Regression Models and Gradient Descent

Douglas Chielle
douglas.chielle@ifce.edu.br

Objectives

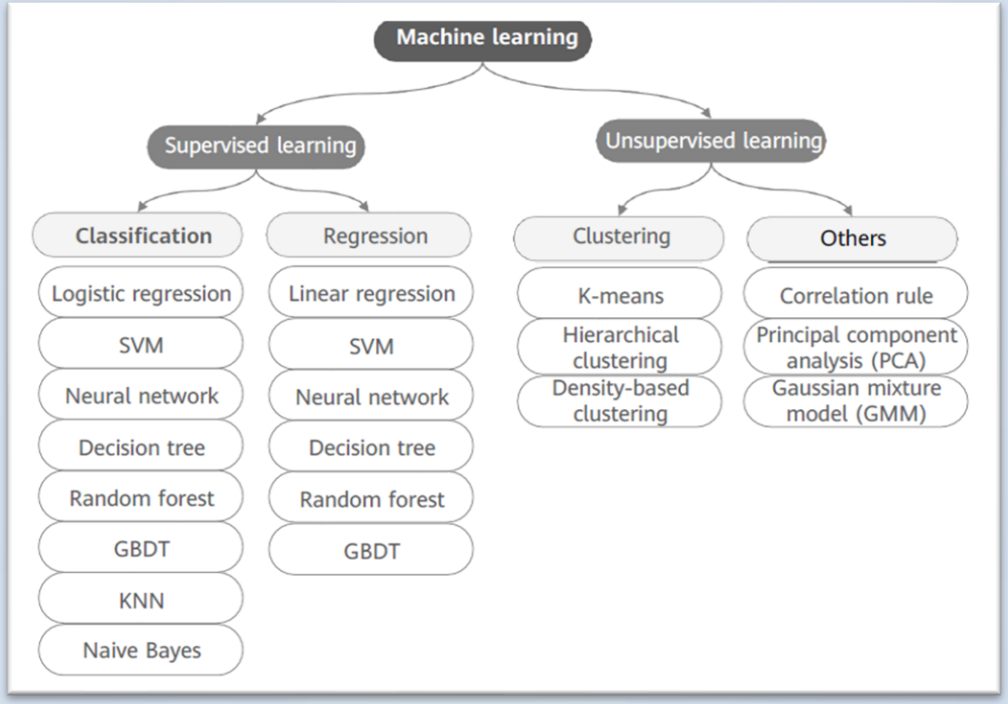
Upon completion of this lecture, you will be able to:

- Understand Linear Regression;
- Understand Gradient Descent;
- Understand Regularized Linear Models;
- Understand Polynomial Regression; and
- Build Regression models with scikit-learn.

Content

1. ML Algorithm Overview
2. Linear Regression
3. Gradient Descent
4. Regularized Linear Models
    a) Ridge Regression
    b) Lasso Regression
5. Polynomial Regression
6. Regression with scikit-learn

ML Algorithm Overview



Regression

What is Regression?

- *Regression analysis* consists of a set of machine learning methods that allow us to estimate the relationship between a dependent variable and one or more independent variables.
- It is widely used in supporting decision making, making predictions, time series modeling, etc.
- It is a supervised learning technique.

Linear Regression

Linear Regression

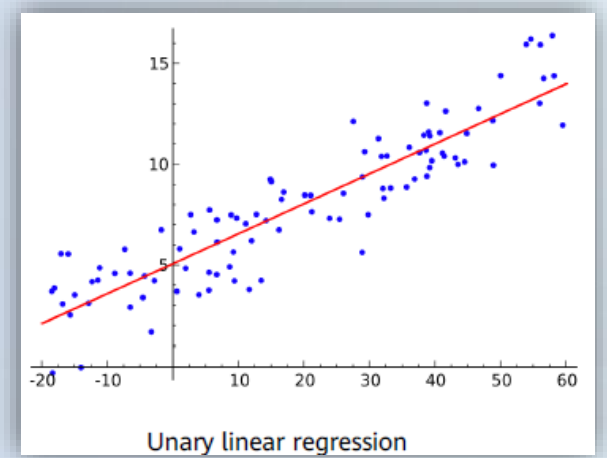
- Linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables.

