PRACTICAL WEEK 2

8BB020: Introduction to Machine Learning

Solutions

Linear regression

Function

Input: m parameters $x_i, \ldots, x_m$. Output: y.

$y = \hat{y} + \epsilon$;

$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$;

For n observations, each with m variables:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\theta}$$

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1m} \\ 1 & x_{21} & x_{22} & \ldots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{nm} \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

Two methods

Using least squares

$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{y}$, provided $(\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}$ is non-singular.

This is actually the product of a matrix and a vector:

$$\boldsymbol{\theta} = (\underbrace{\mathbf{X}^\top\mathbf{X}}_{\mathbf{A}})^{-1}\underbrace{\mathbf{X}^\top\mathbf{y}}_{\mathrm{b}}$$

Using gradient descent

$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{current})} - \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{current})} \underbrace{-}_{\text{opposite direction}} \overbrace{\eta}^{\text{small step}} \underbrace{\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})}_{\text{direction of the gradient}}$$

Using least squares

$$J(\boldsymbol{\theta}) = RSS(\boldsymbol{\theta}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - (\theta_0 + \sum_{j=1}^{m} x_{ij}\theta_{ij}))^2$$

OR

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta)$$

Best solution of this function is given when **gradient = 0!!!**

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\theta) = 0$$

$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^\mathbf{T}\mathbf{y}$, provided $(\mathbf{X}^\mathbf{T}\mathbf{X})^{-1}$ is non-singular.

This is actually the product of a matrix and a vector:

$$\boldsymbol{\theta} = (\underbrace{\mathbf{X}^\top\mathbf{X}}_{\mathbf{A}})^{-1}\underbrace{\mathbf{X}^\top\mathbf{y}}_{\mathrm{b}}$$

Linear regression

Function

Input: m parameters $x_i, \ldots, x_m$. Output: y.

$y = \hat{y} + \epsilon$;

$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$;

For n observations, each with m variables:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\theta}$$

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1m} \\ 1 & x_{21} & x_{22} & \ldots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \ldots & x_{nm} \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

Two methods of training

Using least squares

$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^\mathbf{T}\mathbf{y}$, provided $(\mathbf{X}^\mathbf{T}\mathbf{X})^{-1}$ is non-singular.

This is actually the product of a matrix and a vector:

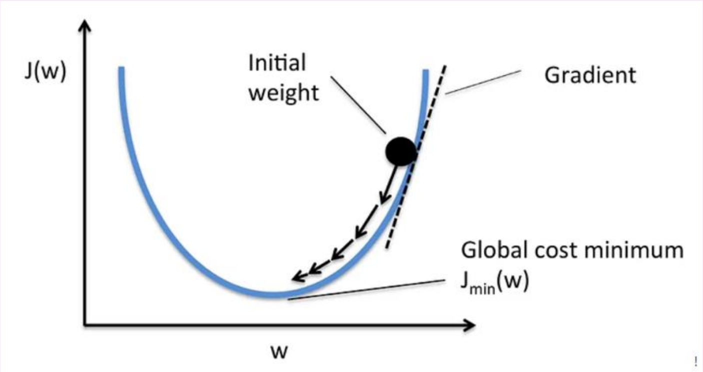
$$\boldsymbol{\theta} = (\underbrace{\mathbf{X}^\top\mathbf{X}}_{\mathbf{A}})^{-1}\underbrace{\mathbf{X}^\top\mathbf{y}}_{\mathrm{b}}$$

Using gradient descent

$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{current})} - \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{current})} \underbrace{-}_{\text{opposite direction}} \overbrace{\eta}^{\text{small step}} \underbrace{\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})}_{\text{direction of the gradient}}$$

Using gradient descent (Use when $(X^T X)^{-1}$ is singular.)



STEP 1: start off with random solution for θ and calculate J(θ).

