Módulo 4 - Aprendizaje Supervisado: Técnicas de Regresión.

4.3 Métodos de *Machine Learning* para regresión.

- ### k-vecinos para regresión
- ### Modelo de regresión $M5$ (*Cubist*)

Autores:

Por María José Gacto

Profesora Titular de la Universidad de Granada.

Departamento de Lenguajes y Sistemas Informáticos, ETSIIT.

Y Augusto Anguita-Ruiz

Investigador postdoctoral en Instituto de Salud Global de Barcelona- ISGlobal.

Recordatorio: Introducción a NoteBook

Dentro de este cuaderno (*NoteBook*), se le guiará paso a paso desde la carga de un conjunto de datos hasta el análisis descriptivo de su contenido.

El cuaderno de *Jupyter* (Python) es un enfoque que combina bloques de texto (como éste) junto con bloques o celdas de código. La gran ventaja de este tipo de celdas, es su interactividad, ya que pueden ser ejecutadas para comprobar los resultados directamente sobre las mismas. *Muy importante*: el orden las instrucciones es fundamental, por lo que cada celda de este cuaderno debe ser ejecutada secuencialmente. En caso de omitir alguna, puede que el programa lance un error, así que se deberá comenzar desde el principio en caso de duda.

Antes de nada:

Es muy muy importante que al comienzo se seleccione "*Abrir en modo de ensayo*" (draft mode), arriba a la izquierda. En caso contrario, no permitirá ejecutar ningún bloque de código, por cuestiones de seguridad. Cuando se ejecute el primero de los bloques, aparecerá el siguiente mensaje: "*Advertencia: Este cuaderno no lo ha creado Google.*". No se preocupe, deberá confiar en el contenido del cuaderno (*NoteBook*) y pulsar en "Ejecutar de todos modos".

¡Ánimo!

Haga clic en el botón "play" en la parte izquierda de cada celda de código. Las líneas que comienzan con un hashtag (#) son comentarios y no afectan a la ejecución del programa.

También puede pinchar sobre cada celda y hacer "*ctrl+enter*" (*cmd+enter* en Mac).

Cada vez que ejecute un bloque, verá la salida justo debajo del mismo. La información suele ser siempre la relativa a la última instrucción, junto con todos los `print()` (orden para imprimir) que haya en el código.

ÍNDICE

En este *notebook*:

1. Aprenderemos los conceptos generales de la técnica de k-vecinos más cercanos para regresión.
2. Aprenderemos los conceptos generales de la técnica de aprendizaje de modelos de regresión M5.
3. Aplicaremos ambas técnicas como alternativas más automatizadas frente a la regresión sobre nuestro conjunto de datos de obesidad infantil.

Contenidos:

1. KNN (K-VECINOS MÁS CERCANOS PARA REGRESIÓN)

> *Idea básica: «Si grazna como un pato, camina como un pato y se comporta como un pato, entonces, ¡Seguramente es un pato!»*

Se basa en las mismas ideas que su conocida versión para clasificación. Necesita de los siguientes tres componentes: Disponer del conjunto de datos de entrenamiento con la posibilidad de poder almacenarlo, una métrica para calcular la distancia entre cualquier par de datos, y un valor de k (número de vecinos cercanos a considerar durante la estimación). Una vez se dispone de los tres componentes mencionados, ***no hay algoritmo de aprendizaje ni ajuste de un modelo***, la idea principal del **algoritmo del vecino más cercano ($k = 1$)** es:

- Almacenar todas las instancias/datos de entrenamiento $< x^i, f(x^i) >$
- Dada una instancia nueva de consulta x^q, primero encontrar la instancia de entrenamiento más cercana x^n, y entonces estimar directamente que $f(x^q) = f(x^n)$

Resulta bastante evidente el hecho de que para un **algoritmo de k-vecinos más cercanos ($k = 1$)** siempre se obtendrá error cero con el propio conjunto de entrenamiento. Aunque sea algo que puede resultar impresionante para quienes aún no tengan mucha experiencia, hay que recordar que el objetivo final de cualquier algoritmo de aprendizaje automático es el de generalizar correctamente, es decir, el de inferir con precisión cuando se presenten nuevos casos que no han sido vistos previamente. El hecho de que esta técnica se ajuste de manera tan extrema a los datos de entrenamiento suele provocar un alto sobreaprendizaje y, por lo tanto, una capacidad de generalización bastante pobre con errores muy elevados en los datos de *test*.

Es por ello, que salvo en situaciones muy específicas, se suele evitar el uso de $k = 1$. En el caso de un **algoritmo de k-vecinos más cercanos ($k > 1$)**, dada una instancia nueva de consulta x^q, reemplazaríamos la estimación directa por:

- Clasificación (variable categórica) - Escoger el voto entre sus k vecinos más cercanos como estimación de $f(x^q)$. Se suelen utilizar valores impares de k como una forma de intentar evitar empates en el voto.
- **Regresión (variable continua)** - Tomar la **media** de los valores f de sus k vecinos más cercanos como estimación, $f(x^q) = \sum_{i=1}^{k} f(x^i)/k$. En este caso es indiferente tener valores de k pares o impares (véase la Figura 1 para un ejemplo con $k = 3$, donde las líneas horizontales provienen de la media de los tres ejemplos más cercanos).

