Training Models

Chapter 4: pp 111 – 152

Regression & Classification

- Regression
  - Linear Regression
    - Analytical solution (Normal Equation)
    - Iterative optimization approach (Gradient Descent)
  - Polynomial Regression
- Classification
  - Logistic Regression
  - Softmax Regression
- Mathematics
  - https://github.com/ageron/handson-ml2/blob/master/math_differential_calculus.ipynb
  - https://github.com/ageron/handson-ml2/blob/master/math_linear_algebra.ipynb

Linear Regression

Equation 4-1. Linear Regression model prediction

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

- $\hat{y}$: predicted value
- n : the number of features
- x_i : the i-th feature value
- θ_j : the j-th model parameter
  - θ_0 : the bias term
  - $\theta_1, \ldots, \theta_n$: the feature weights

In Vectorized Form

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 $\boldsymbol{\theta}$.

Alternatively,

$$\hat{y} = \boldsymbol{\theta}^{\mathbf{T}}\mathbf{x}$$

Linear Regression Cost Function

Equation 2-1. Root Mean Square Error (RMSE)

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

Equation 4-3. MSE cost function for a Linear Regression model

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

- Find the value of $\boldsymbol{\theta}$ that minimizes the error

The Normal Equation

- Closed-form, analytical solution

Equation 4-4. Normal Equation

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

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

- Pseudo-inverse can be computed by SVD

- Example:

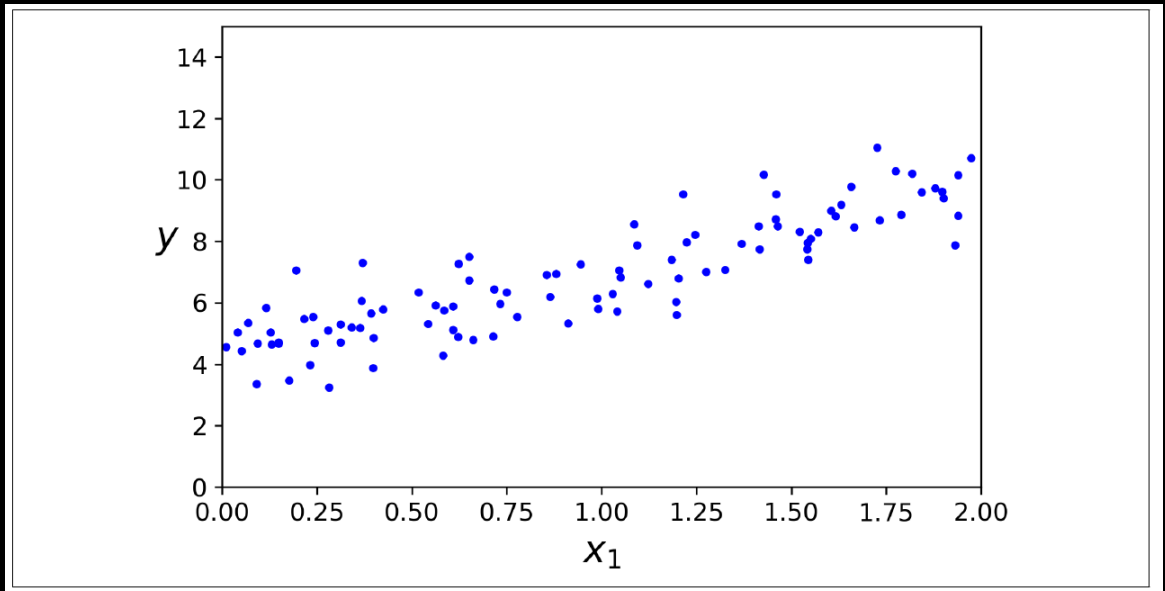


Figure 4-1. Randomly generated linear dataset

- Compute $\boldsymbol{\theta}$ using the Normal Equation.

```
array([[4.21509616],
       [2.77011339]])
```

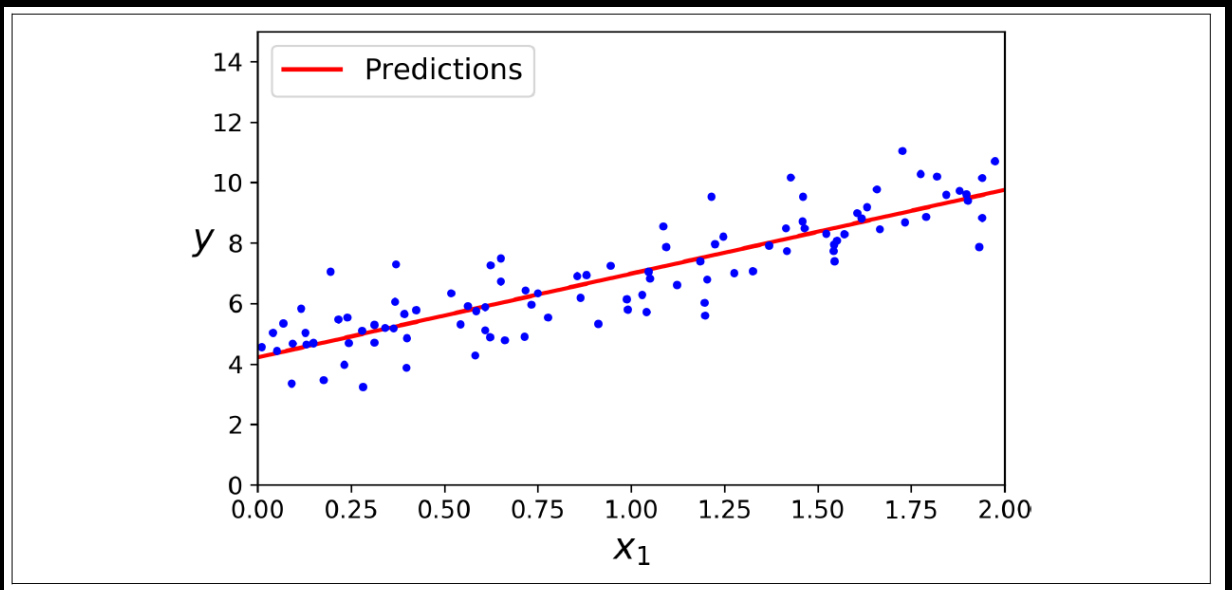


Figure 4-2. Linear Regression model predictions

Computational Complexity

- Grows exponentially with the number features
  - About $O(n^{2.4})$ to $O(n^3)$ where n is the number features
- Is linear with respect to the number of examples
  - $O(m)$ where is m the number of examples

