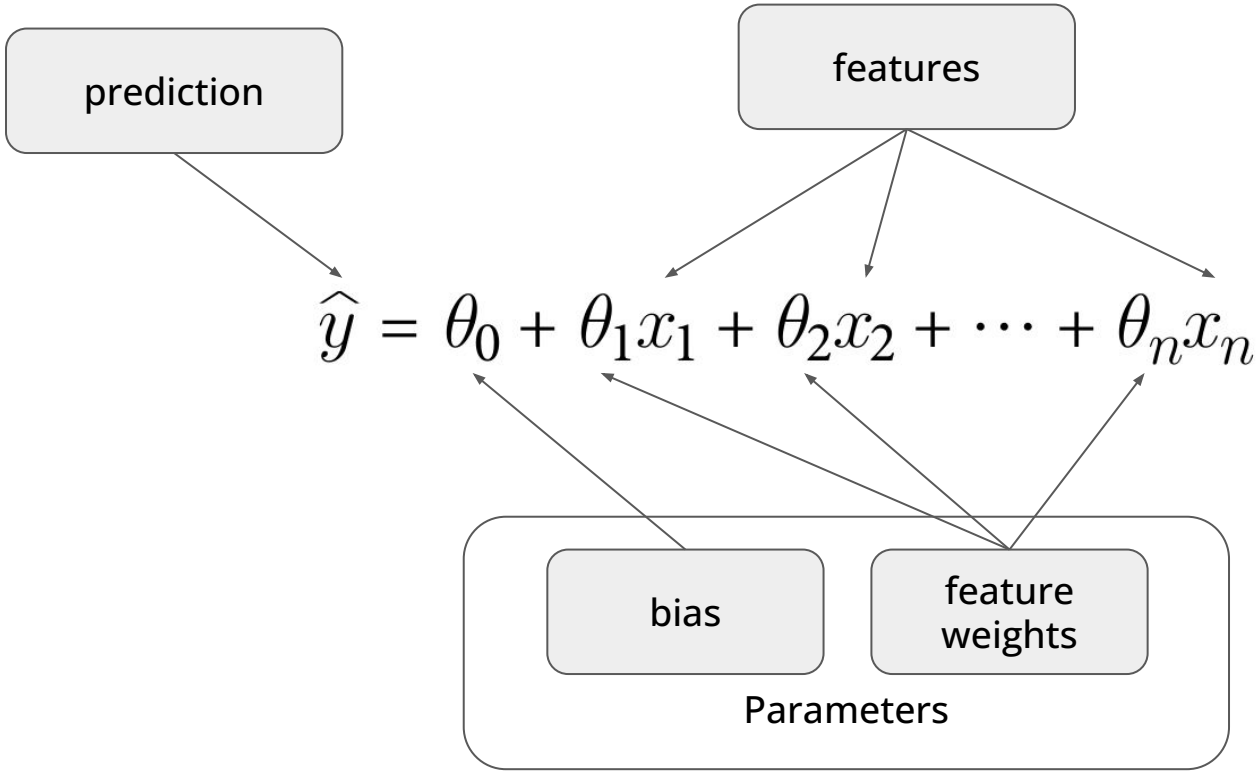
Entrenamiento de modelos

Agenda

- Regresión lineal
  - Repaso
  - Entrenamiento - ecuación normal
- Descenso por gradiente
  - Algoritmo genérico de optimización
  - Aplicación al caso de la regresión lineal
- Descenso por gradiente estocástico (SGD)
- Regresión polinomial
  - Es una regresión lineal con más características
- Regularización de modelos lineales
  - Restringir los parámetros para evitar el sobre-ajuste
- Regresión logística
  - Predecir probabilidades

Regresión lineal


prediction
features
$\widehat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
bias
feature weights
Parameters

$$\hat{y} = h_{\underline{\theta}}(\underline{x}) = \underline{\theta} \cdot \underline{x} = \underline{\theta}^t \underline{x}$$

$$\underline{\theta}^t = [\theta_0, \theta_1, \theta_2, \text{———}, \theta_n]$$

$$\underline{x}^t = [1, x_1, x_2, \text{———}, x_n]$$

$$\underline{\theta}^t \underline{x} = [\theta_0, \text{—}, \theta_n] \begin{bmatrix} 1 \\ x_1 \\ | \\ | \\ x_n \end{bmatrix}$$

h_θ = función hipótesis

$\underline{\theta}$ = vector de parámetros

$\underline{x}$ = vector de características

- Hipótesis es una función paramétrica $$\widehat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

- Función de costo: MSE $$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

- Entrenamiento
  - Encontrar los parámetros que minimizan la función de costo

Ecuación normal

- Para este caso concreto podemos encontrar el óptimo en “forma cerrada”
- Ecuación normal $\widehat{\boldsymbol{\theta}} = \left(\mathbf{X}^\intercal \mathbf{X}\right)^{-1} \mathbf{X}^\intercal \mathbf{y}$ $\widehat{\boldsymbol{\theta}} = \mathbf{X}^{+}\mathbf{y}$
- Se calcula usando SVD $\mathbf{X} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^\intercal$ $\mathbf{X}^{+} = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^\intercal$
- Complejidad computacional del entrenamiento
  - Lineal en la cantidad de muestras **m**
  - Cuadrática en la cantidad de características **n**
- Una vez entrenado el modelo lineal la inferencia es muy rápida
  - Lineal en muestras y features

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & & & & \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \quad m \text{ muestras} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$\widehat{\boldsymbol{\theta}} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X}\right)^{-1} \quad \mathbf{X}^{\mathsf{T}} \quad \mathbf{y}$$

Chequeo de dimensiones

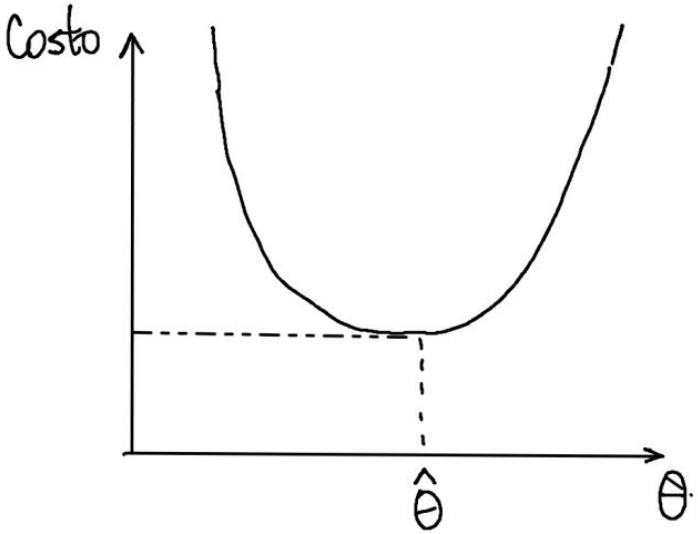
$$\underbrace{\underbrace{\underbrace{X^t}_{[(n+1)\times m]}\underbrace{X}_{[m\times(n+1)]}}_{[(n+1)\times(n+1)]}\underbrace{X^t}_{[(n+1)\times m]}}_{[(n+1)\times m]}\underbrace{y}_{[(m\times 1)]}$$