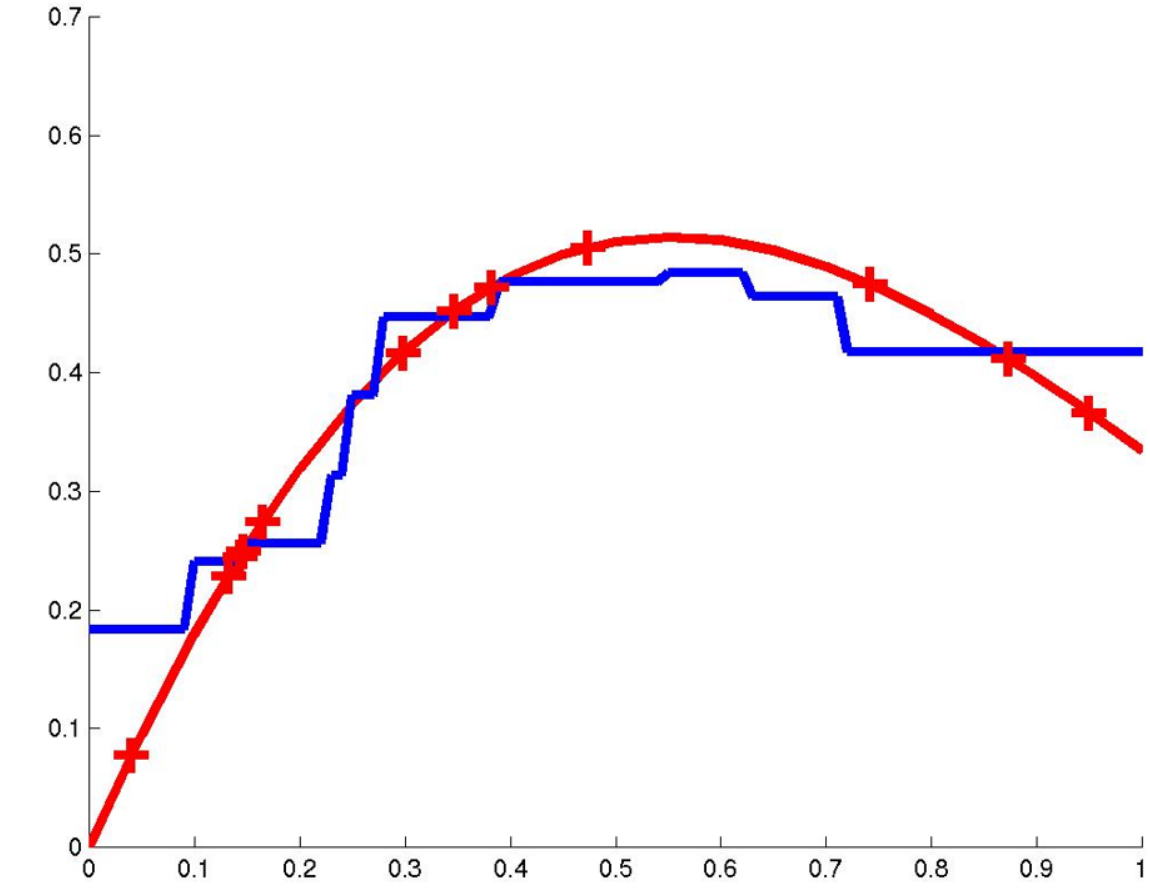


Figura 1. Ejemplo de ajuste con un 3-vacinos más cercanos

1.1 Medida de distancia

La medida de distancia más comunmente utilizada es la distancia euclídea o de Minkowski, aunque no sea la única. También se utilizan otras métricas de distancia como la distancia absoluta, la de *Manhattan*. Los algoritmos suelen implementar la distancia de Minkowski, ya que permite variar la métrica de distancia simplemente cambiando el valor de un parámetro:

$d(x^i, x^j) = \left(\sum_{v=1}^{p} |x_v^i - x_v^j|^q\right)^{\frac{1}{q}}$, q=1 (absoluta o Manhattan); q=2 (euclídea); etc.

En cualquier caso **es necesario normalizar previamente los datos** para que todas las variables afecten equitativamente (aunque en algunas implementaciones ya se hace internamente). Para entender la importancia de la normalización proponemos el siguiente ejemplo: las horas medias de trabajo al día de una persona tienen que estar dentro del rango de las 24 horas (y en teoría no deberían ser superiores a 8), mientras que su salario podría variar de los miles a los cientos de miles. Si no se normalizan los valores de las variables, pequeños cambios en el salario (50.000 a 51.000 euros/año) pesarían muchísimo más que grandes cambios en el número de horas (4 a 8 horas). En concreto, para esos valores tendríamos $|x^i_{salary} - x^j_{salary}|^2 = 1.000.000$ *versus* $|x^i_{hours} - x^j_{hours}|^2 = 16$, con lo que las horas de trabajo prácticamente no influirían en el cáculo de las distancias.

1.2 Escogiendo el valor de k

Debemos tener en cuenta que **la elección del parámetro k puede afectar negativamente en las estimaciones**. Podemos guiarnos por las siguientes dos premisas:

- Si k es demasiado pequeño, sensible a los puntos de ruido.
- Si k es demasiado grande, el vecindario puede incluir puntos no representativos.

Una manera de mitigar estos problemas es el calculo de una media ponderada por la distancia a los k puntos. El caso más extremo de KNN ponderado por distancias, utiliza directamente todo los datos del conjunto de entrenamiento y es conocido como método de Shepard. También existe lo que se denomina funciones Kernel. Una función Kernel sirve para regular la manera en que la distancia a los k puntos afecta al cálculo de la media ponderada para obtener la salida final. Algunos ejemplos son: El kernel rectangular, kernel triangular, kernel de Epanechnikov, kernel de Gauss, etc.

2. MODELO DE REGRESIÓN M5

Como ya se ha indicado en la cápsula 1 de este módulo, **M5 podría ser considerado casi un paradigma en el área de la regresión** (véase el estudio realizado en 2019 entre 164 algorimos por Gacto et. al. donde un simple árbol se muestra competitivo contra los mejores *ensembles* de 500 árboles - *Sección de Bibliografía*). El modelo de regresión M5 (*model tree*) se basa en una estructura de tipo árbol de regresión (arboles de decisión con constantes numéricas en sus hojas en lugar de clases). La principal diferencia de los modelos de regresión respecto a dichos árboles de regresión, es que éstos incluyen funciones lineales multi-variables en las hojas en lugar de una constante. En definitiva, **lo que tenemos es un conjunto de funciones lineales por trozos**, haciéndolo perfecto para ajustarse a todo tipo de datos tanto lineales como no lineales. **Véase en la Figura un ejemplo de modelo obtenido con M5, que incluye 7 funciones lineales distintas a aplicar según se cumplan las condiciones aprendidas para las distintas variables contempladas en el árbol**. A diferencia de los métodos de regresión que hemos visto hasta ahora, M5 **si que permite** y se aprovecha de variables de tipo categórico no ordinal. Así, en el ejemplo mostrado en la figura tenemos que:
$Si\ \ pgain <= 3.5\ \ y\ \ motor = B$ ó A se aplicaría el modelo lineal LM3 (aprovechando la variable categórica $motor$ como variable de decisión en un nodo); y que sin embargo $Si\ \ pgain \in (3.5, 4.5]\ \ y\ \ vgain > 2.5$ se aplicaría el modelo lineal LM5 (atendiendo sólo a variables numéricas en este caso).

