Co-funded by the
Erasmus+ Programme
of the European Union

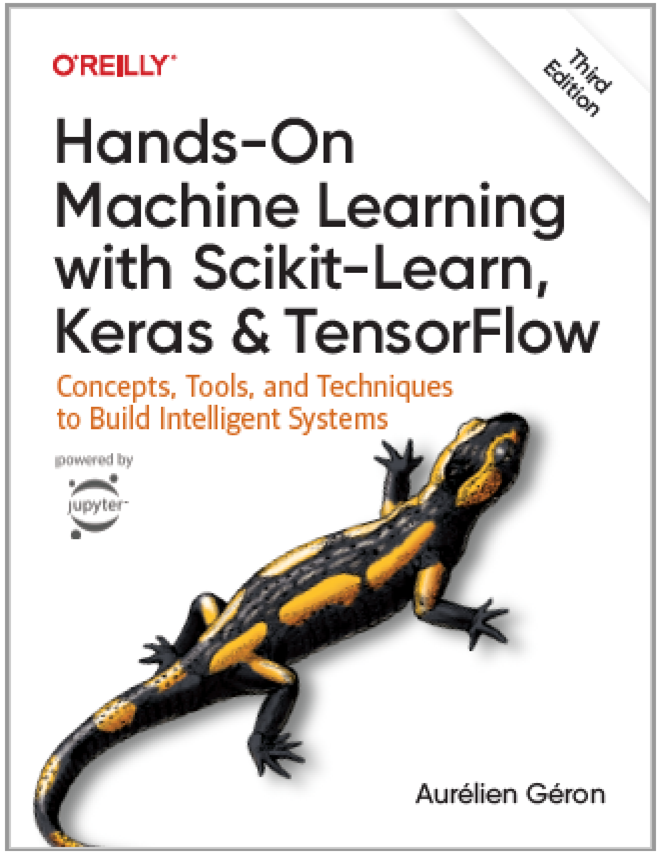
DeCAIR

Training Models and Regression

Prof. Gheith Abandah

Developing Curricula for Artificial Intelligence and Robotics (DeCAIR)
618535-EPP-1-2020-1-JO-EPPKA2-CBHE-JP

Reference



- Chapter 4: **Training Models**

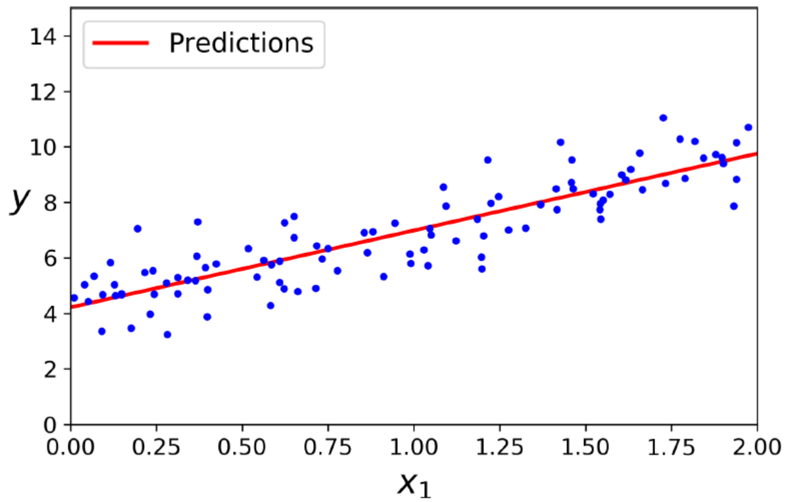
- Aurélien Géron, **Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow**, O'Reilly, 3rd Edition, 2022
    - Material: https://github.com/ageron/handson-ml3

Outline

1. Linear Regression
2. Gradient Descent
3. Gradient Descent Variants
    1. Batch Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-batch Gradient Descent
4. Polynomial Regression
5. Learning Curves
6. Regularized Linear Models
7. Logistic Regression
8. Exercises

Linear Regression

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



$$\boxed{life\_satisfaction = \theta_0 + \theta_1 \times GDP\_per\_capita}$$

- $\hat{y}$ 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 $\theta_1, \theta_2, \cdots, \theta_n$).

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

Analytical Solution

- The Root Mean Square Error (RMSE) is used as **cost function**.

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

- Minimizing this cost gives the following solution (**normal function**):

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

Complexity $O(mn^2)$

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

Example

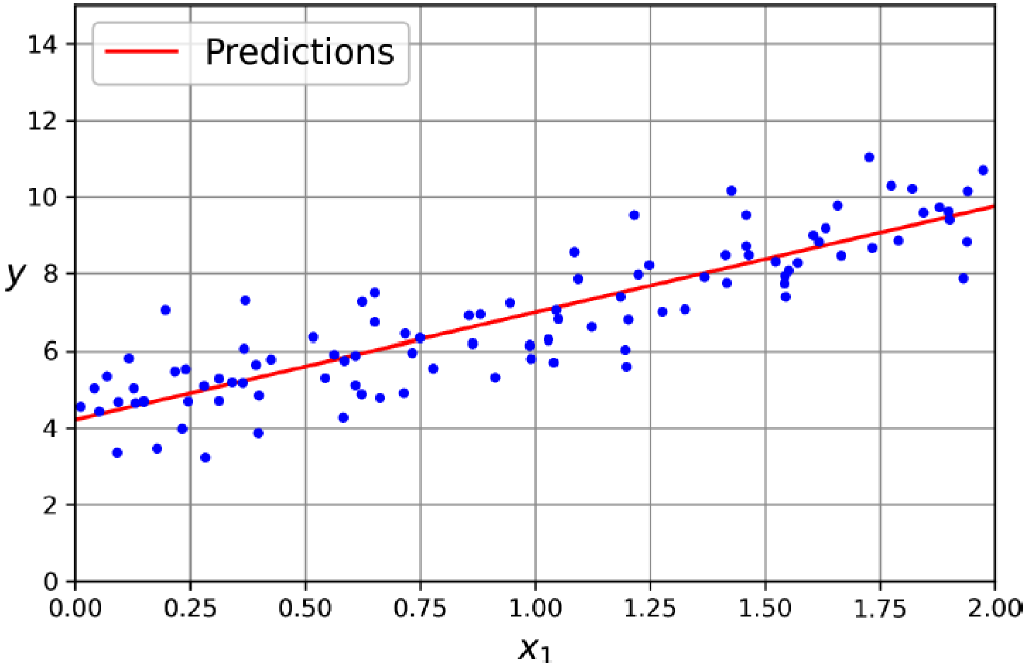
Better Solution $O(n^2)$:

```
>>> from sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X, y)
```