Gradient Descent

- Going downhill in the direction of the steepest slope

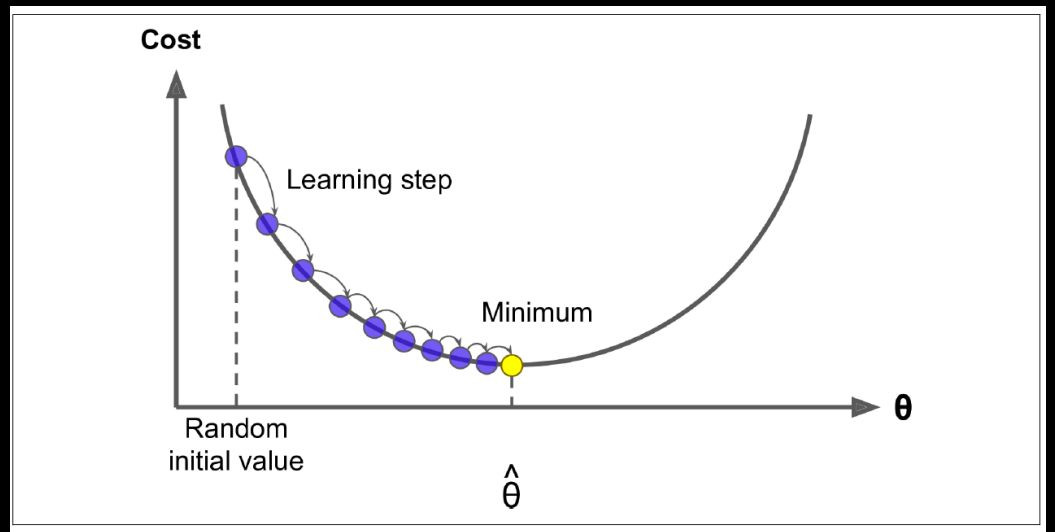


Figure 4-3. Gradient Descent

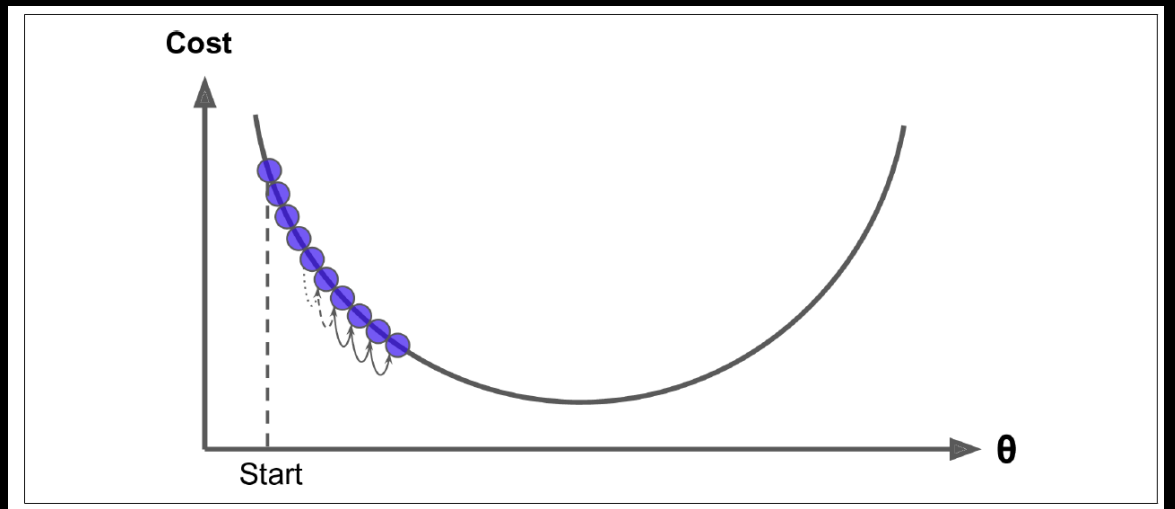


Figure 4-4. Learning rate too small

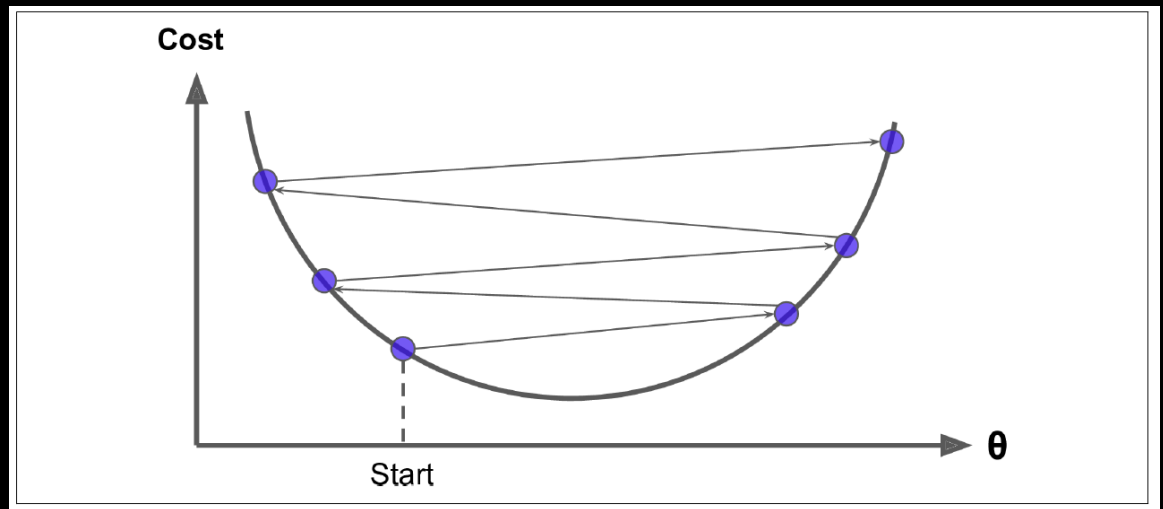


Figure 4-5. Learning rate too large

Local Minima vs Global Minimum

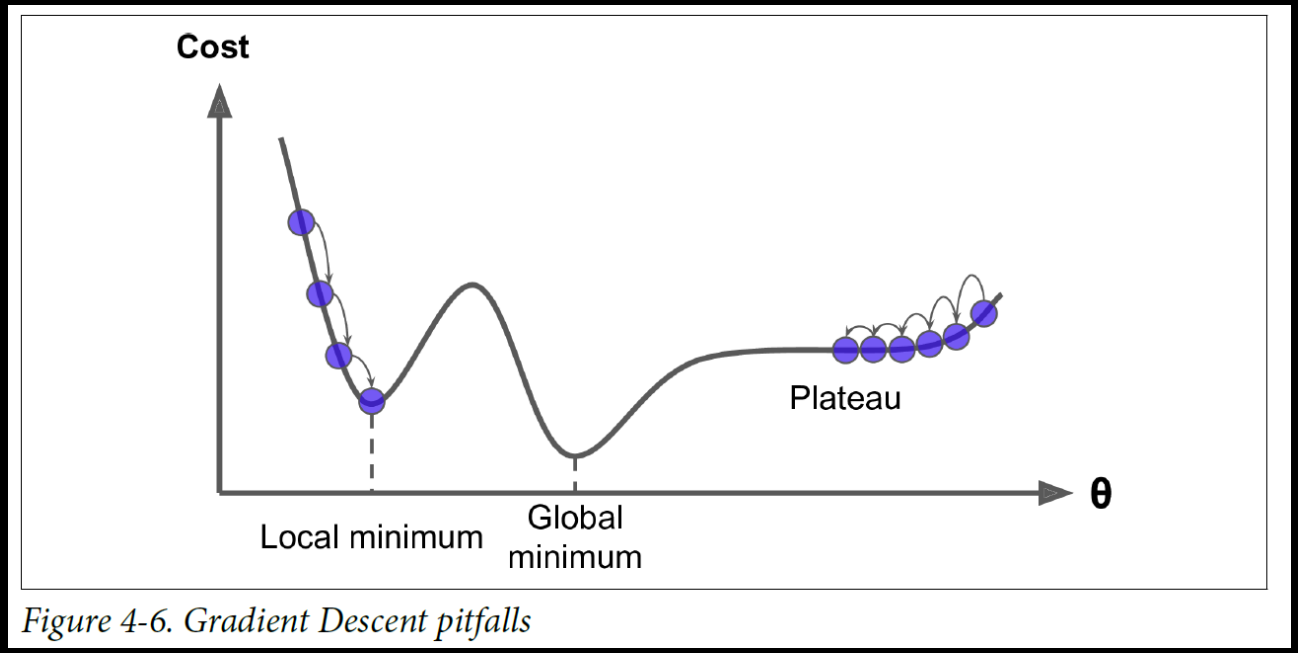


Figure 4-6. Gradient Descent pitfalls

- MSE cost function for Linear Regression is a convex function → only one global minimum

Feature Scaling in Gradient Descent

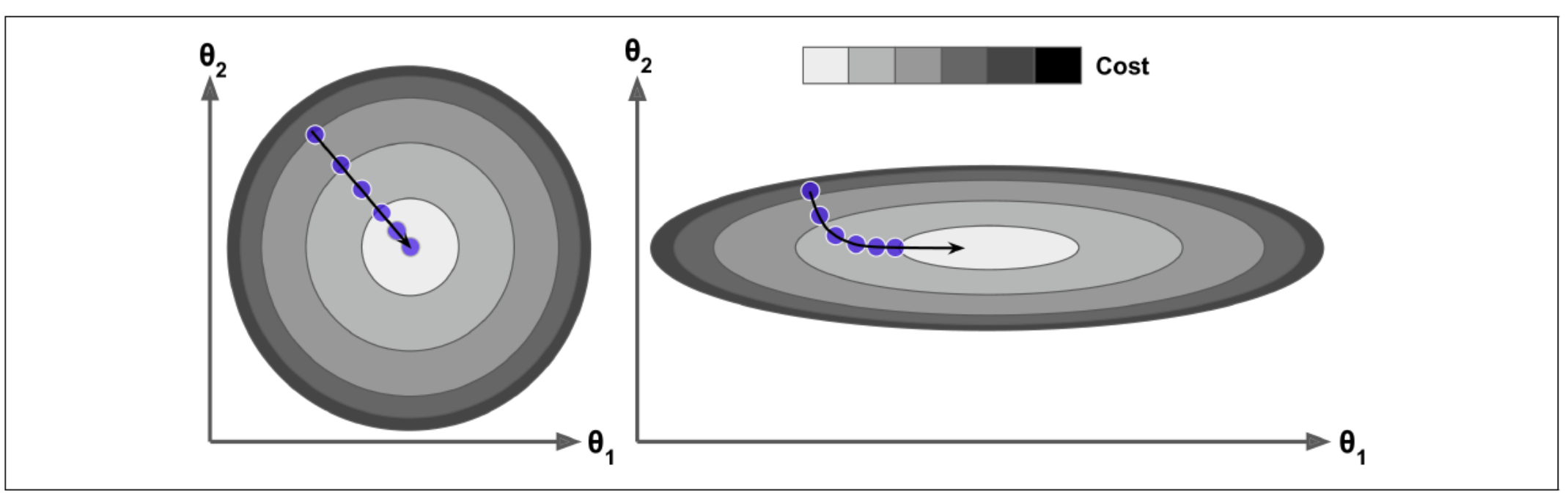


Figure 4-7. Gradient Descent with and without feature scaling

Batch Gradient Descent

Equation 4-5. Partial derivatives of the cost function

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

- Rewrite in a more compact form:

Equation 4-6. Gradient vector of the cost function

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

Batch Gradient Descent

- Uses the whole batch of training data at every step

Equation 4-7. Gradient Descent step

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

- Example:

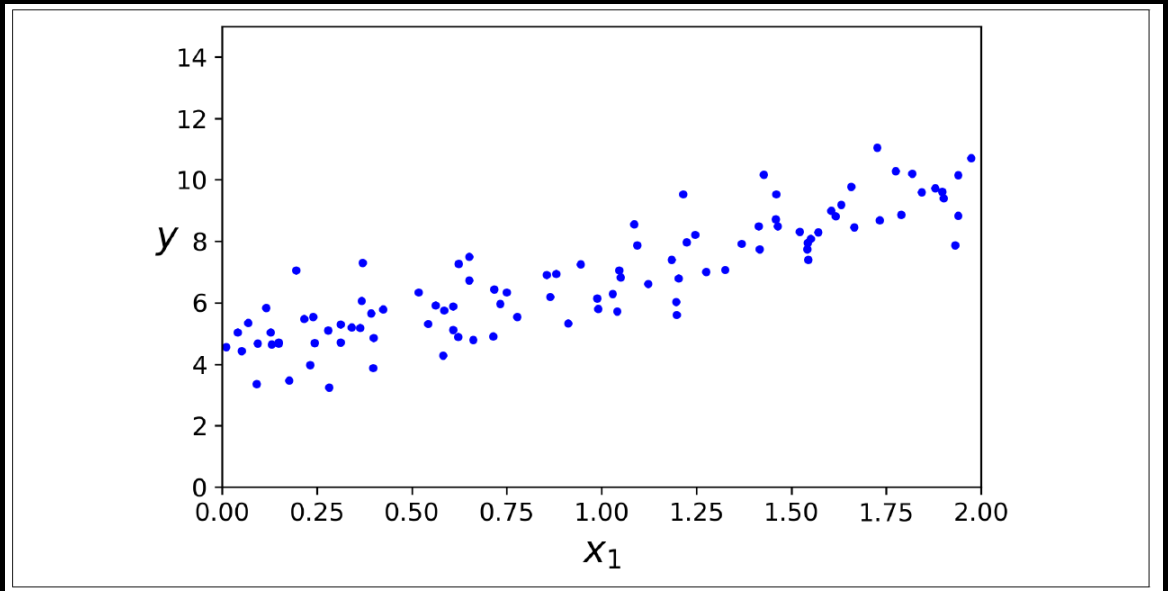


Figure 4-1. Randomly generated linear dataset

- With $\eta = 0.1$ → same result as Normal Equation

```
array([[4.21509616],
       [2.77011339]])
```

- But what happens if eta is different?