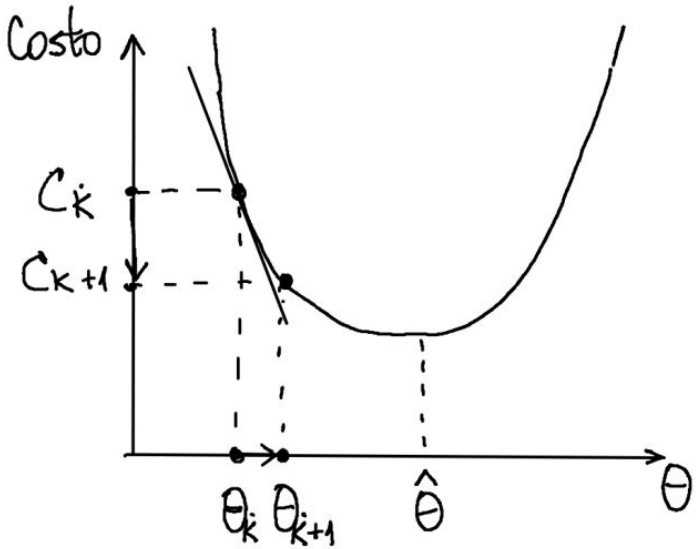
$$(n+1)\times 1$$

Descenso por gradiente

Gradient descent (GD)

- Algoritmo genérico de optimización
- Idea
  - Llegar al óptimo en forma iterativa
  - En cada paso:
    - Buscar la dirección de mayor crecimiento del costo y moverse en el sentido opuesto



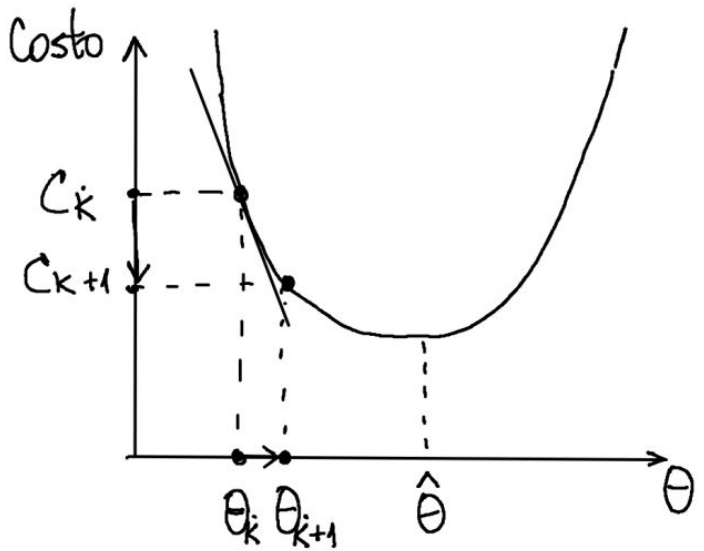


Idea

- Llegar al óptimo en forma iterativa
- En cada paso:
  - Buscar la dirección de mayor crecimiento del costo y moverse en el sentido opuesto

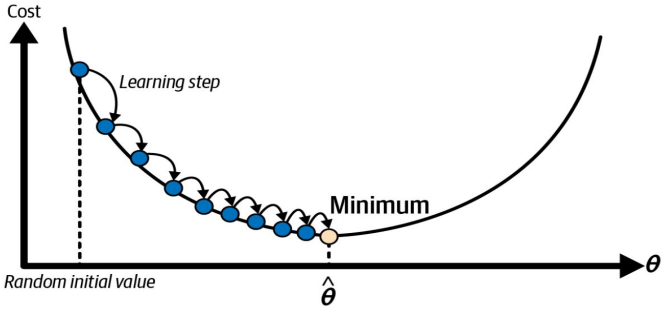
$$\nabla C(\theta) = \begin{bmatrix} \frac{\partial C}{\partial \theta_0} \\ \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_n} \end{bmatrix}$$

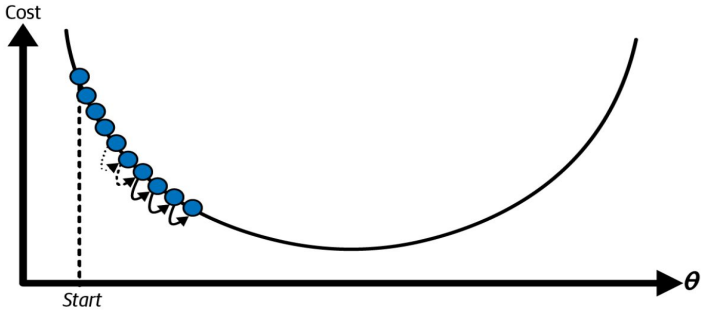
$$\theta^{k+1} = \theta^{k} - \eta \nabla_{\theta} C(\theta)$$



