哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Big data technology and Practice

李春山

2020年9月30日

内容提要

Chapter 11 Training Models

Chapter 11 Training Models

- How to Train models?
- Different ways lead to very different cost and performance.
- By looking at the Linear Regression model, one of the simplest models, to discuss this topic.

1. Linear regression using the Normal Equation

- **Linear Regression model prediction**

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

- $\hat{y}$ 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 (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

Linear Regression model prediction (vectorized form

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

- θ is the model's *parameter vector*, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- θ^T is the transpose of θ (a row vector instead of a column vector).
- $\mathbf{x}$ is the instance's *feature vector*, containing x_0 to x_n, with x_0 always equal to 1.
- $\theta^T \cdot \mathbf{x}$ is the dot product of θ^T and $\mathbf{x}$.
- h_θ is the hypothesis function, using the model parameters θ.

Linear Regression model prediction (vectorized form

- **To find the value of θ that minimizes the cost function, there is a mathematical equation that gives the result directly. This is called the Normal Equation**

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

  - $\hat{\theta}$ is the value of θ that minimizes the cost function.
  - $\mathbf{y}$ is the vector of target values containing $y^{(1)}$ to $y^{(m)}$.

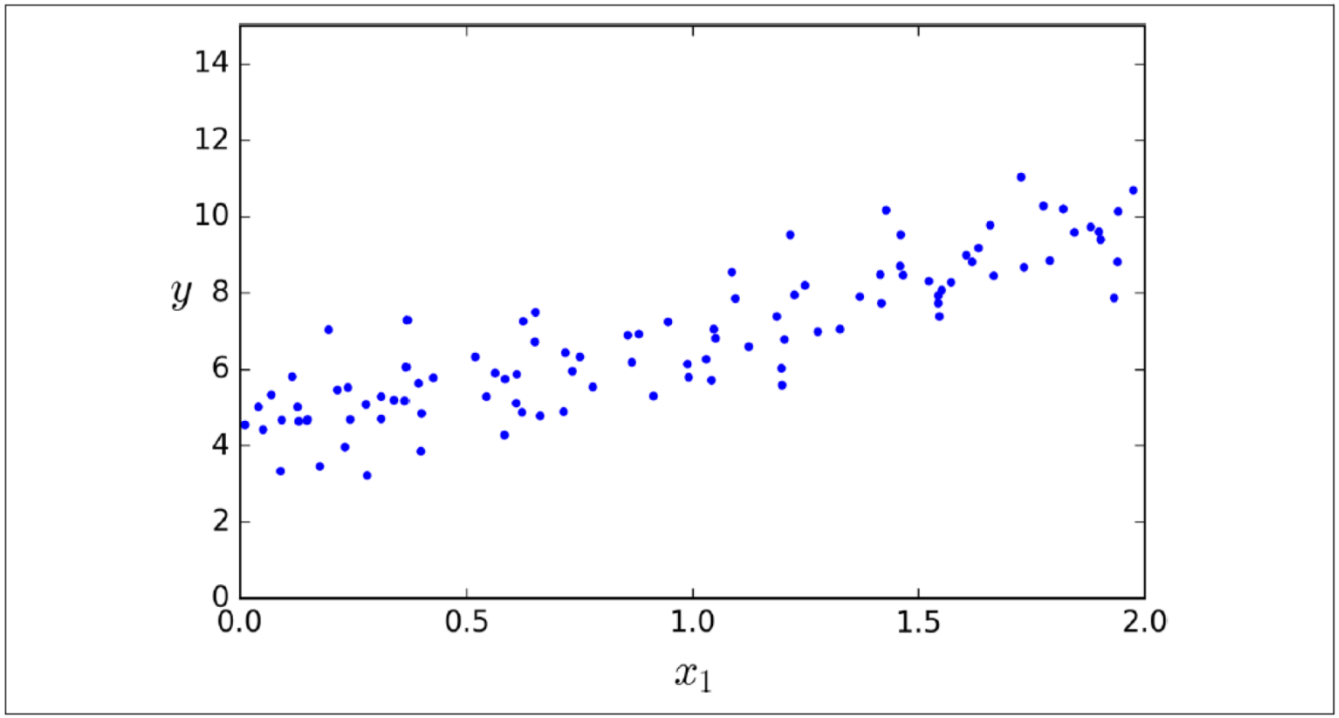
- MSE cost function for a Linear Regression model

