clase 5 de Mayo

Mariano Dominguez

April 29, 2018

Análisis exploratorio y curación de datos

- Mariano Dominguez @ IATE-OAC-UNC & CONICET
- Edgardo Hames @ Bitlogic
- Gabriel Miretti @ Bitlogic

Figure 1: Diplodatos @ FaMAF

Un breve vistazo de Aprendizaje supervisado

- Tenemos un conjunto de variables medidas que las llamaremos **inputs**. Estas tienen alguna influencia en una o mas variables que llamaremos **outputs**. La **meta** es utilizar los inputs **para predecir** los valores de los outputs.
- En la literatura estadistica los inputs son frecuentemente llamados como **predictores**, un termino que puede usarse equivalentemente a inputs, y mas clasicamente seran las **variables independientes**.
- En la literatura de reconocimiento de patrones se prefiere el termino **features** . Los outputs son llamados tambien las **respuestas**, o clasicamente las **variables dependientes**.

Modelos Lineales y Cuadrados minimos

Los modelos lineales son una de las herramientas estadisticas mas importantes. Dado un vector de inputs $X^T = (X_1, X_2, ..., X_p)$, se puede predecir el output Y via el siguiente modelo:
$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$

$\hat{\beta}_0$ es la ordenada al origen, tambien conocida como el **bias** en aprendizaje automatico (machine learning). A menudo es conveniente incluir la variable constante 1 en X, y de manera similar $\hat{\beta}_0$ en el vector de coeficientes $\hat{\beta}$, y entonces escribir el modelo lineal en forma de un producto interno vectorial: $\hat{Y} = X^T \hat{\beta}$.

Especificar modelos estadisticos como regresion es muy simple en R. Simplemente escribiendo "y $\sim$ model", donde "y" sera la variable de respuesta y model seran todos los predictores a ser incluidos en el modelo. Para la regresion lineal utilizaremos la funcion lm.

Una simple Regresion Lineal 1:

Utilizaremos un conjunto de datos de 92 estrellas del grupo de las Hyades las cuales filtraremos por coordenadas y errores de medicion en la paralaje.

```
loc <- "http://astrostatistics.psu.edu/datasets/"
hip <- read.table(paste(loc,"HIP_star.dat",sep=""),
header=T,fill=T)
attach(hip)
filter1 <- (RA>50 & RA<100 & DE>0 & DE<25)
filter2 <- (pmRA>90 & pmRA<130 & pmDE>-60 & pmDE< -10)
filter <- filter1 & filter2 & (e_Plx<5)
sum(filter)
```

```
## [1] 92
```

Una simple Regresion Lineal 2:

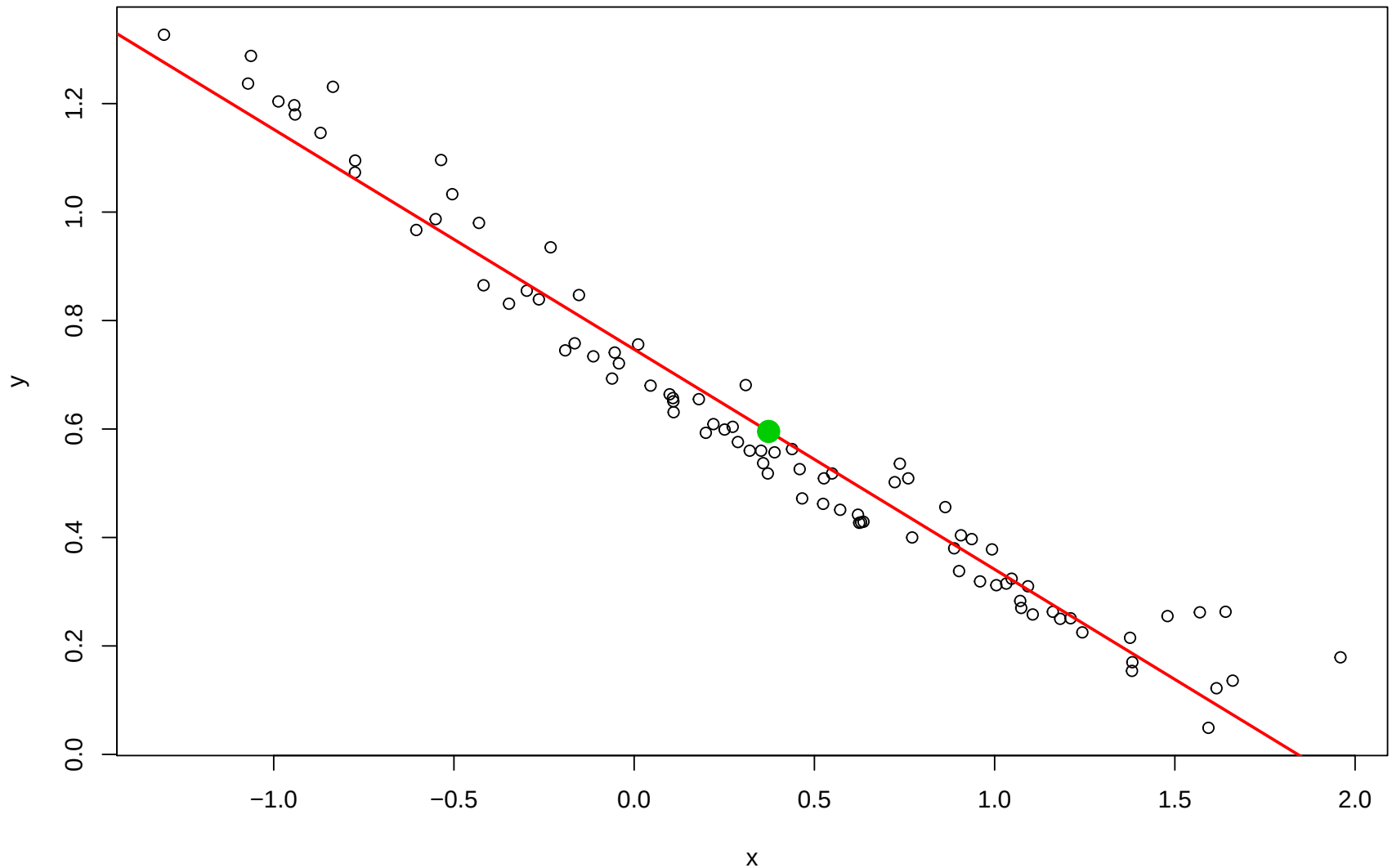
Un breve ejemplo de una regresion linear entre $B - V$ y $\log(L)$.

```
mainseqhyades <- filter & (Vmag>4 | B.V<0.2)
logL <- (15-Vmag-5 * log10(Plx)) / 2.5
x <- logL[mainseqhyades]
y <- B.V[mainseqhyades]
regline <- lm(y~x)
summary(regline)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.08578 -0.04846 -0.01741  0.04004  0.22711
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

Una simple Regresion Lineal 3:

```
plot(x,y)
abline(regline,lwd=2,col=2)
points(mean(x),mean(y),col=3,pch=20,cex=3)
```



Representaciones

Aqui nos preguntamos algo comunmente pasado por alto; esto es que si los datos son mejor analizados en la forma en la cual vienen dados (hint: la respuesta es frecuentemente NO, vease el ejemplo anterior).

Esta es una pregunta sobre **representaciones** cual es la mejor forma de analizar los datos de forma de resaltar los features de interes?

La **meta final** seria cocinar el un conjunto cualquiera de mediciones a un conjunto menor pero mas independiente, liberando asi al analisis posterior (el ML) de tener que redescubrir sus redundancias.

Una buena representacion nos puede llevar a muy lejos en resolver un problema dificil, como una mala puede condenar un esfuerzo bien intencionado.

Transformadas ortogonales 1:

Estas son simples de deshacer, esto es importante por que no queremos que nuestras transformaciones de los datos tiren informacion.

Una matriz es ortogonal si su inversa es igual a su transpuesta:

$$M^T.M = I$$

donde la matriz transpuesta se define por

$$M_{ij}^T = M_{ji}$$

e

$$I$$

es la matriz identidad con unos en la diagonal y zeros en todas las otras componentes.

La multiplicacion de una matriz ortogonal por un vector de datos define una transformacion ortogonal de un vector.

Transformadas ortogonales 2:

Para una matriz compleja la matriz **adjunta** es la complejo conjugado de la transpuesta

$$M^{\dagger}_{ij} = M^{*}_{ji}$$

.

Si la matriz adjunta es la inversa,

$$M^{\dagger}.M = I$$

entonces

$$M$$

es unitaria.

Una importante propiedad de la matriz adjunta es que intercambia el orden del producto de matrices:

$$(A.B)^{\dagger} = B^{\dagger}.A^{\dagger}$$

Transformadas ortogonales 3:

Ahora consideremos una transformacion linear del vector columna

$$\bar{x}$$

a uno nuevo

$$\bar{y} = M.\bar{x}$$

. La norma Euclidiana de

$$\bar{x}$$

es su longitud medida por la suma de los cuadrados de sus elementos.

$$|\bar{x}|^2 = \bar{x}^{\dagger}.\bar{x} = \sum_{i=1}^{N} x_i^*.x_i = \sum_{i=1}^{N} |x_i|^2$$

.

Si

$$M$$

es una matriz unitaria, entonces la norma de y es:

Transformadas ortogonales 4:

$$
\begin{aligned}
|\bar{y}|^2 &= |(M.\bar{x})^\dagger.(M.\bar{x})| \\
&= |(\bar{x}^\dagger.M^\dagger).(M.\bar{x})| \\
&= |\bar{x}^\dagger.(M^\dagger.M).\bar{x}| \\
&= |\bar{x}^\dagger.\bar{x}| \\
&= |\bar{x}|^2
\end{aligned}
$$

Entonces una transformacion unitaria (u ortogonal) preserva la norma del vector. Solamente rota los datos a una nueva posicion, pero no cambia su distancia del origen. Esto significa que puede reordenar los puntos, pero no hacerlos desaparecer por ej.

Transformadas de Fourier:

La **Transformada discreta de Fourier (DFT)** es un ejemplo familiar de transformacion unitaria. Dado un vector de datos $x_0, x_1, ..., x_{N-1}$ la DFT se define por:

$$X_f = \frac{1}{\sqrt{(N)}} \sum_{n=0}^{N-1} \exp(2\pi i f n/N) x_n = \sum_{n=0}^{N-1} M_{fn} x_n = M.\bar{x}$$

y la correspondiente transformada inversa:

$$x_n = \frac{1}{\sqrt{(N)}} \sum_{n=0}^{N-1} \exp(-2\pi i f n/N) X_f$$

donde los X_f son los coeficientes de la expansion en terminos de sinusoides.

Transformadas rapidas de Fourier:

Computar las DFT requiere multiplicar el vector de datos por la matriz de transformacion, por lo que computar un elemento requiere N multiplicaciones y sumas, y como hay N elementos esto parece ser un algoritmo de orden N^2.

Remarcablemente y significantemente este no es el caso, notese que la suma DFT puede dividirse en dos sumas (una con elementos pares y otra impares) sucesivamente hasta llegar a trasnformadas de un solo punto. Reensamblando nos lleva a un algoritmo de orden $(Nlog_2(N))$ pasos.

La encarnacion moderna de esta idea es llamada la Transformada rapida de Fourier (Cooley and Tuckey 1965@IBM) que es uno de los algoritmos mas importantes de la matematica numerica, ver http://cacs.usc.edu/education/phys516/c12-2.pdf

Otras posible transformadas:

Wavelets son otra familia de transformadas ortgonales que generalizan las transformadas de Fourier de una forma importante introduciendo localidad.

Las funciones trigonometricas estan definidas en todos lados y esto las hace particularmente buenas para describir propiedades globales, pero por lo mismo no son buenas para describir comportamientos localizados.

Estas transformadas y otras (Wigner) pueden generalizarse a varias dimensiones y pueden aplicarse a imagenes, volumenes etc. Los coeficientes de las mismas constituyen conjuntos de nuevos features que pueden proveer la representacion mas adecuada para algunos problemas.

Esto puede originar una rapida proliferacion de features, vease por ejemplo https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2478650/, lo que nos conduce a pensar en como seleccionar o generar los adecuados.

Componentes Principales 1:

Las **wavelets** se construyen basadas en la suposicion de que tiempo y frecuencia son las ejes interesantes sobre los cuales hay que mirar una senal. Esto no es necesariamente cierto y no se aplica para datos que no tienen un orden espacial y temporal particularmente.

En lugar de disenar una trasnformada para aplicarla a todos lo datos posibles, seria mejor adaptar una transformada que nos de la mejor representacion posible para un dado conjunto de datos.

Sea

$$\bar{x}$$

de nuevo un vector de mediciones, e

$$\bar{y} = M.\bar{x}$$

una transformacion a un nuevo set de variables (features) con propiedades mas deseables. La **matriz de covarianza** de

$$\bar{y}$$

Componentes Principales 2:

esta definida por:

$$C_y = < (\bar{y} - < \bar{y} >).(\bar{y} - < \bar{y} >)^T >$$

donde el **producto externo** de dos vectores columna

$$\bar{A}$$

y

$$\bar{B}$$

es:

$$(\bar{A}.\bar{B}^T)_{ij} = A_i B_j$$

, y el promedio se toma sobre un conjunto de mediciones.

Un pedido razonable es pedir que la matriz de covarianza de

$$\bar{y}$$

sea diagonal, ya que por lo tanto cada uno de sus elementos no este

Componentes Principales 3:

Para encontrar la transformacion requerida, la matriz de covarianza de

$$\bar{y}$$

esta relacionada con la de

$$\bar{x}$$

.

$$C_y = < (\bar{y} - < \bar{y} >).(\bar{y} - < \bar{y} >)^T >$$
$$= < [M.(\bar{x} - < \bar{x} >)].[M.(\bar{x} - < \bar{x} >)]^T >$$
$$= < [M.(\bar{x} - < \bar{x} >)].[(\bar{x} - < \bar{x} >)^T.M^T] >$$
$$= M. < (\bar{x} - < \bar{x} >).(\bar{x} - < \bar{x} >)^T > .M^T$$
$$= M.C_x.M^T$$

Componentes Principales 4:

Debido a que

$$C_x$$

es una matriz real y simetrica, es posible encontrar un conjunto ortonormal de autovectores.

Ahora considere que ocurre si las columnas de

$$M^T$$

se toman como esos autovectores. Luego de multiplicar por

$$C_x$$

cada autovector vuelve multiplicado por su correspondiente autovalor. Entonces debido a la ortonormalidad, la multiplicacion de esta matriz por

$$M$$

nos da zeros fuera de la diagonal y los autovalores en la diagonal, por lo que la matriz

Componentes Principales 4:

Si existen correlaciones entre los elementos de

$$\bar{x}$$

entonces algunos de los autovalores sera cero (o un numero muy chico), estas componentes de

$$\bar{y}$$

seran deshechadas en los analisis subsiguientes.

Utilizar la matriz de covarianza de un conjunto de datos para encontrar una transformacion a nuevas variables que no estan correlacionadas es llamado: **Analisis de Componentes Principales**

Este tipo de ideas a llevado a crear tecnicas similares conocidas como TLAs (Three Letters Acronyms) entre los que podemoss mencionar: Karhunnen-Loeve Transform (KLT), Independent Component Analysis (ICA), y que conduce a las arquitecturas

Componentes Principales 5:

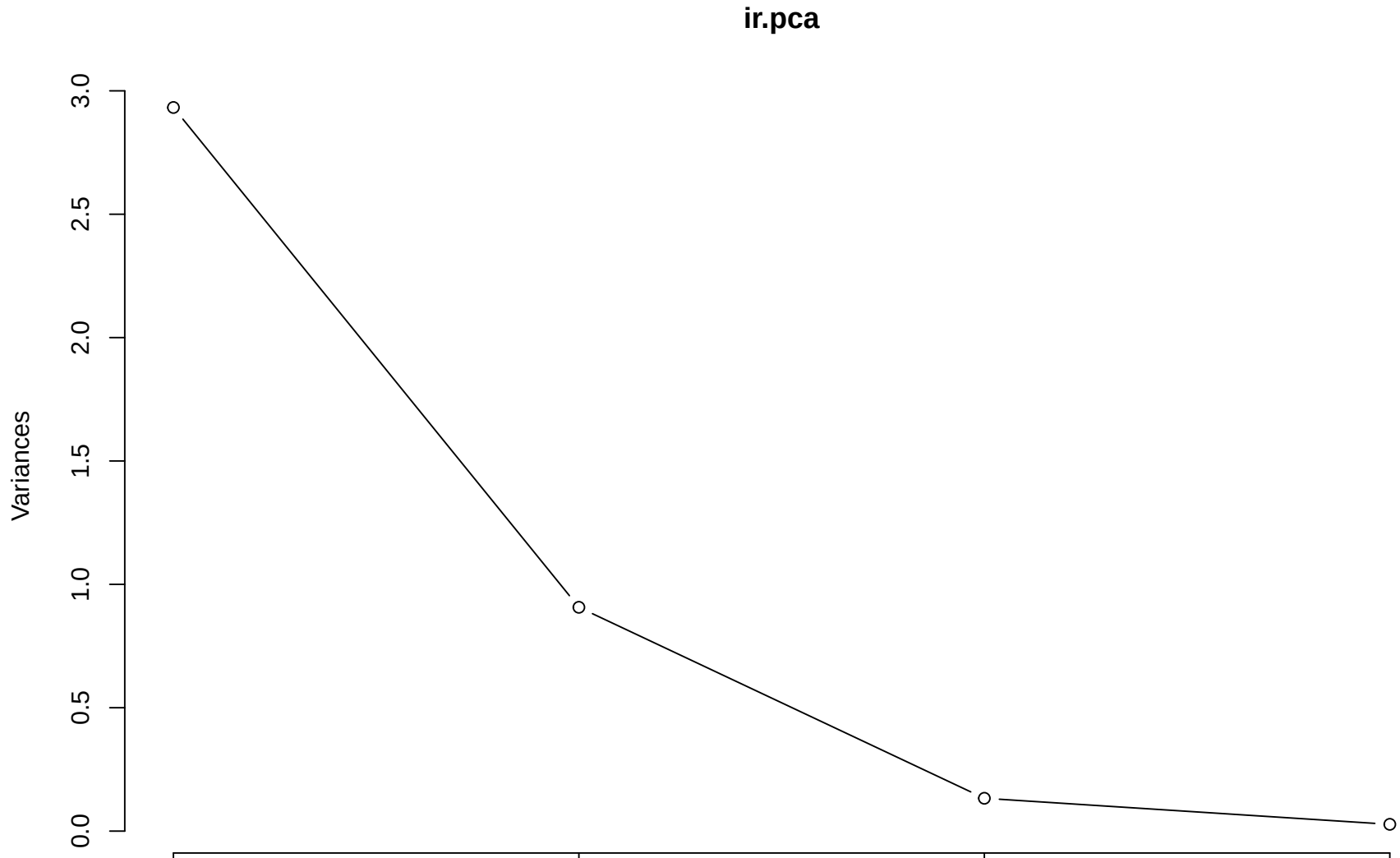
```
# log transform
data("iris")
log.ir <- log(iris[, 1:4])
ir.species <- iris[, 5]

