At this point, we introduce a critical function in machine learning called the softmax function. The softmax function is used to compute the probability that an instance belongs to one of the K classes when $K > 2$. We will see the softmax function show up again when we discuss (artificial) neural networks.

In order to build a classification model with k classes, the multinomial logistic model is formally defined as

$$\hat{y}(k) = \theta_0^k + \theta_1^k x_1 + \theta_2^k x_2 + \ldots + \theta_n^k x_n$$

The preceding model takes into consideration the parameters for the k different classes. The softmax function is formally written as

$$p(k) = \sigma\left(\hat{y}(k)\right)_i = \frac{e^{\hat{y}(k)_i}}{\sum_{j=1}^{K} e^{\hat{y}(k)_j \hat{} (k_j}}$$

where

- $i = \{1, \ldots, K\}$ classes.
- $\sigma\left(\hat{y}(k)\right)_i$ outputs the probability estimates that an example in the training dataset belongs to one of the K classes.

The cost function for learning the class labels in a multinomial logistic regression model is called the ***cross-entropy*** cost function. Gradient descent is used to find the optimal values of the parameter θ that will minimize the cost function to ***predict the class with the highest probability estimate accurately***.

Logistic Regression with Scikit-learn

In this example, we will implement a multi-class logistic regression model with Scikit-learn. The model will predict the three species of flowers from the Iris dataset. The dataset contains 150 observations and 4 features. For this example, we use the accuracy metric and confusion matrix to access the model's performance.

```
# import packages
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import multilabel_confusion_matrix

# load dataset
data = datasets.load_iris()
# separate features and target
X = data.data
y = data.target

# split in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True)

# create the model
logistic_reg = LogisticRegression(solver='lbfgs', multi_class='ovr')

# fit the model on the training set
logistic_reg.fit(X_train, y_train)

# make predictions on the test set
predictions = logistic_reg.predict(X_test)

# evaluate the model performance using accuracy metric
print("Accuracy: %.2f" % accuracy_score(y_test, predictions))

'Output':
Accuracy: 0.97

# print the confusion matrix
multilabel_confusion_matrix(y_test, predictions)

'Output':
array([[[26,  0],
        [ 0, 12]],

       [[25,  0],
        [ 1, 12]],

       [[24,  1],
        [ 0, 13]]])
```

Take note of the following in the preceding code block:

- The logistic regression model is initialized by calling the method LogisticRegression(solver='lbfgs', multi_class='ovr'). The attribute 'multi_class' is set to 'ovr' to create a one-vs.-rest classifier.
- The confusion matrix for a multi-class learning problem uses the `multilabel_confusion_matrix' to calculate classwise confusion matrices where the labels are binned in a one-vs.-rest manner. As an example, the first matrix is interpreted as the difference between the actual and predicted targets for class 1 against other classes.

Optimizing the Logistic Regression Model

This section surveys a few techniques to consider in optimizing/improving the performance of logistic regression models.

In the case of Bias (i.e., when the accuracy is poor with training data)

- Remove highly correlated features. Logistic regression is susceptible to degraded performance when highly correlated features are present in the dataset.
- Logistic regression will benefit from standardizing the predictors by applying feature scaling.
- Good feature engineering to remove redundant features or recombine features based on intuition into the learning problem can improve the classification model.
- Applying log transforms to normalize the dataset can boost logistic regression classification accuracy.

In the case of variance (i.e., when the accuracy is good with training data, but poor on test data)

Applying regularization (more on this in Chapter 21) is a good technique to prevent overfitting.

This chapter provides a brief overview of logistic regression for building classification models. The chapter includes practical steps for implementing a logistic regression classifier with Scikit-learn. In the next chapter, we will examine the concept of applying regularization to linear models to mitigate the problem of overfitting.