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

Linear Regression model prediction (vectorized form

- **Let's generate some linear-looking data to test this equation**

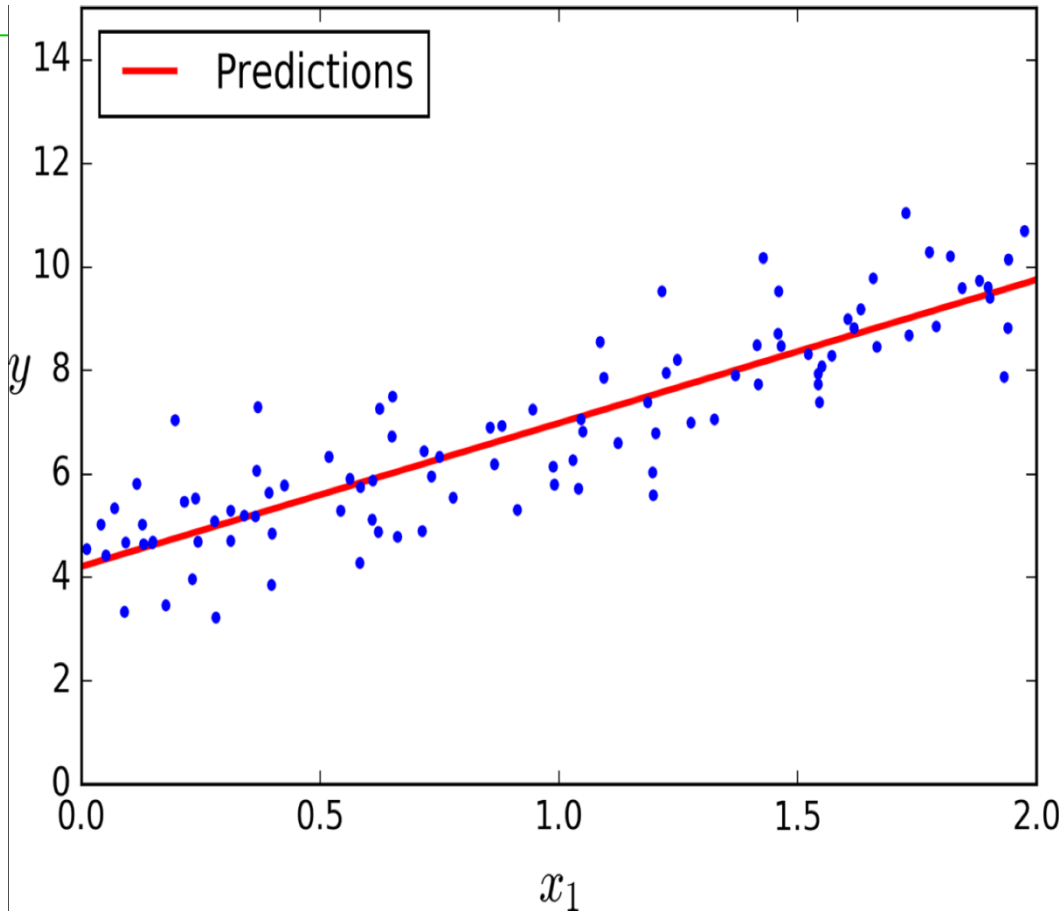
```
import numpy as np

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



- Randomly generated linear dataset, as shown as above.

Computational Complexity:

- The Normal Equation computes the inverse of XT . X, which is an (n x n) matrix). The computational complexity of inverting such a matrix is typically about O(n^2.4) to O(n^3) (depending on the implementation).
- In other words, if you double the number of features, you multiply the computation time by roughly 2^2.4 = 5.3 to 2^3 = 8.



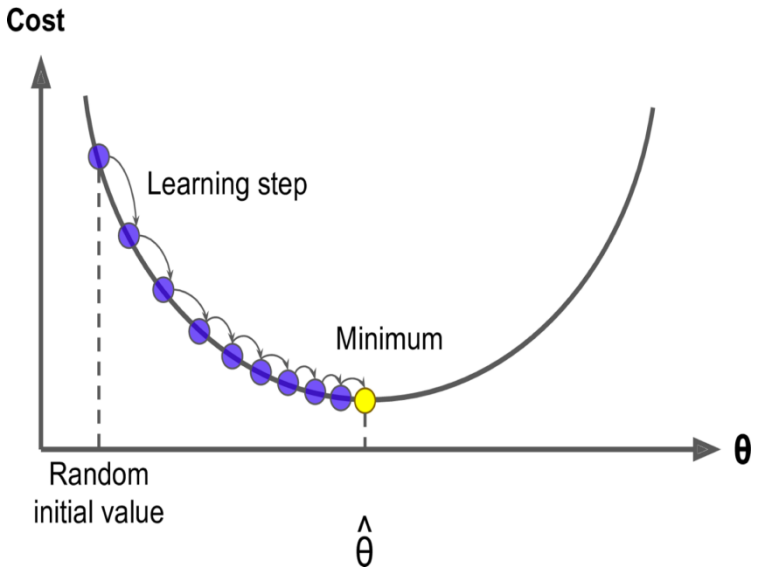
The Normal Equation gets very slow when the number of features grows large (e.g., 100,000)

2. Linear regression using batch gradient descent

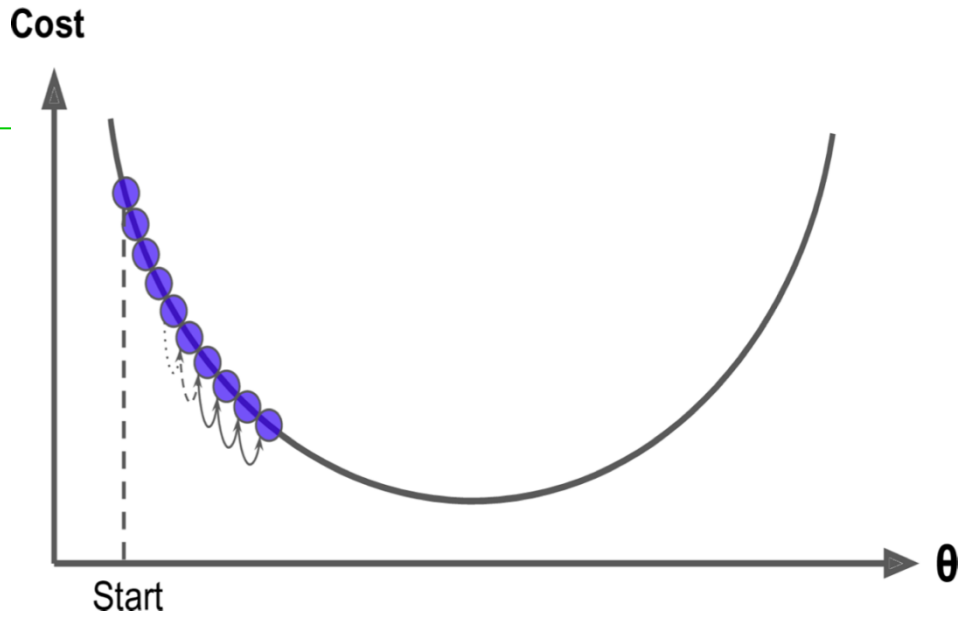
- **1). Gradient Descent**
  - It is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
  - The idea is to tweak parameters iteratively in order to minimize a cost function.
  - The gradient points in the direction of the greatest rate of increase of a function, and its magnitude is the slope of the graph of the function in that direction.
  - Like the derivative, the gradient represents the slope of the tangent of the graph of the function.

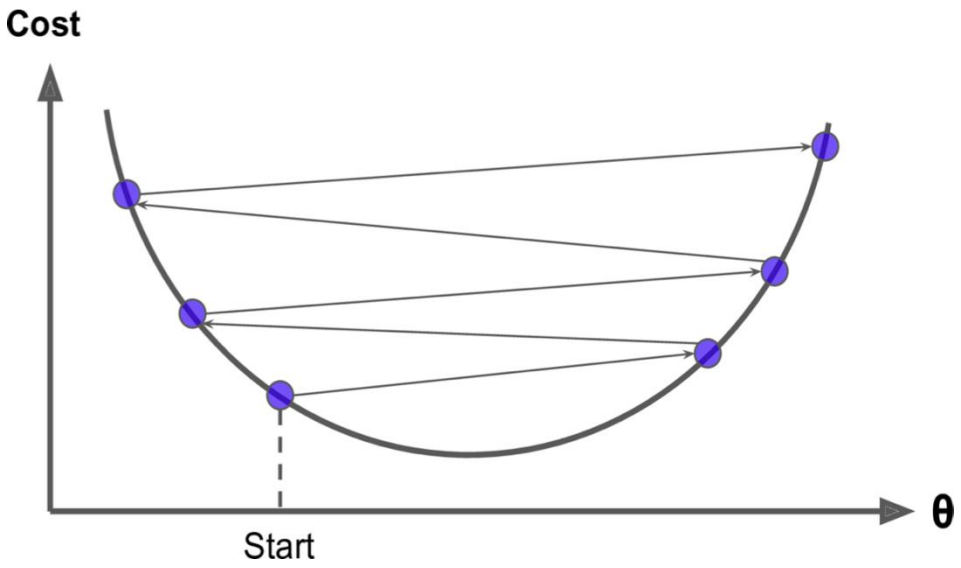
2. Linear regression using batch gradient descent

- **1). Gradient Descent**
  - It measures the local gradient of the error function (or cost function) with regards to the parameter vector θ, and it goes in the direction of descending gradient.