# apply PCA - scale. = TRUE is highly
# advisable, but default is FALSE.
ir.pca <- prcomp(log.ir, center = TRUE, scale. = TRUE)
# print method
print(ir.pca)
```

```
## Standard deviations (1, .., p=4):
## [1] 1.7124583 0.9523797 0.3647029 0.1656840
##
## Rotation (n x k) = (4 x 4):
##                     PC1         PC2        PC3          P
## Sepal.Length  0.5038236 -0.45499872  0.7088547  0.191475
## Sepal.Width  -0.3023682 -0.88914419 -0.3311628 -0.091254
## Petal.Length  0.5767881 -0.03378802 -0.2192793 -0.786187
```

Componentes Principales 6:

EL metodo print aplicado nos devuelve la desviacion standart de cada uno de los cuatro coeficientes componentes principales y los coeficientes de las combinaciones lineales de cada variable en el dataset.



Componentes Principales 7:

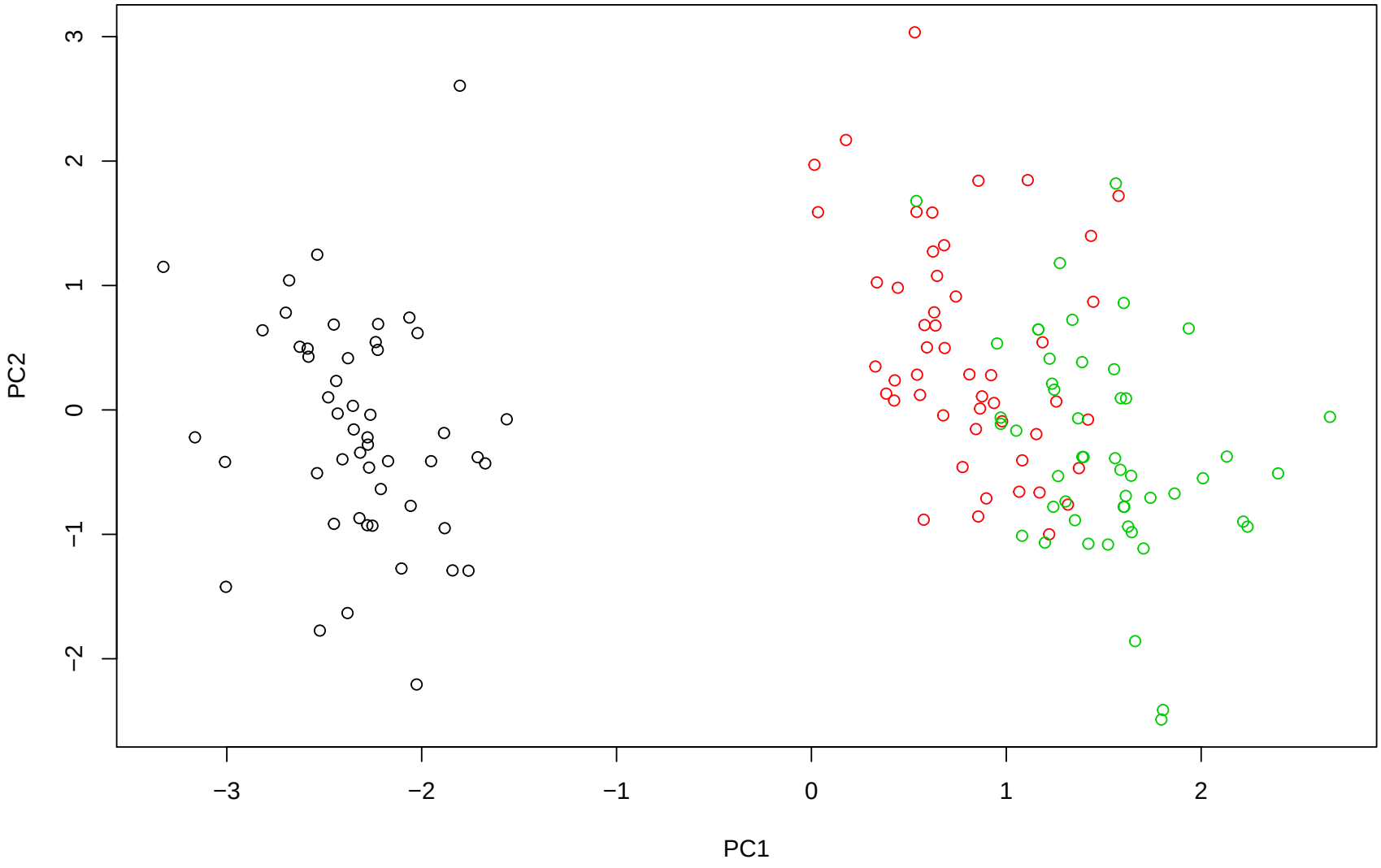
```
# summary method
summary(ir.pca)
```

```
## Importance of components%s:
##                             PC1    PC2     PC3     PC4
## Standard deviation     1.7125 0.9524 0.36470 0.16568
## Proportion of Variance 0.7331 0.2268 0.03325 0.00686
## Cumulative Proportion  0.7331 0.9599 0.99314 1.00000
```

El metodo summary describe la importancia de las componentes principales. La primera columna describe la importancia de las PCs, la segundo la fracccion de la varianza en los datos, explicada por cada componente y la tercera porcion cumulativa de la varianza de los datos.

Componentes Principales 8:

```
plot(ir.pca$x[,1:2], col= ir.species)
```



Seleccion de Features

Un poco de clasificacion usando los vecinos:

- Se uttiliza para clasificar nuevos ejemplo asignandoles la clase de los ejemplos mas similares.
- Se utiliza exitosamente en reconocimiento de caracteresen imagenes y video (CV), ver por ejemplo <www.opencv.org>
- predecir si una dada persona le gustara cierta pelicula como por ej en el Netflix challenge https://www.kaggle.com/netflix-inc/netflix-prize-data.
- Identificar patrones en datos geneticos, para detectar proteinas especificas o enfermedades.

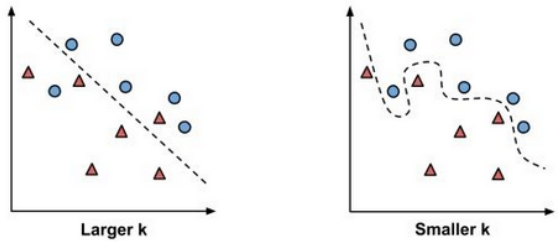
En general los NN (Nearest Neighbours) clasificadores son muy adecuados cuando las relaciones entre features y clases objetivo son complicadas, numerosas o dificiles de entender.

The kNN algorithm:

- Fortalezas: Simple y efectivo, rapido de entrenar
- Debilidades: no da un modelo, lento para clasificar, costoso en memoria.
- Para cada dato en el conjunto a clasificar, kNN identifica k datos en el conjunto de datos ya clasificado (de entrenamiento) que estan “cerca”, donde k es un numero especificado.
- Para localizar un punto como vecino se requiere una funcion distancia, como la distancia Euclidiana o la distancia Manhattan, leer sobre esto usando ?dist.
- El dato a clasificar es asignado a la clase de la mayoria de sus k-vecinos.

Eligiendo el numero apropiado de vecinos:

- Esta relacionado con un problema muy importante. Elegir un k grande puede reducir el impacto causado por datos muy ruidosos, pero asi tambien se corre el riesgo de ignorar patrones locales importante (superficie o borde de decision).



In practice, choosing k depends on the difficulty of the concept to be learned and the number of records in the training data. Typically, k is set somewhere between 3 and 10. One common practice is to set k equal to the square root of the number of

Preparando los datos para un algoritmo:

- Los Features son tipicamente transformados a una medida estandar previamente a la aplicacion de un algoritmo como kNN. La razon de esto es que la formula de distancia es afectada por como son medidos los features.
- En particular si ciertos features tienen valores mucho mayores que otros, las mediciones de distancia seran muy fuertemente dominadas por los valores mas grandes.
- Lo que se necesita es una manera de lograr que todos los features contribuyan equitativamente a la formula de distancia.

Rescalando los features

El metodo tradicional para kNN es la **minimizacion minmax**. Este proceso transforma cada feature de forma tal que todos sus valores caigan entre 0 y 1.

Otra transformacion comun es llamada la **z-score normalization**. Que consiste en substraer a cada dato la media del mismo y dividirla por su desviacion estandart, quedando entonces en algun rango de numeros negativos a positivos.

La formula de la distancia Euclidea no esta definida para datos nominales, por lo que es necesario convertir cada feature nominal en algun formato numerico. Como por ejemplo con **dummy coding**.

Diagnosticando Cancer:

- Investigaremos ahora la utilidad del ML para detectar cancer aplicando el algoritmo kNN a mediciones de biopsias de mujeres, utilizando el conjunto de datos "Breast Cancer Winscosin Diagnostic" del UCI ML Repository http://archive.ics.uci.edu/ml que incluye 569 ejemplos de biopsias, en cada una se midieron 32 features (diferentes caracteristicas de las nucleos celulares) y el diagnostico codificado como M (Maligno) o B (Benigno).

```
data <- read.csv("http://archive.ics.uci.edu/ml/machine-lea
data <- data[-1]
str(data)
```

```
## 'data.frame':    569 obs. of  31 variables:
##  $ V2 : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2
##  $ V3 : num  18 20.6 19.7 11.4 20.3 ...
##  $ V4 : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ V5 : num  122.8 132.9 130 77.6 135.1 ...
##  $ V6 : num  1001 1326 1203 386 1297 ...
```

Entendiendo los datos:

Independientemente del metodo de ML aplicado, las variables de identificacion **deben** ser excluidas. No hacerlo puede llevar a hallazgos erroneos a causa de que la identificacion puede ser utilizada para predecir muy bien. La siguiente variable, el diagnostico es de particular interes, por que es lo que se quiere predecir.

```
table(data$V2)
```

```
##
##   B   M
## 357 212
```

Ya que estamos miremos el resto de las variables, sus rangos etc.

```
summary(data)
```

```
##  V2             V3               V4               V5
##  B:357   Min.   : 6.981   Min.   : 9.71   Min.   : 43.79
##  M:212   1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17
##          Median :13.370   Median :18.84   Median : 86.24
```

Transformacion de los datos:

Necesitamos crear una funcion normalizacion en R:

```
normalize <- function(x) {
  return ((x-min(x))/(max(x)-min(x)))
}
```

Despues de ejecutar el codigo previo, la funcion esta disponible para sus uso. Veamos si funciona en algunos vectores.

```
normalize(c(1,2,3,4,5))
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

```
normalize(c(10,20,30,40,50))
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

No podemos aplicar la funcion a los features numericos del dataframe directamente.

Transformacion de los datos:

La funcion lapply() de R toma una lista y aplica una funcion a cada elemento de la lista.

```
data_n <- as.data.frame(lapply(data[2:31], normalize))
summary(data_n$V3)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.2233  0.3024  0.3382  0.4164  1.0000
```

```
summary(data_n$V8)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1397  0.2247  0.2606  0.3405  1.0000
```

Bingo! En ausencia de nuevos datos de laboratorio, vamos a simular este escenario dividiendo nuestros datos en una **muestra de entrenamiento** que usaremos para construir el modelo kNN y una **muestra de validacion** que usaremos para medir la presicion predictiva del mismo.

Entrenando un clasificador:

Notese que dichos conjuntos de datos deben ser representativos del conjunto de datos, i.e. **metodos de muestreo aleatorios**!

```
data_train <- data_n[1:469, ]
data_test  <- data_n[470:569, ]
```

Excluimos la variable objetivo (Benigno/Maligno), pero necesitamos guardar estos factores en vectores!

```
data_train_labels <- data[1:469, 1]
data_test_labels  <- data[470:569, 1]
```

Para el algoritmo kNN la fase de entrenamiento no involucra construir un modelo, para clasificar nuestros datos de validacion utilizaremos el paquete class, instalarlo ia!

La datps de validacion son clasificados tomando los votos entre los k vecinos mas cercanos del conjunto de entrenamiento. Si hay empate se decide aleatoriamente. Entonces usamos la funcion knn() para clasificar.

Evaluando la performance del modelo.

```
library(class)
data_test_pred <- knn(train=data_train, test=data_test, cl=
```

El siguiente paso en el proceso es evaluar como las clases predichas en data_test_pred se condicen con los valores verdaderos en el vector data_test_labels.