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

```
X_b = add_dummy_feature(X)  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

Adds a column of 1s



Now we can make predictions using $\hat{\theta}$:

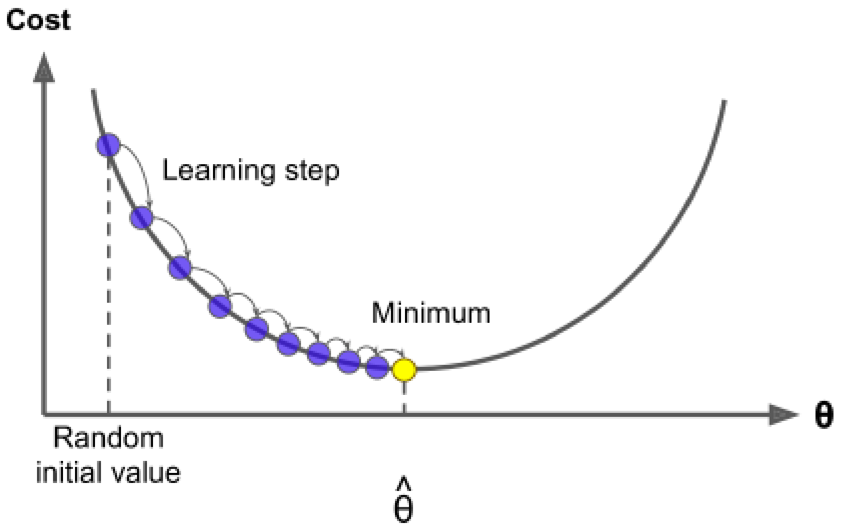
```
>>> X_new = np.array([[0], [2]])
>>> X_new_b = add_dummy_feature(X_new)
>>> y_predict = X_new_b @ theta_best
>>> y_predict
array([[4.21509616],
       [9.75532293]])
```

Outline

1. Linear Regression
2. Gradient Descent
3. Gradient Descent Variants
    1. Batch Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-batch Gradient Descent
4. Polynomial Regression
5. Learning Curves
6. Regularized Linear Models
7. Logistic Regression
8. Exercises

Gradient Descent

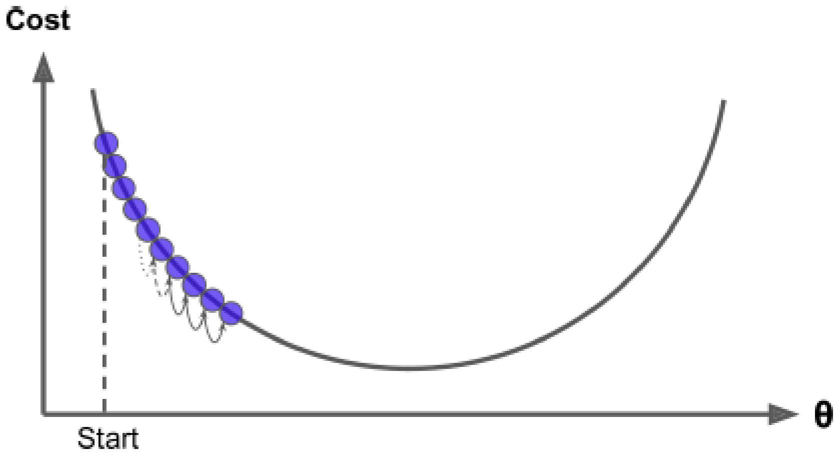
- **Generic optimization algorithm** capable of finding optimal solutions to a wide range of problems.
- **Tweaks parameters** iteratively in order **to minimize a cost function**.



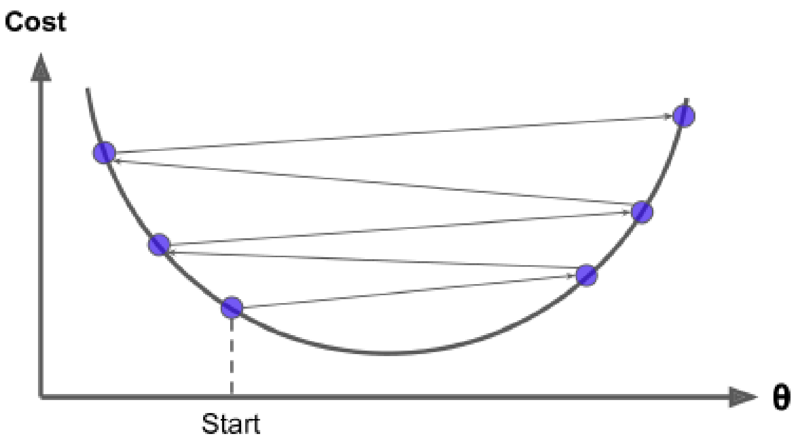
$$\boxed{\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \mathrm{MSE}(\theta)}$$

Learning Rate η

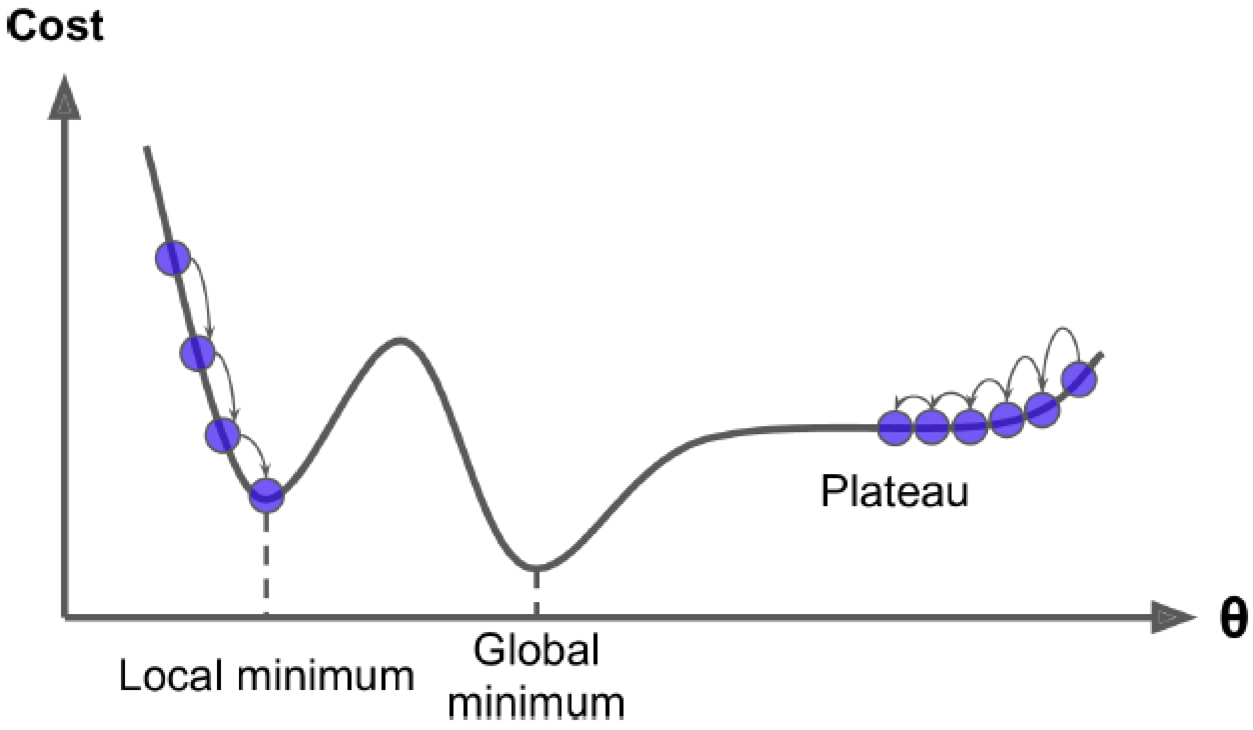
Too Small



Too Large

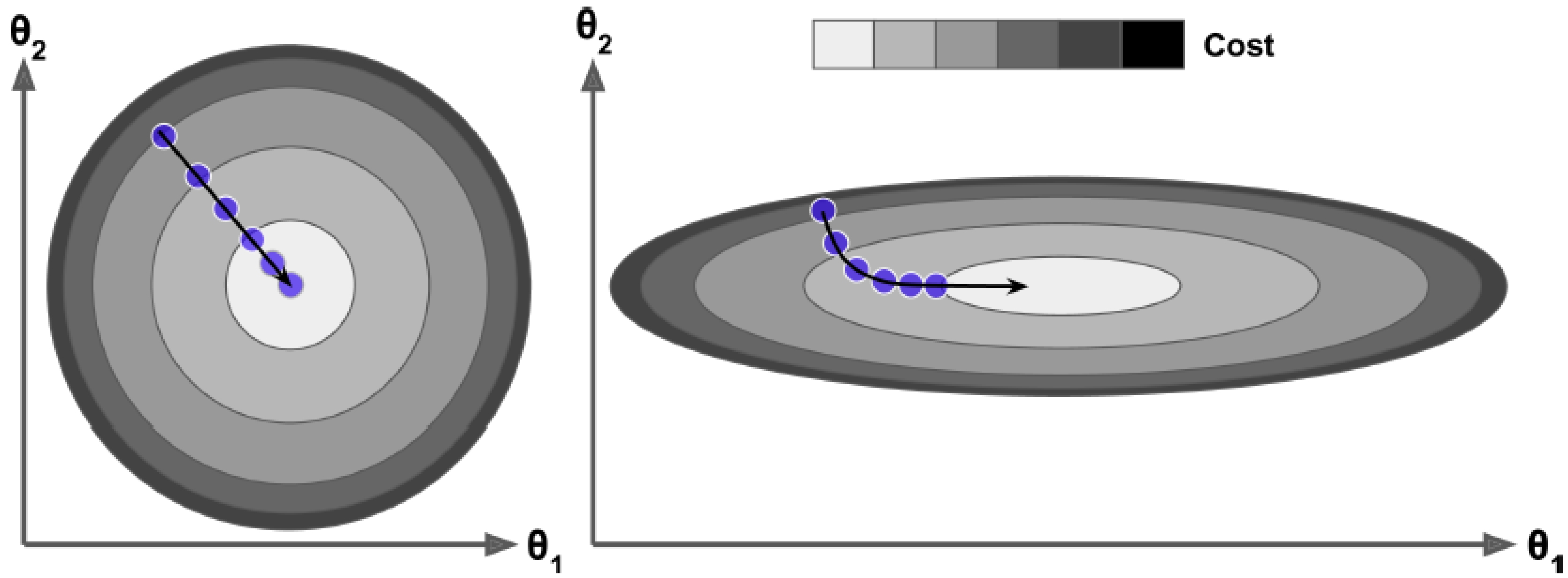


Gradient Descent Pitfalls



Feature Scaling

