de simulación

vector de bytes, por lo tanto la implementación de esta parte fue bastante sencilla.

Por último, el cliente tenía que leer este vector de bytes y mostrarlo como una imagen en la aplicación:

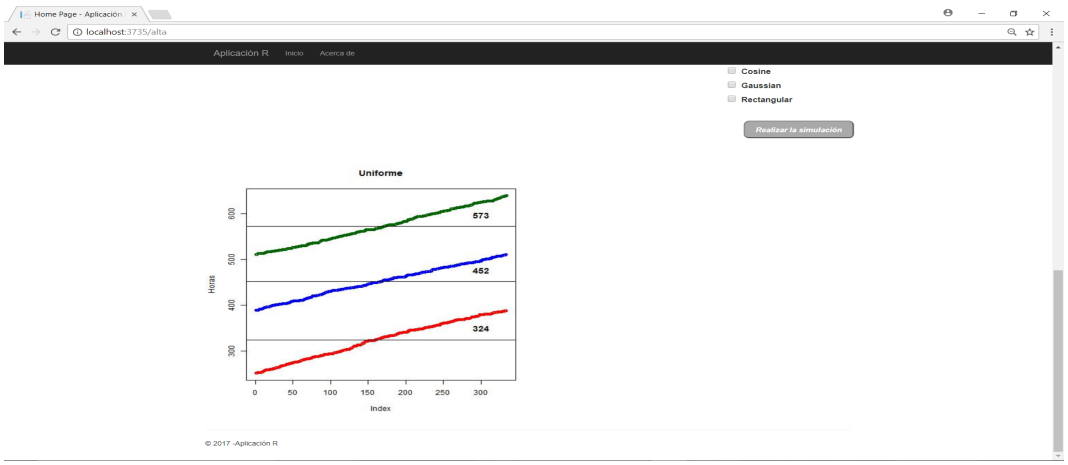


Figura 5.6: Resultado con 1 gráfico

Además, el gráfico que se generaba redimensionaba el espacio disponible según el número de núcleos que se hayan seleccionando. Por ejemplo, en la gráfica 5.7 vemos que el cliente ha seleccionado los 6 núcleos.

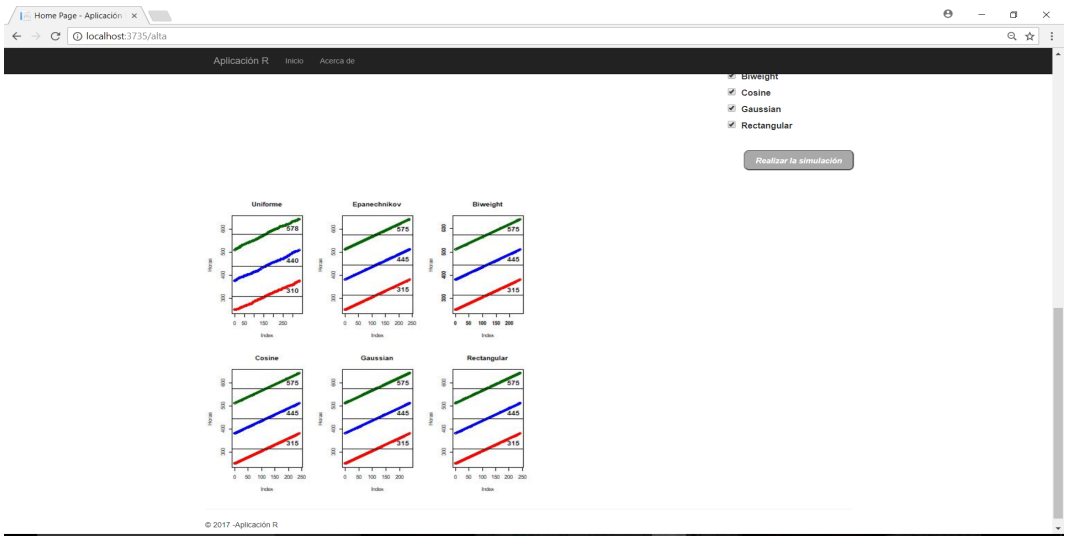


Figura 5.7: Tabla con todas las tareas

5.1.5. Parte por hacer: Generación de un informe

Aquí presento la parte que me quedó por hacer en mi estancia en prácticas y qué, con ello, finalizaría mi proyecto: la generación de un informe con explicaciones sobre la simulación obtenida.

