Algoritmos de Clasificación

Curso de aprendizaje automático para el INE

Víctor Gallego y Roi Naveiro

2019-04-11

Intro a git

- Git es un sistema de **control de versiones** utilizado para gestionar archivos de código.

En la terminal de bash (Linux, MacOS)

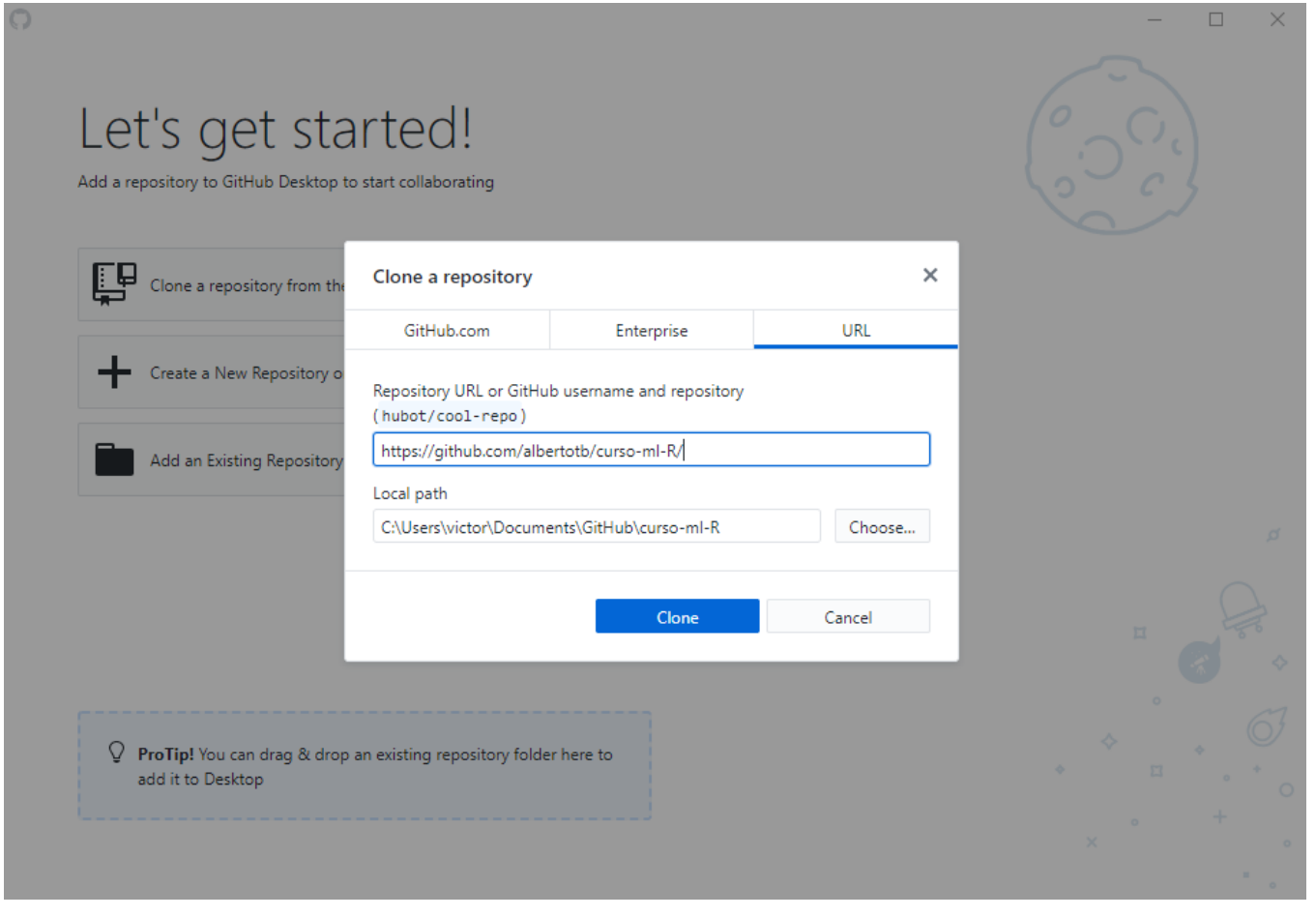
- Obtener e instalar el programa **git**: https://git-scm.com/.
- Descarga de un repositorio: **git clone https://github.com/albertotb/curso-ml-R**.

  Esto nos creará un nuevo directorio **curso-ml-R**, descargando todo lo que hubiera en la copia remota (la alojada en Github en este caso).
- Actualización de cambios: **git pull** (ejecutado dentro del directorio del repositorio).