$y = f(x)$

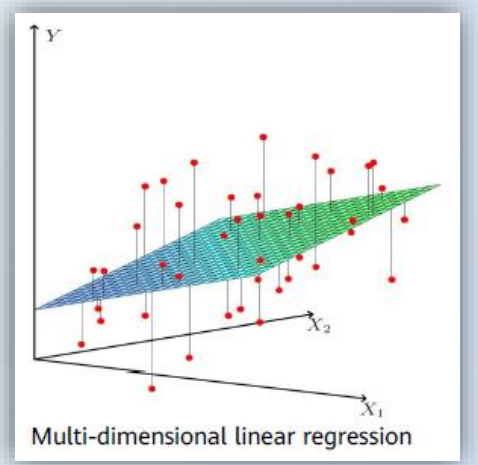
$y = ax + b$

↓

Eq. da reta



Unary linear regression



Multi-dimensional linear regression

$y = f(x_1, x_2)$

$y = ax_1 + bx_2 + c$

↓

Eq. do plano

Linear Regression

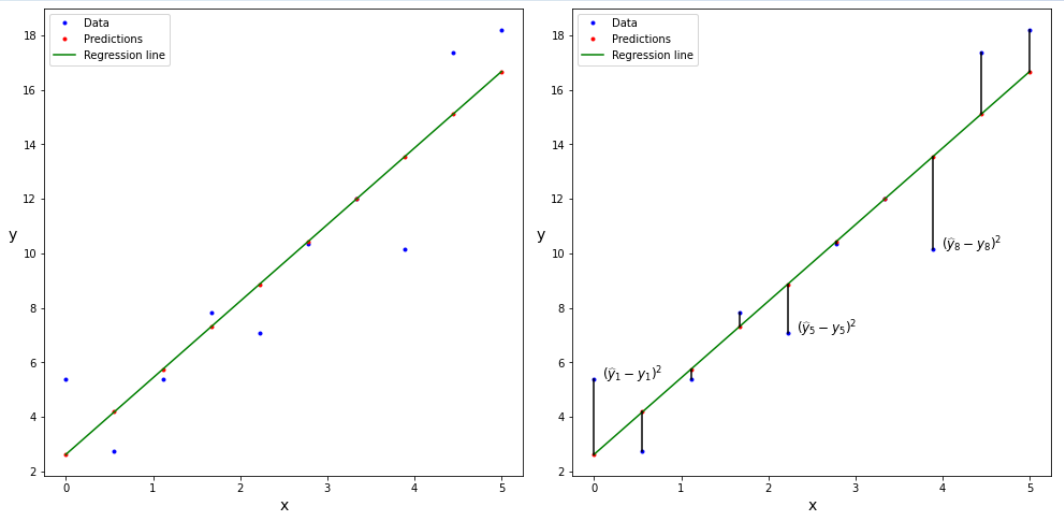
- A linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the *bias term*:

$$\hat{y} = \theta_0 \cdot 1 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \boldsymbol{\theta}^T \boldsymbol{x}$$

- In this equation:
  - $\hat{y}$ is the predicted value
  - n is the number of features
  - x_i is the i^{th} feature value
  - θ_j is the j^{th} model parameter

How to train it

- Training a model means setting its parameters so that the model best fits the training set.
- Now, what is “the best fit”?



How to train it

- To measure how well this fit is, a common performance measure for is the Mean Squared Error (MSE), which for Linear Regression is:

$$MSE(\boldsymbol{X}, h_\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\overbrace{\boldsymbol{\theta}^T\boldsymbol{x}^{(i)}}^{\hat{y}^{(i)}} - y^{(i)})^2$$

The Normal Equation

- The ***Normal Equation*** is a closed-form solution that finds the value of $\boldsymbol{\theta}$ that minimizes the cost function:

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

- In this equation
  - $\widehat{\boldsymbol{\theta}}$ is the value of $\boldsymbol{\theta}$ that minimizes the cost function.
  - $\boldsymbol{y}$ is the vector of target values containing y_1 to y_m.

Gradient Descent

Gradient Descent (GD)

Imagine you are a mountain climber on top of a mountain, and night has fallen. You need to get to your base camp at the bottom of the mountain, but in the darkness with only your dinky flashlight, you can't see more than a few feet of the ground in front of you. So how do you get down?

