Logistic Regression

This lesson will focus on logistic regression in Python.

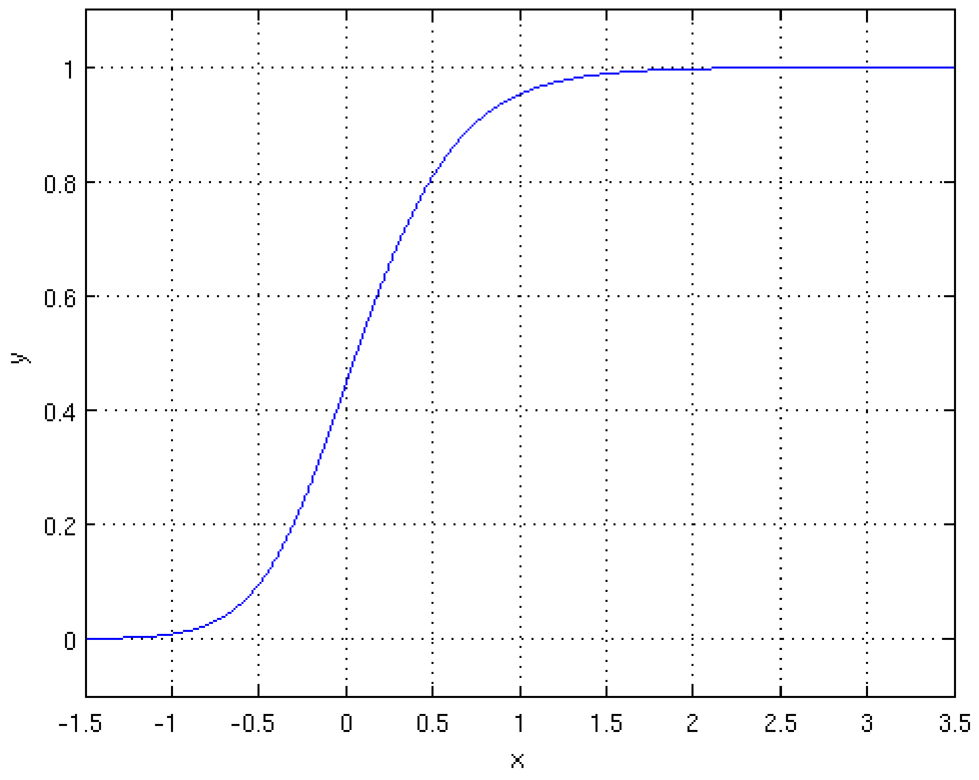
Until now, we have been predicting numerical quantities. But what if we want a model to predict a categorical variable? Categorical data is divided into distinct classes. The task of predicting a categorical variable is known as **classification**. We can perform classification using logistic regression.

Logistic function

Logistic Regression is a classification method that is built on the same concept as linear regression. In linear regression, we take a linear combination of different variables plus an intercept term to predict the output. But in classification problems, the predicted variable is categorical. The simplest case of classification is when the predicted variable is binary, i.e., it has only two classes, e.g., yes/no, male/female, etc. Logistic regression also takes the linear combination of different variables plus the intercept term, but afterward, it takes the result and passes it through a **logistic** function. The logistic function also known as **sigmoid** is defined as:

$$sigmoid(t) = \frac{1}{1 + e^{-t}}$$

where t is the output of the linear regression equation, i.e., linear combination of variables plus the intercept term. Let's look at the plot of the logistic function below.

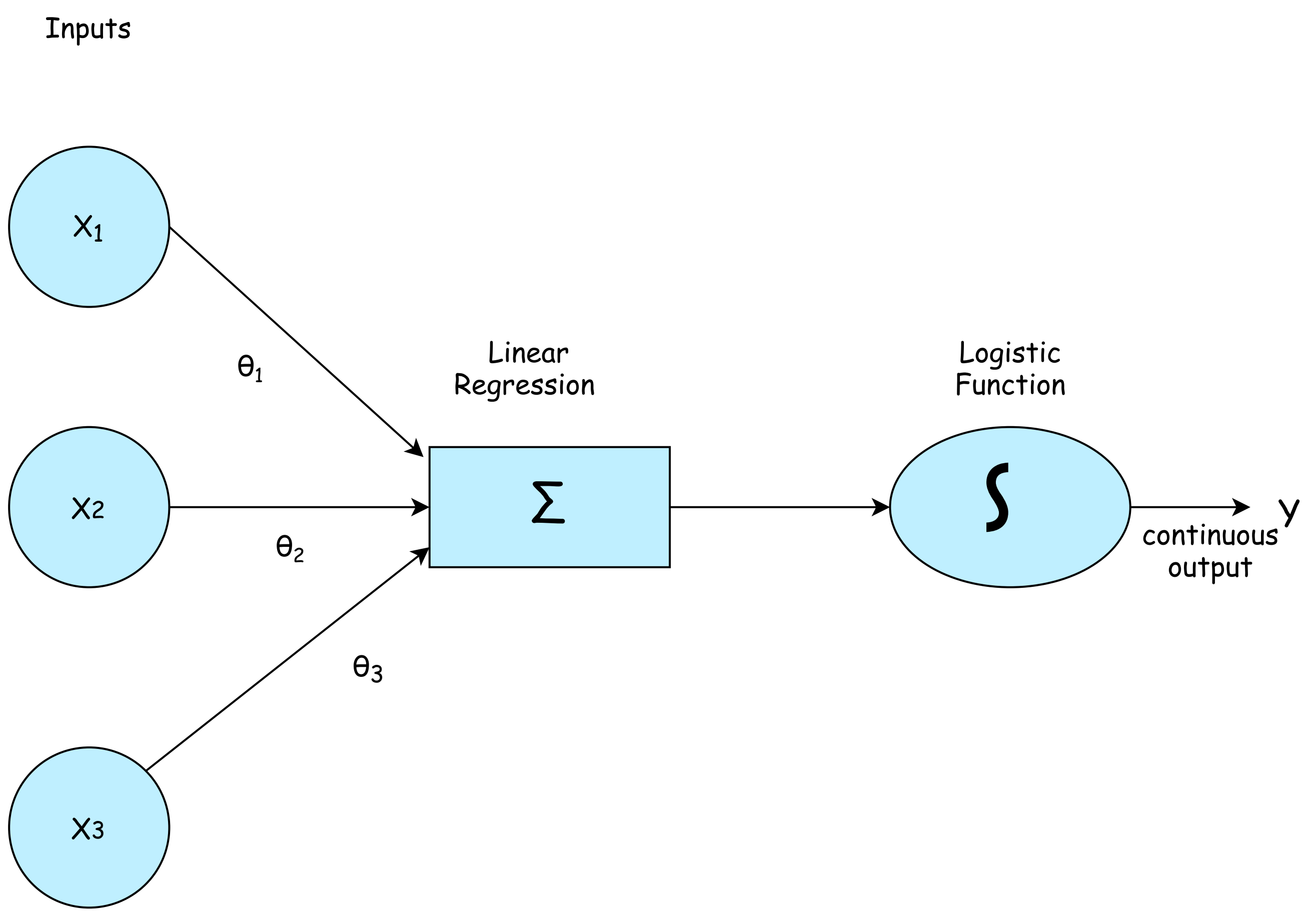


The logistic function has a fixed range. It will always output numbers in the range of 0 to 1. This means that we will always get an output in the range of 0 to 1. Therefore, our prediction function $h(x)$ becomes:

$$h(x) = g(f(x))$$

where `g` is the sigmoid function and $f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$

The illustration below explains this concept.



In logistic regression, this output is interpreted as the probability of the observation belonging to the second class. In binary classification, if the result is greater than 0.5, we say that the observation belongs to the second class, and if it is less than 0.5, it belongs to the first class.

Cost function

The cost function used instead of mean squared error is the **cross-entropy** function.

$$J(y) = \frac{1}{m}\sum_{i=1}^{m}[\ -y_i\ log(h(x_i)) - (1-y_i)log(h(x_i))\ ]$$

where y_i denotes labels of our classes. It can be 1 or 0 for binary classification. The expression inside the square brackets is the loss for one observation. The error is summed for all observations.

This function will be minimized using gradient descent, as we did earlier for linear regression. However, we will not go further into the math of how gradient descent would optimize this function at this point and move straight to using it in Python.

to using it in Python.

Logistic Regression in Python

Now it is time for us to perform logistic regression in Python. Fortunately, `sklearn` has us covered. We will use the `LogisticRegression` class available in `sklearn.linear_model`.

To evaluate the performance, we will be using the function `accuracy_score` from `sklearn.metrics`, which tells us the percentage of accurate results.

We will be predicting whether a credit card client defaults or not by using the Credit Card Clients Default Dataset. The binary prediction variable is `default.payment.next.month`.

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df = pd.read_csv('credit_card_cleaned.csv')
# Make data
X = df.drop(columns = ['default.payment.next.month','MARRIAGE','GENDER'])
Y = df[['default.payment.next.month']]
# Fit model
lr = LogisticRegression()
lr.fit(X,Y)
# Print parameters
print(lr.coef_)
print(lr.intercept_)
# Get predictions and accuracy
preds = lr.predict(X)
acc = accuracy_score(y_true = Y,y_pred = preds)

print('accuracy = ',acc)
```

We load the data in **line 5**. Then we separate our training data and the predictions in **lines 7 and 8**. We initialize the class in **line 10**. We use the `fit` function to fit the model and obtain the best parameters in **line 11**. Then we print the model parameters and the intercept parameter in **lines 13 and 14**. Afterward, we obtain predictions using the `predict` function. We calculate the accuracy in **line 17**. `accuracy_score` expects the actual values and the predicted values. Then we print the accuracy in the last line.

From the output, we can see that our model gives the correct prediction 77%

From the output, we can see that our model gives the correct prediction 77% of the time.

In the next lesson, we will look at how we can evaluate logistic regression models.