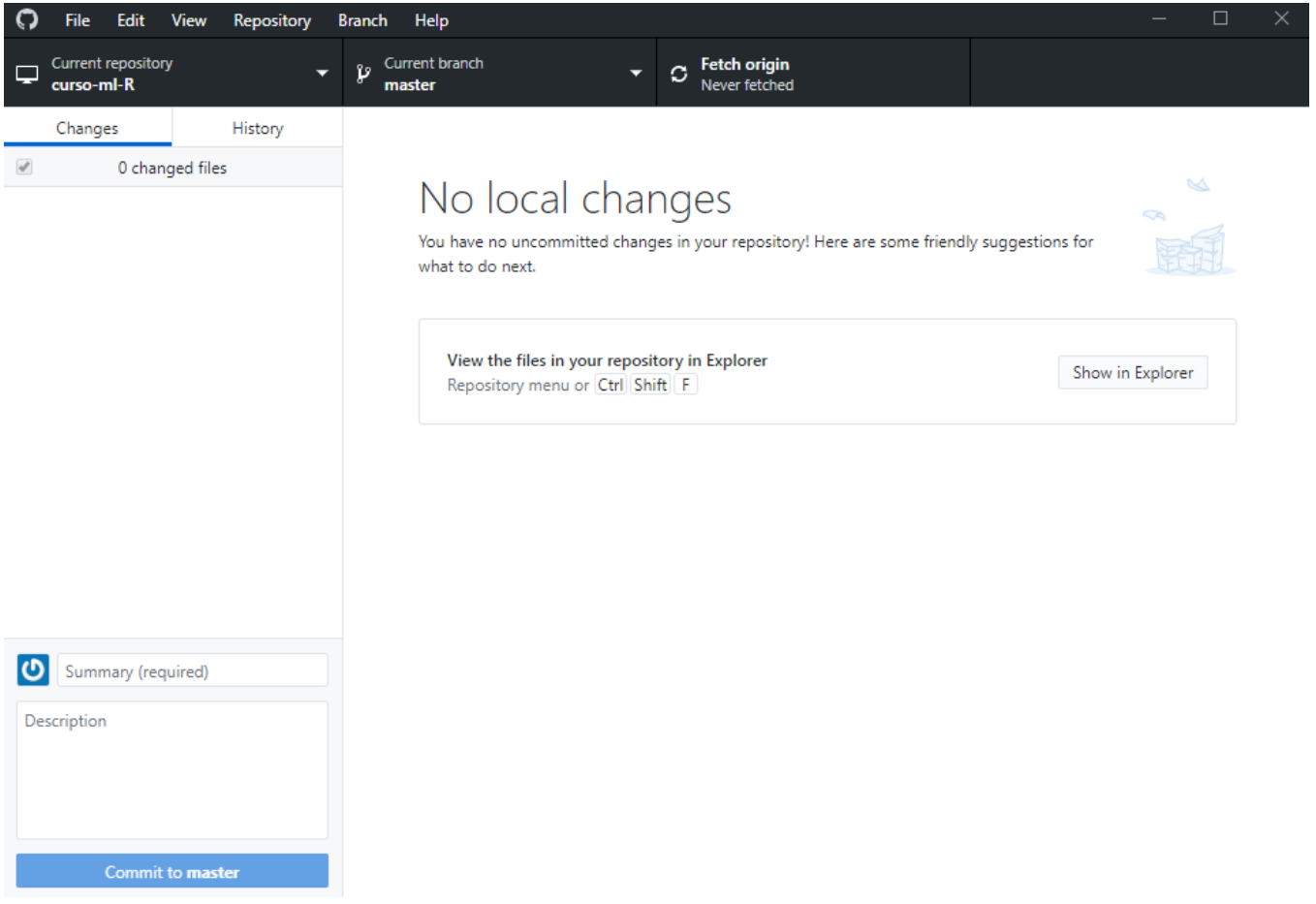
  En caso de que la copia remota tenga cambios respecto a nuestra copia local, actualiza nuestra copia local del repositorio. Esto evita volver a descargar todo como al usar git clone.

En Windows

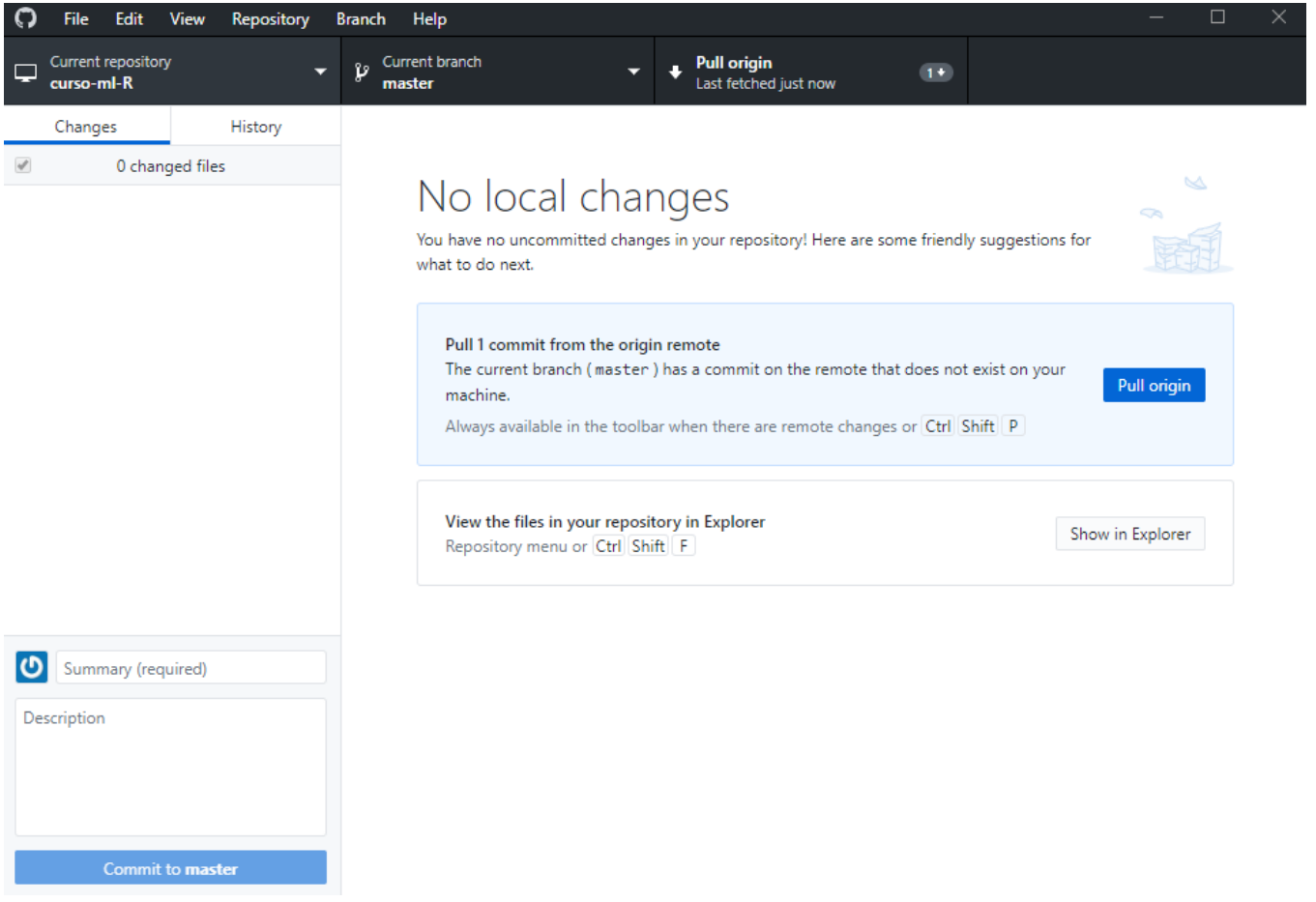
- Podemos utilizar la interfaz gráfica oficial desde https://desktop.github.com. Tras instalarlo, en el menú principal escogemos **Clone a new repository**:



- Ésta es la pantalla por defecto del repositorio. Podemos abrir los archivos en el explorador, o ver los cambios recientes en el lateral.



- Para **sincronizar** nuestra copia local con el remoto, pulsamos en **Fetch** arriba, y en caso de haber cambios, pulsamos en **Pull origin** para confirmar.



Regresión Lineal en problemas de clasificación

¿Cómo aplicar regresión lineal a problema de clasificación multiclase?

- Consideramos K clases.
- **One Hot Encoding** de las categorías: para categoría k, crear vector K dimensional t_k con tan solo un 1 en posición k (resto ceros).
- $y_i = t_k$ si la categoría del ejemplo i-ésimo es k.
- Problema de predicción: reproducir el target de cada observación. Resolver

$$\min_{\mathbf{B}} \sum_{i=1}^{N} \|y_i - [(1, x_i^\top)\mathbf{B}]^\top\|^2$$

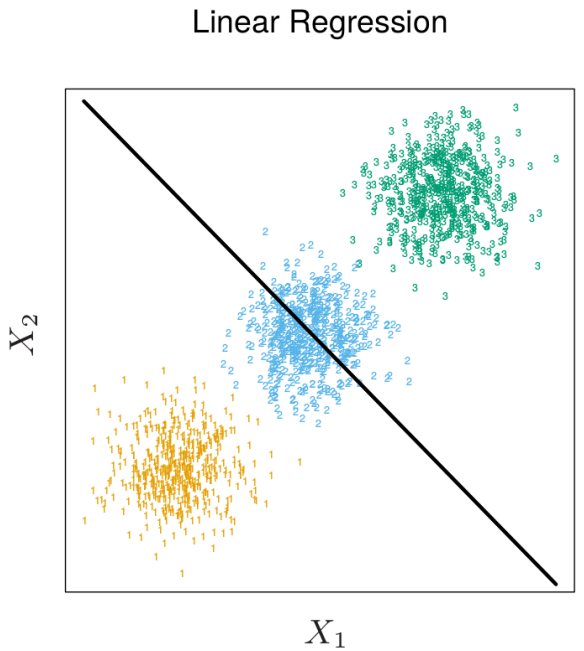
- Para clasificar nueva observación se calcula el vector $\hat{f}\,(x)$ y se clasifica resolviendo

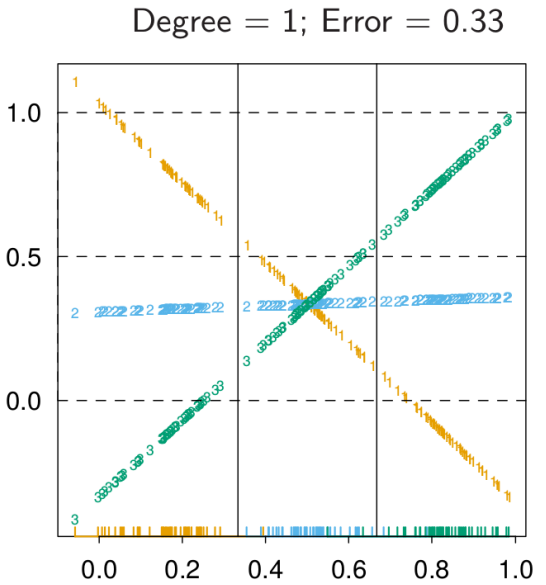
$$\arg\min_k \|\hat{f}\,(x) - t_k\|^2$$

- Es fácil ver que el problema desacopla en K problemas de regresión (uno para cada clase).

Problema - *masking*

- Cuando $K \geq 3$ unas clases pueden enmascarar otras.