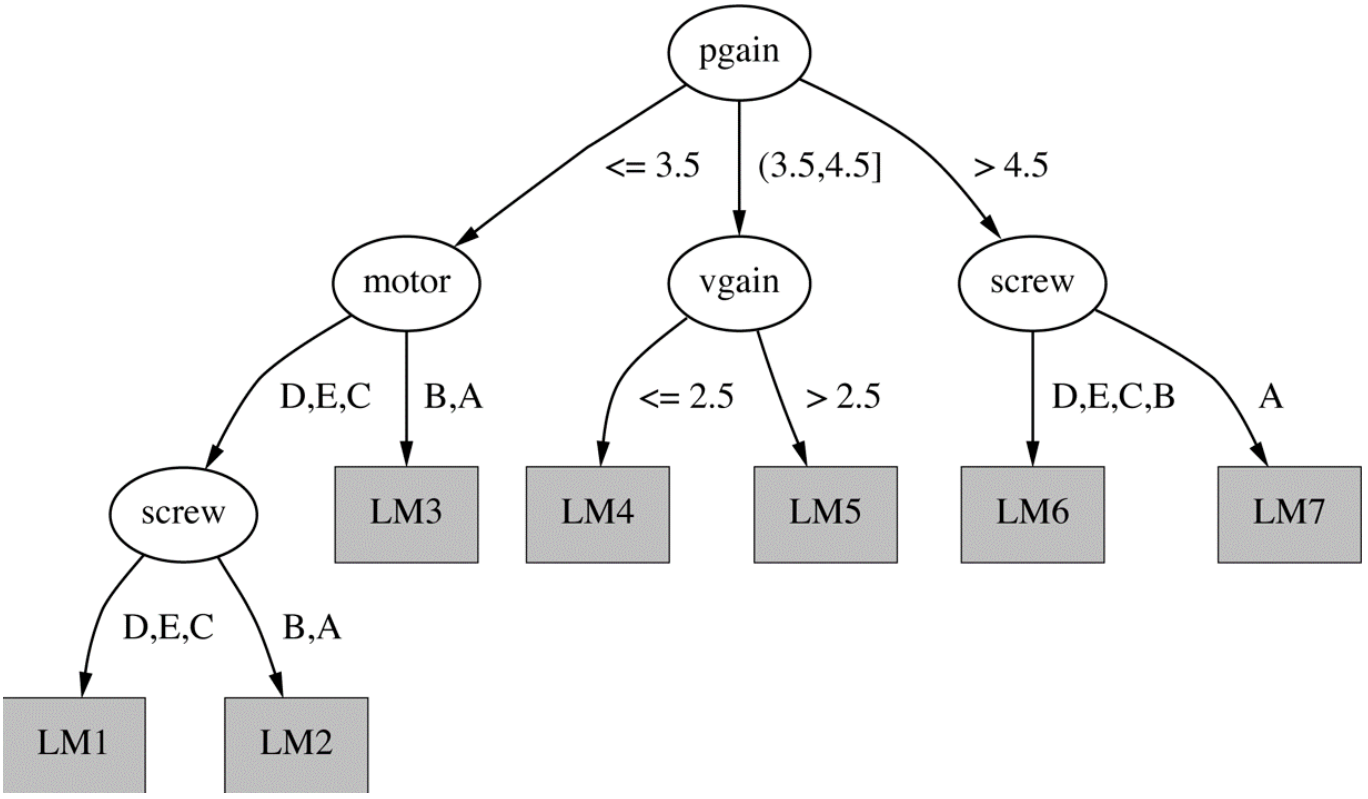


Figura 2. Ejemplo de modelo obtenido con M5

Cuando está correctamente implementado, M5 es uno de los métodos más eficientes, y **puede manejar conjuntos de datos con una dimensionalidad relativamente alta (hasta cientos de atributos, siendo aún así un método bastante rápido)**. Esta capacidad diferencia a M5 de otros tipos de técnicas de regresión como por ejemplo MARS (*Multivariate Adaptive Regression Splines*), cuyo coste computacional (tiempo de ejecución) crece muy rápidamente cuando el número de variables de entrada aumenta. Por otro lado, **una ventaja de M5 sobre CART y otros árboles de regresión es que los modelos obtenidos son generalmente mucho más pequeños/simples y demuestran ser más precisos**.

NOTA: *A partir de aquí se presenta el funcionamiento del método de aprendizaje M5, es decir, la manera a través de la que, a partir de los datos de entrenamiento para un problema dado, se obtienen modelos de regresión como el del ejemplo mostrado en la figura. Entendemos, que si bien las técnicas de regresión descritas hasta el momento podrían haber representado cierto nivel de esfuerzo en cuanto a su comprensión, no deberían de haber sido muy difíciles de seguir y entender. Sin embargo, en este punto, debido a la mayor complejidad técnica de M5, el contenido se hará más técnico y dependiendo de los conocimientos previos puede llegar a ser más difícil de entender.*

Como el hecho de cursar este MOOC demuestra un interés por iniciarse en el área de la Bioinformática (y a pesar de que en este caso se requiere de una explicación mucho más técnica y "quizás" aburrida), se ha tomado la decisión de incluir a continuación una descripción lo más breve y simplificada posible del algoritmo M5 para quienes tengan la curiosidad sana de saber como funcionan estos modelos. Saber cómo se aprenden los modelos hará mucho más fácil la comprensión posterior de los mismos. Por todo ello, hemos creido conveniente incluir una muy breve explicación del método para quién quiera seguirla. No obstante, ***si con lo explicado hasta ahora ha entendido a lo que nos referimos cuando hablamos de un modelo de regresión obtenido por M5*** *(un árbol con condiciones desde su raíz hasta las hojas, las cuales determinan la aplicación local de pequeños modelos lineales)* ***podría continuar directamente con las explicaciones de la cápsula a partir de la siguiente sección (Sección 3)****, donde veremos como ejecutar KNN y M5 sobre nuestro conjunto de datos de obesidad infantil (y saltarse esta explicación).*

El método M5 fue propuesto por **J.R. Quinlan en 1992**, y aunque es uno de los métodos clásicos, a día de hoy sigue reconociéndose como uno de los más potentes. A pesar de que M5 emplea el mismo enfoque que el árbol de regresión *CART* al intentar minimizar el Error Cuadrático Medio (*ECM*) con su función de impureza o de ganancia (utilizada para decidir sobre las posibles divisiones de un nodo), como se ha mencionado, M5 no asigna una constante al nodo de la hoja sino que en su lugar se ajusta a un modelo de regresión lineal multi-variable. En lo que sigue veremos los tres componentes principales del algoritmo de aprendizaje que permiten describir su funcionamento.

2.1 Construcción del árbol

El método M5 sigue la **estrategia de división recursiva de nodos** de los árboles de decisión. Supongamos que en un paso dado del algoritmo (un nodo del arbol a generar) disponemos de un grupo de instancias o datos de entrenamiento T. O bien T se asociará con una hoja, o bien se escogerá alguna de las posibles divisiones atendiendo al criterio de impureza o ganancia, dividiendo T en dos subgrupos. El mismo proceso se aplica recursivamente a todos los subconjuntos. Este proceso a menudo produce estructuras con sobre-aprendizaje que posteriormente deben ser podadas.

Para conseguir **minimizar el *ECM*** cuando se intenta aplicar una división, la ganancia de información que dicha división provocaría se mide por la reducción de la desviación estándar en los valores de la variable de salida de los datos, antes y después de la división. Para ello, el primer paso es calcular la desviación estándar (*standard deviation -sd-*, como se la conoce en Inglés) de los valores de salida de las instancias en T, $sd(T)$. En el caso de que T contenga muy pocas instancias (por ejemplo menos de 4) o que $sd(T)$ ya sea menor que un pequeño porcentaje de la sd inicial, el nodo se convertiría directamente en nodo hoja. En otro caso, se calcularán todas las posibles divisiones de T para cada posible punto de corte sobre cada variable de entrada. Llamemos T_i al subgrupo de instancias $i-th$ correspondiente a una división específica. Si la desviación $sd(T_i)$ de los valores de la variable de salida de las instancias en T_i se utiliza como una medida del error, la reducción esperada del error puede ser definida de la siguiente manera:

$$\triangle error = sd(T) - \sum_i \frac{|T_i|}{|T|} sd(T_i)$$

La división que maximice la reducción de errores esperada será la que finalmente se aplique. Aunque esta formulación trata de ser lo más general posible, en el caso de M5 (como en la mayoría de algoritmos basados en árboles), el número máximo de subconjuntos que cada división generará es dos. Es decir, una división sólo daría lugar a los subgrupos T_1 y T_2. Por lo tanto, para cada nodo con división sólo se generan dos nodos hijo (árbol binario).