- Ensure that all features have a similar scale (*e.g.*, using Scikit-Learn's `StandardScaler` class).
- Gradient Descent with and without feature scaling.



Outline

1. Linear Regression
2. Gradient Descent
3. Gradient Descent Variants
    1. Batch Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-batch Gradient Descent
4. Polynomial Regression
5. Learning Curves
6. Regularized Linear Models
7. Logistic Regression
8. Exercises

Batch Gradient Descent

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

- **Partial derivatives** of the cost function in θ_j

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

- **Gradient vector** of the cost function

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

The entire training Batch

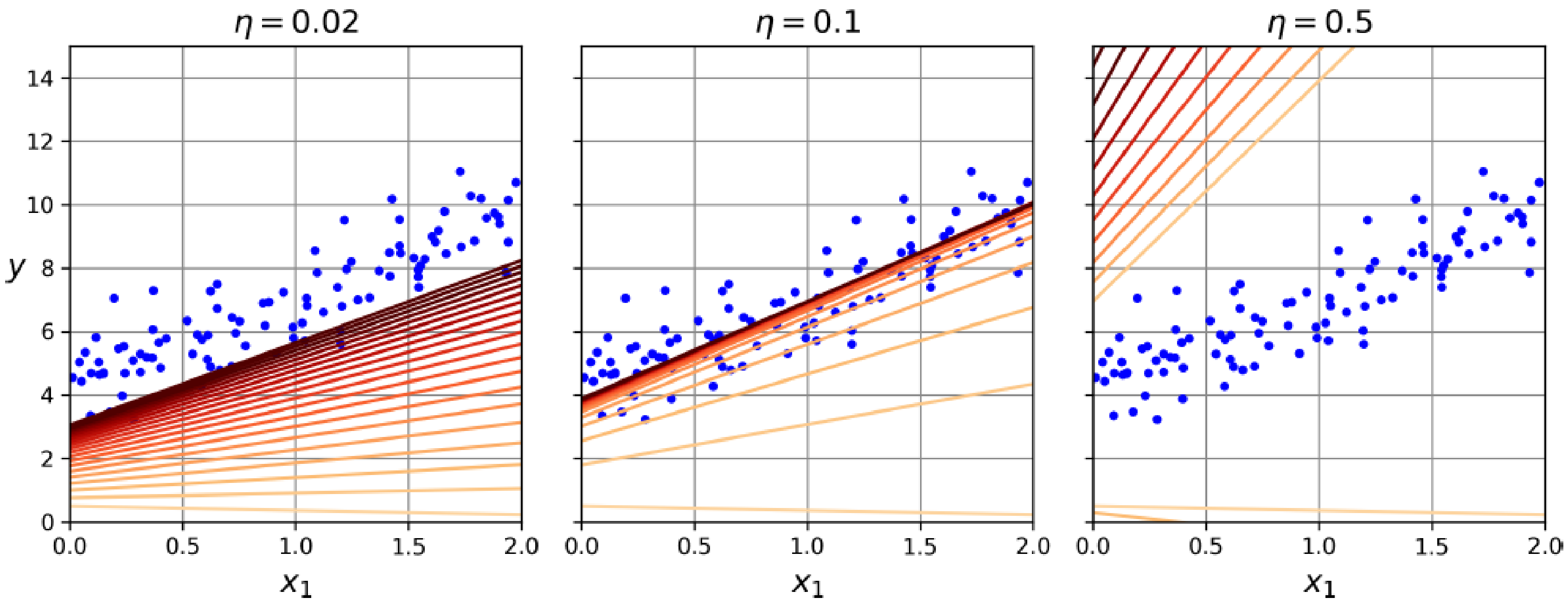
$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta\,\nabla_{\boldsymbol{\theta}}\,\text{MSE}(\boldsymbol{\theta})$$

Batch Gradient Descent

- **Gradient Descent step**

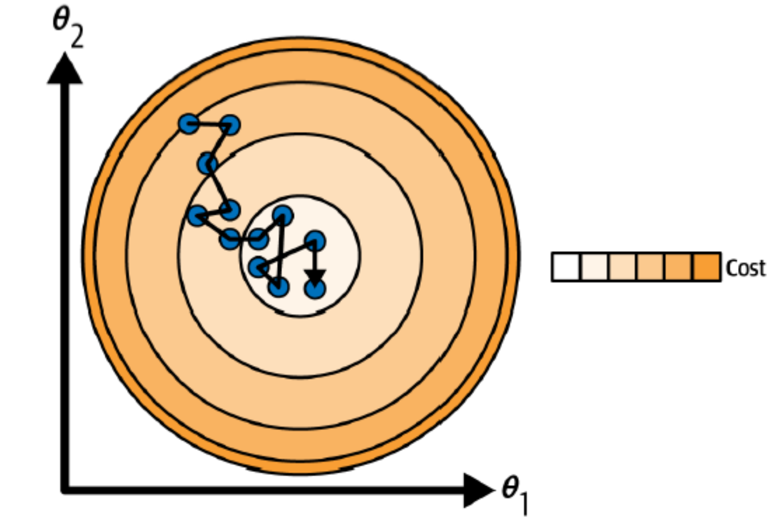
$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$

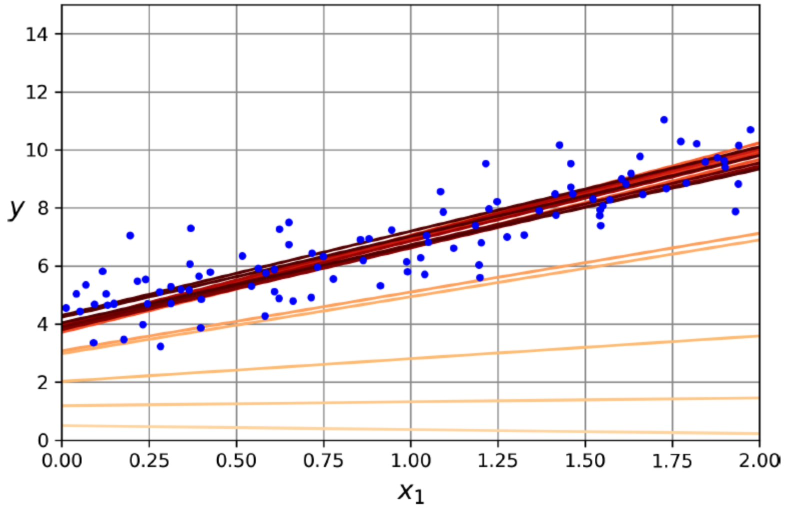
- Gradient Descent with **various learning rates**



Stochastic Gradient Descent

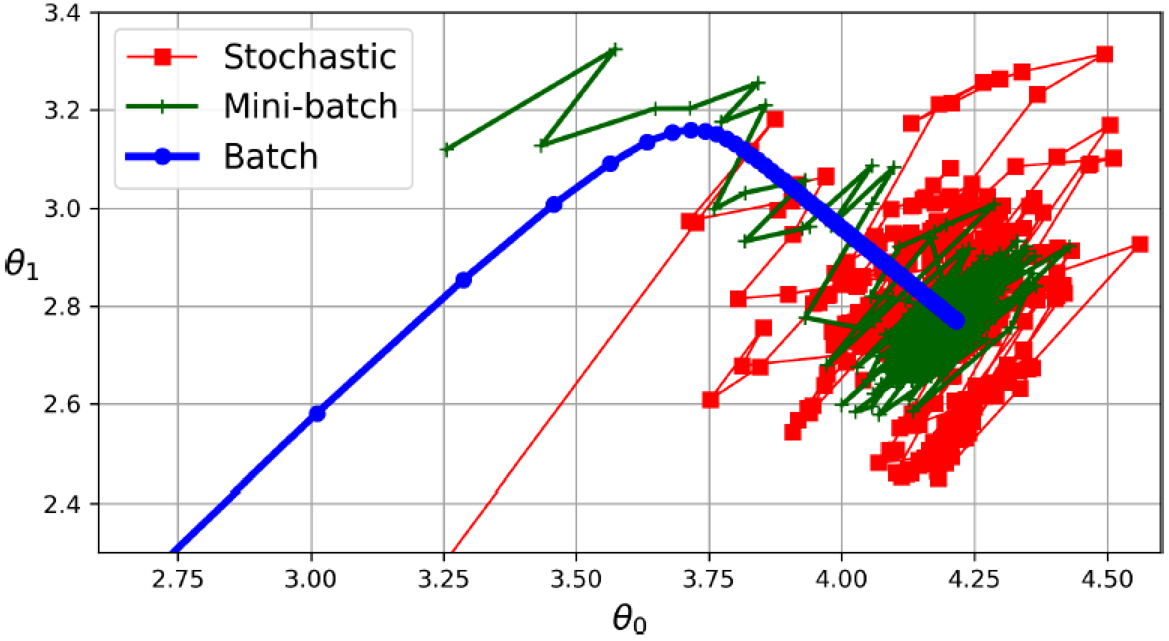
- SGD **picks a random instance** in the training set at every step and computes the gradients.
