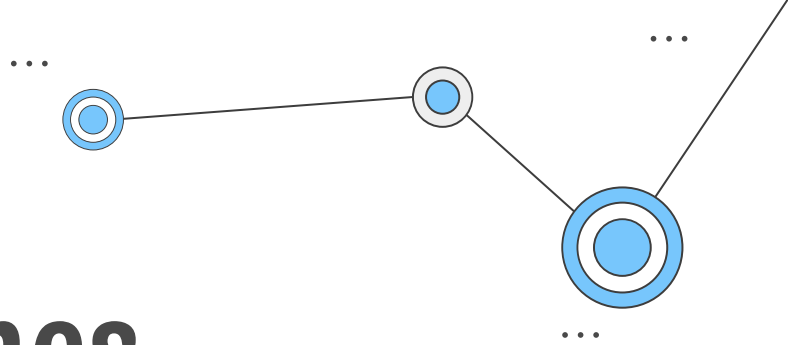


Regresiones

UNIVERSIDAD CATÓLICA DE CÓRDOBA
JESUITAS

Dr. Francisco Arduh
2023

Introducción

- Vimos procedimientos sin detenernos en los modelos
- Entender los algoritmos nos va a ayudar elegir el correcto para el problema qué queremos atacar, elegir el rango de hiper parámetros correctos o entender por qué falla en ciertos contextos.

En esta presentación vamos a

- Analizar la regresión lineal en dos formas diferentes:
  - Usando la “forma cerrada” de la regresión. Esto significa su solución exacta.
  - Usando un enfoque iterativo qué nos va a acercar a la solución gradualmente llamado “Descenso de gradiente” (GD).
- Regresión Polinomial.
- Regresión Logística y Softmax.

Regresión Lineal

Se construye como la suma pesada de las características:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

O en forma vectorial:

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

¿Cómo ajustamos el modelo a los datos?

Necesitamos medir qué tan bien nuestro modelo se ajusta o no a los datos.

Regresión Lineal

Se construye como la suma pesada de las características:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

O en forma vectorial:

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

¿Cómo ajustamos el modelo a los datos?
Necesitamos medir qué tan bien nuestro modelo se ajusta o no a los datos.
Como vimos en la clase anterior podríamos utilizar MSE (RMSE)

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

Ecuación normal (solución exacta)

Para encontrar el valor de los pesos **θ** qué minimiza la función de costo, existe un solución cerrada.

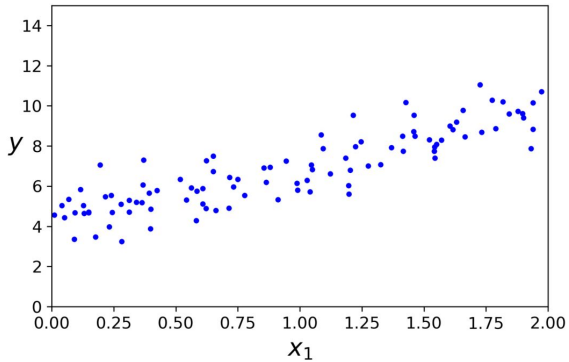
$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^\intercal \mathbf{X})^{-1} \quad \mathbf{X}^\intercal \quad \mathbf{y}$$

Donde $\widehat{\boldsymbol{\theta}}$ es el valor de **θ** qué minimiza MSE.

Lo veamos en un ejemplo, generamos un muestra lineal con ruido:

```
import numpy as np

X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```



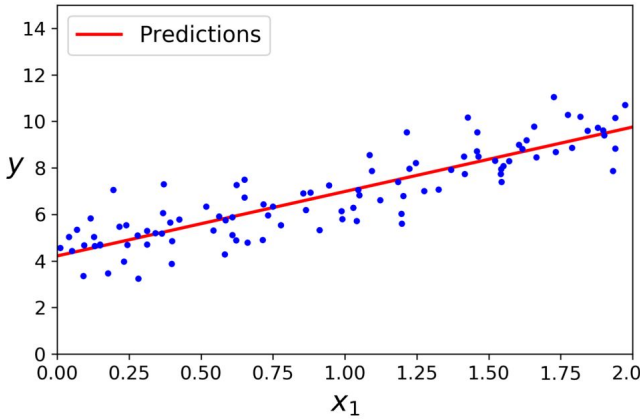
Ecuación normal: Implementación

Utilizando NumPy (Solución exacta):