En este informe se pretendía, indicar, por ejemplo, diversos intervalos de confianza de cada una de las simulaciones: por ejemplo, para un nivel de significación del 5 %, cuál era el intervalo de confianza; para un nivel de significación de 1 %, lo mismo.

Además, quería añadir la calidad de la simulación que se había generado, comparando entre todos los núcleos que había utilizado y especificando qué simulación era la mejor.

5.2. Planificación temporal de las tareas

5.2.1. Primera quincena

Como he comentado anteriormente, durante las primeras semanas dejamos a un lado el proyecto estadístico y realicé, por recomendación de mi supervisor, varios pequeños proyectos de prueba que me sirvieron para conocer tanto el funcionamiento del software que utiliza la empresa (Visual Studio, SQL Management, etc.) como el funcionamiento de la propia empresa.

Antes de empezar con mi estancia en prácticas, y para que el periodo de aprendizaje en mi empresa fuera lo más ágil posible, estudié por mi propia cuenta el funcionamiento básico tanto de Visual Studio como de .NET. Fue ya, una vez dentro de la empresa cuando empecé a practicar con dicho lenguaje. Para ello, hice un formulario básico en .NET. El usuario debía rellenar los campos básicos que aparecen en un formulario: Nombre, contraseña, teléfono, dirección, etc. Además, añadí ciertas restricciones a la hora de introducir dichos datos:

- La contraseña debía de introducirse en dos campos diferentes. Si la contraseña introducida en los dos campos no coincidía o si no cumple unos requisitos de seguridad mínimos, salta un error.
- El teléfono introducido debía de tener 9 cifras.
- El código postal tenía que tener 5 cifras.

Luego, la aplicación se conectaba con una base de datos creada previamente. Antes de introducir el usuario en la base de datos, y si cumplía todas las restricciones anteriormente nombradas, comprobaba que no existía un usuario con ese nombre.

Si todo lo anterior era correcto, el usuario era añadido a la base de datos y daba un mensaje de bienvenida a la aplicación.

Por último, añadí una página "*Login*" donde el usuario podía iniciar sesión con su cuenta si anteriormente se había registrado en la aplicación. Aquí simplemente había que buscar si el nombre que introducía el usuario estaba en la base de datos registrado. Si era así, la aplicación consultaba cuál era la contraseña asociada a este nombre y comprobaba que coincidía con la introducida por el usuario. Si todo funcionaba correctamente, saltaba el mismo mensaje de bienvenida que anteriormente.

5.2.2. Segunda quincena

Durante este período tuve que buscar información y documentarme acerca de la estimación no paramétrica vía kernel. Puesto que no conocía su funcionamiento, investigué primero la parte teórica (cómo funciona, tipos de kernel, elección del ancho de banda, parámetros óptimos, etc.) puesto que después sería más sencillo entender la simulación en R.

Realicé unos pequeños ejemplos en R para entender el funcionamiento básico de la función *density* en R. Y como encontré una función de R llamada *Shiny*[3] que me pareció muy interesante, y que funcionaba de manera interactiva con el usuario, decidí hacer una primera versión de la simulación para el usuario. De momento, su funcionamiento no era el correcto, pero mi objetivo de momento era hacer una primera versión de prueba para conocer el funcionamiento de Shiny.

