

Machine Learning - Homework 2

- Logistic Regression -

ChangYoon Lee

Dankook University, Korea Republic
32183641@gmail.com

Abstract. Logistic regression is one of the simplest classification models in the machine learning algorithm. The following model fits the best model by changing the coefficients and the intercept variables of the model which best classifies the labeled data. The sigmoid function combined with the concepts of decision boundary is used in the implementation and the error, which is also called the cost, of the classification model is evaluated by the cross-entropy function. By using these concepts, we will present the implemented logistic regression model and the process of training it by using the given train data set and performing classification on the test data set, which represents task 1 and task 2. Also, there will be the extra implementation of the logistic regression model that uses the polynomial graph.

Keywords: Logistic Regression, Sigmoid Function, Hyperbolic Tangent, Least, Cross-Entropy.

1 Introduction

A logistic regression model is used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as binary categories, which is based on a given dataset of independent variables. Since the outcome is a probability, the dependent variables are bound between zero and one (What is logistic regression?).

In the following operation, independent variables are chosen that generate the best decision boundaries which classify the dependent variables, based on the coefficients of the equations. It minimizes the discrepancies between the predicted and real output labels by fitting a straight line or a surface. To perform the logistic regression, we use some main concepts: logistic regression model, sigmoid, logistic regression, hyperbolic tangent functions, decision boundary, and underfitting and overfitting. Also, the error of the prediction, which is called cost, is calculated through the cross-entropy function.

In this paper, we will discuss the basic concepts that are used to implement the logistic regression model: sigmoid, logistic regression, hyperbolic function, decision boundary, and underfitting and overfitting. Then, we will discuss how we evaluate the performance of the prediction model by evaluating the error: cross entropy function. After discussing all the concepts, we will present the implemented python code for task 1 and task 2, which represents training the model with the training data set and classifying the data in the test data set using the trained model. In the result, we will show the prediction graph according to the given data set and the coefficients for the predicted model. Also, there will be comparisons between the built-in logistic regression models and the implemented models.

2 Concepts

2.1 Logistic Regression

Logistic regression is a classification model rather than a regression model. Logistic regression is a simple and efficient method for binary and linear classification problems. It is a classification model, that is very easy to realize and achieves very good performance with linearly separable labels (*Logistic regression*).

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Figure 1 - Logistic Function (Lee, *Logistic Regression*)

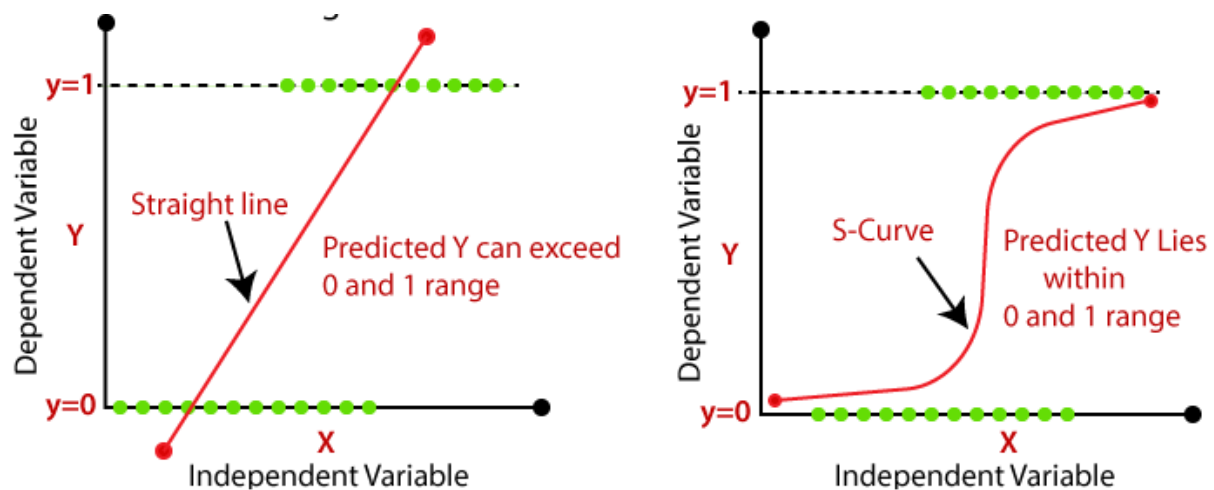


Figure 2 - Difference between Linear Regression and Logistic Regression Model (Lee, *Logistic Regression*)

The best way to think about logistic regression is that it is a linear regression for classification problems. Logistic regression essentially uses a logistic function, which is presented in Figure 1. The primary difference between the linear regression and logistic regression model is that the range of the logistic regression model is bounded between zero and 1. In addition, as opposed to the linear regression model, the logistic regression model does not require a linear relationship between input and output. Figure 2 shows the following difference between linear regression and the logistic regression model. By using the logistic regression model, we can perform a classification of the data with the labels.

2.2 Sigmoid Function

The main problem with the application of the linear regression model in the classification is that the result of the prediction can have a boundary of less than zero or bigger than one. This result makes the following model unreliable, as it gives the wrong classification result. To overcome the following situation, we used the sigmoid function for the classification.

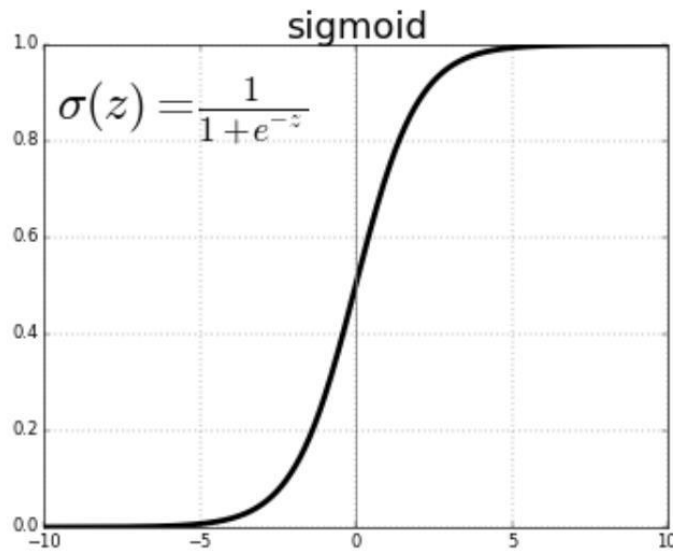


Figure 3 - Sigmoid Function and the Derivative Form of the Sigmoid Function (Lee, *Logistic Regression*)

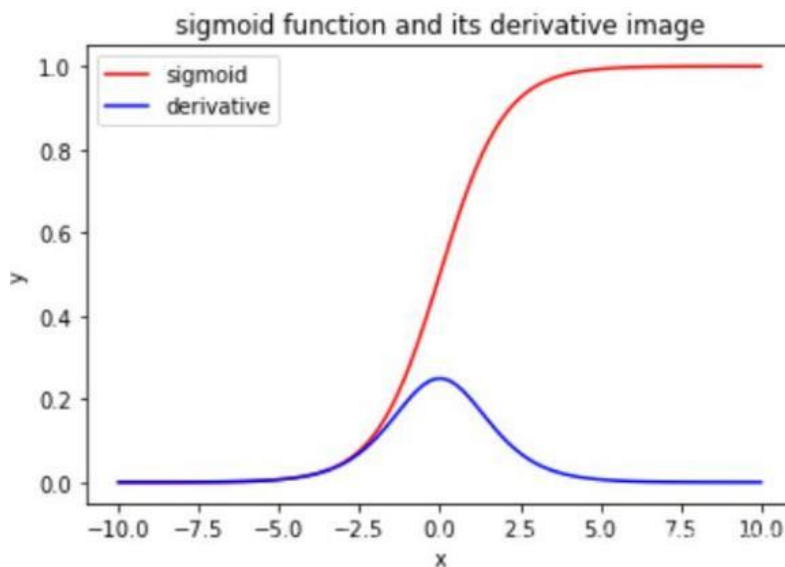


Figure 4 - First Derivative of the Sigmoid Function (Lee, *Logistic Regression*)

$$h'(x) = h(x)(1-h(x))$$

Figure 5 - Derivative Function of the Sigmoid Function (Lee, *Logistic Regression*)