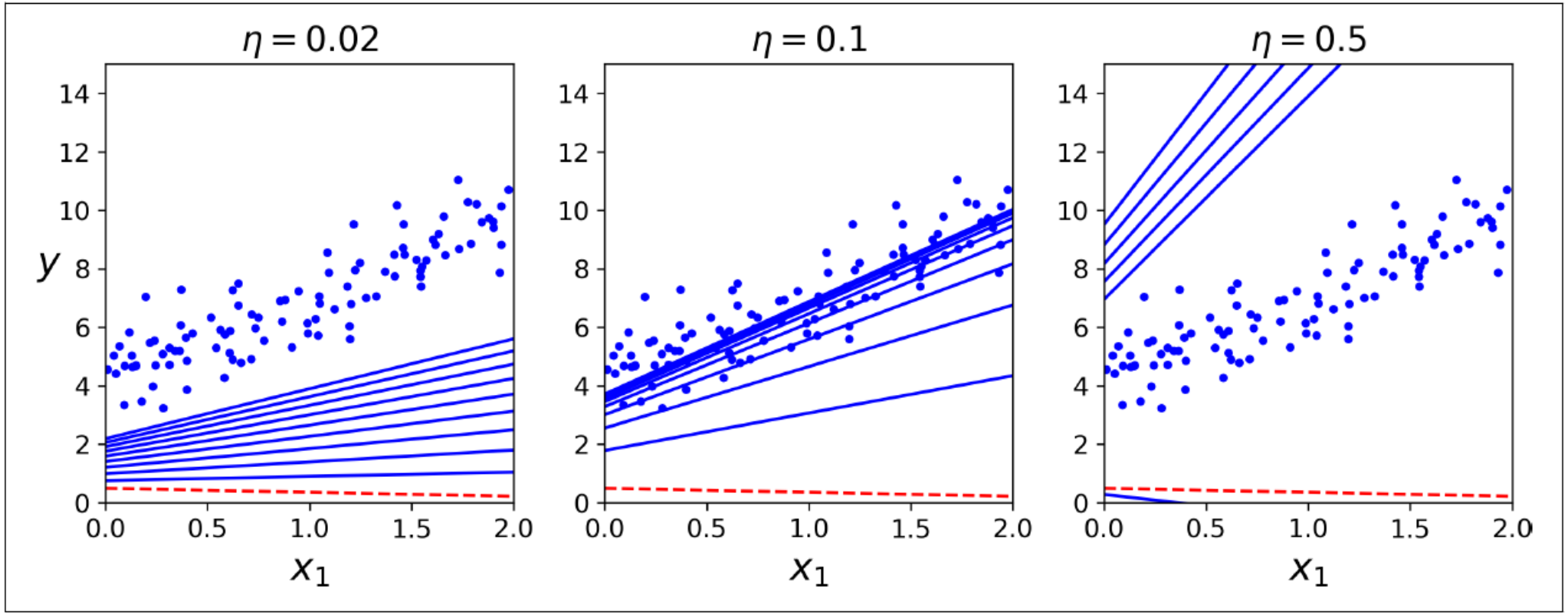


Figure 4-8. Gradient Descent with various learning rates

Stochastic Gradient Descent

- Picks a random example from the training set to compute the gradients

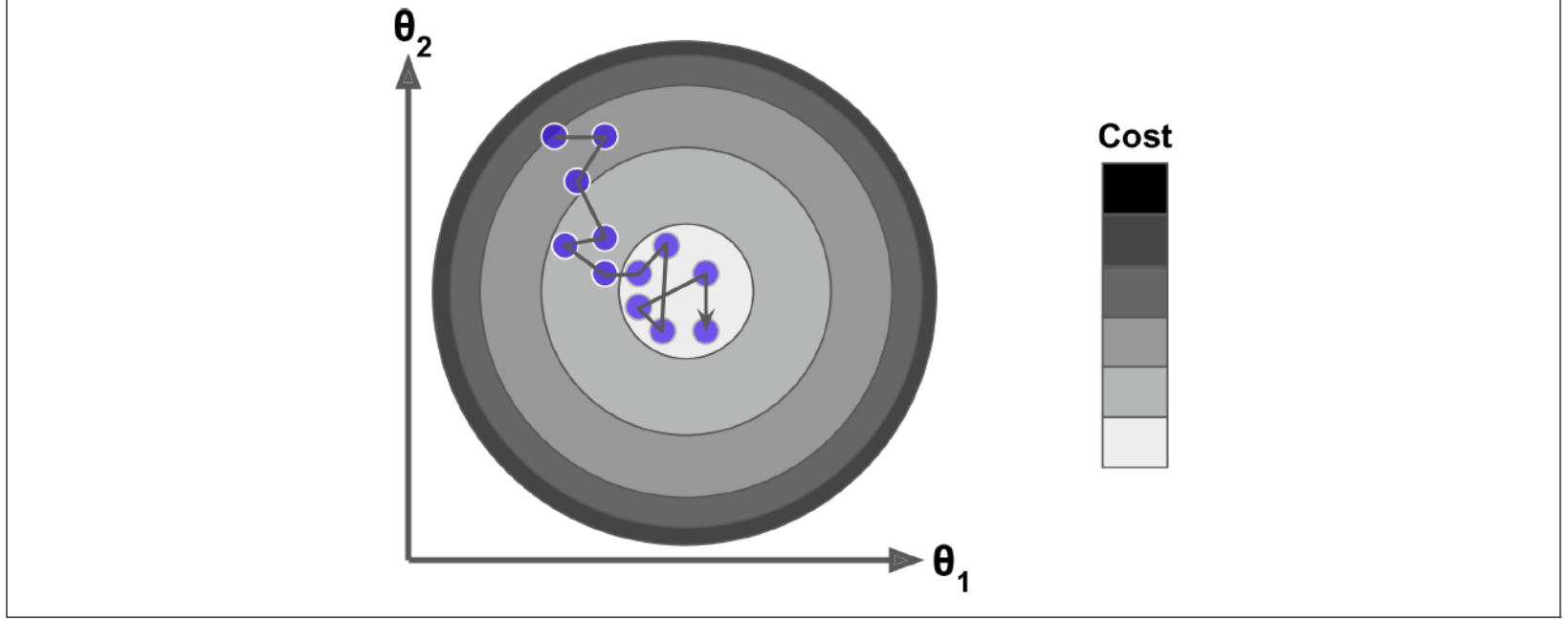


Figure 4-9. Stochastic Gradient Descent

- SGD can avoid local optima but may not settle at the minimum
- Simulated annealing
  - Start with large learning rate then gradually reduce it based on some function (learning schedule)
- Epoch
  - A round of *m* iterations

```
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
```

