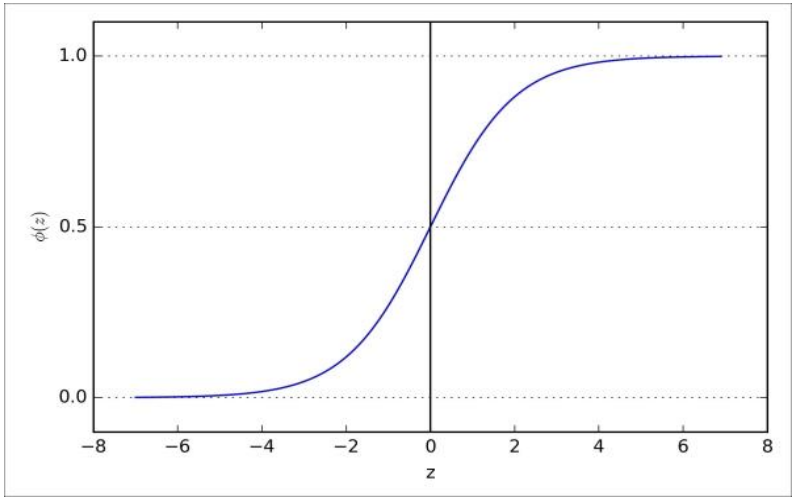
Modeling class probabilities via logistic regression

- Logistic regression intuition and conditional probabilities
- Learning the weights of the logistic cost function
- Converting an Adaline implementation into an algorithm for logistic regression
- Training a logistic regression model with scikit-learn
- Tackling overfitting via regularization

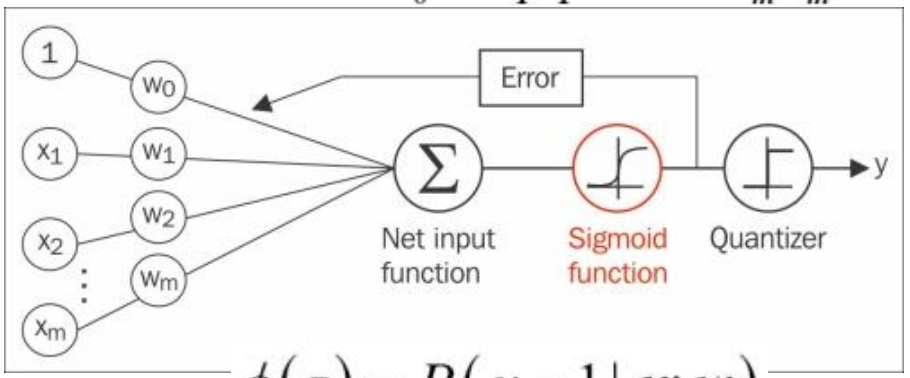
- Logistic regression과 conditional probabilities
  - odds ratio $\frac{p}{(1-p)}$
  - logit function $logit(p) = log\frac{p}{(1-p)}$

$$logit\left(p\left(y=1 \mid \boldsymbol{x}\right)\right) = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{i=0}^{n} w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