  - Start by filling θ with random values, "random initialization"
  - Improve the cost gradually, taking one by step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum.



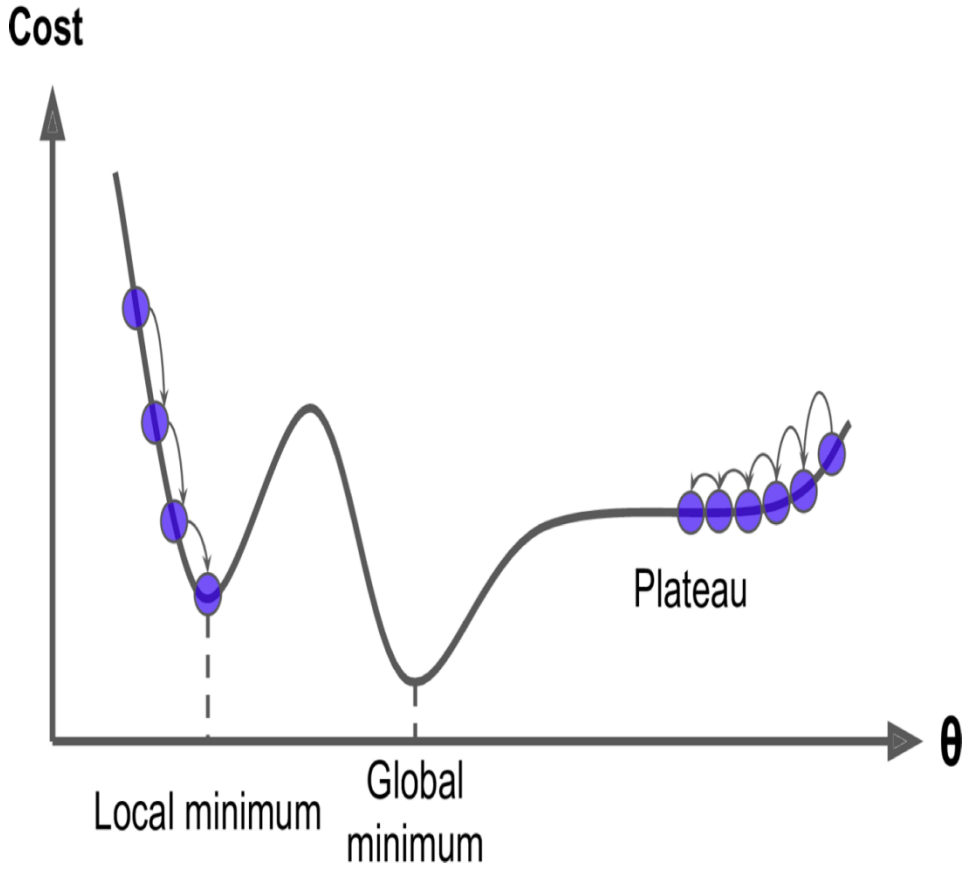
- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate.
- If the learning rate is too small, the algorithm will have to go through many iterations (or a long time) to converge.
- If the learning rate is too large, you might jump across the valley and end up on the other side, possibly even higher up than you were before





2. Linear regression using batch gradient descent

- The two main challenges with Gradient Descent:
- 1) if the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum.
- 2) If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum.



2. Linear regression using batch gradient descent

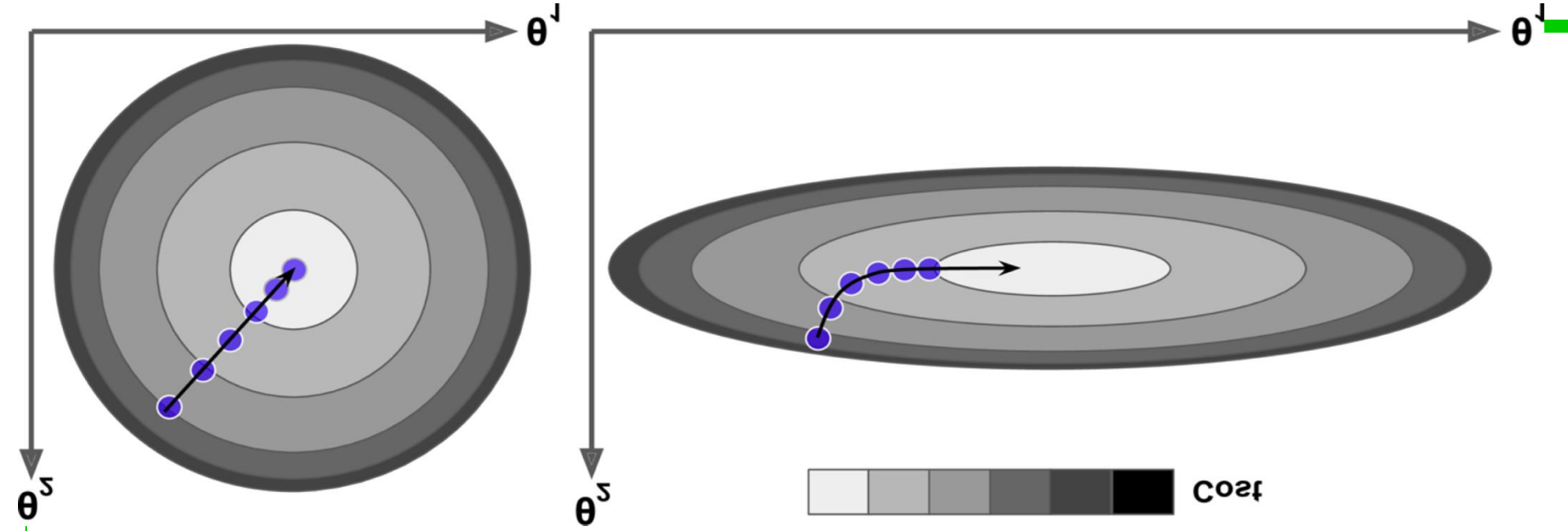
- Fortunately, the MSE cost function for a Linear Regression model happens to be a convex function, which means that if you pick any two points on the curve, the line segment joining them never crosses the curve.

- This implies that there are no local minima, just one global minimum. It is also a continuous function with a slope that never changes abruptly.

2. Linear regression using batch gradient descent

- These two facts have a great consequence:

- Gradient Descent is guaranteed to approach arbitrarily close the global minimum (if you wait long enough and if the learning rate is not too high).
- In fact, the cost function has the shape of a bowl, but it can be an elongated bowl if the features have very different scales.



