Machine Learning Introduction

Supervisor
PD Dr. Friedhelm Schwenker
Institute of Neural Information Processing
Ulm University
Students
Abdelrahman Mahmoud And Mohamed Ashry

April 1, 2020

chapter 4: Training Models!

Frame Title

- So far we have treated Machine Learning models and their training algorithms mostly like black boxes.
- However, having a good understanding of how things work can help you quickly home in on the appropriate model, the right training algorithm to use, and a good set of hyperparameters for your task.
- In this chapter, we will start by looking at the Linear Regression model, one of the simplest models there is. We will discuss two very different ways to train it:
  - Using a direct "closed-form" equation.
  - Using an iterative optimization approach, called Gradient Descent (GD).
- Next we will look at Polynomial Regression, a more complex model that can fit non- linear datasets. Since this model has more parameters than Linear Regression, it is more prone to overfitting the training data, so we will look at how to detect whether or not this is the case, using learning curves, and then we will look at several regulari- zation techniques that can reduce the risk of overfitting the training set.

Linear Regression

Equation 4-2. Linear Regression model prediction (vectorized form)

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

- $\boldsymbol{\theta}$ is the model's *parameter vector*, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- $\mathbf{x}$ is the instance's *feature vector*, containing x_0 to x_n, with x_0 always equal to 1.
- $\boldsymbol{\theta} \cdot \mathbf{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\mathbf{x}$, which is of course equal to $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$.
- $h_{\boldsymbol{\theta}}$ is the hypothesis function, using the model parameters $\boldsymbol{\theta}$.

Okay, that's the Linear Regression model, so now how do we train it ?!

Well, recall that training a model means setting its parameters so that the model best fits the training set. For this purpose, we first need a measure of how well (or poorly) the model fits the training data. In Chapter 2 we saw that the most common performance measure of a regression model is the Root Mean Square Error (RMSE) There- fore, to train a Linear Regression model, you need to find the value of that minimizes the RMSE. In practice, it is simpler to minimize the Mean Square Error (MSE)

solution !!

- The Normal Equation.
- Gradient Descent

The Normal Equation.

- To find the value of that minimizes the cost function, there is a closed-form solution —in other words, a mathematical equation that gives the result directly. This is called the Normal Equation (Equation 4-4).

Equation 4-4. Normal Equation

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

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

Gradient Descent

- Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.
- Suppose you are lost in the mountains in a dense fog; you can only feel the slope of the ground below your feet. A good strategy to get to the bottom of the valley quickly is to go downhill in the direction of the steepest slope. This is exactly what Gradient Descent does: it measures the local gradient of the error function with regards to the parameter vector , and it goes in the direction of descending gradient. Once the gradient is zero, you have reached a minimum!

- Concretely, you start by filling with random values (this is called random initializa- tion), and then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum (see Figure 4-3).

