TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A REPORT
ON
LOGISTIC REGRESSION AND ITS APPLICATION

SUBMITTED BY:
AAYUSH PURI (PUL077BCT005)
BIBEK NIROULA(PUL077BCT016)
BISHRAM ACHARYA (PUL077BCT023)

SUBMITTED TO:
DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING
PULCHOWK CAMPUS

FEBRUARY, 2024

Acknowledgments

We extend our deepest gratitude to the Supreme Personality of Godhead for guiding us on the right path throughout this journey. The divine presence has been a source of inspiration and strength, leading us to successfully navigate the challenges of studying and implementing Logistic Regression.

We would like to express our sincere appreciation to the administration of IOE, Pulchowk Campus for providing the conducive environment and resources essential for the completion of this project. Their unwavering support has played a crucial role in the successful fulfillment of our Artificial Intelligence course. We are indebted to the Department of Electronics and Computer Engineering for their continuous encouragement and valuable insights that have enriched our understanding of operating systems.

We extend our special thanks to our subject teacher, Dr. Basanta Joshi, and our lab instructors, Mr. Banshee Ram Pradhan and Ms. Anila Kansakar, for providing guidance, constructive feedback, and the opportunity to delve into the fascinating world of Machine Learning.

Our heartfelt thanks go to our friends, seniors, and other fellow students who have been a constant source of motivation and camaraderie. Their shared experiences, brainstorming sessions, and collaborative efforts have significantly contributed to the success of this project.

We express our sincere appreciation to all those who have played a role, directly or indirectly, in the completion of this case study. Your support and encouragement have been instrumental in shaping this endeavor, and we are truly grateful for the guidance and opportunities provided.

Abstract

This report explores Logistic Regression, a pivotal topic in Artificial Intelligence (AI), providing a comprehensive understanding of its history, theory, and algorithmic functioning. The study details an experimental methodology wherein Logistic Regression is implemented from scratch using only the numpy library. The algorithm is applied to a dataset encompassing crucial health information such as cholesterol levels, blood pressure, and more. The primary objective is to facilitate binary classification, determining whether an individual is afflicted with a health disease based on their vital health parameters. The report delves into the intricacies of model development, training, and evaluation, shedding light on the practical implications of Logistic Regression in the context of health data analysis.

Keywords: *Logistic Regression, Artificial Intelligence, Health Informatics, Binary Classification, Numpy, Data Analysis, Model Development, Health Parameters, Disease Prediction, Binary Crossentropy, Precision, Recall, F1 Score*

Contents

List of Abbreviations

AI	Artificial Intelligence
JAMA	The Journal of the American Medical Association
MSE	Mean Squared Error
MLE	Maximum Likelihood Estimation
EDA	Exploratory Data Analysis

1. Introduction

In the context of AI, Logistic Regression is a pivotal machine learning algorithm utilized for binary classification, predicting outcomes by leveraging a logistic function to analyze input features.

1.1 Background

Artificial Intelligence (AI) has become integral across industries, and within this landscape, Logistic Regression stands as a fundamental algorithm widely applied in predictive modeling and classification.

The uniqueness of Logistic Regression lies in its ability to model the probability of an event, making it adept at binary classification tasks. Our project's distinctive angle involves a meticulous exploration of algorithmic development, aiming to unravel the mathematical foundations and theoretical constructs of Logistic Regression to contruct a Logistic Regression Model from scratch. This departure from pre-built solutions not only enhances our understanding of the algorithm but also aligns with a broader goal of advancing healthcare analytics.

In the context of healthcare, Logistic Regression proves crucial for predicting outcomes, especially in disease diagnosis. By tailoring our exploration to healthcare analytics, we aim to address the distinct challenges posed by medical data. This involves a careful consideration of risk factors and vital health parameters, ultimately aiming to improve the accuracy and interpretability of predictions related to cardiovascular diseases. Our project stands at the crossroads of AI and healthcare, seeking to contribute valuable insights and solutions for more effective medical interventions.

1.2 Problem statement

Central to our investigation is a crucial challenge: the precise prediction of cardiovascular disease likelihood, leveraging a complex array of risk factors. In contrast to the conventional approach of implementing logistic regression through established libraries, our project unveils a distinctive challenge: the construction of a bespoke model from scratch, devoid of reliance on pre-existing libraries.

1.3 Objectives

Our project is driven by two major overarching objectives.

I. **In-depth Exploration and Implementation of Logistic Regression:** To develop a comprehensive understanding of Logistic Regression, encompassing its mathematical foundations, theoretical constructs, practical applications, and simultaneously, construct a logistic regression model from scratch, emphasizing hands-on exploration and eschewing reliance on pre-existing libraries.

II. **Cardiovascular Disease Prediction:** To utilize the implemented logistic regression model to predict the probability of cardiovascular diseases, showcasing its practical application and relevance in real-world healthcare scenarios.

1.4 Scope

Our project's scope transcends traditional logistic regression implementations, specifically honing in on healthcare and cardiovascular disease prediction. This strategic focus is driven by the ambition to contribute significantly to the dynamic intersection of AI and the medical domain. Encompassing pivotal stages such as model development, rigorous training, and meticulous evaluation, our project aspires to confront and overcome the distinctive challenges presented by cardiovascular risk assessment through the lens of logistic regression.

The emphasis on healthcare signifies a deliberate and impactful choice, acknowledging the potential for AI to revolutionize medical practices, particularly in predictive analytics and personalized patient care. The scope extends beyond mere algorithmic implementation; it encapsulates a comprehensive approach that integrates domain-specific considerations, such as vital health parameters, risk factors, and disease prediction. By navigating through the intricacies of cardiovascular risk assessment, our project aims not only to advance the understanding and application of logistic regression but also to provide practical insights for healthcare practitioners.

The scope, therefore, is not confined to theoretical exploration but extends to the pragmatic realm, aligning with the broader goal of enhancing the effectiveness of healthcare interventions through the utilization of cutting-edge AI methodologies.

2. Literature Review

1. Hosmer and Lemeshow in their book *Applied Logistic Regression* [1] have given a comprehensive study of the theory, applications and practical aspects of logistic regression analysis. They also provide insights into the limitations and challenges of logistic regression while providing a guide towards alternative approaches for a broad category of problems.

2. In an issue of JAMA, Seymour et al in their paper *Logistic Regression: Relating Patient Characteristics to Outcomes* [2] presented a new method for estimating the probability of a patient dying of sepsis using information on the patient's respiratory rate, systolic blood pressure, and altered mentation. The method used these clinical characteristics—called "predictor" or explanatory or independent variables—to estimate the likelihood of a patient having an outcome of interest, called the dependent variable. To determine the best way to use these clinical characteristics, the authors used logistic regression, for quantifying the relationship between patient characteristics and clinical outcomes.

3. Zou and Hou in their paper *Logistic Regression Model Optimization and Case Analysis* [3] have studied the mathematical model of logistic, defined the error function, found the regression coefficient by gradient descent method, and improved the Sigmiod function. The paper also gives insights on how to optimize the algorithm such that it takes a lesser number of iterations to make the classification task better while maintaining the same level of accuracy.

4. Zhang and Diao have made an appreciable effort in predicting the risk of suffering from heart disease among the elderly by exploring the feasibility of using logistic regression models in their paper *Logistic Regression Models in Predicting Heart Disease* [4]. Through the technology of data mining, the main pathogenic factors of heart disease were found, and the incidence of heart disease was predicted by using the regression model. The accuracy of logistic regression model was also compared with other explored algorithm.