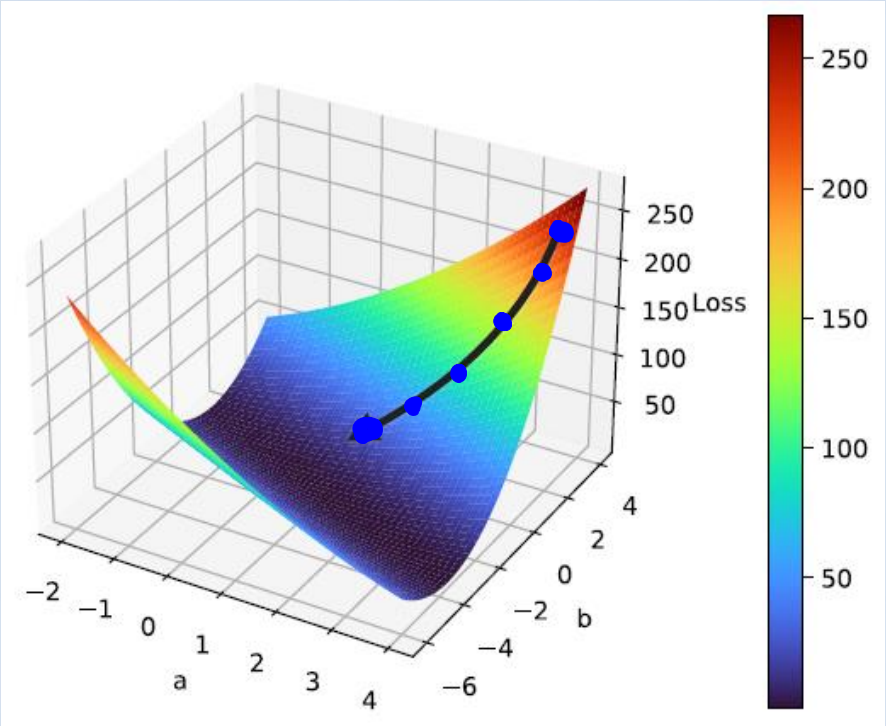
One strategy is to look in every direction to see which way the ground steeps downward the most, and then step forward in that direction. Repeat this process many times, and you will gradually go farther and farther downhill. You may sometimes get stuck in a small trough or valley; in which case you can follow your momentum for a bit longer to get out of it.

Caveats aside, this strategy will eventually get you to the bottom of the mountain.

Gradient Descent

- Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. Intuitively, it works similarly to the mountain climber analogy.
- First, we start with a random guess at the parameters. We then figure out which direction the loss function steeps downward the most and step slightly in that direction.
- We repeat this process until we are satisfied, or we have found the lowest point.
- To figure out which direction the loss steeps downward the most, one must calculate the loss function's gradient concerning all the parameters.
- A gradient is a multidimensional generalization of a derivative; it is a vector containing each of the partial derivatives of each variable. In other words, it is a vector that contains the slope of the loss function along every axis.

Gradient Descent



https://ml4a.github.io/ml4a/how_neural_networks_are_trained/

Gradient Descent

- To implement GD, we need to compute the gradient of the cost function w.r.t. each model parameter. For the MSE loss function, we have:

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

- We can also compute all these partial derivatives in one go:

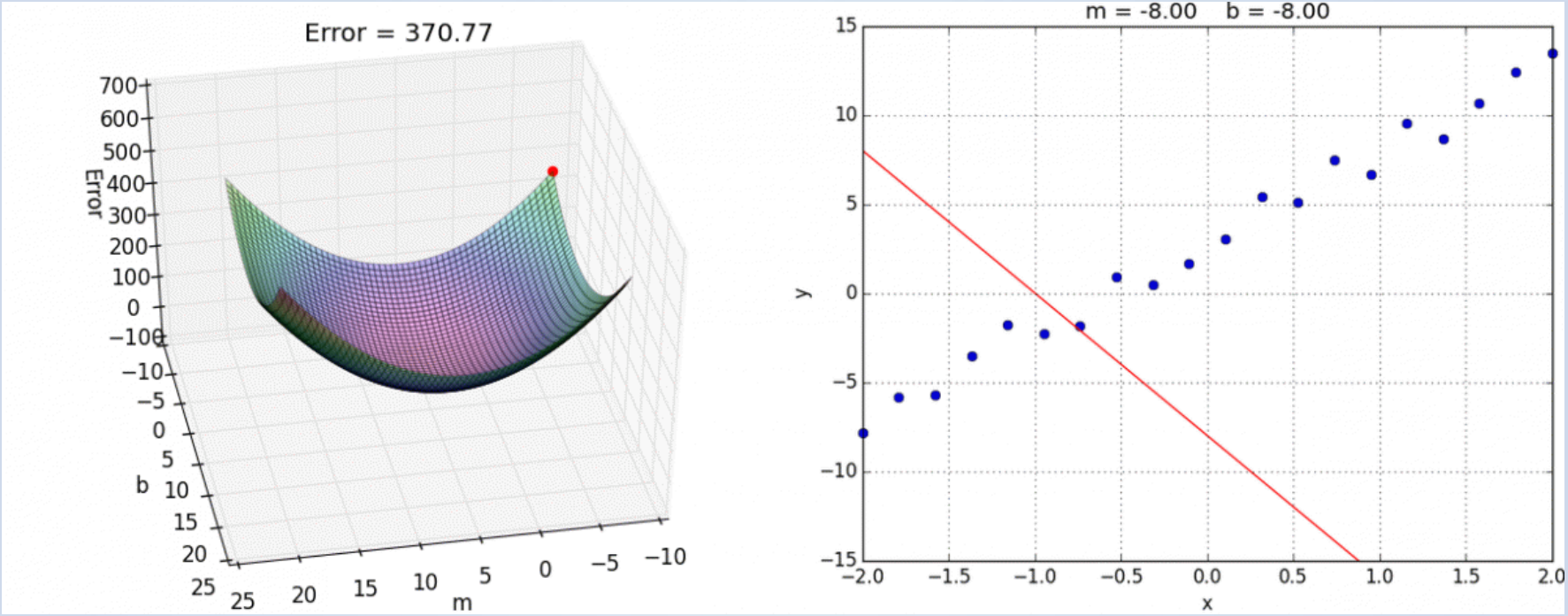
$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} MSE(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} MSE(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} MSE(\boldsymbol{\theta}) \end{pmatrix} = \frac{1}{m} \boldsymbol{X}^T (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})$$

Gradient Descent

- Now that we know the direction we need to step to, we must ask: how far in that direction should we step?
- Such a choice turns out to be an important one, and in ordinary gradient descent, this is a hyperparameter, known as the **learning rate**, and is often denoted as η.
- If η is set too low, it may take an unacceptably long time to get to the bottom. If η is too high, we may overshoot the correct path or even climb upwards.
- Now, we can write the update steps for each parameter as follows:

$$\theta_j^{t+1} = \theta_j^t - \eta \frac{\partial MSE}{\partial \theta_j}$$

Gradient Descent in Action!



https://gifer.com/en/NqNM

Gradient Descent variants

- Now let’s discuss the three variants of gradient descent algorithm.
- The main difference between them is the amount of data we use when computing the gradients for each learning step.
- The trade-off between them is the accuracy of the gradient versus the time complexity to perform each parameter’s update (learning step).

https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Batch GD

THE MAIN ADVANTAGES:

- We can use fixed learning rate during training without worrying about learning rate decay.
- It has straight trajectory towards the minimum and it is guaranteed to converge, in theory, to the global minimum if the loss function is convex and to a local minimum if the loss function is not convex.

PYTHON: BATCH GRADIENT DESCENT

```
for i in range(num_epochs):
  grad = compute_gradient(data, params)
  params = params - eta * grad
```

THE MAIN DISADVANTAGES:

- Even though we can use vectorized implementation, it may still be slow to go over all examples especially when we have large datasets.
- Each step of learning happens after going over all examples where some examples may be redundant and don't contribute much to the update.

https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Mini-batch GD

THE MAIN ADVANTAGES:

- Faster than Batch version because it goes through a lot less examples than Batch (all examples).
- Randomly selecting examples will help avoid redundant examples or examples that are very similar that don't contribute much to the learning.
- With $batch\ size < size$ of training set, it adds noise to the learning process, which helps improving generalization error.