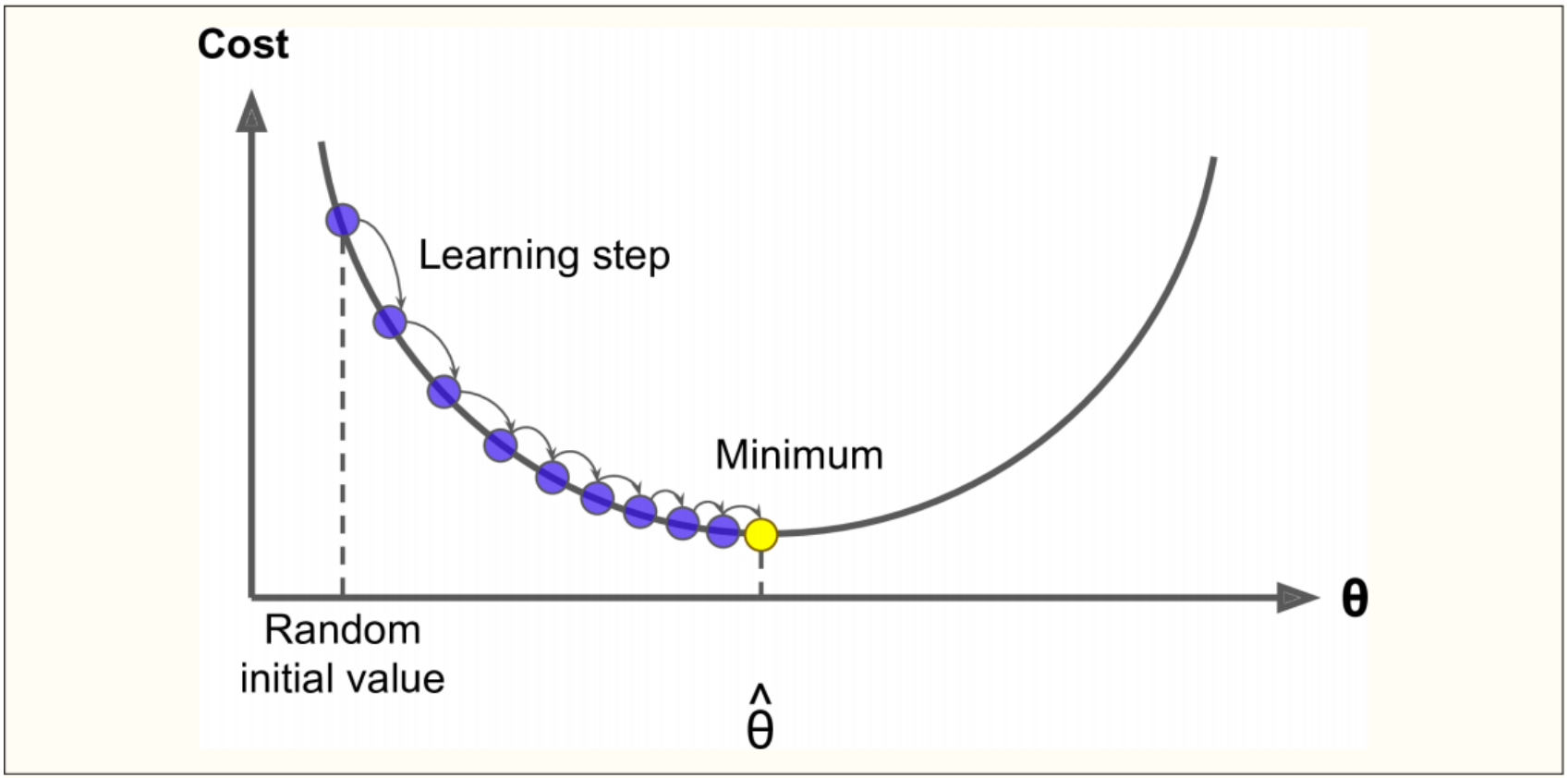

Figure 4-3. Gradient Descent

Learning Rate

- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time (see Figure 4-4).
- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time (see Figure 4-4).
- On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge, with larger and larger values, failing to find a good solution (see Figure 4-5).