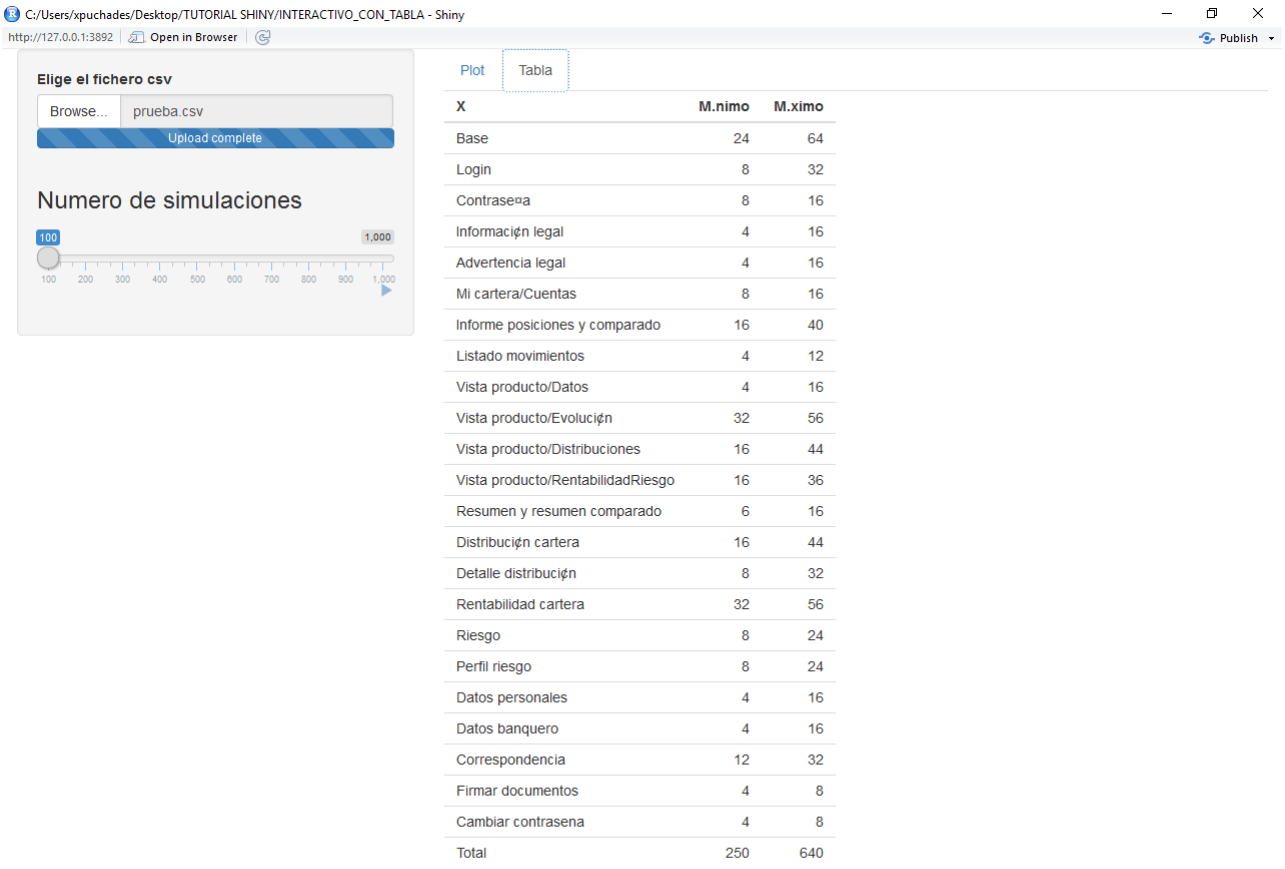
La aplicación Shiny permitía al usuario adjuntar un archivo .xml. A partir de este, se podía visualizar una tabla con los datos que en él estaban:

5.2.3. Tercera quincena

En esta tercera quincena empecé con la parte de estimación de datos. El objetivo era dejar listo el R-script que se iba a ejecutar en la aplicación el cuál debía de realizar la estimación deseada.

Como hemos comentado anteriormente, este script debía de leer un fichero .xml, el cuál contenía una tabla de tareas con un mínimo y un máximo de horas asociado a cada una. En esta primera versión, el tipo de núcleo que debía de simular lo obvié y la simulación la

[3]Shiny es un paquete R que facilita la creación de aplicaciones web interactivas directamente desde R. Más información en `https://shiny.rstudio.com`



X	M.nimo	M.ximo
Base	24	64
Login	8	32
Contrase¤a	8	16
Informaci¢n legal	4	16
Advertencia legal	4	16
Mi cartera/Cuentas	8	16
Informe posiciones y comparado	16	40
Listado movimientos	4	12
Vista producto/Datos	4	16
Vista producto/Evoluci¢n	32	56
Vista producto/Distribuciones	16	44
Vista producto/RentabilidadRiesgo	16	36
Resumen y resumen comparado	6	16
Distribuci¢n cartera	16	44
Detalle distribuci¢n	8	32
Rentabilidad cartera	32	56
Riesgo	8	24
Perfil riesgo	8	24
Datos personales	4	16
Datos banquero	4	16
Correspondencia	12	32
Firmar documentos	4	8
Cambiar contrasena	4	8
Total	250	640

Figura 5.8: Aplicación Shiny

realizaba sobre los kernels que yo le decía. Lo mismo pasaba con el número de simulaciones.

