Volume 10, Issue 9, September– 2025 International Journal of Innovative Science and Research Technology
ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

Mathematical Foundations of Regression Analysis: A Study of Linear and Logistic Models

Mritunjay Mukherjee[1]

[1]Delhi Private School Dubai

Publication Date: 2025/10/11

Abstract: Regression models form the backbone of modern statistical inference and predictive analytics. This paper presents a rigorous mathematical examination of two fundamental approaches: linear regression and logistic regression. Beginning with the formulation of each model, we derive their objective functions—the least squares criterion for linear regression and the log-likelihood for logistic regression. Closed-form solutions for linear regression are contrasted with the iterative optimization required in logistic regression, highlighting the importance of gradient-based methods. Special emphasis is placed on demonstrating how these mathematical principles can be applied to real-life datasets.

How to Cite: Mritunjay Mukherjee (2025). Mathematical Foundations of Regression Analysis: A Study of Linear and Logistic Models. *International Journal of Innovative Science and Research Technology*, 10(9), 2891-2905. https://doi.org/10.38124/ijisrt/25sep997

I. INTRODUCTION

Mathematics has always played a central role in the development of statistical methods, and regression analysis stands as one of its most influential contributions. Regression models provide a systematic framework to describe, interpret, and predict relationships between variables. In the modern era of data-driven decision making, these models form the backbone of fields ranging from economics and social sciences to biology, engineering, and artificial intelligence. By allowing researchers and practitioners to quantify associations between independent and dependent variables, regression not only aids in understanding existing phenomena but also in forecasting future outcomes with measurable accuracy.

Two of the most widely used approaches in regression analysis are linear regression and logistic regression. Linear regression focuses on modeling relationships where the dependent variable is continuous, relying on the principle of minimizing error through the least squares method. In contrast, logistic regression addresses classification problems, where the outcome is categorical, often binary. Both models, while conceptually distinct, share deep mathematical foundations that rely on optimization, calculus, and probability theory. Their underlying formulations—closed-form solutions in the case of linear regression—highlight the essential role of mathematical rigor in ensuring reliability and interpretability.

The significance of regression extends far beyond theoretical mathematics. In practical applications, regression models allow us to analyze real-world datasets, detect patterns, and guide policy or strategy. For instance, linear regression can be used to project refugee population growth using historical data, while logistic regression can support healthcare decision-making by classifying medical outcomes such as tumor malignancy.

This paper aims to present a comprehensive examination of the mathematical structures underlying linear and logistic regression. By deriving their objective functions, discussing optimization strategies, and demonstrating their utility through case studies, we emphasize both their theoretical depth and their practical relevance.

II. LINEAR REGRESSION

Linear Regression is a simple yet powerful statistical and mathematical concept used to model relationships between independent and dependent variables. The essence of Linear Regression lies in establishing a linear relationship between these variables by fitting a straight line to the data. This relationship can then be used to predict future values of dependent variable based on new values of the independent variable. The mathematical equation behind a Linear Regression model is:

$$\hat{y} = mx_i + b$$

Where:

$\hat{y}$ is the dependent variable

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

x_i is the independent variable

is the slope (or gradient) of the line is y-intercept of the line

The above equation has 2 parameters. The goal of a Linear Regression model is to find those parameters that minimizes the difference between the actual value and the predicted value of the dependent variable, a term called residual ($e = y_i - \hat{y}$, where $\hat{y}$ is the predicted value).

➢ *Least Squares Method*

Linear Regression models try to find the line of best fit such that there is least error. This line of best fit depends upon two parameters, and. To find these parameters, Linear Regression models can use the Least Squares Method, a closed-form solution method. The idea behind this method is to determine the value of slope and y-intercept such that the sum of the squares of the residuals is minimized. First, we define a function $S(m, b)$ to depict this sum, taking parameters as slope and y-intercept:

$$S(m, b) = \sum_{i=1}^{n} (y_i - \hat{y})^2$$

$$S(m, b) = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

Then, the minimum of the function can be found by using calculus and the concept of derivatives. Applying this concept, the minimum of the function $S(m, b)$ is found by solving the following system of equations:

$$\frac{\partial S}{\partial m} = 0 \qquad \frac{\partial S}{\partial b} = 0$$

By solving this system of equations, we can estimate the values of and for a simple Linear Regression, as shown in the next section.

We can think of minimizing the sum of the squares of residual as ensuring the distance of each data point from the regression line we are fitting is smallest. The squaring also helps in emphasizing larger errors. This makes sure that the model does not focus on correcting tiny details (a process called overfitting) and instead focuses on addressing major errors.

➢ *Estimating Parameters of Linear Regression*

$$\frac{\partial S}{\partial b} = \frac{\partial(\sum_{i=1}^{n} (y_i - mx_i - b)^2)}{\partial b}$$

$$= \sum_{i=1}^{n} \frac{\partial (y_i - mx_i - b)^2}{\partial b}$$

$$= \sum_{i=1}^{n} 2(y_i - mx_i - b) \times (-1)$$

$$= -2(\sum_{i=1}^{n} y_i - m\sum_{i=1}^{n} x_i - bn)$$

$$\text{Since } \frac{\partial S}{\partial b} = 0,$$

$$-2(\sum_{i=1}^{n} y_i - m\sum_{i=1}^{n} x_i - bn) = 0$$

$$\frac{\sum_{i=1}^{n} y_i}{n} - m\frac{\sum_{i=1}^{n} x_i}{n} = b$$

$$b = \bar{y} - m\bar{x}$$

$$\frac{\partial S}{\partial m} = \frac{\partial(\sum_{i=1}^{n} (y_i - mx_i - b)^2)}{\partial m}$$

$$= \sum_{i=1}^{n} \frac{\partial (y_i - mx_i - b)^2}{\partial m}$$

$$= \sum_{i=1}^{n} (2(y_i - mx_i - b) \times (-x_i))$$

$$= -2(\sum_{i=1}^{n} (y_i - mx_i - (\bar{y} - m\bar{x})) \times (x_i))$$

$$= -2[\sum_{i=1}^{n} x_i(y_i - \bar{y}) - \sum_{i=1}^{n} x_i \times m(x_i - \bar{x})]$$

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

$$\text{Since } \frac{\partial S}{\partial m} = 0,$$

$$-2[\sum_{i=1}^{n} x_i(y_i - \bar{y}) - \sum_{i=1}^{n} x_i \times m(x_i - \bar{x})] = 0$$

$$\sum_{i=1}^{n} x_i(y_i - \bar{y}) = m \sum_{i=1}^{n} x_i(x_i - \bar{x})$$

$$m = \frac{\sum_{i=1}^{n} x_i(y_i - \bar{y})}{\sum_{i=1}^{n} x_i(x_i - \bar{x})}$$

Also,

$$\bar{x} \sum_{i=1}^{n} y_i - \bar{y} = 0$$

$$\bar{x} \sum_{i=1}^{n} x_i - \bar{x} = 0$$

$$m = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

- *So this Gives us 2 Equations to Find the Values of the Coefficients:*

$$m = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$$b = \bar{y} - m\bar{x}$$

In practice, finding the exact values (closed form solution) of the parameters using Least Squares method gets increasingly more difficult as the datasets become larger. In such cases, an optimization algorithm like Gradient Descent is used.

