



National University of Computer & Emerging Sciences,
Karachi



FALL 2024, LAB MANUAL – 09

LOGISTIC REGRESSION AND NEURAL NETWORK

COURSE CODE :	AL3002
INSTRUCTOR :	Usama Bin Umar

OBJECTIVE

1. Implementation of Logistic Regression
2. Parameters of Logistic Regression
3. Gradient Descent and Cost Function

LOGISTIC REGRESSION

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

TYPES OF LOGISTIC REGRESSION

- Ordinal Logistic Regression
- Binary Logistic Regression
- Multinomial Logistic Regression

Binary logistic regression

Binary logistic regression is used to predict the probability of a binary outcome, such as yes or no, true or false, or 0 or 1. For example, it could be used to predict whether a customer will churn or not, whether a patient has a disease or not, or whether a loan will be repaid or not.

Multinomial logistic regression

Multinomial logistic regression is used to predict the probability of one of three or more possible outcomes, such as the type of product a customer will buy, the rating a customer will give a product, or the political party a person will vote for.

Ordinal logistic regression

Ordinal Logistic Regression is used to predict the probability of an outcome that falls into a predetermined order, such as the level of customer satisfaction, the severity of a disease, or the stage of cancer.

ASSUMPTION OF LOGISTIC REGRESSION

Logistic regression is a statistical method used for modeling the relationship between a binary dependent variable (0/1, Yes/No, True/False) and one or more independent variables. Like other statistical models, logistic regression makes certain assumptions about the data and the underlying relationships. These assumptions include:

Binary Dependent Variable: Logistic regression assumes that the dependent variable is binary or dichotomous. In other words, it has only two possible outcomes.

Linearity: Logistic regression assumes a linear relationship between the log-odds of the dependent variable and the independent variables. This means that the log-odds change linearly with changes in the independent variables.

Independence of Observations: Logistic regression assumes that the observations are independent of each other. This means that the value of the dependent variable for one observation should not be influenced by the values of the independent variables for other observations.

No Multicollinearity: There should be no multicollinearity among the independent variables. Multicollinearity occurs when two or more independent variables are highly correlated, which can make it difficult to estimate the individual effects of each variable.

Large Sample Size: A reasonably large sample size is often assumed for logistic regression to ensure stable parameter estimates. As a rule of thumb, having at least 10-20 cases with the least frequent outcome per independent variable is recommended.

No Outliers: Logistic regression is sensitive to outliers, so the data should be examined for extreme values that might unduly influence the results.

Linearity of Log-Odds: The relationship between the independent variables and the log-odds of the dependent variable should be roughly linear. If the relationship is not linear, you may need to transform the independent variables or consider other modeling techniques.

Absence of Endogeneity: Logistic regression assumes that the independent variables are not influenced by the outcome variable (endogeneity). If there is evidence of endogeneity, it can lead to biased parameter estimates.

It's important to note that these assumptions are idealized conditions, and real-world data often deviates from them to some extent. Violations of these assumptions can affect the validity of the logistic regression model and may require further data preprocessing or alternative modeling approaches. Researchers typically assess the assumptions through diagnostic tests and exploratory

data analysis to make informed decisions about model suitability and potential issues.

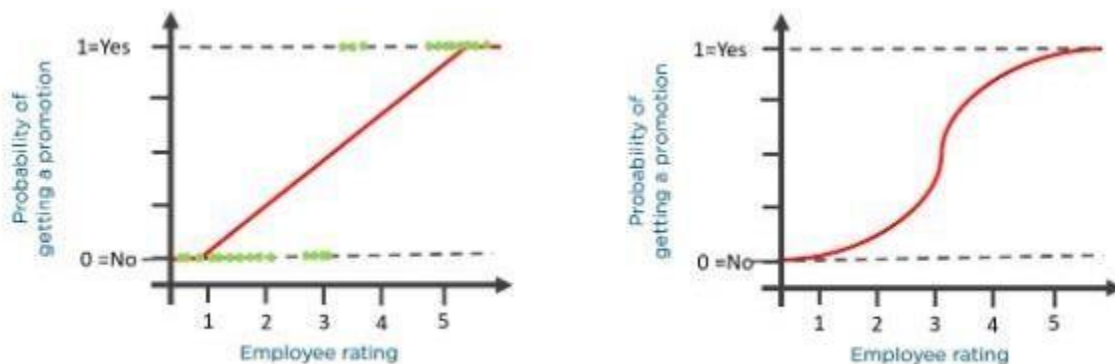
HOW DOES THE LOGISTIC REGRESSION ALGORITHM WORK?

Consider the following example: An organization wants to determine an employee's salary increase based on their performance.

For this purpose, a linear regression algorithm will help them decide. Plotting a regression line by considering the employee's performance as the independent variable, and the salary increase as the dependent variable will make their task easier.



Now, what if the organization wants to know whether an employee would get a promotion or not based on their performance? The above linear graph won't be suitable in this case. As such, we clip the line at zero and one, and convert it into a sigmoid curve (S curve).



Based on the threshold values, the organization can decide whether an employee will get a salary increase or not.

To understand logistic regression, let's go over the odds of success.

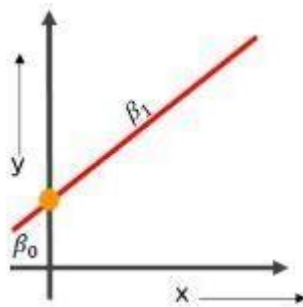
Odds (θ) = Probability of an event happening / Probability of an event not happening

$$\theta = p / 1 - p$$

The values of odds range from zero to ∞ and the values of probability lies between zero and one.

Consider the equation of a straight line:

$$y = \beta_0 + \beta_1 x$$



Here, β_0 is the y-intercept

β_1 is the slope of the line

x is the value of the x coordinate

y is the value of the prediction

Now to predict the odds of success, we use the following formula:

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 x$$

Exponentiation both the sides, we have:

$$e^{\ln\left(\frac{p(x)}{1-p(x)}\right)} = e^{\beta_0 + \beta_1 x}$$

$$\left(\frac{p(x)}{1-p(x)}\right) = e^{\beta_0 + \beta_1 x}$$

Let $Y = e^{\beta_0 + \beta_1 x}$

Then $p(x) / 1 - p(x) = Y$

$$p(x) = Y(1 - p(x))$$

$$p(x) = Y - Y(p(x))$$

$$p(x) + Y(p(x)) = Y$$

$$p(x)(1+Y) = Y$$

$$p(x) = Y / 1+Y$$

$$p(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

The equation of the sigmoid function is:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

The sigmoid curve obtained from the above equation is as follows:



SIGMOID FUNCTION:

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value

below the threshold values tends to 0.

We apply the sigmoid activation function on the hypothetical function of linear regression. So the resultant hypothetical function for logistic regression is given below:

$$h(x) = \text{sigmoid}(wx + b)$$

Here, w is the weight vector.

x is the feature vector.

b is the bias.

$$\text{sigmoid}(z) = 1 / (1 + e^{(-z)})$$

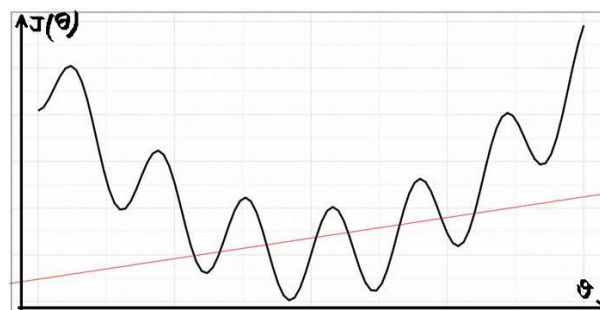
COST FUNCTION

The cost function represents optimization objective i.e. we create a cost function and minimize it so that we can develop an accurate model with minimum error.

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2.$$

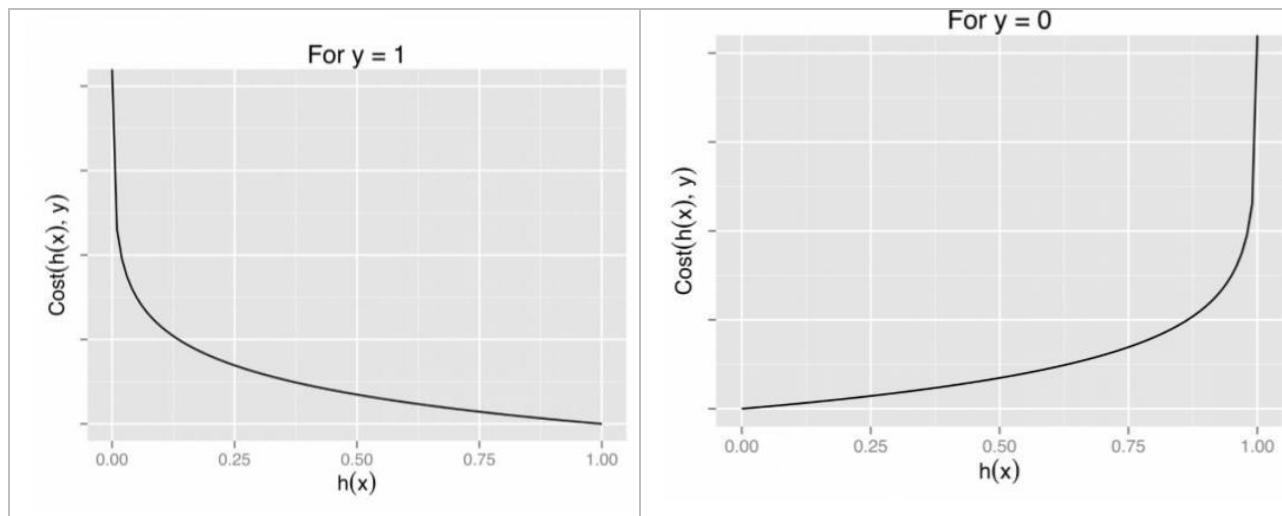
Figure 1 Cost Function of Linear regression

If we try to use the cost function of the linear regression in 'Logistic Regression' then it would be of no use as it would end up being a **non-convex** function with many local minimums, in which it would be very **difficult** to **minimize the cost value** and find the global minimum.



For logistic regression, the Cost function is defined as:

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



The above two functions can be compressed into a single function i.e.

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

GRADIENT DESCENT

Now the question arises, how do we reduce the cost value. Well, this can be done by using **Gradient Descent**. The main goal of Gradient descent is to **minimize the cost value**. i.e. $\min J(\theta)$.

Now to minimize our cost function we need to run the gradient descent function on each parameter i.e.

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

Objective: To minimize the cost function we have to run the gradient descent function on each parameter

Want $\min_{\theta} J(\theta)$:

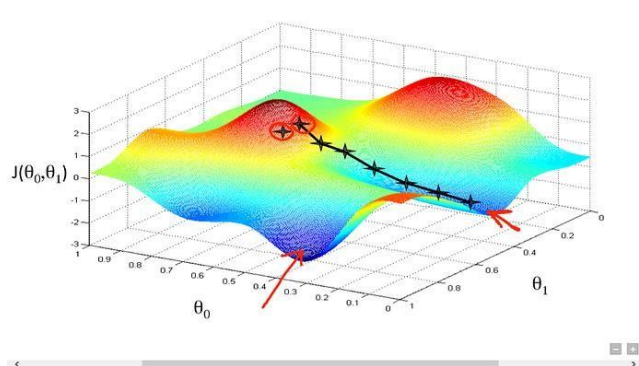
Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

}

Gradient descent has an analogy in which we have to imagine ourselves at the top of a mountain valley and left stranded and blindfolded, our objective is to reach the bottom of the hill. Feeling the slope of the terrain around you is what everyone would do. Well, this action is analogous to calculating the gradient descent, and taking a step is analogous to one iteration of the update to the parameters.



Logistic Regression

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
modelLR = LR.fit(x_train,y_train)
prediction= modelLR.predict(x_test)

from sklearn.metrics import accuracy_score
print("====Training Accuarcy====")
trac=LR.score(x_train,y_train)
print(trac)
print("====Testing Accuarcy====")
teacLR=accuracy_score(y_test,prediction)
print(teacLR)
```


Exploring Slope and Coefficients

```
print(modelLR.coef_)  
print(modelLR.intercept_)
```

Multinomial logistic regression, also known as softmax regression or maximum entropy classifier, is an extension of binary logistic regression to handle problems with more than two categories. It is a classification algorithm used when the dependent variable is categorical with more than two levels. In multinomial logistic regression, the model predicts the probability of each category and assigns the observation to the category with the highest probability. The output is a probability distribution over multiple classes, and the probabilities sum to 1

Multinomial Logistic Regression

```
MLR = LogisticRegression(multi_class='multinomial')  
modelMLR = MLR.fit(x_train,y_train)  
prediction= modelMLR.predict(x_test)
```

In scikit-learn's Logistic Regression, the **multi_class='auto'** parameter is used as default to automatically determine the multi-class strategy based on the nature of the problem and the solver selected. Specifically, it selects the appropriate strategy based on the number of unique classes in the target variable. The behavior of multi_class='auto' can be understood as follows:

Binary Classification (Two Classes):

If there are only two unique classes in the target variable, logistic regression defaults to the binary classification strategy ('ovr' or 'ovr_balanced' depending on the value of the class_weight parameter).

Multiclass Classification (More than Two Classes):

If there are more than two unique classes, logistic regression automatically switches to the "multinomial" strategy. In this strategy, the model is trained to handle multiple classes simultaneously, and it optimizes a single objective function.

Solver in Logistic Regression

```
model1 = LogisticRegression(solver='newton-cg')  
model1.fit(x_train,y_train)
```

The solver parameter also plays a role in determining the behavior. If solver='liblinear', it is designed

for binary classification and uses the 'ovr' strategy for multiclass problems. If solver is set to other algorithms like 'lbfgs', 'newton-cg', or 'sag', logistic regression

The choice of the solver depends on various factors, including the size of the dataset, the number of features, and the regularization requirements. Here are some considerations:

- For small to medium-sized datasets, 'liblinear' is often a good choice.
- For high-dimensional datasets, 'lbfgs' or 'newton-cg' might be more efficient.
- For large datasets, especially when the number of samples is much larger than the number of features, 'sag' or 'saga' could be beneficial.

Penalty in Logistic Regression

```
from sklearn.linear_model import LogisticRegression

#L1 regularization
model = LogisticRegression(penalty='l1', solver='liblinear')

# L2 regularization
model = LogisticRegression(penalty='l2', solver='liblinear')

# Elastic Net regularization
model = LogisticRegression(penalty='elasticnet', solver='saga', l1_ratio=0.5)
```

In scikit-learn's Logistic Regression model, the penalty parameter is used to specify the type of regularization applied to the model. Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function. The penalty parameter has two main options: 'l1' and 'l2', each corresponding to a different type of regularization.

L1 Regularization ('l1'):

- Also known as Lasso regularization.
- Adds the absolute values of the coefficients as a penalty term to the loss function.
- Encourages sparsity in the feature coefficients, leading to some coefficients being exactly zero.
- Useful for feature selection if you suspect that many features are irrelevant.

L2 Regularization ('l2'):

- Also known as Ridge regularization.
- Adds the squared values of the coefficients as a penalty term to the loss function.

- Encourages small, but non-zero, coefficients for all features.
- Generally used when all features are expected to contribute to the prediction.

Elastic Net Regularization ('elasticnet'):

- A combination of L1 and L2 regularization.
- The l1_ratio parameter controls the balance between L1 and L2 regularization.
- Useful when there are many features, and some of them are expected to be irrelevant.

By default, the penalty parameter is set to 'l2'. The choice between 'l1' and 'l2' (or 'elasticnet') depends on the characteristics of your data and the problem at hand. Experimenting with different penalties and regularization strengths can help determine the optimal configuration for your logistic regression model.

[Click here](#) to explore more Logistic Regression Parameters

PERCEPTRON

A perceptron is a fundamental building block of artificial neural networks and serves as a basic computational unit. It was developed by Frank Rosenblatt in 1957. The perceptron is a type of linear classifier that takes a set of binary inputs and produces a binary output. It's a simple mathematical model inspired by the way biological neurons work.

Here are the key components and concepts related to perceptrons:

Inputs and Weights:

The perceptron receives multiple binary inputs (0 or 1).

Each input is associated with a weight, which determines the influence of that input on the perceptron's output.

Weighted Sum:

The weighted sum of inputs and weights is calculated.

The perceptron performs a weighted sum: $\text{sum} = \sum_{i=1}^n \text{input}_i \times \text{weight}_i$

Activation Function:

The weighted sum is passed through an activation function.

The result is typically compared to a threshold.

If the result exceeds the threshold, the perceptron outputs 1; otherwise, it outputs 0.

Thresholding:

The activation function acts as a thresholding function.

A common choice is a step function: output

$$\begin{cases} 1 & \text{if } \text{sum} \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}.$$

Mathematically, the output (output) of a perceptron can be represented as:

$$\text{output} = \begin{cases} 1 & \text{if } \sum_{i=1}^n \text{input}_i \times \text{weight}_i \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Perceptron have limitations, particularly their inability to learn more complex patterns and relationships. However, they form the basis for more advanced neural network architectures. Multilayer perceptron (MLPs) and other neural network structures were later developed to address the limitations of single-layer perceptron, allowing for the learning of more complex functions.

NEURAL NETWORK

A neural network is a computational model inspired by the way biological neural networks in the human brain work. It consists of interconnected nodes, often referred to as neurons or artificial neurons, organized into layers. Neural networks are a fundamental component of machine learning, particularly in the field of deep learning.

COMPONENTS OF NEURAL NETWORK

Here are the key components and concepts related to neural networks:

Neurons (Nodes):

Neurons are the basic computational units in a neural network.

Each neuron receives one or more inputs, performs a computation, and produces an output.

Layers:

Neurons are organized into layers. The three main types of layers are:

Input Layer: Receives the initial input data.

Hidden Layers: Intermediate layers between the input and output layers. Deep neural networks have multiple hidden layers.

Output Layer: Produces the final output or prediction.

Weights and Connections:

Connections between neurons are associated with weights.

Weights determine the strength of the connection between neurons.

Learning in a neural network involves adjusting these weights based on training data.

Activation Function:

Each neuron typically applies an activation function to its weighted sum of inputs.

Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU).

Feedforward and Backpropagation:

In a feedforward neural network, information flows from the input layer through the hidden layers to the output layer.

Backpropagation is a training algorithm used to adjust weights based on the error between predicted and actual outputs.

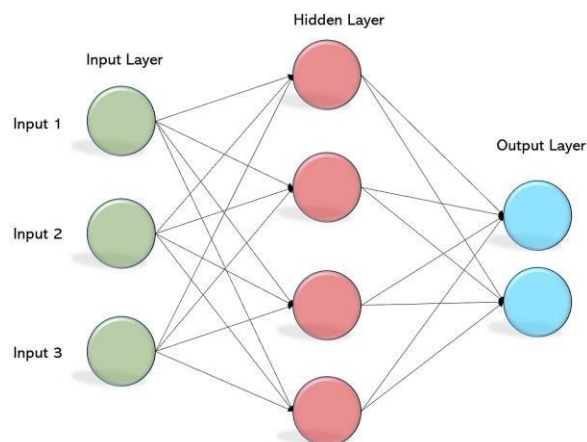
Training and Learning:

Neural networks learn from data by adjusting their weights during the training process.

Training involves presenting input data with known outputs and updating weights to minimize prediction errors.

MULTI LAYER PERCEPTRON

MLP stands for "Multilayer Perceptron," and it refers to a type of neural network architecture. An MLP is a class of feedforward artificial neural networks characterized by having multiple layers of nodes (neurons), including an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next layer.

**Input Layer with One Hidden Layer**

```
# Add an input layer with 20 neurons (input features)
model.add(Dense(units=20, input_dim=20, activation='relu'))
```

2nd Hidden Layer

```
# Add a hidden layer with 10 neurons
model.add(Dense(units=10, activation='relu'))
```

Output Layer

```
# Add an output layer with 1 neuron (binary classification)
model.add(Dense(units=1, activation='sigmoid'))
```

Compile MLP

```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Model Training

```
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
```

MLP For Binary Classification

```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score

# Create an MLP model
model = Sequential()
# Add an input layer with 20 neurons (input features)
model.add(Dense(units=20, input_dim=20, activation='relu'))
# Add a hidden layer with 10 neurons
model.add(Dense(units=10, activation='relu'))
# Add an output layer with 1 neuron (binary classification)
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
# Make predictions on the test set
y_pred = model.predict(x_test)
y_pred_binary = np.round(y_pred)
# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"Accuracy: {accuracy}")
```

For Training a binary classification model , the input dim should be equal to the number of features, activation function in input and hidden can be chosen as Relu or any other. But in Output Layer we have to choose Sigmoid for Binary Classification. The loss parameter in compile should be Binary Cross entropy for Binary Classification. The number of nodes in Output Layer should be One for Binary Classification.

MLP For Multi Classification

```
# Create an MLP model
model = Sequential()
# Add an input layer with 20 neurons (input features)
model.add(Dense(units=20, input_dim=4, activation='relu'))
# Add a hidden layer with 10 neurons
model.add(Dense(units=10, activation='relu'))
# Add an output layer with total unique attribute in target variable (Multiclassification)
model.add(Dense(units=3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
# Make predictions on the test set
y_pred = model.predict(x_test)
y_pred_binary = np.round(y_pred)
# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"Accuracy: {accuracy}")
```


For Training a Multi classification model , the input dim should be equal to the number of features, activation function in input and hidden can be choosen as Relu or any other. But in Output Layer we have to choose Softmax for Multi Classification. The loss parameter in compile should be Categorical Cross entropy for Multi Classification. The number of nodes in Output layer should be equal to the unique records found in Class/Target Variable.

MLP For Regression

```
# Create an MLP model
model = Sequential()
# Add an input layer with 20 neurons (input features)
model.add(Dense(units=20, input_dim=4, activation='relu'))
# Add a hidden layer with 10 neurons
model.add(Dense(units=10, activation='relu'))
# Add an output layer with 1 neuron (regression)
model.add(Dense(units=1, activation='linear'))
# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
# Make predictions on the test set
y_pred = model.predict(x_test)
y_pred_binary = np.round(y_pred)
# Evaluate the mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

For Training a Regression model , the input dim should be equal to the number of features, activation function in input and hidden can be choosen as Relu or any other. But in Output Layer we have to choose Linear for Regression. The loss parameter in compile should be Mean Square Error for Regression. The number of nodes in Output layer should be equal to 1.

TASKS

TASK 1:

Load the heart disease dataset

- Perform necessary EDA and Data Wrangling and Implement Logistic Regression
- First train your model using penalty as l1 regularization
- Train your model with l2 regularization
- Train your model with penalty = elastic net
- Now compare the results of training and testing accuracy
- Discuss in a text cell, what error you have gone through while implementing this penalty, Have you changed other parameter to apply these Penalties??
- What additional parameter you have changed while implementing the mentioned penalty, what is the relationship between these?

TASK 2:

Implement Logistic regression on the **Iris dataset** , now choose different solvers ***lbfgs***, ***liblinear***, ***newton-cg***, ***newton-cholesky***, ***sag***, ***saga***

Compare the results of Training and testing accuracy in a Dataframe like this

Solver	Training Accuracy	Testing Accuracy

- Briefly discuss the effect of solver on your dataset ,
- Have you found the similarity as mentioned by Sklearn that which solver is best for small, medium or larger dataset.
- Which solver is best in your case and why?
- Now copy this file and apply a new Dataset (Heart Disease) and compare, Does it really affected by the size of dataset??

TASK 3:

- Implement Perceptron on Iris Data and the Compare the Results of Logistic Regression (Sklearn) model with Perceptron model. Compare Training and Testing results of Perceptron and LR
- What is the difference between Perceptron and LR

TASK 4

You are tasked with developing a more sophisticated MLP model for fraud detection in financial transactions. The dataset contains information about various transactions, and the goal is to classify each transaction as either fraudulent (1) or not fraudulent (0).

Your tasks are as follows:

- Load the dataset.
- Explore the dataset to understand its structure and characteristics.
- Preprocess the data if necessary.
- Split the dataset into training and testing sets.
- Create an MLP model for classification with the following Hidden Layer architecture:
 - Hidden Layer 1: 128 neurons, ReLU activation function.
 - Hidden Layer 2: 64 neurons, Tanh activation function.
 - Hidden Layer 3: 32 neurons, ReLU activation function.
 - Decide input and output by yourself
- Compile the model with the appropriate loss function and the Adam optimizer.
- Train the model on the training data for 50 epochs.
- Evaluate the model on the testing data.
- Additionally, monitor key metrics like accuracy, precision, recall, or F1 score during training and testing phases.

TASK 5

Implement MLP on these Dataset

[Dataset 1](#)

[Dataset 2](#)