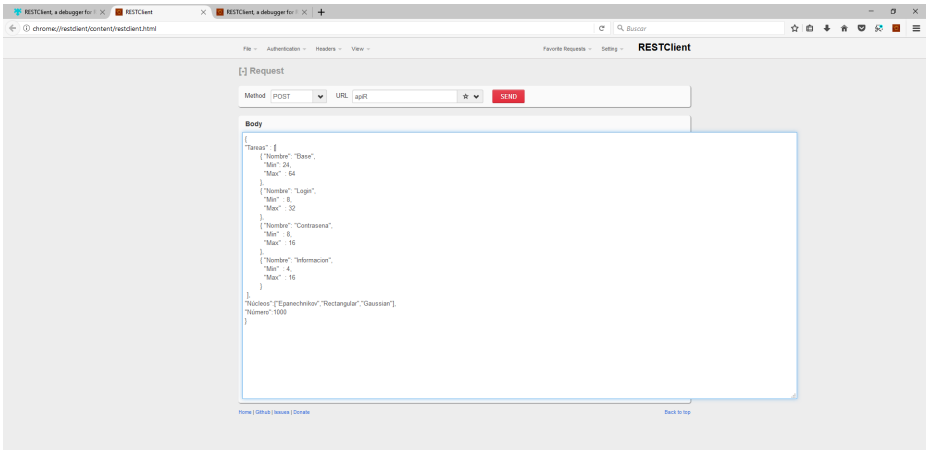
Luego, una vez realizada la tarea de simulación, mostraba una gráfica con la densidad estimada para visualizar mejor los resultados obtenidos.

5.2.4. Cuarta quincena

Durante esta quincena desarrollé la API que iba a ejecutar el R-script implementado en la anterior quincena. La tarea, a priori, parecía bastante sencilla. Simplemente (y como he comentado en la sección anterior) se debía de deserializar los datos que se recibían en la

petición del cliente y pasarlos al R-script. Pero, al ser mi primera API desarrollada íntegramente en .NET y desconocer, casi por completo, su funcionamiento, se complicó bastante la implementación de la misma.

Para empezar el desarrollo de la API, necesitaba de alguna herramienta que simulara peticiones a la misma. Gracias a la ayuda de mis compañeros de la empresa, conocí un complemento que ofrece el navegador Firefox llamado *RESTClient*[4] que sirve para simular peticiones de cliente a un determinado servidor. Por tanto, en esta herramienta simulaba peticiones POST a mi servidor dónde el cuerpo de la misma era un json con todos los datos requeridos para la simulación: lista de tareas, lista de núcleos y número de simulaciones. Aquí adjunto una captura que realicé durante una de las peticiones:



A partir de aquí, la tarea de deserializar los datos enviados en la petición (tal y como he comentado antes) no fue tarea sencilla, puesto que tuve que desarrollar mi propio deserializador.

5.2.5. Quinta quincena

Durante esta quincena, y ya que en la anterior no me dio tiempo a finalizar, terminé la parte de la API. Quedó en el aire la respuesta del servidor al cliente. No sabíamos exactamente que debía devolver el servidor, si la gráfica que generaba el R-script o bien, los datos de la simulación. De momento, dejamos en el aire esa cuestión.

Además, empecé a implementar la parte de la aplicación. Más que nada, durante el poco tiempo que me quedaba en esta quincena, el objetivo era dejar lista la parte visual, es decir, el diseño de la página.

[4] `http://restclient.net/`

Definí un estilo personalizado para los botones, otro diferente para los campos de introducción de datos, y otro para la tabla de tareas. Para ello, me creé un fichero .css [5] donde iba especificando el estilo que iba a tener cada uno de estos elementos.

5.2.6. Sexta quincena

Durante esta última quincena realicé la parte más tediosa del proyecto: la comunicación entre el cliente y el servidor.

Empecé con la serialización de los datos introducidos en la tabla. Para ello, reutilicé el serializador utilizado en la API. Como he explicado en la sección anterior, se construía un objeto json que contenía la lista de tareas serializadas, la lista de núcleos y el número de simulaciones introducido por el usuario. Con esto, se construye una petición POST al servidor con este objeto json como cuerpo de la misma. Una vez enviada la misma, el usuario se quedaba en espera de la respuesta del servidor.

Una vez hecha esta parte, y en consenso con mi supervisor, debíamos aclarar la parte de la respuesta del servidor. Decidimos, por falta de tiempo, que lo más sencillo era devolver la gráfica generada por R en la respuesta. Aunque aclaramos que lo más completo hubiera sido o bien devolver los datos simulados al cliente, o bien ambas; es decir, los datos simulados (por si el usuario precisa de ellos) junto con la gráfica.