2.2 Poda del árbol

La poda se realiza desde las hojas hasta el nodo raíz. En cada nodo interno, M5 **compara el error estimado de ese nodo** (como si no se hubiese dividido y fuese un nodo hoja) **y el error estimado del sub-árbol que de él cuelga**. Entonces, **el sub-árbol se poda si no produce una mejora** suficiente en el rendimiento del árbol.

El factor clave de este método es cómo estima ese error para que no se sobre-aprenda demasiado y se generalice bien en nuevas instancias no vistas. M5 lo calcula promediando primero la diferencia absoluta entre los valores de la variable de salida de los datos de entrenamiento y los valores de sus correspondientes predicciones. Esto generalmente subestimará el error en instancias no vistas, por lo que M5 lo multiplica por $(n+v) = (n-v)$, donde n es el número de instacias asociadas al nodo y v es el número de parámetros en el modelo lineal que quedaría si finalmente fuese una hoja. La intención es aumentar el error estimado de modelos con muchos parámetros construidos a partir de un pequeño número de instancias. El error estimado de un subárbol se calcula como la suma ponderada del error estimado de sus ramas izquierda y derecha, multiplicando por la proporción de muestras que descienden al árbol izquierdo y derecho, respectivamente.

2.3 Obtención de los modelos lineales

Un modelo lineal multi-variable se ajusta a los datos de entrenamiento en cada nodo del árbol utilizando técnicas de regresión estándar. Antes de realizar la poda, aprovechando la vuelta hacia atrás de la recursividad utilizada para generar el arbol, **se ajusta un modelo de regresión lineal para cada nodo sobre las instancias de entrenamiento asociadas al mismo**.

Comienza, por lo tanto, por los **nodos hoja** considerando solo las **variables utilizadas en las divisiones desde la raíz del árbol hasta el nodo hoja** (se mira hacia arriba). Sin embargo, en el caso de los **nodos internos** (aquellos que no son hoja/terminales), M5 no utiliza las variables de las divisiones mirando hacia arriba, sino que se restringe a las **variables que son referenciadas por las divisiones o por los modelos lineales en los subárboles que cuelgan de dicho nodo** (se mira hacia abajo). Estos modelos lineales internos se utilizan únicamente para poder aplicar los criterios de poda. Como M5 comparará la precisión de dicho modelo lineal con la precisión de los subárboles que cuelgan de dicho nodo, esto garantiza unas condiciones de competencia equitativas en las que los dos tipos de modelos utilizan la misma información.

Además, después de construir cualquier modelo lineal M5 lo simplifica, eliminando los coeficientes uno a uno mediante un algoritmo voraz (al estilo del enfoque descendente que utilizamos en la cápsula 2 de este módulo pero automatizado). Por lo general, esto podría resultar en un aumento en el error promedio, sin embargo también reducen los factores multiplicativos anteriores, por lo que el error estimado (para las nuevas instancias aún no vistas) puede disminuir.

3. INSTALACIÓN DE R, BIBLIOTECAS Y LECTURA DE LOS DATOS DE OBESIDAD INFANTIL

Como se explicó en la primera cápsula de este módulo, necesitamos ejecutar las siguientes 3 celdas antes de empezar con los algoritmos Knn y M5.

In [1]:
```
# Tiempo estimado de ejecución: 20 segundos aprox.

### Instalación de R en notebooks de Google Colab ###
!apt-get update
!apt-get install r-base
!pip install rpy2==3.5.1
%load_ext rpy2.ipython
print ("Instalación de R en Google Colab terminada")
```

```
Get:1 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease [3,
626 B]
Ign:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_6
4  InRelease
Get:3 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:4 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease [1
5.9 kB]
Hit:5 http://archive.ubuntu.com/ubuntu bionic InRelease
Ign:6 https://developer.download.nvidia.com/compute/machine-learning/repos/ubun
tu1804/x86_64  InRelease
Get:7 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_6
4  Release [697 B]
Get:8 https://developer.download.nvidia.com/compute/machine-learning/repos/ubun
tu1804/x86_64  Release [564 B]
Get:9 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_6
4  Release.gpg [836 B]
Get:10 https://developer.download.nvidia.com/compute/machine-learning/repos/ubu
ntu1804/x86_64  Release.gpg [833 B]
Get:11 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Hit:12 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
Get:13 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:14 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease [15.9 k
B]
Get:15 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease [2
1.3 kB]
Ign:16 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_
64  Packages
Get:16 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_
64  Packages [577 kB]
Get:17 https://developer.download.nvidia.com/compute/machine-learning/repos/ubu
ntu1804/x86_64  Packages [73.8 kB]
Get:18 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main Sources
[1,745 kB]
Get:19 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages
[1,963 kB]
Get:20 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Package
s [31.4 kB]
Get:21 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packa
ges [24.5 kB]
Get:22 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packa
ges [324 kB]
Get:23 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Package
s [1,396 kB]
Get:24 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages
[2,163 kB]
Get:25 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main amd64 P
ackages [893 kB]
Get:26 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [2,3
94 kB]
Get:27 http://archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Package
s [353 kB]
Get:28 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic/main amd64 Package
s [39.5 kB]
Get:29 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic/main amd64 P
ackages [49.4 kB]
Fetched 12.3 MB in 3s (3,594 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
r-base is already the newest version (4.0.4-1.1804.0).
0 upgraded, 0 newly installed, 0 to remove and 60 not upgraded.
Requirement already satisfied: rpy2 in /usr/local/lib/python3.7/dist-packages
(3.4.2)
```

```
Requirement already satisfied: cffi>=1.10.0 in /usr/local/lib/python3.7/dist-pa
ckages (from rpy2) (1.14.5)
Requirement already satisfied: tzlocal in /usr/local/lib/python3.7/dist-package
s (from rpy2) (1.5.1)
Requirement already satisfied: pytz in /usr/local/lib/python3.7/dist-packages
(from rpy2) (2018.9)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages
(from rpy2) (2.11.3)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packa
ges (from cffi>=1.10.0->rpy2) (2.20)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dis
t-packages (from jinja2->rpy2) (1.1.1)
Instalación de R en Google Colab terminada
```