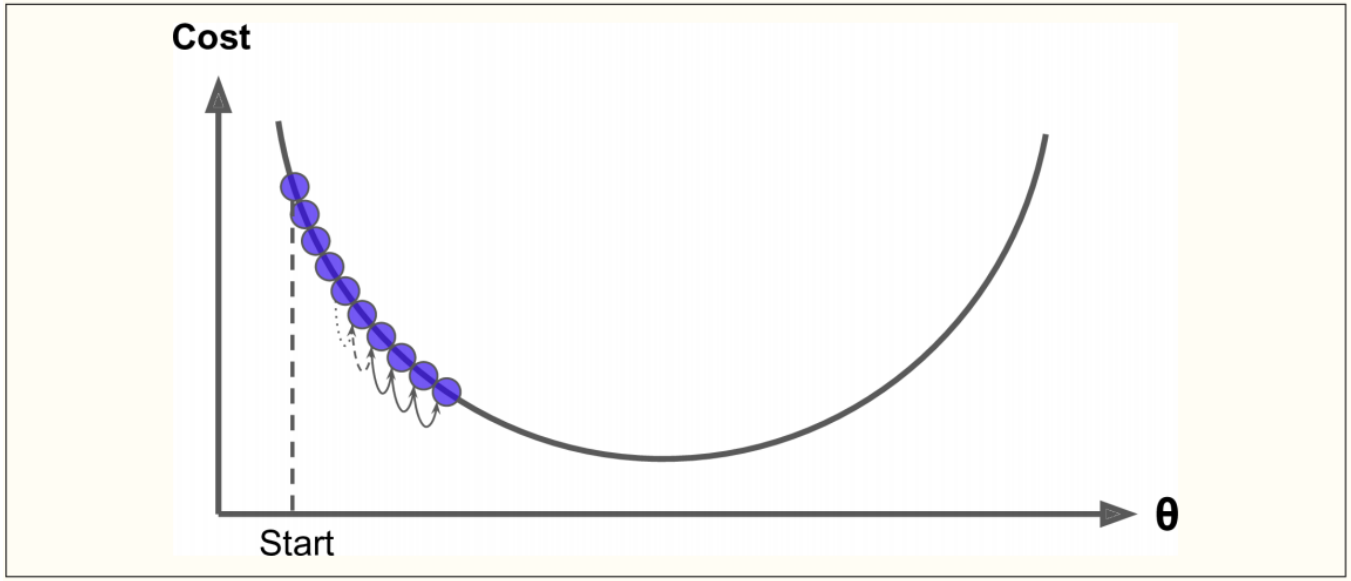

Figure 4-4. Learning rate too small

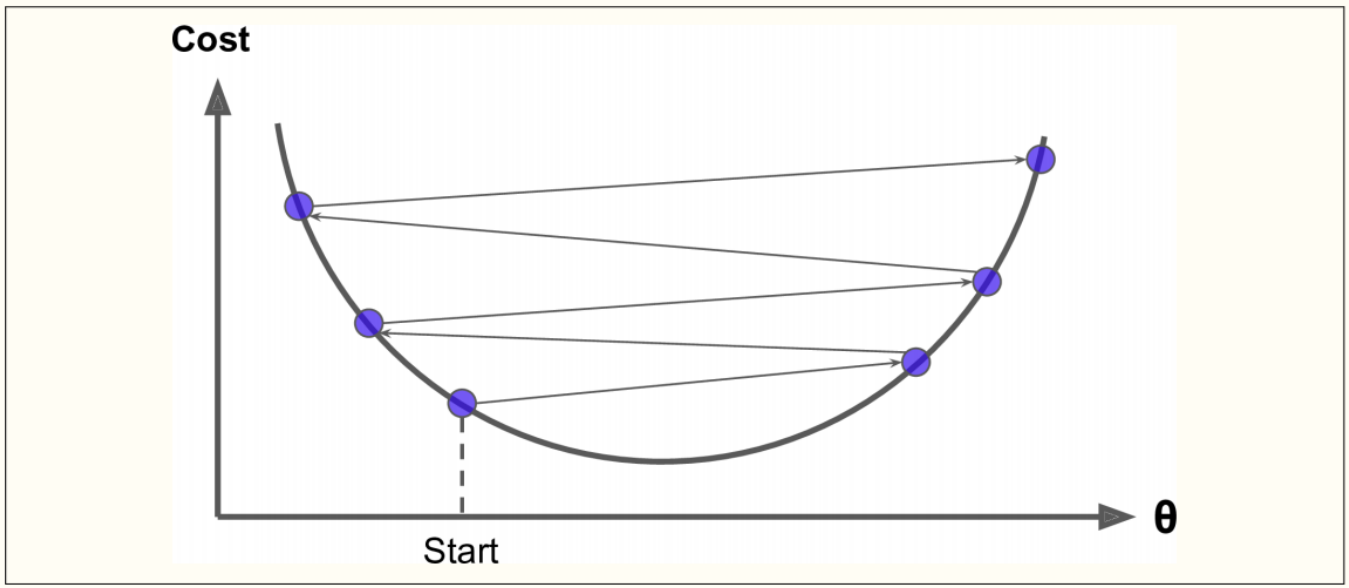

Figure 4-5. Learning rate too large

Batch Gradient Descent

- To implement Gradient Descent, you need to compute the gradient of the cost func- tion with regards to each model parameter j. In other words, you need to calculate how much the cost function will change if you change j just a little bit
- This is called a partial derivative. It is like asking "what is the slope of the mountain under my feet if I face east?" and then asking the same question facing north (and so on for all other dimensions, if you can imagine a universe with more than three dimensions).

Equation 4-5. Partial derivatives of the cost function

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

Stochastic Gradient Descent

- The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.
- At the opposite extreme, Stochastic Gradient Descent just picks a random instance in the training set at every step and computes the gradients based only on that single instance.
- Obviously this makes the algorithm much faster since it has very little data to manipulate at every iteration. It also makes it possible to train on huge training sets, since only one instance needs to be in memory at each iteration (SGD can be implemented as an out-of-core algorithm)

- On the other hand, due to its stochastic (i.e., random) nature, this algorithm is much less regular than Batch Gradient Descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on aver- age. Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down (see Figure 4-9). So once the algorithm stops, the final parameter values are good, but not optimal.