5.3. Estimación de recursos del proyecto

Los recursos materiales que utilicé durante mi estancia en prácticas fueron básicamente el ordenador de sobremesa con acceso a internet que me facilitó la empresa junto con dos monitores.

A nivel de software, la empresa dispone de un CD que suele instalar en todos los ordenadores el cuál incluye todos los programas que suelen utilizar los trabajadores de ésta: *Visual Studio*, *SQL Management*, etc. Además, tuve que descargar e instalar *R-Studio* puesto que precisaba de él para la realización del proyecto.

Uno de los programas que conocí durante la estancia en prácticas y que me sirvió de mucha ayuda fue *Team-Viewer*. Este programa me sirvió a la hora de conectarme a mi ordenador de la empresa cuando estaba en casa o incluso, cuando iba a preguntarle alguna duda a mi profesor de prácticas.

[5] Hoja de estilo donde se especifica la presentación o el estilo de un determinado elemento

5.4. Grado de consecución de los objetivos propuestos

El objetivo principal de mi estancia en prácticas era desarrollar una aplicación que fuese capaz de, a partir de unos tareas introducidas por el usuario a fin de estimar el tiempo necesitado, realizara una estimación del tiempo total que iba a necesitar el usuario para realizar todas las tareas introducidas.

Sin embargo, hubiera deseado disponer de más tiempo para, además de realizar la simulación correctamente, generar y entregar al cliente un documento donde se explicara mejor todo el estudio que se había realizado, así como una explicación detallada del resultado obtenido. Por ejemplo, intervalos de confianza para ciertos niveles de confianza. Este aspecto me pareció muy interesante añadir a la aplicación, pero por falta de tiempo no pude implementarla y se quedó en el aire.

Capítulo 6

Conclusiones

Durante mi estancia en prácticas realicé una aplicación web en el lenguaje de Microsoft .NET que simulaba las horas precisadas para realizar un determinado número de tareas. Para ello, tuve que estudiar la simulación no paramétrica. Dentro de los distintos métodos que existen, decidí aplicar para mi proyecto la estimación vía kernel, puesto que el funcionamiento en R es bastante sencillo.

Mejoré mis conocimientos considerablemente en el diseño de aplicaciones web, en el diseño y gestión de bases de datos, en la implementación de APIs y en el uso algunas funciones de R.

A pesar de no poder finalizar mi proyecto en la estancia en prácticas, he contactado con la empresa personalmente y me han comunicado que han desarrollado la función de realizar un informe a partir de la simulación y que ésta está funcionando dentro de la página de la empresa y que está a disposición de los trabajadores de la empresa para aquel que quiera utilizarla.

Como valoración personal de mi estancia en prácticas, estoy bastante orgulloso y satisfecho de mi labor realizada dentro de la empresa. Creo que he adquirido muchos conocimientos nuevos, tanto de informática como de matemáticas. Además, he conocido como es el día a día dentro de una empresa profesional. En cuanto a la empresa y a los trabajadores, he trabajado muy cómodo dentro de la misma, puesto que he sentido que la empresa contaba conmigo, tanto en el presente como para un posible futuro. Además, mis compañeros dentro de la empresa me han ayudado bastante durante mi estancia en prácticas.

Bibliografía

[1] ANTONIO MIÑARRO, Estimación no paramétrica de la función de densidad, http://www.ub.edu/stat/personal/minarro/documents/Nonpar.pdf, Sec.8

[2] NISA BOUKICHOU ABDELKADER, http://masteres.ugr.es/moea/pages/tfm0809/regresin-no-paramtrica-en-r/!

[3] MIGUEL ÁNGEL REYES CORTÉS, Estimación Paramétrica y No Paramétrica de la Tendencia en Datos con Dependencia Espacial. Un Estudio de Simulación, http://eio.usc.es/pub/mte/descargas/ProyectosFinMaster/Proyecto_401.pdf

[4] TERESA ZAMORA GARCÍA, Métodos de regresión no paramétrica en muestreo en poblaciones finitas, http://masteres.ugr.es/moea/pages/tfm0910/metodosderegresionnoparametricaenmuestreoenpoblacionesfinitas/!

[5] *Documentación básica de la función density*, https://stat.ethz.ch/R-manual/R-devel/library/stats/html/density.html

[6] *Ejemplo de uso de la función density en R* https://www.r-bloggers.com/the-density-function/