- SGD is **faster** when the training set is large.
- Is **bouncy**
- Eventually gives **good solution**
- Can **escape local minima**





Mini-batch Gradient Descent

- Computes the gradients on small random sets of instances called **mini batches**.
- Benefits from **hardware accelerators** (*e.g.*, GPU).
- **Less bouncy**, **better solution**, **escapes some local minima**

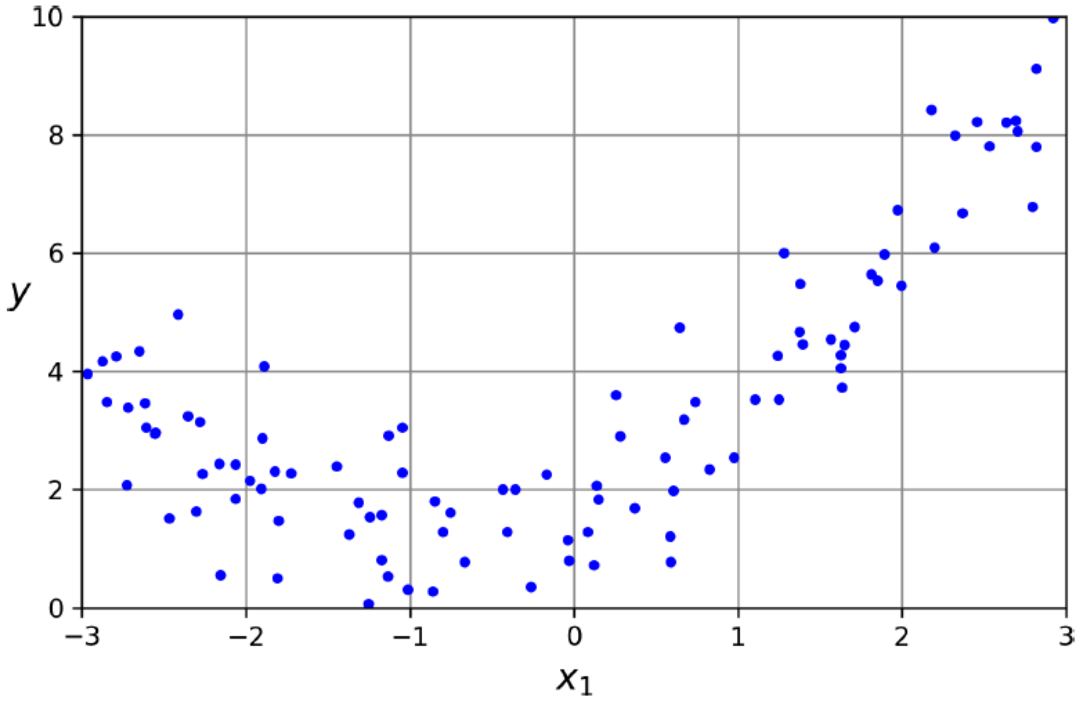


Outline

1. Linear Regression
2. Gradient Descent
3. Gradient Descent Variants
    1. Batch Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-batch Gradient Descent
4. Polynomial Regression
5. Learning Curves
6. Regularized Linear Models
7. Logistic Regression
8. Exercises

Polynomial Regression

- The shown data cannot be accurately modeled using linear regression.
- We can **use a linear model to fit nonlinear data**.
- Can try **polynomial regression** of degree 2 by adding the **feature squared**.



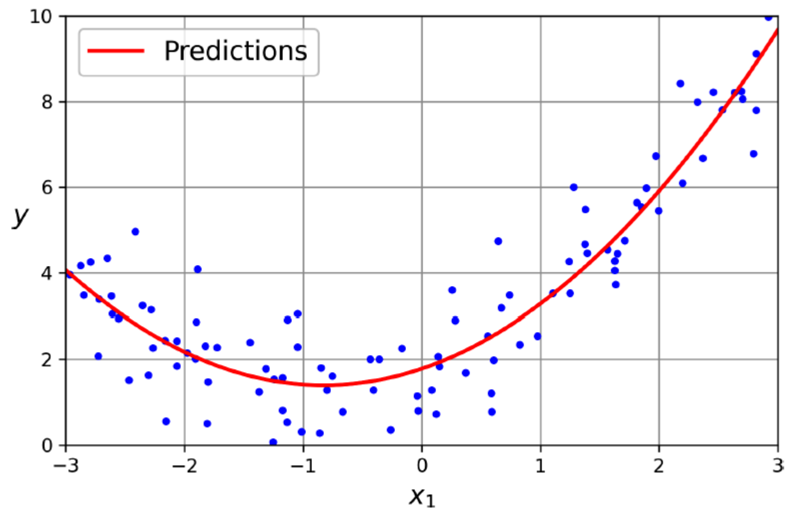
Polynomial Regression

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly_features = PolynomialFeatures(degree=2, include_bias=False)
>>> X_poly = poly_features.fit_transform(X)
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929,  0.56664654])
```