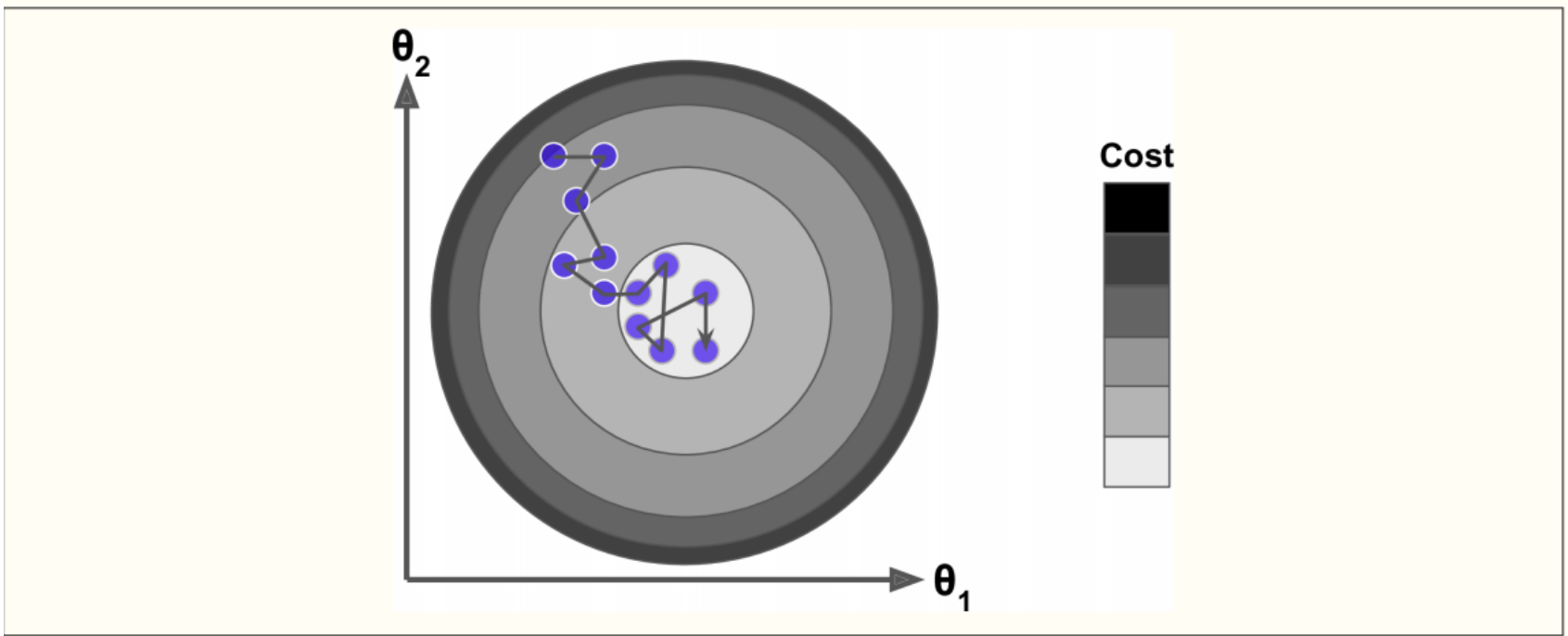

Figure 4-9. Stochastic Gradient Descent

Mini-batch Gradient Descent

- The last Gradient Descent algorithm we will look at is called Mini-batch Gradient Descent. It is quite simple to understand once you know Batch and Stochastic Gradient Descent: at each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini batch GD computes the gradients on small random sets of instances called mini batches.
- The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs.

Comparison

- The algorithm's progress in parameter space is less erratic than with SGD, especially with fairly large mini-batches. As a result, Mini-batch GD will end up walking around a bit closer to the minimum than SGD. But, on the other hand, it may be harder for it to escape from local minima (in the case of problems that suffer from local minima, unlike Linear Regression as we saw earlier). Figure 4-11 shows the paths taken by the three Gradient Descent algorithms in parameter space during training. They all end up near the minimum, but Batch GD's path actually stops at the minimum, while both Stochastic GD and Mini-batch GD continue to walk around. However, don't forget that Batch GD takes a lot of time to take each step, and Stochastic GD and Mini-batch GD would also reach the minimum if you used a good learning schedule.