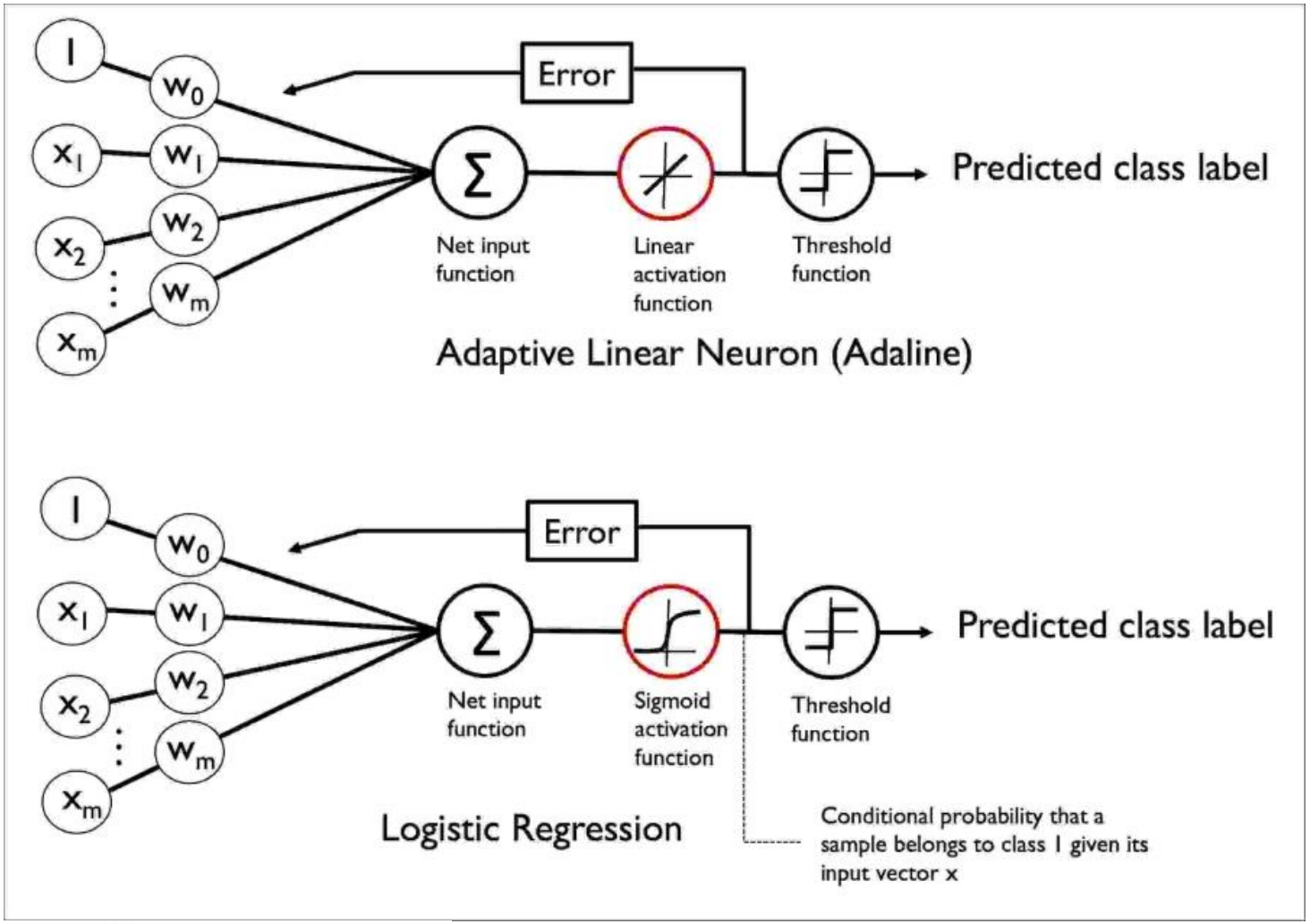
  - logistic function
    - sigmoid function $\phi(z) = \frac{1}{1+e^{-z}}$
      - (단, z is a linear combination of weights & sample features)



$$z = \boldsymbol{w}^T \boldsymbol{x} = w_0 + w_1 x_1 + \cdots + w_m x_m$$



$$\phi\left(z\right) = P\left(y=1 \mid \boldsymbol{x}; \boldsymbol{w}\right)$$



$$\hat{y} = \begin{cases} 1 & if\ \phi(z) \geq 0.5 \\ 0 & otherwise \end{cases}$$

$$\hat{y} = \begin{cases} 1 & if\ z \geq 0.0 \\ 0 & otherwise \end{cases}$$

- Logistic cost function의 weights 학습

$$J(\boldsymbol{w}) = \sum_i \frac{1}{2}\left(\phi\left(z^{(i)}\right) - y^{(i)}\right)^2$$

  - Define likelihood L to maximize
    - 단, assume independence of individual samples

$$L(\boldsymbol{w}) = P(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{w}) = \prod_{i=1}^{n} P\left(y^{(i)} \mid x^{(i)}; \boldsymbol{w}\right) = \prod_{i=1}^{n} \left(\phi\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \phi\left(z^{(i)}\right)\right)^{1-y^{(i)}}$$