```
>>> X_b = np.c_[np.ones((100, 1)), X] # add x0 = 1 to each instance
>>> theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
>>> theta_best
array([[4.21509616],
[2.77011339]])
```

Utilizando Scikit-learn (cálculo de pseudo inversa):

```
>>> from sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([4.21509616]), array([[2.77011339]]))
```

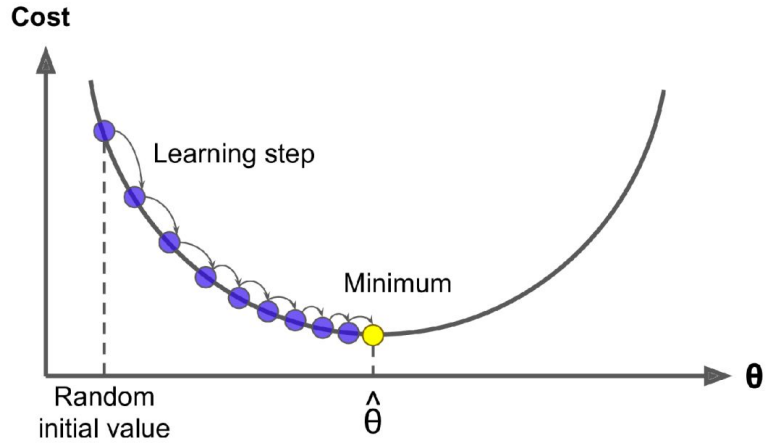


Ecuación normal: complejidad computacional

- Solución exacta (invertir la matriz) su complejidad temporal es de **$O(n^{2.4})$** a **$O(n^3)$**; n es el número de características.
- Solución con Scikit-learn (cálculo de pseudo inversa) su complejidad temporal es de **$O(n^2)$**; n es el número de características.
- La complejidad espacial es lineal con respecto al número de instancias.
- La complejidad para hacer predicciones es lineal al número de instancias y características.

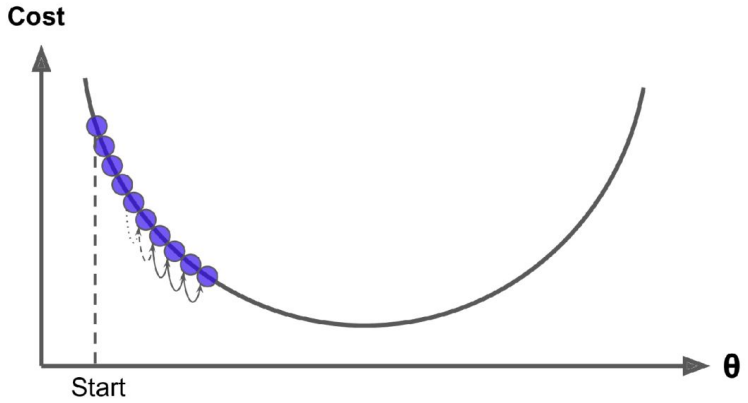
Descenso de gradiente

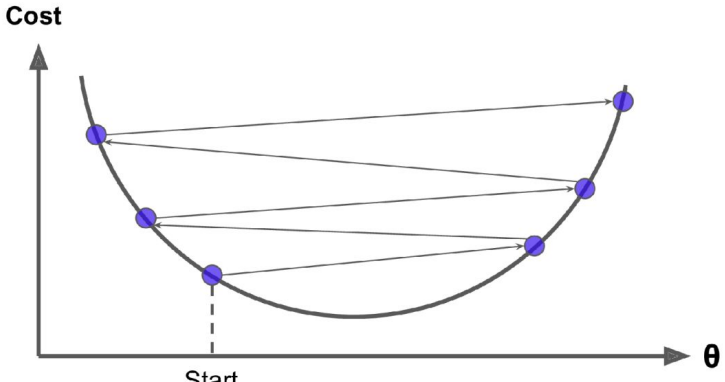
- Es un algoritmo de optimización genérico capaz de encontrar soluciones óptimas a un gran número de problemas.
- Se adapta mejor a casos con gran número de parámetros e instancias.
- La idea general es inicializar de forma aleatoria **θ** e ir modificando iterativamente los parámetros para minimizar la función de costo.



Descenso de gradiente: Learning rate

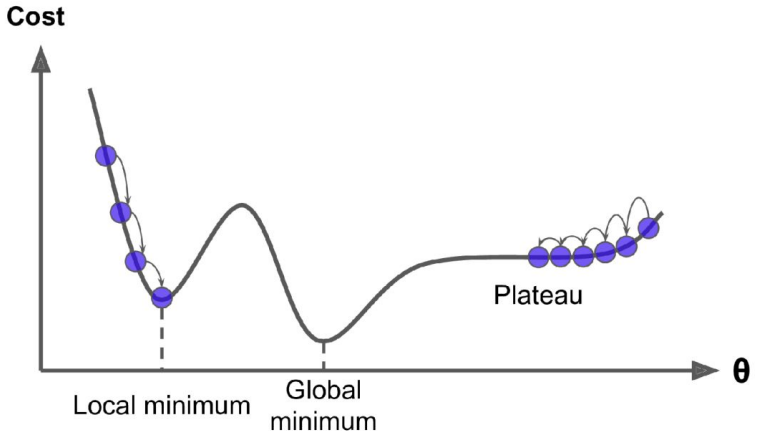
- Un hiperparámetro importante del algoritmo Descenso de gradiente es el *learning rate*.
- Un learning rate bajo va a hacer qué al algoritmo le tome mucho tiempo alcanzar el mínimo.