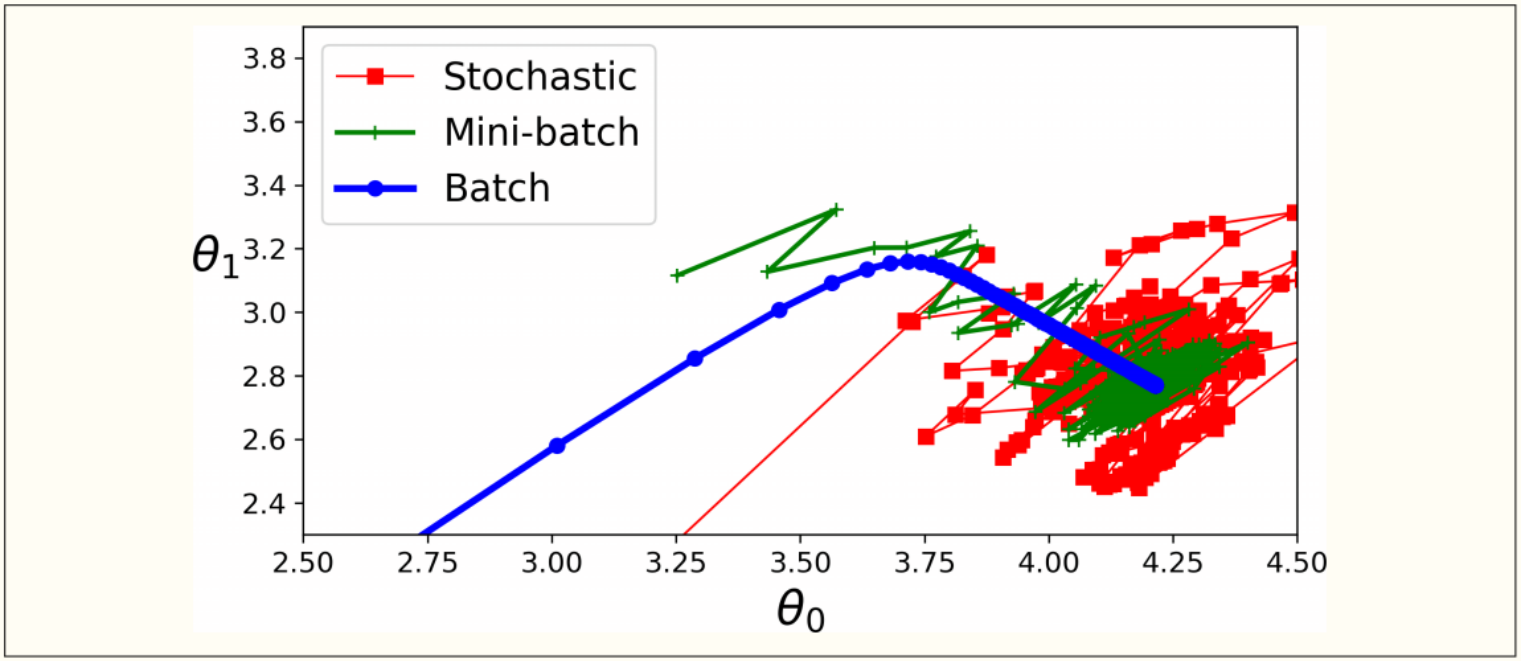

Figure 4-11. Gradient Descent paths in parameter space

Table 4-1. Comparison of algorithms for Linear Regression

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`

Polynomial Regression

- What if your data is actually more complex than a simple straight line? Surprisingly, you can actually use a linear model to fit nonlinear data. A simple way to do this is to 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.
- Note that when there are multiple features, Polynomial Regression is capable of find- ing relationships between features (which is something a plain Linear Regression model cannot do). This is made possible by the fact that PolynomialFeatures also adds 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.

The Bias/Variance Tradeoff

- An important theoretical result of statistics and Machine Learning is the fact that a model's generalization error can be expressed as the sum of three very different errors:
  - 1) Bias
  - 2) Variance...
  - 3) Irreducible Error

Regularized Linear Models

- As we saw in Chapters 1 and 2, a good way to reduce overfitting is to regularize the model (i.e., to constrain it): the fewer degrees of freedom it has, the harder it will be for it to overfit the data. 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
- there is three types of regression which implement three different ways to constrain the weights.
  - Ridge Regression
  - Lasso Regression
  - Elastic Net
  - Early Stopping

Ridge Regression

- Ridge Regression (also called Tikhonov regularization) is a regularized version of Lin- ear Regression: a regularization term equal to $\alpha \sum_{i=1}^{n} \theta^2 = 1$
  A 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.
- Note that the regularization term should only be added to the cost function during training. Once the model is trained, you want to evaluate the model's performance using the unregularized performance measure.

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 1 norm of the weight vector instead of half the square of the 2 norm (see Equation 4-10)

Equation 4-10. Lasso Regression cost function

$j(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} \mid \theta_1 \mid$

- An important characteristic of Lasso Regression is that it tends to completely eliminate the weights of the least important features

Elastic Net

- Elastic Net is a middle ground between Ridge Regression and Lasso Regression.
- The regularization term is a simple mix of both Ridge and Lasso's regularization terms, and you can control the mix ratio r. When r = 0, Elastic Net is equivalent to Ridge Regression, and when r = 1, it is equivalent to Lasso Regression
- *Equation 4-12. Elastic Net cost function*
  $j(\theta) = MSE(\theta) + r\alpha\sum_{i=1}^{n} \mid \theta_1 \mid + \frac{1-r}{2}\sum_{i=1}^{n} \theta^2$

Early Stopping

- A very different way to regularize iterative learning algorithms such as Gradient Descent is to stop training as soon as the validation error reaches a minimum.

Logistic Regression

- Logistic Regression (also called Logit Regression) is commonly used to estimate the probability that an instance belongs to a particular class (e.g., what is the probability that this email is spam?).
- If the estimated probability is greater than 50(called the positive class, labeled "1"), or else it predicts that it does not (i.e., it belongs to the negative class, labeled "0"). This makes it a binary classifier.