- It shows Gradient Descent on a training set where features 1 and 2 have the same scale (on the left), and on a training set where feature 1 has much smaller values than feature 2 (on the right).
- By scaling, it reaches convergence quickly, otherwise, it takes a long time.
- *When using Gradient Descent, you should ensure that all features have a similar scale (e.g., using Scikit-Learn's StandardScaler class), or else it will take much longer to converge.

2). Batch Gradient Descent

- To implement Gradient Descent, you need to compute the gradient of the cost function with regards to each model parameter θj.
- In other words, you need to calculate how much the cost function will change if you change θj just a little bit.
- This is called a partial derivative.

Gradient vector of the cost function

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

2). Batch Gradient Descent

- Notice that this formula involves calculations over the full training set X, at each Gradient Descent step!
- As a result it is usually slow on
- However, training a Linear Regression model when there are hundreds of thousands of features is much faster using Gradient Descent than using the Normal Equation.

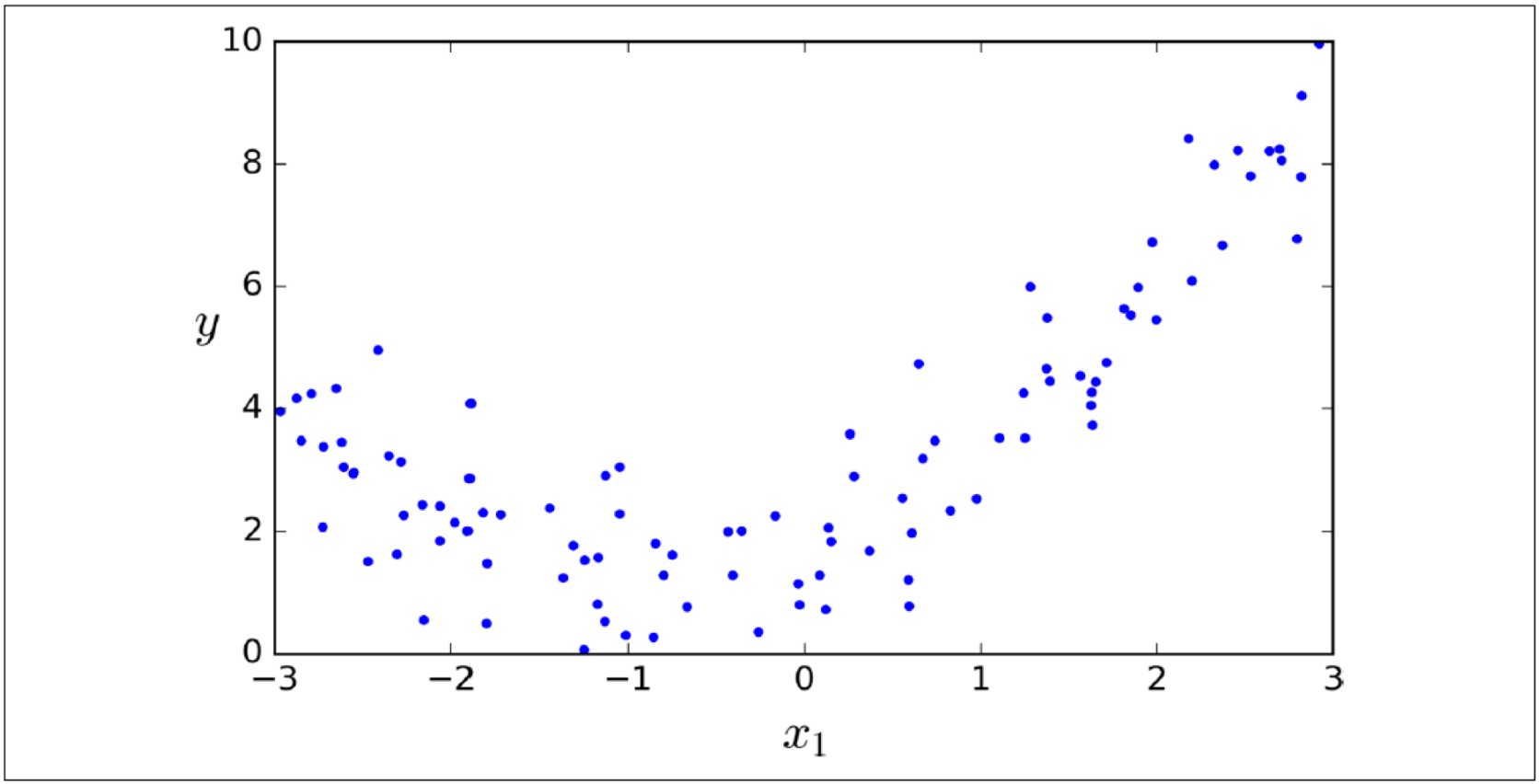
Gradient vector of the cost function

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

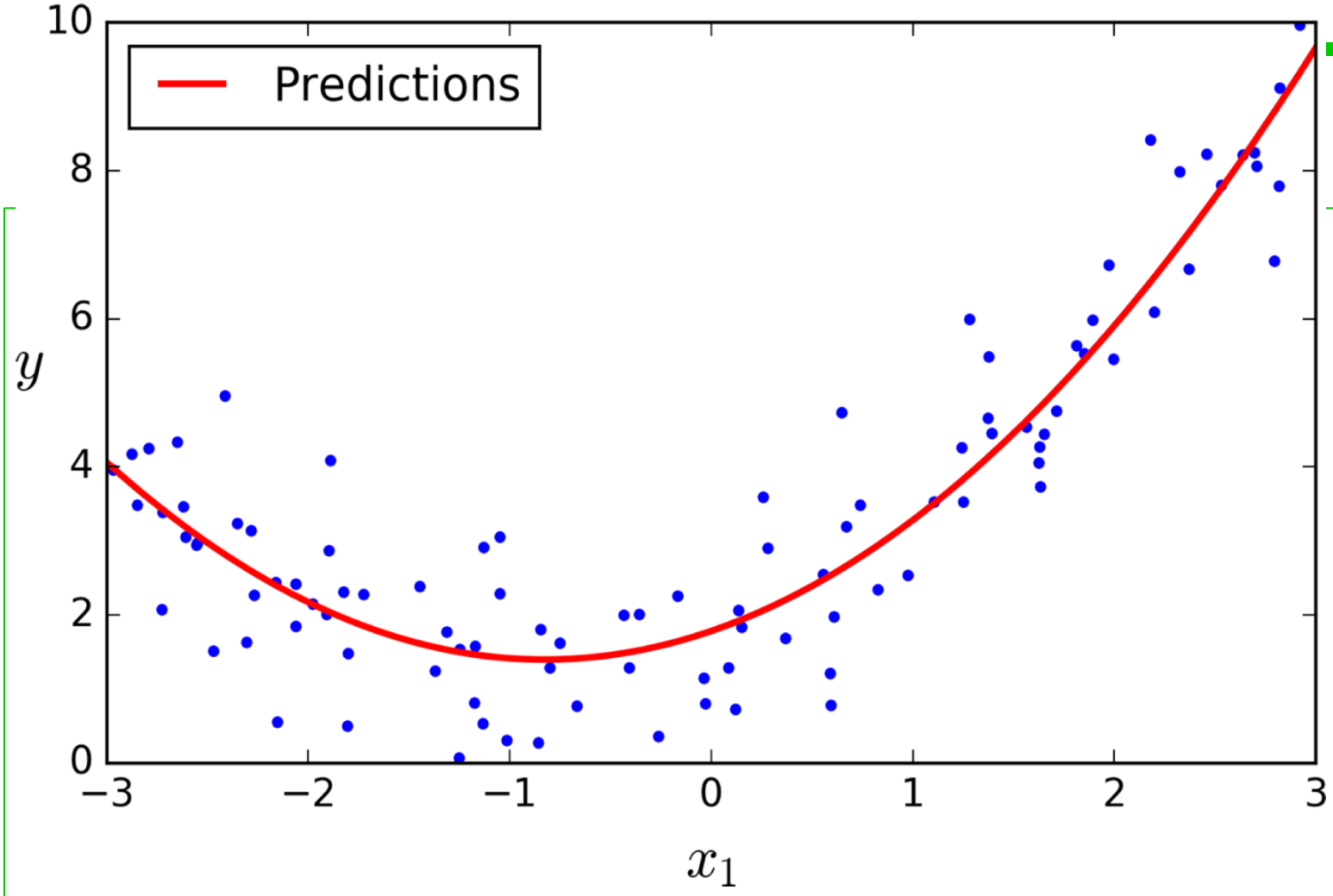
3. Polynomial Regression

- If data is more complex than a simple straight line, surprisingly, a linear model can be used to fit nonlinear data.
- How? simply add powers of each feature as new features, then train a linear model on this extended set of features.