In [2]:
```
# Tiempo estimado de ejecución: 2:15 minutos aprox.
# Bibliotecas necesarias:
# ISLR para regresión lineal multivariable
# kknn para k-vecinos más cercanos de regresión
# Cubist para modelos de regresión basados en M5

%%R
### Instalación de las bibliotecas necesarias
#install.packages(c("ISLR", "kknn", "Cubist"))
install.packages(c("kknn", "Cubist")) #ISLR se utilizaba en la anterior cápsula
print ("Instalación de las bibliotecas de R para este módulo terminada")

### Importación de las bibliotecas necesarias ###
#require(ISLR)
require(kknn)
require(Cubist)
print ("Importación de las bibliotecas de R para este módulo terminada")
```

```
R[write to console]: Installing packages into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

R[write to console]: also installing the dependencies ‘plyr’, ‘igraph’, ‘reshap
e2’


R[write to console]: trying URL 'https://cran.rstudio.com/src/contrib/plyr_1.8.
6.tar.gz'

R[write to console]: Content type 'application/x-gzip'
R[write to console]:  length 401191 bytes (391 KB)

R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]:
```

```
R[write to console]: downloaded 391 KB


R[write to console]: trying URL 'https://cran.rstudio.com/src/contrib/igraph_1.
2.6.tar.gz'

R[write to console]: Content type 'application/x-gzip'
R[write to console]:  length 2715500 bytes (2.6 MB)

R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]:

R[write to console]: downloaded 2.6 MB
```

```
R[write to console]: trying URL 'https://cran.rstudio.com/src/contrib/reshape2_
1.4.4.tar.gz'

R[write to console]: Content type 'application/x-gzip'
R[write to console]:  length 37307 bytes (36 KB)

R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]:

R[write to console]: downloaded 36 KB


R[write to console]: trying URL 'https://cran.rstudio.com/src/contrib/kknn_1.3.
1.tar.gz'
```

```
R[write to console]: Content type 'application/x-gzip'
R[write to console]:  length 388410 bytes (379 KB)

R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]:

R[write to console]: downloaded 379 KB


R[write to console]: trying URL 'https://cran.rstudio.com/src/contrib/Cubist_0.
2.3.tar.gz'

R[write to console]: Content type 'application/x-gzip'
R[write to console]:  length 463513 bytes (452 KB)
```

```
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]: =
R[write to console]:

R[write to console]: downloaded 452 KB


R[write to console]:

R[write to console]:
R[write to console]: The downloaded source packages are in
        ‘/tmp/RtmpRrk6pO/downloaded_packages’
R[write to console]:
R[write to console]:
```

```
[1] "Instalación de las bibliotecas de R para este módulo terminada"

R[write to console]: Loading required package: kknn

R[write to console]: Loading required package: Cubist

R[write to console]: Loading required package: lattice


[1] "Importación de las bibliotecas de R para este módulo terminada"
```

In [3]:

```
# Tiempo estimado de ejecución: 2 segundos aprox.

%%R
### Lectura
data <- read.csv(url("https://drive.google.com/uc?id=1GO2NBxYw54K6HkN-YgXbNadrLo
5O6-0u"))

### Visualización de una pequeña parte de los datos
head(data)
```

```
  Sex  Age Tanner Height   BMI   WC TAGmgDL HDLCmgDL LDLCmgDL SBP DBP Sedentary
1   1  9.5      0   1.55 11.34 60.0      55       51       93  97  60  411.0893
2   1  8.0      0   1.15 12.40 46.3      51       70       59  90  55  435.6071
3   0 10.5      0   1.42 12.99 67.5      65       60       96  96  54  483.9048
4   0  8.1      0   1.27 13.43 53.1      41       78      100 108  46  429.2976
5   1 10.4      0   1.32 13.72 51.9      39      100      120 107  69  512.0714
6   0 10.4      0   1.29 14.02 54.9      57       76       73  87  59  451.2321
     Light Moderate  Vigorous HOMA
1 321.5804 22.13393  3.982143 1.98
2 316.9762 48.05952 14.273810 0.87
3 337.7857 33.30952  7.988095 1.46
4 241.9762 39.67857 11.821429 1.07
5 216.0357  9.75000  2.410714 0.80
6 257.6429 36.40179  9.767857 1.35
```

4. APLICACIÓN DE KNN AL PROBLEMA

Una vez importadas las librerías y leídos los datos estamos en disposición de aplicar k-vecinos más cercanos a nuestro problema (k-nn). La implementación de k-nn que vamos a utilizar está incluida como parte de la librería *kknn* de R, que ya hemos instalado en nuestro entorno. La función también es conocida como $kknn(.)$ y sigue la siguiente sintaxis:
$kknn(formula = formula(train), train, test, k = 7, distance = 2, kernel =" optimal ", scale = TR$

```
(Aplica k-NN a un conjunto de test dado un conjunto de entrenamiento -> Recordemos!:
 No hay algoritmo de aprendizaje)
```

formula – variables usadas al estilo de lm; *train y test* – conjuntos de datos de entrenamiento y test; *k* – número de vecinos (si no se indica es 7); *distance* – distancia de Minkowski, es decir con 1 Manhattan y con 2 euclídea (si no se indica es 2); *scale* – escalado de las variables para tener igual desviación estandar (si no se indica es TRUE); *kernel* – tipo de kernel usado para el uso de pesos según distancias (si no se indica es "optimal"). El "rectangular" es k-NN estandar sin pesos. Tipos existentes: "rectangular", "triangular", "epanechnikov", "biweight", "tri-weight", "cos", "inv", "gaussian", "rank" y "optimal“.

Nosotros dejaremos los parámetros por defecto y veremos como ejecutarlo, calcular errores, etc., mediante el uso del conjunto completo como entrenamiento y test por ahora. De esta manera veremos también cómo le afectan algunas variables variando la fórmula e incluso probando con la misma fórmula que obtuvimos para la regresión lineal en la cápsula anterior. También mostraremos cómo aplicar una validación cruzada más adelante.

In [4]:
```
# Tiempo estimado de ejecución: 2 segundos aprox.

%%R

fitKNN <- kknn(HOMA ~ ., data, data)
yprime = fitKNN$fitted.values
cat('\n(fórmula con todas las variables) RMSE:', sqrt(sum((data$HOMA-yprime)^2)/
length(yprime)), "\n") #RECM->en inglés RMSE

fitKNN_lm <- kknn(HOMA ~ BMI+Height+TAGmgDL+Sex+WC+LDLCmgDL+I(BMI^2), data, dat
a)
yprime = fitKNN_lm$fitted.values
cat('(fórmula obtenida con lm) RMSE:', sqrt(sum((data$HOMA-yprime)^2)/length(ypr
ime)), "\n") #RECM->en inglés RMSE
```

```
(fórmula con todas las variables) RMSE: 0.4974704
(fórmula obtenida con lm) RMSE: 0.5062813
```

Según los valores obtenidos, está claro que **la fórmula obtenida para otra técnica no tiene por qué comportarse de igual manera con k-vecinos más cercanos**. En el caso de la regresión lineal se consiguieron mejoras significativas mientras que ahora los resultados incluso empeoran.