PYTHON: MINI-BATCH GRADIENT DESCENT

```
for i in range(num_epochs):
  np.random.shuffle(data)
  for batch in random_minibatches(data, batch_size=32):
    grad = compute_gradient(batch, params)
    params = params - learning_rate * grad
```

THE MAIN DISADVANTAGES:

- It won't converge. On each iteration, the learning step may go back and forth due to the noise. Therefore, it wanders around the minimum region but never converges.
- Due to the noise, the learning steps have more oscillations and requires adding learning-decay to decrease the learning rate as we become closer to the minimum.

https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Stochastic GD

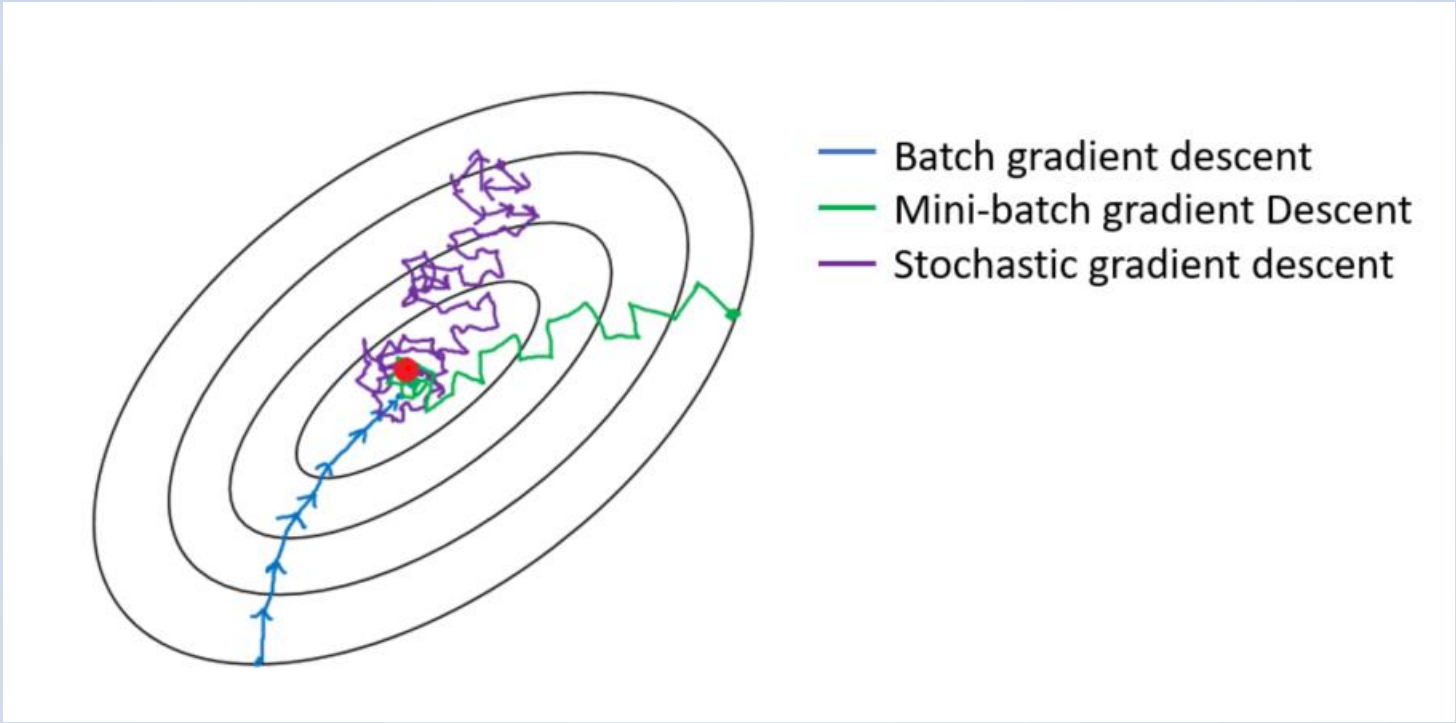
- Stochastic-GD picks a random instance in the training set at every step and computes the gradients based only on that instance
- This makes the algorithm much faster than Batch-GD.
- However, due to its stochastic nature, it is much less regular than Batch-GD.
- It shares most of the advantages and the disadvantages with mini-batch version

