Chapter 4. Training Models

Two different ways to train a Linear Regression model:

- Use a direct "closed-form" equation that directly computes the model parameters that best fit the model to the training set.
- Use an iterative optimization approach, Gradient Descent (GD), Batch GD, Mini-batch GD, and Stochastic GD.

Polynomial Regression, more prone to overfitting. Detect overfitting using learning curves, reduce using regularization.

Logistic Regression

Softmax Regression

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 (intercept term).

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

Cost function

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

The Normal Equation

Closed-form solution

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

Computational Complexity

WARNING

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

On the positive side, this equation is linear with regards to the number of instances and features in the training set (it is $O(m)$), so it handles large training sets efficiently, provided they can fit in memory.

Gradient Descent

Concretely, you start by filling θ with random values (*random initialization*), and then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function, until the algorithm *converges* to a minimum.

Learning rate hyperparameter: small, slow convergence; large, algorithm diverge.

Two main challenges with Gradient Descent:

- Random initialization, it could converge to a local minimum;
- It may take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum.

WARNING

> When using Gradient Descent, you should ensure that all features have a similar scale, or else it will take much longer to converge.

Batch Gradient Descent

It involves calculations over the full training set X, at each Gradient Descent step.

How to set the number of iterations: set a very large number of iterations but to interrupt the algorithm when the norm of the gradient vector becomes smaller than ϵ (tolerance).

Stochastic Gradient Descent

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. (fast, train on huge training sets)

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 average. 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. So once the algorithm stops, the final parameter values are good, but not optimal.

Randomness is good to escape from local optima, but bad because it means that the algorithm can never settle at the minimum. One solution: gradually reduce the learning rate. *Simulated annealing*: the steps start out large (which helps make quick progress and escape local minima),

then get smaller and smaller, allowing the algorithm to settle at the global minimum. The function that determines the learning rate at each iteration is called the *learning schedule*.

Epoch: each round of m iterations.

Mini-batch Gradient Descent

Advantage: performance boost from hardware optimization of matrix operations, especially with GPUs.

Less erratic than with SGD, harder for it to escape from local minima.

Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	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

Table 4-1. Comparison of algorithms for Linear Regression

Polynomial Regression

Polynomial Regression is capable of finding relationships between features.

Learning Curves

One way to estimate a model's generalization performance is using cross-validation.

Another way is to look at the *learning curves*: plots of the model's performance on the training set and the validation set as a function of the training set size.

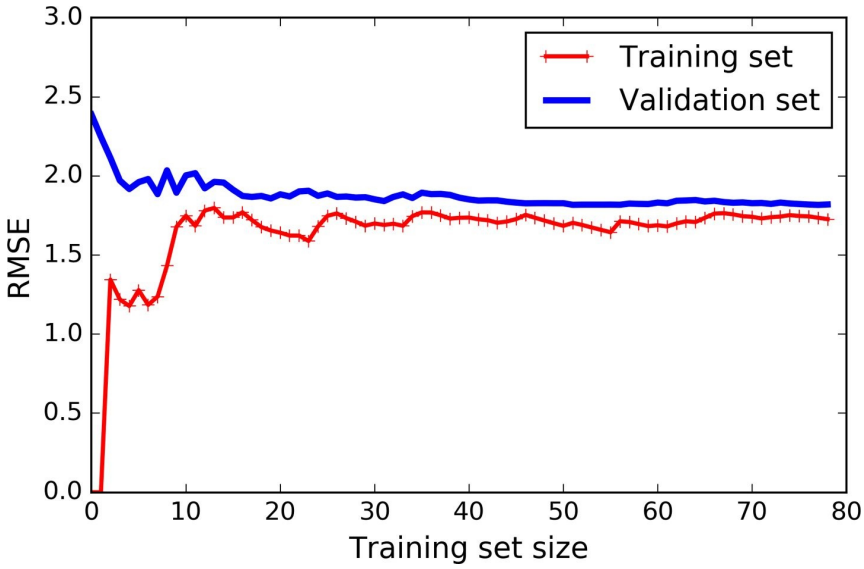

Figure 4-15. Underfitting model. Both curves have reached a plateau; they are close and fairly high