- This technique is called Polynomial Regression
- Let's look at an example.
- 1). let's generate some nonlinear data, based on a simple quadratic equation plus some noise.

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



- Clearly, a straight line will never fit this data properly.
- So, use Scikit-Learn's PolynomialFeatures class to transform the training data, adding the square (2nd-degree polynomial) of each feature in the training set as new features :



Polynomial Regression model predictions

- The model estimates y = 0 . 56x1^2 + 0 . 93x1 + 1 . 78 when in fact the original function was y = 0 . 5x1^2 + 1 . 0x1 + 2 . 0 + Gaussian noise. (not bad!)

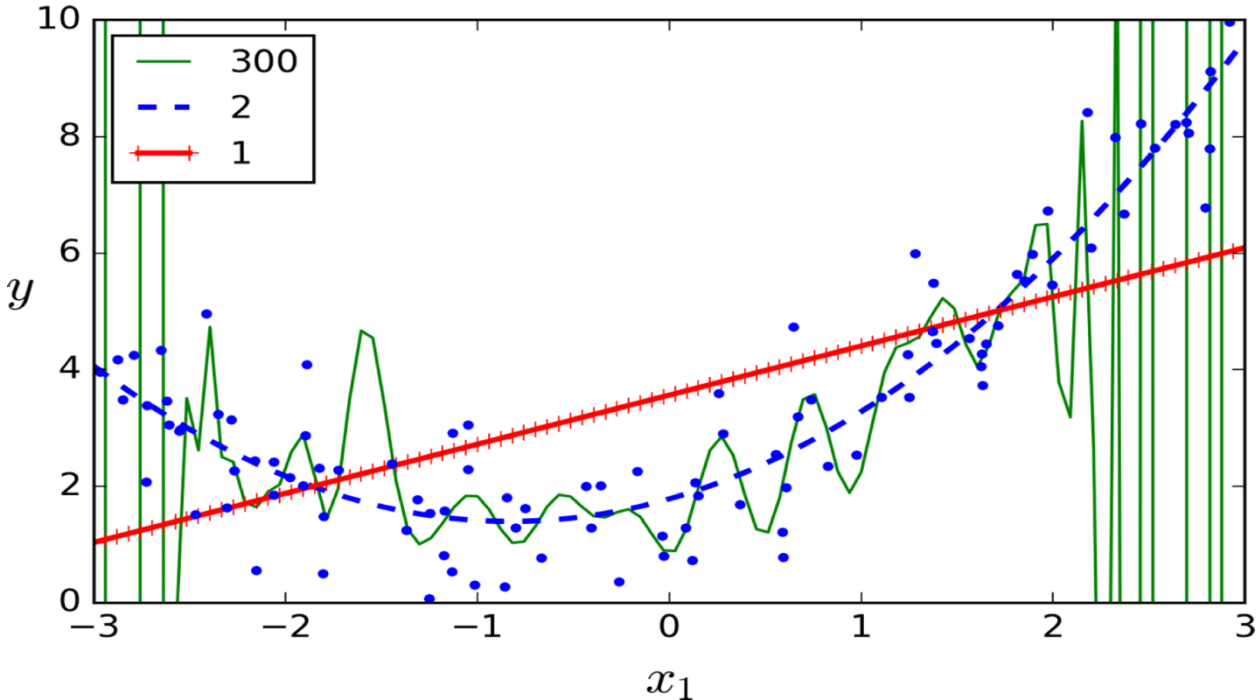
Polynomial Regression

- Note: when there are multiple features, Polynomial Regression is capable of finding relationships between features, which is something a plain Linear Regression model cannot do.
- This is made possible by adding all combinations of features up to the given degree.
- For example, if there were two features a and b, PolynomialFeatures with degree=3 would not only add the features a2, a3, b2, and b3, but also the combinations ab, a2b, and ab2

`PolynomialFeatures(degree=d)` transforms an array containing n features into an array containing $\frac{(n+d)!}{d!\,n!}$ 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!

4. Cross Validation and Learning Curves

- If you perform high-degree Polynomial Regression, you will likely fit the training data much better than with plain Linear Regression.