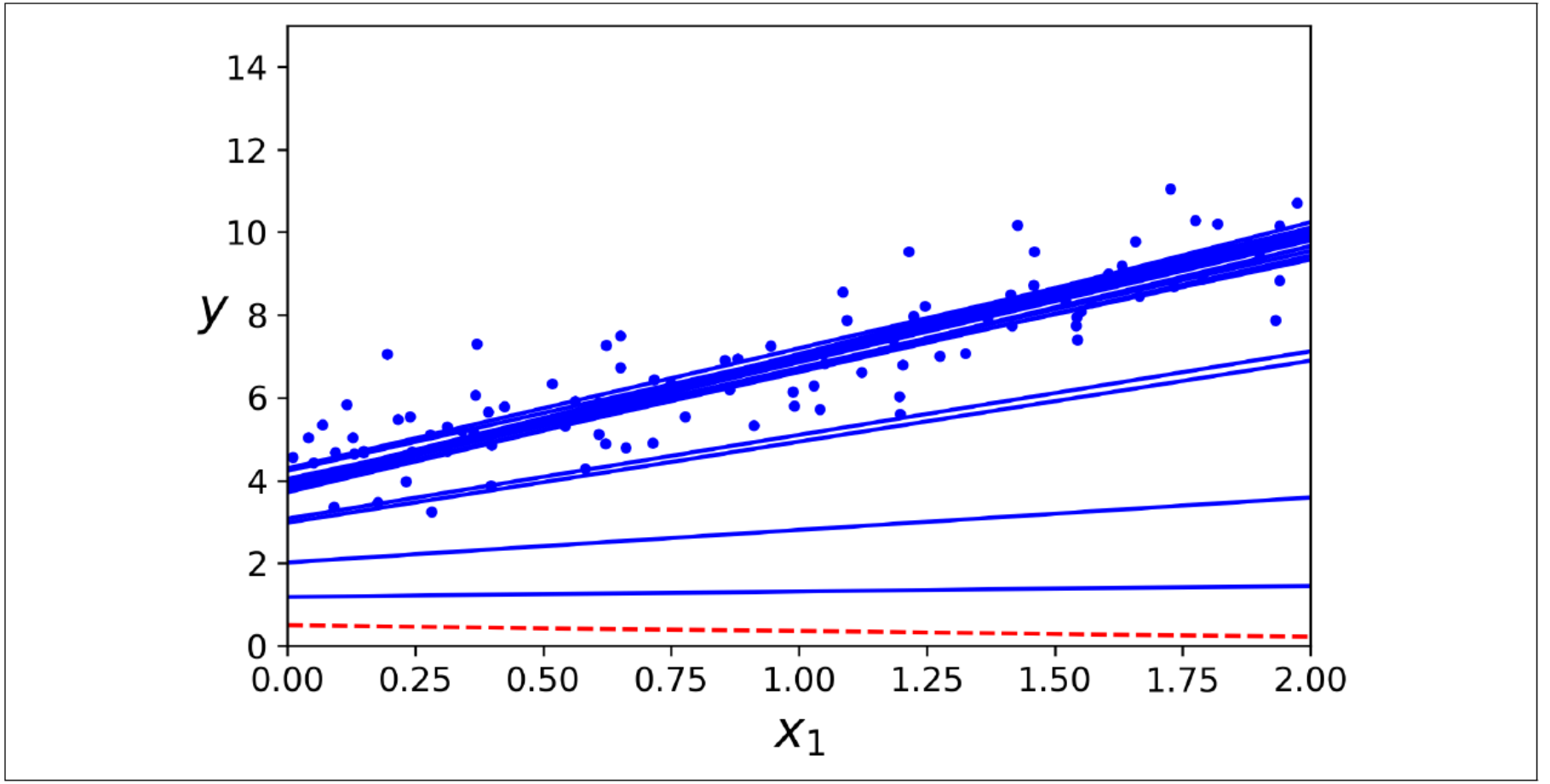


Figure 4-10. Stochastic Gradient Descent first 20 steps

Mini-batch Gradient Descent

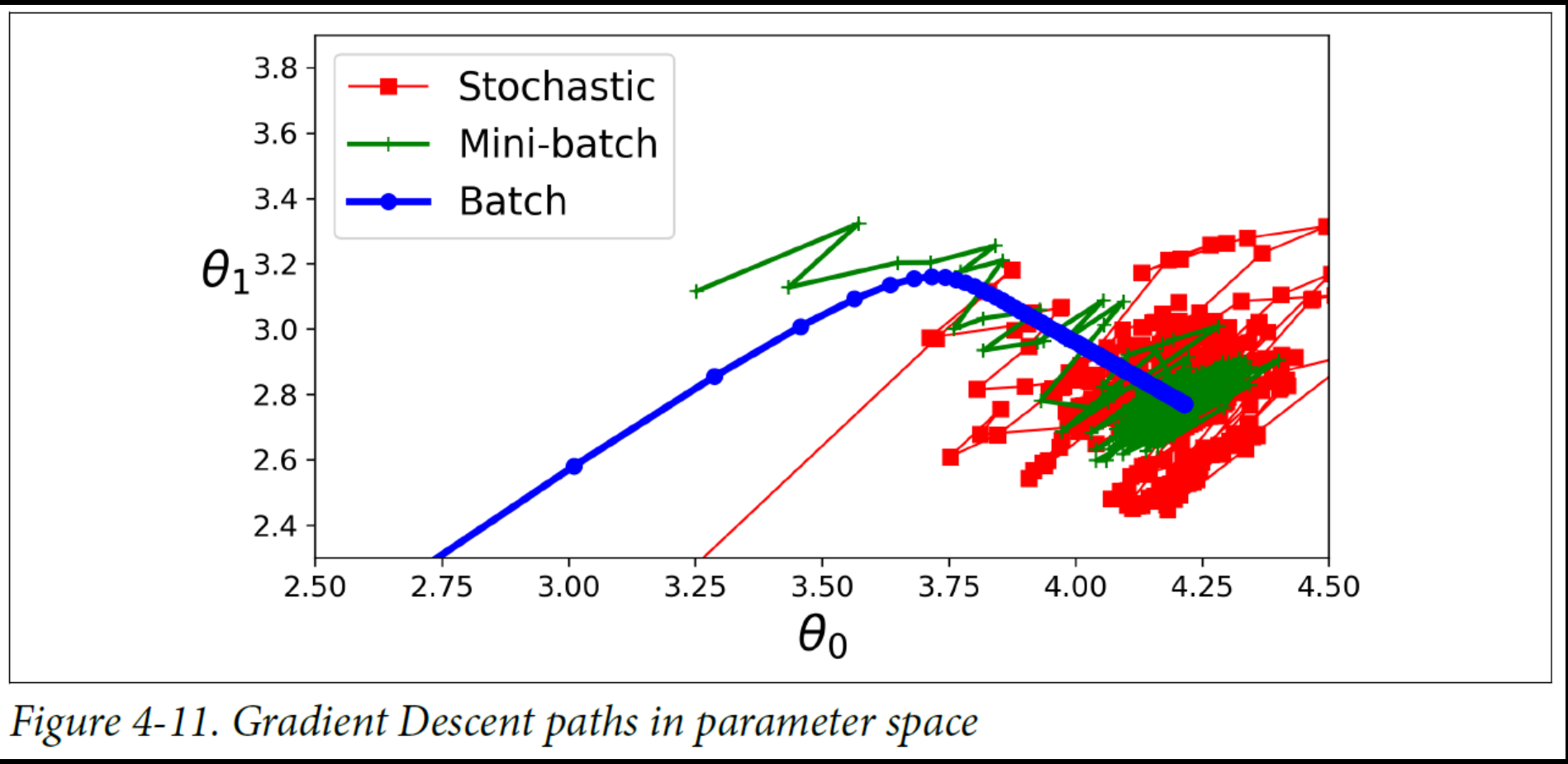


Figure 4-11. Gradient Descent paths in parameter space

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

- Data is complex that it can't be modeled by a straight line

$$y = 0.5x_1^2 + 1.0x_1 + 2.0 + \text{Gaussian noise}$$

$$\hat{y} = 0.56x_1^2 + 0.93x_1 + 1.78$$

  - Try quadratic equation

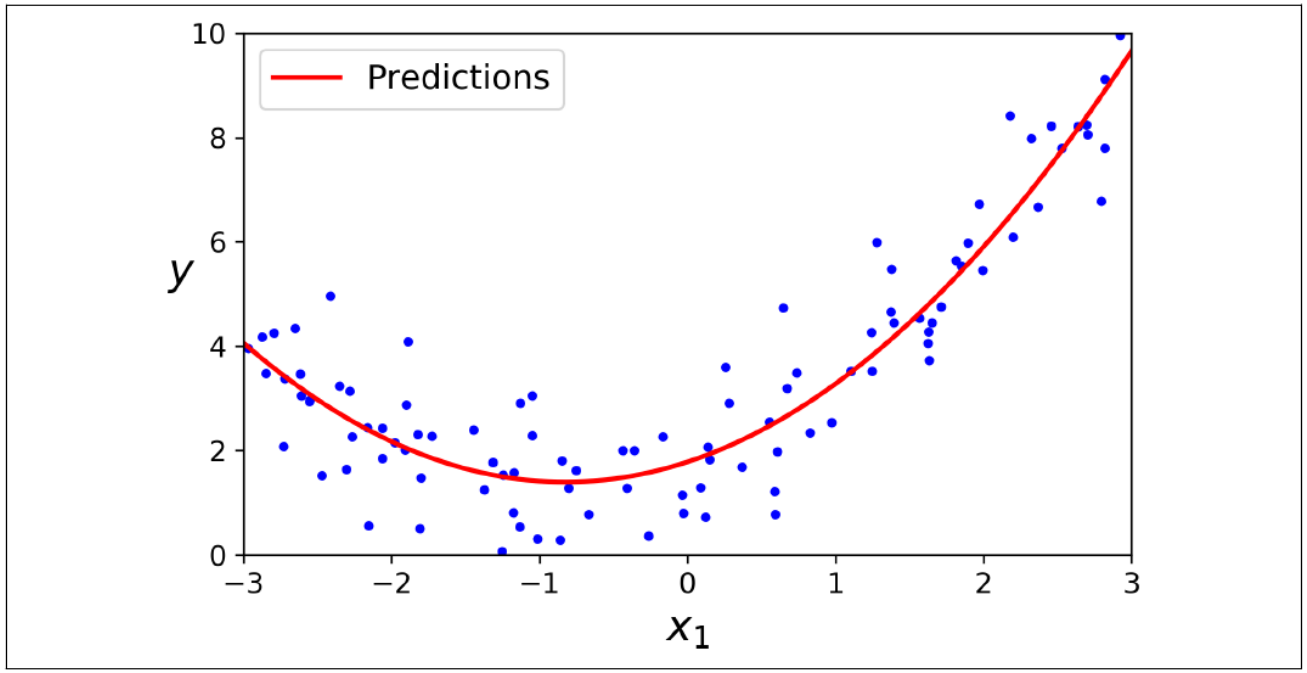


Figure 4-13. Polynomial Regression model predictions

High-degree Polynomial Regression

- What's the problem?
- Overfitting vs underfitting?