Este comportamiento es el esperado. Esta técnica se basa en distancias y normalmente si se consiguen mejoras es porque se eliminan algunas variables que puedan entorpecer o contrarrestar las distancias calculadas para el resto. En lo que sigue, se puede comprobar el resultado final de intentar eliminar algunas variables. Al probar a quitar todas las variables de una en una, la que mayor reducción del error proporciona es "Sex". En una segunda iteración también se puede quitar la variable de presión sistólica "SBP". Y a partir de ahí, seguir quitando variables ya sólo produce un empeoramiento del modelo.

In [5]:
```
# Tiempo estimado de ejecución: 2 segundos aprox.

%%R

fitKNN <- kknn(HOMA ~ .-Sex-SBP, data, data)
yprime = fitKNN$fitted.values
cat('\n(fórmula con todas las variables) RMSE:', sqrt(sum((data$HOMA-yprime)^2)/
length(yprime)), "\n") #RECM->en inglés RMSE
```

```
(fórmula con todas las variables) RMSE: 0.4824407
```

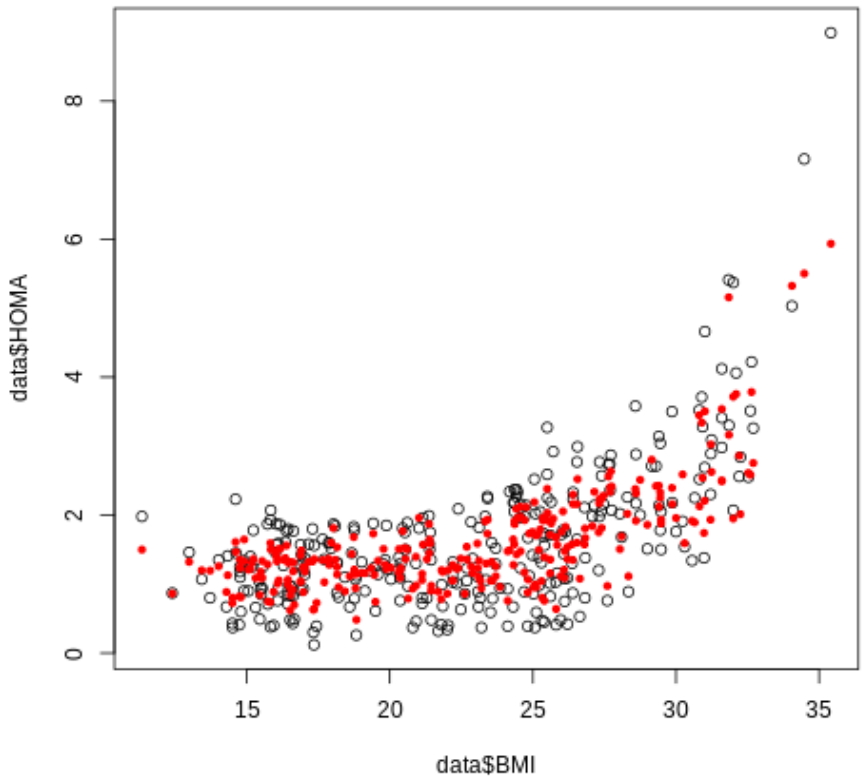
Igualmente, también se puede probar a incluir algún término no lineal que mejore el cálculo de las distancias. En nuestro caso sabíamos que BMI mostraba un comportamiento cuadrático. Vamos a probar a incluirlo.

In [6]:
```
# Tiempo estimado de ejecución: 2 segundos aprox.

%%R

fitKNN <- kknn(HOMA ~ .-Sex-SBP+I(BMI^2), data, data)
yprime = fitKNN$fitted.values
cat('\n(fórmula con todas las variables) RMSE:', sqrt(sum((data$HOMA-yprime)^2)/
length(yprime)), "\n") #RECM->en inglés RMSE
```

```
(fórmula con todas las variables) RMSE: 0.4590836
```

Visualización:

In [7]:
```
# Tiempo estimado de ejecución: 2 segundos aprox.
#yprime = fitKNN$fitted.values

%%R
plot(data$HOMA~data$BMI)
points(data$BMI,yprime,col="red",pch=20)
```



¡¡¡ ¿Y no se nos ha olvidado algo? !!!

¡¡¡ ¿Qué pasa con la NORMALIZACIÓN de los datos? !!!

Efectivamente, es un paso crucial que no debemos olvidar. Si no lo hubiésemos tenido en cuenta, tendríamos que desechar todo lo realizado, normalizar los datos y repetir todo el proceso. Pero... ¿no lo hemos tenido en cuenta? No es nuestro caso, el algoritmo utilizado ya realiza internamente dicha normalización. Si recordamos los parámetros que aceptaba la función $kknn(.)$ como entrada, teníamos $scale = TRUE$, que hace que por defecto la normalización esté activa. Éste parámetro se utiliza para decirle si queremos o no que normalice los datos antes de aplicar el algoritmo (TRUE o FALSE).

No todas las implementaciones de k-vecinos más cercanos permiten realizar la normalización como parte del propio algoritmo. Por ello, es esencial prestar mucha atención a éste aspecto y realizar la normalización a mano si la versión que utilicemos no los normaliza por sí sola.

5. APLICACIÓN DE M5 AL PROBLEMA

En esta sección veremos cómo aplicar M5 a nuestro problema de estimación de la insulino-resistencia. La implementación de M5 que vamos a utilizar está incluida como parte de la librería *Cubist* de R, que ya hemos instalado en nuestro entorno. En realidad, Cubist es una extensión de M5 que incluye una serie de iteraciones de M5 basadas en *boosting* para combinar varios árboles. La combinación de árboles o *ensembles* representa técnicas más complejas o avanzados que no son objeto de este MOOC de introducción a la bioinformática. Además, tampoco está del todo claro que Cubist funcione significativamente mejor que su algoritmo base, y definitivamente no es mejor que un *bagging* de M5 (técnica alternativa al *boosting* para igualmente combinar varios árboles). Véase el estudio de *Gacto et. al.* 2019 donde se ve que M5 se muestra competitivo con respecto a Cubist y un *bagging* de M5 es la técnica con mejor puntuación - *Sección de Bibliografía*.

Nos centraremos, por lo tanto, en M5 indicándole mediante parámetro a la función $cubist(.)$ que el número de iteraciones del *boosting* es igual a 1. De esta manera, tenemos justamente el comportamiento del M5 original. La función es conocida como $cubist(.)$ y sigue la siguiente sintaxis:

$$cubist(x, y, committees = 1)$$

```
(Aplica Cubist a los datos de entrenamiento con X's e Y separados, y obtiene un único
árbol para el consenso entre modelos -> M5)
```

x – una matriz de datos con los valores de las variables de entrada. Se permiten valores perdidos pero únicamente numéricos. También se permiten valores categóricos; *y* – un vector numérico con los correspondientes valores de la variable de salida; *committees* – un entero, número de modelos para el consenso (si no se indica es 1 -> M5).