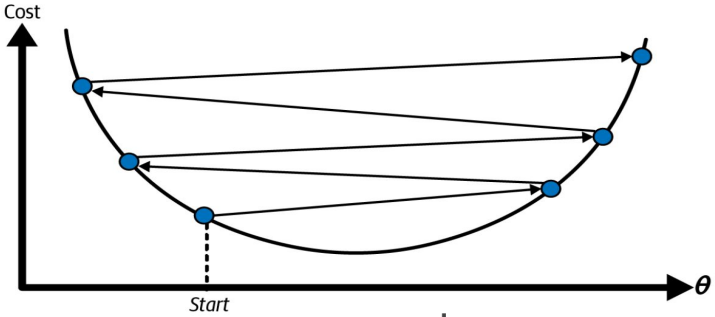
Learning rate

- Hiperparámetro η





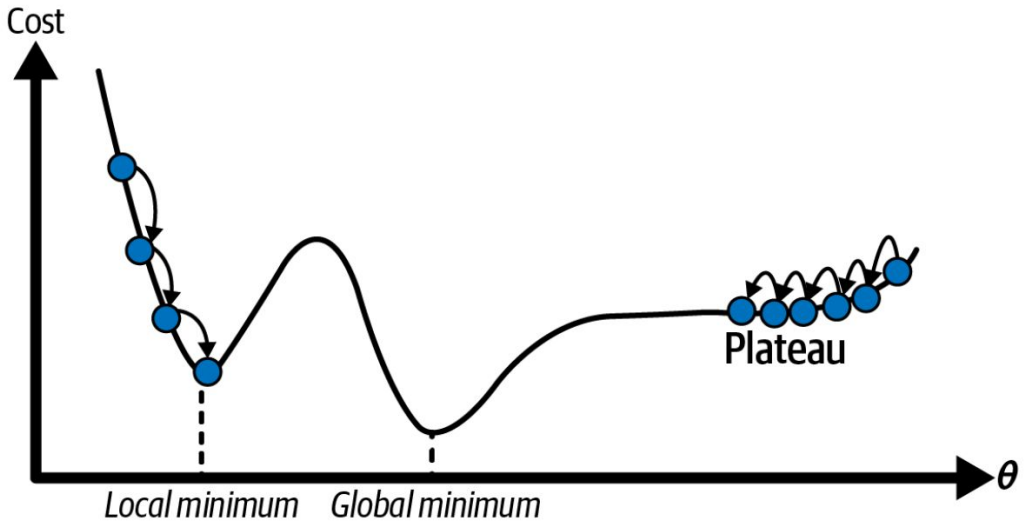
muy chico



muy grande

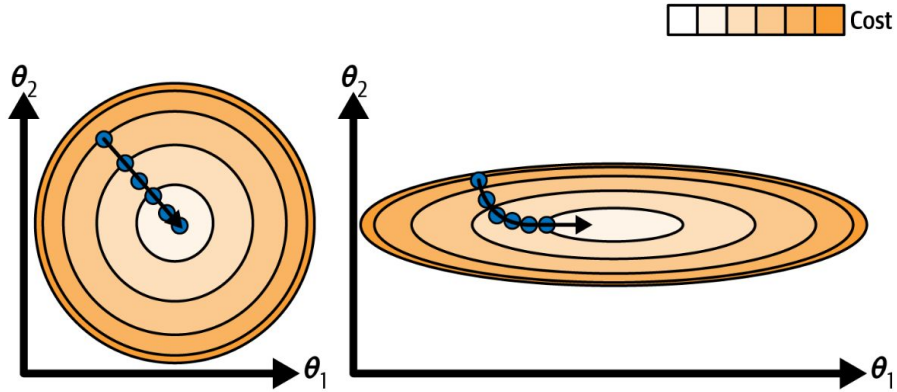
Mínimos locales y globales

- Si la función de costo es convexa -> llegamos a un mínimo global
  - Es el caso de LR con el costo MSE
- Si la función de costo no es convexa
  - dependiendo de la inicialización aleatoria de parámetros
    - descenso puede llegar a un mínimo local
    - descenso puede ser muy lento en una región donde el costo sea casi plano



Escalado de features

- En la gráfica derecha
  - el feature 1 tiene una escala mucho menor al feature 2
  - gradientes más grandes según Θ_2
  - descenso más rápido al inicio y luego más lento
- En la gráfica izquierda
  - Igual escala
- Al optimizar mediante GD es importante que los features tengan similar escala
  - Es recomendable pre-procesar los datos



GD para la regresión lineal

GD para la regresión lineal

$$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^{(i)} - y^{(i)}\right)^2$$

$$\frac{\partial}{\partial\theta_j}\mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

$$\nabla_{\boldsymbol{\theta}}\,\mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial\theta_0}\mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial\theta_1}\mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial\theta_n}\mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m}\mathbf{X}^{\mathsf{T}}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathrm{MSE}(\boldsymbol{\theta})$$

GD para la regresión lineal

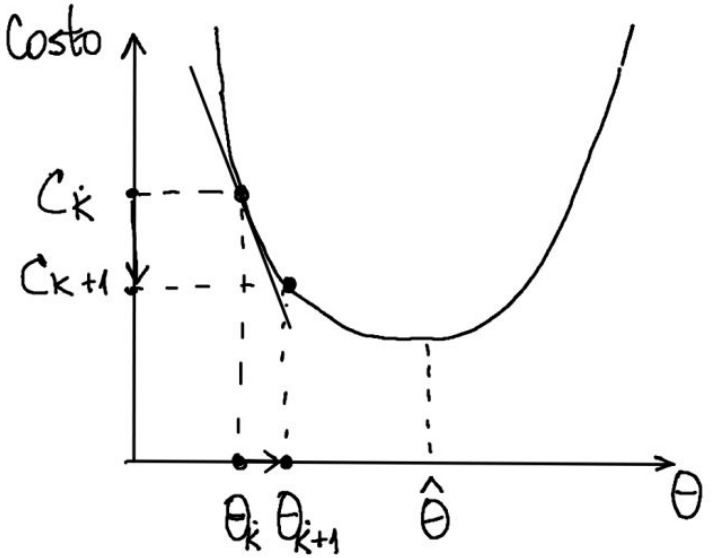
$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\intercal (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix}$$