```
library(gmodels)
CrossTable(x=data_test_labels, y=data_test_pred, prop.chisq
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
```

Ejercicios.

Los valores correctos estan en la diagonal de la matriz, 98% de precision para unas pocas lineas de R!

- Mejore el rendimiento utilizando una normalizacion con z-scores provista por la funcion scale() de R.
- Pruebe algunos valores alternativos de k=1, 5, 11, 15, 21 y seleccione el mejor valor de k.
- mientras termina su merecido cafe verifique si el resultado cambia utilizando paciente elegidos aleatoriamente para el conjunto de validacion.

Una breve excursion en dimensiones mas altas:

Ajustar una funcion (como el modelo lineal) nos permite generalizar nuestra estima de la funcion densidad mas alla de los puntos medidos, pero no nos permite manejar otras generalizaciones bien.

Un primer problema es que tipo de funciones necesitamos para representar, consideren por ejemplo unos puntos distribuidos en una superficie de baja dimensionalidad en un espacio de dimensionalidad mayor, por ejemplo una hoja de cuaderno en 3D. En la direccion perpendicular a la superficie la distribucion es muy angosta, algo que es dificil de expandir con funciones bases trigonometricas o polinomicas.

Este problema es muy comun, en un espacio de features de alta dimensionalidad toda distribucion se vuelve superficial. Para entender este concepto, considere el volumen de una hiper esfera de radio r en un espacio de dimension d:

$$V(r) = \frac{\pi^{d/2} r^d}{(d/2)!}$$

y los problemas de las distribuciones alli.

Ahora miremos a la fraccion de volumen que ocupa una cascara de ancho ϵ, comparada al volumen total de la esfera, en el limite cuando

$$d \to \infty$$

:

$$\frac{V(r) - V(r-\epsilon)}{V(r)} = \frac{r^d - (r-\epsilon)^d}{r^d} = 1 - (1 - \frac{r}{d}) = 1$$

Cuando d crece, todo el volumen esta en una cascara fina sobre la superficie.

Ahora considere lo que implica esto para una una coleccion de puntos que provienen de una distribucion: puntos tipicos provienen de una distribucion no de los bordes, pero en un espacio de alta dimensionalidad las distribuciones son esencialmente bordes y los analisis estaran dominados por los efectos de borde.

Otro punto importante es puede que no podamos muestrear bien la distribucion alli.

Using mixture models for clustering

Dados los problemas de estimacion mencionados, una posible solucion es proponer por ejemplo poner alguna distribucion (Gaussiana por ejemplo) en cada punto medido, esto se conoce como **Kernel density estimation**.

Una mejor aproximacion seria encontrar lugares interesantes donde poner un numero chico (respecto al numero de datos) donde poner algunas funciones locales que modelen bien su vecindad. Esto se conoce como **mixture models**.

Esto esta conectado con el problema de partir un conjunto de datos aglomerando o **clustering**, que es un ejemplo de **aprendizaje no supervisado**. A diferencia de los metodos de fiteo con una funcion distribucion determinada, el algoritmo debe aprender por si mismo donde estan esos lugares interesantes en el conjunto de datos.

Importantes aplicaciones de estos algoritmos se encuentran en la compresion de datos, la deteccion de outliers y crear clasificadores generativos, donde se modela cada densidad de clase $p(x|y = c)$ por una mixtura.

Mixturas de Gausianas en D dimensiones:

un modelo de mixturas puede escribirse factorizando la densidad sobre multiples Gaussianas:

$$p(\bar{x}) = \sum_{m=1}^{M} p(\bar{x}, c_m) = \sum_{m=1}^{M} p(\bar{x}|c_m)p(c_m)$$

$$p(\bar{x}) = \sum_{m=1}^{M} \frac{|C_m^{-1}|^{1/2}}{(2\pi)^{D/2}} \exp[-(\bar{x} - \bar{\mu})^T . C_M^{-1} . (\bar{x} - (\bar{\mu}))/2] p(c_m)$$

donde $|.|^{1/2}$ es la raiz cuadrada del determinante, y c_m se refieren a la m-esima Gaussiana con media μ_m y matriz de covarianza C_m. El desafio es por supuesto encontrar esos parametros!

Mixturas de Gausianas en D dimensiones:

Si tenemos una sola Gaussiana, el valor medio μ puede ser estimado simplemente promediando los los datos:

$$\bar{\mu} = \int_{-\infty}^{\infty} \bar{x} p(\bar{x}) d\bar{x} \simeq \frac{1}{N} \sum_{n=1}^{N} \bar{x}_n$$

dado que una integral sobre una funcion densidad de probabilidad puede aproximarse por una suma de variables extraidas de la distribucion (importante recordar). Todavia no la conocemos a la funcion distribucion, pero por definicion es de donde nuestro conjunto de datos fue extraido.

La idea puede extenderse a mas Gaussianas reconociendo que la *m*esima media es la integral respecto a la distribucion condicional:

$$\bar{\mu} = \int \bar{x} p(\bar{x}|c_m) d\bar{x} = \int \bar{x} \frac{p(c_m|\bar{x})}{p(c_m)} p(\bar{x}) d(\bar{x}) \simeq \frac{1}{N p(c_m)} \sum_{n=1}^{N} \bar{x}_n p(c_m|\bar{x}_n)$$

Mixturas de Gausianas en D dimensiones:

de forma similar se puede encontrar que la matriz de covarianza es:

$$C_m \simeq \frac{1}{Np(c_m)} \sum_{n+1}^{N} (\bar{x}_n - \bar{\mu}_m)(\bar{x}_n - \bar{\mu}_m)^T p(c_m|x_n)$$

y los pesos en la expansion por,

$$p(c_m) = \int_{-\infty}^{\infty} p(\bar{x}, c_m) d\bar{x} = \int_{-\infty}^{\infty} p(c_m|\bar{x}) p(\bar{x}) d\bar{x} \simeq \frac{1}{N} \sum_{i=1}^{N} p(c_m|\bar{x}_n)$$

Pero, como encontramos la probabilidad posterior $p(c_m|\bar{x})$ utilizada en estas sumas? Por definicion es:

El Algoritmo EM

$$p(c_m|\bar{x}) = \frac{p(\bar{x}, c_m)}{p(\bar{x})} = \frac{p(\bar{x}|c_m)p(c_m)}{\sum_{m=1}^{M} p(\bar{x}|c_m)p(c_m)}$$

lo cual puede calcularse segun el modelo inicial propuesto. Esto puede sonar a un razonamiento circular y lo es! Las probabilidades de los puntos pueden ser calculados si conocemos los parametros de las distribuciones (medias, varianzas, pesos) y los parametros pueden ser encontrados si conocemos las probabilidades.

Como comenzamos no conociendo nada, comenzamos suponiendo los parametros aleatoriamente y vamos y venimos iterativamente computando iterativamente las probabilidades y los parametros.

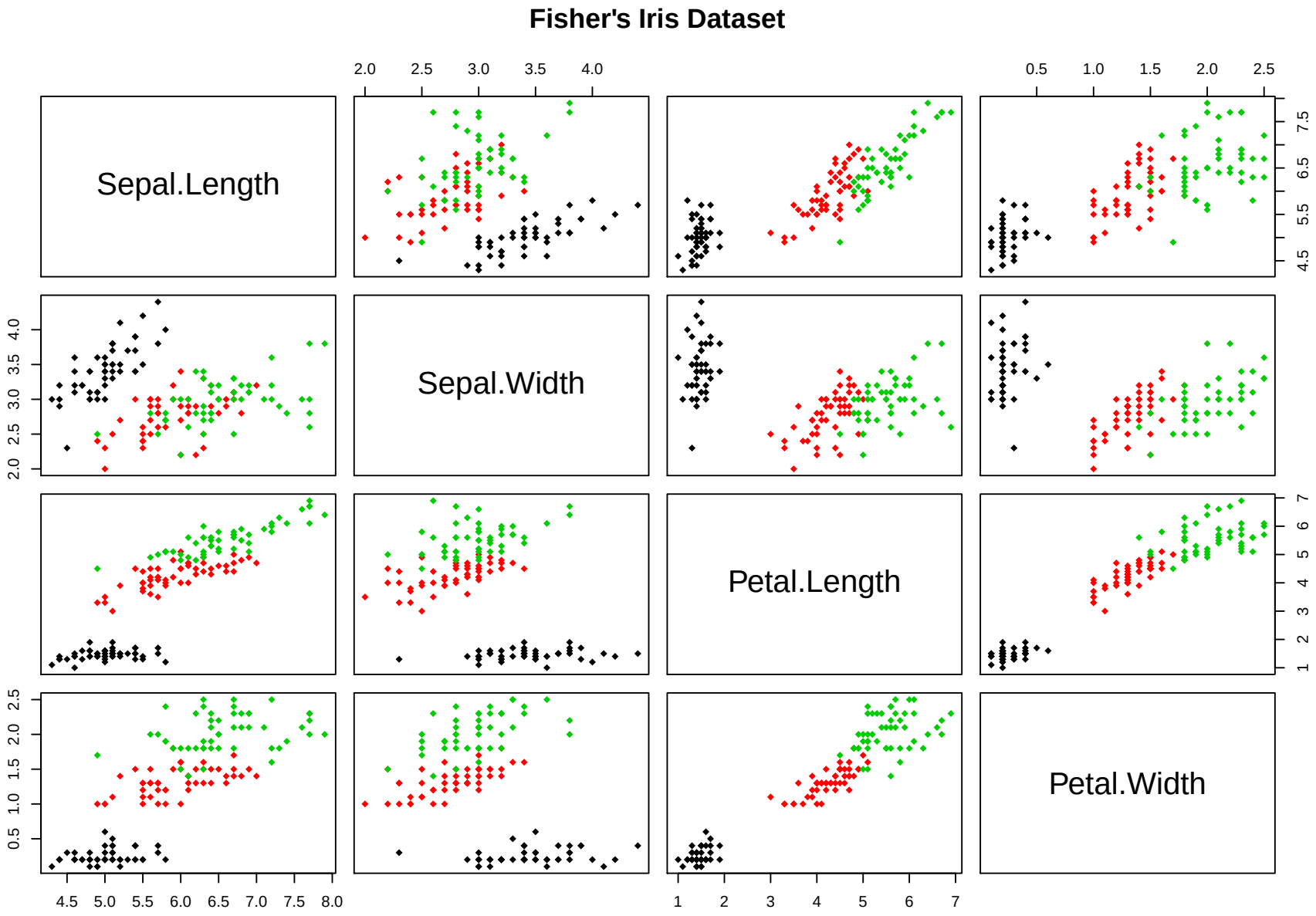
El algoritmo EM y sus parientes:

Calculando una distribucion esperada dados los parametros y luego encontrando los parametros mas probables dada una distribucion, es llamado el algoritmo de maximizacion de expectacion EM y este converge a una distribucion de maximo likelihood comenzando con una semilla aleatoria.

Las Gaussianas pueden por ejemplo ser inicializadas con medias aleatorias y varianzas lo suficientemente grandes como para "sentir" el conjunto de datos. El problema es que las Gausianas captura solo proximidad, el problema surge cuando hay superposicion de las mismas o alguna colapsa.

Una mejor alternativa es basar la expansion de la funcion distribucion alrededor de modelos que pueden capturar mejor la complejidad localmente. Esta poderosa idea ha llevado ha desarrollar diferentes algoritmos como: **redes Bayesianas**, **mixtures of experts** o **cluster wigthed models**.

El algoritmo EM, una aplicacion:

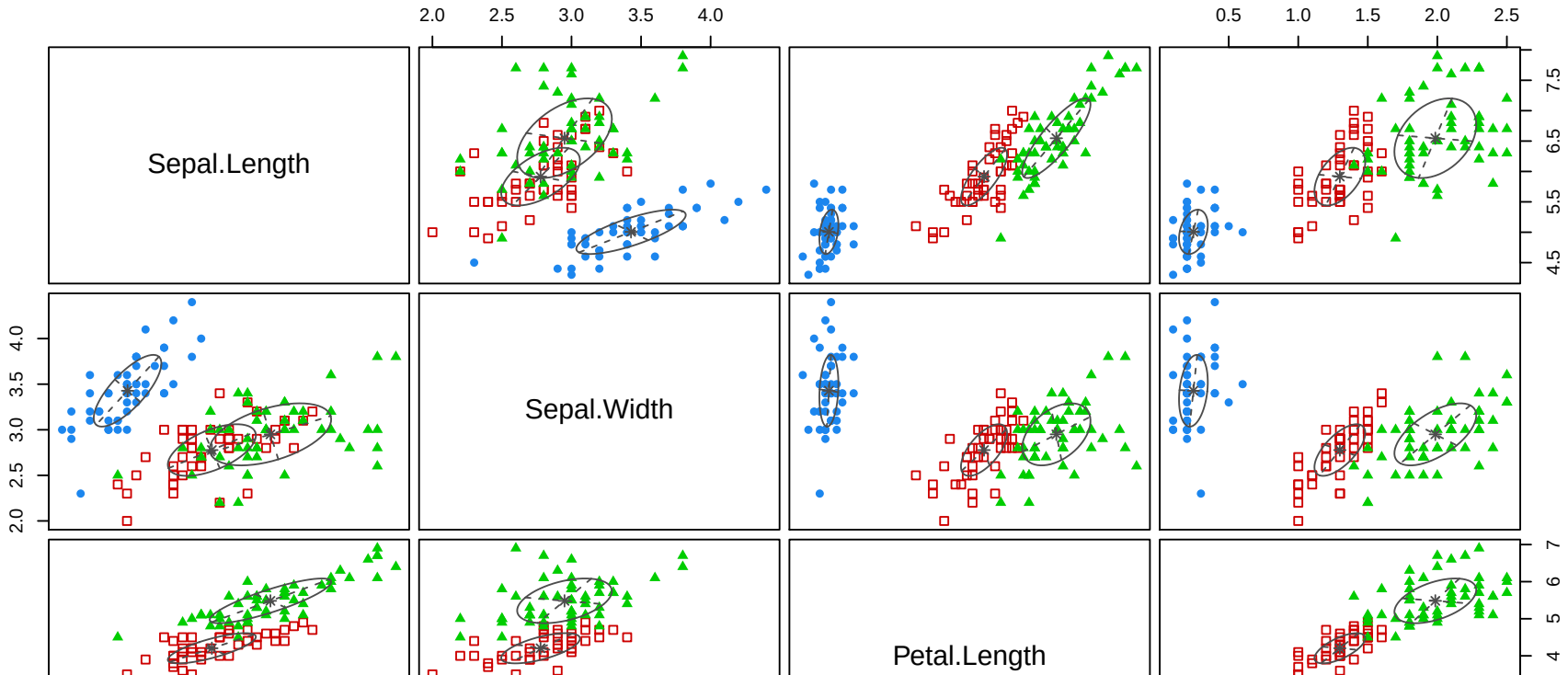


El algoritmo EM, una aplicacion:

```
## Package 'mclust' version 5.4
## Type 'citation("mclust")' for citing this R package in p
```

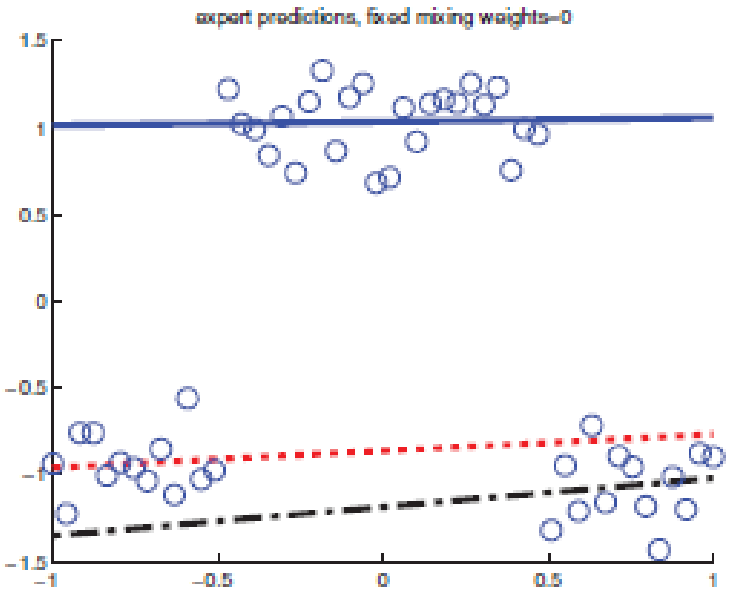
Ahora se utilizan las ecuaciones definidas anteriormente para calcular