Al igual que con k-nn veremos como ejecutarlo, calcular errores, y en este caso visualizar el árbol, mediante el uso del conjunto completo separando X's e Y. No obstante, M5 ya selecciona las variables más relevantes durante el propio proceso de aprendizaje, por lo que en este caso no tendremos que hacerlo manualmente. De hecho, podemos ver cómo la función $cubist(.)$ no incluye ninguna manera de especificar la *fórmula*. Igualmente, mostraremos cómo aplicar una validación cruzada más adelante junto con k-nn.

In [8]:
```
# Tiempo estimado de ejecución: 2 segundos aprox.

%%R

tam <- length(names(data))
fitM5 <- cubist(x = data[, -tam], y = data$HOMA) #committees = 1
yprime = predict(fitM5,data)
cat('\n(M5) RMSE:', sqrt(sum((data$HOMA-yprime)^2)/length(yprime)), "\n") #RECM->en inglés RMSE

summary(fitM5)
```

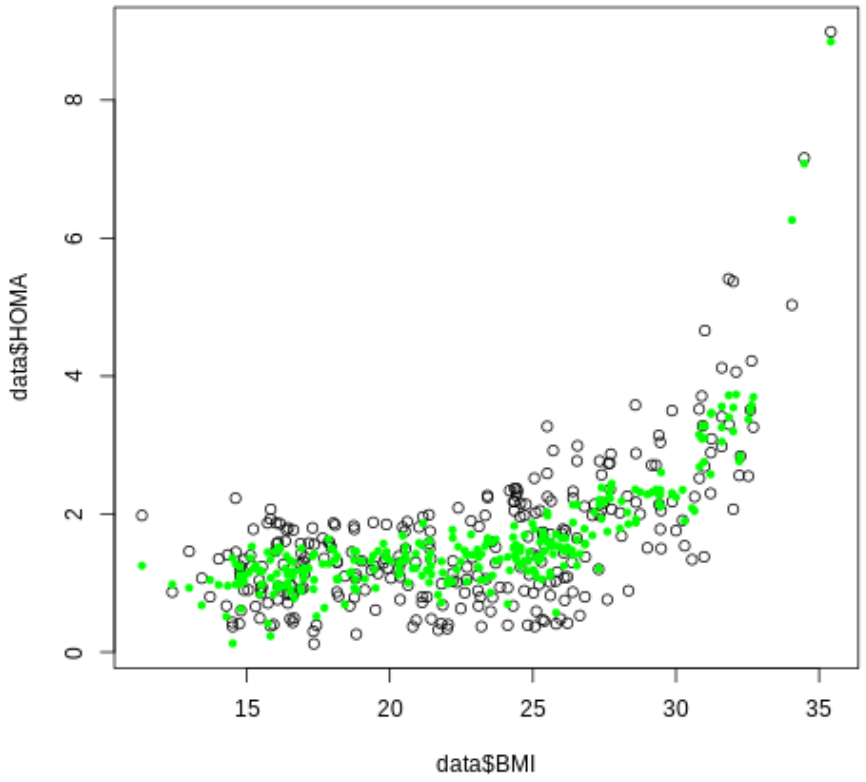
```
(M5) RMSE: 0.5392671

Call:
cubist.default(x = data[, -tam], y = data$HOMA)


Cubist [Release 2.07 GPL Edition]  Sun Mar  7 22:36:17 2021
---------------------------------

    Target attribute `outcome'

Read 292 cases (16 attributes) from undefined.data

Model:

  Rule 1: [120 cases, mean 1.309, range 0.26 to 3.27, est err 0.428]

    if
        Age <= 9.5
        BMI <= 30.65
    then
        outcome = -2.244 + 0.069 BMI + 0.154 Age + 0.39 Sex + 0.66 Height
                  - 0.0047 WC + 0.0017 TAGmgDL + 0.0027 HDLCmgDL
                  - 0.0012 LDLCmgDL

  Rule 2: [118 cases, mean 1.310, range 0.12 to 2.92, est err 0.419]

    if
        Age > 9.5
        BMI <= 27.3
    then
        outcome = -1.301 + 0.038 BMI + 0.0013 Sedentary + 0.67 Height
                  + 0.17 Tanner + 0.0053 HDLCmgDL - 0.0024 LDLCmgDL
                  + 0.0018 TAGmgDL - 0.0007 WC

  Rule 3: [35 cases, mean 2.217, range 0.76 to 3.58, est err 0.532]

    if
        BMI > 27.3
        BMI <= 30.65
    then
        outcome = -0.678 + 0.044 BMI + 0.79 Height + 0.0062 HDLCmgDL
                  - 0.0028 LDLCmgDL + 0.0022 TAGmgDL + 0.0006 Sedentary

  Rule 4: [18 cases, mean 3.376, range 1.38 to 5.41, est err 0.733]

    if
        BMI > 30.65
        BMI <= 32.1
    then
        outcome = -12.599 + 0.439 BMI - 0.0154 WC + 1.45 Height + 0.0081 SBP
                  + 0.0032 TAGmgDL + 0.18 Sex + 0.0046 HDLCmgDL - 0.002 LDLCmgD
L

  Rule 5: [9 cases, mean 4.458, range 2.55 to 8.99, est err 0.622]

    if
        BMI > 32.1
    then
        outcome = -58.467 + 1.901 BMI


Evaluation on training data (292 cases):
```

```
Average  |error|              0.584
Relative |error|               0.79
Correlation coefficient        0.73


    Attribute usage:
      Conds  Model

      100%   100%    BMI
       79%    40%    Age
              97%    Height
              97%    TAGmgDL
              97%    HDLCmgDL
              97%    LDLCmgDL
              85%    WC
              51%    Sedentary
              46%    Sex
              39%    Tanner
               6%    SBP


Time: 0.0 secs
```

En este caso, se puede ver cómo se obtiene un modelo con RECM de 0.5392671 que incluye distintas líneas (o modelos lineales) atendiendo a distintos puntos de corte en BMI y Age. Desde que empezamos con el problema de insulino-resistencia hemos visto que estas dos variables son probablemente las más determinantes. Tiene mucho sentido que M5 las haya seleccionado para tomar las decisiones que conforman el árbol.

Visualización:

In [9]:
```
#yprime = predict(fitM5,data)

%%R
plot(data$HOMA~data$BMI)
points(data$BMI,yprime,col="green",pch=20)
```



6. VALIDACIÓN CRUZADA PARA AMBAS TÉCNICAS: COMPARACIÓN Y ELECCIÓN DE LOS MODELOS

En este punto, ya hemos llegado a ejecutar todos los métodos estudiados. Hemos obtenido distintos valores de error, *RECM*. Sin embargo, no podemos apresurarnos para escoger un posible método ganador. Como bien sabemos, necesitamos realizar una estimación del error real para nuestro problema realizando una validación cruzada, y comparando por los errores medios obtenidos en los conjuntos de *test*. Ya vimos cómo calcularlo para la regresión lineal multi-variable, con la que obtuvimos un *RECM* de 0.6956061. En esta última sección mostramos como aplicar validación cruzada con k-nn y M5.