Estimating Probabilities

- So how does it work? Just like a Linear Regression model, a Logistic Regression model computes a weighted sum of the input features (plus a bias term), but instead of outputting the result directly like the Linear Regression model does, it outputs the logistic of this result
- segmid function $\sigma(t) = \frac{1}{1+exp(-t)}$

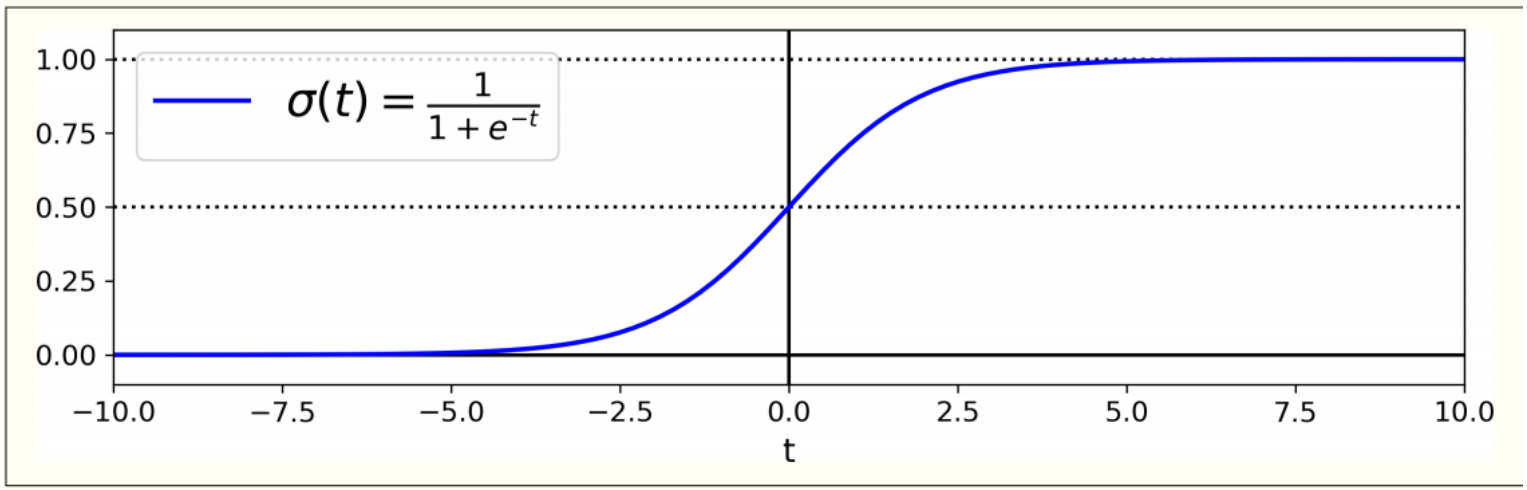

Figure 4-21. Logistic function

Training and Cost Function

- now you know how a Logistic Regression model estimates probabilities and makes predictions. But how is it trained? The objective of training is to set the param- eter vector so that the model estimates high probabilities for positive instances (y = 1) and low probabilities for negative instances (y = 0). This idea is captured by the cost function shown in Equation 4-16 for a single training instance x

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

costfunction

- The cost function over the whole training set is simply the average cost over all train- ing instances.

$$j(\theta) = \frac{1}{m} \sum_{i=1}^{m} [y^i \log(\hat{p}^i) + (1 - y^i) \log(1 - \hat{p}^i)]$$

- Logistic cost function partial derivatives

$$\frac{\partial}{\theta_j} j(\theta) = \frac{1}{m} \sum_{i=1}^{m} (\sigma(\theta^T x^i) - y^i) x_j^i$$

Softmax Regression

- The Logistic Regression model can be generalized to support multiple classes directly, without having to train and combine multiple binary classifiers (as discussed in Chapter 3). This is called Somax Regression, or Multinomial Logistic Regression
- The idea is quite simple: when given an instance x, the Softmax Regression model first computes a score sk(x) for each class k, then estimates the probability of each class by applying the somax function (also called the normalized exponential) to the scores. The equation to compute sk(x) should look familiar, as it is just like the equa- tion for Linear Regression prediction $s_k(x) = x^T \theta^k$

Thank You !