Fuente: Elements of Statistical Learning

Introducción

Teoría de la decisión estadística

- Sea $X \in \mathbb{R}^p$, vector de variables predictoras.
- Sea $Y \in \{Y_1, Y_2, \ldots, Y_K\}$, respuesta categórica.
- Distribución de los datos $X, Y \sim p(X, Y)$.
- Dado nuevo X necesitamos estimar $\widehat{Y}(X) \in \{\{Y_1, Y_2, \ldots, Y_K\}$.
- Definimos función de coste $L[Y, \widehat{Y}(X)]$. Una bastante común: coste $0/1$ (0 si acertamos, 1 si nos equivocamos).
- Objetivo: Escoger $\widehat{Y}(X)$ que minimice coste esperado

$$\begin{aligned}\mathbb{E}_{X,Y}\left[L(Y, \widehat{Y}(X))\right] &= \mathbb{E}_X \mathbb{E}_{Y|X}\left[L(Y, \widehat{Y}(X))\right] \\ &= \mathbb{E}_X \sum_{i=1}^{K} L[Y_i, \widehat{Y}(X)] P(Y = Y_k|X)\end{aligned}$$

Teoría de la decisión estadística

- Es suficiente minimizar el coste esperado para cada x

$$\widehat{Y}(x) = \arg\min_y \sum_{i=1}^{K} L[Y_i, y] P(Y = Y_k | X = x)$$

- Con el coste $0/1$

$$\widehat{Y}(x) = \arg\min_y [1 - P(y|X = x)]$$

- Asignamos la clase con más probabilidad a posteriori.

Teoría de la decisión estadística

- Hemos separado el problema de clasificación en dos partes

    1. **Inferencia**: usar datos de entrenamiento para encontrar $P(Y = Y_k|X = x)$.
    2. **Decisión**: usar las distribuciones a posteriori para tomar decisión óptima de clasificación (minimizar coste esperado, maximizar utilidad esperada...)

- Posibilidad alternativa: aprender directamente funciones que mapeen X en Y.

Tres maneras de enfrentar los problemas de clasificación

1. *Modelos generativos*: tratan de modelizar $P(Y, X)$ (Naive-Bayes).
    - Permiten muestrear.
    - Detección de outliers (si $P(X)$ es pequeño).
    - Más difícil (si X es de dimensión alta...).
2. *Modelos discriminativos*: tratan de modelizar $P(Y|X)$ (Regresión logística).
    - Si solo interesa clasificar: más fácil computacionalmente.
3. *Funciones discriminantes*: Aprenden funciones que mapean X en Y directamente (Perceptrón).
    - No tenemos acceso a las probabilidades a posteriori cada vez que queramos tomar nuevas decisiones.
    - Probabilidades a posteriori **muy útiles**: cambaimos frecuentemente función de coste, queremos tener opción de rechazo, combinar modelos, etc.

Análisis Discriminante Lineal

LDA

- Sea $f_k(x)$ la densidad de probabilidad de x condicionada a la clase Y_k.
- Sea π_k el prior de la clase Y_k. Se tiene

$$P(Y = Y_k|X = x) = \frac{f_k(x)\pi_k}{\sum_{i=1}^{K} f_i(x)\pi_i}$$

- ¿Es este un modelo generativo, discriminativo o función discriminante?

LDA

- Asumamos modelo Gaussiano para $f_k(x)$

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma_k}|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_k)^\top \boldsymbol{\Sigma_k}^{-1}(x-\mu_k)\right]$$

- LDA: Asume que las clases tienen matriz de covarianza común $\boldsymbol{\Sigma_k} = \boldsymbol{\Sigma}\ \forall k$.

- Comparamos dos clases

$$\begin{aligned}\log \frac{P(Y=Y_k|x)}{P(Y=Y_j|x)} &= \log \frac{f_k(x)}{f_j(x)} + \log \frac{\pi_k(x)}{\pi_j(x)} \\ &= \log \frac{\pi_k(x)}{\pi_j(x)} - \frac{1}{2}(\mu_k+\mu_j)^\top \boldsymbol{\Sigma}^{-1}(\mu_k+\mu_j) + x^\top \Sigma^{-1}(\mu_k-\mu_l)\end{aligned}$$

- Frontera de decisión lineal!!

- El hecho de que $\boldsymbol{\Sigma}$ no dependa de la clase causa la linealidad.

LDA

- Vemos que asignar a $X = x$ la clase con más probabilidad a posteriori es equivalente a asignar la clase con *función discriminante lineal* $\delta_k(x)$ más grande

$$\delta_k(x) = x^\top \mathbf{\Sigma}^{-1} \mu_k - \frac{1}{2} \mu_k^\top \mathbf{\Sigma}^{-1} \mu_k + \log \pi_k$$

- Para estimar parámetros desconocidos, usamos datos de entrenamiento (MLE)

1. $\hat{\pi}_k = N_k / N$
2. $\hat{\mu}_k = \sum_{x_i | y_i = Y_k} x_i / N_k$
3. $\mathbf{\Sigma} = \sum_{k=1}^K \sum_{x_i | y_i = Y_k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^\top / (N - K)$

QDA

- Si no asumimos que las matrices de covarianza son independientes de las clases, llegamos al **Análisis Discriminante Cuadrático**.

$$\delta_k(x) = -\frac{1}{2}\log|\mathbf{\Sigma}_k| - \frac{1}{2}(x-\mu_k)^\top \mathbf{\Sigma}_k^{-1}(x-\mu_k) + \log \pi_k$$

- La frontera de decisión ahora es cuadrática.

