Apuntes Tema 1

Albert Ribes Marzá

11 de diciembre de 2017

Todo list

Resumen

Los apuntes que vaya tomando en clase

1.

1.1. Introducción a ML

Ejemplo 1 Se pretende medir la temperatura (t) en un punto de una central nuclear, pero la temperatura es tan alta que no se puede medir directamente con ningún sensor. Se intentará deducir la temperatura así:

- t - temperatura a predecir (variable)
- x - vector de variables medibles que posiblemente inciden en t
- z - vector de variables **NO medibles** que posiblemente inciden en t

La relación completa es $t = \delta(x, z)$, que es una función.

Pero no conocemos z ➔Aun conociendo x, el valor de t oscila. La relación entre t y x se hace *estocástica*

$p(x, t)$ será la probabilidad de que con esa x se tenga esa temperatura.

Hay que construir una función $t = y(x)$ donde t sea el valor más plausible. El problema es que no conocemos p

La forma de atacar el problema será recolectar datos $\{(x_1, t_1), (x_2, t_2), \ldots, (x_n, t_n)\} = D$

$(x_n, t_n) \sim^{i.i.p.} p$ (variables independientes e identicamente distribuidas)

El objetivo del *Machine Learning* (ML) es obtener y a partir de D. En este caso es un problema de *regresión*

Ejemplo 2 Se tiene una planta de reciclaje, y se quieren clasificar los objetos que van pasando por la cinta. Los datos son:

- t - tipo de producto
- x - los atributos de los productos que captamos con una cámara
- z - los atributos que no captamos con la cámara

Es el mismo problema de antes, pero ahora t es discreta. Se trata entonces de un problema de *clasificación*.

1.2. Ejemplo introductorio: el ajuste polinómico

Tenemos $x \in \mathbb{R}$ y queremos predecir $t \in \mathbb{R}$ (*Regresión*)

En todos los problemas nos encontraremos con:

- $x_n \in (0, 1)$, $x_n \sim U(0, 1)$, (un conjunto de observaciones)
- $t_n = sin(2\pi x_n) + \varepsilon, \varepsilon \in N(0, \sigma^2)$, normalmente $\sigma^2 = 0{,}3^2$, y donde ε es el ruido aparentemente aleatorio, que proviene de los datos que no conocemos o de errores en la medición.

Vamos a intentar ajustar los datos. Sabemos que si los datos son continuos (no dan saltos) podemos ajustarlos con un polinomio en un intervalo:

- $P_n := \{c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n\} = \{\sum_{i=0}^{n} c_i x^i | c_i \in \mathbb{R}\}$
- $C \in \mathbb{R}^{n+1}$ son todos los parámetros.
- Llamamos $y(x; c) = \sum_{i=0}^{M} c_i x^i$ un modelo
- Respecto a X, y es una función no lineal
- Respecto a C, y es una función lineal

Diremos que un modelo es lineal cuando lo es respecto a los parámetros.

Ajustamos y a los datos $\{(x_1, t_1), (x_2, t_2), \ldots (x_n, t_n)\}$ definiendo una función de error (la función de error cuadrático):

$$E(c) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n; C) - t_n)^2$$

Como E depende automáticamente de C, derivamos e igualamos a 0 para encontrar el mínimo. Hay que hacer n derivadas, una para cada c_j:

$$\frac{\partial E}{\partial c_j} = \frac{1}{2}\sum_{n=1}^{N} 2(y(x_n;C) - t_n)(x_n)^j$$

$$\frac{\partial E}{\partial c_j} = \sum_{n=0}^{N}\Big(\sum_{i=0}^{M}(c_i x_n^i - t_n)\Big)x_n^j = 0$$

El problema de todo esto es que no sabemos qué grado de polinomio deberíamos usar para reflejar el comportamiento de la variable.

- Si es demasiado pequeño no seremos capaces de ajustar la parte regular (y) de los datos (infra-ajuste)
- Si es demasiado grande se ajustará la parte regular (y) y también el ruido (sobre-ajuste)

Cómo elegir el grado del polinomio?

Únicamente conociendo $E(C)$ no se puede saber. Para hacerlo se usa una muestra alternativa de datos de validación. Esta muestra debería tener más datos.

Si observamos el error producido en nuestros datos con diferentes grados de polinomios, obviamente será más pequeño cuanto más grande sea el polinomio, puesto que tiene más flexibilidad. Pero si miramos qué ocurre con los datos de validación, veremos que al principio el error desciende, pero llega un punto en que empieza a subir. El punto mínimo se corresponde con el grado correcto.

El error empezará a subir porque el sobre-ajuste se ha adaptado a los datos aleatorios, pero en la muestra de validación no tienen por qué ser los mismos, y produce más error.

Si la muestra de validación tiene pocos datos, el mínimo estará poco definido, será más redondeado y más difícil de localizar.

Alternativa

Pero no siempre es posible tener suficientes datos de validación. Para esto hay una alternativa.

Uno se pregunta: ¿Si un polinomio de grado 9 "contiene" a los de grado más pequeño, no podría ocurrir que eligiéramos uno de grado mayor, y que él mismo anulara los coeficientes sobrantes hasta que sea del grado adecuado?

La respuesta es que espontaneamente esto no pasa, puesto que para igualar los datos aleatorios son necesarios coeficientes muy grandes. Si queremos que ocurra tenemos que forzarlo de alguna manera. Para hacerlo, redefiniremos nuestra función de error, de manera que también penalice los coeficientes demasiado grandes. Penalizaremos la norma 2, que equivale a la "distancia" pitagórica.

¿Pero cuanto tenemos que penalizarlo? Si nos pasamos o nos quedamos cortos no servirá de nada. Como no sabemos cuanto tenemos que penalizar, usaremos un parámetro λ que regulará la penalización que hacemos.