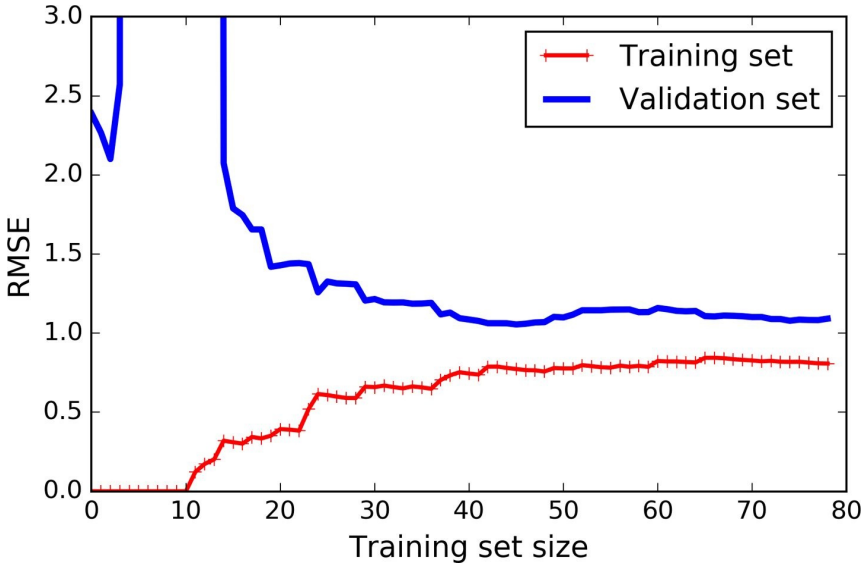

Figure 4-16. Overfitting model.

The error on the training data is much lower than with the underfitting model.

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

Bias/variance tradeoff

Amodel's generalization error can be expressed as the sum of three very different errors:

- Bias. Underfit due to wrong assumption.
- Variance. Overfit due to the model's excessive sensitivity to small variation in the training data.
- Irreducible error. Due to the noiseness of the data itself, clean up the data.

Regularized Linear Models

Ridge Regression

Tikhonov regularization: a *regularization term* equal to $\alpha \sum_{i=1}^{n} \theta_i^2$ is added to the cost function.

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.

NOTE

> It is quite common for the cost function used during training to be different from the performance measure used for testing. Apart from regularization, another reason why they might be different is that a good training cost function should have optimization-friendly derivatives, while the performance measure used for testing should be as close as possible to the final objective.

Ridge Regression cost function:

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

The penalty hyperparameter α controls how much you want to regularize the model. Increasing α leads to flatter (i.e., less extreme, more reasonable) predictions; this reduces the model's variance but increases its bias.

WARNING

> It is important to scale the data before performing Ridge Regression, as it is sensitive to the scale of the input features. This is true of most regularized models.

Lasso Regression

Least Absolute Shrinkage and Selection Operator Regression, l_1 norm.

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

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). In other words, Lasso Regression automatically performs feature selection and outputs a *sparse model* (i.e., with few nonzero feature weights).

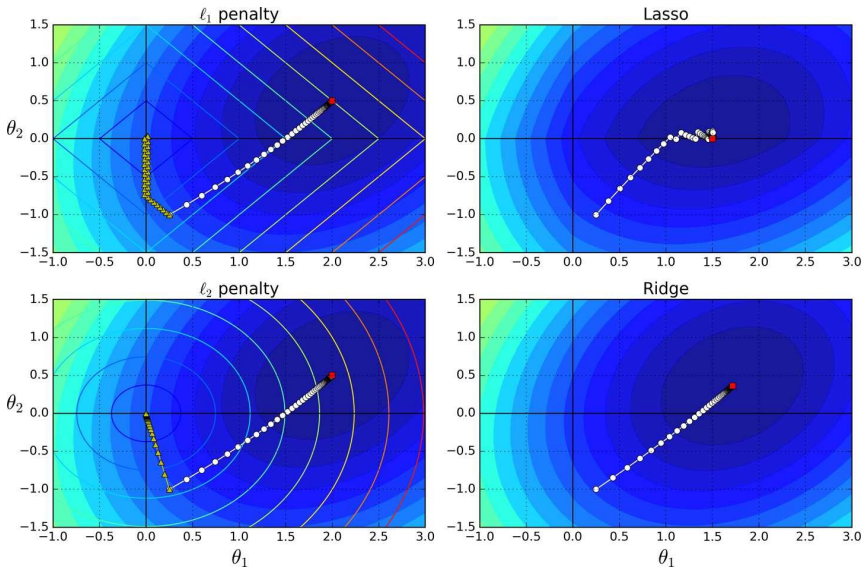

Figure 4-19. Lasso versus regularization

Top-right, l_1 penalty with $\alpha = 0.5$, The global minimum is on $\theta_2 = 0$ axis. BGD first reaches $\theta_2 = 0$, then rolls down the gutter until it reaches the global minimum.

On the Lasso cost function, the BGD path tends to bounce across the gutter toward the end. This is because the slope changes abruptly at $\theta_2 = 0$. You need to gradually reduce the learning rate in order to actually converge to the global minimum.

Elastic Net

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

Ridge is a good default, but if you suspect that only a few features are actually useful, you should prefer Lasso or Elastic Net since they tend to reduce the useless features' weights down to zero. In general, Elastic Net is preferred over Lasso since Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.