- Then **fit a linear regression model**.

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```

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

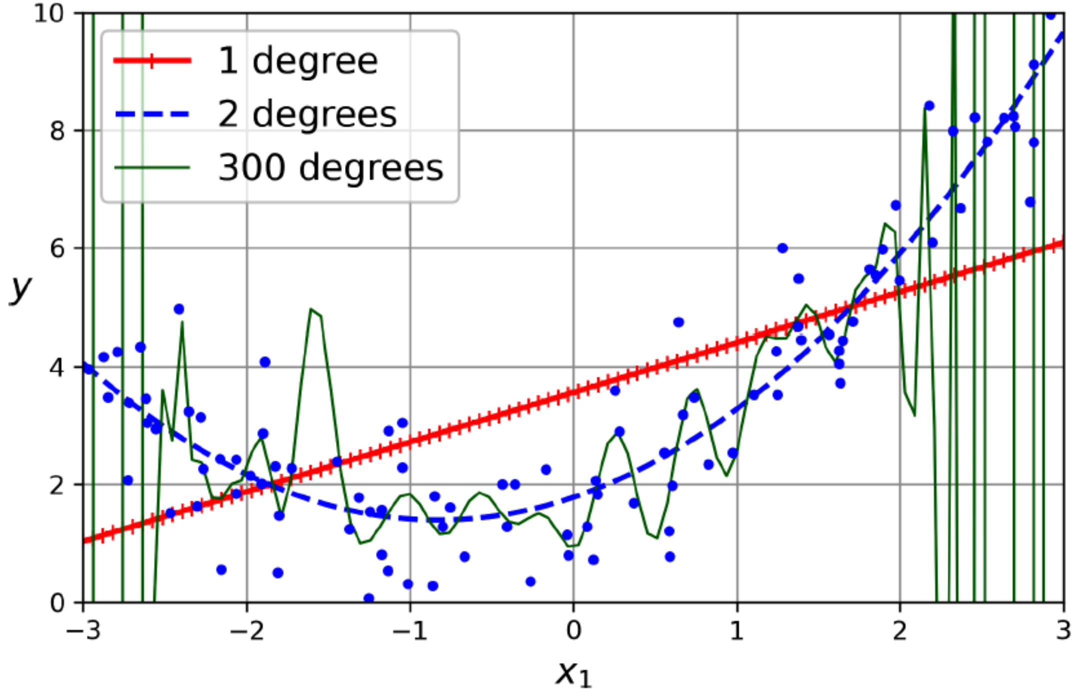


Outline

1. Linear Regression
2. Gradient Descent
3. Gradient Descent Variants
    1. Batch Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-batch Gradient Descent
4. Polynomial Regression
5. Learning Curves
6. Regularized Linear Models
7. Logistic Regression
8. Exercises