III. GRADIENT DESCENT OPTIMIZATION

Gradient Descent Optimization Algorithm is an iterative method of finding the minimum of a function. It is used in machine learning models to optimize the model by minimizing the cost function. The algorithm iteratively adjusts the model parameters in the direction of the steepest descent of the cost function. A cost function is a special function which is used to assess how much error the model is making when the output is compared to the true results. For Linear Regression models, the cost function is the mean of the sum of squares of the residual.

$$J(m, b) = \frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{n}$$

We first begin with arbitrary values, m_0 and b_0 in Linear Regression models, for the parameters. Then, we calculate the gradient (rate of change) of the cost function ($J(\theta)$) with respect to the parameters.

The gradient can be represented as the partial derivative of the cost function with respect to the parameters ($\frac{\partial J}{\partial \theta}$). For Linear Regression models:

$$\frac{\partial J}{\partial b} = \frac{\sum_{i=1}^{n} -2(y_i - mx_i - b)}{n}$$

$$\frac{\partial J}{\partial m} = \frac{-2\sum_{i=1}^{n} (y_i - mx_i - b) \times (x_i)}{n}$$

Now, we update the parameter in the opposite direction of the gradient to reduce the cost. For Linear Regression models:

$$m_i = m_0 - \alpha \frac{\partial J}{\partial m}$$

$$b_i = b_0 - \alpha \frac{\partial J}{\partial b}$$

is the learning rate, a constant factor which controls how much is subtracted from the parameters.

m_i and b_i are adjusted iteratively until the minimum value of $J(m, b)$ is achieved.

The process can be understood intuitively as a person standing on a mountain (cost function) and trying to reach the lowest point (global minimum). The slope of the mountain (the gradient) dictates which direction to move (if the gradient is positive, then we must reduce the

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

parameter values and vice versa). The size of each step on the mountain (learning rate) determines how much the person moves in that direction. This ensures that the person gradually descends to the lowest point.

Gradient Descent is a fundamental algorithm for optimization in machine learning. Because it finds the minimum of a function without the need of solving the differential equation $\frac{\partial J}{\partial \theta} = 0$, Gradient Descent can be used on complex and multidimensional functions.

➢ *The Normal Equation*

So far, we have dealt with Linear Regressions in which the dependent variable only depends upon one independent variable, giving us 2 parameters to find. But what happens when the number of variables increases, increasing the number of parameters? For this purpose, The Normal Equation is used to get a solution.

The Normal Equation makes use of the same rule as The Least Squares Method but extends it to multiple variables by using matrix notation and formulas. Without going in depth, the formula is:

$$\theta = (X^T X)^{-1} X^T y$$

Where θ is the matrix of parameters, X is the input matrix and is the observed output values.

➢ *Case Study 1: The Idea*

Having established the mathematical foundation behind Linear Regression models, we can now explore how these concepts can be used in real life situations. Particularly, let us understand how these tools can be used to create a model that can predict refugee numbers in the future based on historical data (prediction model).

To begin with, this ML model will be a simple Linear Regression: future refugee numbers depend only on the year and not on any other factor. Although this is not the case practically, making a simple Linear Regression model would simplify the problem significantly and still use the same tools.

Next, the historical data for the model will be taken from the UNHCR website, ensuring a reliable source. The data contains refugee numbers from the year 2010 through 2022 for every country. The goal is to predict refugee numbers from 2023 through 2032. The ML model will first fit a line of best fit into the data for refugee numbers from 2010 till 2022 using past data and Least Squares method and then use the line to predict future values.

Finally, we need to assess the model's effectiveness. This is done by comparing the model's predicted values for the years 2023 and 2024 with the actual values. We can find out the Root Mean Squared Error which will help us in establishing a range for the prediction where the actual number would likely lie.

With that being said, let us start with the code for the model.

- *Case Study 1: The Code*

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
  import math
  import matplotlib.pyplot as plt
  import numpy as np
  import pandas as pd

  import matplotlib.colors as mcolors
  palette=mcolors.TABLEAU_COLORS
  palettev=list(palette.values())
```

Fig 1 Importing Modules

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

The model will be built with Python 3.10 language on Google Colab. We start off by importing some basic data analytics packages. We also import SciKit-Learn packages which are used for making the Linear Regression model.

After some data processing, we select a specific country's numbers for analyzing (USA for example). All analysis will be done on USA's refugee numbers.

```
# Prepare the data: Filter relevant columns
data = df[['year', 'coa_name', 'refugees']]

# Group by year and country, summing up refugee numbers
data = data.groupby(['year', 'coa_name'], as_index=False).sum()

# Focus on one specific country for prediction (e.g., "United States of
America")
country_name = "United States of America"  # Change as needed
country_data = data[data['coa_name'] == country_name]
```

Fig 2 Data Processing

```
X_train = country_data[['year']].values
y_train = country_data['refugees'].values


# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)


# Predict refugee numbers for future years (2023-2032)
future_years = np.array(range(2023, 2033)).reshape(-1, 1)
future_predictions = model.predict(future_years)
```

Fig 3 Scikit Learn Code

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

We now get our independent (x) variable and dependent (y) variable. Then we create a Linear Regression model using SciKit-Learn and predict future values Finally, we plot this data in a graph using matplotlib library to visualize the historical data and the prediction data.