- Notice: the 300-degree polynomial model wiggles around to get as close as possible to the training instances, which is "overfitting", while the linear model is "underfitting". A quadratic model (2nd-degree polynomial) is better..

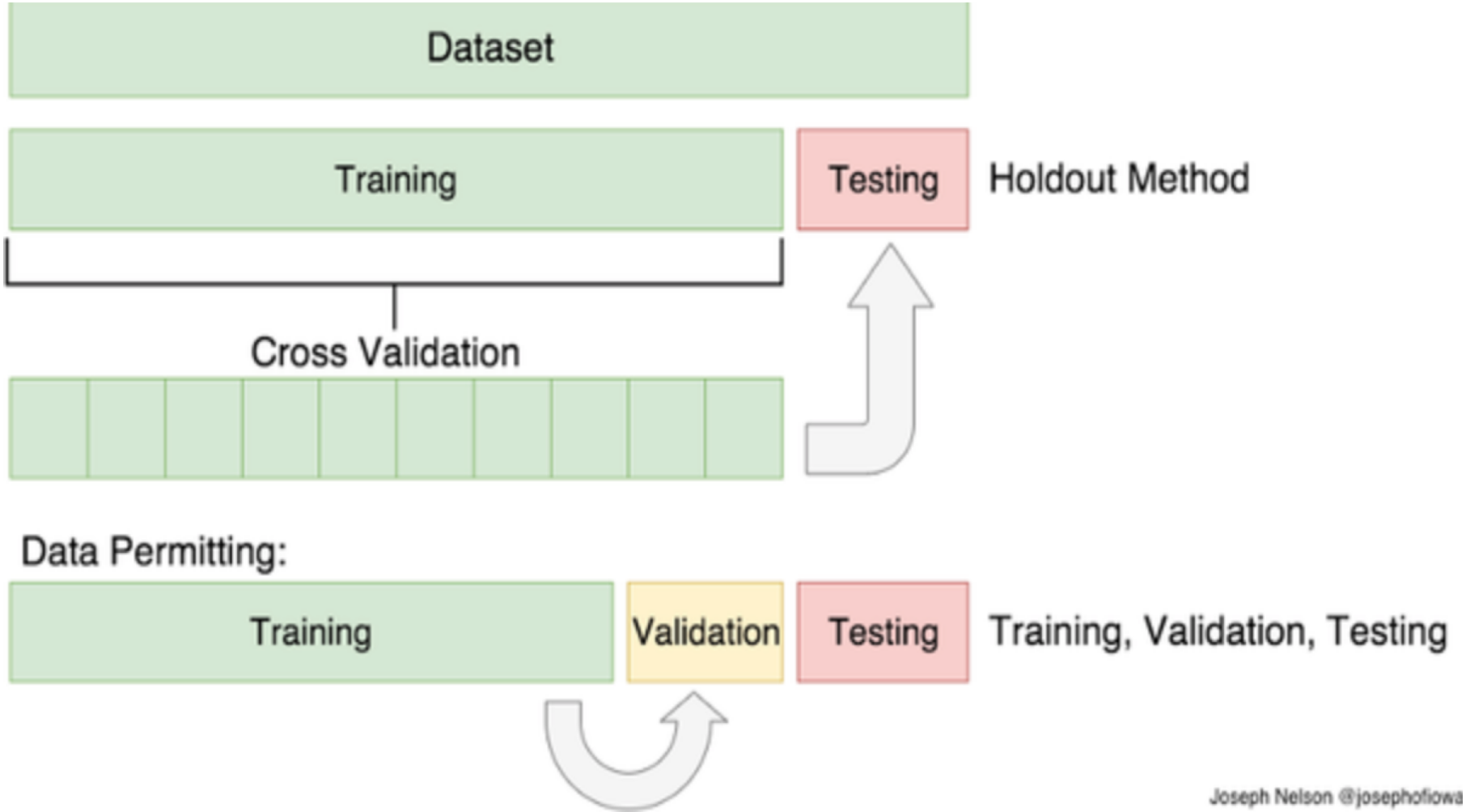


How can you tell that your model is overfitting or underfitting the data?

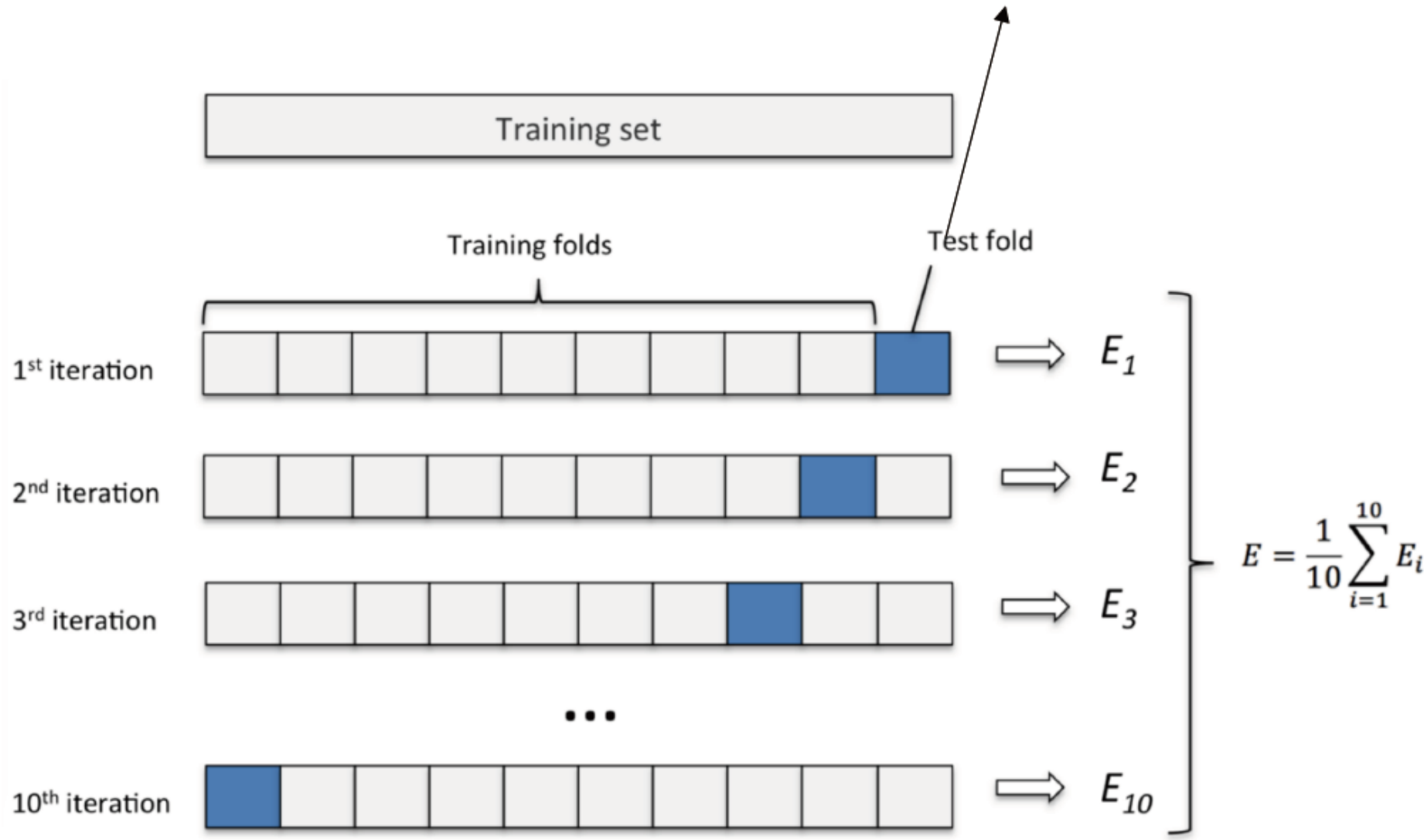
- **1). Cross-Validation**
  - For example of using "Cross Validation", to evaluate the Decision Tree model, we can randomly split the training set into a smaller training set and a validation set, then train your models against the smaller trainin set and evaluate them against the validation set.
  - To estimate a model's generalization performance. If a model performs well on the training data but generalizes poorly, according to the cross-validation metrics, then your model is overfitting.
  - If it performs poorly on both, then it is underfitting. This is one way to tell when a model is too simple or too complex.

Cross-Validation

For example, K-fold cross-validation



Cross-Validation



Randomly split the training set into a smaller training set and a validation set

Cross Validation

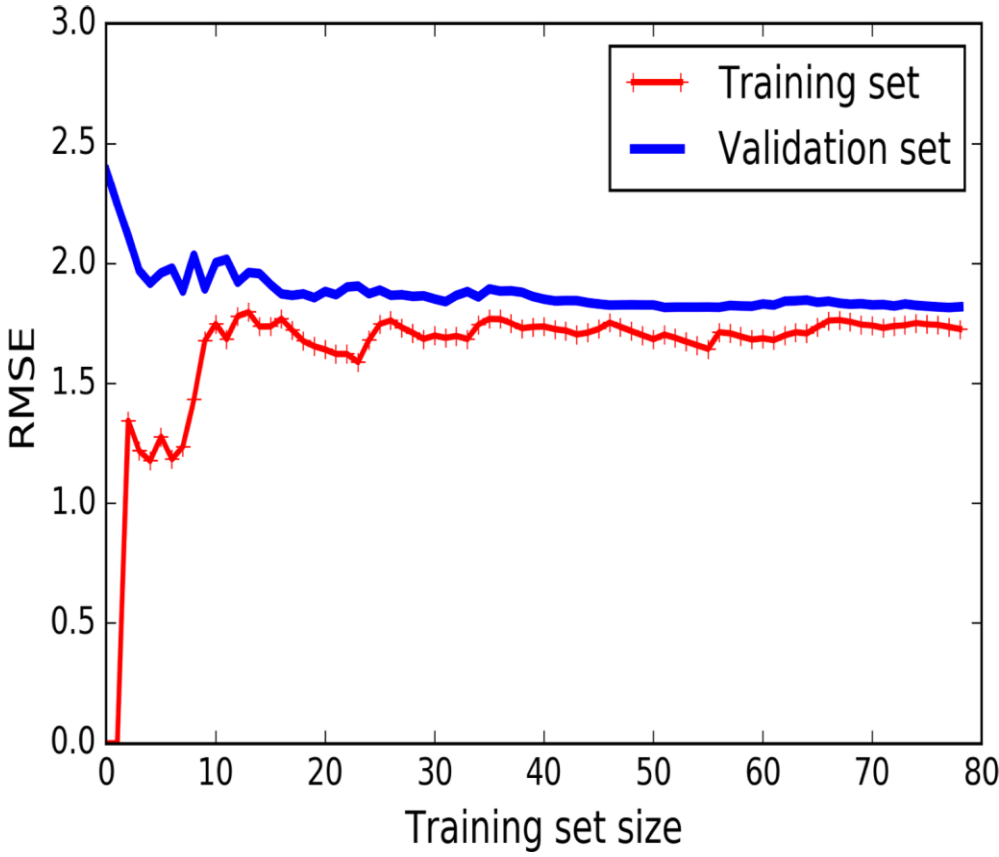
- We can use Scikit-Learn's cross-validation feature.
- The following code performs K-fold cross-validation:
- it randomly splits the training set into 10 distinct subsets called folds, then it trains and evaluates the Decision Tree model 10 times, picking a different fold for evaluation every time and training on the other 9 folds
- The result is an array containing the 10 evaluation scores:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
rmse_scores = np.sqrt(-scores)
```