$$\hat{y} = X\theta$$

GD para la regresión lineal

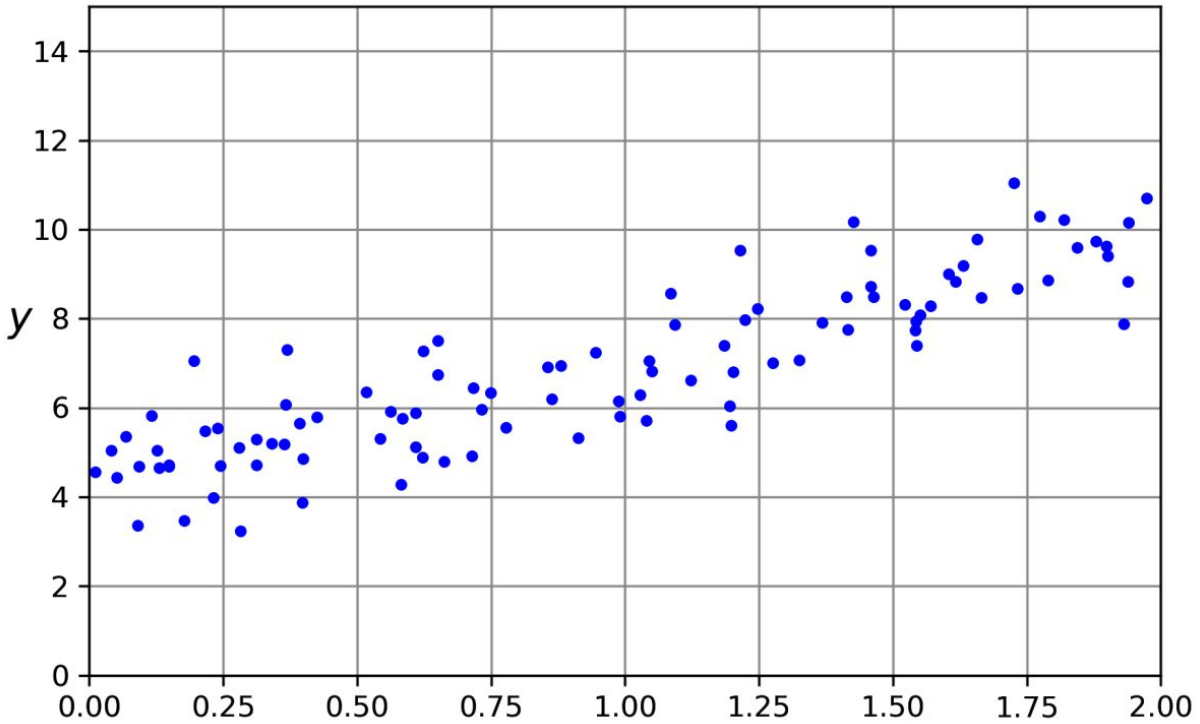
- Inicializar en forma aleatoria el vector de parámetros
- Iterar:
  - Calcular el gradiente para el vector actual de parámetros
  - Actualizar el vector de parámetros
  - Salir si
    - norma del gradiente menor que tol



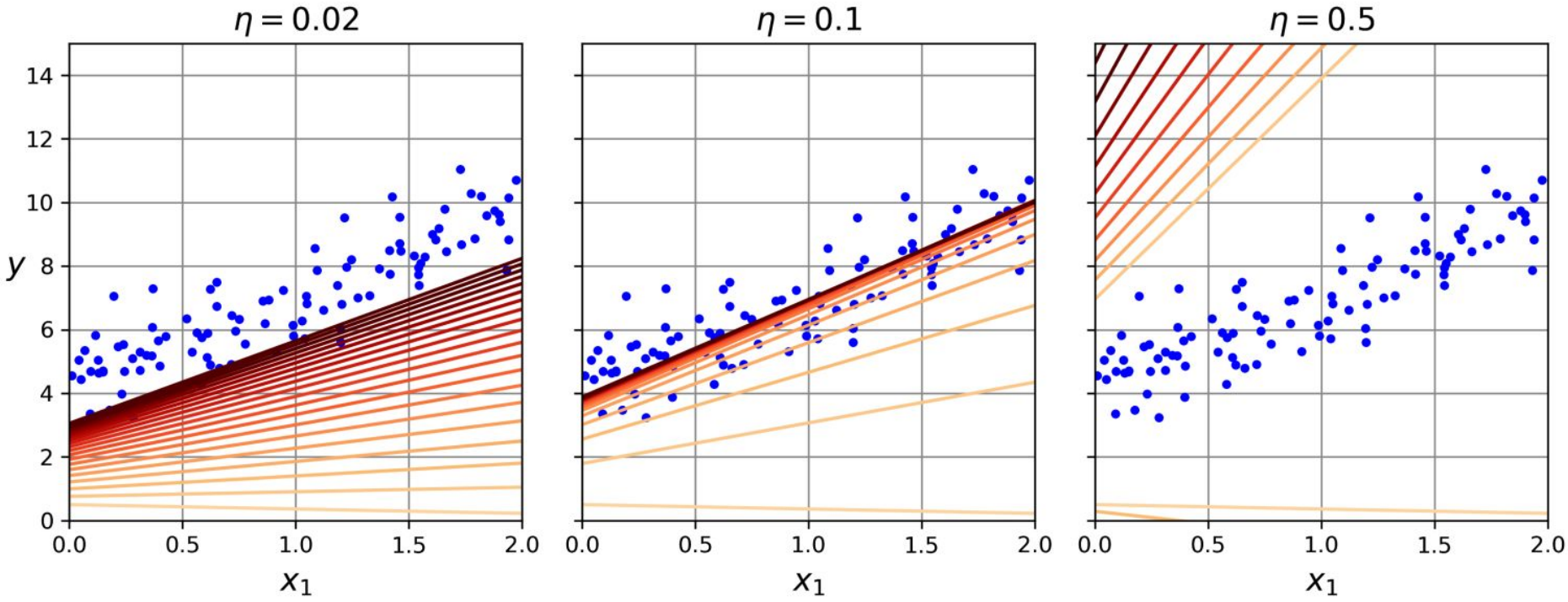
$$\nabla C(\theta_k) = \frac{2}{m} X^t (X\theta_k - y)$$

$$\theta_{k+1} = \theta_k - \eta \nabla C(\theta_k)$$

GD para la regresión lineal



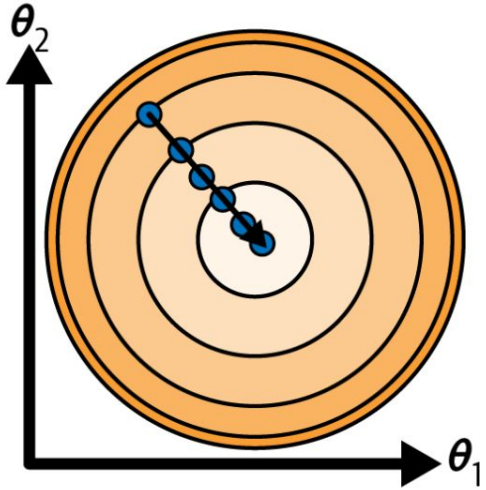
Learning rate



Stochastic gradient descent

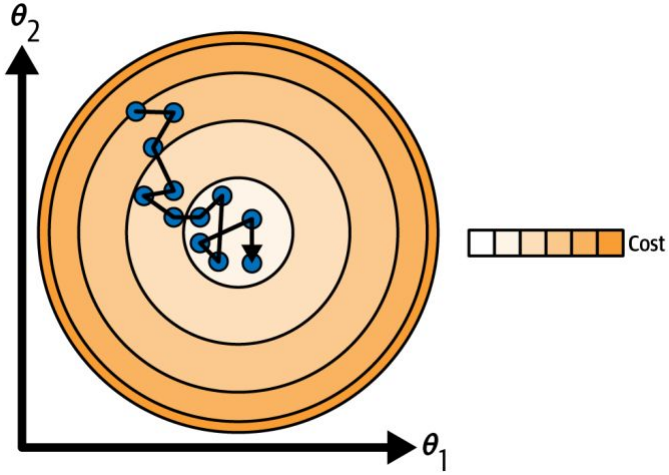
Batch gradient descent

- Lo que vimos hasta el momento
- En cada paso de la iteración
  - Se usan todas las muestras para calcular el gradiente
- Buena estimación del gradiente
- Costoso computacionalmente