- En el otro extremo, un learning rate alto resulta en qué el algoritmo empiece a ir de un extremo a otro, llegando incluso a diverger en ciertos casos.





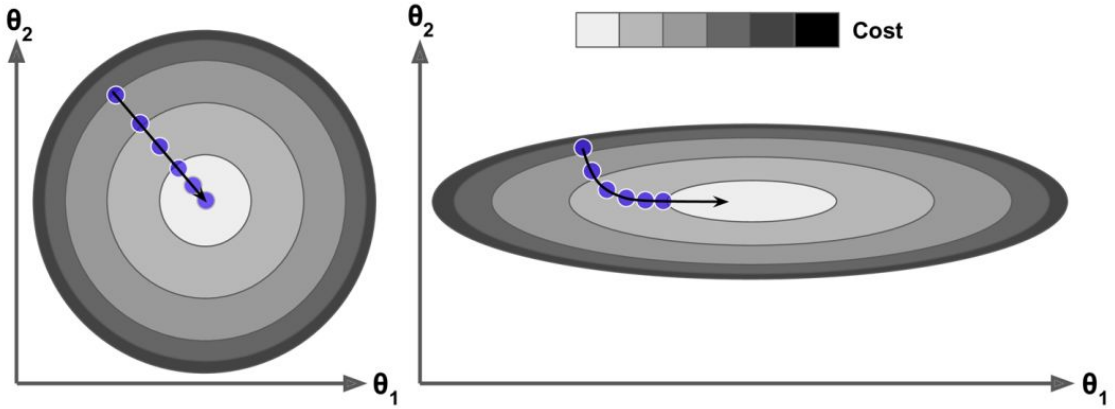
Descenso de gradiente: Forma de las funciones de costo

- La convergencia del algoritmo a un mínimo global no está garantizada. Va a depender de la inicialización.
- Podríamos encontrarnos con un mínimo local o con una meseta en la función de costo qué haría qué al algoritmo le tome mucho tiempo llegar al mínimo.
- En el caso de la función de costo MSE es convexa.



Descenso de gradiente: Funciones de costo MSE

- En el ejemplo el algoritmo de Descenso de Gradiente se dirige hacia el mínimo global.
- En caso de normalizarse la características le va a tomar más tiempo converger.



Descenso de gradiente por lote (Batch gradient descent)

- Para aplicar el Descenso de Gradiente es necesario calcular el gradiente de la función de costo con respecto a cada parámetro $\boldsymbol{\theta}_j$.
- Cada paso del algoritmo incluye los cálculos sobre el training set completo **X**.

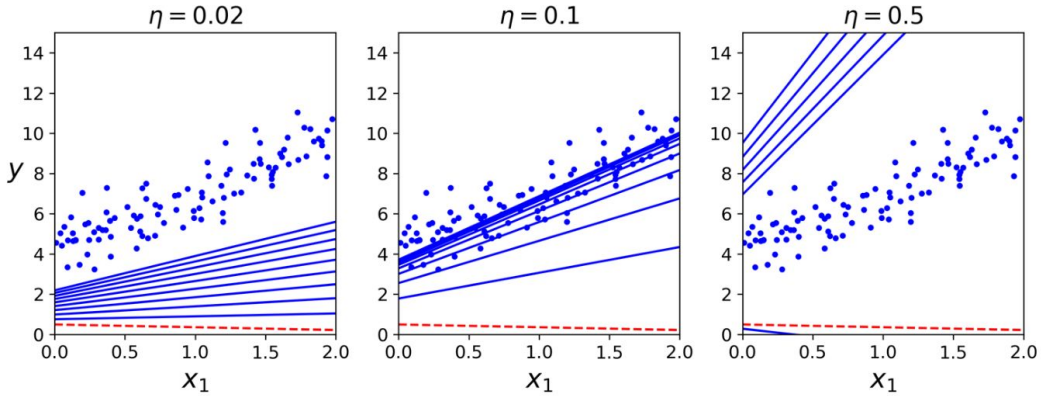
$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta})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

- Una vez computado el gradiente se lo multiplica por el learning rate η y se lo sustrae de $\boldsymbol{\theta}$.

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

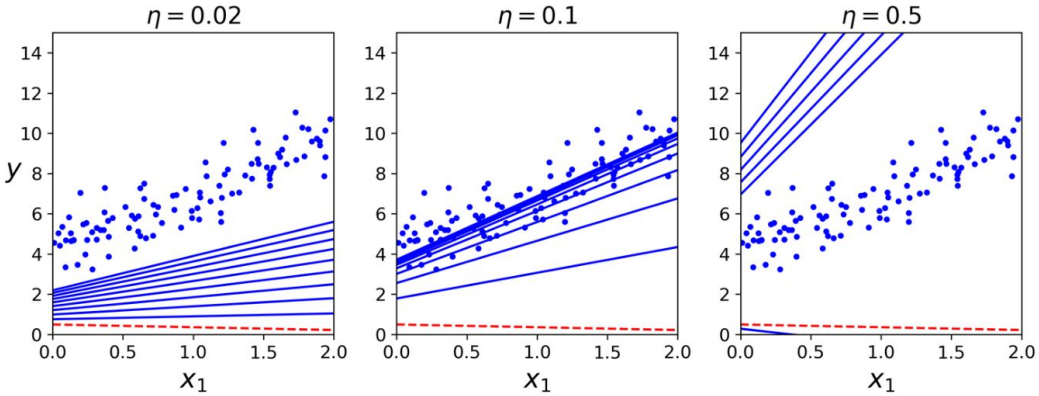
Descenso de gradiente por lote: Implementación