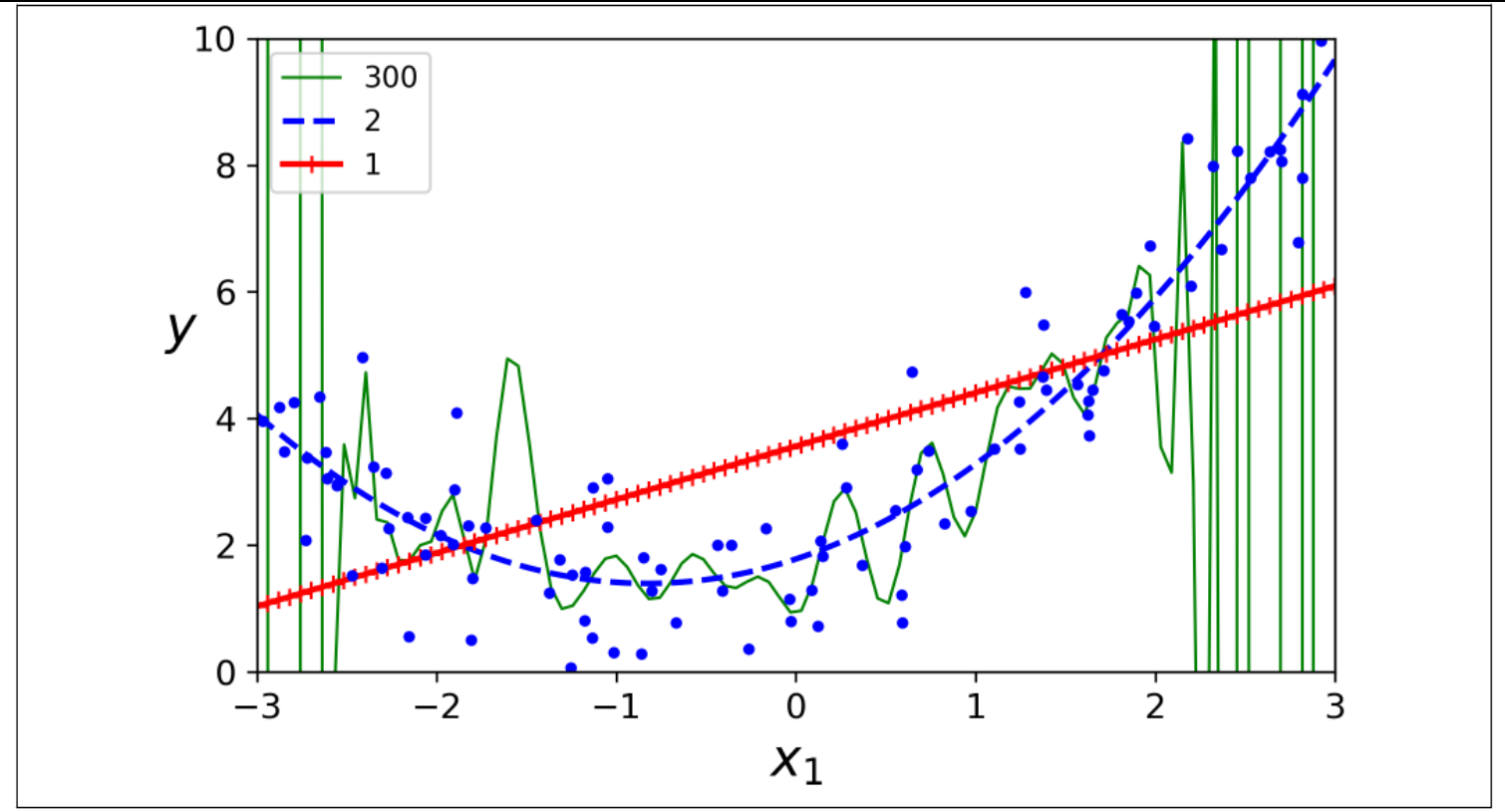


Figure 4-14. High-degree Polynomial Regression

Learning Curves

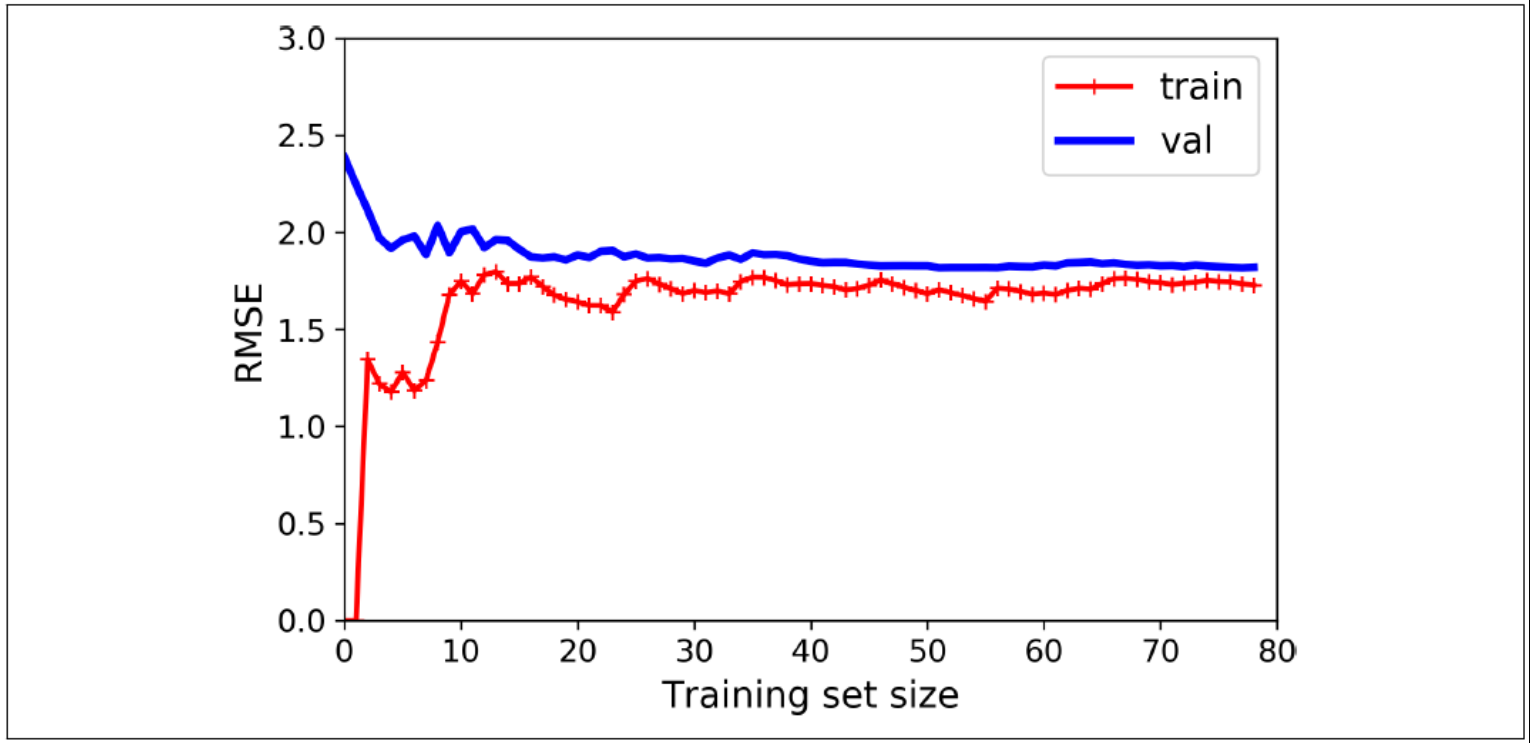


Figure 4-15. Learning curves

- If model is underfitting, adding more training examples will not help

Learning Curves

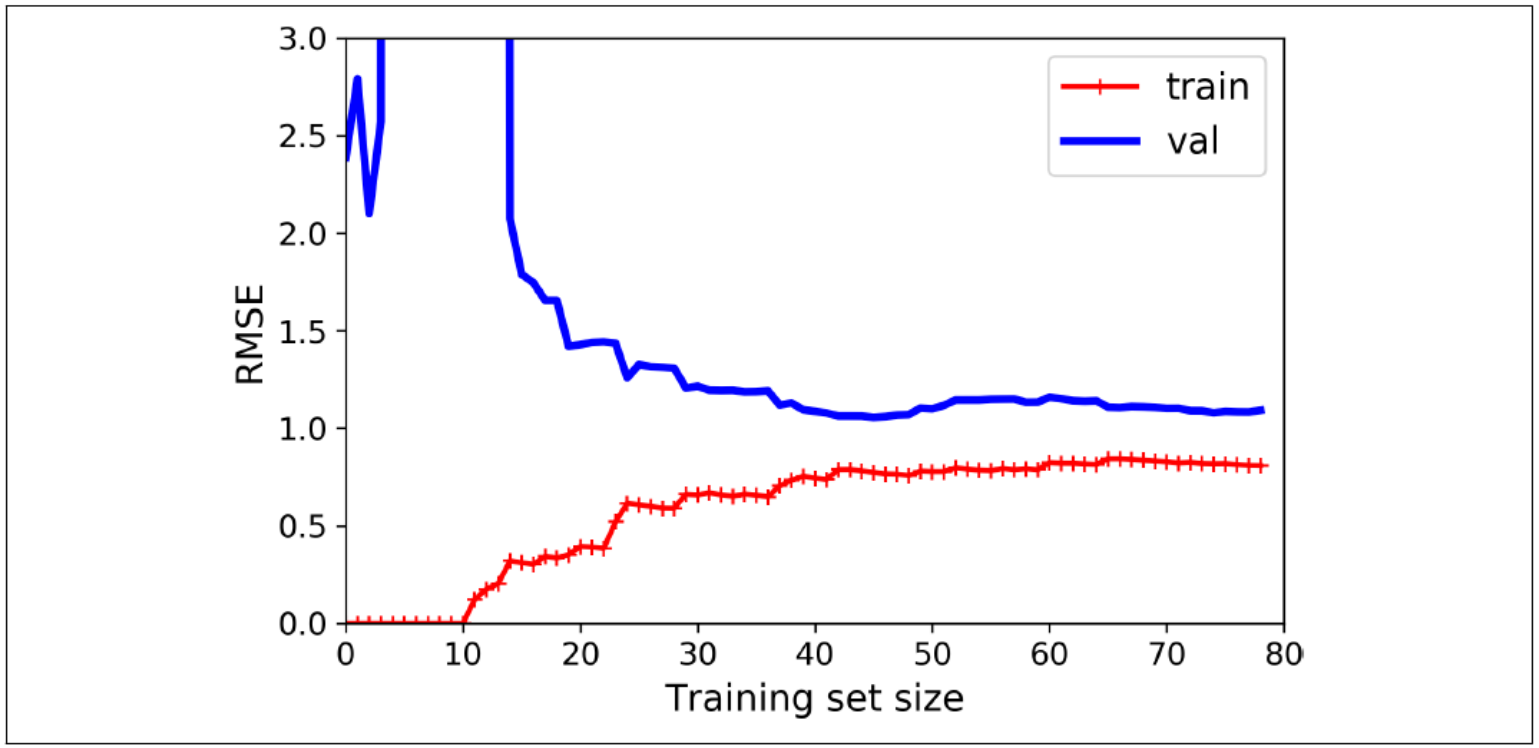


Figure 4-16. Learning curves for the polynomial model

- If model is overfitting, adding more training examples will help reduce the validation error

Bad Models: Overfitting & Underfitting

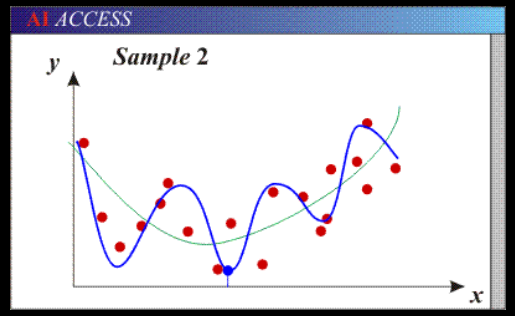
- Generalization
  - How well does a learned model generalize from the training data to new instances?
  - Causes of error: overfitting & underfitting
  - Overfitting: model is too complex
  - Underfitting: model is too simple

  - Bias: error from erroneous assumptions
  - Variance: error from being too sensitive to small fluctuations in training data
  - Irreducible error: noise error

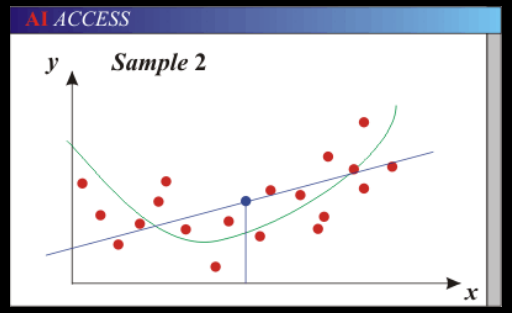
Bias & Variance Tradeoff

Error = noise + bias + variance

- Irreducible error
- Error due to incorrect assumptions
- Error due to variance of training



- Overfitting: model is too complex. Inaccuracy caused by large variance (too much sensitivity to training data). Low bias, high variance.



- Underfitting: model is too simple. Inaccuracy caused by large bias (not enough flexibility). High bias, low variance.

Regularization

- Reduces overfitting
  - Ridge Regression
  - Lasso Regression
  - Elastic Net

Equation 4-8. Ridge Regression cost function

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

Equation 4-10. Lasso Regression cost function

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

Equation 4-12. Elastic Net cost function

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

Ridge Regression

- aka Tikhonov regularization

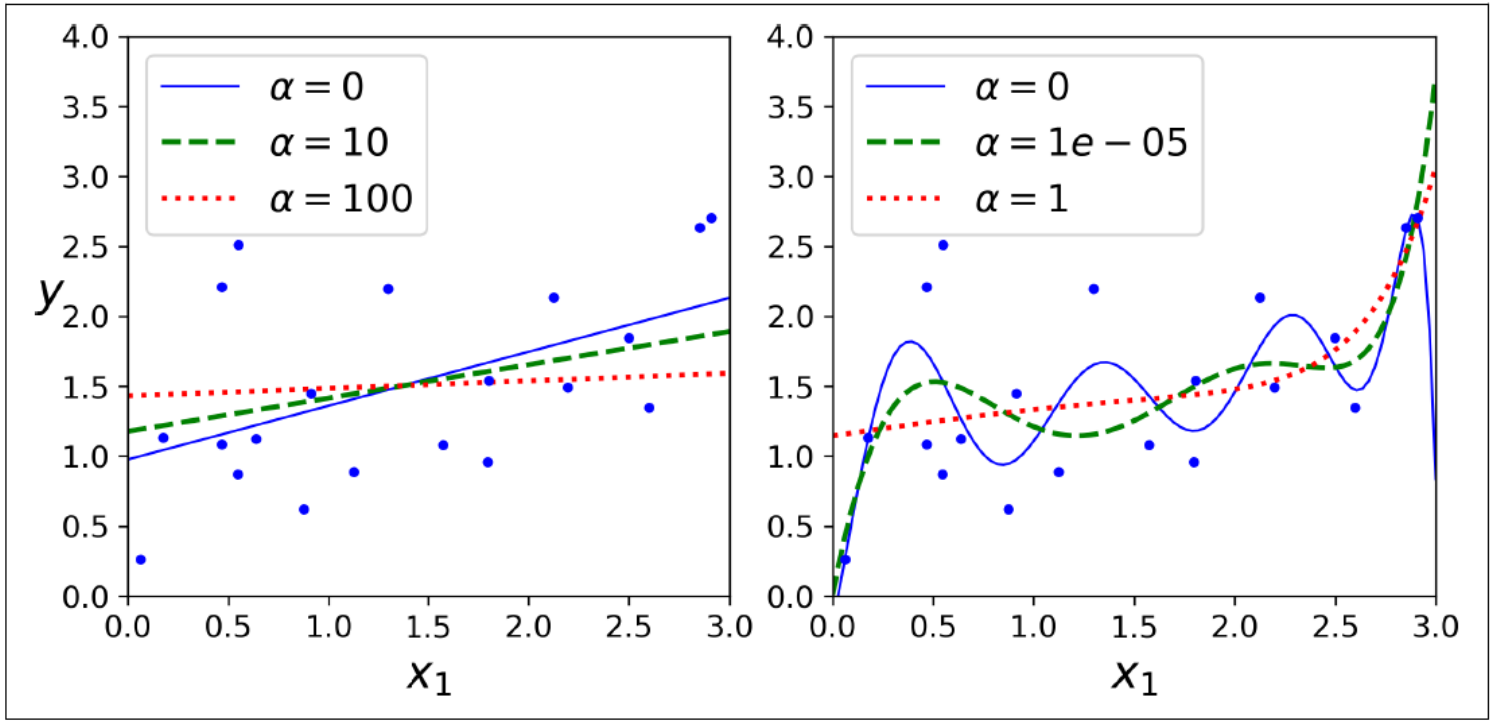


Figure 4-17. Ridge Regression

Equation 4-8. Ridge Regression cost function

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

Equation 4-9. Ridge Regression closed-form solution

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\intercal\mathbf{X} + \alpha\mathbf{A})^{-1}\ \mathbf{X}^\intercal\ \mathbf{y}$$