STEP 2: update θ in order to reduce cost function. This is done by changing θ in direction where gradient is less.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = 2(\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} - \mathbf{X}^\top \mathbf{y}) \\ = 2\mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = 2\mathbf{X}^\top \overbrace{(\underbrace{\mathbf{X}\boldsymbol{\theta}}_{\text{prediction}} - \mathbf{y})}^{\text{error}}$$

STEP 3: We stop when certain criterium is met (e.g. stop when θ no longer significantly changes, or after a certain # of iterations)

$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{current})} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{current})} \underbrace{-}_{\text{opposite direction}} \overbrace{\eta}^{\text{small step}} \underbrace{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}_{\text{direction of the gradient}}$$

η = learning rate

ANSWERS

```python
def _solve_normal(self, X, y):

    X_b = augment_matrix(X)
    n_samples, n_features = X_b.shape

    # replace the code below (which returns random values for theta) with your solution to return the correct solution for theta
    # START EXERCISE 1.1 #

    X_trans = X_b.transpose()

    A = np.linalg.inv(X_trans.dot(X_b))
    b = X_trans.dot(y)

    self.theta = A.dot(b)

    #OR
    self.theta = np.dot(np.linalg.inv(np.dot(np.transpose(X_b),X_b)),np.dot(np.transpose(X_b),y))


    # END EXERCISE 1.1 #

    return self.theta
```

$$\boxed{\boldsymbol{\theta} = \underbrace{(\mathbf{X}^\top\mathbf{X})}_{\mathbf{A}}{}^{-1}\underbrace{\mathbf{X}^\top\mathbf{y}}_{\mathbf{b}}}$$

```python
def _solve_gradient_descent(self, X, y, n_epochs, learning_rate): #linear regression with gradient descent
    X_b = augment_matrix(X)
    n_samples, n_features = X_b.shape

    # replace the code below (which returns random values for theta) with your solution to return the correct solution for theta
    # START EXERCISE 1.2 #
    self.theta = np.random.randn(n_features)

    #om deze loop te gebruiken moet je kijken naar de formules
    for epoch in range(n_epochs):
        prediction = X_b.dot(self.theta)
        error = prediction - y

        gradient = 2 * X_b.T.dot(error)
        self.theta -= learning_rate*gradient
    # END EXERCISE 1.2 #

    return self.theta
```

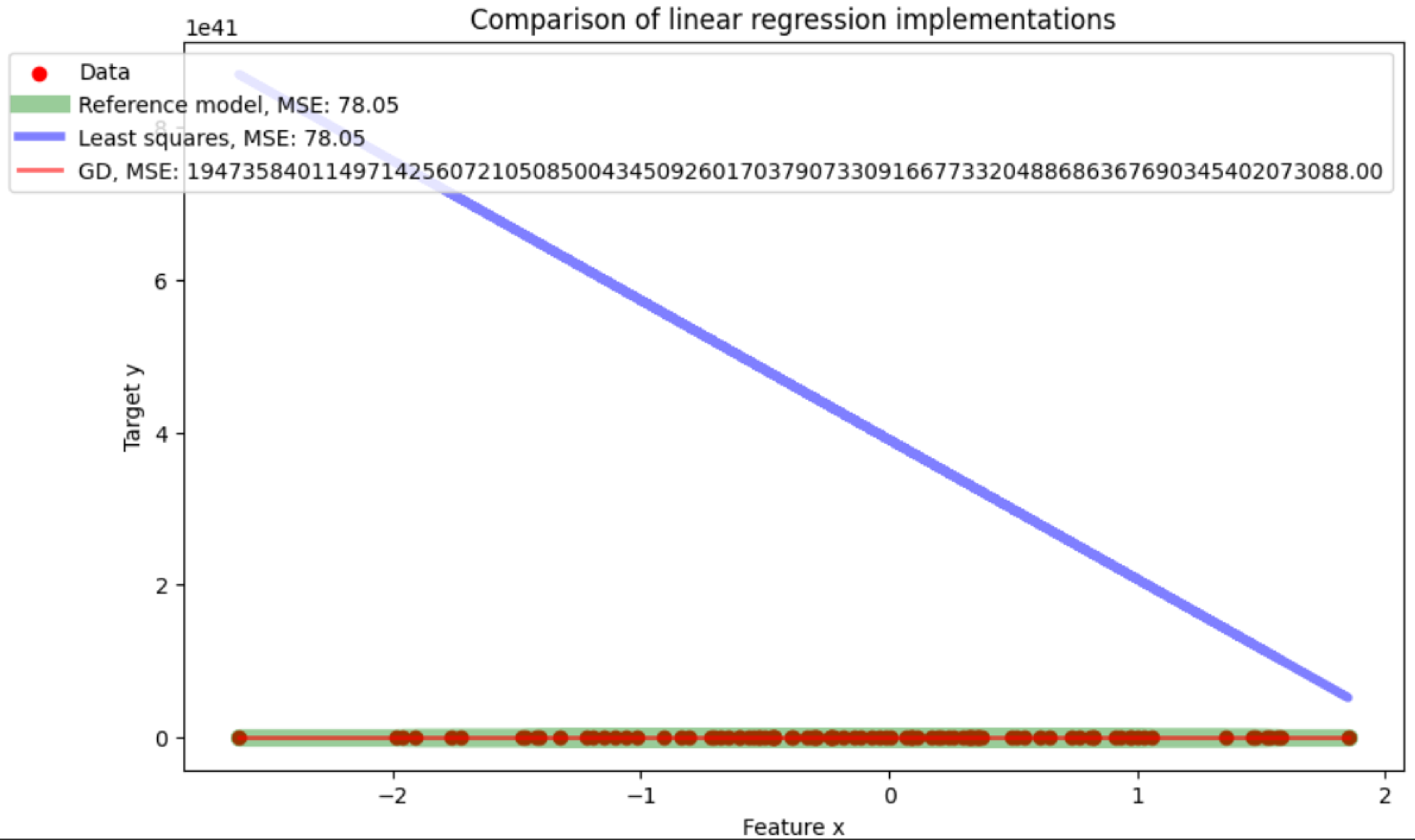
$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{current})} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

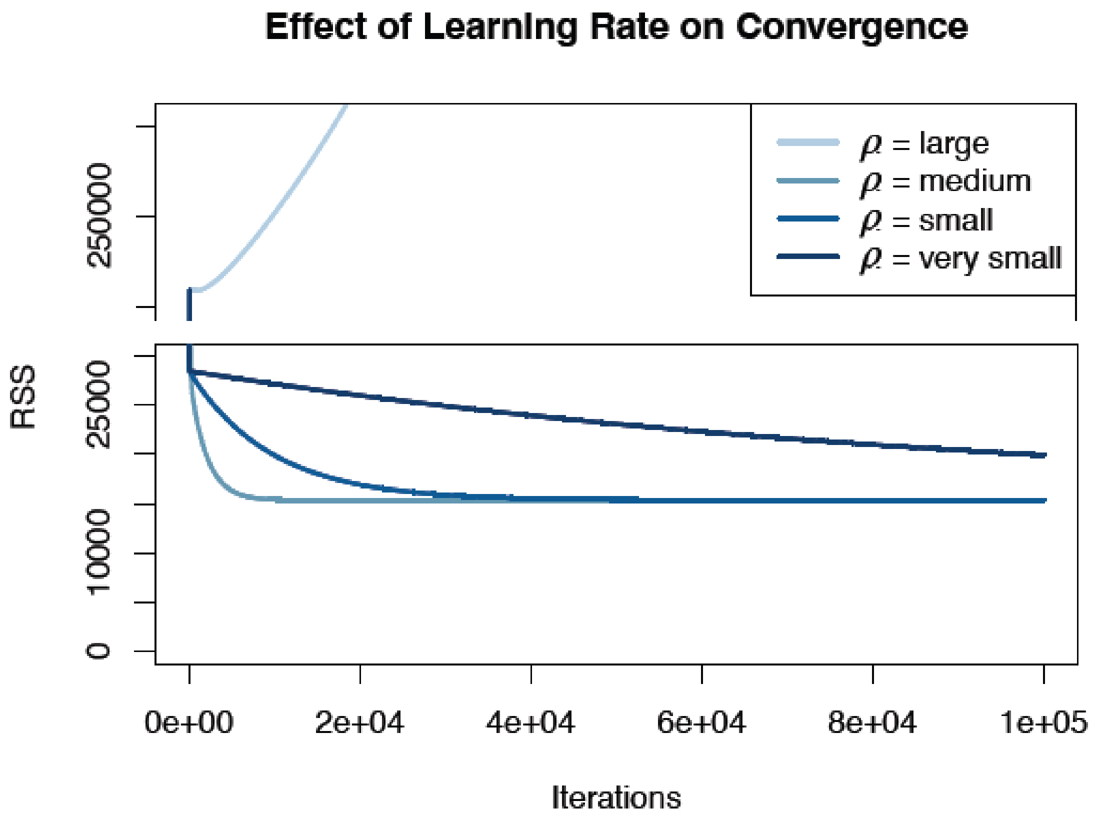
$$\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{current})} \underbrace{-}_{\text{opposite direction}} \overbrace{\eta}^{\text{small step}} \underbrace{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}_{\text{direction of the gradient}}$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = 2\mathbf{X}^\top \big(\overbrace{\underbrace{\mathbf{X}\boldsymbol{\theta}}_{\text{prediction}} - \mathbf{y}}^{\text{error}} \big)$$

```python
import tests
import numpy as np
from sklearn.linear_model import LinearRegression

tests.linear_regression(MyLinearRegression, LinearRegression, epochs=1000, learning_rate=0.01)
```

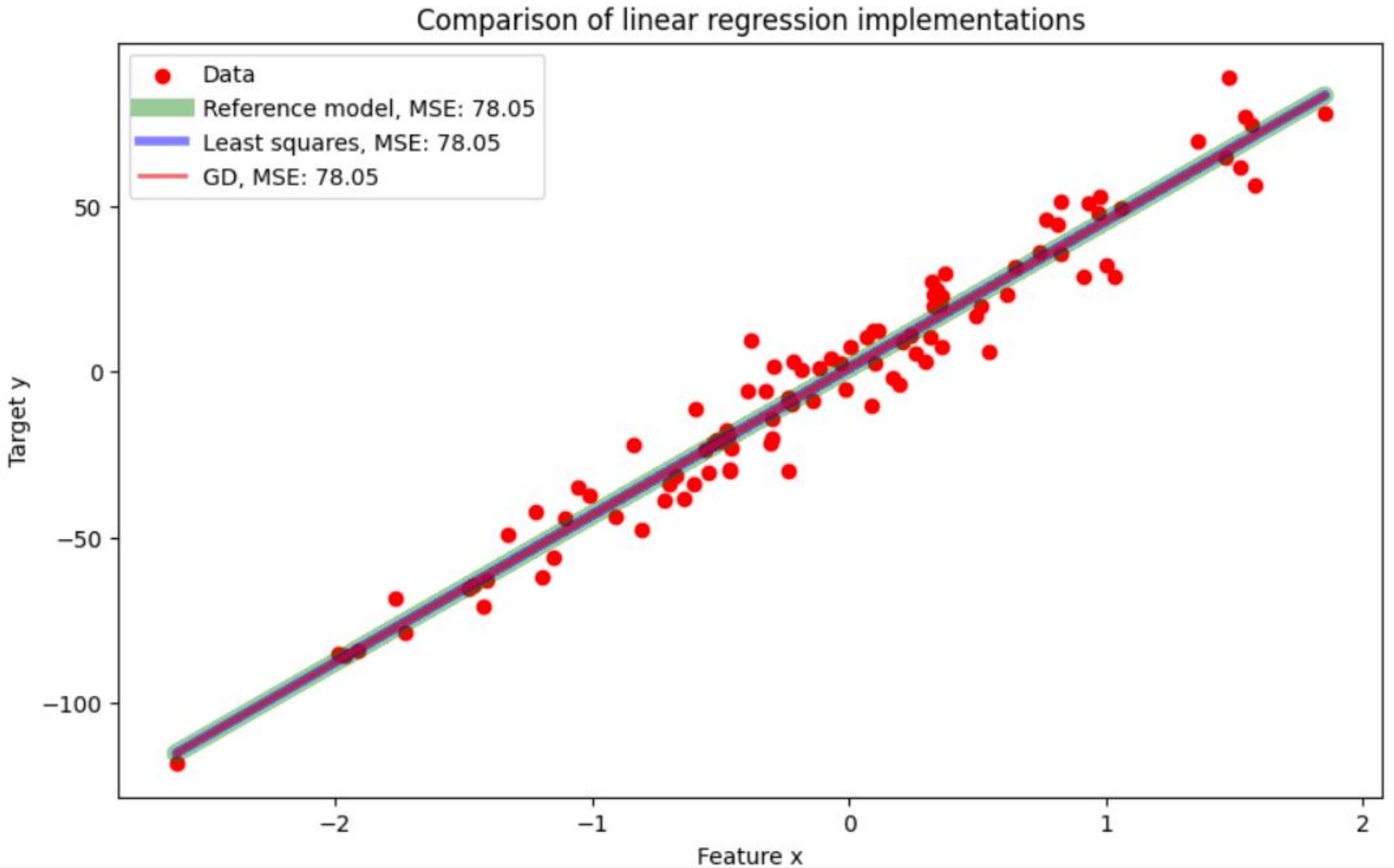




Large	The algorithm overshoots the minimum, leading to oscillations or even divergence (increasing error).
Medium/ small	The algorithm quickly converges to the minimum without overshooting.
Very small	The algorithm takes many iterations to reach the minimum because each step is tiny

```python
import tests
import numpy as np
from sklearn.linear_model import LinearRegression

tests.linear_regression(MyLinearRegression, LinearRegression, epochs=1000, learning_rate=0.0001)
```



Logistic regression

Function

$$p(y=1|X) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\ldots+\theta_n x_n)}}$$

This means the probability that y=1 given the value X.

$$\mathbf{p} = \frac{1}{1+e^{-\mathbf{X}\boldsymbol{\theta}}}$$

$$\mathbf{p} = \begin{bmatrix} p(y_1=1) \\ p(y_2=1) \\ \vdots \\ p(y_m=1) \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1n} \\ 1 & x_{21} & x_{22} & \ldots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \ldots & x_{mn} \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Gradient of logistic regression loss function

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

$$\boldsymbol{\theta}^{(\text{next})} = \boldsymbol{\theta}^{(\text{current})} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}^{(\text{next})} = \boldsymbol{\theta}^{(\text{current})} - \eta \frac{1}{m}\mathbf{X}^\top(\mathbf{p}-\mathbf{y})$$

Logistic regression loss function

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

1. What is the name of the summation factor?

Given a training data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ we can estimate the model coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ that maximise the *likelihood function:*

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$$

i.e. it gives a probability close to 0 for observations in class 0 and close to 1 for observations in class 1.

Logistic regression

Function

$$p(y=1|X) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\ldots+\theta_n x_n)}}$$

This means the probability that y=1 given the value X.

$$\mathbf{p} = \frac{1}{1+e^{-\mathbf{X}\boldsymbol{\theta}}}$$

$$\mathbf{p} = \begin{bmatrix} p(y_1=1) \\ p(y_2=1) \\ \vdots \\ p(y_m=1) \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1n} \\ 1 & x_{21} & x_{22} & \ldots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \ldots & x_{mn} \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Gradient of logistic regression loss function

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

$$\boldsymbol{\theta}^{(\text{next})} = \boldsymbol{\theta}^{(\text{current})} - \eta\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}^{(\text{next})} = \boldsymbol{\theta}^{(\text{current})} - \eta\frac{1}{m}\mathbf{X}^{\top}(\mathbf{p}-\mathbf{y})$$

ANSWERS

```python
class MyLogisticRegression:
    def __init__(self, epochs=1000, learning_rate=0.01):
        self.theta = None
        self.epochs = epochs
        self.learning_rate = learning_rate

    def fit(self, X, y):
        X_b = augment_matrix(X)
        n_samples, n_features = X_b.shape
        self.theta = np.random.randn(n_features)

        # END EXERCISE 3.1 #
        #'''
        for _ in range(self.epochs):
            predictions = sigmoid(X_b.dot(self.theta))
            error = predictions - y
            gradients = 1 / n_samples * X_b.T.dot(error)
            self.theta -= self.learning_rate * gradients
        #'''
        # END EXERCISE 3.1 #
```

$$\boxed{\begin{aligned}\boldsymbol{\theta}^{(\text{next})} &= \boldsymbol{\theta}^{(\text{current})} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\ \boldsymbol{\theta}^{(\text{next})} &= \boldsymbol{\theta}^{(\text{current})} - \eta \frac{1}{m} \mathbf{X}^{\top} (\mathbf{p} - \mathbf{y})\end{aligned}}$$