```
>>> eta = 0.1 # learning rate
>>> n_iterations = 1000
>>> m = 100
>>> theta = np.random.randn(2,1)
>>> # random initialization
>>> for iteration in range(n_iterations):
>>>     gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
>>>     theta = theta - eta * gradients
>>> theta
array([[4.21509616],
[2.77011339]])
```



Descenso de gradiente por lote: Implementación

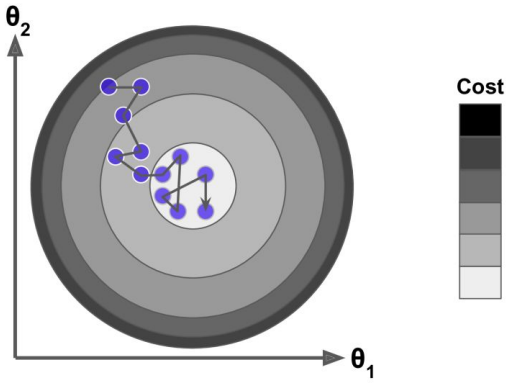
```
>>> eta = 0.1 # learning rate
>>> n_iterations = 1000
>>> m = 100
>>> theta = np.random.randn(2,1)
>>> # random initialization
>>> for iteration in range(n_iterations):
>>> gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
>>> theta = theta - eta * gradients
>>> theta
array([[4.21509616],
[2.77011339]])
```



Se puede setear una tolerancia ϵ, tal que si la norma se vuelve menor a este valor el algoritmo termine.

Descenso de Gradiente Estocástico (Stochastic Gradient Descent)

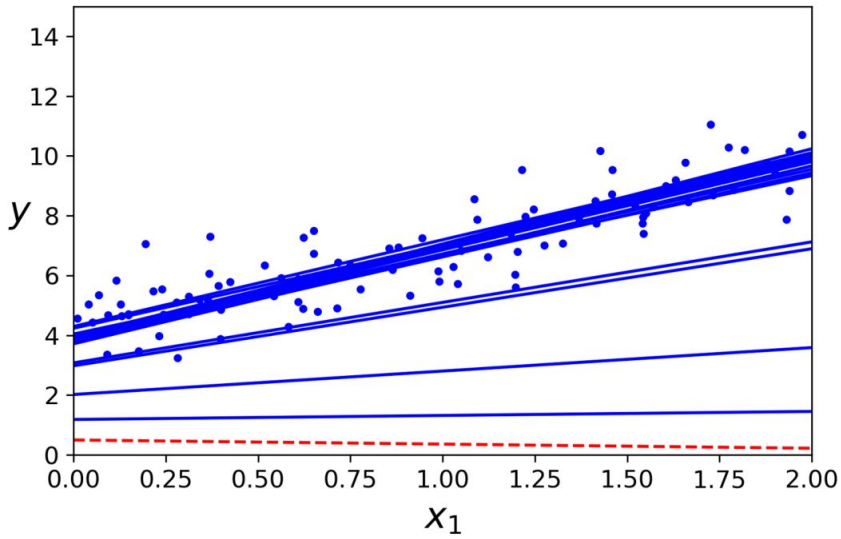
- A diferencia del descenso de gradiente por lote que incluye en los cálculos todo el training set **X**, el descenso de gradiente estocástico sólo incluye una instancia de forma aleatoria.
- A diferencia del DG por lote, este algoritmo no va a descender suavemente al mínimo.
- Cuando el algoritmo para el valor final de los parámetros es bueno, pero no óptimo.
- Si la función de costo es muy irregular, el algoritmo tiene posibilidad de escapar de mínimos locales.



- Debido a qué el algoritmo no alcanza el mínimo se puede ir reduciendo el learning rate en cada iteración con un *cronograma de apredizaje* (*learning schedule*).

Descenso de Gradiente Estocástico Implementación

```
>>> n_epochs = 50
>>> t0, t1 = 5, 50
>>> # learning schedule hyperparameters
>>> def learning_schedule(t):
>>>     return t0 / (t + t1)
>>> theta = np.random.randn(2,1)
>>> # random initialization
>>> for epoch in range(n_epochs):
>>>     for i 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.dot(xi.dot(theta) - yi)
>>>         eta = learning_schedule(epoch * m + i)
>>>         theta = theta - eta * gradients
>>> theta
array([[4.21076011],
[2.74856079]])
```



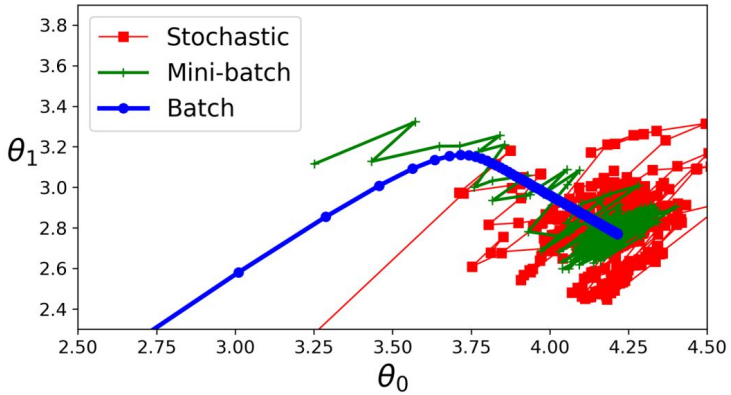
Descenso de Gradiente Estocástico en Scikit-learn

- La clase implementada es **SGDRegressor**
- Variables:
  - max_iter: cuantas iteraciones va a realizar el algoritmo.
  - tol: la tolerancia con respecto al cambio en la función de pérdida.
  - eta0: valor de learning rate a utilizar
  - learning_rate: learning rate schedule, por defecto 'inv_scaling' $\frac{\eta}{step^{\text{power\_t}}}$

```
>>> from sklearn.linear_model import SGDRegressor
>>> sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3, penalty=None, eta0=0.1)
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.intercept_, sgd_reg.coef_
(array([4.24365286]), array([2.8250878]))
```

Descenso de Gradiente en mini lote (Mini-Batch Gradiente Descent)

- Computa el gradiente en un set de instancias random llamada mini lote (o mini-batch).
- Es menos errático qué el SGD y termina más cerca del mínimo.
- Puede tener más problemas para escapar de un mínimo local.
- Al igual qué SGD necesita un parámetros de tolerancia.



Comparación entre algoritmos

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	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`