Lasso Regression

- Least Absolute Shrinkage and Selection Operator Regression

Equation 4-10. Lasso Regression cost function

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

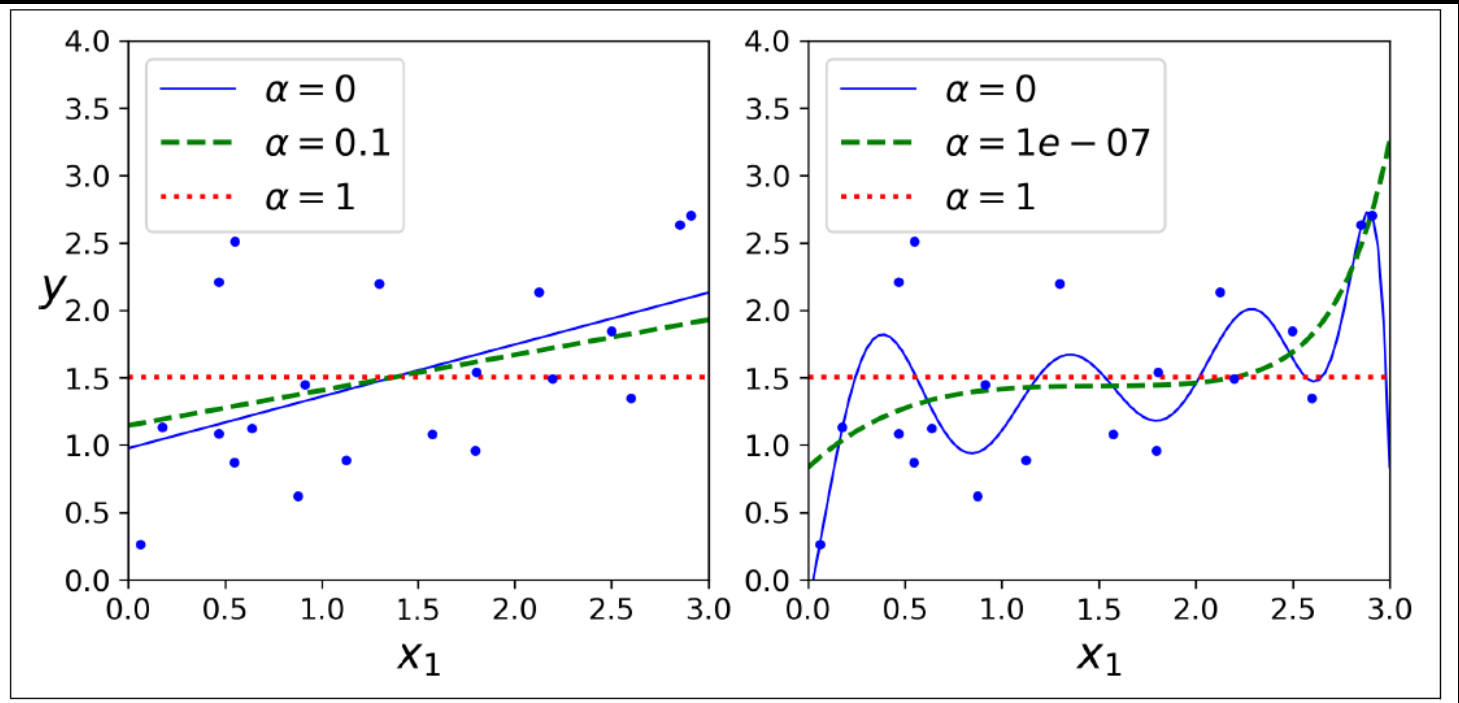


Figure 4-18. Lasso Regression

Differentiability of Lasso Regression Cost Function

- The cost function itself is not differentiable, but GD still works using a subgradient vector at a non-differentiable point as an intermediate vector between the gradient vectors around that point

Equation 4-11. Lasso Regression subgradient vector

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

Elastic Net

- Middle ground between Ridge and Lasso

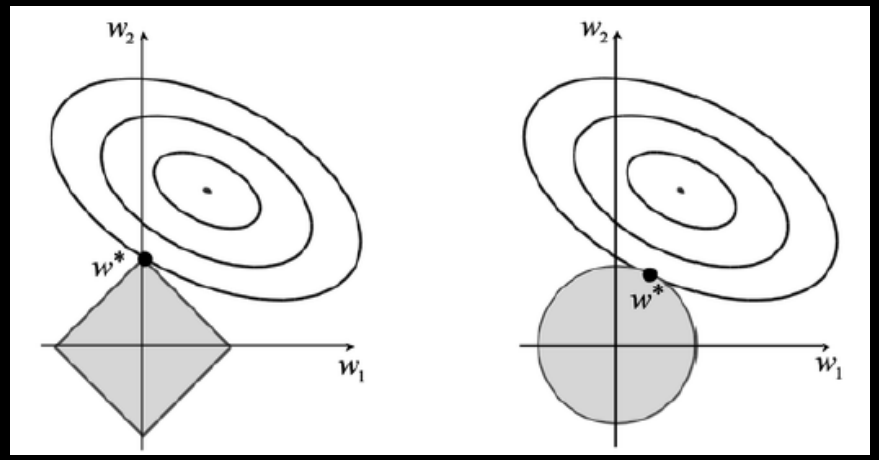
Equation 4-12. Elastic Net cost function

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

- Almost always want some regularization
- Use Ridge as default
- Use Lasso or Elastic Net for more aggressive regularization
- Elastic Net is preferred when n > m or if features are strongly correlated where n = number of features, m = number of instances

The Geometry Behind Regularization

- Ridge Regularization constraints θ_i^2 to be in a hypersphere of radius less than B, which is determined by the choice of α
- Lasso Regression constraints $|\theta_i|$ to be in a hypercube of side length B



- *Machine Learning: A Probabilistic Perspective* by K. Murphy
- *Pattern Recognition and Machine Learning* by C. Bishop

Early Stopping

- Stop training when the validation error reaches a minimum

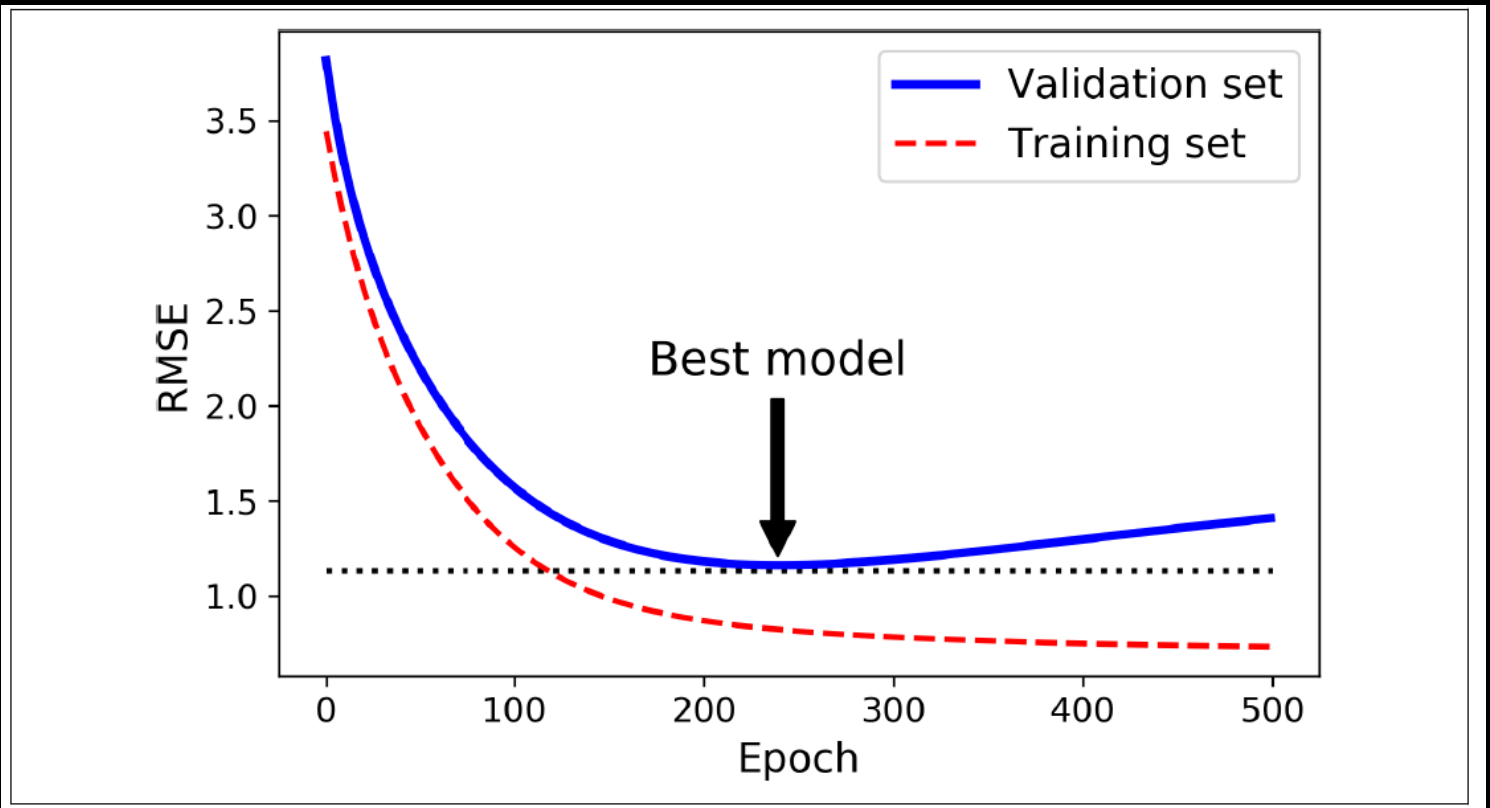


Figure 4-20. Early stopping regularization

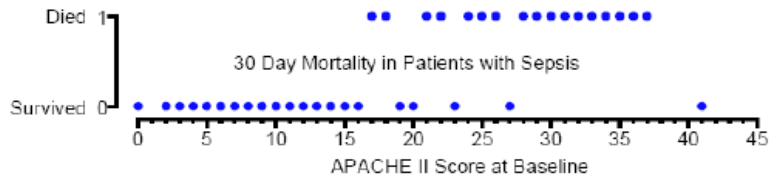
Logistic Regression

- A binary classifier based on a logistic (sigmoid) function

Logistic regression in one dimension

a) Example: APACHE II Score and Mortality in Sepsis

The following figure shows 30 day mortality in a sample of septic patients as a function of their baseline APACHE II Score. Patients are coded as 1 or 0 depending on whether they are dead or alive in 30 days, respectively.