```
# Select 4 continuous variables and look for three distinct
mcl.model <- Mclust(iris[, 1:4], 3)
# Plot our results.
plot(mcl.model, what = "classification", main = "Mclust Cla
```

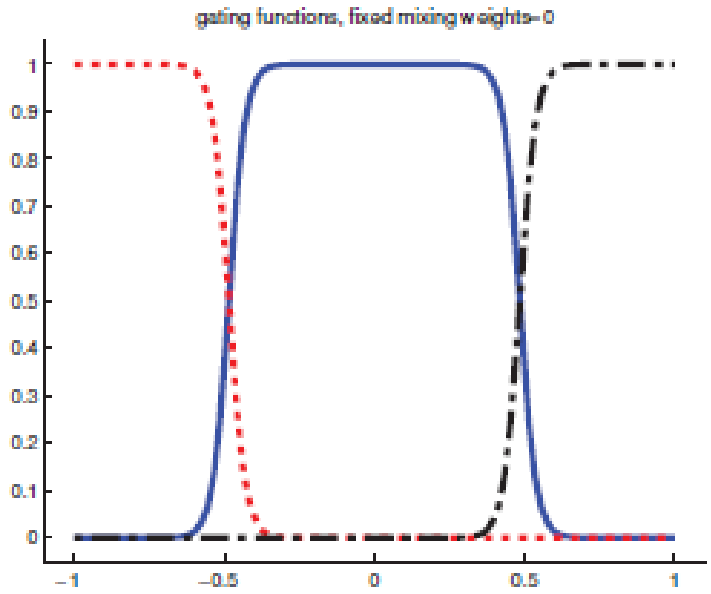


Mixturas de expertos:

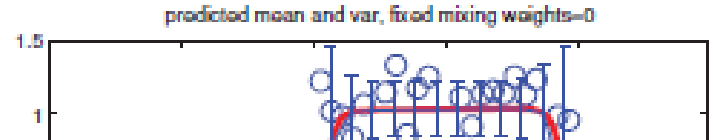
Tambien podemos usar modelos de mixturas para crear modelos discriminativos de clasificacion y regresion. Por ejemplo consideren los siguientes datos, parece una buena idea usar tres modelos diferentes de regresion lineal, cada una aplicada a diferentes partes del espacio de inputs. Esto puede modelarse si permitimos que los pesos de la mixtura y las densidades sean dependientes del input.



(a)



(b)



un poco mas sobre Mixturas:

La idea en una mixtura de expertos es que cada submodelo es considerado un **experto** en cierta region del espacio de los inputs. La funcion: $p(z_i = k|x_i, \theta)$ is llamada funcion gatillo y decide que experto usar en cada rango segun:
$p(y_i|x_i, \theta) = \sum_k p(z_i = k|x_i, \theta)p(y_i|x_i, z_i = k, \theta)$.

Resulta claro que podemos "pegar" cualquier **arquitectura** como un experto. Por ejemplo a veces se utilizan redes neuronales para representar los expertos y las funciones gatillos, el resultado es conocido como **mixture density network**. Son modelos lentos para entrenar pero muy flexibles.

Las mixturas de expertos son muy utiles para resolver **problemas de inversion** donde hay que invertir un mapeo de muchos a uno. Un ejemplo tipico aparece en robotica, donde la localizacion de una mano (robotica) y es unicamente determinada por los diferentes angulos x de las articulaciones de un brazo robot. Sin embargo, para una dada ubicacion y, hay muchas combinaciones de las articulaciones x, por lo que el mapeo inverso $x = f^{-1}(y)$ no es

Clustering hard K-means:

Una variante particular del algoritmo EM para GMMs es conocido como el **Algoritmo K-means**. Consideremos un modelo de mixturas Gaussianas en el cual asumimos que: $\sum_k = \sigma^2 \mathbb{I}_D$ y $\pi_k = 1/K$ estan fijos, por lo que solamente los centros de los cumulos, $\mu_k \in \mathbb{R}^D$, tienen que ser estimados.

Si consideramos la aproximacion de $p(z_i = k|x_i, \theta) \sim \mathbb{I}(k = z_i^*)$,como $z_i^* = max_k p(z_i = k|x_i, \theta)$. obtenemos el llamado **hard K-means**, donde estamos asignando cada punto de datos a un cumulo. Dado que asumimos una matriz de covarianza esferica (en el espacio de features) para cada cumulo, el cumulo mas probable para cada dato for x_i puede ser computado encontrando el cumulo propuesto mas cercano en cada paso.

En cada paso, debemos encontrar las distancias entre los N puntos de datos y los K centros de los cumulos lo que lleva un tiempo de orden NKD.Sin embargo esto puede acelerarse. Dada las asignaciones duras, en cada paso puedo recalcular el centro de los cumulos computando la media de todos los puntos asignados al

ejemplo de K-means:

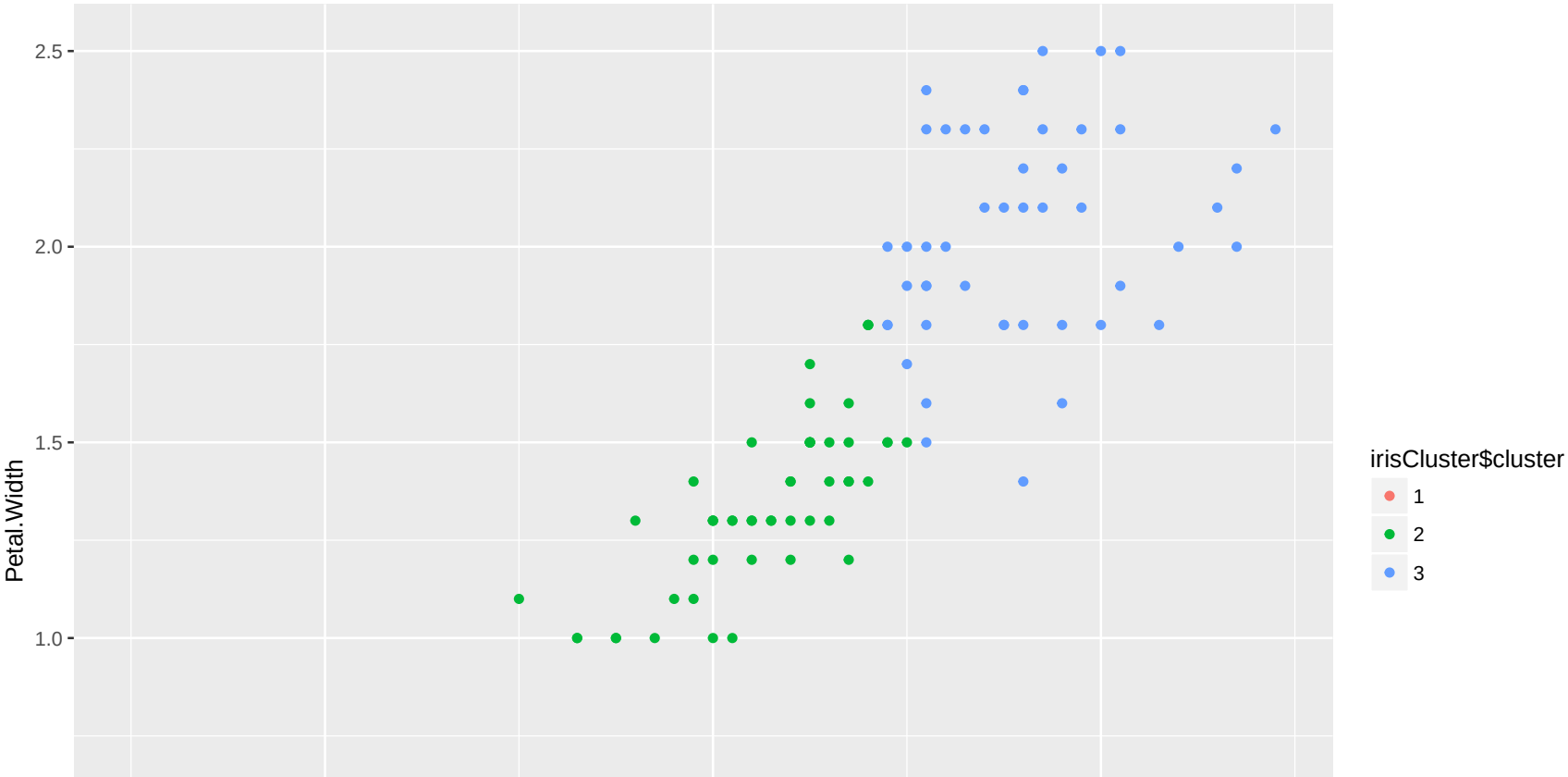
```
set.seed(20)
irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
irisCluster
```

```
## K-means clustering with 3 clusters of sizes 50, 52, 48
##
## Cluster means:
##   Petal.Length Petal.Width
## 1     1.462000    0.246000
## 2     4.269231    1.342308
## 3     5.595833    2.037500
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [106] 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [141] 3 3 3 3 3 3 3 3 3 3
##
```

ejemplo de K-means:

```
##
##      setosa versicolor virginica
##   1      50          0         0
##   2       0         48         4
##   3       0          2        46
```

```
## Warning: package 'ggplot2' was built under R version 3.4
```

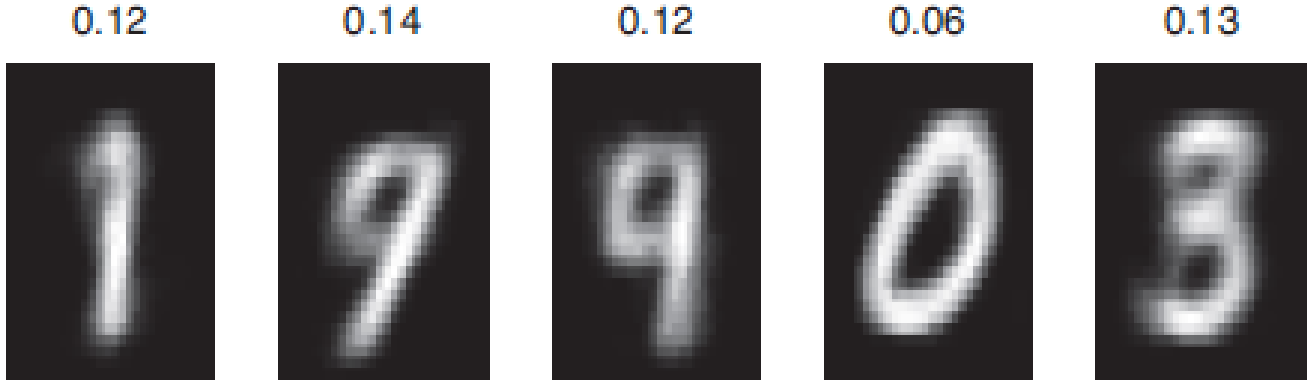


Clustering soft K-means:

Cuando computamos la $p(z_i = k|x_i, \theta)$, la probabilidad posterior que el punto i pertenezca al cumulo k, esta cantidad se puede pensar como la **responsabilidad** del cumulo k por el punto i la cual puede calcularse siguiendo la regla de Bayes :

$$r_{ik} = p(z_i = k|x_i, \theta) = \frac{p(z_i{=}k|\theta)p(x_i|z_i{=}k,\theta)}{\sum_{j=1}^{K} p(z_i{=}j|\theta)p(x_i|z_i{=}j,\theta)}$$

Este procedimiento es llamado **soft clustering**, y permite una mejor clasificacion automatica. Por ejemplo usando una version binarizada del conjunto de carateres digitalizados MNIST, se puede hacer un clustering con $k = 10$ y visualizar los centroides, tal como vemos el metodo descubre correctamente algunas clases de digitos, pero otros no (discutir).



Weapons of math destruction:

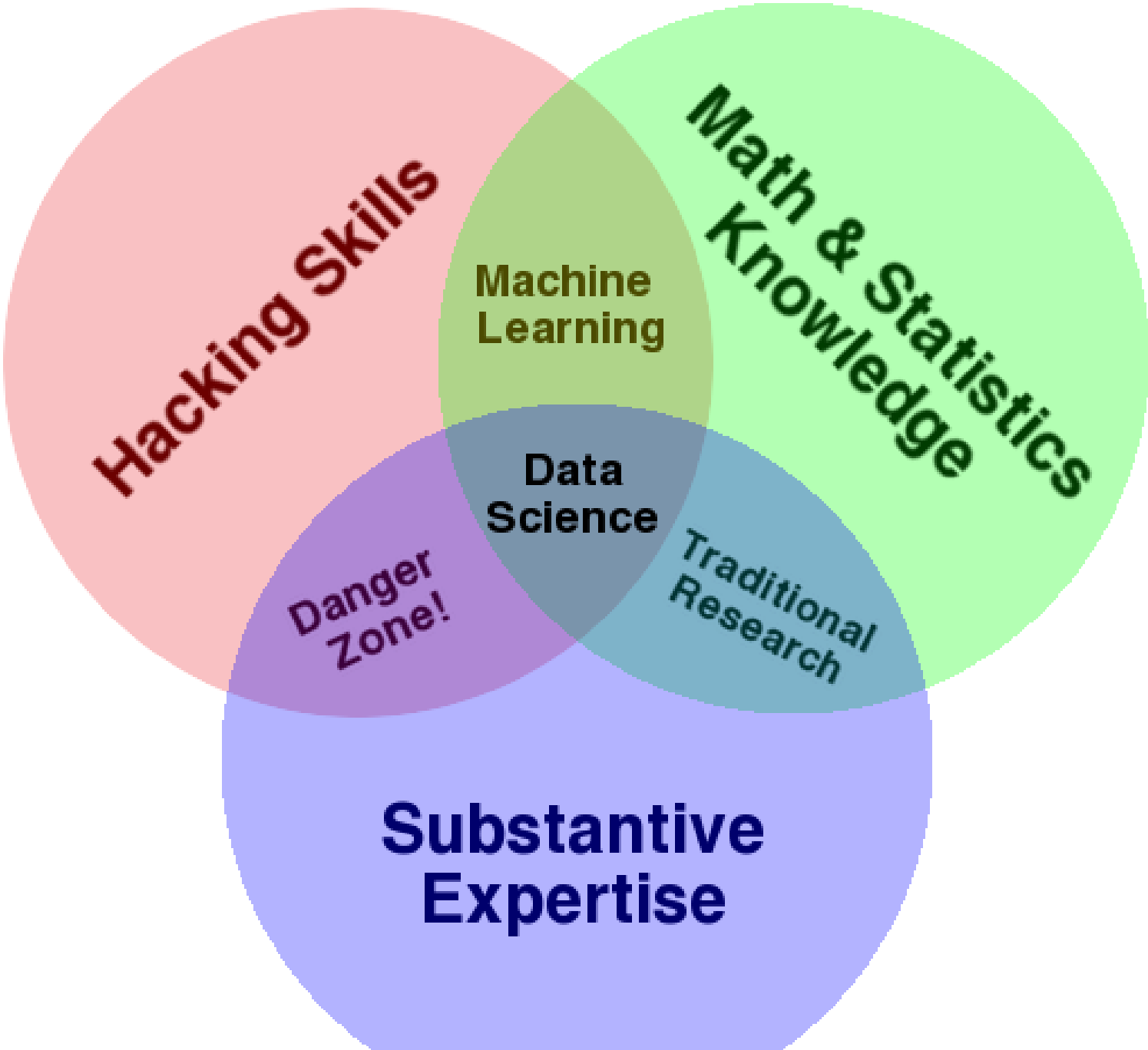
Por mas detalles consultar el siguiente libro http://www.inference.org.uk/itprnn/book.html.

Vale la pena mencionar tambien la posibilidad de llamar de python desde R utilizando https://rstudio.github.io/reticulate/ o trasnferir dataframes a pandas y viceversa con https://github.com/wesm/feather y el uso de ambos lenguajes en sistemas distribuidos (Sparck).

Recuerde que los modelos por su naturaleza son simplificaciones, y cuando lo creamos tomamos decisiones respecto que es importante incluir (muchas veces en el proceso de seleccion de muestras) por lo que podemos generar enormes puntos ciegos. Tenga en cuenta los posibles sesgos en los conjuntos de datos, pq todo algoritmo los perpetuara y amplificara.

El reciente caso de Cambridge Analytics deberia llamar vuestra atencion de los posibles mal usos de las tecnologias que desarrollamos, o los problemas generados por el algoritmic trading, i.e the flash crack etc.

La Ciencia de Datos es multidisciplinaria:



Practico 2: Entregar un Rmd donde se:

- Elija un dataset clasificado de su preferencia y area (domain expertise), aplique un metodo de clustering y/o mixtura de Gaussianas en el mismo.
- Investigue los resultados en el meta parametro K numero de cumulos e investigue posibles procesos de seleccion del mismo.
- Elabore un resumen, y selecione un mejor valor segun el/los criterios aplicados, discuta el significado de los cumulos encontrados.
- Comente la influencia de la normalizacion de los datos en los resultados del clustering.