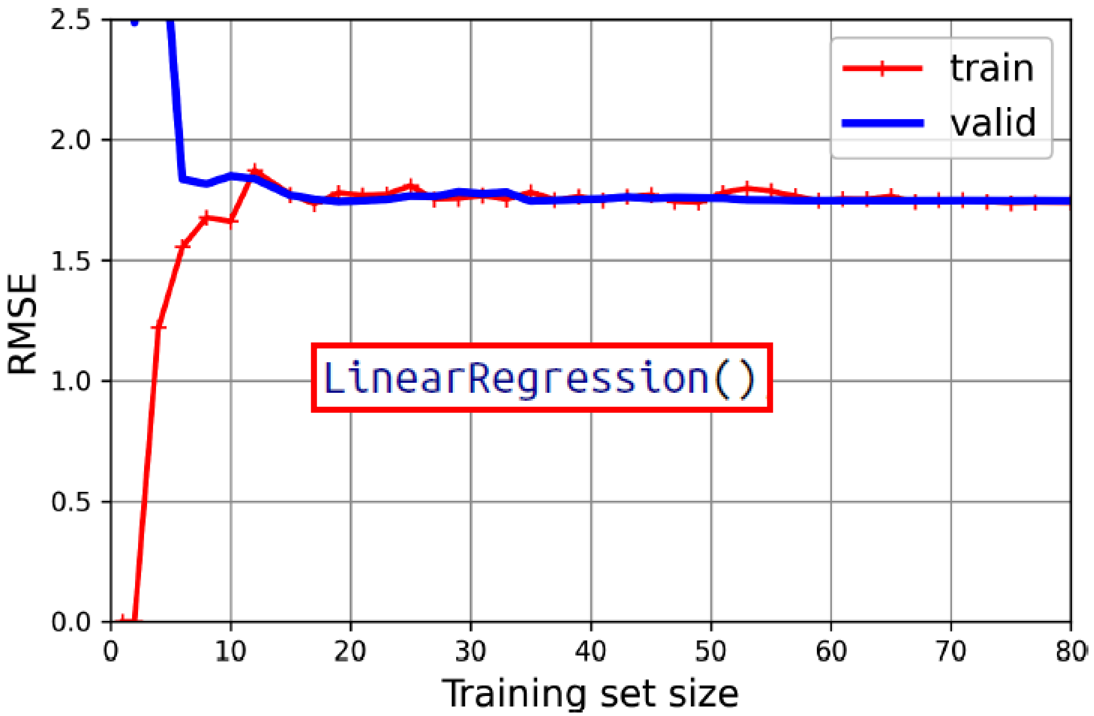
Learning Curves

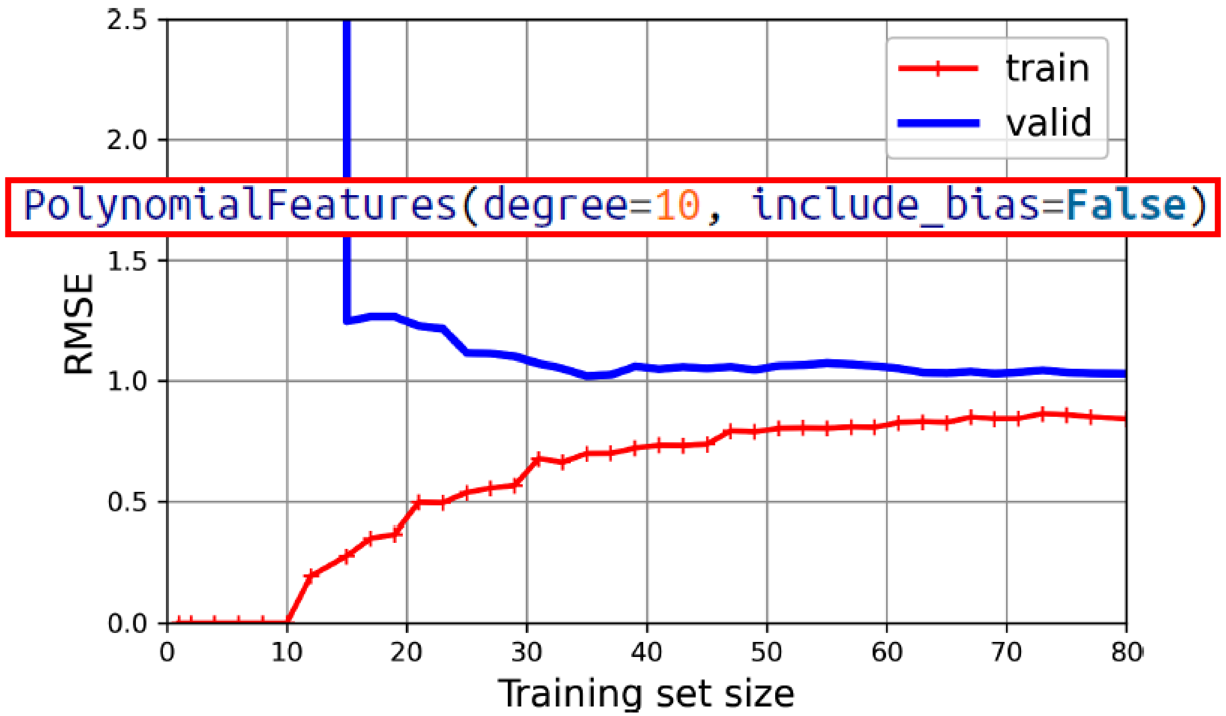
- If you perform **high-degree polynomial regression**, you will likely **fit the training data much better** than with plain linear regression.
- This **high-degree polynomial regression** model is severely **overfitting** the training data.
- The **linear model** is **underfitting** it.



Learning Curves

- The **accuracy** on the **validation set** generally **increases** as the **training set** size increases.
- **Overfitting decreases** with larger training set.





Outline

1. Linear Regression
2. Gradient Descent
3. Gradient Descent Variants
    1. Batch Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-batch Gradient Descent
4. Polynomial Regression
5. Learning Curves
6. Regularized Linear Models
7. Logistic Regression
8. Exercises

Regularized Linear Models

- **Ways to reduce overfitting**

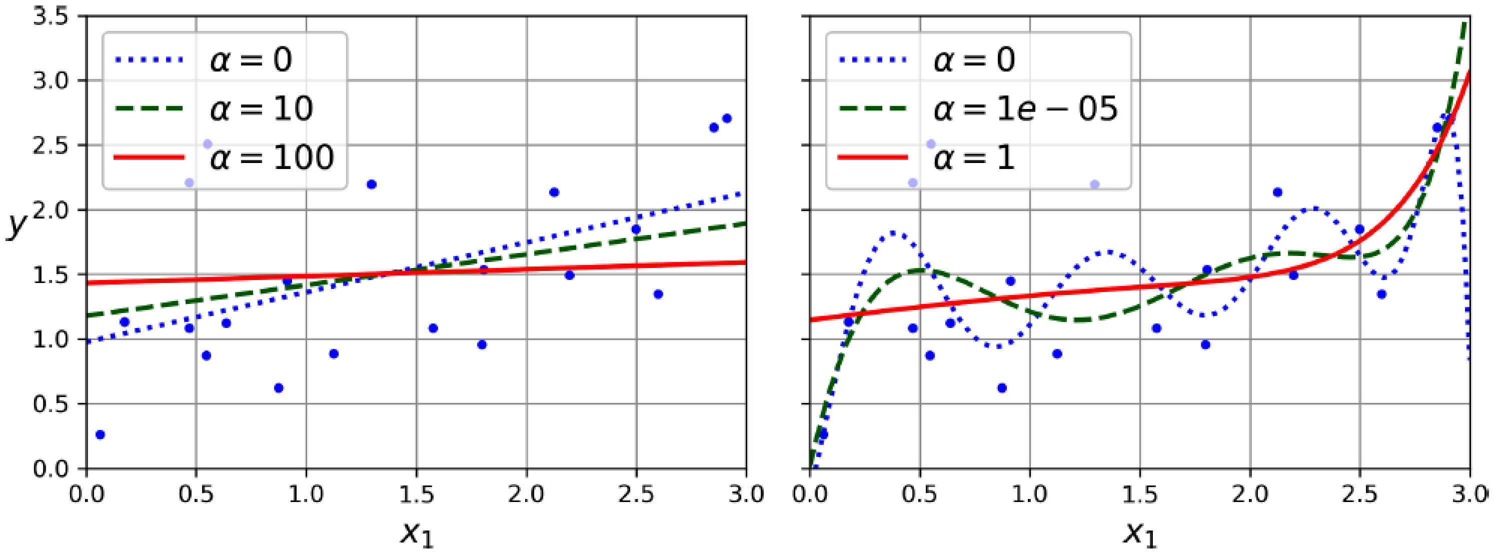
1. **Reduce** the number of **polynomial degrees**.
2. **Constrain the weights** of the model.
    - Ridge regression (L2)
    - Lasso Regression (L1)
3. Use **early stopping**.