2). Learning Curves

- learning curves are plots of the model's performance on the training set and the validation set as a function of the training set size.
- To generate the plots, simply train the model several times on different sized subsets of the training set.
- For example: (see more details in lab class).

a). The learning curves of the plain Linear Regression model

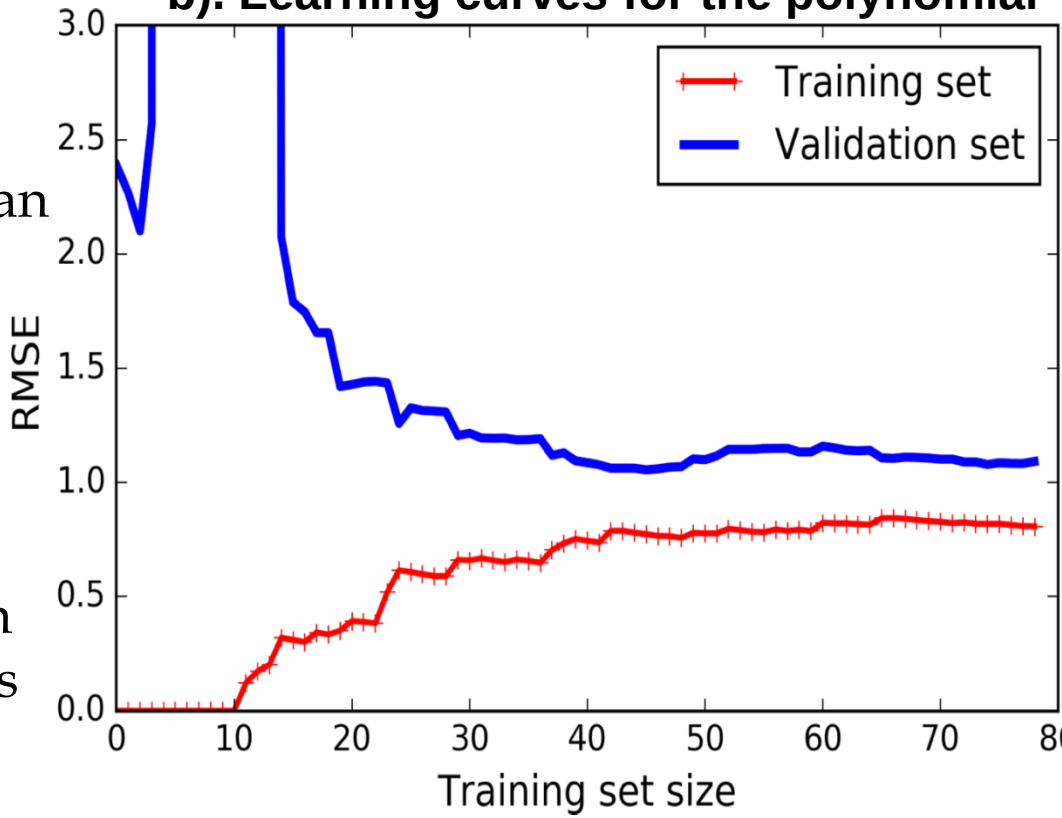


2). Learning Curves

However, if you used a much larger training set, the two curves would continue to get closer.

- There are two very important differences:
- (i) The error (<1.5)on the training data is much lower than with the Linear Regression model (>1.5).
- (ii) There is a gap between the curves. This means that the model performs significantly better on the training data than on the validation data, which is the hallmark of an overfitting model.

b). Learning curves for the polynomial



So, one way to improve an overfitting model is to feed it more training data until the validation error reaches the training error.

4. Regularized Models

- A good way to reduce overfitting is to regularize the model (i.e., to constrain it). For example, a simple way to regularize a polynomial model is to reduce the number of polynomial degrees.
- For a linear model, regularization is typically achieved by constraining the weights of the model's parameters.

- **1). Ridge Regression**
- Ridge Regression is a regularized version of Linear Regression. A regularization term, as shown in next slide, is added to the cost function.
- This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible.

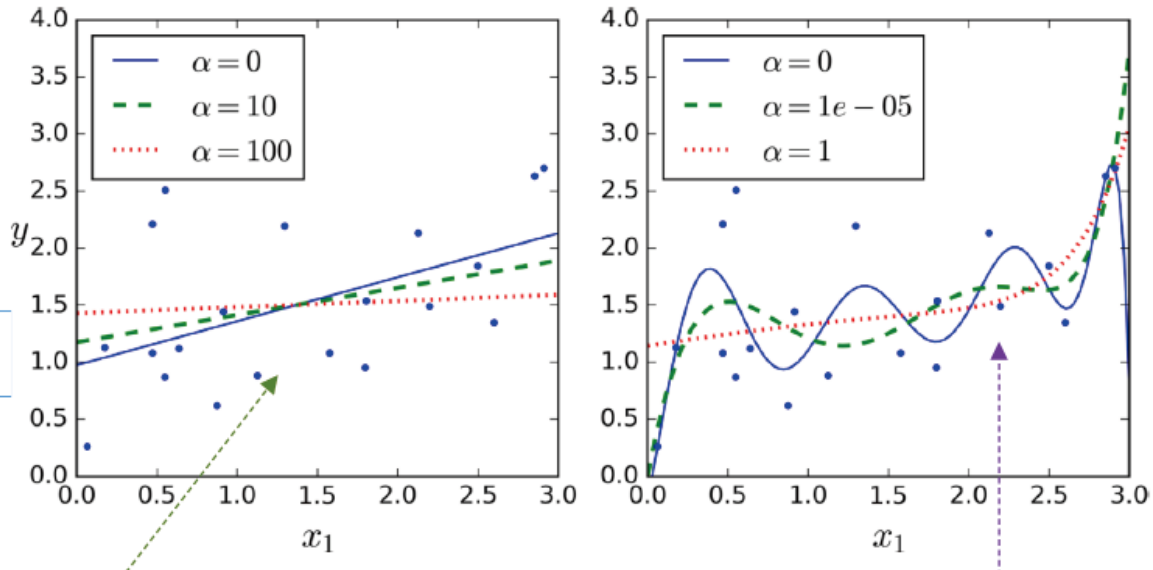
Ridge Regression cost function

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

a regularization term, "$\ell 2$ norm"

- The hyperparameter α controls how much you want to regularize the model.
- If α = 0, then Ridge Regression is just Linear Regression. If α is very large, then all weights (or model's parameters values) end up very close to zero and the result is a flat line going through the data's mean.