Stochastic gradient descent (SGD)

- En cada paso, para calcular el gradiente
  - No se usan todas las muestras
  - Se usa una sola muestra al azar
- Gradiente
  - mucho más rápido de calcular
  - peor estimado, más “ruidoso”
- El descenso no es tan regular hacia el mínimo
- La irregularidad puede eventualmente permitir “zafar” de algún mínimo local



- Al acercarse al mínimo continuaría oscilando
  - Se usa reducir el “learning rate” en forma progresiva (learning schedule)

Batch GD - Ejemplo de implementación a mano para LR

```
eta = 0.1  # learning rate
n_epochs = 1000
m = len(X_b)  # number of instances

np.random.seed(42)
theta = np.random.randn(2, 1)  # randomly initialized model parameters

for epoch in range(n_epochs):
    gradients = 2 / m * X_b.T @ (X_b @ theta - y)
    theta = theta - eta * gradients
```

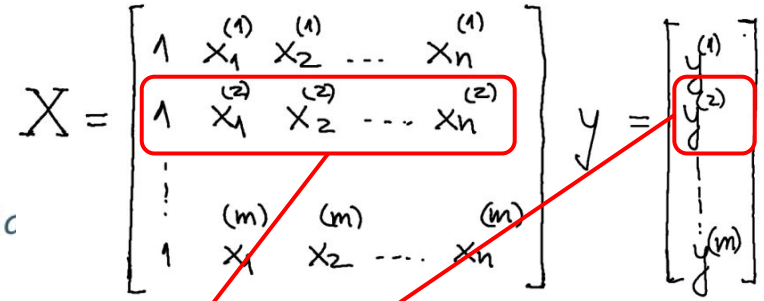
$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & & & & \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$\theta^{(\text{next step})} = \theta - \eta \nabla_\theta \text{MSE}(\theta)$$

$$\nabla C(\theta_k) = \frac{2}{m} X^t (X\theta_k - y)$$

SGD - Ejemplo de implementación a mano para LR

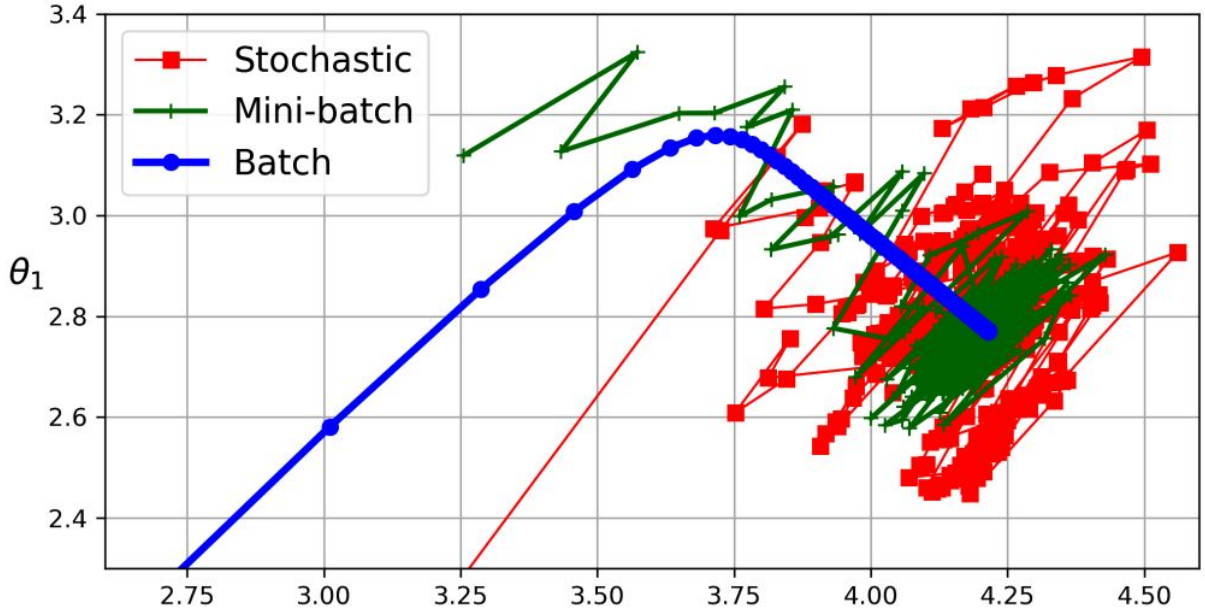
```
n_epochs = 50
t0, t1 = 5, 50  # learning schedule hyperparameters

def learning_schedule(t):
    return t0 / (t + t1)

np.random.seed(42)
theta = np.random.randn(2, 1)  # random initic

for epoch in range(n_epochs):
    for iteration in range(m):
        random_index = np.random.randint(m)
        xi = X_b[random_index : random_index + 1]
        yi = y[random_index : random_index + 1]
        gradients = 2 * xi.T @ (xi @ theta - yi)  # for SGD, do not divide by m
        eta = learning_schedule(epoch * m + iteration)
        theta = theta - eta * gradients
```



Minibatch SGD

- En lugar de una muestra, tomar al azar un conjunto pequeño de muestras
- Gradiente algo mejor estimado
- Sólo levemente más lento
- Descenso un poco más regular