```
# Visualize the predictions
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(future_years, future_predictions, color='red', label='Predictions
(2023-2032)')
plt.xlabel('Year')
plt.ylabel('Number of Refugees')
plt.title(f'Refugee Prediction for {country_name} (2010-2032)')
plt.legend()
plt.grid()
plt.show()
```

Fig 4 Visualization Code

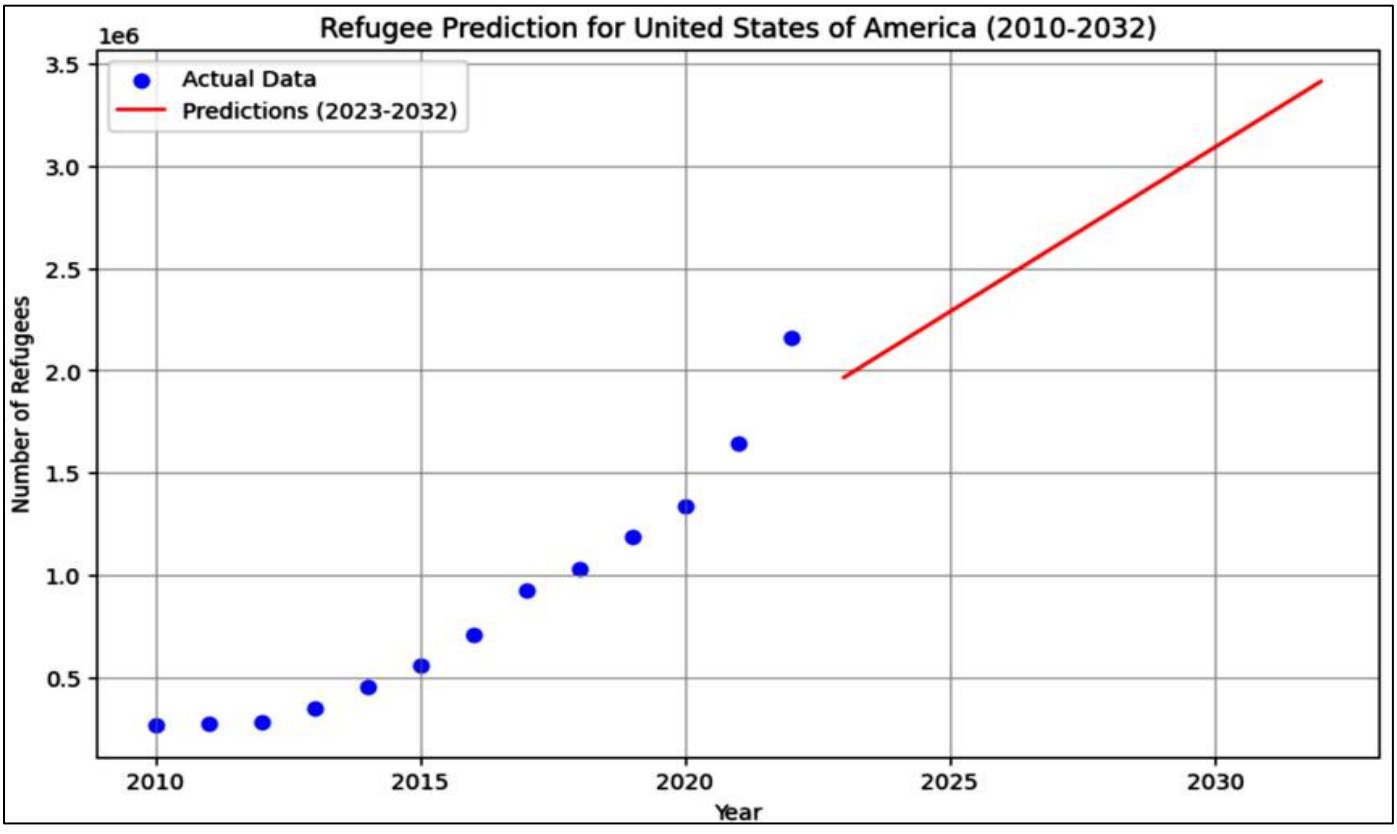

Fig 5 Regression Data

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

➢ *Logistic Regression*

Logistic regression is a statistical method which is used in binary classification models. Unlike Linear Regression, which predicts a range of continuous values, Logistic Regressions predict a probability of a particular data belonging to a particular class, mapping the possibility to outputs of 0 or 1 (binary). Put simply, Logistic Regression is used to classify input as either 0 or 1. For most of this paper, we limit ourselves to only one input variable (independent variable x_i) which is mapped to one output class (binary variable y_i).

Logistic regressions work by first converting the independent variable (x_i) into an adjustable parameter through the following linear relationships:

$$z_i = mx_i + b$$

Where:

: weight of the input

: bias attached

Now, the linear relationship above gives a range of values from $-\infty$ to $+\infty$. However the goal of Logistic Regression is to get a probability as output so that it can be classified as one class or the other. Probability of 0.5 or more assigns the input to one class chosen from before ($y = 1$) and probability of less than 0.5 assigns the input to the other class ($y = 0$). So we need to pass this linear relationship into a function whose domain is $\mathbb{R}$ and range is [0,1]. This function can then be used for calculating probabilities. For this, we have a special function called the sigmoid function represented as:

$$h(z)_i = \frac{1}{1 + e^{-z_i}}$$

We can graph this function in Desmos to find the behavior of this function.

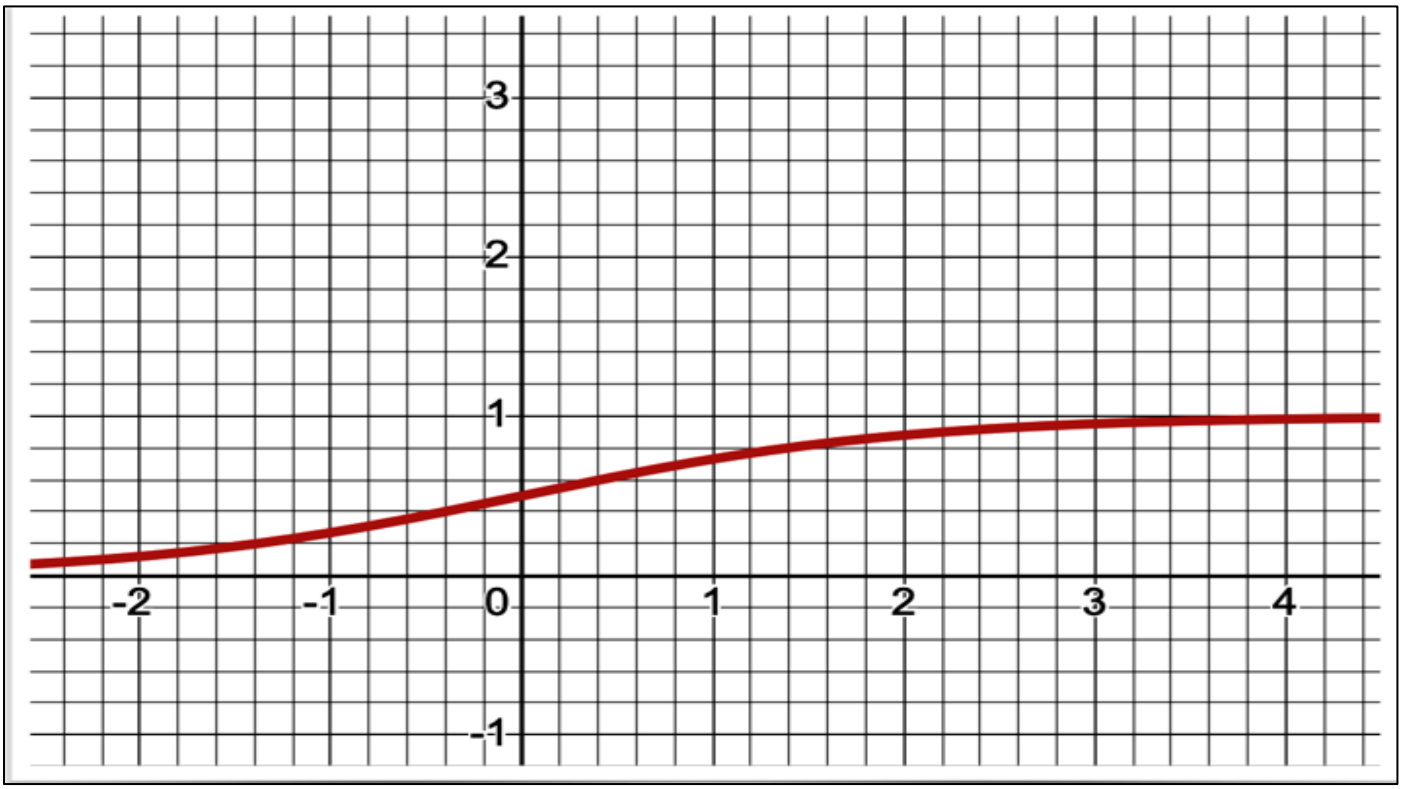

Fig 6 The Sigmoid Function in Desmos

We can see that for any value of input x, the function lies between 0 and 1. Moreover, the function is also monotonic, meaning the value of function increases when input x increases. Finally, its asymptotic behavior also helps in it being an ideal function as the values between 0 and 1 don't repeat.

- *Replacing the Value of z into the Sigmoid Function:*

$$h(x)_i = \frac{1}{1 + e^{-mx_i - b}}$$

This function now gives the probability of an input data belonging to a specific class. The above function needs to now be optimized so that for every x_i, $h(x_i)$ gives

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