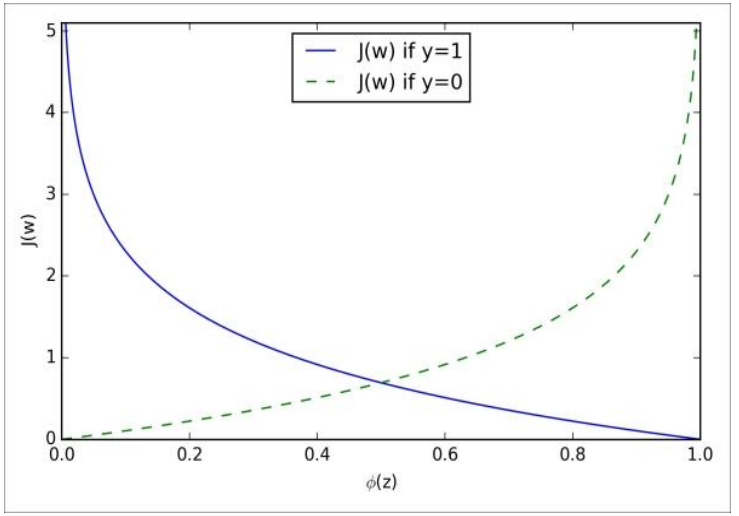
    - log-likelihood function

$$l(\boldsymbol{w}) = \log L(\boldsymbol{w}) = \sum_{i=1}^{n} \log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right)$$

Training and Cost Function

- Cost function
  - – log(t) grows large when t approach 0 → cost will be large if …
  - Vice versa

$$J\left(\phi(z), y; \boldsymbol{w}\right) = \begin{cases} -\log\left(\phi(z)\right) & \text{if } y = 1 \\ -\log\left(1-\phi(z)\right) & \text{if } y = 0 \end{cases}$$



- cost function over whole training

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

- Converting an Adaline implementation into an algorithm for logistic regression

$$J(\boldsymbol{w}) = -\sum_i y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right)$$

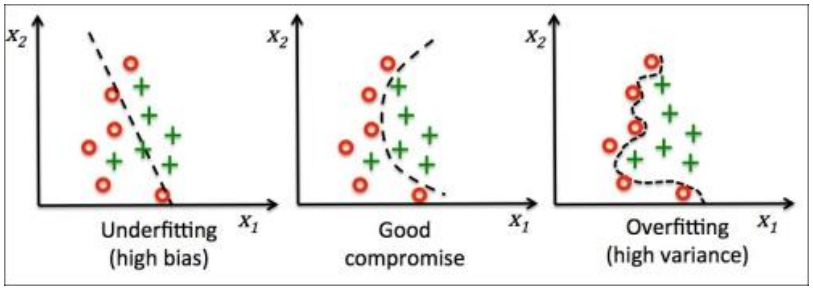
```
class LogisticRegressionGD(object):
    """
    Parameters
    ------------
    eta : float : Learning rate (between 0.0 and 1.0)
    n_iter : int : Passes over the training dataset.
    random_state : int : Random number generator seed for random weight initialization.

    Attributes
    -----------
    w_ : 1d-array : Weights after fitting.
    cost_ : list : Sum-of-squares cost function value in each epoch.
    """
    def __init__(self, eta=0.05, n_iter=100, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        """ Fit training data.
        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
          Training vectors, where n_samples is the number of  samples and
          n_features is the number of features.
        y : array-like, shape = [n_samples] : Target values.

        Returns
        -------
        self : object
        """
```

```
rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            net_input = self.net_input(X)
            output = self.activation(net_input)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()

            # note that we compute the logistic `cost` now
            # instead of the sum of squared errors cost
            cost = (-y.dot(np.log(output)) -  ((1 - y).dot(np.log(1 - output))))
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, z):
        """Compute logistic sigmoid activation"""
        return 1. / (1. + np.exp(-np.clip(z, -250, 250)))

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, 0)
        # equivalent to:
        # return np.where(self.activation(self.net_input(X)) >= 0.5, 1, 0)
```

- Training a logistic regression model with scikit-learn

```
>>> from sklearn.linear_model import LogisticRegression
>>> lr = LogisticRegression(C=1000.0, random_state=0)
>>> lr.fit(X_train_std, y_train)
   >>> plot_decision_regions(X_combined_std,
...                       y_combined, classifier=lr,
...                       test_idx=range(105,150))
>>> plt.xlabel('petal length [standardized]')
>>> plt.ylabel('petal width [standardized]')
>>> plt.legend(loc='upper left')
>>> plt.show()
```

- Tackling overfitting via regularization
  - Overfitting vs. Underfitting



  - [note]
    - Variance measures the consistency (or variability) of the model prediction for a particular sample instance if we would retrain the model multiple times
    - bias measures how far off the predictions are from the correct values in general;
      - measure of the systematic error that is not due to randomness.
      - 예: L2 regularization $C = \frac{1}{\lambda}$ $\frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$

$$J(\boldsymbol{w}) = C\left[\sum_{i=1}^{n}\left(-\log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\right)\left(-\log\left(1 - \phi(z)\right)\right)\right] + \frac{1}{2}\|\boldsymbol{w}\|^2$$