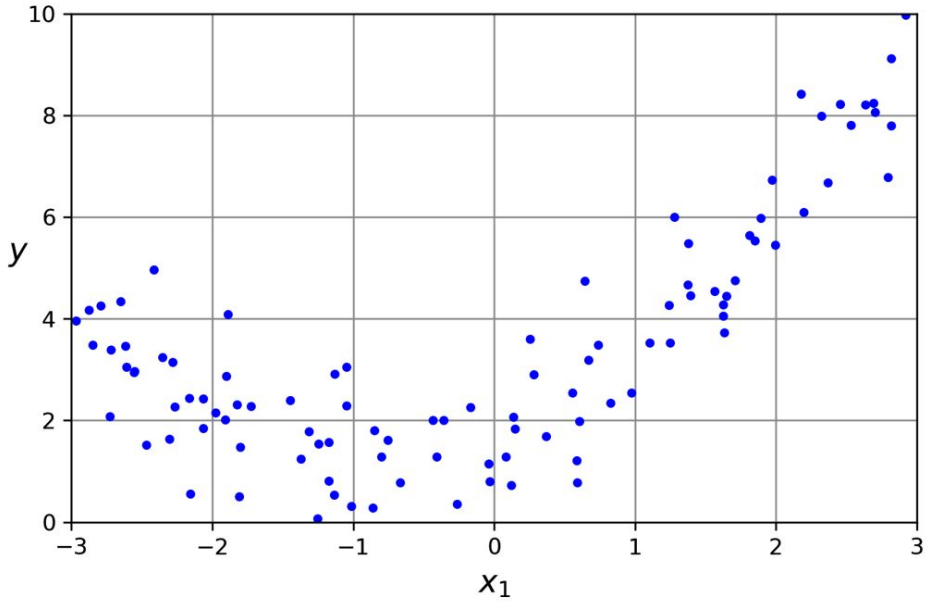
Evolución hacia el óptimo en el espacio de parámetros



Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	N/A
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	N/A

Regresión polinomial

```
np.random.seed(42)
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```

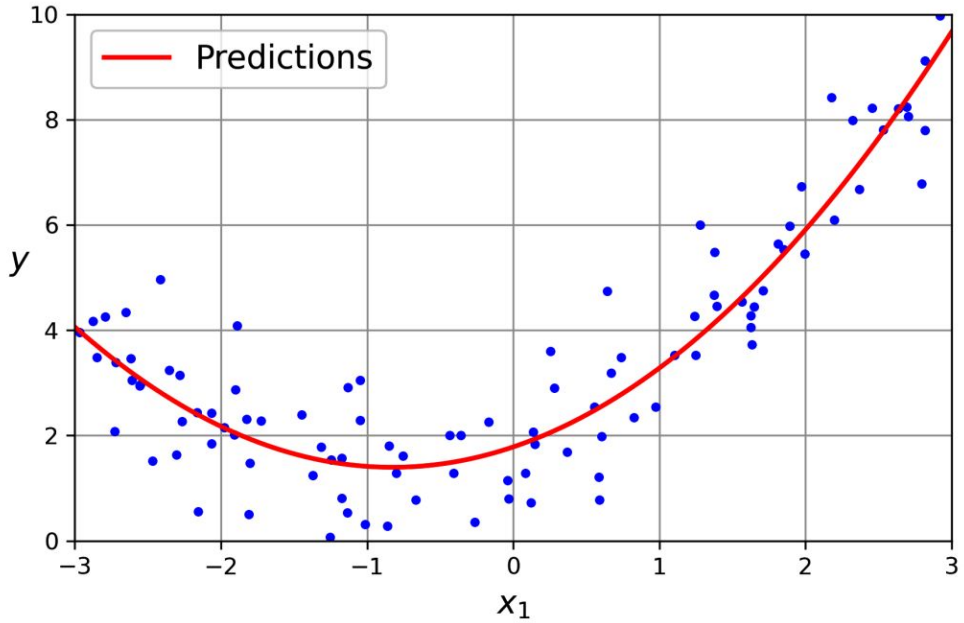


Features polinomiales

- Agregamos más features con las potencias de las entradas
- Queda un problema de mayor dimensión
- En el ejemplo:
  - Feature 1: x
  - Feature 2: x^2 (nueva)
- Aplicamos una regresión lineal para las nuevas características

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly_features = PolynomialFeatures(degree=2, include_bias=False)
>>> X_poly = poly_features.fit_transform(X)
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929,  0.56664654])
```

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```



Nuevas features

n = 2 x_1 x_2 — 2 features

d=2 $x_1, x_2, x_1^2, x_1x_2, x_2^2$ — 5 features

d=3 $x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3$ — 9 features

`PolynomialFeatures(degree=`*d*`)` transforms an array containing *n* features into an array containing $(n + d)! / d!n!$ features, where *n*! is the *factorial* of *n*, equal to $1 \times 2 \times 3 \times \cdots \times n$. Beware of the combinatorial explosion of the number of features!

Regularización de modelos lineales

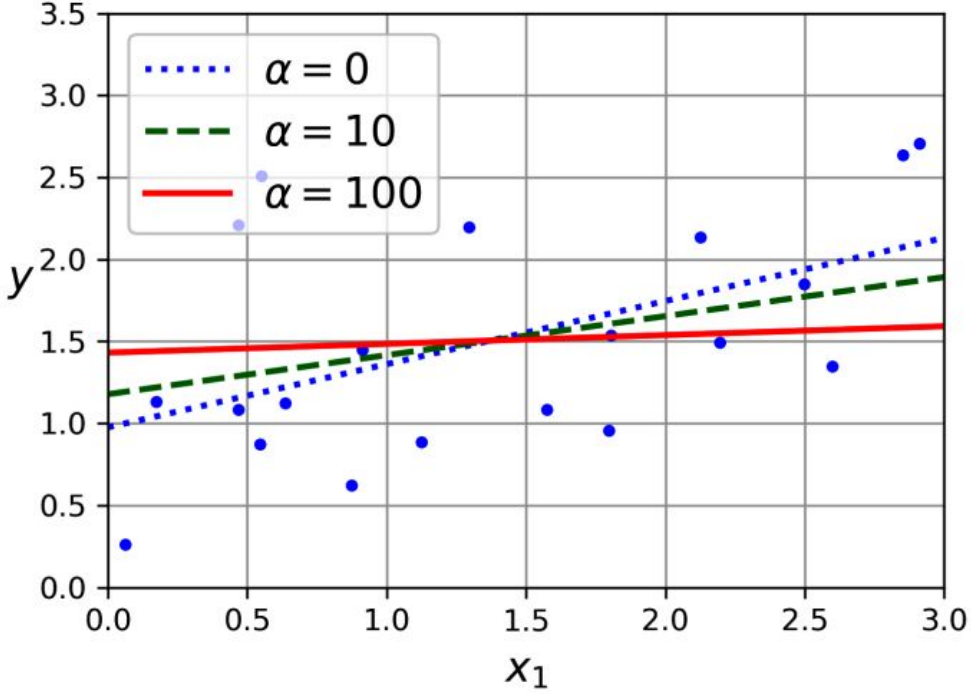
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