the correct probability of it belonging to a class.

IV. GRADIENT DESCENT OPTIMIZATION

We know that the sigmoid function's two parameters, m and b, need to be optimized. We can do this using the same method we used when optimizing Linear Regression- Gradient Descent Optimization.

➢ *Let us Begin with Defining the Cost Function for Logistic Regression:*

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

Where:

: total number of data points

➢ *The Cost Function Can be Broken Down into Two Parts:*

- When $y_i = 1$: The term $\log(h(x_i))$ is used. If the sigmoid function predicts a value close to 1, the log term is less which means a less cost function value. However, if the sigmoid function predicts a value close to 0, the log term will become more negative which means a large cost function value (due to a minus sign outside the summation).
- When $y_i = 0$: The term $\log (1 - h(x_i))$ is used. If the sigmoid function predicts a value close to 0, the log term is less which means a less cost function value. However, if the sigmoid function predicts a value close to 1, the log term will become more negative which means a large cost function value (due to a minus sign outside the summation).

To show this in an example, suppose the value of $h(x_i)$ is 0.9 and y_i is 1. Then, the log term becomes log (0.9), which is approximately -0.105, a relatively small cost. However, if the value of $h(x_i)$ is 0.1 and y_i is 1, then the log term becomes log (0.1), which is approximately -2.302, a much larger cost.

Similarly, if y_i is 0 and $h(x_i)$ is 0.9, then the log term becomes $log\ (1 - 0.9) = log\ (0.1)$ which is a much higher cost than if $h(x_i) = 0.1$, in which case the log term becomes $log\ (1 - 0.1) = log\ (0.9)$.

Now that we understand the cost function, let us understand why logarithmic cost is used. Put simply, the cost function is designed to punish the model more when it is confidently wrong (probability is high but the model is wrong). This makes it so that the model learns faster. Moreover, the logarithmic nature of the cost function makes it differentiable which allows us to use gradient descent.

We can now minimize $J\ (m, b)$ using partial differentiation and updating initial parameters of m and b:

$$m_i = m_0 - \alpha \frac{\partial J}{\partial m}$$

$$b_i = b_0 - \alpha \frac{\partial J}{\partial b}$$

➢ *Decision Boundary*

After optimizing the sigmoid function, the model can then classify based on the following rule:

- $y = 1$ if $h(x) \geq 0.5$
- $y = 0$ if $h(x) < 0.5$

We can then attach y value 1 and 0 to two classes.

However, the above rule also gives the definition of another important concept in Logistic Regression- Decision Boundary.