- Para estimar parámetros desconocidos, usamos datos de entrenamiento (MLE)

1. $\hat{\pi}_k = N_k/N$
2. $\hat{\mu}_k = \sum_{x_i|y_i=Y_k} x_i/N_k$
3. $\mathbf{\Sigma}_k = \sum_{x_i|y_i=Y_k}(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^\top/(N-K)$

Análisis Discriminante Regularizado

- Compromiso entre LDA y QDA, regularizando la matriz de covarianza.

$$\hat{\boldsymbol{\Sigma}}_k(\alpha) = \alpha\hat{\boldsymbol{\Sigma}}_k + (1-\alpha)\hat{\boldsymbol{\Sigma}}$$

- $\alpha \in [0, 1]$ permite un contínuo de modelos entre LDA y QDA.
- α suele escogerse usando validación cruzada, validación hold-out,...
- Otra posibilidad

$$\hat{\boldsymbol{\Sigma}}_k(\gamma) = \gamma\hat{\boldsymbol{\Sigma}} + (1-\gamma)\sigma^2\boldsymbol{I}$$

Computación para LDA

- La computación se simplifica diagonalizando la matriz $\hat{\mathbf{\Sigma}}$.
- Sea $\hat{\mathbf{\Sigma}} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^\top$ la descomposición en autovalores de la matriz de covarianza.
- Para clasificar podemos:

  1. *Esferizar* los datos usando $X^* = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{U}^\top X$. Ahora la matriz de covarianza es la identidad.
  2. Clasificar una nueva instancia a la clase del centroide más cercano en el espacio transformado, modulo el efecto de los priors π_k.
  3. Esto es así pues podemos escribir la función discriminante como

$$\delta_k'(x) = -\frac{1}{2}(x - \mu_k)^\top \mathbf{\Sigma}^{-1}(x - \mu_k) + \log \pi_k$$

Regresión Logística y Optimización Estocástica

Regresión logística (repaso)

- Clasificación binaria:

$$p(y = 1|x) = \sigma(w^\intercal x + b)$$

- Clasificación en $M > 2$ clases: cambiar la función sigmoide σ por la **softmax** $s : \mathbb{R}^M \to \mathbb{R}^M$, definida como

$$s(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{M} e^{z_j}}$$

donde $z = Wx + B$, con $W \in \mathbb{R}^{M \times D}$, $B \in \mathbb{R}^{M \times 1}$.

- Aprendizaje mediante mínimos cuadrados no lineales, descenso por el gradiente: los algoritmos vistos requieren acceso a la matriz X entera en cada iteración.

- ¿Qué hacer cuando X no cabe en memoria?

Descenso por el gradiente (GD)

- Habitualmente se considera el problema de minimizar una función con la siguiente forma:

$$f(w) = \frac{1}{N} \sum_{i=1}^{N} f_i(w)$$

(por ejemplo, al minimizar el error/pérdida promedio sobre la muestra de entrenamiento)

- Optimizamos iterando:

$$w^{t+1} = w^t - \eta_t \nabla_w f(w^t)$$

- Problema: complejidad $\mathcal{O}(N)$

Descenso por el gradiente estocástico (SGD)

- En cada iteración, barajeamos los datos y escogemos **uno** al azar

$$w^{t+1} = w^t - \eta_t \nabla_w f_i(w^t)$$

- También podemos seleccionar un **minilote** $\mathcal{B} = \{i_1, \dots, i_B\} \subset \{1, \dots N\}$ al azar en cada iteración:

$$w^{t+1} = w^t - \eta_t \nabla_w \frac{1}{B} \sum_{i \in \mathcal{B}} f_i(w^t)$$

- La complejidad pasa de $\mathcal{O}(N)$ a $\mathcal{O}(B)$, además, no es necesario tener toda la matriz X, sino solo los datos del minilote $\mathcal{B}$.

- Otra ventaja: mayor probabilidad de escapar óptimos locales que con GD: un punto estacionario de la función objetivo en GD no lo será en SGD generalmente.

Propiedades del SGD

- *Ejercicio: demostrar que el estimador por minilotes es **insesgado**.*
- Usando resultados de aproximación estocástica de Robbins & Monro (1954), se puede demostrar que si las tasas de aprendizaje cumplen estas condiciones:

$$\sum_{t=0}^{\infty} \eta_t = \infty$$
$$\sum_{t=0}^{\infty} \eta_t^2 < \infty$$

entonces

$$|f(w^t) - f^*| = \mathcal{O}(1/t)$$

Nuevos desarrollos desde SGD

Momento (1986)

Ayuda a amortiguar las oscilaciones que hacen que SGD sea lento.