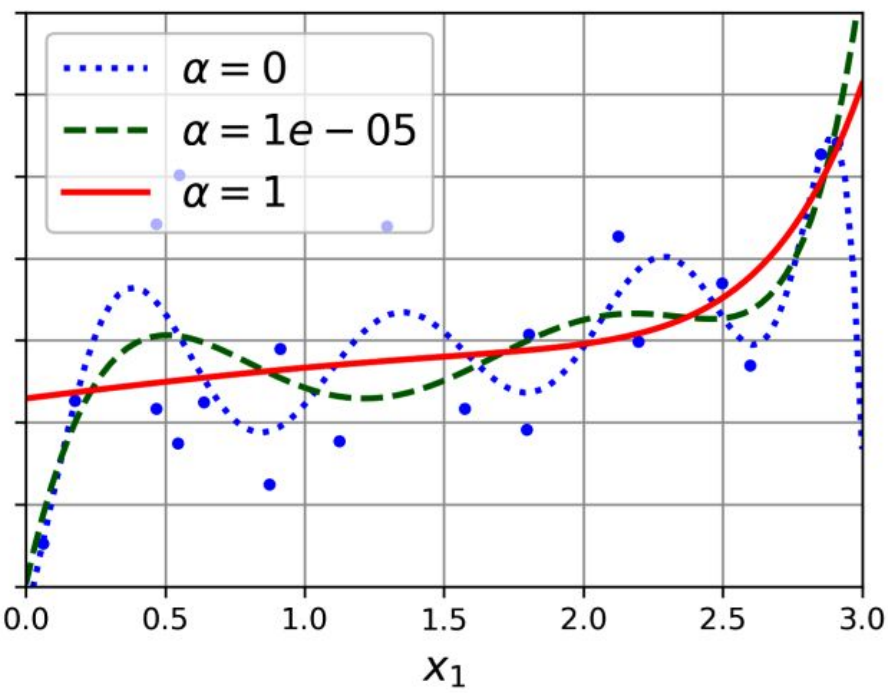
Regularización

- Idea:
  - Evitar overfitting acotando el modelo
  - Poner restricciones sobre los parámetros
  - Reducir la varianza a costa de un sesgo algo mayor

- Implementación:
  - Agregamos un término a la función de costo que impone una restricción sobre los parámetros
    - Sólo para el entrenamiento
    - No se incluye el bias
  - Se aproximan los datos pero se mantienen a la vez los parámetros con valores pequeños
  - El hiperparámetro α controla la regularización
    - Con α = 0 es directamente el costo MSE
    - Si α es grande los parámetros óptimos serán pequeños

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \frac{\alpha}{m}\sum_{i=1}^{n}\theta_i^2$$


$\alpha = 0$
$\alpha = 10$
$\alpha = 100$
y
x_1


$\alpha = 0$
$\alpha = 1e-05$
$\alpha = 1$
x_1

Alternativas

- Ridge regression

$$J(\mathbf{\theta}) = \text{MSE}(\mathbf{\theta}) + \frac{\alpha}{m}\sum_{i=1}^{n}\theta_i^2$$

- Lasso regression

  "Least absolute shrinkage and selection operator"

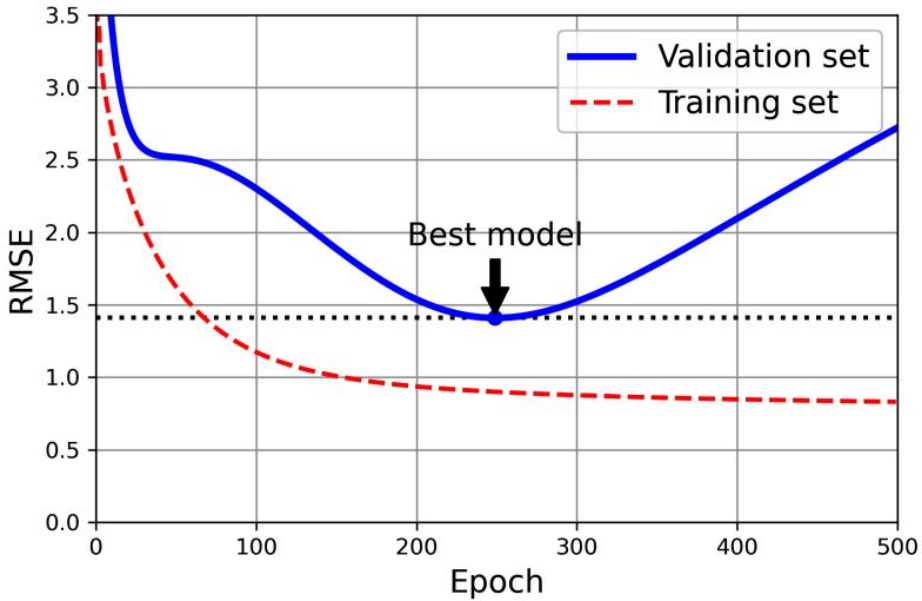
$$J(\mathbf{\theta}) = \text{MSE}(\mathbf{\theta}) + 2\alpha\sum_{i=1}^{n}|\theta_i|$$

- Elastic Net

$$J(\mathbf{\theta}) = \text{MSE}(\mathbf{\theta}) + r\left(2\alpha\sum_{i=1}^{n}|\theta_i|\right) + \left(1-r\right)\left(\frac{\alpha}{m}\sum_{i=1}^{n}\theta_i^2\right)$$

Otra forma de regularizar

- Early stopping
  - Parar el entrenamiento (por ejemplo el GD) cuando el error de validación comienza a crecer



Regresión logística

Modelos lineales

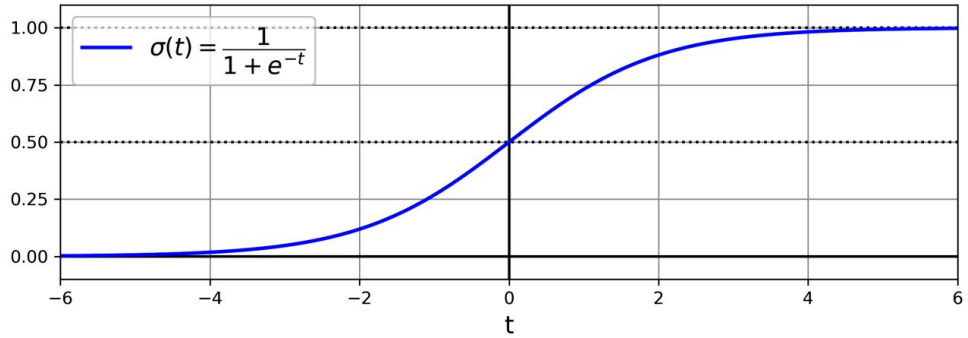
Problema de ejemplo	Sub-problema	Modelo	Función de costo Algoritmo
	Aprobar o rechazar	Perceptrón	Error de clasificación PLA
Análisis de crédito	Cantidad de crédito	Regresión lineal	MSE Ecuación normal
	Probabilidad de default	Regresión logística	Cross-entropy error Gradient descent