Decision Boundary is a hyper-surface which acts as the border between the two classes. Mathematically, it is all values for which the sigmoid function in a Logistic Regression model gives value 0.5.

$$h(x) = \frac{1}{1 + e^{-mx-b}} = \frac{1}{2}$$

$$e^{-mx-b} = 1$$

$$x = \frac{-b}{m}$$

So, for a Logistic Regression with 1 independent variable, the Decision Boundary is a point on the x axis numerically equal to the negative of the bias by weight.

➢ *Trigonometric Functions and Logistic Regression*

While discussing the sigmoid function, we understood the purpose behind using the sigmoid function in Logistic Regression models as a means to convert an input of real numbers to an output in the range of 0 to 1. However, could other functions be used in place of the sigmoid function?

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

The first possible replacement to the sigmoid function that comes to mind can be the trigonometric functions of sine and cosine. Both these functions have domain as all real numbers but their range is from -1 to 1. However, if we want to use these functions, their ranges need to be from 0 to 1. This can be rectified by squaring the trigonometric functions. So can $sin^2 x$ or $cos^2 x$ be used in place of sigmoid function?

The answer remains no. Although $sin^2 x$ and $cos^2 x$ are similar to the sigmoid function in many ways, a major concern is that both the functions are periodic and oscillatory in nature. This means for different inputs, the output probability can still be the same. This leads to ambiguity in deciding Decision Boundary.

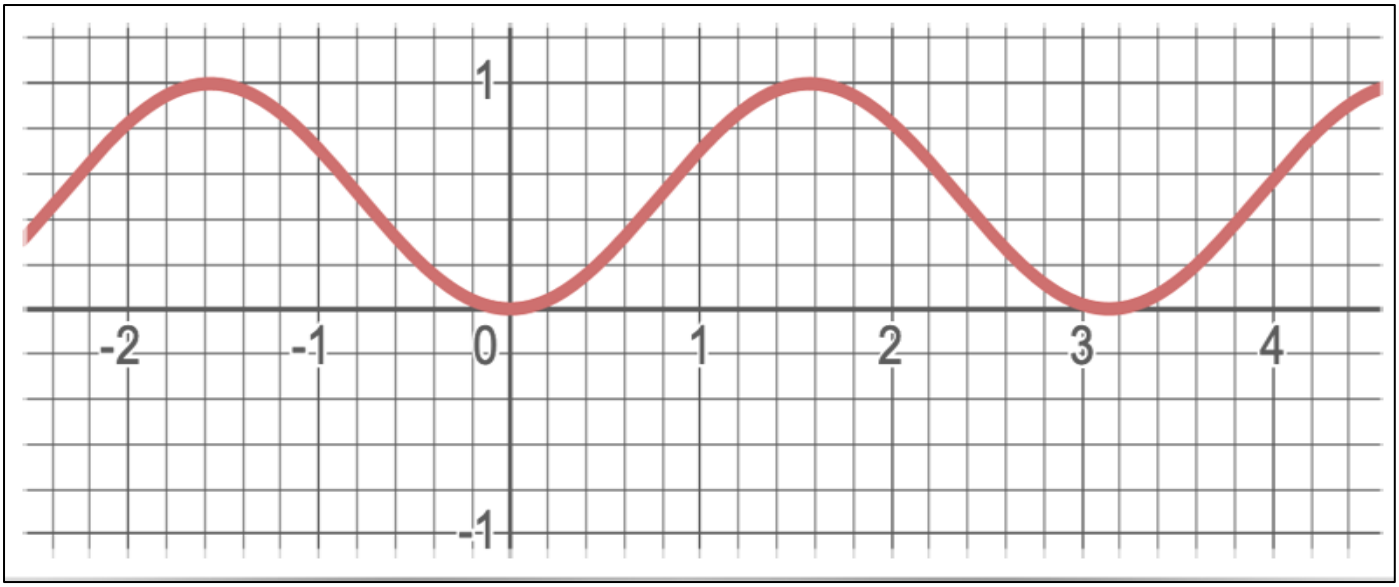

Fig 7 Sine Squared Function in Desmos

Another possible replacement could be hyperbolic trigonometric functions, the hyperbolic analogue for the circular trigonometric function. While the hyperbolic sine and hyperbolic cosine functions cannot be used due to their range being out of the needed limit, hyperbolic tan and hyperbolic sec functions could be considered. However, they both have their own set of problems.

The hyperbolic tan function behaves very similarly to the sigmoid function, being our closest candidate so far, except that its range is from -1 to 1.

Squaring the function wud then remove its asymptotic nature and would give the same problem as squaring the sine or cosine functions: multiple inputs can give the same output.

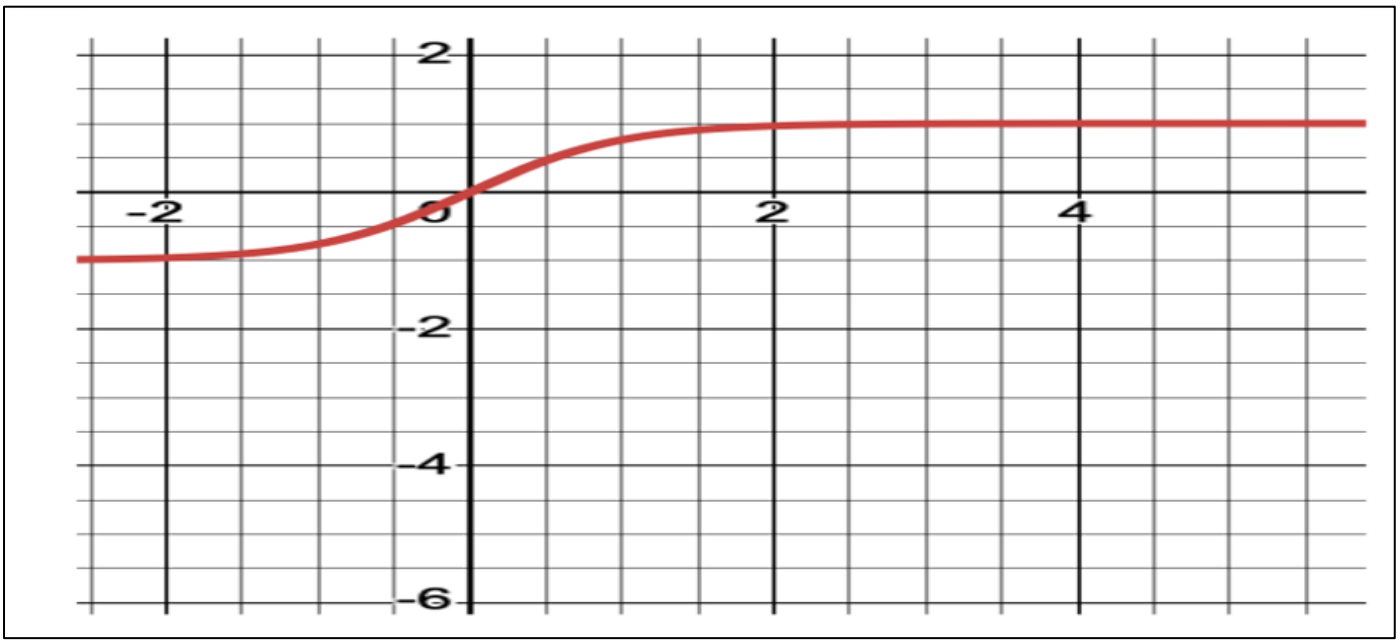

Fig 8 Hyperbolic Tan Function in Desmos

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

The hyperbolic sec function satisfies the range criteria and even has an asymptote at $y = 0$. However, the function lacks an asymptote at $y = 1$, instead having a hill type shape and thereby again leading to multiple inputs for the same output.