Early Stopping

Stop training as soon as the validation error reaches a minimum. ("beautiful free lunch", Geoffrey Hinton)

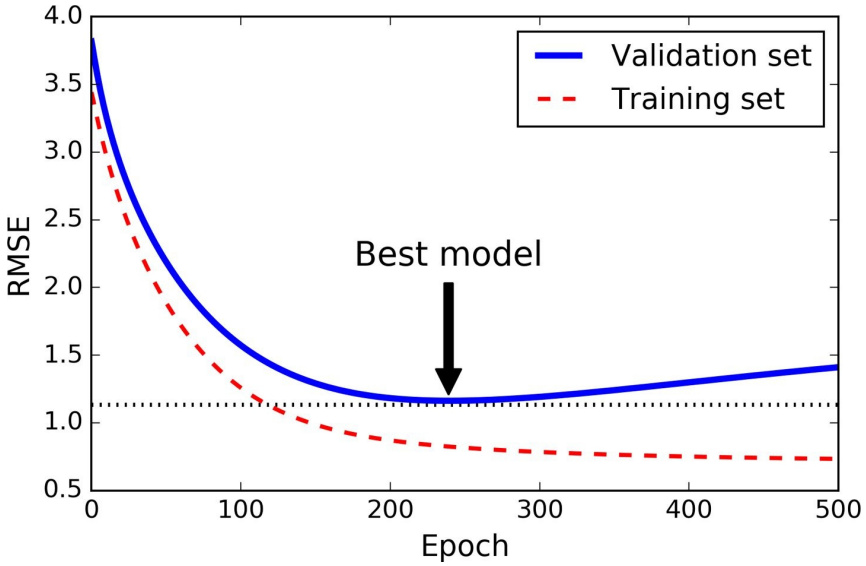

Figure 4-20. Early stopping regularization

Logistic Regression

It is used to estimate the probability that an instance belongs to a particular class.

Logistic Regression model estimated probability (vectorized form):

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

The logistic — also called the *logit*, noted $\sigma()$ — is a *sigmoid function* (i.e., S-shaped) that outputs a number between 0 and 1.

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

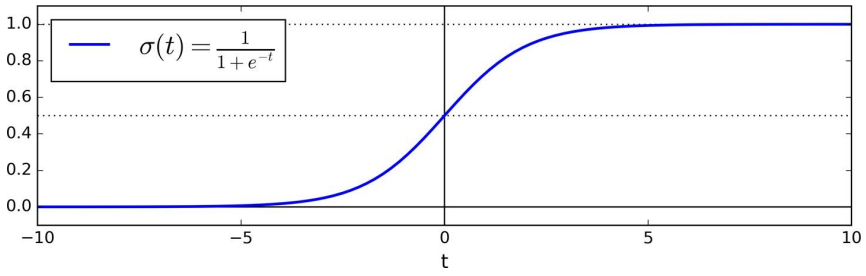

Figure 4-21. Logistic function

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

Training and Cost function

Logistic Regression cost function (log loss)

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

Bad news: no known closed-form (Normal) equation. Good news: convex, Gradient Descent works.

Logistic cost function partial derivatives

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} [\sigma(\theta^T \cdot \mathbf{x}^{(i)}) - y^{(i)}] x_j^{(i)}$$

Decision Boundaries

The dashed line represents the points where the model estimates a 50 probability: this is the model's decision boundary.

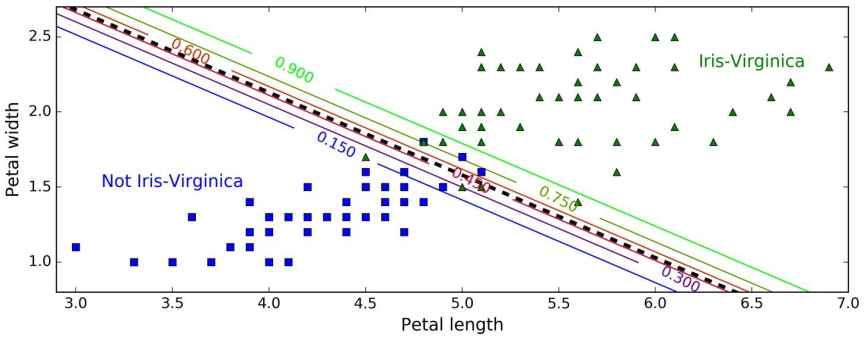

Figure 4-24. Linear decision boundary

Softmax Regression

Idea: when given an instance x, the Softmax Regression model first computes a score s k (x) for each class k, then estimates the probability of each class by applying the *softmax function* (*normalized exponential*) to the scores.

Softmax score for class k

$$s_k(x\mathbf{x}) = \theta_k^T \cdot \mathbf{x}$$

Softmax function

$$\hat{p}_k = \sigma(s(\mathbf{x}))_k = \frac{exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} exp(s_j(\mathbf{x}))}$$

Softmax Regression classifier prediction

$$\hat{y} = \underset{k}{argmax}\ \sigma(s(\mathbf{x}))_k = \underset{k}{argmax}\ s_k(\mathbf{x}) = \underset{k}{argmax}\ (\theta_k^T \cdot \mathbf{x})$$

Cross entropy cost function:

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

Cross entropy gradient vector for class k

$$\nabla_{\theta_k} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (\hat{p}_k^{(i)} - y_k^{(i)}) \mathbf{x}^{(i)}$$