Logistic Regression

- A binary classifier based on a logistic (sigmoid) function

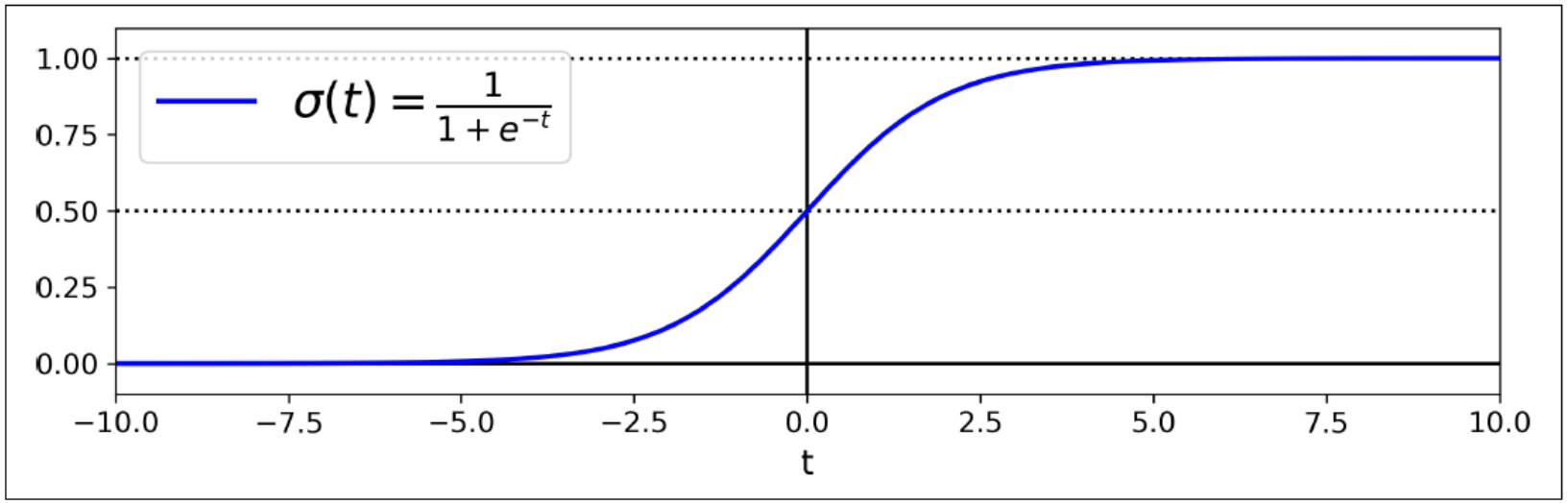


Figure 4-21. Logistic function

Equation 4-15. Logistic Regression model prediction

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

Training and Cost Function

- There is no closed-form equation

Equation 4-16. Cost function of a single training instance

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

Equation 4-17. Logistic Regression cost function (log loss)

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

Equation 4-18. Logistic cost function partial derivatives

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

Decision Boundaries

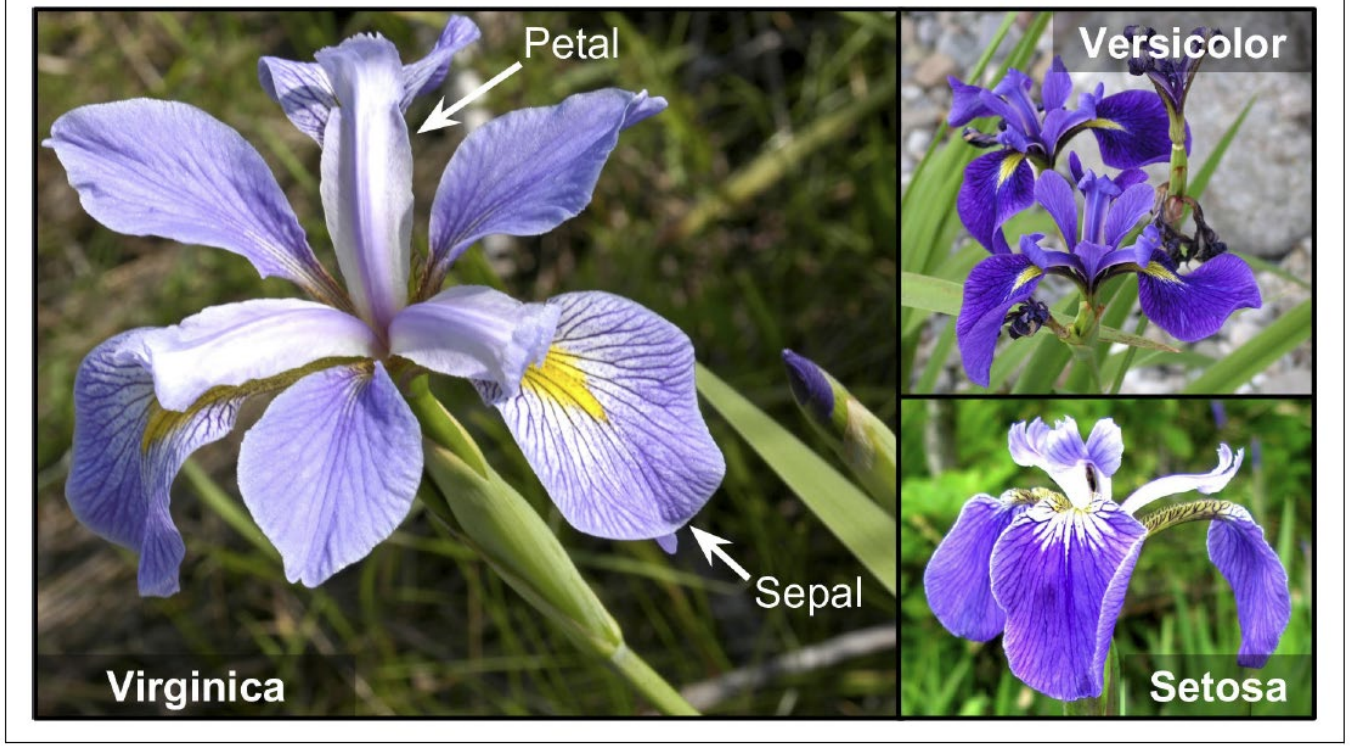


Figure 4-22. Flowers of three iris plant species[16]