A sigmoid function is a special form of the logistic function, which can be used to transform a continuous space value into a binary value. This function is monotonic, which means that it is constrained by a pair of horizontal asymptotes as the x-axis approaches infinity (Lee, *Logistic Regression*). The sigmoid function is presented in Figure 3. Also, the first derivative of the sigmoid function shows the bell-shaped graph, which is presented in Figure 4 and has the formula presented in Figure 5.

$$h(x) = g(\theta^T x)$$

Figure 6 - Sigmoid Function with the Linear Decision Boundary (Lee, *Logistic Regression*)

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Figure 7 - Applied Sigmoid Function (Lee, *Logistic Regression*)

To apply the sigmoid function to the classification, we need to transform the original sigmoid function to the given data set to fit the data. Before applying the following function to the classification, we use the sigmoid function with the linear decision boundary, which is presented in Figure 6. The final applied function will be the formula presented in Figure 7. The application of the linear decision boundary will be discussed in the later sections. By using the sigmoid function, we can perform the classification for the given data set.

2.3 Decision Boundary

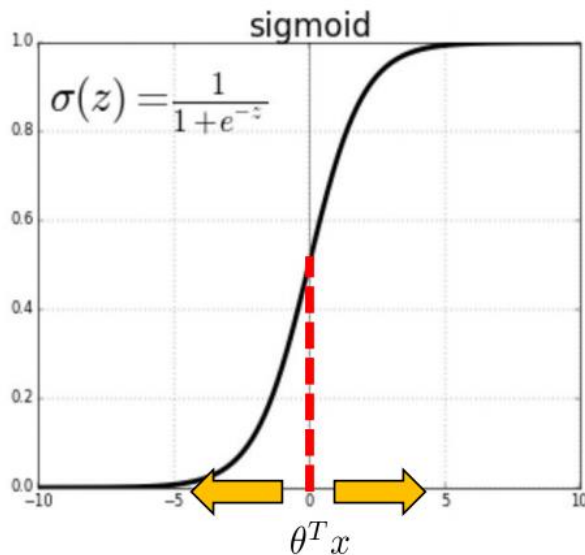


Figure 8 - Visualization of the Sigmoid Function with Linear Decision Boundary (Lee, *Logistic Regression*)

In the previous section, we discussed about the application of the sigmoid function for the classification problem. Before the application, we used the linear decision boundary. When we apply the linear decision boundary, the sigmoid function will be presented as the graph which is presented on the Figure 7.

$$\theta^T x \geq 0 \Rightarrow y = 1$$

$$\theta^T x < 0 \Rightarrow y = 0$$

Figure 9 - Decision Rule of the Sigmoid Function with Linear Decision Boundary (Lee, *Logistic Regression*)

According to the sigmoid function with the linear decision boundary, we can set the decision rule, which is presented on the Figure 8. If the linear decision boundary is over the following variable, the result becomes one, and if not, it is zero. Also, the following variable determines the shape of the decision boundary. By applying this concept, we can apply the sigmoid function for the classification problem.

2.4 Cross-Entropy Function

Now, we will focus on the process of how to choose the proper value for the linear decision boundary variable to minimize the errors, which is also called a cost. In the classification problem, the mean squared error (MSE) formula, which is used to evaluate the errors in the linear regression mode, is not appropriate, due to main two reasons: the MSE cost function is not convex for the classification data, and the following function does not give the strong penalty on the misclassified data.

Let's focus on the second reason. If the actual label of the data for the given input is one, and the prediction is zero, which presents the case of the perfect mismatch, the difference between them is one. Therefore, in the case of the MSE function, the formula will measure the loss value as one. This loss is a small amount of the wrong prediction compared to the perfect match case.

$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases}$$

Figure 10 – Cross-Entropy Function (Lee, *Logistic Regression*)

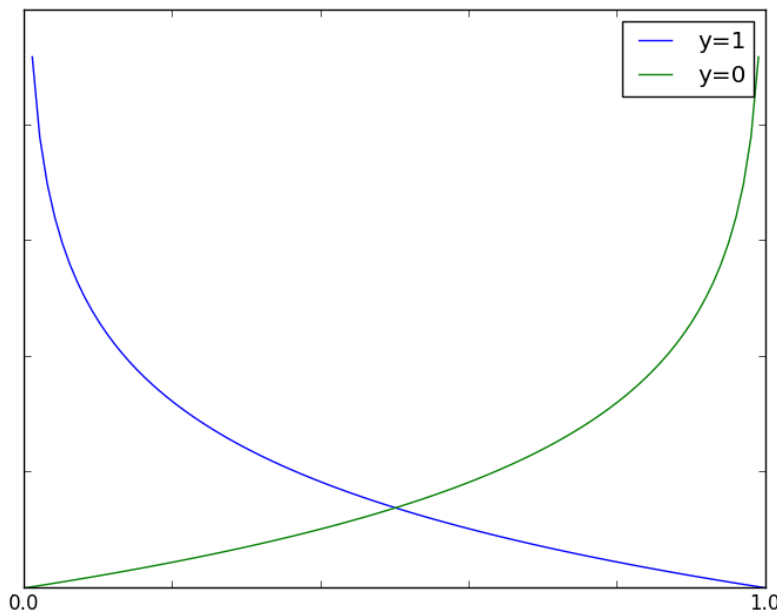


Figure 11 - Visualization of the Cross-Entropy Function (Lee, *Logistic Regression*)

As a result, there should be a significant penalty for the perfect mismatch cases to avoid the following cases. To give such a significant penalty, we use cross-entropy, which is formed by a logarithmic function. Figures 10 and 11 present the cross-entropy function and its visualization. When the real label of the data is one, the following function penalizes a very large cost if the prediction towards zero, and vice versa.

By using the cross-entropy function, we can measure the reliable errors for performing the classification of the logistic regression model and use the error in training the classification models.