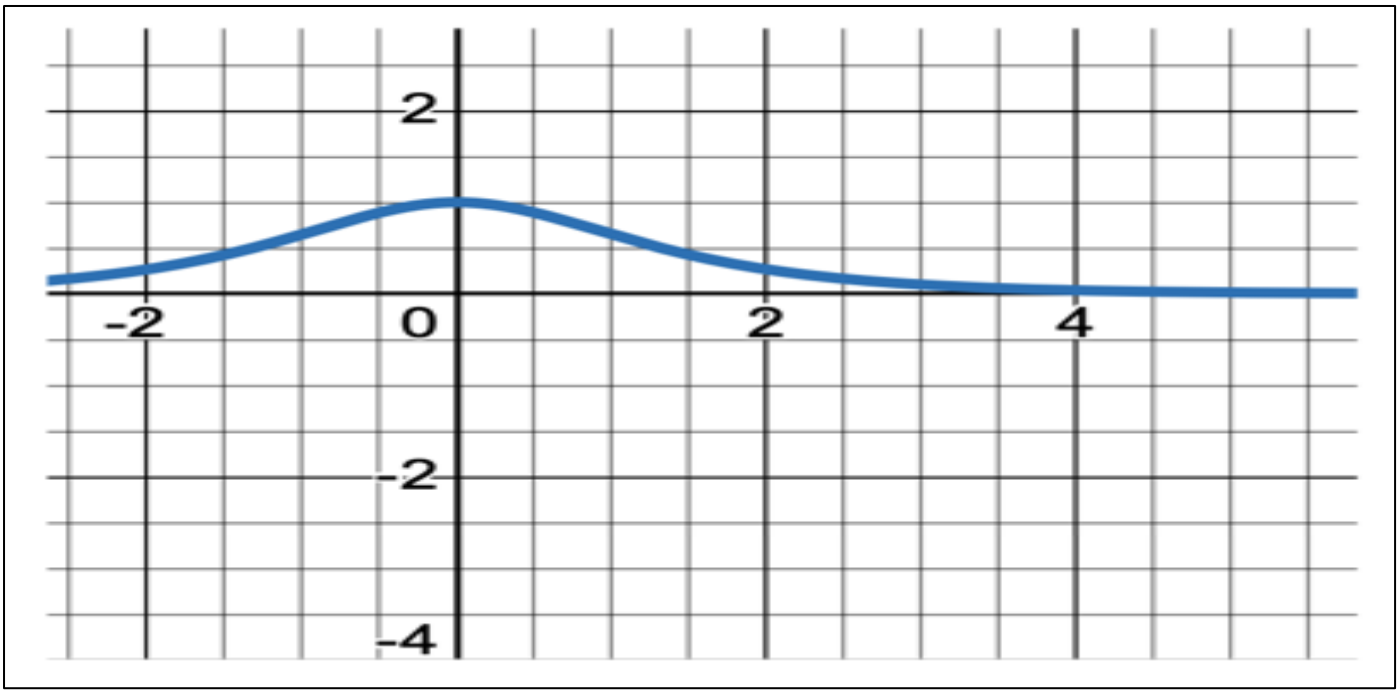

Fig 9 Hyperbolic Sec Function in Desmos

So even hyperbolic trigonometric functions cannot be used in place of the sigmoid function. However, the $tanh(x)$ function can still be used in neural networks as activation functions.

➢ *Bivariate and Multivariate Logistic Regressions*

We know how logistic regressions work for one independent variable. The same concepts can be extended to two or more variables by just modifying the adjustable parameter to be a combination of the bias, the variables and their weights:

$$z = b + \sum_{i=1}^{n} m_i x_i$$

This can now be inputted into the same sigmoid function and the same cost function with the only difference being that multiple partial derivatives need to be found with respect to each parameter.

Differences occur in the Decision Boundary for bivariate and multivariate Logistic Regression. The Decision Boundary for bivariate Logistic Regression is a straight line in the planes of x_1 and x_2.

$$b + m_1x_1 + m_2x_2 = 0$$

Similarly, the Decision Boundary for multivariate Logistic Regression is a hyperplane and is given by the equation:

$$b + m_1x_1 + m_2x_2 + \ldots m_nx_n = 0$$

➢ *Case Study 2: The Idea*

Breast cancer is one of the most common forms of cancer affecting women worldwide. It develops when cells in the breast tissue grow uncontrollably, forming a tumor that can invade surrounding cells. Breast cancer tumors are very complicated at the cellular level, and this makes determining whether a patient's tumor is malignant (dangerous) or benign (not dangerous) a challenge. Now, using Logistic Regression, we can try to create a ML model that can classify a tumor cell as benign or malignant.

Let us first make a Logistic Regression model that can classify cells based on one variable, radius of cell. This decision is based on the graph of the data which looks like after radius of 15 micrometer, the cells become malignant.

So it is clear that radius could be a good metric for the classification. Also, let us assign the number 1 to malignant and the number 0 to benign. The goal is to find the radius (Decision Boundary) after which the cells become more like to be malignant. This Decision Boundary can then be used in classifying new cells.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

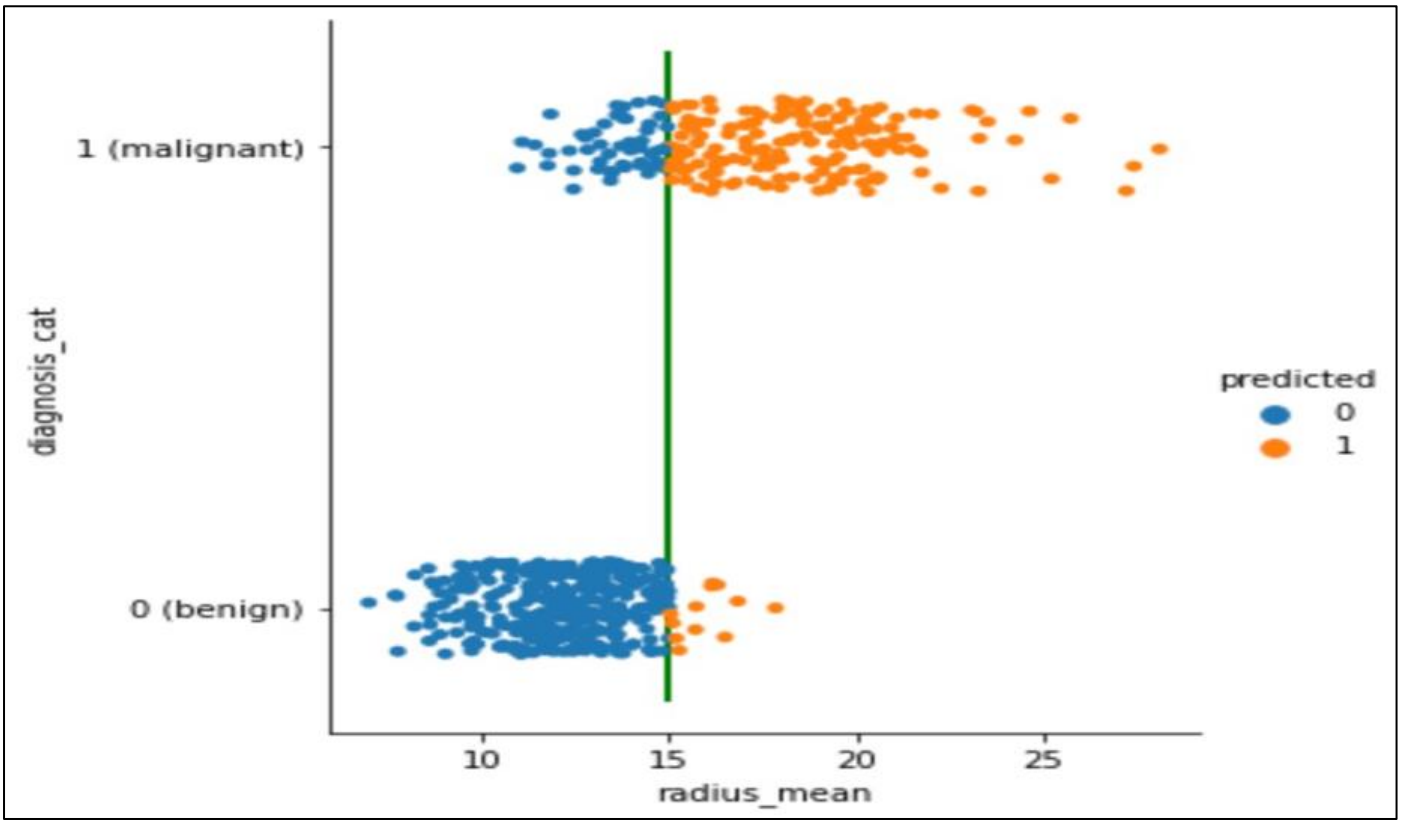

Fig 10 Final Model Predictions

- *Case Study 2: The Code*

We begin with downloading the dataset used to train the model: Breast Cancer Wisconsin (Diagnostic). It consists of 569 biopsy samples, each included with the radius, perimeter, area, smoothness and a few other parameters along with the diagnosis.