In [10]:
```
# Tiempo estimado de ejecución: 2 segundos aprox.
### KNN ###

%%R
set.seed(123456)  #fijamos la misma semilla utilizada con regresión lineal
                  #para conseguir las mismas particiones
k <- 5
data$kfold <- sample(1:k, nrow(data), replace = T)

performances <- c()
# Una iteración por fold
for (fold in 1:k){
  # Se crea el conjunto de entrenamiento para la iteración
  training_set <- data[data$kfold != fold,]
  nombres <- names(training_set)
  tam <- length(nombres)-1
  training_set <- training_set[,nombres[1: tam]]

  # Se crea el conjunto de test para la iteración
  testing_set <- data[data$kfold == fold,]
  nombres <- names(testing_set)
  tam <- length(nombres)-1
  testing_set <- testing_set[,nombres[1: tam]]

  ## Entrenando el modelo para la iteración
  model <- kknn(HOMA ~ .-Sex-SBP+I(BMI^2), training_set, testing_set)

  ## Calculando el error de test
  yprime = model$fitted.values  #yprime <- predict(model, testing_set)
                                #cambiamos porque no funciona predict con kknn,
                                #ya que el propio kknn es el predict
  RMSE <- sqrt(sum((testing_set$HOMA-yprime)^2)/length(yprime))

  # Se añade el RECM a la lista de errores
  performances[fold] <- RMSE
}

#Eliminamos la columna artificial añadida para kfold
#(para que no acumule columnas si se ejecuta varias veces)
nombres <- names(data)
tam <- length(nombres)-1
data <- data[,nombres[1: tam]]

cat("RECM medio en test para 5-fcv en K-nn:", mean(performances))
```

```
RECM medio en test para 5-fcv en K-nn: 0.7929245
```

In [11]:
```
# Tiempo estimado de ejecución: 2 segundos aprox.
### M5 ###

%%R
set.seed(123456)  #fijamos la misma semilla utilizada con regresión lineal
                  #para conseguir las mismas particiones
k <- 5
data$kfold <- sample(1:k, nrow(data), replace = T)

performances <- c()
# Una iteración por fold
for (fold in 1:k){
  # Se crea el conjunto de entrenamiento para la iteración
  training_set <- data[data$kfold != fold,]
  nombres <- names(training_set)
  tam <- length(nombres)-1
  training_set <- training_set[,nombres[1: tam]]

  # Se crea el conjunto de test para la iteración
  testing_set <- data[data$kfold == fold,]
  nombres <- names(testing_set)
  tam <- length(nombres)-1
  testing_set <- testing_set[,nombres[1: tam]]

  ## Entrenando el modelo para la iteración
  tam <- length(names(training_set))
  model <- cubist(x = training_set[, -tam], y = training_set$HOMA) #committees =
1

  ## Calculando el error de test
  yprime <- predict(model, testing_set)
  RMSE <- sqrt(sum((testing_set$HOMA-yprime)^2)/length(yprime))

  # Se añade el RECM a la lista de errores
  performances[fold] <- RMSE
}

#Eliminamos la columna artificial añadida para kfold
#(para que no acumule columnas si se ejecuta varias veces)
nombres <- names(data)
tam <- length(nombres)-1
data <- data[,nombres[1: tam]]

cat("RECM medio en test para 5-fcv en M5:", mean(performances))
```

```
RECM medio en test para 5-fcv en M5: 0.7807851
```

A continuación se muestran los resultados que obtuvimos cuando estuvimos seleccionando las variables para fijar la fórmula o viendo cómo se ejecuta M5, es decir, sin validación cruzada:

- Regresión lineal sin validación - RECM: 0.6507716
- k-nn sin validación - RECM: 0.4590836
- M5 sin validación - RECM: 0.5392671

En este caso hubiésemos escogido k-nn como técnica para aplicar a nuestro problema (si pensamos sólo en precisión y no en interpretación).

Sin embargo comprobemos qué resulta cuando realizamos la validación cruzada:

- Regresión lineal con validación - RECM: 0.6956061
- k-nn con validación - RECM: 0.7929245
- M5 con validación - RECM: 0.7807851

Saque sus propias conclusiones. De esta manera, el paso final cuando se ejecutan distintas técnicas para resolver un problema dado sería comparar los errores medios en test obtenidos por validación cruzada, seleccionar la mejor (o las mejores) atendiendo a dichos valores y/o a criterios de interpretabilidad, y una vez fijada la técnica volver a aplicarla con todos los datos para obtener un modelo final candidato.

REFERENCIAS BIBLIOGRÁFICAS

KNN

- Hechenbichler K. and Schliep K.P. (2004) Weighted k-Nearest-Neighbor Techniques and Ordinal Classification, Discussion Paper 399, SFB 386, Ludwig-Maximilians University Munich (http://www.stat.uni-muenchen.de/sfb386/papers/dsp/paper399.ps (http://www.stat.uni-muenchen.de/sfb386/papers/dsp/paper399.ps))
- Samworth, R.J. (2012) Optimal weighted nearest neighbour classifiers. Annals of Statistics, 40, 2733-2763. (http://www.statslab.cam.ac.uk/~rjs57/Research.html (http://www.statslab.cam.ac.uk/~rjs57/Research.html))

M5

- Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. An Introduction to Statistical Learning with Applications in R Springer, 2013 (**Chapter 08**)
- J. R. Quinlan, Learning With Continuous Classes, In Proc. of the 5th Australian Joint Conf. on Artificial Intelligence, pp. 343-348, 1992
- Y. Wang and I.H. Witten, Inducing Model Trees for Continuous Classes, In Proc. of the 9th European Conf. on Machine Learning, pp. 128-137, 1997

Librerías

- Usando rpy2 en notebooks: https://rpy2.github.io/doc/latest/html/notebooks.html (https://rpy2.github.io/doc/latest/html/notebooks.html)
- Usando read.csv de R: https://www.rdocumentation.org/packages/utils/versions/3.6.2/topics/read.table (https://www.rdocumentation.org/packages/utils/versions/3.6.2/topics/read.table)
- Usando kknn de R: https://cran.r-project.org/web/packages/kknn/index.html (https://cran.r-project.org/web/packages/kknn/index.html)
- Usando Cubist de R: https://cran.r-project.org/web/packages/Cubist/index.html (https://cran.r-project.org/web/packages/Cubist/index.html)

REFERENCIAS ADICIONALES

- M.J. Gacto, J.M. Soto-Hidalgo, J. Alcalá-Fdez, and R. Alcalá (2019). Experimental Study on 164 Algorithms Available in Software Tools for Solving Standard Non-Linear Regression Problems. IEEE Access 7, 2019, pp. 108916-108939; https://doi.org/10.1109/ACCESS.2019.2933261 (https://doi.org/10.1109/ACCESS.2019.2933261)