Ridge Regression

- Start with scaled features.
- **Constrain the weights** of the model using the $\|\mathbf{w}\|_2 = l_2$ norm of the weight vector.

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \frac{\alpha}{m} \underline{\Sigma_{i=1}^{n} \theta_i^2}$$

Excludes Θ_0

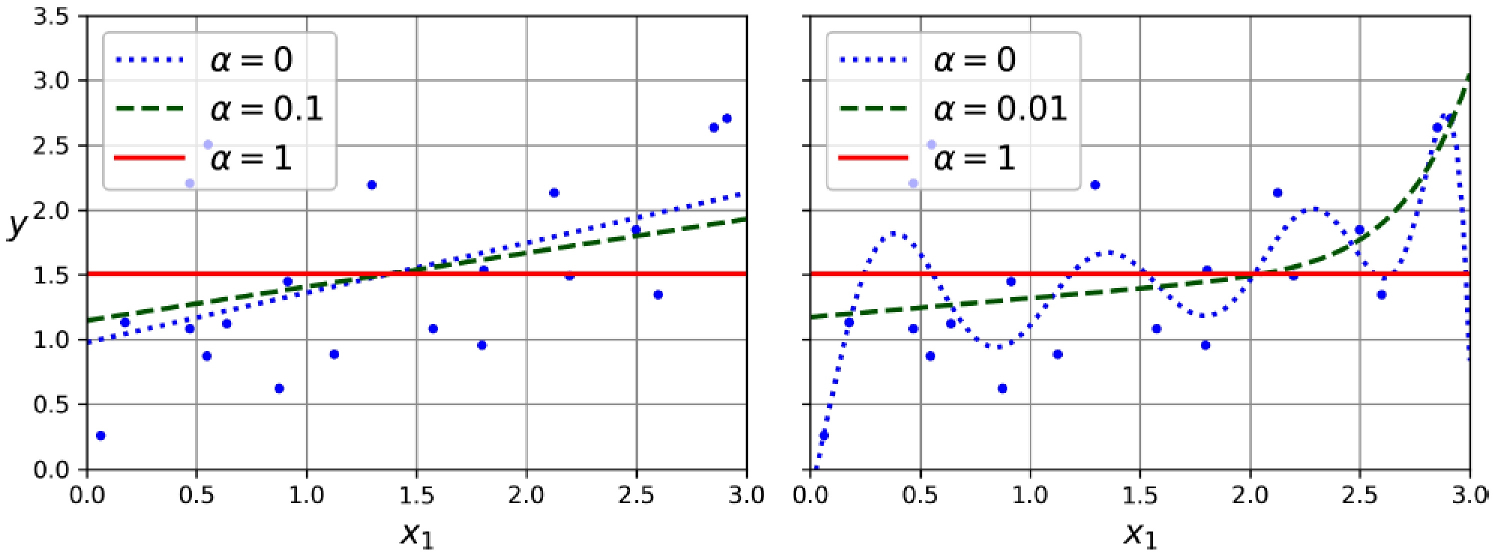


Lasso Regression

- LASSO: Least absolute shrinkage and selection operator regression
- **Constrains the weights** of the model using the $\|\mathbf{w}\|_1 = l_1$ norm of the weight vector.

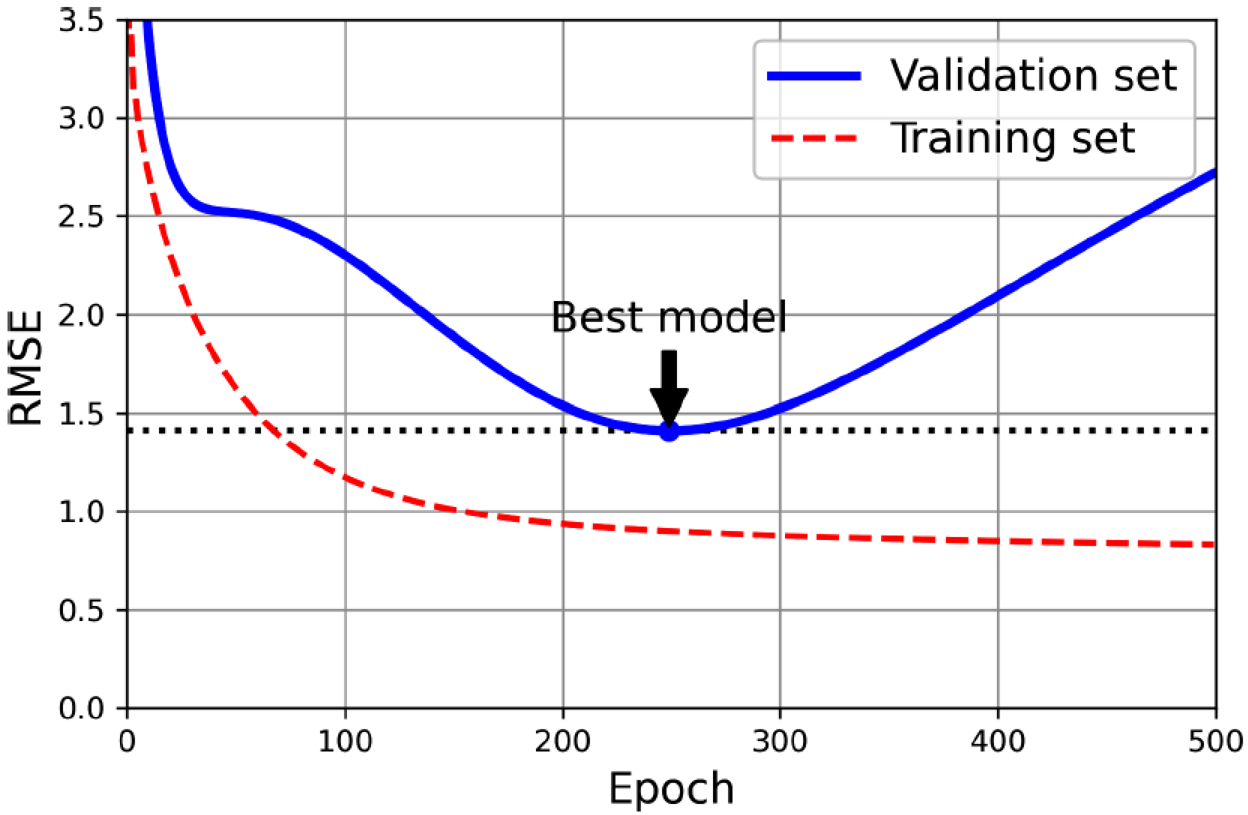
$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + 2\alpha \underline{\Sigma_{i=1}^{n} |\theta_i|}$$

Excludes Θ_0



Early Stopping

- **Stop** training when the **validation error reaches a minimum**.
- Need to **save the best model**.