Linear and Polynomial Regression with Ridge regularization



- The data is first expanded using PolynomialFeatures(degree=10), then it is scaled using a StandardScaler, and then the Ridge models are applied to the resulting features.
- Note how increasing α leads to flatter (i.e., less extreme, more reasonable) predictions; this reduces the model's variance but increases its bias.

- It is important to scale the data (e.g., using a StandardScaler) before performing Ridge Regression, as it is sensitive to the scale of the input features.

Ridge Regression

- **Ridge Regression closed-form solution**

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X} + \alpha \mathbf{A}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

- Perform Ridge Regression with Scikit-Learn, using a closed-form solution:

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[ 1.55071465]])
```

- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html
- "cholesky" uses the standard scipy.linalg.solve function to obtain a closedform solution.

Ridge Regression

- And using Stochastic Gradient Descent (SGD)

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([[ 1.13500145]])
```

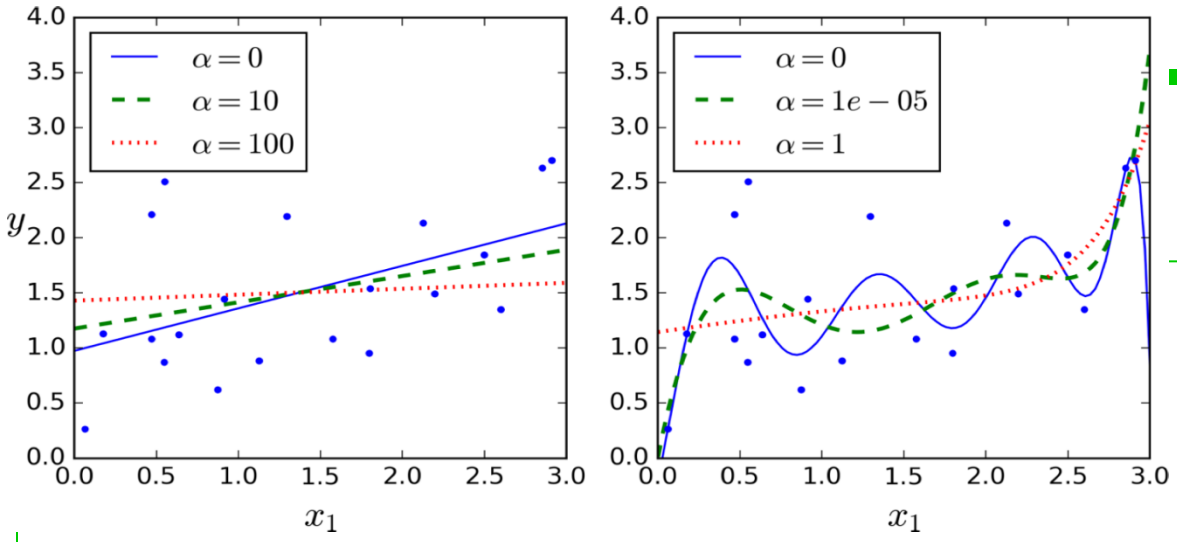
- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html
- The “penalty” , specifying " ℓ2 norm ”, (defined in the last slide) indicates that you want SGD to add a regularization term to the cost function equal to half the square of the ℓ2 norm of the weight vector: this is simply Ridge Regression.

2). Lasso Regression

- **Least Absolute Shrinkage and Selection Operator Regression (simply called Lasso Regression) is another regularized version of Linear Regression:**
- just like Ridge Regression, it adds a regularization term to the cost function, but it uses the ℓ1 norm of the weight vector instead of half the square of the ℓ2 norm.

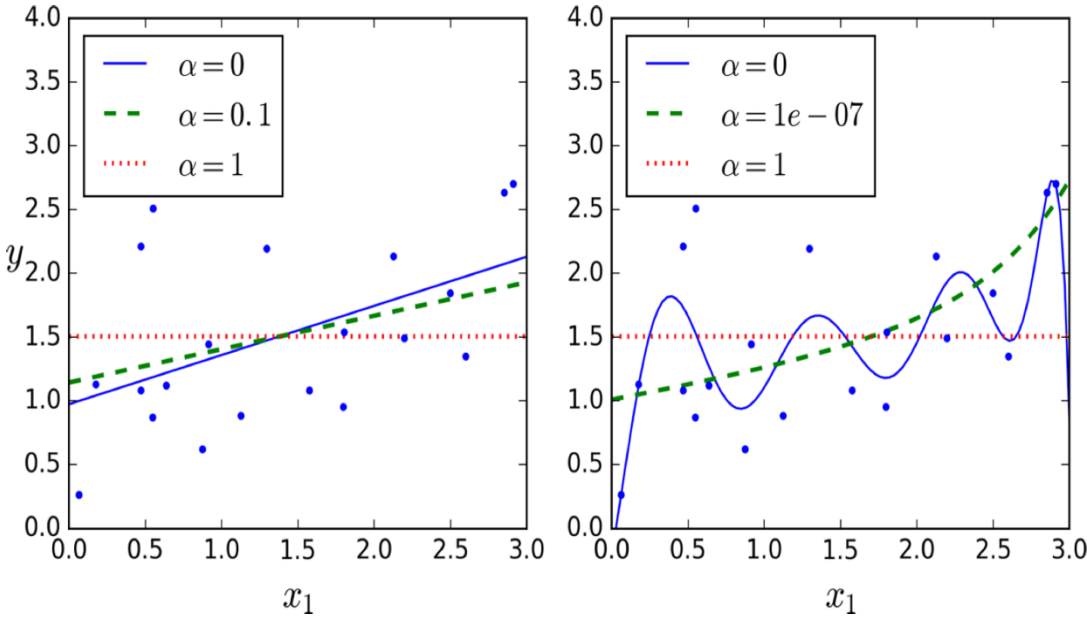
Lasso Regression cost function

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



Ridge Regression

- It replaces Ridge models with Lasso models and uses smaller α values.
- An important characteristic of Lasso Regression is that it tends to completely eliminate the weights of the least important features (i.e., set them to zero).



The dashed line in the right plot (with α = 10^(-7)) looks quadratic, almost linear: all the weights for the high-degree polynomial features are equal to zero.

Lasso Regression

- The Lasso cost function is not differentiable at θi = 0 (for i = 1, 2, ⋯, n), but Gradient Descent still works fine if you use a subgradient vector g instead when any θi = 0.
- A subgradient vector equation you can use for Gradient Descent with the Lasso cost function.

$$g(\theta, J) = \nabla_\theta \mathrm{MSE}(\theta) + \alpha \begin{pmatrix} \mathrm{sign}\,(\theta_1) \\ \mathrm{sign}\,(\theta_2) \\ \vdots \\ \mathrm{sign}\,(\theta_n) \end{pmatrix} \quad \text{where} \quad \mathrm{sign}\,(\theta_i) = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{cases}$$

```
>>> from sklearn.linear_model import Lasso
>>> lasso_reg = Lasso(alpha=0.1)
>>> lasso_reg.fit(X, y)
>>> lasso_reg.predict([[1.5]])
array([ 1.53788174])
```

Here is a small Scikit-Learn example using the Lasso class.
Note that you could instead use an SGDRegressor(penalty=" ℓ1 ").

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

结束

2020年9月30日