3 Implementation

3.1 Representation of Given Data Set

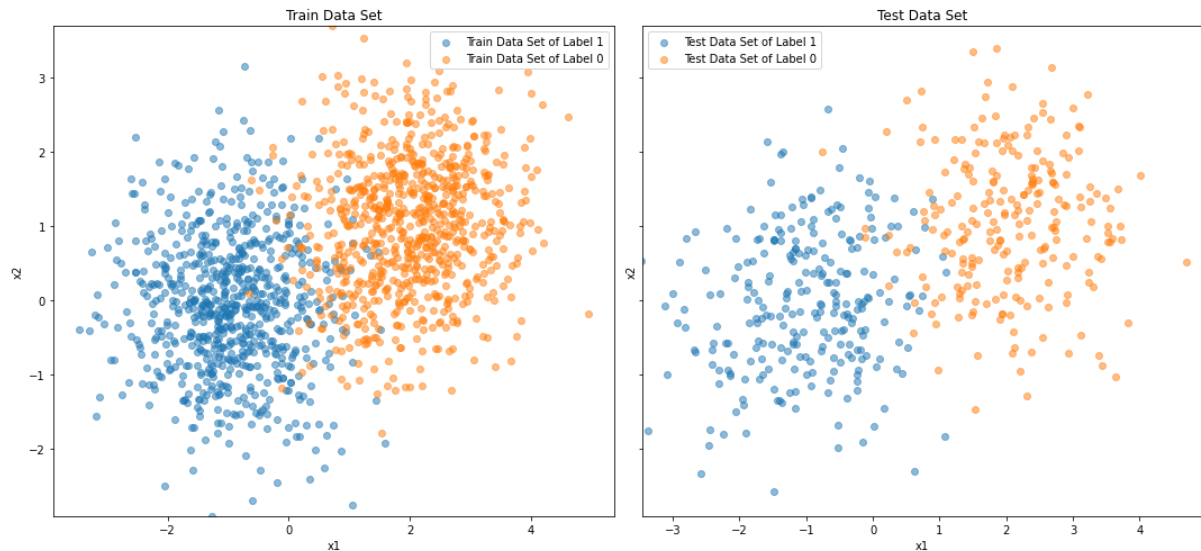


Figure 12 - Visualization of the Given Train and Test Data Set

The main purpose of this project is to code a binary classifier using the logistic regression model. We were given two files, the train, and the test file. These files are characterized by two features, x_1 , and x_2 , and each data point belongs to the label zero and one. Figure 13 presents a visualization of the given dataset. The orange dots represent label zero, and the blue dots represent label one.

3.2 Task 1: Train a Classifier with the Train Data Set

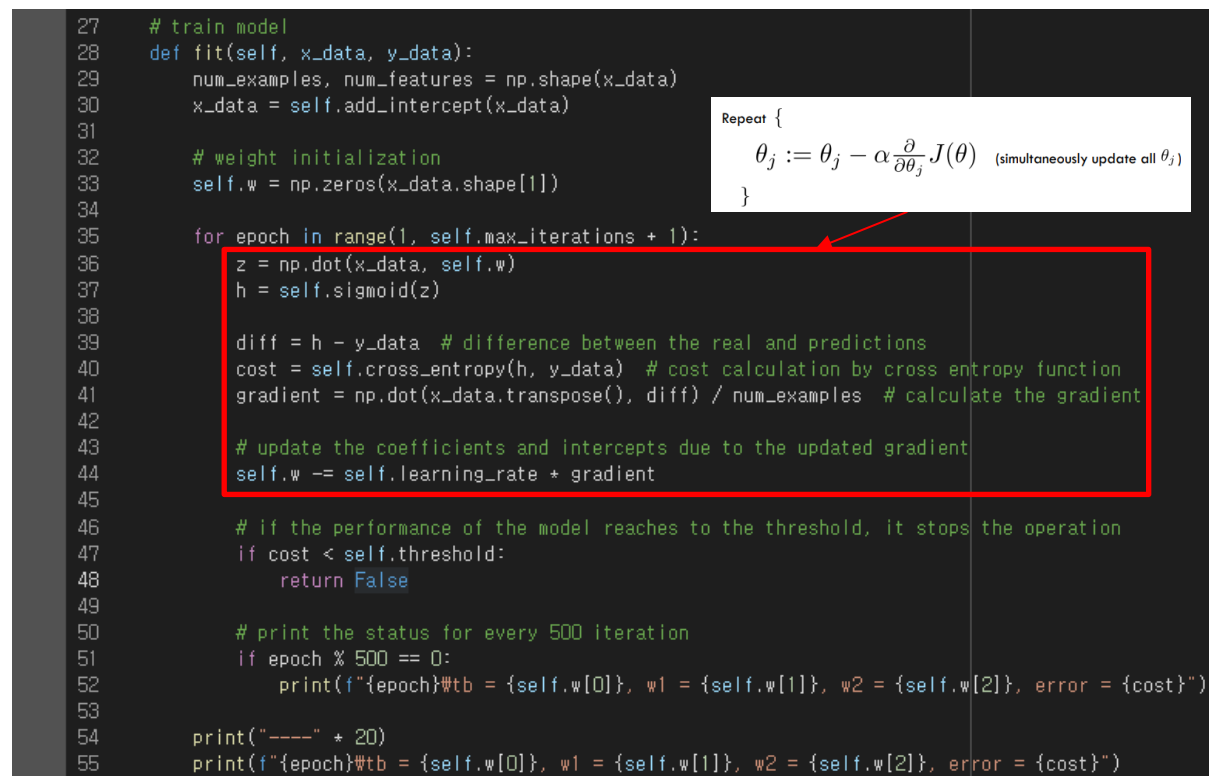
```
1 import numpy as np
2
3
4 class LogisticRegression:
5     def __init__(self, learning_rate=0.01, threshold=0.01, max_iterations=50000):
6         self.threshold = threshold
7         self.learning_rate = learning_rate
8         self.max_iterations = max_iterations
9
10
11     # return the coefficients and intercepts
12     def coeff(self):
13         return self.w
14
15
16     # add the intercept
17     def add_intercept(self, x_data):
18         self.intercept = np.ones((x_data.shape[0], 1))
19         return np.concatenate((self.intercept, x_data), axis=1)
20
21
22     # sigmoid function
23     def sigmoid(self, z):
24         return 1 / (1 + np.exp(-z))
25
26     # cost function
27     def cross_entropy(self, h, y):
28         return -(y * np.log(h) + (1 - y) * np.log(1 - h)).mean()
```

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$J(\theta) = -\frac{1}{m} \left(\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right)$$

Figure 13 - Logistic Regression Model 1

First, we implemented the logistic regression class that will be used for the classification. In Figure 14, we first define the learning rate, the threshold of the error, and the maximum number of iterations. The learning rate and the maximum number of iterations are used to control the gradient descent algorithm, and the threshold is set to avoid the overfitting situation. Then, we implemented the sigmoid function and cross-entropy function. The sigmoid function will be used to set the classification model, and the cross-entropy function will be used to evaluate the error so that we can use it in the fitting operation of the gradient descent algorithm.



```

27 # train model
28 def fit(self, x_data, y_data):
29     num_examples, num_features = np.shape(x_data)
30     x_data = self.add_intercept(x_data)
31
32     # weight initialization
33     self.w = np.zeros(x_data.shape[1])
34
35     for epoch in range(1, self.max_iterations + 1):
36         z = np.dot(x_data, self.w)
37         h = self.sigmoid(z)
38
39         diff = h - y_data # difference between the real and predictions
40         cost = self.cross_entropy(h, y_data) # cost calculation by cross entropy function
41         gradient = np.dot(x_data.transpose(), diff) / num_examples # calculate the gradient
42
43         # update the coefficients and intercepts due to the updated gradient
44         self.w -= self.learning_rate * gradient
45
46         # if the performance of the model reaches to the threshold, it stops the operation
47         if cost < self.threshold:
48             return False
49
50         # print the status for every 500 iteration
51         if epoch % 500 == 0:
52             print(f"{epoch}#tb = {self.w[0]}, w1 = {self.w[1]}, w2 = {self.w[2]}, error = {cost}")
53
54     print("----" * 20)
55     print(f"{epoch}#tb = {self.w[0]}, w1 = {self.w[1]}, w2 = {self.w[2]}, error = {cost}")

```

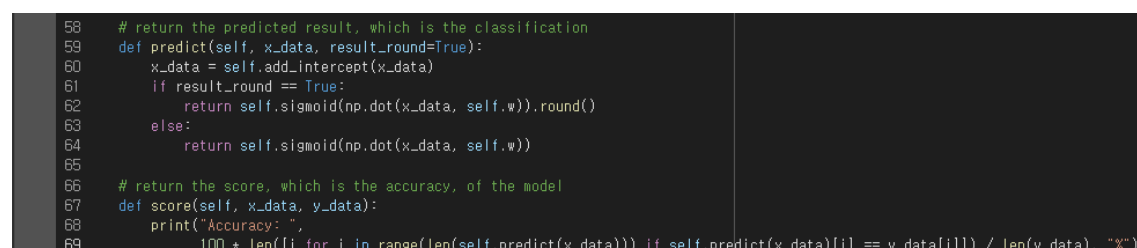
Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (\text{simultaneously update all } \theta_j)$$

}

Figure 14 - Logistic Regression Model 2

Second, we implemented the fitting function of the logistic regression model. In Figure 15, the approach used in this code is the gradient descent algorithm. While the error does not reach the threshold, the following function trains the logistic regression model. For each epoch, the fitting function calculates the prediction by using the implemented sigmoid function. After the prediction, it calculates the cost by using the implemented cross-entropy function and performs the prediction until the epoch reaches the maximum number of the iteration. For every 500 epochs, the following function prints out its fitting process, and at the end of the execution, it returns the final coefficients, intercepts, and the error.



```

58 # return the predicted result, which is the classification
59 def predict(self, x_data, result_round=True):
60     x_data = self.add_intercept(x_data)
61     if result_round == True:
62         return self.sigmoid(np.dot(x_data, self.w)).round()
63     else:
64         return self.sigmoid(np.dot(x_data, self.w))
65
66 # return the score, which is the accuracy, of the model
67 def score(self, x_data, y_data):
68     print("Accuracy: ",
69         100 * len([i for i in range(len(self.predict(x_data))) if self.predict(x_data)[i] == y_data[i]]) / len(y_data), "%")

```

Figure 15 - Logistic Regression Model 3

Third, we implemented the function that generates the prediction result for the classification, and the score function that returns the correct ratio of the prediction compared to the real label. The following functions are implemented in Figure 16.

```
1 # logistic regression model initialization
2 model = LogisticRegression()
3
4 # logistic regression model training
5 model.fit(train_data[['x1', 'x2']], train_data['y'])
```

Figure 16 -Initializing and Fitting the Logistic Regression Model

```
26500 b = 2.1245020759024142, w1 = -3.5628087813868574, w2 = -1.073852602982841, error = 0.10661991181923092
27000 b = 2.129897991141697, w1 = -3.569733758653435, w2 = -1.0763645197755922, error = 0.10660323195302245
27500 b = 2.135099062670621, w1 = -3.576412487161737, w2 = -1.078787763041129, error = 0.1065877228324728
28000 b = 2.1401135798145248, w1 = -3.582855165080661, w2 = -1.0811259787005498, error = 0.10657329556465568
28500 b = 2.1449493900120022, w1 = -3.589071467529605, w2 = -1.08338262608914, error = 0.10655986869949025
29000 b = 2.149613928841135, w1 = -3.595070580573552, w2 = -1.085560990138717, error = 0.10654736753296247
29500 b = 2.1541142475201895, w1 = -3.600861232480554, w2 = -1.0876641925742117, error = 0.10653572348366061
30000 b = 2.1584570381351305, w1 = -3.6064517225046755, w2 = -1.0896952022187587, error = 0.10652487353398282
30500 b = 2.162648656817211, w1 = -3.6118499474281855, w2 = -1.0916568444912695, error = 0.10651475972850509
31000 b = 2.1666951450686414, w1 = -3.6170634260713115, w2 = -1.093551810171491, error = 0.10650532872296836
31500 b = 2.170602249412477, w1 = -3.6220993219553335, w2 = -1.0953826634996113, error = 0.10649653137817418
32000 b = 2.1743754395233887, w1 = -3.6269644642852725, w2 = -1.097151849670445, error = 0.10648832239379397
32500 b = 2.1780199249792935, w1 = -3.6316653674008914, w2 = -1.0988617017760494, error = 0.10648065997771419
33000 b = 2.181540670758756, w1 = -3.636208248829753, w2 = -1.1005144472451331, error = 0.10647350554707093
33500 b = 2.1849424115962304, w1 = -3.6405990460622593, w2 = -1.1021122138228152, error = 0.10646682345759116
34000 b = 2.1882296652954665, w1 = -3.6448434321569207, w2 = -1.1036570351298722, error = 0.10646058075825662
34500 b = 2.1914067450914168, w1 = -3.6489468302733425, w2 = -1.1051508558368548, error = 0.10645474696865594
35000 b = 2.194477771141779, w1 = -3.65291442722102, w2 = -1.10659553648502, error = 0.10644929387669357
35500 b = 2.1974466812214035, w1 = -3.656751186103692, w2 = -1.1079928579829763, error = 0.10644419535459011
36000 b = 2.2003172406856617, w1 = -3.6604618581314603, w2 = -1.1093445258051762, error = 0.10643942719133945
36500 b = 2.2030930517625102, w1 = -3.664050993666212, w2 = -1.1106521739160176, error = 0.10643496693999278
37000 b = 2.2057775622273668, w1 = -3.667522952559918, w2 = -1.1119173684410824, error = 0.10643079377831717
37500 b = 2.2083740735098414, w1 = -3.670881913839344, w2 = -1.1131416111051389, error = 0.10642688838153437
38000 b = 2.21088574827687, w1 = -3.6741318847908393, w2 = -1.114326342454702, error = 0.10642323280598284
38500 b = 2.2133156175327224, w1 = -3.6772767094775283, w2 = -1.1154729448814171, error = 0.10641981038266919
39000 b = 2.215668587272815, w1 = -3.6803200767510833, w2 = -1.1165827454611452, error = 0.10641660561978293
39500 b = 2.2179414447248598, w1 = -3.683265527774743, w2 = -1.1176570186222137, error = 0.10641360411334339
40000 b = 2.2201428642080274, w1 = -3.686116463104542, w2 = -1.1186969886553164, error = 0.10641079246523369
40500 b = 2.222273412638218, w1 = -3.688876149356162, w2 = -1.1197038320763375, error = 0.10640815820795058
41000 b = 2.224335554704994, w1 = -3.691547725486944, w2 = -1.1206786798525372, error = 0.10640568973546724
41500 b = 2.2263316577436574, w1 = -3.694134208719717, w2 = -1.1216226195016443, error = 0.10640337623966445
42000 b = 2.228263996324106, w1 = -3.6966385001328472, w2 = -1.122536697072596, error = 0.10640120765183998
42500 b = 2.230134756576063, w1 = -3.6990633899390692, w2 = -1.1234219190159933, error = 0.1063991745888524
43000 b = 2.23194804026898, w1 = -3.7014115624739024, w2 = -1.124279253951712, error = 0.10639726830349866
43500 b = 2.2336998686632383, w1 = -3.703685600912686, w2 = -1.125109634340471, error = 0.10639548063876293
44000 b = 2.235398186148057, w1 = -3.7058879917339858, w2 = -1.1259139580656834, error = 0.10639380398560706
44500 b = 2.2370428636802506, w1 = -3.7080211289457257, w2 = -1.126693089931401, error = 0.10639223124400533
45000 b = 2.238635702036974, w1 = -3.7100873180890543, w2 = -1.127447863081719, error = 0.1063907557869522
45500 b = 2.2401784348944433, w1 = -3.712088780034059, w2 = -1.128179080346639, error = 0.10638937142719693
46000 b = 2.2416727317438774, w1 = -3.71402765458018, w2 = -1.1288875155189515, error = 0.1063880723864814
46500 b = 2.2431202006549142, w1 = -3.7159060038733736, w2 = -1.129573914566465, error = 0.10638685326707754
47000 b = 2.2445223908961323, w1 = -3.717725815651268, w2 = -1.1302389967834494, error = 0.10638570902543763
47500 b = 2.2458807954214843, w1 = -3.7194890063265036, w2 = -1.1308834558850525, error = 0.10638463494778959
48000 b = 2.247196853230878, w1 = -3.721197423918058, w2 = -1.1315079610480345, error = 0.10638362662752096
48500 b = 2.248471951612551, w1 = -3.7228528508394123, w2 = -1.1321131579010306, error = 0.10638267994421184
49000 b = 2.249707428274318, w1 = -3.7244570065519653, w2 = -1.132699669467236, error = 0.10638179104418616
49500 b = 2.2509045733702613, w1 = -3.726011550091425, w2 = -1.1332680970623532, error = 0.10638095632246435
50000 b = 2.2520646314290436, w1 = -3.727518082474473, w2 = -1.1338190211502568, error = 0.10638017240600826
-----
50000 b = 2.2520646314290436, w1 = -3.727518082474473, w2 = -1.1338190211502568, error = 0.10638017240600826
```

Figure 17 - Training Process of the Logistic Regression Model

Using the implemented class, we initialized our logistic regression model and trained the model using the given train data set. Figure 16 shows the code for the following operation, and Figure 17 shows the training process generated by the following codes.

3.3 Task 2: Classify the Data in the Test Data Set using the Trained Model

```
1 from tabulate import tabulate
2 from sklearn.metrics import log_loss
3
4
5 test_data = pd.read_csv("./hw2_test.csv")
6 train_data = pd.read_csv("./hw2_train.csv")
7
8 test_data_0 = test_data[test_data['y'] == 0]
9 test_data_1 = test_data[test_data['y'] == 1]
10
11 # initialize the coefficients and intercept
12 b, w1, w2 = model.w
13 c, m = -b/w2, -w1/w2
14 p_train = m*np.sort(train_data['x1'])+c
15
16 b, w1, w2 = model.w
17 c, m = -b/w2, -w1/w2
18 p_test = m*np.sort(test_data['x1'])+c
19
20 # cost of task 1 and task 2
21 index = ['Task 1', 'Task 2']
22 df = pd.DataFrame({'cost' : [model.cross_entropy(model.predict(train_data[['x1', 'x2']], False), train_data['y']),
23                               model.cross_entropy(model.predict(test_data[['x1', 'x2']], False), test_data['y'])]},
24                   index=index)
25 print(tabulate(df, headers='keys', tablefmt='pretty') + '\n')
```

Figure 18 - Classification on the Test Data Set by the Implemented Logistic Regression Model

```
1 from tabulate import tabulate
2 from sklearn.metrics import log_loss
3 from sklearn.linear_model import LogisticRegression
4
5
6 # model initialization and prediction
7 model = LogisticRegression().fit(train_data[['x1', 'x2']], train_data['y'])
8 p = model.predict(test_data[['x1', 'x2']])
9
10
11 test_data = pd.read_csv("./hw2_test.csv")
12 train_data = pd.read_csv("./hw2_train.csv")
13
14 test_data_0 = test_data[test_data['y'] == 0]
15 test_data_1 = test_data[test_data['y'] == 1]
16
17 # initialize the coefficients and intercept
18 b = model.intercept_
19 w1, w2 = model.coef_[0]
20 c, m = -b/w2, -w1/w2
21 p_train = m*np.sort(train_data['x1'])+c
22
23 b = model.intercept_
24 w1, w2 = model.coef_[0]
25 c, m = -b/w2, -w1/w2
26 p_test = m*np.sort(test_data['x1'])+c
27
28 # coefficients and intercept
29 index = ['Task 1']
30 df = pd.DataFrame({'b' : model.intercept_[0],
31                   'w1' : model.coef_[0][0],
32                   'w2' : model.coef_[0][1]},
33                   index=index)
34 print(tabulate(df, headers='keys', tablefmt='pretty') + '\n')
35
36 # cost of task 1 and task 2
37 index = ['Task 1', 'Task 2']
38 df = pd.DataFrame({'cost' : [log_loss(train_data['y'], model.predict(train_data[['x1', 'x2']])),
39                               log_loss(test_data['y'], model.predict(test_data[['x1', 'x2']])))],
40                   index=index)
41 print(tabulate(df, headers='keys', tablefmt='pretty') + '\n')
```

Figure 19 - Classification on the Test Data Set by the Built-In Logistic Regression Model

Using the implemented logistic regression model, which is trained by the given train data set, we classified the data in the test data set. First, we initialized the coefficients and intercept according to the variables x1 and x2 to plot the graph. Second, we generated the matrix which represents the cost for task 1 and task 2. After this process, we plot the generated classification model with the given test dataset. The following code implementation is presented in Figure 18. Also, for the comparison of the implemented logistic regression model with the built-in models, we implemented the same operation by using the built-in logistic regression model. The following operations are presented in Figure 19.

3.4 Extra: Classification with the Polynomial Logistic Regression Model

```

1 from tabulate import tabulate
2 from sklearn.pipeline import Pipeline
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.preprocessing import PolynomialFeatures
5
6
7 np.seterr(divide='ignore', invalid='ignore')
8 test_data = pd.read_csv('./hw2_test.csv')
9 train_data = pd.read_csv('./hw2_train.csv')
10
11 poly = PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)
12 x_poly = poly.fit_transform(train_data[['x1', 'x2']])
13
14 model = LogisticRegression()
15 model.fit(x_poly, train_data['y'])
16
17 pipe = Pipeline([('polynomial_features', poly), ('logistic_regression', model)])
18 pipe.fit(train_data[['x1', 'x2']], train_data['y'])
19
20
21 test_data = pd.read_csv('./hw2_test.csv')
22 train_data = pd.read_csv('./hw2_train.csv')
23
24 test_data_0 = test_data[test_data['y'] == 0]
25 test_data_1 = test_data[test_data['y'] == 1]
26
27 # initialize the coefficients and intercept
28 x = np.sort(train_data['x1'])
29 b = model.intercept_[0]
30 w1, w2, w3, w4, w5 = model.coef_[0]
31 p_train = -(w2 + w3*x + np.sqrt(w2**2 + 2*w2*w3*x + w3**2*x**2 - 4*w4*w5*x**2 - 4*w1*w5*x - 4*b*w5))/(2*w5)
32
33 x = np.sort(test_data['x1'])
34 b = model.intercept_[0]
35 w1, w2, w3, w4, w5 = model.coef_[0]
36 p_test = -(w2 + w3*x + np.sqrt(w2**2 + 2*w2*w3*x + w3**2*x**2 - 4*w4*w5*x**2 - 4*w1*w5*x - 4*b*w5))/(2*w5)
37
38 # coefficients and intercept
39 index = ['Task 1']
40 df = pd.DataFrame({'b' : model.intercept_[0],
41                    'w1' : model.coef_[0][0],
42                    'w2' : model.coef_[0][1],
43                    'w3' : model.coef_[0][2],
44                    'w4' : model.coef_[0][3],
45                    'w5' : model.coef_[0][4]},
46                    index=index)
47 print(tabulate(df, headers='keys', tablefmt='pretty') + '\n')
48
49 # cost of task 1 and task 2
50 index = ['Task 1', 'Task 2']
51 df = pd.DataFrame({'cost' : [log_loss(train_data['y'], pipe.predict(train_data[['x1', 'x2']])),
52                               log_loss(test_data['y'], pipe.predict(test_data[['x1', 'x2']]))],
53                    index=index)
54 print(tabulate(df, headers='keys', tablefmt='pretty') + '\n')

```

Figure 20 - Classification on the Test Data Set by the Built-In Polynomial Logistic Regression Model

In this section, we will discuss our extra-implemented logistic regression model that uses the polynomial decision boundary, not the linear decision boundary. Implementing the prediction model itself is not a big problem. However, visualizing the result of the prediction model was the biggest difficulty. As the number of coefficients for the following model increased, it takes some effort to implement the following function with the variables x_1 and x_2 .

```

13 test_score, test_error = [], []
14 train_score, train_error = [], []
15
16 for i in range(1, 20):
17     poly = PolynomialFeatures(degree=i, interaction_only=False, include_bias=False)
18     x_poly = poly.fit_transform(train_data[['x1', 'x2']])
19
20     model = LogisticRegression(max_iter=100000)
21     model.fit(x_poly, train_data['y'])
22
23     pipe = Pipeline([('polynomial_features', poly), ('logistic_regression', model)])
24     pipe.fit(train_data[['x1', 'x2']], train_data['y'])
25
26     test_error.append(log_loss(test_data['y'], pipe.predict(test_data[['x1', 'x2']]])))
27     train_error.append(log_loss(train_data['y'], pipe.predict(train_data[['x1', 'x2']]])))
28
29     test_score.append(pipe.score(test_data[['x1', 'x2']], test_data['y']))
30     train_score.append(pipe.score(train_data[['x1', 'x2']], train_data['y']))

```

Figure 21 - Polynomial Logistic Regression Initialization of Degree from 1 to 20

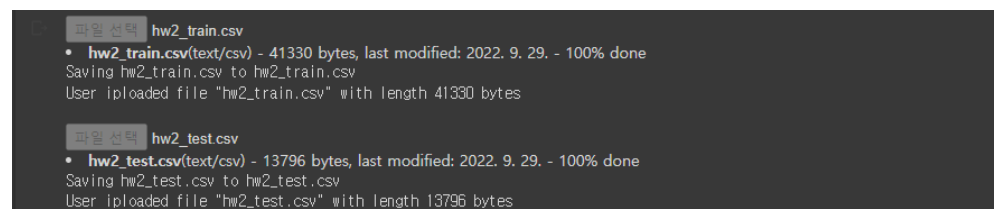
Also, by using the same code, we implemented the code that generates the errors and accuracy of various versions of the polynomial regression model by editing the degree of the polynomial model.

4 Build Environment

The following build environments are required to execute the implemented code for task 1, task 2, and extra implemented prediction model.

Building Environment:

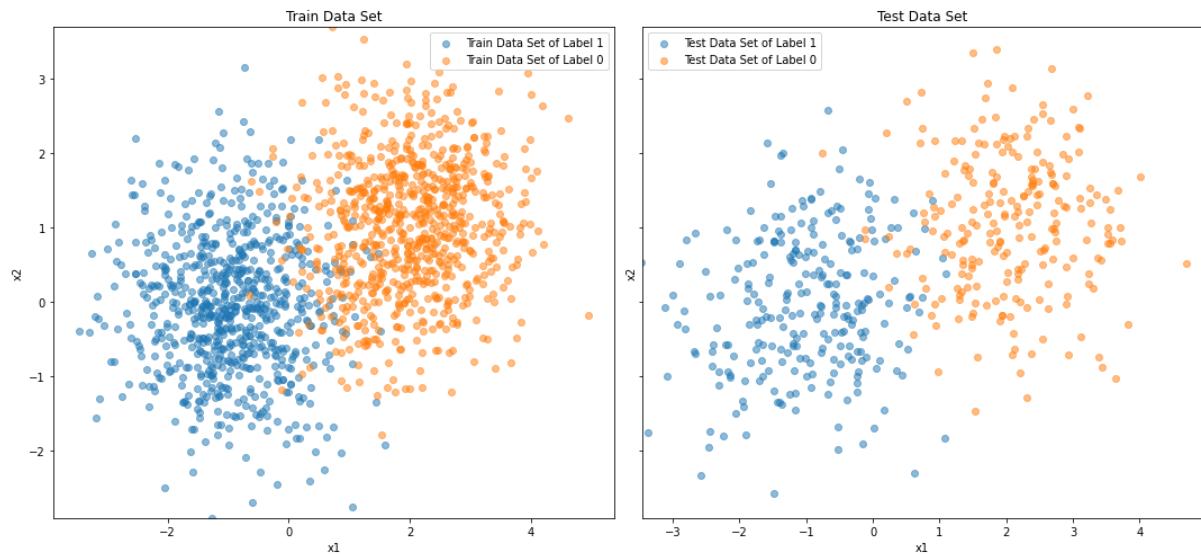
- The environment that can open the ipython notebook: COLAB, Jupyter Notebook
- Each cell in the notebook should be executed sequentially.



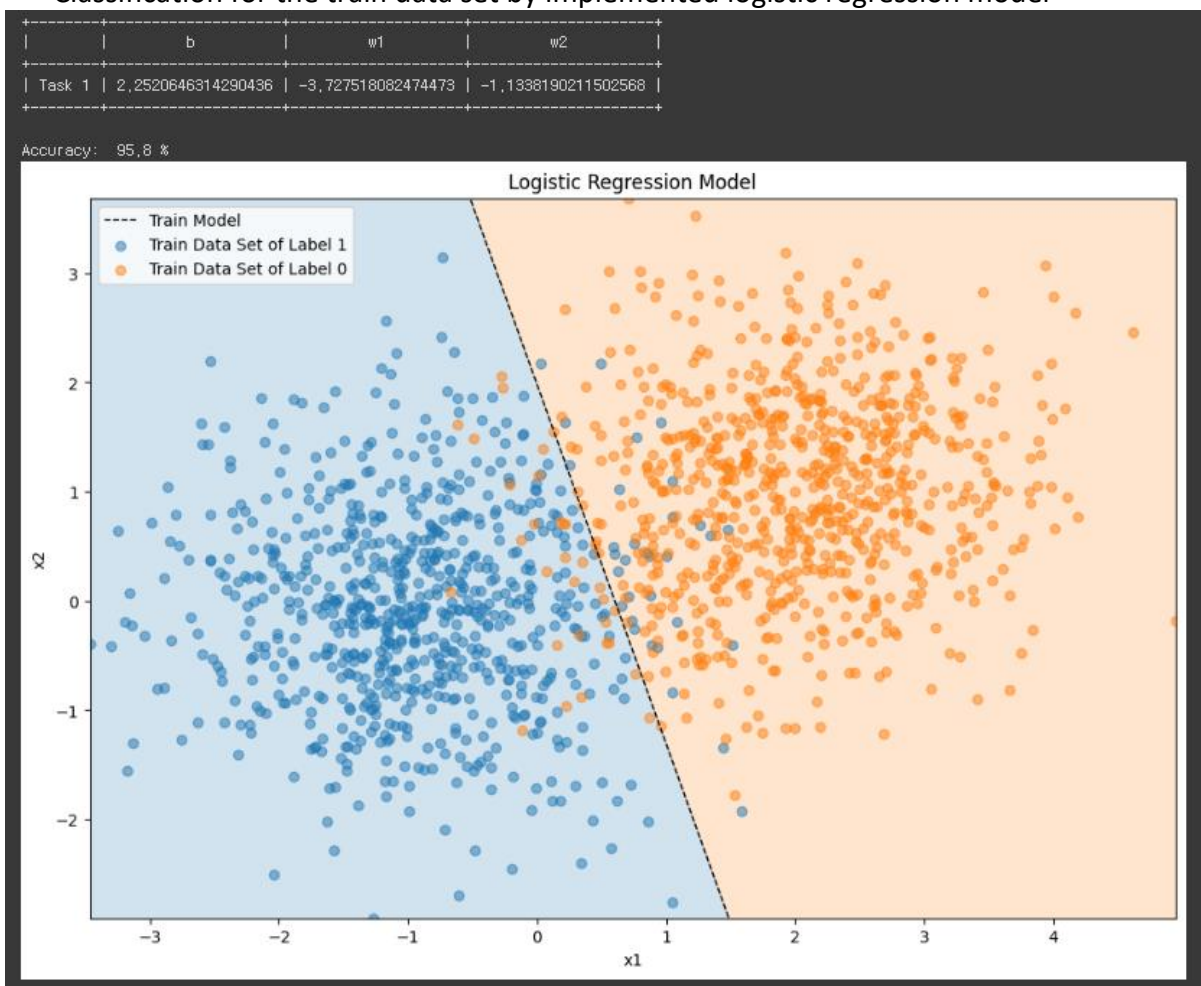
File Upload: Before cell execution, the given data set must be uploaded first.
Both train and test data set should be uploaded.

5 Results

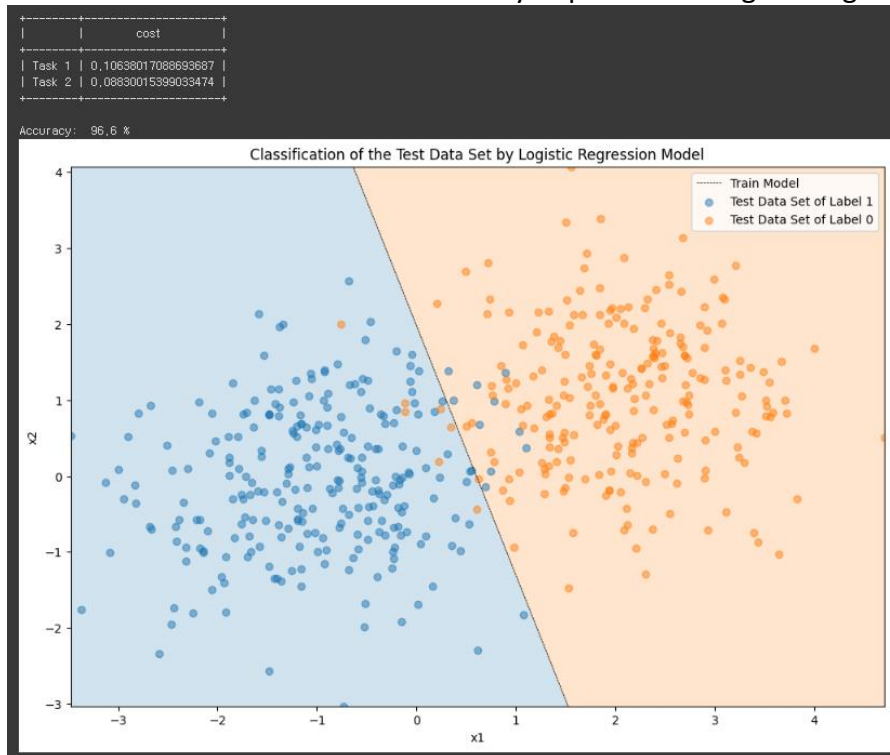
- Visualization of the given train and test data set of x_1 and x_2 , which are labeled by y .



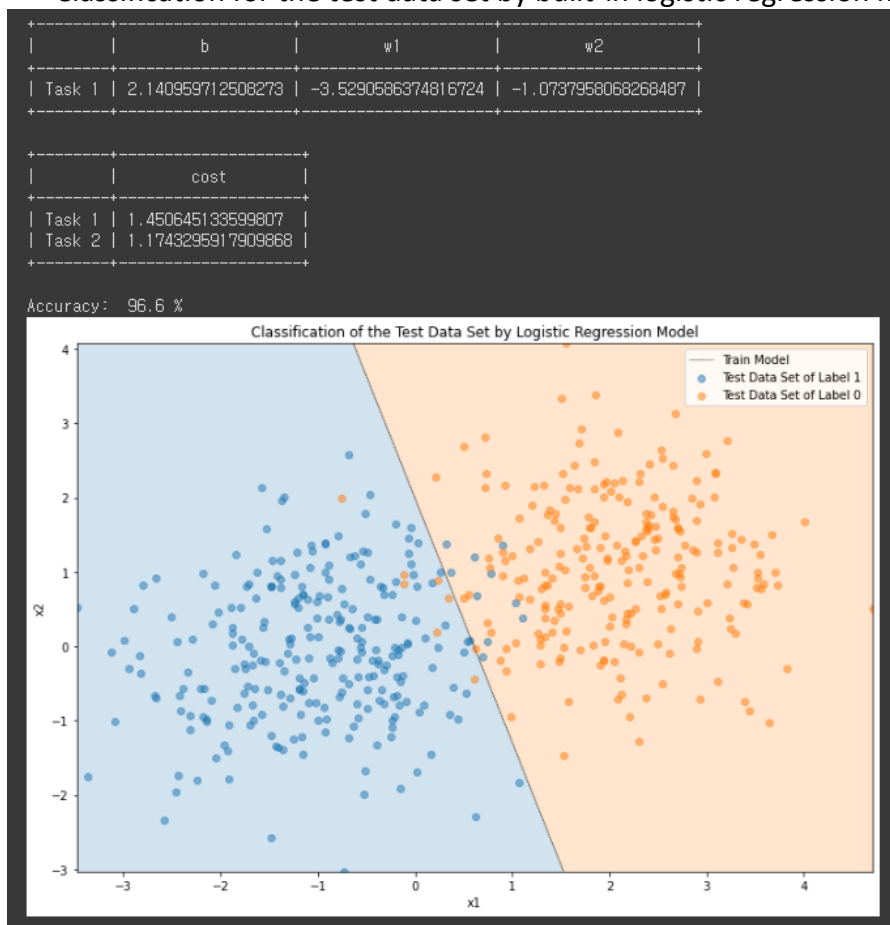
- Classification for the train data set by implemented logistic regression model



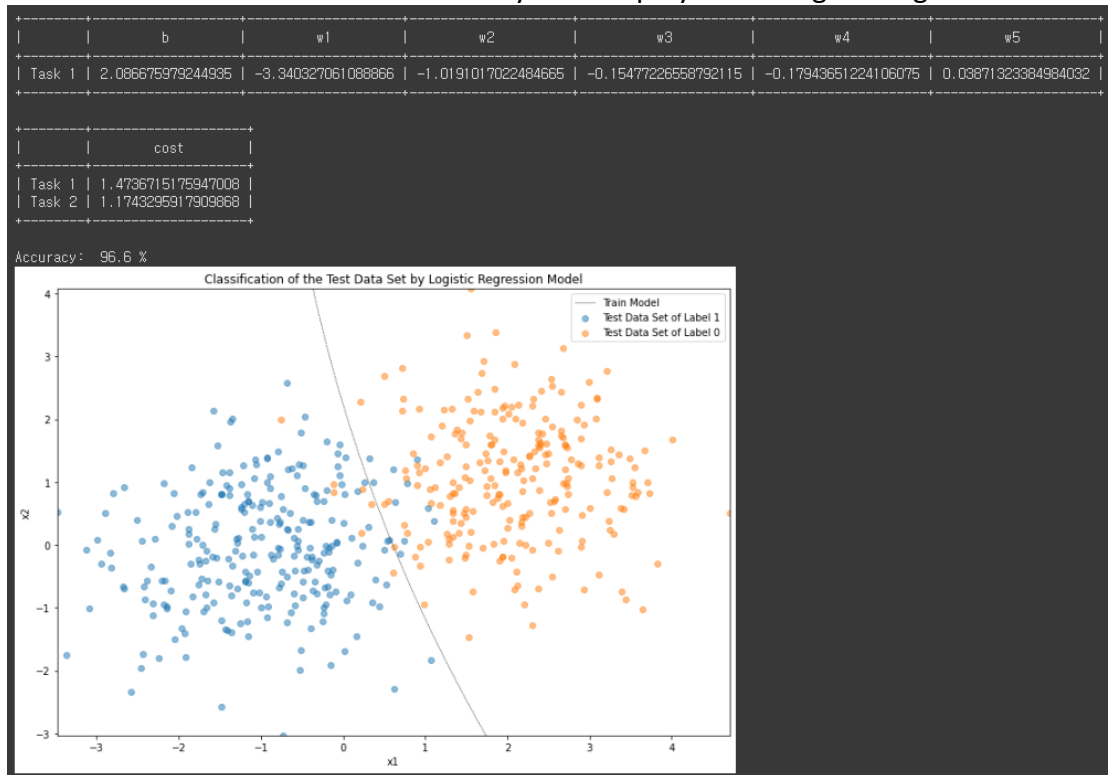
- Classification for the test data set by implemented logistic regression model



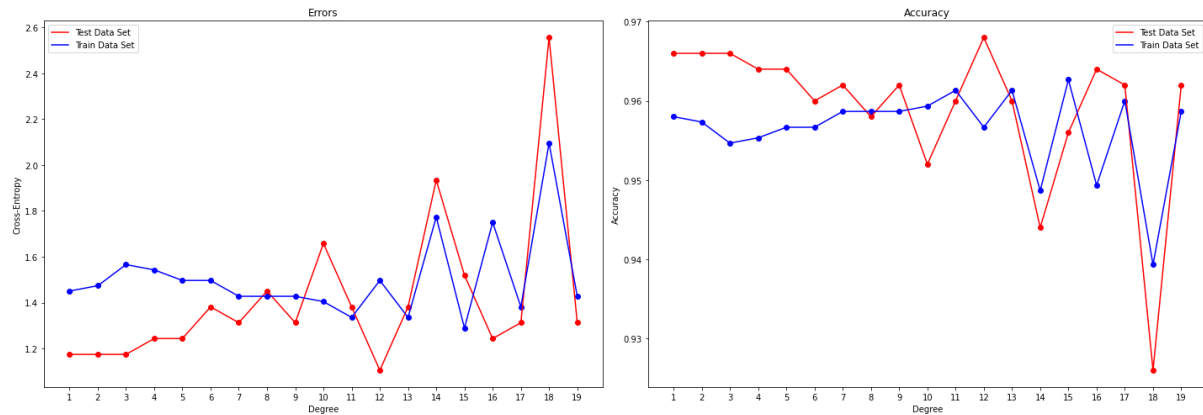
- Classification for the test data set by built-in logistic regression model



- Classification for the test data set by built-in polynomial logistic regression model



- Errors and accuracies of the polynomial logistic regression model of different degrees.



6 Evaluation

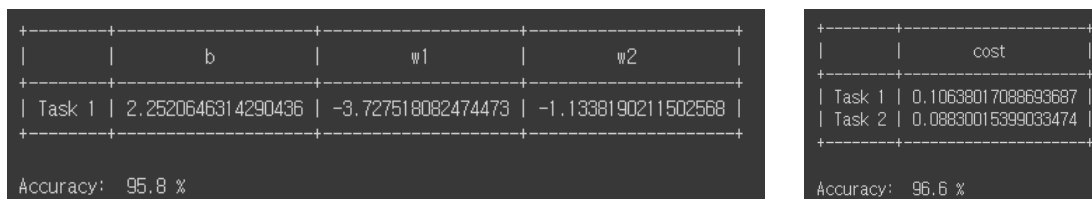


Figure 22 - Classification Result of Train and Test Data Set by Implemented Logistic Regression Model

	b	w1	w2
Task 1	2.140959712508273	-3.5290586374816724	-1.0737958068268487

	cost
Task 1	1.450645133599807
Task 2	1.1743295917909868

Accuracy: 96.6 %

Figure 23 - Classification Result of the Train and Test Data Set by Built-In Logistic Regression Model

	b	w1	w2	w3	w4	w5
Task 1	2.086675979244935	-3.340327061088866	-1.0191017022484665	-0.15477226558792115	-0.17943651224106075	0.03871323384984032

	cost
Task 1	1.4736715175947008
Task 2	1.1743295917909868

Accuracy: 96.6 %

Figure 24 - Classification Result of the Train and Test Data Set by Polynomial Logistic Regression Model

Figures 22 to 24 show the result of the coefficients, intercept, cost, and accuracy for each logistic regression model. Let's compare the results presented in Figures 21 and 22, which are the result of the implemented and built-in logistic regression model. Both models show almost the same coefficients, intercept, and accuracy. There are only a few differences between the coefficients and intercept, and the accuracy of both two models is the same. Since the cost of task 1 and task 2 for the implemented logistic regression model is slightly lower than the built-in model, the implemented model slightly performed well on classification than the built-in model.

In the case of the polynomial logistic regression model, which result is presented in Figure 24, It showed the same accuracy as the former classification models. However, due to the more coefficients and the complex operation of the equation compared to the former models, its performance is lower than the former logistic regression models.

Logistic Regression Model with Degree of 12	
Cost	1.1052488406114447
Accuracy	96.8%

Figure 25 - Error and Accuracy of the Polynomial Logistic Regression Model of Degree 12

According to the graph presented in the errors and accuracies of the polynomial logistic regression model of different degrees, a model with degree 12 shows the best accuracy in the classification of the test data set. However, because it shows a lower accuracy in the classification of the trained model, we cannot say this is the ideal classification model.

7 Conclusion

A logistic regression model is a statistical analysis method to predict a binary outcome, such as true or false, based on the prior observations of the given data set. This model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables. Also, a logistic regression model can be taken into consideration of multiple input criteria. This model is important because it transforms complex calculations around probability into a straightforward arithmetic problem (Lawton et al., 2022).

In this paper, we have explained the concepts that are used to implement the logistic regression model: logistic regression, sigmoid function, and decision boundary. Also, we discussed the cost function in the classification model, which is a cross-entropy function. Then, we presented tasks 1 and task 2, which are the implementation of the logistic regression model with the training by using the training data set and the classification of the test data set using the implemented logistic regression model. After that, we compared the performance of the model to the built-in logistic regression model. The coefficients and intercepts were almost the same among the two models, but according to the cost difference between the two models, the implemented model showed slightly better performance than the built-in model. In the end, we presented the polynomial logistic regression model implemented by the built-in model. Despite that, there were more coefficients, and the operation was complex, it showed the same accuracy as the previously implemented models.

By understanding this paper, we can understand the basic concepts of the logistic regression model and the formulas that are used to implement the following model: logistic regression, sigmoid function, decision boundary, and cross-entropy function. Also, we can understand the comparison between the logistic regression model with the linear and polynomial graphs.

Citations

- [1] *What is logistic regression?* IBM. (n.d.). Retrieved October 1, 2022, from <https://www.ibm.com/topics/logistic-regression>
- [2] Lee, K.-H. (n.d.). Logistic Regression.
- [3] *Logistic regression. Logistic Regression - an overview* | ScienceDirect Topics. (n.d.). Retrieved October 1, 2022, from <https://www.sciencedirect.com/topics/computer-science/logistic-regression>
- [4] Lawton, G., Burns, E., & Rosencrance, L. (2022, January 20). *What is logistic regression? - definition from Searchbusinessanalytics*. SearchBusinessAnalytics. Retrieved October 2, 2022, from <https://www.techtarget.com/searchbusinessanalytics/definition/logistic-regression>