PYTHON: STOCHASTIC GRADIENT DESCENT

```
for i in range(num_epochs):
  np.random.shuffle(data)
  for example in data:
    grad = compute_gradient(example, params)
    params = params – learning_rate * grad
```


https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Stochastic GD

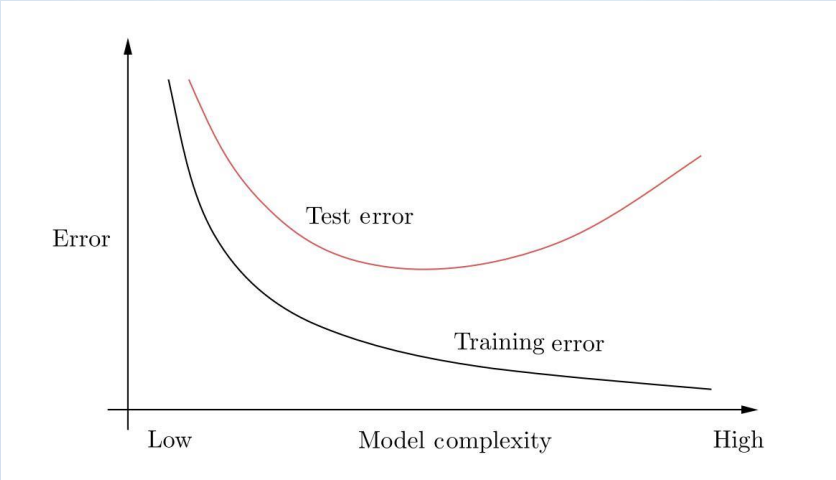


https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Regularized Linear Models

Regularized Linear Models

- Regularization is a technique for reducing the complexity of a model.
- For a linear model, regularization is typically achieved by constraining the model's parameters.



Ridge Regression

- In Ridge Regression a regularization term is added to the cost function of Linear Regression:

$$J(\boldsymbol{\theta}) = MSE(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} \theta_i^2 = MSE(\boldsymbol{\theta}) + \alpha \|\boldsymbol{\theta}\|_2^2$$

norma l_2

- Characteristics:
  - There is a closed-form equation for Ridge Regression.
  - For GD we need to add $2\alpha\theta$ to the MSE gradient vector.

Lasso Regression

- For Lasso Regression we also add a regularization term cost function of Linear Regression:

$$J(\boldsymbol{\theta}) = MSE(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} |\theta_i| = MSE(\boldsymbol{\theta}) + \alpha \|\boldsymbol{\theta}\|_1$$

norma l_1

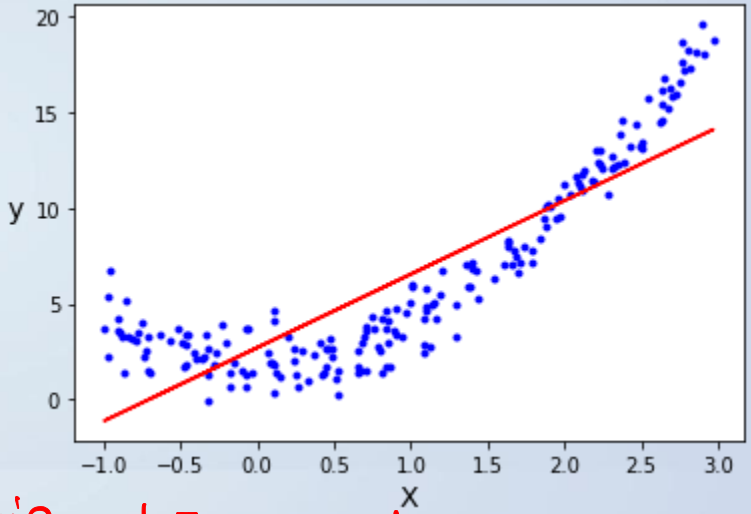
- Chatacteristics:
  - Tends to eliminate the weights of the least important features.
  - Cost function is non differentiable at $\theta_i = 0$, but we can still use GD.