```
import gdown
import pandas as pd
from sklearn import metrics

# gdown.download('https://drive.google.com/uc?
id=1grV8hSxULsGvnbwEMPjPakncfIOlcoB','cancer_data.csv',True);

from google.cloud import storage
def download_public_file(bucket_name, source_blob_name, destination_file_name):
    storage_client = storage.Client.create_anonymous_client()

    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(source_blob_name)
    blob.download_to_filename(destination_file_name)

    print(
        "Downloaded public blob {} from bucket {} to {}.".format(
            source_blob_name, bucket.name, destination_file_name
        )
    )

download_public_file('inspirit-ai-data-bucket-1','Data/AI Scholars/Sessions 1 -
5/Session 2b - Logistic Regression/cancer.csv','cancer_data.csv')

data = pd.read_csv('cancer_data.csv')
data['diagnosis'].replace({'M':1, 'B':0}, inplace = True)
data.to_csv('cancer_data.csv')
```

Fig 11 Importing Data

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

```
# First, we'll import some handy data visualizatior
import seaborn as sns
import matplotlib.pyplot as plt

# First, import helpful Python tools for loading/navigating data
import os              # Good for navigating your computer's files
import numpy as np     # Great for lists (arrays) of numbers
import pandas as pd    # Great for tables (google spreadsheets, micrc
csv)
from sklearn.metrics import accuracy_score   # Great for creating qu
models


data_path  = 'cancer_data.csv'

# Use the 'pd.read_csv('file')' function to read in read our data ar
in a variable called 'dataframe'
dataframe = pd.read_csv(data_path)

dataframe = dataframe[['diagnosis', 'perimeter_mean', 'radius_mean',
'texture_mean', 'area_mean', 'smoothness_mean', 'concavity_mean',
'symmetry_mean']]
```

Fig 12 Importing Modules

Next, we import some essential modules to use such as pandas, numpy and Sci-Kit Learn. We then extract only the essential parts of the database and save it in a pandas data frame. We also import some handy data visualization tools like Seaborn and Matplotlib After that, we split the dataset into training and testing dataset with training being a larger part of the dataset and testing being a smaller part of the dataset.

```
1  print('Number of rows in training dataframe:', train_df.shape[0])
2  train_df.head()
```

Number of rows in training dataframe: 455

	diagnosis	perimeter_mean	radius_mean	texture_mean	area_mean	smoothness_mean
408	1	117.80	17.99	20.66	991.7	0.10360
4	1	135.10	20.29	14.34	1297.0	0.10030
307	0	56.36	9.00	14.40	246.3	0.07005
386	0	78.78	12.21	14.09	462.0	0.08108
404	0	78.29	12.34	14.95	469.1	0.08682

```
1  print('Number of rows in test dataframe:', test_df.shape[0])
2  test_df.head()
```

Number of rows in test dataframe: 114

	diagnosis	perimeter_mean	radius_mean	texture_mean	area_mean	smoothness_mean
421	0	98.22	14.69	13.98	656.1	0.10310
47	1	85.98	13.17	18.66	534.6	0.11580
292	0	83.14	12.95	16.02	513.7	0.10050
186	1	118.60	18.31	18.58	1041.0	0.08588
414	1	96.71	15.13	29.81	719.5	0.08320

Fig 13 Training and Testing Datasets

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

We then train our logistic regression model with the training dataset X and Y values and then use the testing dataset X values to then predict new values of Y which is then plotted along with information of its actual value.

```
logreg_model = linear_model.LogisticRegression()
logreg_model.fit(X_train, y_train)
X_test = test_df[X]
y_test = test_df[y]
y_pred = logreg_model.predict(X_test)
test_df['predicted'] = y_pred.squeeze()
sns.catplot(x = X[0], y = 'diagnosis_cat', hue = 'predicted', data=test_df,
order=['1 (malignant)', '0 (benign)'])
```

Fig 14 Scikit Learn Code

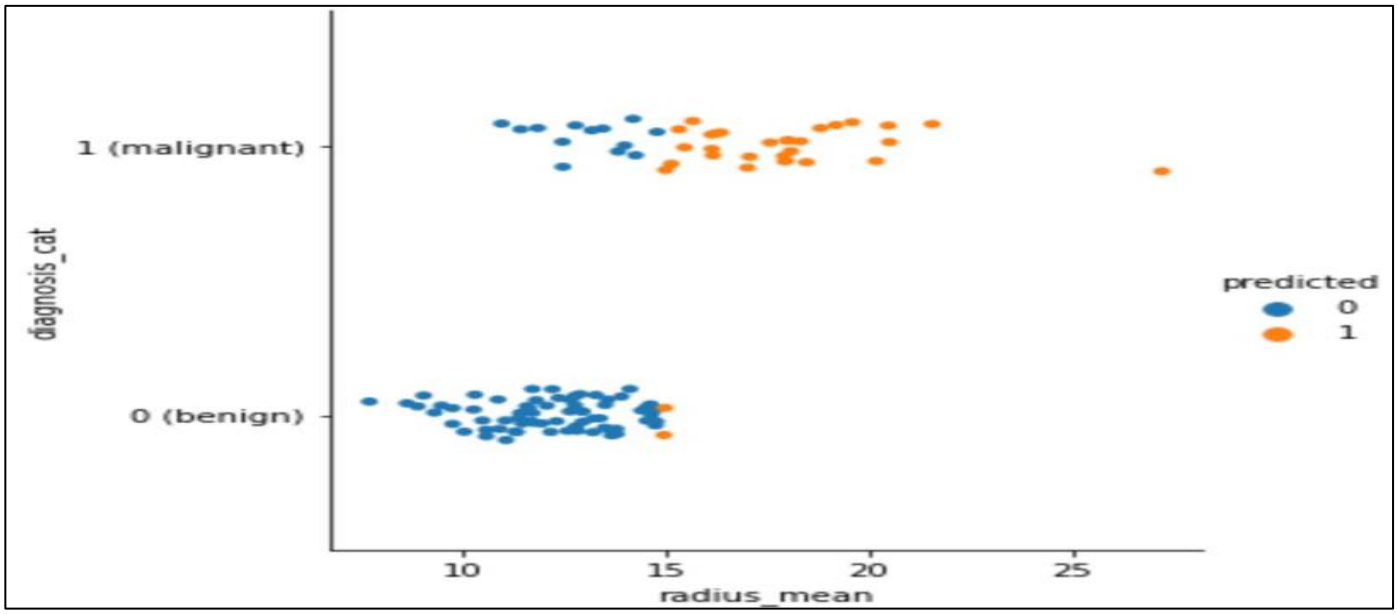

Fig 15 Prediction Data

The graph shows the actual values in y-axis, radius in x-axis and the color represents the prediction.

- *Additionally, We Can Also Find the Accuracy of the Model:*

```
1  accuracy = accuracy_score(y_test, y_pred)
2  print(accuracy)
```

```
0.868421052631579
```

Fig 16 Accuracy of the Model

Moreover, we can also access the actual soft predictions given by the sigmoid function in which Y-axis is the probability of being malignant.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

```
y_prob = logreg_model.predict_proba(X_test)
X_test_view = X_test[X].values.squeeze()
plt.xlabel('radius_mean')
plt.ylabel('Predicted Probability')
sns.scatterplot(x = X_test_view, y = y_prob[:,1], hue = y_test, palette=
['purple','green'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ac5c4c910>
```

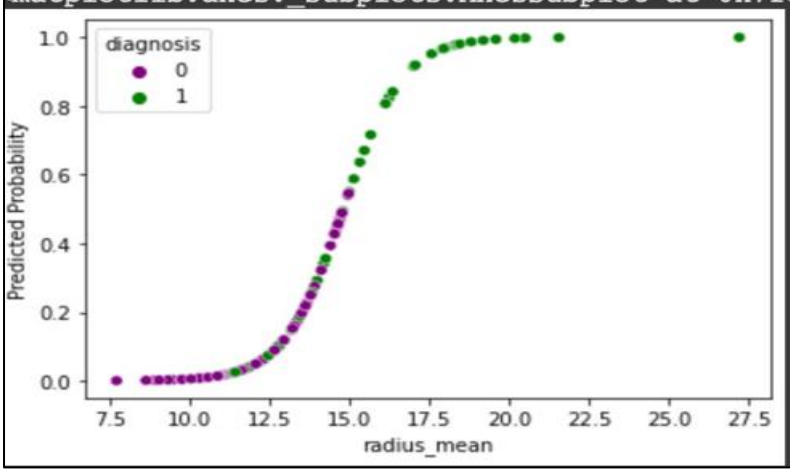

Fig 17 Sigmoid Function of the Model

Furthermore, we can then extend the same project to use multiple features instead of one feature in the following manner:

```
X = ['radius_mean','texture_mean','perimeter_mean','area_mean']
y = 'diagnosis'

# 1. Split data into train and test
multi_train_df, multi_test_df = train_test_split(dataframe, test_size = 0.2,
random_state = 1)

# 2. Prepare your X_train, X_test, y_train, and y_test variables by extracti
the appropriate columns:

X_train = multi_train_df[X]
y_train = multi_train_df[y]
X_test = multi_test_df[X]
y_test = multi_test_df[y]

# 3. Initialize the model object

logreg_model = linear_model.LogisticRegression()

# 4. Fit the model to the training data

logreg_model.fit(X_train, y_train)

# 5. Use this trained model to predict on the test data

y_guess = logreg_model.predict(X_test)

# 6. Evaluate the accuracy by comparing to to the test labels and print out
accuracy.

multi_test_df['predicted'] = y_guess.squeeze()
sns.catplot(x = X[0], y = 'diagnosis_cat', hue = 'predicted',
data=multi_test_df, order=['1 (malignant)', '0 (benign)'])

accuracy = accuracy_score(y_test, y_guess)
print(accuracy)
```

Fig 18 Scikit Learn Code

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25sep997

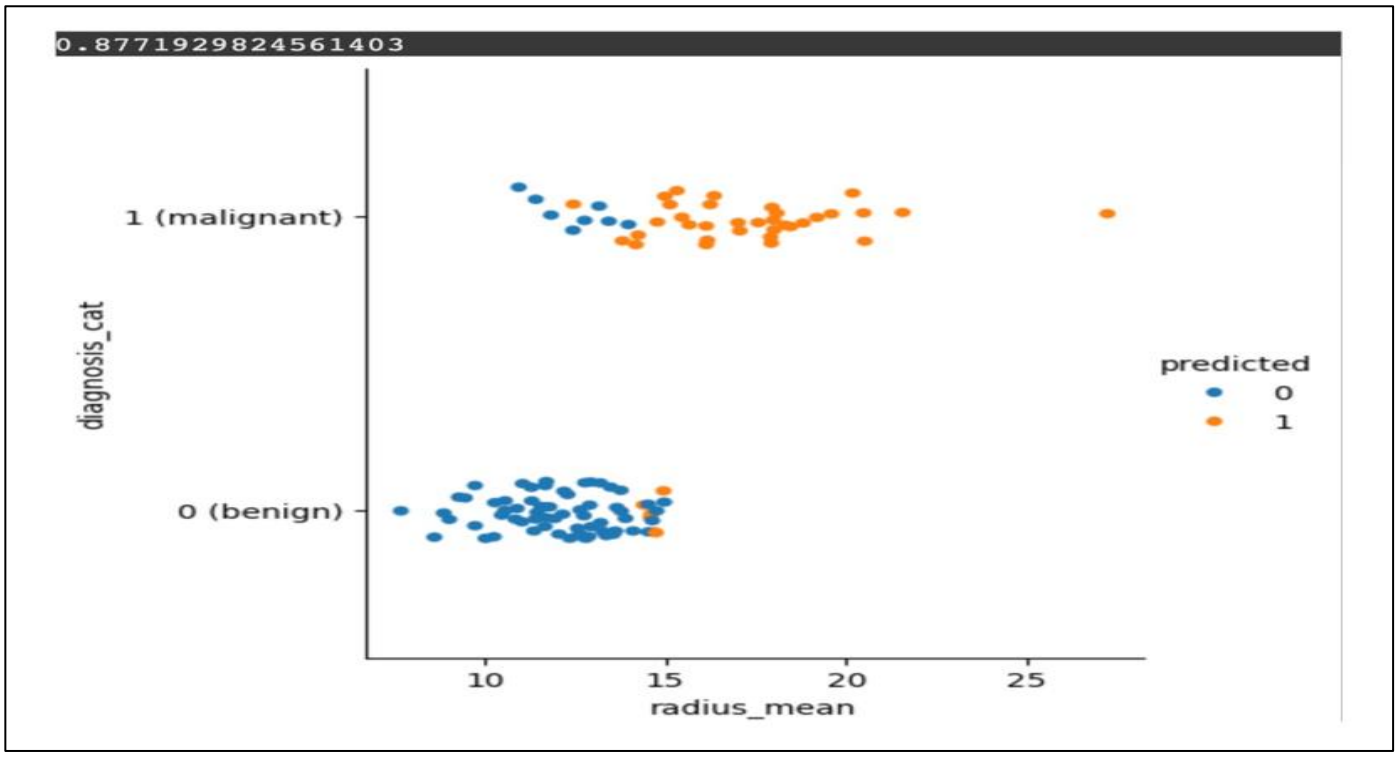

Fig 19 Multivariate Prediction Data

V. CONCLUSION

This research highlights the integral role of mathematics in developing machine learning models. By using various techniques such as Linear and Logistic Regression, ML models can form relationships between variables effectively and predict, as well as classify, new data. These techniques can be used to address various real-world issues, underscoring the importance of mathematics in our lives.

REFERENCES

[1]. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2023). *An introduction to statistical learning: With applications in Python.* Springer. Retrieved from https:// www.statlearning.com/

[2]. Šimundić, A. M. (2014). Bias in research. *Biochemia Medica, 24*(1), 12–15. https://doi.org/10.11613/BM.2014.003

[3]. Idriss, J. (2020, December 8). *Ordinary least squares and normal equations in linear regression.* Medium. Retrieved January 16, 2025, from https:// medium.com/@jairiidriss/ordinary-least-squares-and-normal-equations-in-linear- regression-85af6ccc5bf5

[4]. Khan Academy. (n.d.). *Gradient descent*. Retrieved January 16, 2025, from https://www.khanacademy.org

[5]. Google Colab. (n.d.). *Google Colab*: Your Jupyter notebook on the cloud. Retrieved January 16, 2025, from https://colab.research.google.com

[6]. Scikit-learn. (n.d.). *Linear models: Logistic regression*. Scikit-learn documentation. Retrieved January 16, 2025, from https://scikit-learn.org/stable/ modules/linear_model.html#logistic-regression

[7]. Scikit-learn. (n.d.). *Ordinary least squares.* Retrieved January 16, 2025, from https://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares

[8]. Kaggle. (n.d.). *Refugee dataset*. Retrieved January 16, 2025, from https:// www.kaggle.com