La función de error queda así:

$$E(C) = \frac{1}{2}\sum_{n=1}^{N}\Big(y(x_n;C) - t_n\Big)^2 + \frac{\lambda}{2}||c||^2$$

El $\frac{\lambda}{2}$ es simplemente para que al derivar quede más simple. Podría ser solo λ

1.3. Conceptos de inferencia estadística

$D = \{x_1, \dots x_n\}$ es una realización de una variable aleatoria (v.a.) X_n que tiene una función de distribución conocida $p(x_n;\theta), \theta \in \Theta$

Pero esa función de distribución tiene unos parámetros, y nos gustaría saber cuáles usar.

Por ejemplo, si se trata de una distribución normal, sabemos que nuestros datos se corresponden con $N(x, \mu, \sigma^2)$, pero viendo los datos no sabemos qué valores de μ y σ^2 deberíamos cojer para que se adaptaran lo más posible a los datos que tenemos.

Vamos a cojer los datos que maximicen nuestra verosimilitud (likelihood), esto es, los que hagan que sea más probable recoger los datos D

El objetivo es obtener una estimación $\hat{\theta}$ de θ, dado D

La probabilidad de recoger una muestra x_i es $p(x_i, \theta)$

Puesto que sabemos que los datos son i.i.d, la probabilidad de cojer todos los datos D es el producto de cada uno de ellos.

La probabilidad de obtener D es:

$$P_n(D, \theta) = \prod_{n=1}^{N} p(x_n, \theta)$$

Definimos la función de verosimilitud (likelihood) así:

$$\mathcal{L} : \theta \to \mathbb{R}$$

$$\theta \to \mathcal{L}(\theta) = P_n(D;\theta)$$

Es decir, es una función que indica cómo de probable es haber recogido los datos D usando los parámetros θ.

Elegiremos los paráetros θ que maximicen esa probabilidad, los que maximicen $P(D, \theta) = \prod_{n=1}^{N} p(x_n, \theta)$

El estimador de máxima verosimilitud es $\hat{\theta} = \text{argmax}\ \mathcal{L}(\theta)$, $\theta \in \Theta$

Si es de una sola variable, la forma de hacerlo es derivar e igualar a 0.

Es conveniente operar con el logaritmo de α, pues simplifica la maximización de un producto:

$$\ln(p_1 p_2 \dots p_n) = \ln(p_1) + \ln(p_2) + \dots + \ln(p_n)$$

Ejemplo

Tenemos un conjunto de datos $D = \{x_1, x_2, \ldots, x_n\}$ que siguen una distribución normal $X_n \sim N(x_n, \mu, \sigma^2)$

Se sabe que la función de densidad de la distrubión normal es:

$$N(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Entonces la probabilidad de haber cogido los datos D es:

$$P(D, \mu, \sigma^2) = \prod_{i=1}^{n} \Big(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \Big)$$

Pero para hacer los cálculos más sencillos trabajaremos con el logaritmo, que no cambia los máximos relativos. Además, le vamos a cambiar el signo, y ya no buscaremos maximizarlo, sino minimizarlo.

$$l = -\ln(P(D, \mu, \sigma^2)) = -\ln \prod_{i=1}^{n} \Big(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \Big) = -\sum_{i=1}^{N} \ln \big(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \big)$$

Se simplifica así:

$$l = -\sum_{i=1}^{N} \ln \big(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \big)$$

$$l = -\sum_{i=1}^{N} \Big[\ln \big(\frac{1}{\sigma\sqrt{2\pi}} \big) + \ln \big(e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \big) \Big]$$

$$l = -\sum_{i=1}^{N} \Big[-\ln \big(\sigma\sqrt{2\pi} \big) - \frac{(x_i-\mu)^2}{2\sigma^2} \Big]$$

$$l = \sum_{i=1}^{N} \Big[\ln \big(\sigma\sqrt{2\pi} \big) + \frac{(x_i-\mu)^2}{2\sigma^2} \Big]$$

Ahora derivamos respecto de μ y de σ^2 e igualamos a 0 para encontrar los extremos:

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{N} \Big[0 + \frac{1}{2\sigma^2} \cdot 2(x_i - \mu)(-1) \Big]$$

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{N} \Big[-\frac{x_i - \mu}{\sigma^2} \Big]$$

$$\frac{\partial l}{\partial \mu} = -\frac{1}{\sigma^2} \sum_{i=1}^{N} \Big[x_i - \mu \Big] = 0$$

$$\sum_{i=1}^{N} x_i - \sum_{i=1}^{N} \mu = 0$$

$$\sum_{i=1}^{N} x_i - N\mu = 0$$

$$\sum_{i=1}^{N} x_i = N\mu$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Y respecto de σ^2:

$$l = \sum_{i=1}^{N} \Big[\ln\big(\sigma\sqrt{2\pi}\big) + \frac{(x_i-\mu)^2}{2\sigma^2} \Big]$$

$$\frac{\partial l}{\partial \sigma^2} = \sum_{i=1}^{N} \Big[\frac{1}{\sigma\sqrt{2\pi}} \cdot \frac{\sqrt{2\pi}}{2\sigma} + \frac{(x_i-\mu)^2}{2}\big(-\frac{1}{\sigma^4}\big) \Big]$$

$$\frac{\partial l}{\partial \sigma^2} = \sum_{i=1}^{N} \Big[\frac{1}{2\sigma^2} - \frac{(x_i-\mu)^2}{2\sigma^4} \Big] = 0$$

$$\sum_{i=1}^{N} \Big[\frac{1}{2\sigma^2} \Big] - \sum_{i=1}^{N} \Big[\frac{(x_i-\mu)^2}{2\sigma^4} \Big] = 0$$

$$\frac{N}{2\sigma^2} = \frac{1}{2\sigma^4}\sum_{i=1}^{N}(x_i-\mu)^2$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2$$

De este modo hemos encontrado las ecuaciones clásicas para encontrar la media y la varianza muestral, que son la media y varianza más probables teniendo en cuenta los datos que hemos recogido.

Se podría aplicar el mismo procedimiento con alguna otra distribución.