$$w^{t+1} = w^t - v^{t+1}$$
$$v^{t+1} = \gamma v^t + \eta_t \nabla_w f_i(w^t)$$

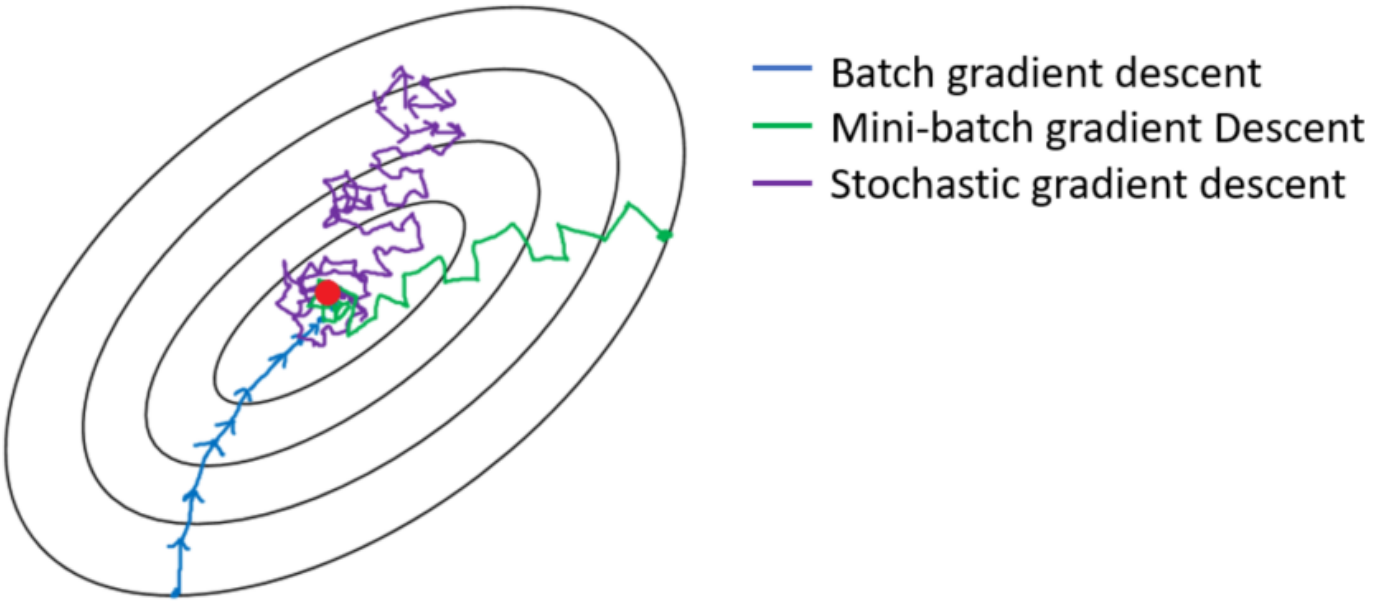
AdaGrad (2011)

Adapta la tasa de aprendizaje a cada parámetro, disminuyéndola en parámetros con actualizaciones frecuentes (resp. aumentándola en parámetros con actualizaciones infrecuentes). Por esta razón, es adecuado para matrices de datos dispersas.

$$w_j^{t+1} = w_j^t - \frac{\eta}{\sqrt{G_{j,j}^t + \epsilon}} \nabla_w f_i(w^t)_j$$

donde $G_{j,j}^t = \sum_{t'=0}^{t} (\nabla_w f_i(w^t)_j)^2$ (la suma de los gradientes al cuadrado para esa coordenada hasta t)

Efecto de la estocasticidad



Con o sin momento

- Símil físico con la **inercia** de una partícula.

SGD con datos dispersos

- Caso real: recomendación de películas a usuarios. Las observaciones son del tipo *(id_usuario, id_pelicula, rating)*
- Los identificadores son variables categóricas (factor). Por ejemplo, asumiendo 50.000 usuarios:

$$\text{id\_usuario} = 24678 \rightarrow (0, 0, 0, \ldots, 0, 1, 0, \ldots) \in \mathbb{R}^{50000}$$

- Esto produce una explosión en el número de variables dummy necesarias: riesgo de que **no quepa en memoria** la matriz de datos dummies.
- Solución: trabajar directamente sin dummificar, usando librerías especializadas como Vowpal Wabbit (Microsoft): https://github.com/VowpalWabbit/vowpal_wabbit
- SGD (o variantes) solo actualizarán el peso correspondiente, ignorando los que tendrían el dummy a 0.

Feature hashing trick

- En muchas ocasiones, las variables categóricas están representadas como **cadenas alfanuméricas**.
- Ejemplo en datos de ad-server: un identificador de un cookie es **76c24efd-ec42-492a-92df-c62cfd4540a3**.
- ¿Cómo lo convertimos a un índice entero de forma eficiente?
- Solución: usar una **función de hash** sobre la representación en binario de la cadena

$$\mathcal{H} : \{0,1\}^D \to \{0,1\}^d$$

- donde $d < D$. Típicamente d se toma entre 15 y 30.
- Por ejemplo, $\mathcal{H}(\text{76c24efd-ec42-492a-92df-c62cfd4540a3}) = 65538$
- El algoritmo de optimización solo hace la computación para actualizar el peso w_{65538}

Interacciones y Máquinas de Factorización (FMs)

Fm's - Problema

- ¿Cómo modelizar interacciones cuando nos enfrentamos a variables categóricas con número alto de categorías?
- **Una** variable con $K + 1$ categorías $\rightarrow \binom{K}{2}$ interacciones a pares. Explota rápido...
- Número medio de valores distintos de cero en los vectores de variables predictoras mucho menor que su dimensión.
- Datos muy dispersos (sparse)! No hay datos suficientes para estimar interacciones complejas de manera independiente...

FM's - Idea

- El modelo de una FM de grado 2 (solo interacciones a pares)

$$\hat{y}(x) := \sigma \left[\omega_0 + \sum_{i=1}^{p} \omega_i x_i \sum_{i=1}^{p} \sum_{j=i+1}^{p} \langle v_i, v_j \rangle x_i x_j \right]$$

- Donde los parámetros a estimar son

$$\omega_0 \in \mathbb{R},\ \boldsymbol{\omega} \in \mathbb{R}^p,\ \ \boldsymbol{V} \in \mathbb{R}^{p \times k}$$

- k es la dimensión latente! $\langle v_i, v_j \rangle$ es un producto escalar

$$\langle v_i, v_j \rangle = \sum_{f=1}^{k} v_{i,f} v_{j,f}$$

- Reducimos número de parámetros de $1 + p + \frac{p(p-1)}{2}$ a $1 + p + kp$.