Polynomial Regression

Polynomial Regression

- What if a straight line does not properly fit the data?
- No problem, we can still use a linear model to fit the nonlinear data!
- To do this, we can add powers of each feature as new features.
- For example, if x is a scalar we might propose a quadratic model of the form:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2$$



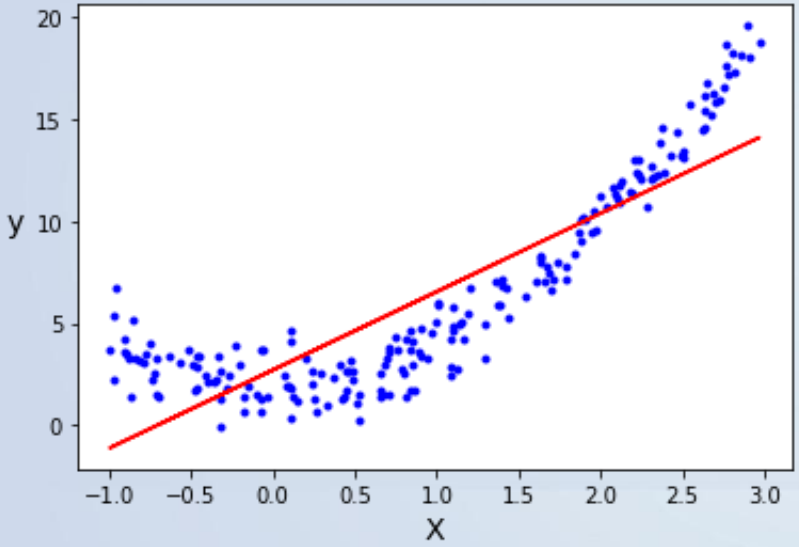
Dados originais

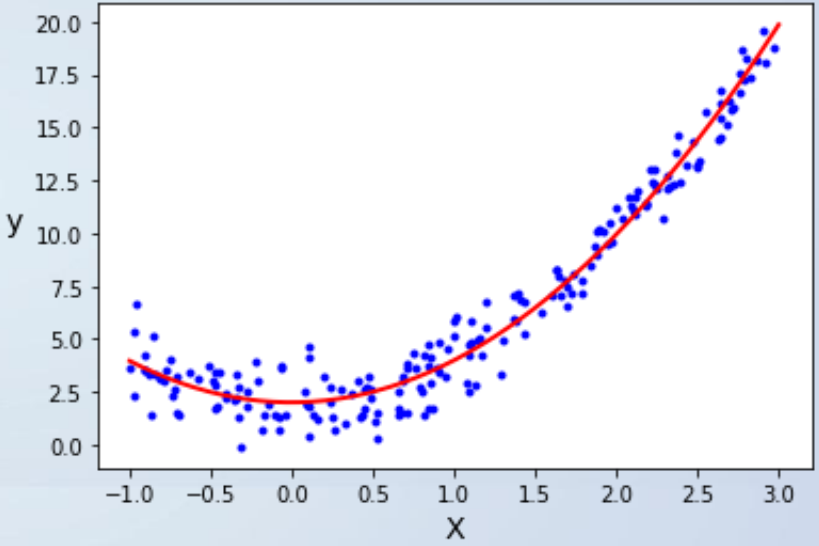
$[x_1, y_1]$
$[x_2, y_2]$
⋮
$[x_m, y_m]$

Transformados

$[x_1, x_1^2, y_1]$
$[x_2, x_2^2, y_2]$
...
$[x_m, x_m^2, y_m]$

Polynomial Regression





Regression with scikit-learn

Regression with scikit-learn

- Linear regression:
  - closed-form: sklearn.linear_model.LinearRegression
  - GD: sklearn.linear_model.SGDRegressor
- Polynomial regression
  - Use sklearn.preprocessing.PolynomialFeatures to transform the training data, then perform Linear Regression
- Ridge Regression
  - closed-form: sklearn.linear_model.Ridge, $solver = 'cholesky'$
  - GD: SGDRegressor with $penalty = 'l2'$
- Lasso Regression
  - sklearn.linear_model.Lasso
  - SGDRegressor with $penalty = 'l1'$

References

1. Géron, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.
2. https://ml4a.github.io/ml4a/how_neural_networks_are_trained/
3. https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3


That's all Folks!