Nota: habría que asegurarse que realmente se trata de mínimos mirando la segunda derivada y comprobando que da valores positivos. Se deja como ejercicio

1.4. Propiedades de un estimador

2. Reducción de dimensión

Para hacer *Machine Learning* es interesante tener los datos lo más simplificados posible, pues eso evita el sobre-ajuste. Existen muchos métodos para reducir la dimensión de un problema. Reducir la dimensión se podría entender como quedarse con una sombra de la imagen real que tenemos. Esto es: si todos los datos que tenemos estuvieran en 3 dimensiones, podría interesarnos trabajar

con la sombra que proyectan esos datos, de manera que trabajaríamos con solo 2 dimensiones.

Pues hacemos los mismo, pero con muchas dimensiones.

Las ventajas que tiene esto son que evita el sobre-ajuste, nos permite entender mejor los datos y que son más fáciles de representar, con plots o dibujos.

Pero hay que cojer una buena proyección de los datos reales. Puesto que está claro que vamos perder información (datos, en realidad), cojeremos una proyección que refleje lo que nos interesa, y que deseche otras cosas.

Es por eso que hay muchas formas de reducir la dimensión de un conjunto de datos, cada una según la prioridad que uno tenga, y cogiendo las proyecciones más adecuadas para cada necesidad.

Ahora veremos algunas de las formas de reducir la dimensión:

2.1. Principal Components Analisis (PCA)

Este algoritmo tiene como prioridad preservar la varianza de los datos, maximizar la dispersión en las proyecciones.

Esto es, en la analogía de la sombra, cojer la sombra que tenga más area.

De forma más técnica:

Tenemos una muestra de datos $\{X_1, X_2, \ldots, X_n\}, X_i \in \mathbb{R}^d$ que provienen de un vector aleatorio $X = \{X_1, \ldots, X_n\}^T$. Cada una de las X_i es una variable (aleatoria?) y tenemos d muestras en cada una de las variables.

Disponemos también de la matriz de covarianzas Σ.

La matriz de covarianzas es una matriz de $n \times n$ donde Σ_{ij} es $\text{var}(X_i, X_j)$ si $i \neq j$ y Σ_{ii} es σ_i^2

Tenemos datos en n dimensiones, y decidimos que queremos únicamente k dimensiones, $k < n$, y no cualesquiera dimensiones, sino las que maximicen la varianza.

Hemos de encontrar entonces k vectores n-dimensionales. Encontraremos n vectores que serán todos "perpendiculares" entre ellos y cojeremos los k vectores que tengan más varianza.

Nuestro objetivo entonces obtener un nuevo sistema de coordenadas $Y = (Y_1, \ldots, Y_n)$ que cumpla estas condiciones:

1. $\text{Covar}(Y_i, Y_j) = 0$ si $i \neq j$
2. $\text{Var}(Y_1) > \text{Var}(Y_2) > \ldots > \text{Var}(Y_n)$ (de hecho los ordenaremos decrecientemente)
3. $\sum_{i=1}^{d} Var(X_i) = \sum_{i=1}^{d} Var(Y_i)$

Encontraremos la proyección Y_i encontrando un vector w_i que cumpla que $Y_i = w_i^T \cdot X$

Como hay muchos vectores que cumplen esa condición (vectores que tienen todos la misma dirección, pero distinto módulo) establecemos la condición sobre w_i de que la norma 2 al cuadrado sea 1, esto es: $||w_i||_2^2 = 1 \Rightarrow w_{i1}^2 + w_{i2}^2 + \cdots + w_{in}^2 = 1$

Objetivo: w_1 ha de maximizar la varianza de Y_i, sujeto a que $||w_i|| = 1$

$$Var(Y_i) = Var(w_i^T \cdot X) = w_1^T \cdot Var(X) \cdot w_i = w_1^T \cdot \Sigma \cdot w_i$$

Este último paso es algo que se sabe y que sale en Wikipedia. Nos lo creemos.

Para resolver un problema de maximización sujeto a algunas condiciones se hace con el método de los multiplicadores de Lagrange.

Anexo: método de Lagrange

El método de Lagrange sirve para encontrar los extremos (máximos o mínimos) que hay en una función sujeto a algunas condiciones de igualdad. La función puede tener una cantidad arbitraria de parámetros, y se acepta también una cantidad arbitraria de condiciones.

Si tenemos la función $f(x_1, \ldots, x_n)$ y las condiciones $g_1(x_1, \ldots, x_n) = 0, \ldots, g_m(x_1, \ldots, x_n) = 0$, definimos nuevas variables λ (habrá una por cada condición que haya) y construimas la función de Lagrange:

$$\mathcal{L}(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m) = f(x_1, \ldots, x_n) - \sum_{k=1}^{m} \lambda_k g_k((x_1, \ldots, x_n))$$

En el caso particular de una función de dos parametros $f(x, y)$ y una restricción $g(x, y) = 0$ sería así:

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y)$$

El siguiente paso es derivar $\mathcal{L}$ sobre cada uno de los parámetros (tanto los reales como los añadidos) e igualar todas las ecuaciones a 0. Eso dará como resultado un sistema de ecuaciones que tendrá tantos resultados como extremos existan cumpliendo todas las condiciones.

Ahora ya únicamente queda mirar todos esos puntos y decidir cuales son máximos o mínimos.

Fin del Anexo: método de Lagrange

Entonces, nuestro problema es maximizar

$$w_i^T \cdot \Sigma \cdot w_i$$

sujeto a que

$$\sum_{j=1}^{d} \left(w_{ik}^2\right) - 1 = 0$$

Construimos la función de Lagrange:

$$\mathcal{L}(w_i, \lambda) = w_i^T \cdot \Sigma \cdot w_i - \lambda\Big(\sum_{j=1}^{d} \left(w_{ik}^2\right) - 1\Big)$$

Y derivamos e igualamos a 0:

(No tengo muy claro cómo se hace esa derivada, pero nos creemos que es así)

$$\frac{\partial \mathcal{L}}{\partial w_i} = 2\Sigma \cdot w_i - 2\lambda w_i = 0$$

$$\Sigma \cdot w_i = \lambda w_i$$

La expresión que nos ha quedado se corresponde con la definición de vector y valor propio de una matriz (*eigenvector* y *eigenvalue*). Se dice que λ es un valor propio de la matriz Σ si existe un vector w tal que:

$$\Sigma \cdot w = \lambda w$$

Al vector w se le llama vector propio de Σ

Vemos entonces que los vectores de proyección w_i que estamos buscando se corresponden con los vectores propios de la matriz Σ.

Pero hemos exigido que los vectores de proyección esten ordenados decrecientemente por varianza, i. e. el primer vector de proyección w_1 debe ser el que tenga más varianza en la proyección, el segundo w_2, etc.

Veamos cuál será la varianza del vector w_i

$$Var(Y_i) = Var(w_i^T \cdot \Sigma) = w_i^T \cdot \Sigma \cdot w_i$$

Pero hemos visto que

$$\Sigma \cdot w_i = \lambda w_i$$

Por lo tanto podemos sustituir:

$$w_i^T \cdot \Sigma \cdot w_i = w_i^T \lambda w_i = \lambda w_i^T w_i$$

Y como hemos exigido que $w_i^T w_i = 1$, tenemos que la varianza será λ

Entonces, el orden en que hemos de cojer los vectores propios es respecto a su valor propio: primero el vector con mayor valor propio, etc.

Pero en realidad también hemos exigido que los vectores de proyección elegidos también sean perpendiculares entre ellos. Para conseguir esto, haremos Lagrange para encontrar únicamente el primero de los vectores de proyección, y luego, para encontrar el segundo hacemos igual pero estableciendo también la condición de que el nuevo vector de proyección sea perpendicular con el primero, i, e: $w_1^T w_2 = 0$

Cuando tenemos todos los vectores de proyección $w_1, w_2, \ldots, w_n$, podemos definir la matriz

$$A = \begin{bmatrix} w_{11} & w_{21} & \ldots & w_{d1} \\ w_{12} & w_{22} & \ldots & w_{d2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1n} & w_{2n} & \ldots & w_{dn} \end{bmatrix}$$

Y entonces podemos decir que nuestros nuevos datos son:

$$Y = A^T X$$

Ahora es cuando hemos de decidir con cuántos de los componentes principales nos quedamos. Si nos los quedamos todos lo único que habremos hecho será una transformación de los datos.

Si queremos quedarnos con m componentes principales y en total había d ($m \leq d$), entonces

$$\frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \times 100$$

Es el porcentaje de la varianza con el que nos estamos quedando

2.2. Fisher Discriminant Analysis (FDA)

Reducción de dimensión con FDA

Queda pendiente para hacer esta sección.

Tiene que ver con hacer otro modelo para reducir la dimensión, dando prioridad a otras cosas

3. Clustering

Clustering quiere decir agrupar. Un algoritmo de clustering recibe como entrada un conjunto de datos, y él clasifica esos datos en varios grupos, en varios clusters, en función de algún parámetro.

Normalmente la cantidad de clusters es arbitraria, decidida por factores ajenos al problema. De hecho es un problema complicado decidir cuál es la cantidad adecuada de clusters, y existen algunos algoritmos que intentan calcularla, pero son poco genéricos.

Para los algoritmos que trataremos aquí supondremos que la cantidad de clusters ya viene dada. Se tratará de encontrar "dónde están" esos clusters y ver qué elementos pertenecen a cada uno de los clusters. Como veremos, la mayoría de problemas problemas de optimización de clustering son NP-completos, y por eso buscaremos una buena solución en vez de la mejor solución.

3.1. Algoritmo de K-means

Disponemos de un conjunto de datos $D = \{x_1, x_2, \ldots, x_n\}, x_i \in \mathbb{R}^d$ y queremos encontrar K prototipos (centroides) $P = \{\mu_1, \mu_2, \ldots, \mu_K\}, \mu_i \in \mathbb{R}^d$ a los que asociar los datos D

El criterio será minimizar la suma de distancias de cada dato x_i a su cluster más cercano, que será el cluster al que esté asignado.

Definimos una variable binaria π_{ik}:

$$\pi_{ik} = \begin{cases} 1 & \text{si el dato } x_i \text{ está asignado al cluster } k \\ 0 & \text{otherwise} \end{cases}$$

Formalmente el criterio a minimizar será:

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} \pi_{ik} ||x_i - \mu_k||^2$$

Está demostrado que dado un conjunto de datos, encontrar la "posición" de K clústers y la asignación de datos a clusters de la manera más optima es un problema NP-completo, de modo que para resolver esto vamos a tener un problema.

Sin embargo, resolver la mitad del problema es fácil: si te dicen a priori la "posición" de cada uno de los clusters, es fácil ver qué datos hay que asignar a cada cluster. Si por el contrario te dicen qué datos irán asignados al mismo cluster, es fácil encontrar la posición de cada uno de los clusters.

Entonces para resolver este problema, partiremos de una de las mitades del problema, que se habrá encontrado arbitrariamente, y entonces se buscará

la otra mitad más óptima. A continuación se recalculará la primera mitad en función de estos datos, y este proceso se repetirá hasta que ya no se mejore más.

De esta manera terminamos teniendo un máximo local, pero no absoluto. La calidad de la solución dependerá de los datos iniciales. Se han hecho muchos estudios sobre qué datos iniciales son más adecuados, pero al final parece que lo mejor es usar datos aleatorios de entre el conjunto de datos D.

Este algoritmo es extremadamente rápido, por lo que normalmente se ejecuta varias veces con datos iniciales distintos y se mantiene el que da mejores resultados.

Formalizar el algoritmo de k-means y demostrar que es correcto

Ahora faltaría ver cómo se encuentra la segunda mitad. Queda para más adelante.

3.2. Mezcla de Gaussianas (Mixture of Gaussians MoG)

El método de K-means tiene el problema de que es binario: un dato pertenece a un cluster o no pertenece, y solamente pertenece a uno. Dos datos que estén muy "cerca" pueden quedar en dos clusters distintos a pesar de que sean muy parecidos.

Otra forma de verlo es que un dato pertenezca a un cluster con una cierta probabilidad. Para ello habría que definir una función de distribución que reflejara la asignación de datos a los distintos clústers. El problema es que una distribución simple (como la normal o la binomial) no tiene la capacidad para reflejar cualquier cantidad de datos y clusters. Por ello vamos a usar la Mezcla de Gaussianas. Esto no es más que asignar una distribución gaussiana (o normal) a cada uno de los clusters, y hacer la suma. De este modo con una cantidad suficiente de distribuciones normales podemos representar cualquier función de probabilidad.

Formalizamos el problema.

Tenemos un conjunto de datos $D = \{x_1, x_2, \ldots, x_n\}, x_i \in \mathbb{R}^d$ y queremos definir K clusters. Diremos que π_k es la probabilidad que el cluster k genere un dato cualquiera. Deberá cumplirse que $\sum_{k=1}^{K} \pi_k = 1$

Entonces la probabilidad de que el dato x sea generado es:

$$p(x) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x; \mu_k, \Sigma_k)$$

Si queremos generar datos siguiendo una MoG, lo primero que hay que hacer es elegir qué componente generará el dato, según la probabilidad que tengan (π_k) y después generar un dato siguiendo la distribución normal que esta define, como ya se ha hecho siempre.

Para simplificar la notación a partir de ahora, supondremos la existencia de un vector z que indicará componentes principales. Se trata de un vector binario de K elementos en el que todos los elementos son 0 excepto uno, que tiene un 1 y que indica de qué componente se trata.

Entonces diremos que:

$$p(z_k = 1) = \pi_k$$

$$p(z) = \prod_{k=1}^{K} \pi_k^{z_k}$$

De hecho, se puede entender como $p(z) = \pi_1 \cdot \pi_2 \cdot \cdots \cdot \pi_K$
De la misma manera podemos decir que:

$$p(x|z_k = 1) = \mathcal{N}(x; \mu_k, \Sigma_k)$$

$p(x|z) = \sum_{k=1}^{K} z_k \cdot \mathcal{N}(x; \mu_k, \Sigma_k)$(La misma que antes hemos definido como p(x))

¿Cómo encontrar los parámetros adecuados de una MoG mediente la máxima verosimilitud.

Explicar la relación que hay entre k-means y E-M (el de MoG)

Ahora vendrían más cosas, pero sigo con otro tema

4. Modelos lineales de regresión

Lo que se vió en el tema 1 sobre el ajuste polinómico no es más que un caso concreto de los modelos lineales de regresión, en el que las funciones de base eran obligatoriamente exponenciales.

Ahora vamos a ver el caso genérico.

4.1. Introducción al concepto de función de base (Basis Function, BF)

Si tenemos datos $x \in \mathbb{R}^d$, y queremos predecir un dato t, igual que con el ajuste polinómico, definimos una colección de funciones $\phi_j : \mathbb{R}^d \to \mathbb{R}$ para definir el siguiente modelo:

$$y(x, w) = \phi_0(x) + \sum_{j=1}^{n-1} w_j \phi_j(x),$$

El caso del ajuste polinómico es cuando se establece la condición de que $\phi_j(x) = x^j$. Ahora podemos usar un conjunto más grande de funciones de base, incluídos los senos y cosenos.

4.2. El método de mínimos cuadrados

Como siempre, consideraremos:

$$t = f(x) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2)$$

Es decir, que los datos de salida dependen de los datos que podemos medir (x) y de datos que no podemos pedir (ϵ), que consideraremos ruido, y trabajaremos con la asunción de que ese ruido sigue una distribución normal centrada en 0 y con varianza σ^2.

Entonces definiremos:

$$p(t|x, w, \sigma^2) = \mathcal{N}(t; y(x, w), \sigma^2)$$

Esto es, la probabilidad de que la salida sea t asumiendo los datos x, w, σ^2.

Abordamos el problema de decidir qué parámetros $w \in \mathbb{R}^M$ y $\sigma^2 \in \mathbb{R}$ deberíamos elegir para modelar los más acertadamente posible un conjunto de datos $D = \{x_1, x_2, \ldots, x_n\}, x_i \in \mathbb{R}^d$ (que son i.i.d) que da como resultado $t \in \mathbb{R}^N$

Igual que en ajuste polinómico, todavía no sabemos qué cantidad de funciones de base deberíamos usar. De momento, definimos que usaremos M funciones de base.

Se usará para los siguientes cálculos la notación matricial, que deja implícitos muchos cálculos. Para que se entienda bien, se definen los siguientes datos:

$$X = \begin{bmatrix} \leftarrow & x_1 & \rightarrow \\ \leftarrow & x_2 & \rightarrow \\ \vdots & & \\ \leftarrow & x_n & \rightarrow \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ x_{21} & x_{22} & \ldots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} \end{bmatrix}$$

$$t = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}$$

$$\phi = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_M \end{bmatrix}$$

$$\Phi = \begin{bmatrix} \phi_0(x_1) & \phi_1(x_1) & \ldots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \ldots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \ldots & \phi_{M-1}(x_n) \end{bmatrix}$$

Sacamos entonces la función de verosimilitud que ofrecen los parámetros $\theta = (w, \sigma^2)$ usando el logaritmo negado de la probabilidad de obtener los datos t mediante los datos X. Esta función es la que habrá que maximizar.

$$
\begin{aligned}
-l(\theta) &= \\
-l(w,\sigma^2) &= \\
-\ln P(t|X,w,\sigma^2) &= \\
-\ln \prod_{i}^{N} p(t_i|X_i,w,\sigma^2) &= \\
-\ln \prod_{i=1}^{N} \mathcal{N}(t_i; y(x_i,w),\sigma^2) &= \\
-\ln \prod_{i=1}^{N} \mathcal{N}(t_i; w^T\phi x_i,\sigma^2) &= \quad \textit{entiéndase bien la notación} \\
-\sum_{i=1}^{N} \Big(\ln \mathcal{N}(t_i; w^T\phi x_i,\sigma^2) \Big) &= \\
-\sum_{i=1}^{N} \Big(\ln \big(\frac{1}{\sqrt{2\pi\sigma^2}} exp\Big[\frac{(t_i - w^T\phi x_i)^2}{2\sigma^2} \Big] \big) \Big) &= \\
-\sum_{i=1}^{N} \Big(-\ln \sqrt{2\pi\sigma^2} - \frac{(t_i - w^T\phi x_i)^2}{2\sigma^2} \Big) &= \\
N \ln \sqrt{2\pi\sigma^2} + \frac{1}{2\sigma^2} \sum_{i=1}^{N} (t_i - w^T\phi x_i)^2 &
\end{aligned}
$$

Definimos:

$$E(w) = \frac{1}{2} \sum_{i=1}^{N} (t_i - w^T\phi x_i)^2$$

Y por lo tanto:

$$-l(\theta) = N \ln(\sqrt{2\pi\sigma^2}) + \frac{1}{\sigma^2} E(w)$$

Se puede observar que:

$$w^T\phi x_i \equiv \Phi w$$

Y entonces se podría cambiar la definición por:

$$E(w) = \frac{1}{2} \sum_{i=1}^{N} (t_i - w^T\phi x_i)^2 = \frac{1}{2} ||t - \Phi w||^2$$

$$E(w) = \frac{1}{2} ||t||^2 + \frac{1}{2} ||\Phi w||^2 - t^T \Phi w$$

Ahora hay que derivar $-l(\theta)$ e igualar a 0 para encontrar los extremos:

$$\frac{\partial(-l)}{\partial w} = \frac{\partial E(w)}{\partial w} = 0 \equiv$$
$$0 + 2 \cdot \frac{1}{2}(\Phi w)^T \Phi - t^T \Phi = 0 \equiv$$
$$\Phi^T \Phi w = \Phi^T t$$

Esa matrix, $\Phi^T \Phi$, se llama *matrix de diseño*.

Y ahora habría que encontrar w en función de las matrices y de t. Hay unas cuantas formas de abordar el problema, y la opción más obvia no da muy buenos resultados.

4.2.1. Con la función inversa

La primera forma de atacar el problema podría ser intentar resolver:

$$w = (\Phi^T \Phi)^{-1} \Phi^T t$$

Y esto asumiendo que existe la función inversa de la *matriz de diseño*. Existe un teorema que demuestra que en este caso particular sí que tiene inversa, y se llama *inversa de Moore-Penrose*

Poner y explicar el teorema que demuestra que la matriz de diseño para resolver las equciones de Gauss tiene inversa

El problema que tiene este método es que el cálculo de la inversa incluye números muy pequeños, muy cerca de 0, y los ordenadores tienen problemas para calcularla, y cometen errores de *underflow*.

4.2.2. SVD: Singular Vaue Decomposition

Para solucionar ese problema, hay un teorema que dice que toda matriz $A_{N \times M}, N > M$, se puede expresar como:

$$A = U \Delta V^T$$

Donde:

- U y V son ortogonales, i.e: $U^T U = I$, $V^T V = I$
-

Terminar de explicar como funciona la SVD (Singular Value Decomposition)

4.3. Regresión por mínimos cuadrados regularizados

Explicar la regularización (con lambda) con el método de mínimos cuadrados

5. Clasificadores lineales

Una clasificación es una función $y : \mathbb{R}^d \Rightarrow \{1 \dots k\}$ que particiona $\mathbb{R}^d$ en k subespacios.

Definimos una región R_k como $R_k = \{x \in \mathbb{R}^d | y(x) = k\}$. Las separaciones entre regiones se llaman *fronteras* o "superficies de decisión". Un modelo lineal de clasificación se caracteriza porque las fronteras que genera son hiperplanos (en $\mathbb{R}^{d-1}$)

Hay muchos tipos de clasificadores, pero los que comunmente son más interesantes son los probabilísticos, i.e, clasificadores que dado un dato x indican la probabilidad de pertenencia a cada una de las regiones.

Puesto que queremos clasificadores lineales, seguiremos usando el método de multiplicar y sumar la entrada con pesos, pero ahora tenemos la condición de que el resultado sea una probabilidad, y para ello el primer paso es que esté acotado en el intervalo $(0, 1)$.

Nuestro problema de momento será encontrar una función g de la forma $x \Rightarrow y(x) = g(w^T x + w_0)$, donde $g : \mathbb{R} \Rightarrow (0, 1)$

A esta función le vamos a exigir que sea continua, estrictamente creciente, derivable en el intervalo y que tenga una inversa, y comunmente a esta función se le llama *función de activación.*

Un posible ejemplo de la función de activación es la *función logística*, que se define como $g(z) = \frac{1}{1-e^{-z}}$. Para esta en particular, $g(z)' = g(z)[1 - g(z)]$, $g(-z) = 1 - g(z)$ y $g(z)^{-1} = \ln(\frac{z}{1-z})$

5.1. Fórmula de Bayes

Explicar de dónde viene la fórmula de Bayes y poner el ejemplo de los Halcones y las Águilas

5.2. Clasificadores (Bayesianos) generativos

Si tenemos dos clases ($K = 2$), la probabilidad de que el dato x pertenezca a la clase C_1 es:

$$\begin{aligned} P(C_1|x) &= \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(P_1) + P(x|C_2)P(C_2)} \\ &= \frac{1}{1 + \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)}} \end{aligned}$$