FM's - Intuición y computación

- Con datos sparse, no hay datos suficientes para estimar todas las interacciones de manera independiente.
- Las FMs pueden estimar interacciones incluso en este contexto porque rompen la independencia entre los parámetros de las interacciones, factorizandolos.
- Los datos de una interacción ayudan a estimar los parámetros de otras interacciones relacionadas.
- La complejidad de computar el modelo de FMs es $\mathcal{O}(kp^2)$, pues hay que computar todas las interacciones!
- Se puede reducir esta complejidad a $\mathcal{O}(kp)$ (tiempo lineal) !!
- Fácil de probar: S. Rendle Factorization Machines

Clasificador Naive-Bayes

Clasificador NB

- Sea $f_k(x)$ la densidad de probabilidad de x condicionada a la clase Y_k.
- Sea π_k el prior de la clase Y_k. Se tiene

$$P(Y = Y_k|X = x) = \frac{f_k(x)\pi_k}{\sum_{i=1}^{K} f_i(x)\pi_i}$$

- El modelo NB asume que las variables predictoras son **condicionalmente independientes** dada la clase. O lo que es lo mismo

$$f_k(X) = \prod_{j=1}^{p} f_{kj}(X_k)$$

- Esto en general no es cierto...

Clasificador NB

- ...pero simplificar tremendamente la estimación, cada marginal f_{kj} puede ser estimada por separado.
- Especialemente apropiado cuando la dimensión p es grande (pues la estimación de densidad se vuelve inviable).
- También cuando el vector de variables predictoras contiene variables discretas y continuas (pues cada una se modeliza por separado).
- Aunque la hipótesis es fuerte, el modelo puede funcionar porque la frontera de decisión puede "no sentir" detalles de las densidades condicionadas a la clase.

Clasificador NB en la práctica - estimación de parámetros

- Los parámetros se estima usando máxima verosimilitud con los datos de entrenamiento.
- Para **variables predictoras contínuas** - NB Gaussiano

$$P(X_i = x_i | Y = Y_k) = \frac{1}{\sqrt{2\pi\sigma_{ki}^2}} \exp\left[-\frac{(x_i - \mu_{ki})^2}{2\sigma_{ki}^2} \right]$$

Clasificador NB en la práctica - estimación de parámetros

- Para **variables predictoras que son cuentas** - Versión suavizada de máxima verosimilitud con modelo multinomial
- Un vector de variables predictoras es un histograma con x_i el número de veces que sucede el evento i.

$$P(X_i = x|Y = Y_k) = \frac{N_{ki} + \alpha}{N_k + \alpha p}$$

- N_{ki} es el número total de cuentas del evento i en la clase k y N_k es el número total de cuentas en la clase k.
- α se puede interpretar como prior, evita probabilidades 0. $\alpha = 1$ *Laplace smoothing*. $\alpha < 1$ *Lidstone smoothing*.

Clasificador NB en la práctica - estimación de parámetros

- Para **variables predictoras categóricas** - MLE con modelo Bernoulli multivariante.
- Cada variable predictora es un booleano, $x_i \in \{0, 1\}$.

$$P(X_i = x | Y = Y_k) = \frac{\sum_{j=1}^{N} I(X_i = x) * I(Y = Y_k)}{\sum_{j=1}^{N} I(Y = Y_k)}$$

k Vecinos Más Próximos (k-NN)

Fundamentos

- k-NN es un algoritmo robusto y versátil que suele usarse como base antes de modelos más complejos.
- Es de tipo **supervisado**: aprende una función $h : \mathcal{X} \to \mathcal{Y}$, donde $\mathcal{Y}$ puede ser discreto (clasificación) o continuo (regresión).
- Es **no paramétrico**: no hace ninguna suposición acerca de la estructura de h (por ejemplo, que sea lineal, $h(x) = w^\intercal x$). Esto ayuda para prevenir errores de modelización.
- El aprendizaje es **basado en instancias**: en lugar de aprender parámetros, **memoriza** los datos de entrenamiento, que serán usados directamente como *conocimiento* para la fase de inferencia.
- En consecuencia: solo al predecir sobre datos de test el algoritmo usa los datos de entrenamiento.

Algoritmo

(Pre)entrenamiento

1. Almacenar el dataset de entrenamiento $\mathcal{D}_{tr}$.
2. Especificar una función de distancia $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$

Predicción

1. Dado una instancia de test x_0, encontrar los k puntos $\{x_{(1)}, \ldots, x_{(k)}\} := \mathcal{A} \subset \mathcal{D}_{tr}$ más cercanos a x_0 según d.
2. La clase predicha será la **mayoritaria** de las clases de los elementos en $\mathcal{A}$, es decir,

$$P(y = j | x_0) = \frac{1}{k} \sum_{i \in \mathcal{A}} I(y_{(i)} = j)$$

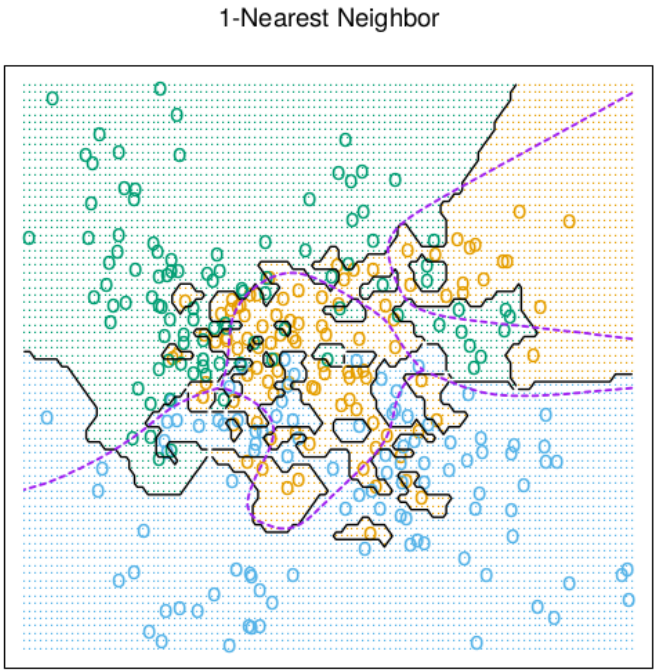
donde I es la función indicatriz.