Regresión logistica

- Se hace una suma ponderada de las entradas (igual que LR)
  - A la suma ponderada se le llama score o logit
- Se devuelve como predicción el “logistic” del score
- Logistic σ(.) es una función sigmoide (forma de S)

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x})$$

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Clasificación

- A pesar de llamarse regresión se usa para clasificación
- Ej. Si la probabilidad estimada para una instancia es mayor que cierto umbral, clasificar como de la clase positiva

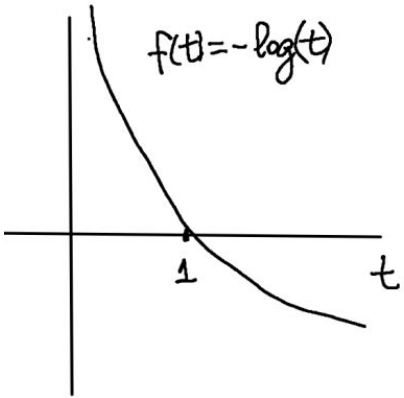
$$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5 \\ 1 \text{ if } \hat{p} \geq 0.5 \end{cases}$$

Entrenamiento

- Objetivo
  - Encontrar parámetros para que
    - modelo estime alta probabilidad para instancias positivas (y=1)
    - modelo estime baja probabilidad para instancias negativas (y=0)

- Costo para una instancia:
  - Costo grande de estimar una probabilidad chica para una muestra positiva
  - Costo grande de estimar una probabilidad grande para una muestra negativa

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\widehat{p}) & \text{if } y = 1 \\ -\log(1 - \widehat{p}) & \text{if } y = 0 \end{cases}$$



- Función de costo para el set de entrenamiento (log loss)

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\Sigma_{i=1}^{m}\left[y^{(i)}log\left(\widehat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \widehat{p}^{(i)}\right)\right]$$

Minimización

- No hay una fórmula cerrada como en el caso de LR
- La función de costo es convexa
- Podemos usar GD y llegar al mínimo global

- Derivadas parciales (gradiente)

$$\frac{\partial}{\partial \theta_j} \mathrm{J}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma\left(\boldsymbol{\theta}^\mathsf{T} \mathbf{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

Similar a las derivadas para el MSE

$$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^\mathsf{T} \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

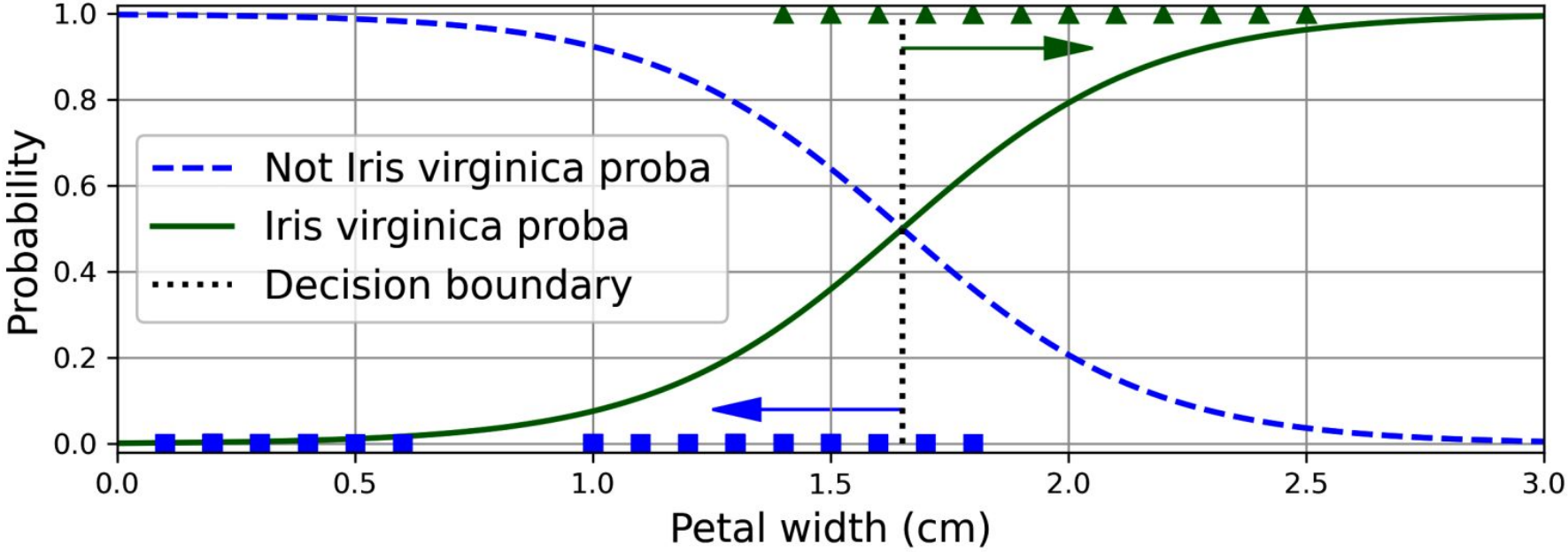
Ejemplo con la base Iris

- Iris virginica vs No Iris virginica sólo considerando el ancho de pétalo
  - Clase 1: Iris virginica
  - Clase 0: No iris virginica

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X = iris.data[["petal width (cm)"]].values
y = iris.target_names[iris.target] == 'virginica'
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

Ejemplo con la base Iris

- Probabilidades estimadas



Extensión a multiples clases - Softmax regression

- K clases
- Un vector de parámetros por clase $\boldsymbol{\theta}^{(k)}$
- Para una instancia x:
  - Calcular los scores por clase $s_k(\mathbf{x}) = \left(\boldsymbol{\theta}^{(k)}\right)^{\mathsf{T}} \mathbf{x}$
  - Normalizar los scores con la función Softmax $\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$

- Función de costo: Cross entropy
  - ver que para K=2 coincide con el log loss

$$J(\boldsymbol{\Theta}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$


FACULTAD DE
INGENIERÍA
UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY