CSE 412: Machine Learning Lab
Lab Activity 4: Logistic Regression

Dr Muhammad Abul Hasan*

October 22, 2023

1 Objective

In this lab activity, you will learn:

- **Understanding Binary Classification**: Enable students to comprehend how Logistic Regression is used for binary classification problems, where data points are categorized into one of two classes.
- **Probability Estimation**: Teach students the concept that Logistic Regression models the probability of an input belonging to a specific class, a fundamental concept in machine learning.
- **Model Training and Parameters**: Equip students with the knowledge and skills to train a Logistic Regression model and optimize its parameters using methods such as maximum likelihood estimation or gradient descent. regression

2 Logistic regression

Logistic regression is a statistical method used to analyze and model the relationship between a categorical dependent variable and one or more independent variables. It is a type of generalized linear model that uses the logistic function to model the probability of the dependent variable being in one category or another, given the values of the independent variables.

Logistic regression is commonly used in binary classification problems, where the dependent variable takes on two values, such as "yes" or "no," "success" or "failure," or "0" or "1". However, it can also be used for multinomial classification problems, where the dependent variable takes on three or more values.

The logistic regression model estimates the coefficients of the independent variables that affect the probability of the dependent variable being in a certain category. These coefficients can be interpreted as the amount of change in the log-odds of the dependent variable for a one-unit change in the corresponding independent variable, holding all other independent variables constant.

Logistic regression has many applications in various fields such as medicine, marketing, and social sciences. It is often used in predicting the likelihood of an event or outcome, such as the probability of a patient having a certain disease based on their medical history or the likelihood of a customer purchasing a product based on their demographic information.

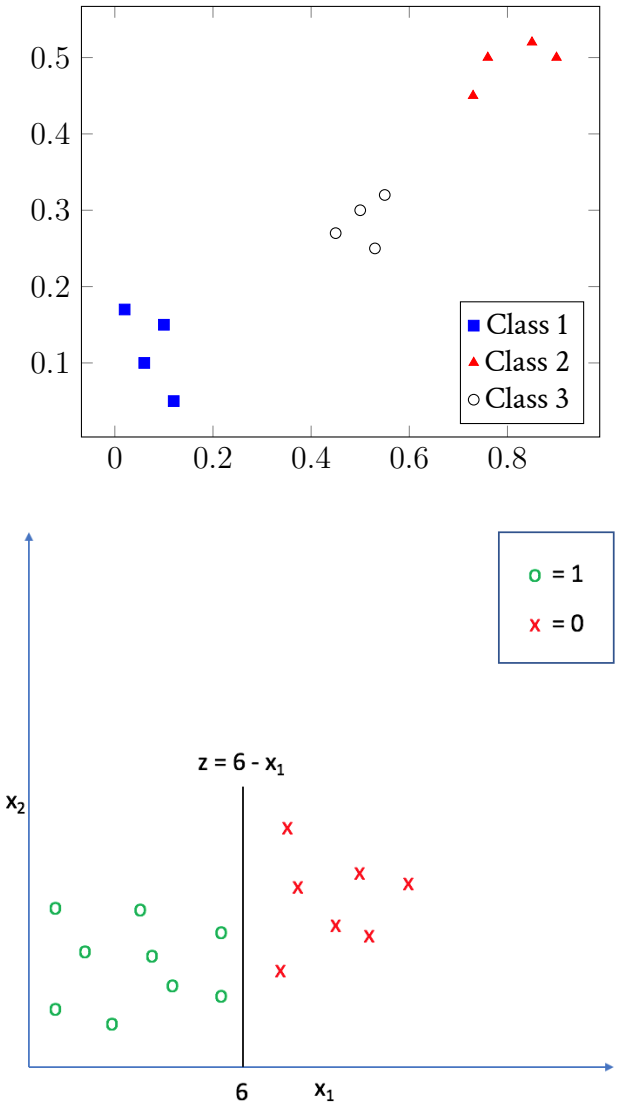
The goal of logistic regression, as with any classifier, is to figure out some way to split the data to allow for an accurate prediction of a given observation's class using the information present in the features. (For instance, if we were examining the Iris flower dataset, our classifier would figure out some method to split the data based on the following: sepal length, sepal width, petal length, petal width.) In the case of a generic two-dimensional example, the split might look something like this.

*Associate Professor, Deptartment of CSE, GUB ⁘ ✉ muhammad.hasan@cse.green.edu.bd

2.1 The decision boundary

Let us suppose we define a line that is equal to zero along this decision boundary. This way, all of the points on one side of the line take on positive values and all of the points on the other side take on negative values.

For example, in the following graph, $z = 6 - x_1$ represents a decision boundary for which any values of $x_1 > 6$ will return a negative value for z and any values of $x_1 < 6$ will return a positive value for z.



We can extend this decision boundary representation as any linear model, with or without additional polynomial features.

$$z(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \theta^T x$$

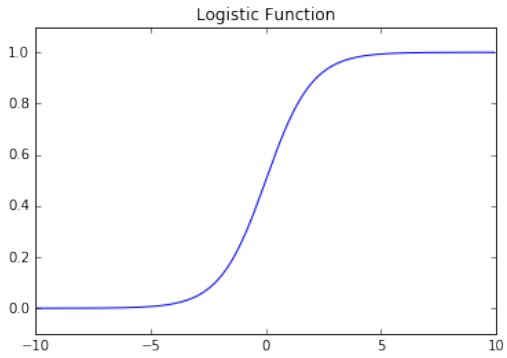
Adding polynomial features allows us to construct nonlinear splits in the data in order to draw a more accurate decision boundary.

As we continue, think of $z(x)$ as a measure of how far a certain data point exists from the decision boundary.

2.2 The logistic function

Next, let us take a look at the logistic function.

$$g(z) = \frac{1}{1 + e^{-z}}$$



As you can see, this function is asymptotically bounded between 0 and 1. Further, for very positive inputs our output will be close to 1 and for very negative inputs our output will be close to 0. This will essentially allow us to translate the value we obtain from z into a prediction of the proper class. Inputs that are close to zero (and thus, near the decision boundary) signify that we do not have a confident prediction of 0 or 1 for the observation.

At the decision boundary $z = 0$, the function $g(z) = 0.5$. We'll use this value as a cutoff establishing the prediction criterion:

$$h_\theta(x_i) \geq 0.5 \rightarrow y_{pred} = 1$$

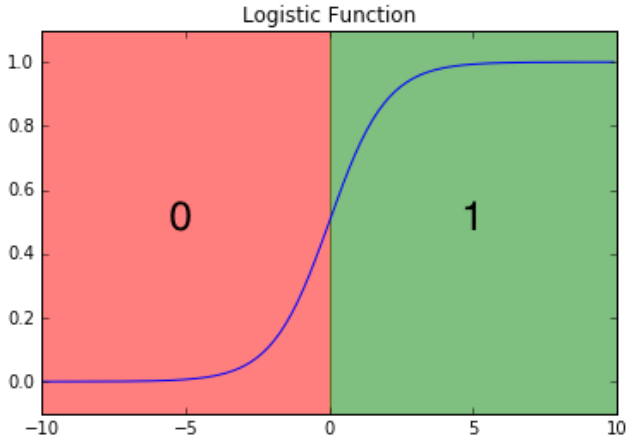
$$h_\theta(x_i) < 0.5 \rightarrow y_{pred} = 0$$

where y_{pred} denotes the predicted class of an observation, x_i, and $h_\theta(x)$ represents the functional composition, $h_\theta(x) = g(z(x))$

More concretely, we can write our model as:

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

Thus, the output of this function represents a binary prediction for the input observation's class.



Another way to interpret this output is to view it in terms of a probabilistic prediction of the true class. In other words, $h_\theta(x)$ represents the estimated probability that $y_i = 1$ for a given input, x_i.

$$h_\theta(x_i) = \mathrm{P}(y_i = 1 | x_i; \theta)$$

Because the class can only take on values of 0 or 1, we can also write this in terms of the probability that $y_i = 0$ for a given input, x_i.

$$\mathrm{P}\left(y_i = 0|x_i;\theta\right) = 1 - \mathrm{P}\left(y_i = 1|x_i;\theta\right)$$

For example, if $h_\theta(x) = 0.85$ then we can assert that there is an 85% probability that $y_i = 1$ and a 15% probability that $y_i = 0$. This is useful as we cannot only predict the class of an observation, but we can quantify the certainty of such prediction. In essence, the further a point is from the decision boundary, the more certain we are about the decision.

2.3 The cost function

Next, we need to establish a cost function which can grade how well our model is performing according to the training data. This cost function, $J(\theta)$, can be considered to be a summation of individual "grades" for each classification prediction in our training set, comparing $h_\theta(x)$ with the true class y_i. We want the cost function to be large for incorrect classifications and small for correct ones so that we can minimize $J(\theta)$ to find the optimal parameters.

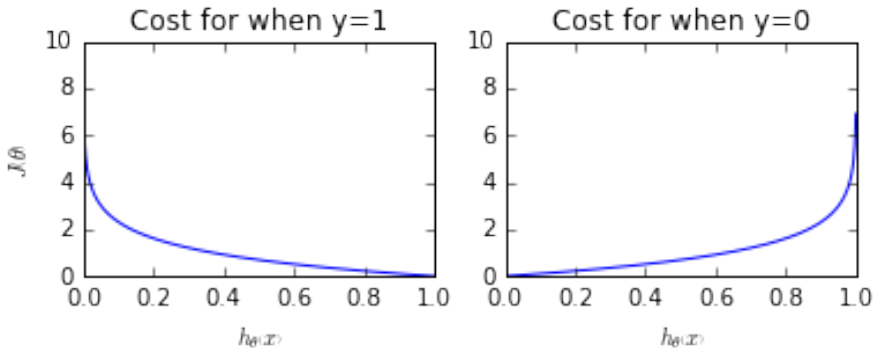
$$J\left(\theta\right) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta\left(x_i\right), y_i)$$

In linear regression, we used the squared error as our grading mechanism. Unfortunately for logistic regression, such a cost function produces a nonconvex space that is not ideal for optimization. There will exist many local optima on which our optimization algorithm might prematurely converge before finding the true minimum.

Using the Maximum Likelihood Estimator from statistics, we can obtain the following cost function which produces a convex space friendly for optimization. This function is known as the binary cross-entropy loss.

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

These cost functions return high costs for incorrect predictions.



More succinctly, we can write this as

$$\mathrm{Cost}(h_\theta\left(x\right), y) = -y\log\left(h_\theta\left(x\right)\right) - \left(1 - y\right)\log\left(1 - h_\theta\left(x\right)\right)$$

since the second term will be zero when $y = 1$ and the first term will be zero when $y = 0$. Substituting this cost into our overall cost function we obtain:

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

2.4 Finding the best parameters using Gradient Descent

The algorithm for gradient descent is provided below.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Taking the partial derivative of the cost function from the previous section, we find:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_{i,j}$$

And thus our parameter update rule for gradient descent is:

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_{i,j}$$

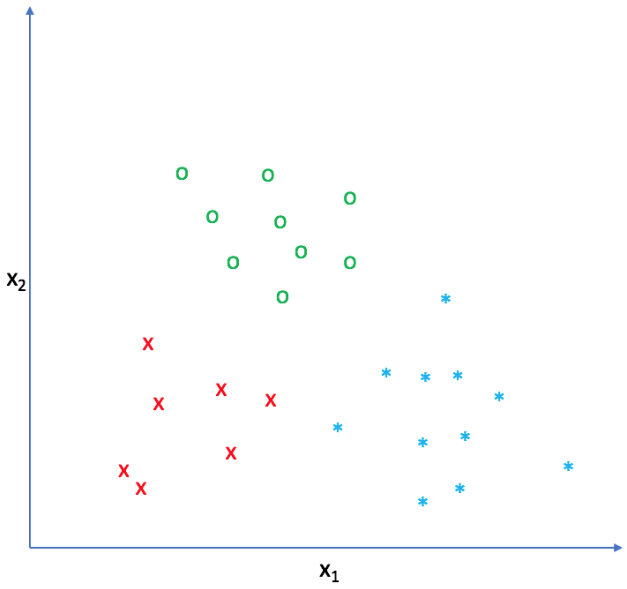
2.5 Advanced optimizations

In practice, there are more advanced optimization techniques that perform much better than gradient descent including conjugate gradient, BFGS, and L-BFGS. These advanced numerical techniques are not within the current scope of this course.

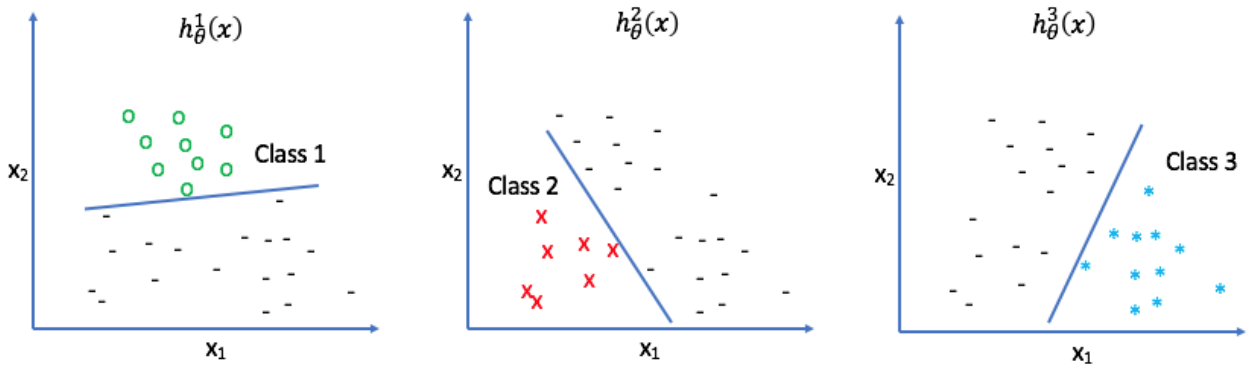
In sklearn's implementation of logistic regression, you can define which optimization is used with the solver parameter.

3 Multiclass Classification

While binary classification is useful, it'd be really nice to be able to train a classifier that can separate the data into more than two classes. For example, the Iris dataset I mentioned in the beginning of this post has three target classes: Iris Setosa, Iris Versicolour, and Iris Virginica. A simple binary classifier simply wouldn't work for this dataset!



However, using a one-vs-all (also known as one-vs-rest) classifier we can use a combination of binary classifiers for multiclass classification. We simply look at each class, one by one, and treat all of the other classes as the same. In a sense, you look at "Class 1" and build a classifier that can predict "Class 1" or "Not Class 1", and then do the same for each class that exists in your dataset.



In other words, each classifier $h_\theta^i(x)$ returns the probability that an observation belongs to class i. Therefore, all we have to do in order to predict the class of an observation is to select the class of whichever classifier returns the highest probability.

4 Iris Dataset Classification

The Iris dataset is a popular dataset in machine learning, consisting of 150 samples of iris flowers, each with four features (sepal length, sepal width, petal length, and petal width) and a target variable specifying the type of iris (Setosa, Versicolour, or Virginica).

Here is an example code for performing logistic regression on the Iris dataset using scikit-learn library:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# load the iris dataset
iris = load_iris()

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.3, random_state=42)

# create a logistic regression model
log_reg = LogisticRegression()

# fit the model on the training data
log_reg.fit(X_train, y_train)

# make predictions on the testing data
y_pred = log_reg.predict(X_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

In this code, we first load the Iris dataset using the load_iris() function from scikit-learn. Then, we split the data into training and testing sets using the train_test_split() function. We then create a logistic regression model using the LogisticRegression() function and fit it on the training data using the fit() method. Next, we make predictions on the testing data using the predict() method and calculate the accuracy of the model using the accuracy_score() function. Finally, we print out the accuracy of the model.

Note that this is a simple example and there are many other ways to improve the performance of the model, such as tuning hyperparameters, feature engineering, and using more advanced machine learning algorithms.

5 Lab Exercises

Please code yourself and write a report based on your findings:

1. Implement logistic regression from scratch (i.e., without using any machine learning library) and compare the performance with the one of scikit-learn.
2. Evaluate the model performance using Sepal Length vs Petal Width only.

My Notes Date:_____