- Using the Iris dataset, detect Iris-Virginica based on the petal width feature

Decision Boundary

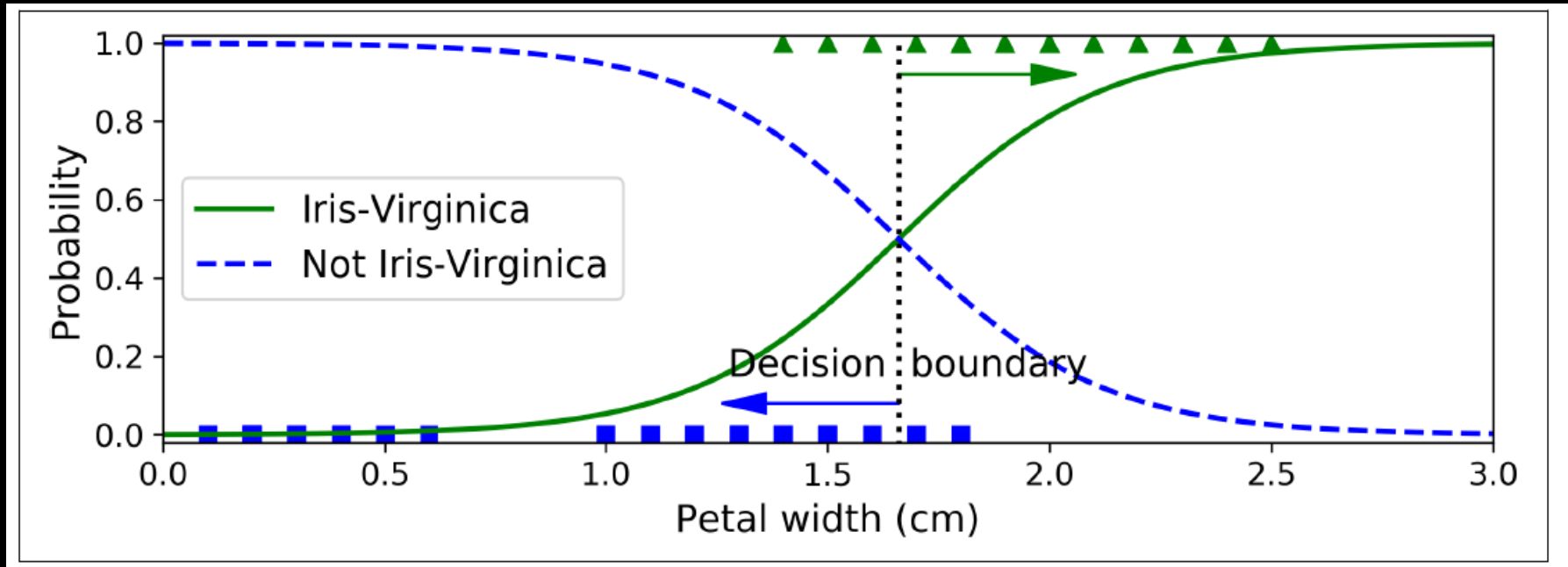


Figure 4-23. Estimated probabilities and decision boundary

Linear Decision Boundary

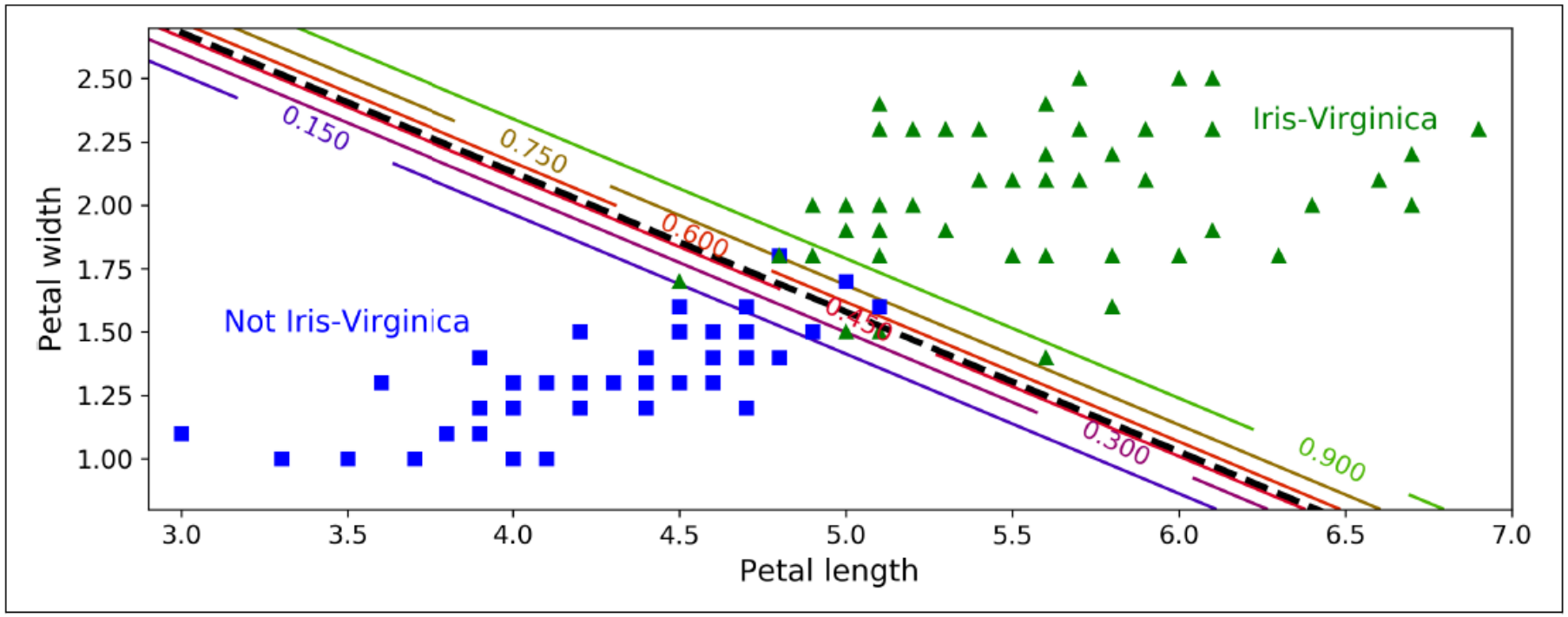


Figure 4-24. Linear decision boundary

- Probabilities for Iris-Virginica

Softmax Regression

- aka Multinomial Logistic Regression, Maximum Entropy Classifier
- Generalization of Logistic Regression for multi-class classification
- For prediction, compute a score for each class:

Equation 4-19. Softmax score for class k

$$s_k(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\theta}^{(k)}$$

- Estimate the probability of each class:

Equation 4-20. Softmax function

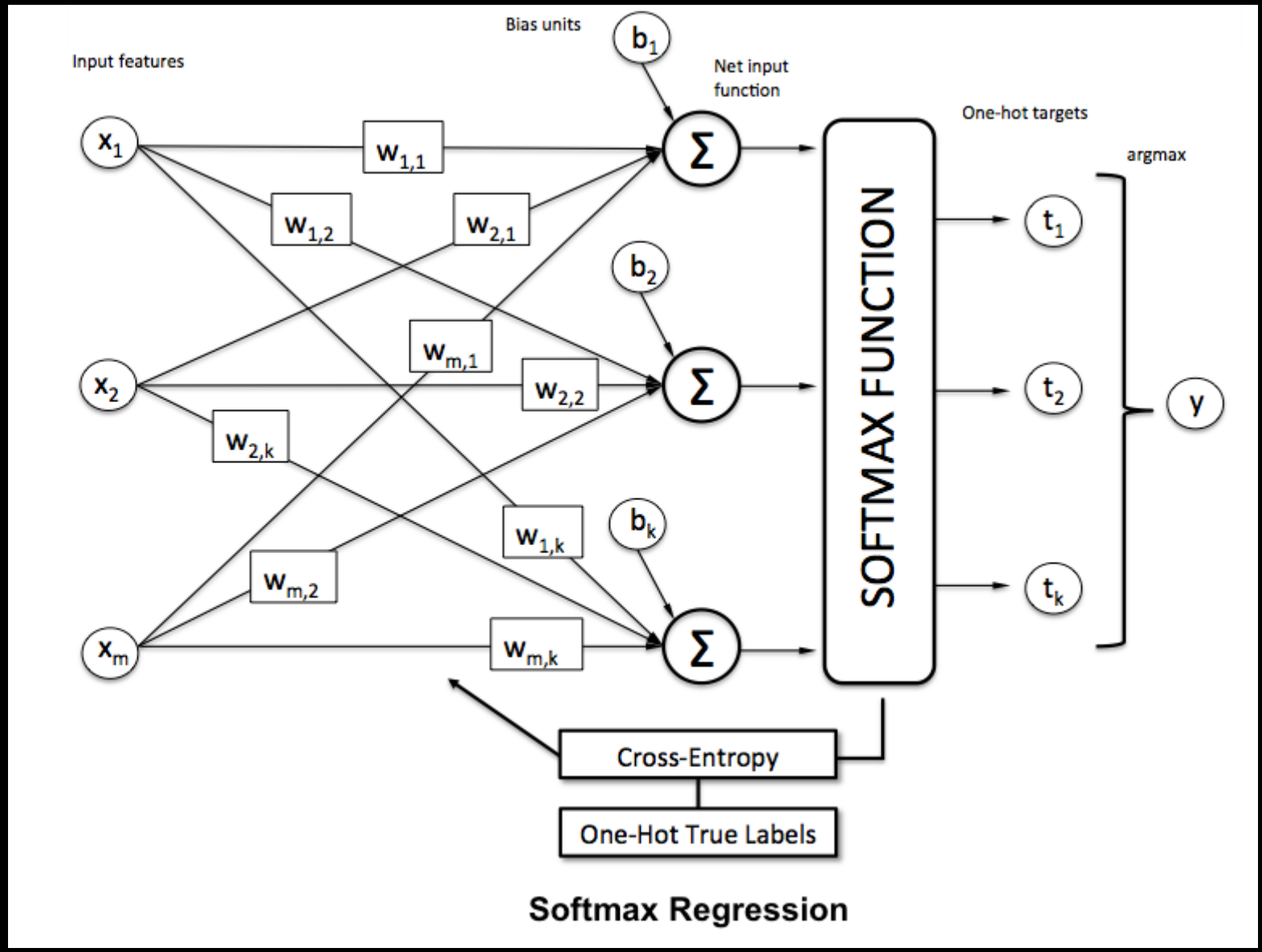
$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

- Predicted class is the class with the highest score:

Equation 4-21. Softmax Regression classifier prediction

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\boldsymbol{\theta}^{(k)}\right)^T \mathbf{x}\right)$$

Architecture



- Courtesy of S. Raschka

- For training, use cross entropy as the cost function:

Equation 4-22. Cross entropy cost function

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

  - Where *m* = number of instances, *k* = number of classes

- Compute the gradient vector for every class:

Equation 4-23. Cross entropy gradient vector for class k

$$\nabla_{\boldsymbol{\theta}^{(k)}} J(\mathbf{\Theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$$

- Use GD to the parameter matrix $\mathbf{\Theta}$ that minimizes the cost function

- Cross entropy originated from information theory invented by Claude Shannon

Decision Boundaries

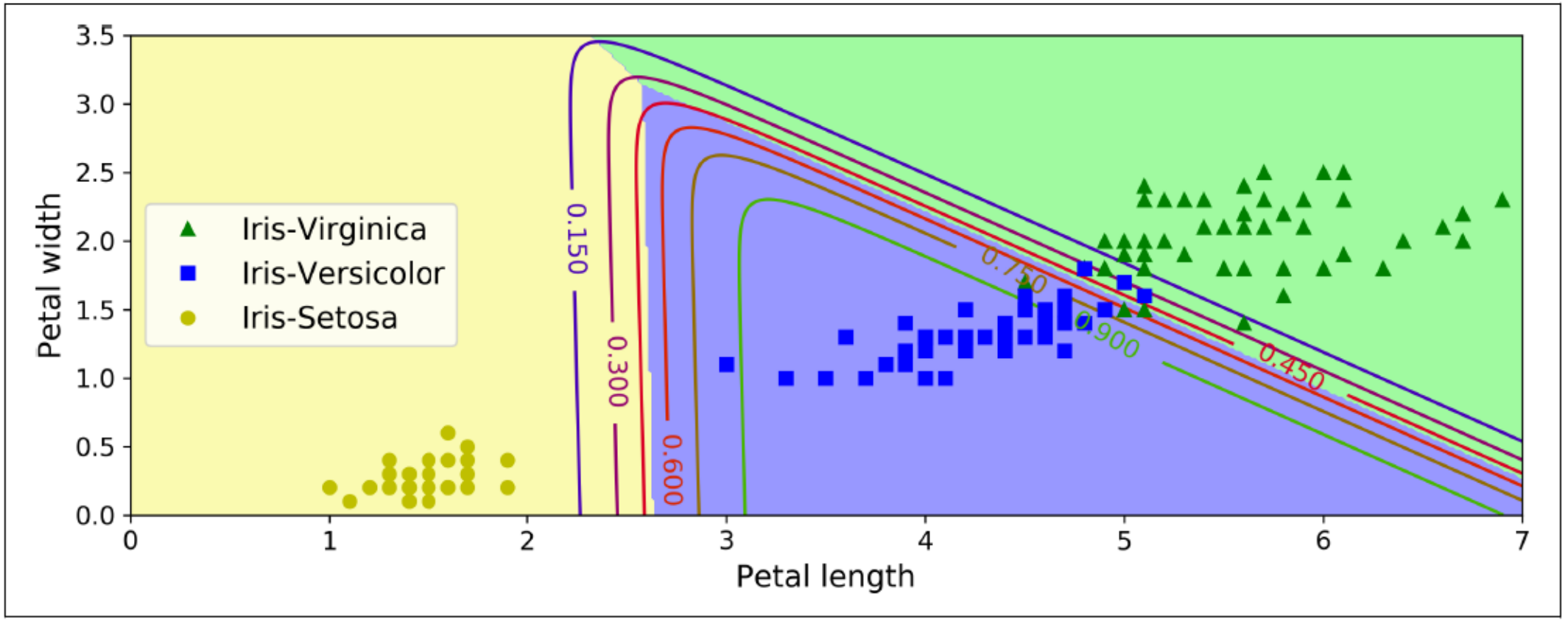


Figure 4-25. Softmax Regression decision boundaries

- Probabilities for Iris-Versicolor