Outline

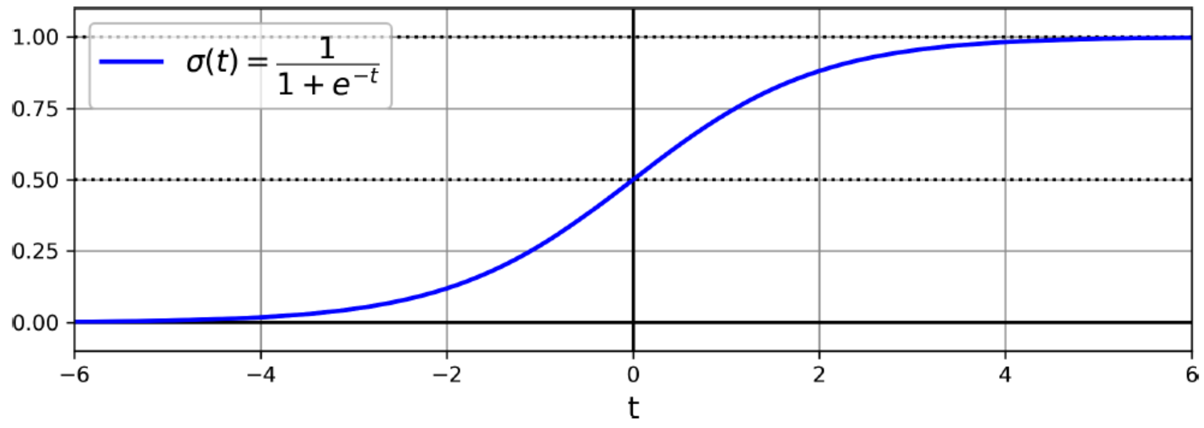
1. Linear Regression
2. Gradient Descent
3. Gradient Descent Variants
    1. Batch Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-batch Gradient Descent
4. Polynomial Regression
5. Learning Curves
6. Regularized Linear Models
7. Logistic Regression
8. Exercises

Logistic Regression

- Some regression algorithms can be used for **classification**.
- The **logistic regression** model **estimates the probability** that an instance belongs to a particular class from the weighted sum of the input features.

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\boldsymbol{\theta}^{\top}\mathbf{x})$$

Logistic or sigmoid function



Decision Boundaries

- The decision boundary is typically at **50%**.
- It can be **changed** for better **recall** or **precision**.

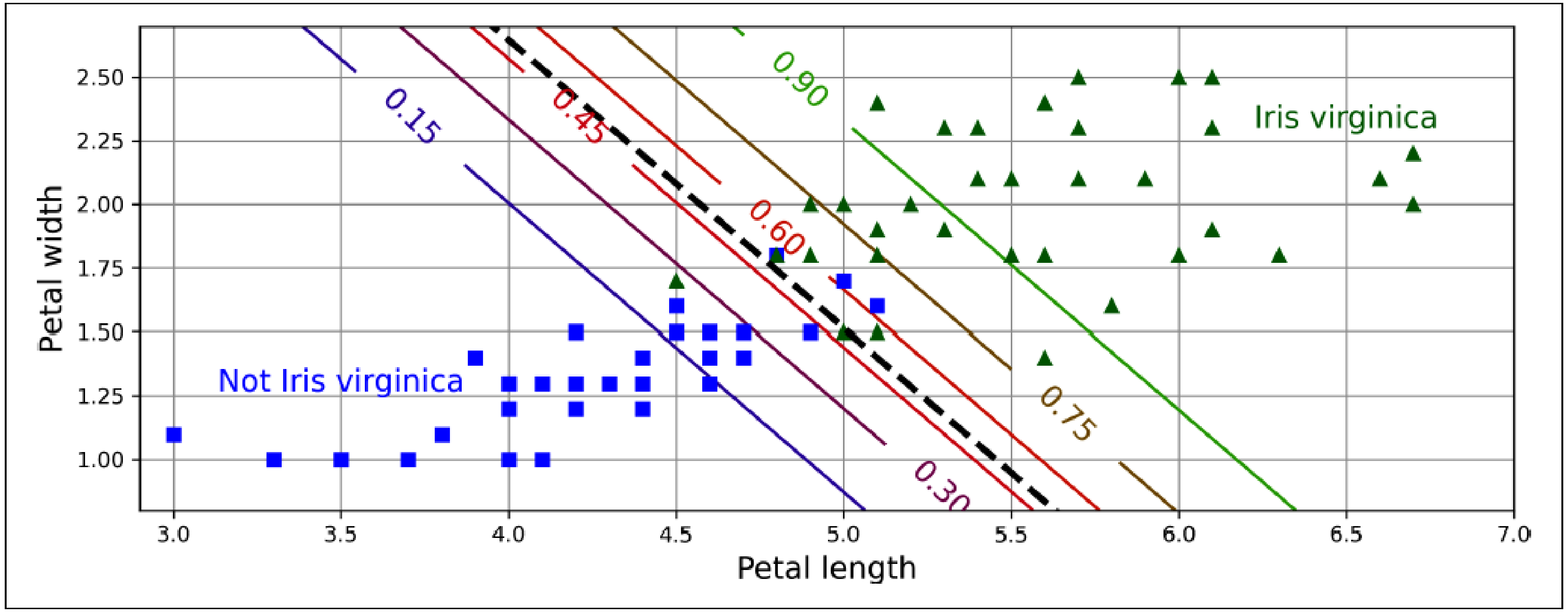


Figure 4-24. Linear decision boundary

Summary

1. Linear Regression
2. Gradient Descent
3. Gradient Descent Variants
    1. Batch Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-batch Gradient Descent
4. Polynomial Regression
5. Learning Curves
6. Regularized Linear Models
7. Logistic Regression
8. Exercises

Exercises

1. What Linear Regression training algorithm can you use if you have a training set with millions of features?
2. Suppose the features in your training set have very different scales. What algorithms might suffer from this, and how? What can you do about it?
3. Do all Gradient Descent algorithms lead to the same model provided you let them run long enough?