Si ahora definimos:

$$a(x) = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$$

La probabilidad es:

$$P(C_1|x) = \frac{1}{1 + exp(-a(x))}$$

Esta es una *función logística* que se conoce como "*log of the odds*"

El caso genérico para $K \geq 2$ clases es:

$$\begin{aligned} P(C_k|x) &= \frac{P(x|C_k)P(C_k)}{\sum_{j=1}^{K} P(x|C_j)P(C_j)} \\ &= \frac{exp(a_k(x))}{\sum_{j=1}^{K} P(x|C_j)P(C_j)} \end{aligned}$$

Ahora hemos generalizado la definición con:

$$a_k(x) = \ln P(x|C_k)P(C_k)$$

Esta función se llama *softmax*, y es una generalización de la *función logística*. Este tipo de modelos se llaman "modelos generativos", porque requieren conocer $P(x|C)$, y una vez sabes eso, eres capaz de generar nuevos datos en la clase C

Ejemplo: Generador Gaussiano Si asumimos que los datos de la clase C_k siguen una distribución guassiana, entonces

$$P(x|C_k) = \frac{1}{(2\pi)^{\frac{d}{2}}} \frac{1}{|\Sigma_k|^{\frac{1}{2}}} exp\Big(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\Big)$$

Para el caso $K = 2$ previamente hemos definido

$$a(x) = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$$

Puesto que sabemos que los datos siguen una distribución normal, lo podemos escribir como:

$$\begin{aligned} a(x) &= \ln(P(x|C_1)P(C_1)) - \ln(P(x|C_2)P(C_2)) \\ a(x) &= \cancel{\ln(\varnothing)} - \frac{1}{2}\ln(|\Sigma_1|) - \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + \ln(P(C_1)) \\ &\quad - \cancel{\ln(\varnothing)} + \frac{1}{2}\ln(|\Sigma_2|) + \frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) - \ln(P(C_2)) \end{aligned}$$

Donde $\cancel{\ln(\varnothing)} = \ln \frac{1}{(2\pi)^{\frac{d}{2}}}$, que es constante en los dos términos y se anula.

Con esto ya podemos calcular $P(C_1|x)$. A este clasificador se le llama *Quadratic Discriminant Analysis* (QDA).

Si se asume que $\Sigma_1 = \Sigma_2 = \Sigma$, entonces la ecuación anterior se simplifica, deja de ser cuadrática y es lineal:

$$a(x) = w^T x + w_0$$

Donde $w = \Sigma^{-1}(\mu_1 - \mu_2)$ y $w_0 = \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \ln \frac{p(C_1)}{P(C_2)}$

Esta, como es lineal, se llama *Linear Discriminant Analysis* (LDA)

¿Y qué ocurre cuando $K \geq 2$? Entonces se usa la definición

$$a_k(x) = \ln P(x|C_k)P(C_k)$$

Que sustituyendo queda como:

$$a_k(x) = -\frac{1}{2}\ln|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \ln P(C_k)$$

Y ésta es la fórmula general para QDA. Si se vuelve a asumir que todas las clases tienen la misma varianza, entonces $a_k(x) = w_k^T x + w_0$, donde $w_k = \Sigma^{-1}\mu_k$ y $w_0 = \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \ln P(C_k)$. Ésta es la fórmula general para LDA.

5.2.1. Cálculos en la práctica

Explicar los problemas en QDA y LDA cuando hay más atributos que número de ejemplos o cuando las matrices de varianza no son invertibles.

Explicar también RDA

5.3. Naive Bayes

Cuando el problema de clasificación es multivariado, lo que queremos representar es:

$$P(x|C_k) = P(X_1 = x_1 \wedge X_2 = x_2 \wedge \cdots \wedge X_d = X_d|C_k)$$

En probabilidades, se sabe que

$$P(X_1, X_2, X_3) = P(X_3|X_1, X_2)P(X_2|X_1)P(X_1)$$

Entonces se podría representar el problema como

$$P(x|C_k)P(C_k) = P(X_1 = x_1|C_k)P(C_k)\prod_{j=2}^{d} P(X_j = x_j|X_{j-1} = x_{j-1}\wedge\cdots\wedge X_1 = x_1 \wedge C_k)$$

Pero si hacemos la asunción de que cada una de las variables es independientemente de todas las demás, la probabilidad se simplifica:

$$\begin{aligned} P(x|C_k)P(C_k) &= P(X_1 = x_1|C_k)P(C_k)\prod_{j=2}^{d} P(X_j = x_j|C_k)P(C_K) \\ &= P(C_k)\prod_{j=1}^{d} P(X_j = x_j|C_k) \end{aligned}$$

Como es un producto de elementos en $[0, 1]$ es conveniente trabajar con el logaritmo:

$$\ln NB_k(x) = \ln(P(C_j)) + \sum_{j=1}^{d} \ln P(X_j = x_j|C_k)$$

Este método tiene la ventaja de que permite trabajar con tipos de datos distintos: la variable X_1 podría ser contínua y se podría modelar con una distribución normal mientras que la variable X_2 podría ser discreta, y usar la proporción de elementos como probabilidad.

5.4. Nearest Neighbours (KNN)

Dado un espacio X, definimos una función $d : X + X \to \mathbb{R}^+ \cup \{0\}$, con las siguientes propiedades para $\forall x, y \in X$:

1. $d(x, y) == 0 \Rightarrow x == y$
2. $d(x, x) == 0$
3. $d(x, y) == d(y, x)$
4. $d(x, y) \leq d(x, z) + d(z, y)$ (Desigualdad triangular)

Decimos que d es una métrica. Dado un conjunto de datos $D = \{(x_1, t_1), \ldots, (x_n, t_n)\}$ fijamos un $K \in \mathbb{N}$. Dado un punto x^* a predecir, donde $x^* \notin D$, t^* es la clase mayoritaria entre los K datos más cercanos a x^* según la métrica que se ha definido. En caso de empate, KNN se resuelve al azar.