Funciones de distancia

- Supongamos $x_1, x_2 \in \mathcal{X} = \mathbb{R}^D$.
- **Distancia L1**: $d_1(x_1, x_2) = \sum_{d=1}^{D} |x_{1,d} - x_{2,d}|$.
- **Distancia L2**: $d_2(x_1, x_2) = \sqrt{\sum_{d=1}^{D} (x_{1,d} - x_{2,d})^2}$.
- **L1 vs L2**: la L2 penaliza mucho más puntos alejados que la L1.
- ¡Es conveniente estandarizar las variables a media 0 y varianza 1 ya que pueden estar en distintas escalas!

El hiperparámetro k

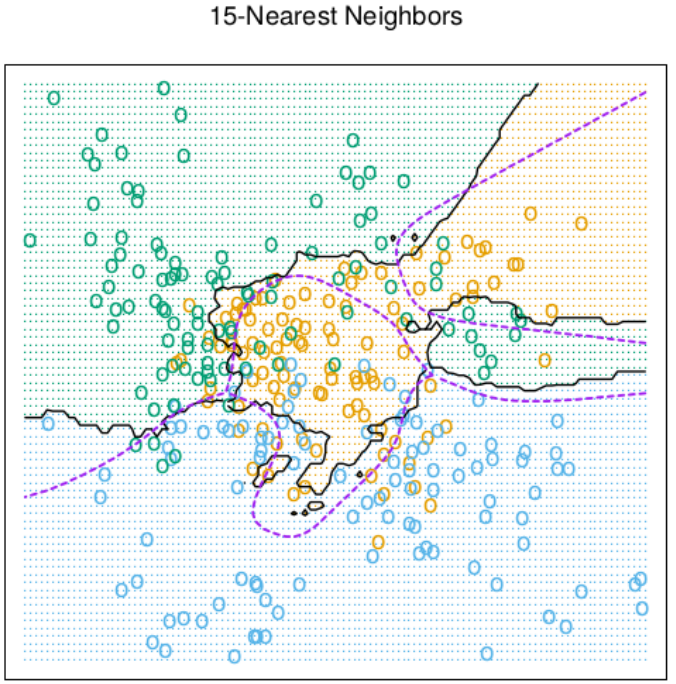
- Intuititivamente, aumentar k tiende a suavizar la frontera de decisión, es decir, el clasificador es más resistente a outliers.



Fuente: Elements of Statistical Learning

El hiperparámetro k

- Intuititivamente, aumentar k tiende a suavizar la frontera de decisión, es decir, el clasificador es más resistente a outliers.



Fuente: Elements of Statistical Learning

Ventajas

- Sencillo y válido para regresión y clasificación multiclase (no solo binaria).
- **No hace suposiciones** sobre la estructura de los datos.

Inconvenientes

- Predicción **muy costosa en tiempo**: en la mayoría de las aplicaciones, interesa que la predicción sea rápida y el entrenamiento lento.
- Necesario almacenar todo el dataset de entrenamiento.
- En **alta dimensión**, las distancias no son intuitivas.

Propiedades asintóticas

- **Ejercicio**: probar que a medida que el número de observaciones de entrenamiento $N \to \infty$, 1-NN tiene una tasa de error acotada por el doble la tasa de error de Bayes.

- Error de Bayes = $1 - p_{k^*}(x)$.

- Error de 1-NN = $\sum_{k=1}^{K} p_k(x)(1 - p_k(x)) \geq 1 - p_{k^*}(x)$.
- Si $K = 2$: Error de 1-NN = $2p_{k^*}(x)(1 - p_{k^*}(x)) \leq 2(1 - p_{k^*}(x))$

En la práctica: resumen

- Preprocesar los datos: estandarizar.
- Si los datos tienen mucha dimensionalidad, considera utilizar técnicas de reducción de dimensionalidad.
- Validar en el hiperparámetro k y la distancia d.
- Si tiempo es crítico, considerar usar variantes aproximadas: https://github.com/eddelbuettel/rcppannoy

Métricas para clasificación

Problema

- La **tasa de acierto** (accuracy) no basta en problemas con clases poco equilibradas.
- Caso real: predicción de CTR (**click-through rate**, click 1, no click 0).
- Alrededor de 10^8 anuncios (observaciones), de los cuales clicks solo 80.000 clicks.
- Un modelo que clasifique siempre 0, obtiene una tasa de acierto del **99.92 %**...

Calidad modelos clasificación

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Medidas:

- Tasa de acierto: $\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}}$
- Sensitividad, recall, TPR: $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
- Especificidad, TNR: $\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$
- Precisión, PPV: $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$
- F1-score: $2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}}$

Tutorial muy extenso: http://people.cs.bris.ac.uk/~flach/ICML04tutorial/

(Análisis ROC, generalizaciones a clasificación n-aria,...)