Hay que cambiar la numeración, pues Naive Bayes y KNN debería estar dentro de Clasificadores Generativos, pero tienen su tema ellos solos

5.5. Clasificadores discriminativos

El coste de hacer LDA es cuadrático en la dimensión de los datos (d). Vamos a intentar mejorarlo haciendo como en regresión:

$$P(C_1|x) = g(w^T x)$$

Pero ahora nuestros datos siguen una distribución Bernuilli. Nuestro modelo (para $K = 2$) será:

$$P(t|x) = \begin{cases} g(w^T x) & \text{si } t = 1 \\ 1 - g(w^T x) & \text{si } t = 0 \end{cases}$$

Los parámetros serán w (θ)

$$\begin{aligned} -l(\theta) &= -\ln \mathcal{L}(\theta) \\ &= -\ln P(T|D) \\ &= -\ln \prod_{n=1}^{N} P(t_n|x_n) \\ &= -\ln \prod_{n=1}^{N} g(w^T x_n)^{t_n} [1 - g(w^T x_n)]^{1-t_i} \end{aligned}$$

Para simplificar la notación, diremos que $y_n = g(w^T x)$, y entonces la función de error queda:

$$E(w) = -\sum_{n=1}^{N} \Big[t_i \ln y_n + (1 - t_i) \ln(1 - y_n) \Big]$$

Esta función se llama función de entropía cruzada (*cross-entropy*). Ahora habría que derivarla e igualarla a 0 para encontrar los valores de w más verosímiles.

Hacer la derivada de la función cross-entropy

Explicar la interpretación de la regresión logística

6. Redes neuronales

Hasta ahora nuestros modelos eran de la forma $y(x; w) = g(w^T \phi(x))$. Son lineales porque w hace una operación lineal. Para salir del mundo lineal, pondremos operaciones no lineales en las funciones de base. Serán funciones de base parametrizadas:

$$\phi(x, v) \quad x, v \in \mathbb{R}^d$$

En redes neuronales, las funciones serán:

$$\phi_i(x) = \phi(\varphi(x, v_i)) \quad x, v \in \mathbb{R}^d$$

Donde v_i son parámetros de la neurona i.
El modelo resultante es:

$$\begin{aligned} y(x, w, \{v\}) &= g(w^T \phi(x)) \\ &= g(\sum_{1}^{n-1} w_i \phi_i(x) + \phi_0) \\ &= g(\sum_{1}^{n-1} w_i \phi(\varphi(x, v_i)) + \phi_0) \\ &= g(\sum_{1}^{n-1} w_i \phi(\sum_{j=1}^{d} v_{ij} x_j + v_{i0}) + \phi_0) \end{aligned}$$

La función g dependerá del tipo de problema que tengamos. Si se trata de un problema de regresión será la identidad, mientras que si se trata de un problema clasificación será la función logística.

La función ϕ, que cada neurona tiene asociada, será la función de activación. Hay muchas válidas. La idea es que si el valor de entrada no supera un cierto humbral, la función da salida nula (0), y si lo supera ya da otros valores. Puede ser la logística, o incluso una función no derivable.

La topología que se define es la siguiente: cada neurona recibe como entrada un vector, cada uno de los elementos de este vector se multiplica por un peso y se suma todo. Después se pasa por la función de activación y devuelve un resultado. Esto lo hacen todas las neuronas en paralelo, todas ellas dan un resultado y lo pasan a la neurona de salida. Esta última ya hace lo mismo que se hacía en modelos lineales, y saca el resultado.

Entonces, se puede ver que las redes neuronales no son más que una generalización de los modelos lineales: una red neuronal será un modelo lineal cuando no tenga ninguna neurona en la capa intermedia.

Esta capa intermedia se llama capa oculta. También está la capa de salida, y en algunos libros se refieren a la entrada como la neurona de entrada, aunque realmente no hace nada.

Los mismo que se ha hecho para pasar de un modelo lineal a uno no lineal se puede seguir haciendo. Lo único que hay que hacer es añadir más capas de neuronas a la capa oculta. La única condición es que las neuronas que están en una misma capa no pueden estar conectadas entre ellas.

Igual que en los modelos lineales, es bueno que cada neurona tenga un w_0 a modo de bias. Para simplificar la notación, cada neurona tendrá siempre una entrada a 1, y w_0 no será más que el peso asociado a esa entrada. La neurona de salida también tendrá esa entrada ficticia.

6.1. El algoritmo de backpropagation

Tenemos un MLP (Perceptrón Multi-capa) con c capas ocultas y una capa de salida. Definimos h_l como el númeoro de neuronas de la capa l, y entonces $h_0 = d$ (la cantidad de datos de entrada) (para simplificar la notación aquí se ha considerado a la entada como una capa más, la capa de entrada) y $h_c = m$ (la cantidad de datos de salida)

Nuestro modelo será entonces:

$$y : \mathbb{R}^d \longmapsto \mathbb{R}^m$$

La notación que usaremos será:

- w^l_{ji} es el peso que conecta la neurona i de la capa $l-1$ con la neurona j de la capa l
- W_l es la matriz de todos los pesos que unen la capa $l-1$ con la capa l, i.e. : $(w^l_{ji})_{ij}$
- $\mathcal{W}$ es el vector que contiene todas las matrices W_l, o una matriz tridimensional.

La salida k será:

$$y_k(x) = \sum_{j=0}^{h_c} w^{c+1}_{kj} \phi^c_j(x) \qquad\qquad k = 1, \ldots, m$$

Donde

$$\phi^l_j = \Phi\Big(\sum_{i=1}^{h_{l-1}} w^l_{ji} \phi^{l-1}_i(x) \Big)$$

Y

$$\phi^0_i(x) = x_i$$
$$\phi^l_0(x) = 1$$

Para el caso de regresión, definiremos la función de error como:

$$E(\mathcal{W}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{m}(t_{nk} - y_k(x_n))^2$$