5. Tania and Oetama in their paper *Logistic Regression Prediction Model for Cardiovascular Disease* [5] have analyzed 14 factors that may be related to cardio vascular diseases using logistic regression analysis.

3. Methodology

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. It 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. Logistic Regression is another statistical analysis method borrowed by Machine Learning. It is used when our dependent variable is dichotomous or binary. It just means a variable that has only 2 outputs, for example, a person will survive this accident or not, the student will pass this exam or not. The outcome can either be yes or no (2 outputs). This regression technique is similar to linear regression and can be used to predict the Probabilities for classification problems.

3.1 Generalized Working

A logistic regression model works in the following steps:

1. Prepare the data: The data should be in a format where each row represents a single observation and each column represents a different variable. The target variable (the variable you want to predict) should be binary (yes/no, true/false, 0/1).

2. Train the model: We teach the model by showing it the training data. This involves finding the values of the model parameters that minimize the error in the training data.

3. Evaluate the model: The model is evaluated on the held-out test data to assess its performance on unseen data.

4. Use the model to make predictions: After the model has been trained and assessed, it can be used to forecast outcomes on new data.

3.2 Logistic Function

The logistic function, denoted by $\sigma(z)$, is a specific type of sigmoid function that maps any real-valued number z to the range [0, 1]. It is defined as:

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

where z is the input to the function.

The logistic function exhibits an S-shaped curve and is characterized by the following properties:

- Asymptotes: $\lim_{z \to -\infty} \sigma(z) = 0$ and $\lim_{z \to +\infty} \sigma(z) = 1$
- Range: $0 \leq \sigma(z) \leq 1$

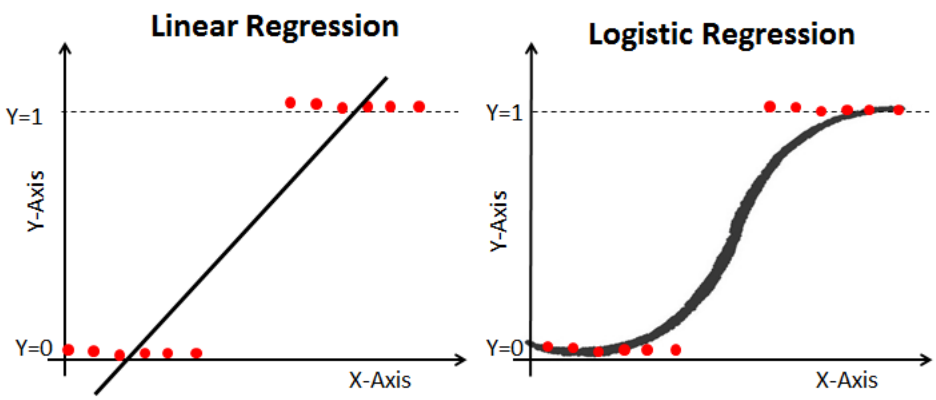


Figure 3.1: Comparison between fitting line
[6]

3.2.1 Derivative:

The derivative of the logistic function can be expressed in terms of the function itself:

$$\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z))$$

This derivative plays a crucial role in the gradient descent optimization algorithm for logistic regression.

3.2.2 Properties:

- Symmetry: The logistic function is symmetric around its midpoint ($z = 0$). This means that $\sigma(-z) = 1 - \sigma(z)$.
- Monotonicity: The logistic function is monotonically increasing.
- Continuity and Smoothness: The logistic function is continuous and infinitely differentiable for all real values of z.

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

3.2.3 Interpretation:

In logistic regression, the logistic function is used to model the probability that a given input x belongs to a particular class. The output of the logistic function, $\sigma(z)$, represents the estimated probability that the output variable belongs to class 1.

In linear regression, the equation for the best-fit line is utilized to model the relationship between independent variables and a continuous dependent variable y. However, when transitioning to logistic regression for classification tasks, utilizing probabilities (denoted as P) as the dependent variable poses a challenge. This is due to the inherent limitations of probabilities, as their values can exceed 1 or fall below 0, which contradicts the valid range of probabilities (0-1).

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

$$P = \beta_0 + \beta_1 x$$

$$\frac{P}{1 - P} = \beta_0 + \beta_1 x$$

To address the limitations associated with probabilities, logistic regression transforms the probabilities into odds. The odds represent the ratio of the probability of success to the probability of failure. While this transformation ensures positivity and a wider range of values (from 0 to positive infinity), it still retains a restricted range.

To overcome the restricted range issue and enable a more flexible modeling approach, the log of odds is employed. This transformation extends the range of values from negative to positive infinity, facilitating a more robust modeling framework.

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

The logistic function, denoted as $\sigma(z)$, is derived from the logit transformation of the odds. By exponentiating both sides of the logit equation and solving for P, we arrive at the logistic function.

$$\exp[\log(\frac{p}{1-p})] \;=\; \exp(\beta_0 + \beta_1 x)$$

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

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

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

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

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

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

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

$$Now\ dividing\ by\ e^{\left(\beta_0 + \beta_1 x\right)},\ we\ will\ get$$

$$p \;=\; \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \quad This\ is\ our\ sigmoid\ function.$$

The sigmoid function compresses a linear relationship into an S-shaped curve, facilitating the modeling of binary classification problems. This curve smoothly transitions between the two extremes, effectively capturing the probabilistic nature of the classification task.
The next thing we can do is estimate the parameters. Here, the primary parameter for us are the betas.

3.3 Cost Function in Logistic Regression

In logistic regression, the cost function plays a pivotal role in assessing the performance of the model and optimizing its parameters. Unlike linear regression, where the mean squared error (MSE) is commonly used as the cost function, logistic regression requires a different approach due to its probabilistic nature and the non-linear relationship between inputs and outputs.

Issue with Mean Squared Error

In linear regression, the mean squared error (MSE) is defined as the average squared difference between the predicted and actual values. However, applying the MSE directly to logistic regression is not suitable due to the following reasons:

1. Non-linearity: Logistic regression models the probability of a binary outcome, resulting in a non-linear relationship between the inputs and outputs. Using the MSE would not accurately capture this non-linear relationship.

2. Non-Gaussian Distribution: In logistic regression, the response variable follows a Bernoulli distribution rather than a Gaussian distribution. The assumptions underlying the derivation of the MSE are not valid for a Bernoulli-distributed response variable.

Instead of the MSE, logistic regression utilizes the log-likelihood function (or cross-entropy loss) as the cost function. The log-likelihood function quantifies the agreement between the predicted probabilities and the actual class labels in the training data. Maximizing the log-likelihood is equivalent to minimizing the prediction error and optimizing the model parameters.

3.3.1 Log-Likelihood Cost Function

The log-likelihood cost function is derived from the maximum likelihood estimation (MLE) framework. Given the binary nature of the response variable y (0 or 1), the likelihood function for a single observation $(x^{(i)}, y^{(i)})$ can be expressed as:

$$L(\theta) = \begin{cases} \sigma(\theta^T x^{(i)}) & \text{if } y^{(i)} = 1 \\ 1 - \sigma(\theta^T x^{(i)}) & \text{if } y^{(i)} = 0 \end{cases}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic (sigmoid) function, and θ represents the model parameters.

To maximize the likelihood function, we take the logarithm of the likelihood (log-likelihood):

$$\ell(\theta) = \sum_{i=1}^{m} \left[y^{(i)} \log(\sigma(\theta^T x^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma(\theta^T x^{(i)})) \right]$$

The negative log-likelihood, often referred to as the cross-entropy loss, is then used as the cost function to be minimized during model training:

$$J(\theta) = -\frac{1}{m} \ell(\theta)$$

where m is the number of training examples.

3.3.2 Derivative of the Cost function

Since the hypothesis function for logistic regression is sigmoid in nature hence, The First important step is finding the gradient of the sigmoid function. We can see from the derivation below that gradient of the sigmoid function follows a certain pattern.

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

$$\frac{d\big(\sigma(\mathrm{x})\big)}{dx} = \frac{0*(1+e^{-\mathrm{x}})-(1)*\big(e^{-\mathrm{x}}*(-1)\big)}{(1+e^{-\mathrm{x}})^2}$$

$$\frac{d\big(\sigma(\mathrm{x})\big)}{dx} = \frac{(e^{-\mathrm{x}})}{(1+e^{-\mathrm{x}})^2} = \frac{1\text{-}1+(e^{-\mathrm{x}})}{(1+e^{-\mathrm{x}})^2} = \frac{1+e^{-\mathrm{x}}}{(1+e^{-\mathrm{x}})^2} - \frac{1}{(1+e^{-\mathrm{x}})^2}$$

$$\frac{d\big(\sigma(\mathrm{x})\big)}{dx} = \frac{1}{1+e^{-\mathrm{x}}}*\left(1-\frac{1}{1+e^{-\mathrm{x}}}\right) = \sigma(\mathrm{x})\big(1-\sigma(\mathrm{x})\big)$$

Applying Chain rule and writing in terms of partial derivatives.

$$\frac{\partial\big(J(\theta)\big)}{\partial(\theta j)} = -\frac{1}{m}*\sum_{i=1}^{m}\left[y^{(i)} * \frac{1}{h_\theta\big(x^{(i)}\big)} * \frac{\partial\Big(h_\theta\big(x^{(i)}\big)\Big)}{\partial(\theta j)}\right] + \sum_{i=1}^{m}\left[\Big(1-y^{(i)}\Big) * \frac{1}{\Big(1-h_\theta\big(x^{(i)}\big)\Big)} * \frac{\partial\Big(1-h_\theta\big(x^{(i)}\big)\Big)}{\partial(\theta j)}\right]$$

$$\frac{\partial\big(J(\theta)\big)}{\partial(\theta j)} = -\frac{1}{m}*\Big(\sum_{\mathrm{i}=1}^{\mathrm{m}}\left[\mathrm{y}^{(\mathrm{i})} * \frac{1}{\mathrm{h}_\theta(\mathrm{x}^{(\mathrm{i})})} * \sigma(\mathrm{z})\big(1-\sigma(\mathrm{z})\big) * \frac{\partial(\theta^{\mathrm{T}}\mathrm{x})}{\partial(\theta\mathrm{j})}\right] + \sum_{i=1}^{m}\left[\Big(1-y^{(i)}\Big) * \frac{1}{\Big(1-h_\theta\big(x^{(i)}\big)\Big)} * \big(\text{-}\sigma(\mathrm{z})\big(1-\sigma(\mathrm{z})\big) * \frac{\partial(\theta^{\mathrm{T}}x)}{\partial(\theta j)}\right]\Big)$$

Evaluating the partial derivative using the pattern of the derivative of the sigmoid function.

$$\frac{\partial(J(\theta))}{\partial(\theta j)} = -\frac{1}{m} * (\sum_{i=1}^{m}\left[y^{(i)} * \frac{1}{h_\theta(x^{(i)})} * \sigma(z)\left(1-\sigma(z)\right) * \frac{\partial(\theta^T x)}{\partial(\theta j)}\right] + \sum_{i=1}^{m}\left[\left(1-y^{(i)}\right) * \frac{1}{\left(1-h_\theta(x^{(i)})\right)} * (-\sigma(z)\left(1-\sigma(z)\right) * \frac{\partial(\theta^T x)}{\partial(\theta j)}\right])$$

$$\frac{\partial(J(\theta))}{\partial(\theta j)} = -\frac{1}{m} * (\sum_{i=1}^{m}\left[y^{(i)} \frac{1}{h_\theta(x^{(i)})} h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right) * x_j^i\right] + \sum_{i=1}^{m}\left[\left(1-y^{(i)}\right) * \frac{1}{\left(1-h_\theta(x^{(i)})\right)} * \left(-h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right) * x_j^i\right])$$

Simplifying the terms by multiplication

$$\frac{\partial(J(\theta))}{\partial(\theta j)} = -\frac{1}{m} * (\sum_{i=1}^{m}\left[y^{(i)} * \left(1-h_\theta\left(x^{(i)}\right)\right) * x_j^i - \left(1-y^{(i)}\right) * h_\theta\left(x^{(i)}\right) * * x_j^i\right]$$

$$\frac{\partial(J(\theta))}{\partial(\theta j)} = -\frac{1}{m} * \left(\sum_{i=1}^{m}\left[y^{(i)} - y^{(i)} * h_\theta\left(x^{(i)}\right) - h_\theta\left(x^{(i)}\right) + y^{(i)} * h_\theta\left(x^{(i)}\right)\right] * x_j^i\right)$$

$$\frac{\partial(J(\theta))}{\partial(\theta j)} = -\frac{1}{m} * \left(\sum_{i=1}^{m}\left[y^{(i)} - h_\theta\left(x^{(i)}\right)\right] * x_j^i\right)$$

Removing the summation term by converting it into a matrix form for the gradient with respect to all the weights including the bias term.

$$\frac{\partial(J(\theta))}{\partial(\theta)} = \frac{1}{m} X^T[\ h_\theta(x) - y]$$

This calculus shows that both linear regression and logistic regression (actually a kind of classification) arrive at the same update rule. What we should appreciate is that the design of the cost function is part of the reasons why such “coincidence” happens.

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

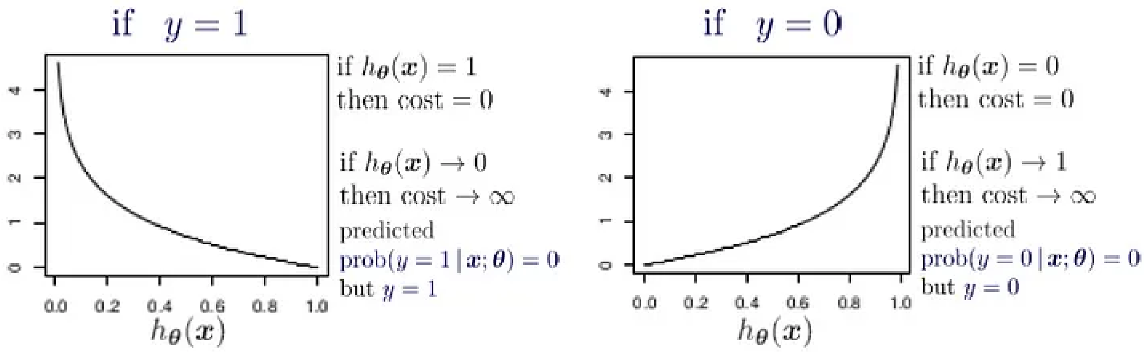


Figure 3.2: Referenced from: *A Beginner's Guide to Logistic Regression*

Here y represents the actual class and log term is the probability of that class. p(y) is the probability of 1. 1-p(y) is the probability of 0.

If we combine both the graphs, we will get a convex graph with only 1 local minimum and now it'll be easy to use gradient descent here.

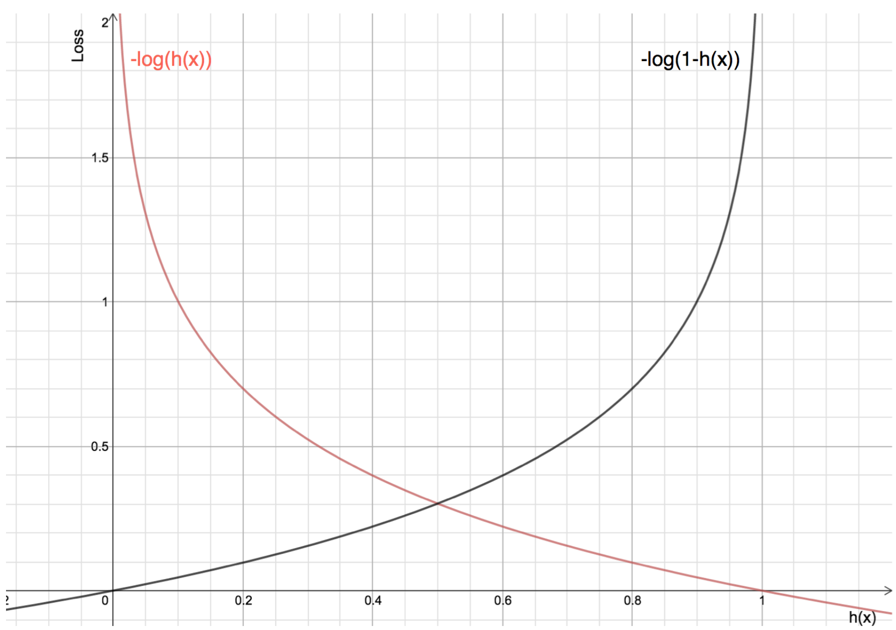


Figure 3.3: Referenced from: *A Beginner's Guide to Logistic Regression*

3.3.3 Gradient Descent Optimization

Gradient descent changes the value of our weights in such a way that it always converges to minimum point or we can also say that, it aims at finding the optimal weights which minimize the loss function of our model. It is an iterative method that finds the minimum

of a function by figuring out the slope at a random point and then moving in the opposite direction.
At first gradient descent takes a random value of our parameters from our function. Now we need an algorithm that will tell us whether at the next iteration we should move left or right to reach the minimum point. The gradient descent algorithm finds the slope of the loss function at that particular point and then in the next iteration, it moves in the opposite direction to reach the minima. Since we have a convex graph now we don't need to worry about local minima. A convex curve will always have only 1 minima.
In logistic regression, the cost function $J(\theta)$ is defined as the negative log-likelihood of the observed data:

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

where m is the number of training examples, $y^{(i)}$ is the actual class label for the i-th example, $x^{(i)}$ is the feature vector for the i-th example, $\sigma(z)$ is the sigmoid function, and θ is the parameter vector to be optimized.
To perform gradient descent, we need to compute the gradient of the cost function $J(\theta)$ with respect to the parameters θ. The gradient vector $\nabla J(\theta)$ is given by:

$$\nabla J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma(\theta^T x^{(i)}) - y^{(i)}\right)x^{(i)}$$

The parameters θ are updated iteratively using the gradient descent update rule:

$$\theta := \theta - \alpha\nabla J(\theta)$$

where α is the learning rate, a hyperparameter that controls the size of the steps taken in the parameter space during optimization.
The update rule adjusts each parameter θ_j in the direction that reduces the cost function $J(\theta)$. By subtracting the gradient $\nabla J(\theta)$ scaled by the learning rate α, the parameters are updated towards the minimum of the cost function.
Gradient descent continues iteratively until a stopping criterion is met, such as reaching a maximum number of iterations or when the change in the cost function becomes negligible between iterations.

4. Implementation

4.1 Dataset

The original data came from the Cleveland data from the UCI Machine Learning Repository (UCI Heart Disease Dataset). There is also a version of it available on Kaggle (Kaggle Heart Disease Classification Dataset).

4.1.1 Feature Dictionary

The following is a brief overview of the different features contained in the dataset regarding vital health informations.

1. **age** - age in years
2. **sex** - (1 = male; 0 = female)
3. **cp** - chest pain type
   - 0: Typical angina - chest pain related decrease blood supply to the heart
   - 1: Atypical angina - chest pain not related to heart
   - 2: Non-anginal pain - typically esophageal spasms (non-heart related)
   - 3: Asymptomatic - chest pain not showing signs of disease
4. **trestbps** - resting blood pressure (in mm Hg on admission to the hospital) anything above 130-140 is typically a cause for concern
5. **chol** - serum cholesterol in mg/dl
   - serum = LDL + HDL + .2 * triglycerides
   - above 200 is a cause for concern
6. **fbs** - (fasting blood sugar ¿ 120 mg/dl) (1 = true; 0 = false)
   - '¿126' mg/dL signals diabetes
7. **restecg** - resting electrocardiographic results
   - 0: Nothing to note

   - 1: ST-T Wave abnormality - can range from mild symptoms to severe problems, signals non-normal heart beat
   - 2: Possible or definite left ventricular hypertrophy - Enlarged heart's main pumping chamber

8. **thalach** - maximum heart rate achieved

9. **exang** - exercise-induced angina (1 = yes; 0 = no)

10. **oldpeak** - ST depression induced by exercise relative to rest looks at stress of the heart during exercise; unhealthy heart will stress more

11. **slope** - the slope of the peak exercise ST segment
   - 0: Upsloping - better heart rate with exercise (uncommon)
   - 1: Flatsloping - minimal change (typical healthy heart)
   - 2: Downsloping - signs of an unhealthy heart

12. **ca** - number of major vessels (0-3) colored by fluoroscopy
   - Colored vessel means the doctor can see the blood passing through
   - The more blood movement the better (no clots)

13. **thal** - thallium stress result
   - 1, 3: Normal
   - 6: Fixed defect - used to be a defect but okay now
   - 7: Reversible defect - no proper blood movement when exercising

14. **target** - have disease or not (1 = yes, 0 = no) (= the predicted attribute)

4.2 Building the Model

4.2.1 Sigmoid Function

```
def sigmoid(z):
    # Compute the sigmoid function using the formula:
              1 / (1 + e^(-z)).
    sigmoid_result = 1 / (1 + np.exp(-z))
    # Return the computed sigmoid value.
    return sigmoid_result
```

Compute the sigmoid function for a given input. The sigmoid function is a mathematical function used in logistic regression and neural networks to map any real-valued number to a value between 0 and 1.

Parameters:

- `z` (`float` or `numpy.ndarray`): The input value(s) for which to compute the sigmoid.

Returns:

- `float` or `numpy.ndarray`: The sigmoid of the input value(s).

4.2.2 Logistic Regression Class Implementation

```
class LogisticRegression:
    def __init__(self, learning_rate=0.0001):
        np.random.seed(1)
        self.learning_rate = learning_rate

    # Methods:
    # initialize_parameter()
    # sigmoid(z)
    # forward(X)
    # compute_cost(predictions)
    # compute_gradient(predictions)
    # fit(X, y, iterations, plot_cost)
    # predict(X)
```

- **Parameters:**
  - `learning_rate` (`float`): Learning rate for the model.
- **Methods:**
  - `initialize_parameter()`: Initializes the parameters of the model.
  - `sigmoid(z)`: Computes the sigmoid activation function for given input z.
  - `forward(X)`: Computes forward propagation for given input X.
  - `compute_cost(predictions)`: Computes the cost function for given predictions.
  - `compute_gradient(predictions)`: Computes the gradients for the model using given predictions.

– `fit(X, y, iterations, plot_cost)`: Trains the model on given input X and labels y for specified iterations.

– `predict(X)`: Predicts the labels for given input X.

4.2.3 Initialize Parameters

```
def initialize_parameter(self):
    self.W = np.zeros(self.X.shape[1])
    self.b = 0.0
```

Initializes the parameters of the model.

Parameters:

- `self`: Instance of the logistic regression model.

4.2.4 Forward Propagation

```
def forward(self, X):
    Z = np.matmul(X, self.W) + self.b
    A = sigmoid(Z)
    return A
```

Computes forward propagation for given input X.

Parameters:

- `X` (`numpy.ndarray`): Input array.

Returns:

- `numpy.ndarray`: Output array.

4.2.5 Compute Cost

```
def compute_cost(self, predictions):
    m = self.X.shape[0]  # number of training examples
    # compute the cost
    cost = np.sum((-np.log(predictions + 1e-8) * self.y)
    + (-np.log(1 - predictions + 1e-8)) * (
            1 - self.y))  # we are adding small value epsilon to
            avoid log of 0
    cost = cost / m
    return cost
```

Computes the cost function for given predictions.

Parameters:

- `predictions` (`numpy.ndarray`): Predictions of the model.

Returns:

- `float`: Cost of the model.

4.2.6 Compute Gradient

```
def compute_gradient(self, predictions):
    # get training shape
    m = self.X.shape[0]

    # compute gradients
    self.dW = np.matmul(self.X.T, (predictions - self.y))
    self.dW = np.array([np.mean(grad) for grad in self.dW])

    self.db = np.sum(np.subtract(predictions, self.y))

    # scale gradients
    self.dW = self.dW * 1 / m
    self.db = self.db * 1 / m
```

Computes the gradients for the model using given predictions.

Parameters:

- `predictions` (`numpy.ndarray`): Predictions of the model.

4.2.7 Fit the Model

```
def fit(self, X, y, iterations, plot_cost=True):
    self.X = X
    self.y = y
    self.initialize_parameter()

    costs = []
    for i in range(iterations):
        # forward propagation
```

```
        predictions = self.forward(self.X)

        # compute cost
        cost = self.compute_cost(predictions)
        costs.append(cost)

        # compute gradients
        self.compute_gradient(predictions)

        # update parameters
        self.W = self.W - self.learning_rate * self.dW
        self.b = self.b - self.learning_rate * self.db

        # print cost every 100 iterations
        if i % 10000 == 0:
            print("Cost after iteration {}: {}".format(i, cost))

    if plot_cost:
        fig = px.line(y=costs, title="Cost vs Iteration",
                      template="plotly_dark")
        fig.update_layout(
            title_font_color="#41BEE9",
            xaxis=dict(color="#41BEE9", title="Iterations"),
            yaxis=dict(color="#41BEE9", title="cost")
        )
        fig.show()
```

Trains the model on given input X and labels y for specified iterations.

Parameters:

- `X` (`numpy.ndarray`): Input features array of shape (n_samples, n).
- `y` (`numpy.ndarray`): Labels array of shape (n_samples, 1).
- `iterations` (`int`): Number of iterations for training.
- `plot_cost` (`bool`): Whether to plot cost over iterations or not.

Returns:

- `None`.

4.3 Applying The Model

4.3.1 Predict

```
def predict(self, X):
    predictions = self.forward(X)
    return np.round(predictions)
```

Predicts the labels for given input X.

Parameters:

- `X` (`numpy.ndarray`): Input features array.

Returns:

- `numpy.ndarray`: Predicted labels.

4.3.2 Save Model

```
def save_model(self, filename=None):
    model_data = {
        'learning_rate': self.learning_rate,
        'W': self.W,
        'b': self.b
    }

    with open(filename, 'wb') as file:
        pickle.dump(model_data, file)
```

Save the trained model to a file using pickle.

Parameters:

- `filename` (`str`): The name of the file to save the model to.

4.3.3 Load Model

```
@classmethod
def load_model(cls, filename):
    with open(filename, 'rb') as file:
        model_data = pickle.load(file)
```

```
    # Create a new instance of the class and initialize
            it with the loaded parameters
    loaded_model = cls(model_data['learning_rate'])
    loaded_model.W = model_data['W']
    loaded_model.b = model_data['b']

    return loaded_model
```

Load a trained model from a file using pickle. We call the class from within using `cls` under `classmethod`.

Parameters:

- `filename` (`str`): The name of the file to load the model from.

Returns:

- `LogisticRegression`: An instance of the LogisticRegression class with loaded parameters.

4.4 Classification Metrics

4.4.1 Classification Metrics Class

Accuracy

```
@staticmethod
def accuracy(y_true, y_pred):
    total_samples = len(y_true)
    correct_predictions = np.sum(y_true == y_pred)
    return (correct_predictions / total_samples)
```

Computes the accuracy of a classification model.

Parameters:

- `y_true` (`numpy array`): A numpy array of true labels for each data point.
- `y_pred` (`numpy array`): A numpy array of predicted labels for each data point.

Returns:

- `float`: The accuracy of the model, expressed as a percentage.

Precision

```
@staticmethod
def precision(y_true, y_pred):
    true_positives = np.sum((y_true == 1) & (y_pred == 1))
    false_positives = np.sum((y_true == 0) & (y_pred == 1))
    return true_positives / (true_positives + false_positives)
```

Computes the precision of a classification model.

Parameters:

- `y_true` (`numpy array`): A numpy array of true labels for each data point.
- `y_pred` (`numpy array`): A numpy array of predicted labels for each data point.

Returns:

- `float`: The precision of the model, which measures the proportion of true positive predictions out of all positive predictions made by the model.

Recall

```
@staticmethod
def recall(y_true, y_pred):
    true_positives = np.sum((y_true == 1) & (y_pred == 1))
    false_negatives = np.sum((y_true == 1) & (y_pred == 0))
    return true_positives / (true_positives + false_negatives)
```

Computes the recall (sensitivity) of a classification model.

Parameters:

- `y_true` (`numpy array`): A numpy array of true labels for each data point.
- `y_pred` (`numpy array`): A numpy array of predicted labels for each data point.

Returns:

- `float`: The recall of the model, which measures the proportion of true positive predictions out of all actual positive instances in the dataset.

F1-Score

```python
@staticmethod
def f1_score(y_true, y_pred):
    precision_value = ClassificationMetrics.precision(y_true, y_pred)
    recall_value = ClassificationMetrics.recall(y_true, y_pred)
    return 2 * (precision_value * recall_value) /
                        (precision_value + recall_value)
```

Computes the F1-score of a classification model.

Parameters:

- `y_true` (`numpy array`): A numpy array of true labels for each data point.
- `y_pred` (`numpy array`): A numpy array of predicted labels for each data point.

Returns:

- `float`: The F1-score of the model, which is the harmonic mean of precision and recall.

4.5 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique widely used in machine learning and data analysis to transform high-dimensional data into a lower-dimensional space while preserving most of the important information. PCA aims to identify the directions (or principal components) that capture the maximum variance in the data and project the data onto these components.

```python
from sklearn.decomposition import PCA

# Apply PCA to reduce the dimensionality to 2 dimensions
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
```

Applies Principal Component Analysis (PCA) to reduce the dimensionality of the data to 2 dimensions.

Parameters:

- `n_components` (`int`): Number of components to keep. In this case, it's set to 2 to reduce the data to 2 dimensions.

Returns:

- `X_reduced` (`numpy.ndarray`): Transformed array with reduced dimensionality.

5. Results

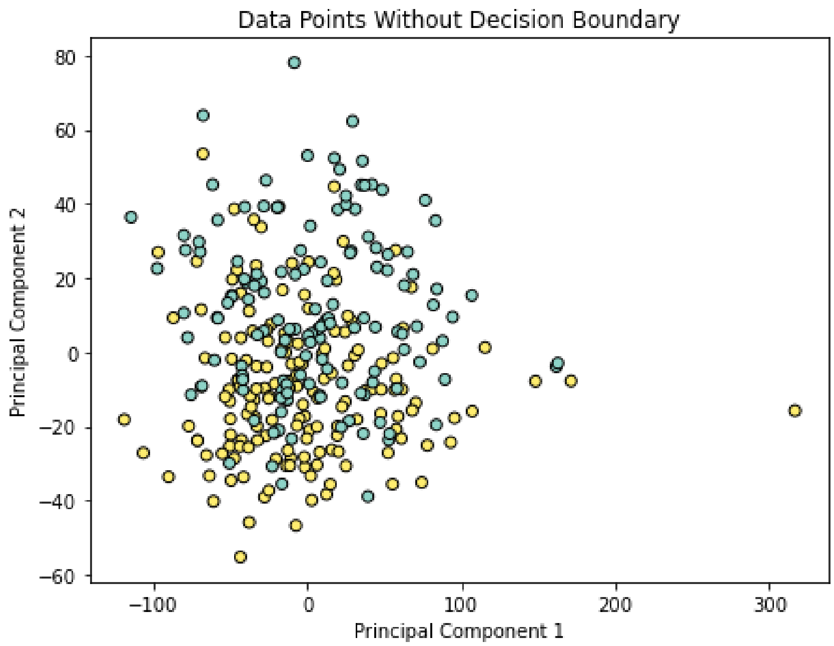


Figure 5.1: Boundary division of classes after PCA

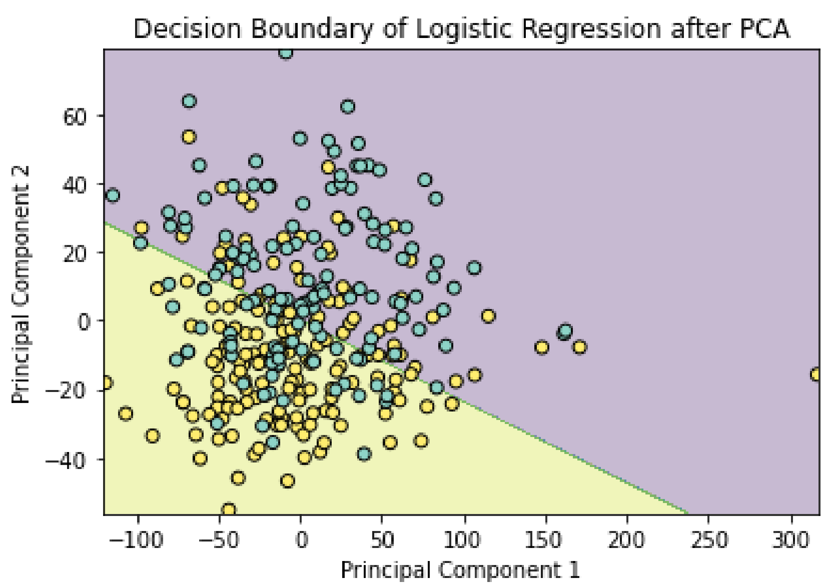


Figure 5.2: Boundary division of classes after Logistic Regression

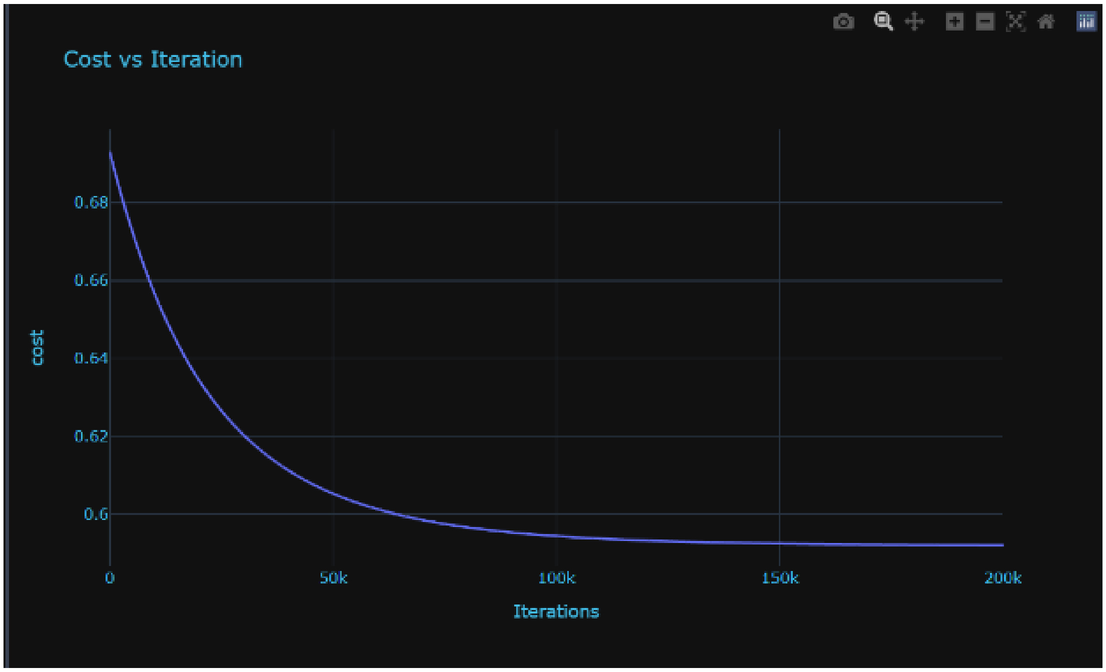


Figure 5.3: Cost vs Iteration Graph

5.1 Evaluation Metrics :

1. Accuracy: 80.33 precent
2. Precision: 81.40 percent
3. Recall: 89.74 percent
4. F1-Score: 85.37 percent

5.2 Test Demo

```
for i in range(5):


    label=y_test.iloc[i]
    print(f'True label:{label}.0')
    pred=model.predict(X_test.iloc[i])
    print(f"Prediction:{pred}")
```

```
True label:0.0
Prediction:0.0
True label:1.0
Prediction:1.0
True label:0.0
Prediction:0.0
True label:0.0
Prediction:0.0
True label:0.0
Prediction:0.0
```

Figure 5.4: Test Demonstration on a portion of data

References

[1] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied Logistic Regression*. John Wiley & Sons, 2013.

[2] Juliana Tolles and William J. Meurer. Logistic Regression: Relating Patient Characteristics to Outcomes. *JAMA*, 316(5):533–534, 08 2016.

[3] Xiaonan Zou, Yong Hu, Zhewen Tian, and Kaiyuan Shen. Logistic regression model optimization and case analysis. In *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*, pages 135–139, 2019.

[4] Yingjie Zhang, Lijuan Diao, and Linlin Ma. Logistic regression models in predicting heart disease. *Journal of Physics: Conference Series*, 1769(1):012024, jan 2021.

[5] Tania Ciu and Raymond Oetama. Logistic regression prediction model for cardiovascular disease. *IJNMT (International Journal of New Media Technology)*, 7:33–38, 07 2020.

[6] Harshwardhan Jadhav. Understanding the power of logistic regression in machine learning, 2023. LinkedIn Article.

Appendices

Appendix 1: Jupyter Notebook of Exploratory Data Analysis(EDA)

Exploratory Data Analysis

March 1, 2024

1 Predicting heart disease using machine learning

This notebook looks into using logistic regression in an attempt to build a machine learning model capable of predicting whether or not someone has heart disease based on their medical attributes.

We're going to take the following approach: 1. Problem definition 2. Data 3. Modelling 4. Experimentation

1.1 1. Problem Definition

In a statement, > Given clinical parameters about a patient, can we predict whether or not they have heart disease?

1.2 2. Data

The original data came from the Cleavland data from the UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/heart+Disease

There is also a version of it available on Kaggle. https://www.kaggle.com/datasets/sumaiyatasmeem/heart-disease-classification-dataset

1.3 3. Features

Data dictionary

1. age - age in years
2. sex - (1 = male; 0 = female)
3. cp - chest pain type
    - 0: Typical angina: chest pain related decrease blood supply to the heart
    - 1: Atypical angina: chest pain not related to heart
    - 2: Non-anginal pain: typically esophageal spasms (non heart related)
    - 3: Asymptomatic: chest pain not showing signs of disease
4. trestbps - resting blood pressure (in mm Hg on admission to the hospital) anything above 130-140 is typically cause for concern
5. chol - serum cholestoral in mg/dl
    - serum = LDL + HDL + .2 * triglycerides
    - above 200 is cause for concern
6. fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
    - '>126' mg/dL signals diabetes
7. restecg - resting electrocardiographic results

- 0: Nothing to note
- 1: ST-T Wave abnormality
  - can range from mild symptoms to severe problems
  - signals non-normal heart beat
- 2: Possible or definite left ventricular hypertrophy
  - Enlarged heart's main pumping chamber

8. thalach - maximum heart rate achieved
9. exang - exercise induced angina (1 = yes; 0 = no)
10. oldpeak - ST depression induced by exercise relative to rest looks at stress of heart during excercise unhealthy heart will stress more
11. slope - the slope of the peak exercise ST segment
    - 0: Upsloping: better heart rate with excercise (uncommon)
    - 1: Flatsloping: minimal change (typical healthy heart)
    - 2: Downslopins: signs of unhealthy heart
12. ca - number of major vessels (0-3) colored by flourosopy
    - colored vessel means the doctor can see the blood passing through
    - the more blood movement the better (no clots)
13. thal - thalium stress result
    - 1,3: normal
    - 6: fixed defect: used to be defect but ok now
    - 7: reversable defect: no proper blood movement when excercising
14. target - have disease or not (1=yes, 0=no) (= the predicted attribute)

1.4 Preparing the tools

We're going to use pandas, Matplotlib and NumPy for data analysis and manipulation.

[ ]:
```
# Import all the tools we need

# Regular EDA (exploratory data analysis) and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# we want our plots to appear inside the notebook
%matplotlib inline

# Models from Scikit-Learn
from sklearn.linear_model import LogisticRegression


# Model Evaluations
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import plot_roc_curve
```

1.5 Load data

[ ]:
```
df = pd.read_csv("heart-disease.csv")
df.shape # (rows, columns)
```

[ ]: (303, 14)

[ ]:
```
# Split data into X and y
X = df.drop("target", axis=1)

y = df["target"]
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X=pca.fit_transform(X)
```

[ ]:
```
y
```

[ ]:
```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

1.6 Data Exploration (exploratory data analysis or EDA)

The goal here is to find out more about the data and become a subject matter export on the dataset you're working with.

1. What question(s) are you trying to solve?
2. What kind of data do we have and how do we treat different types?
3. What's missing from the data and how do you deal with it?
4. Where are the outliers and why should you care about them?
5. How can you add, change or remove features to get more out of your data?

[ ]:
```
df.head()
```

[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	63	1	3	145	233	1	0	150	0	2.3	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	

```
   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

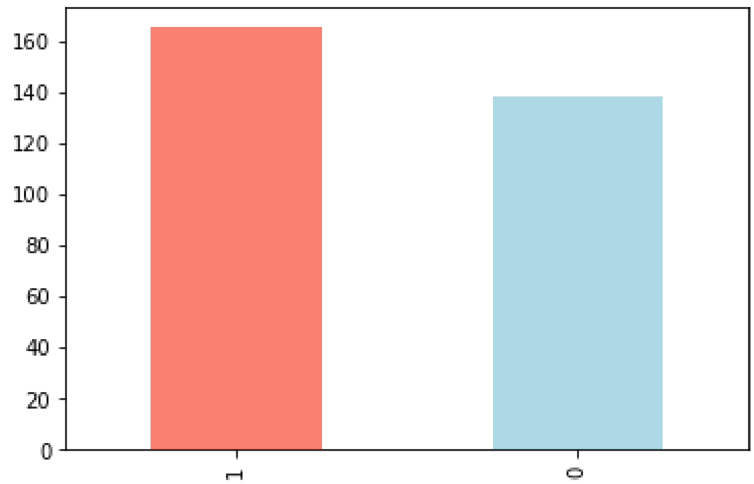
[ ]:
```
df.tail()
```

[ ]:
```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

[ ]:
```
# Let's find out how many of each class there
df["target"].value_counts()
```

[ ]:
```
1    165
0    138
Name: target, dtype: int64
```

[ ]:
```
df["target"].value_counts().plot(kind="bar", color=["salmon", "lightblue"]);
```



[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

[ ]:
```
# Are there any missing values?
df.isna().sum()
```

```
[ ]: age         0
     sex         0
     cp          0
     trestbps    0
     chol        0
     fbs         0
     restecg     0
     thalach     0
     exang       0
     oldpeak     0
     slope       0
     ca          0
     thal        0
     target      0
     dtype: int64
```

```
[ ]: df.describe()
```

```
[ ]:           age         sex          cp    trestbps        chol         fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148515
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729373
std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022606
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
50%      1.000000  153.000000    0.000000    0.800000    1.000000    0.000000
75%      1.000000  166.000000    1.000000    1.600000    2.000000    1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000000

             thal      target
count  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

1.6.1 Heart Disease Frequency according to Sex

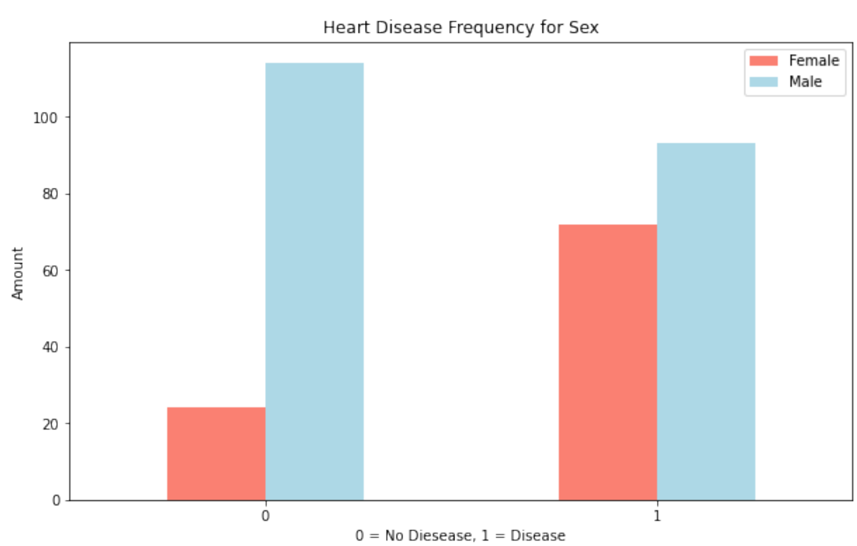
[ ]:
```
df.sex.value_counts()
```

[ ]:
```
1    207
0     96
Name: sex, dtype: int64
```

[ ]:
```
# Compare target column with sex column
pd.crosstab(df.target, df.sex)
```

[ ]:
```
sex      0    1
target
0       24  114
1       72   93
```

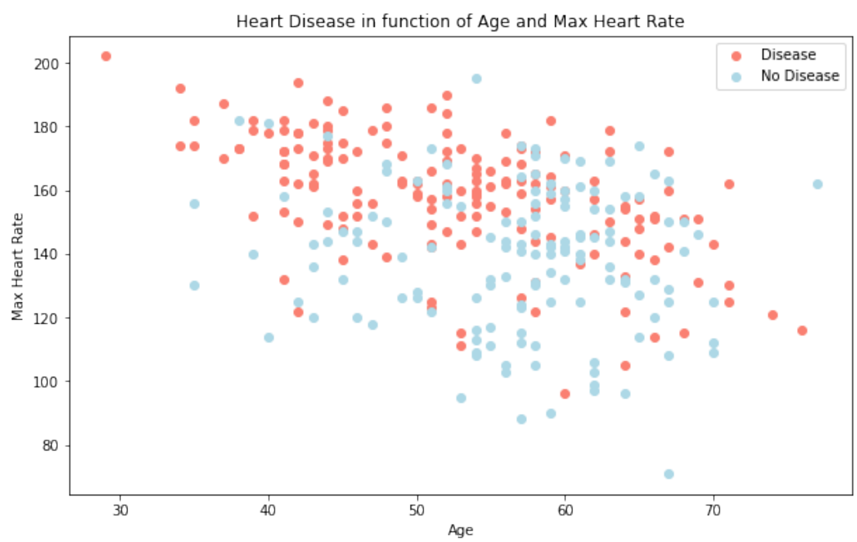
[ ]:
```
# Create a plot of crosstab
pd.crosstab(df.target, df.sex).plot(kind="bar",
                                    figsize=(10, 6),
                                    color=["salmon", "lightblue"])

plt.title("Heart Disease Frequency for Sex")
plt.xlabel("0 = No Diesease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female", "Male"]);
plt.xticks(rotation=0);
```



7

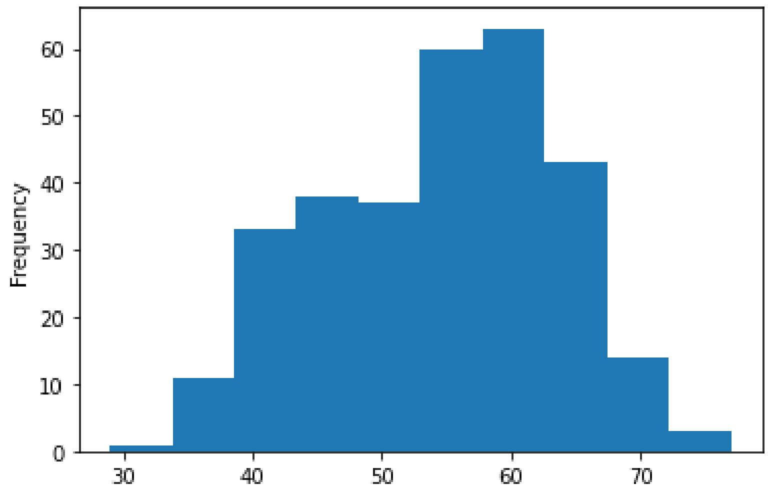
1.6.2 Age vs. Max Heart Rate for Heart Disease

[ ]:
```
# Create another figure
plt.figure(figsize=(10, 6))

# Scatter with postivie examples
plt.scatter(df.age[df.target==1],
            df.thalach[df.target==1],
            c="salmon")

# Scatter with negative examples
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            c="lightblue")

# Add some helpful info
plt.title("Heart Disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease", "No Disease"]);
```



8

```
[ ]: # Check the distribution of the age column with a histogram
df.age.plot.hist();
```



1.6.3 Heart Disease Frequency per Chest Pain Type

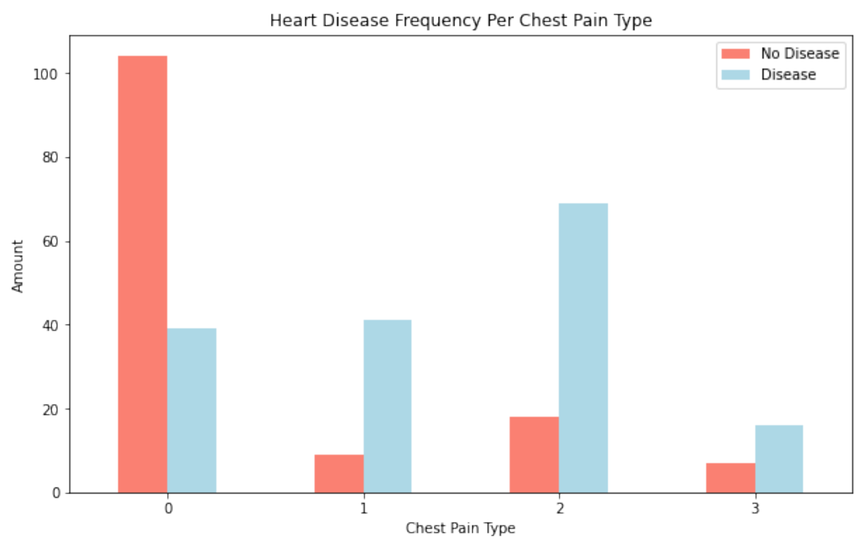
3. cp - chest pain type
    - 0: Typical angina: chest pain related decrease blood supply to the heart
    - 1: Atypical angina: chest pain not related to heart
    - 2: Non-anginal pain: typically esophageal spasms (non heart related)
    - 3: Asymptomatic: chest pain not showing signs of disease

```
[ ]: pd.crosstab(df.cp, df.target)
```

```
[ ]: target    0   1
     cp
     0       104  39
     1         9  41
     2        18  69
     3         7  16
```

```
[ ]: # Make the crosstab more visual
pd.crosstab(df.cp, df.target).plot(kind="bar",
                                   figsize=(10, 6),
                                   color=["salmon", "lightblue"])

# Add some communication
```

```
plt.title("Heart Disease Frequency Per Chest Pain Type")
plt.xlabel("Chest Pain Type")
plt.ylabel("Amount")
plt.legend(["No Disease", "Disease"])
plt.xticks(rotation=0);
```



[ ]:
```
df.head()
```

[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	63	1	3	145	233	1	0	150	0	2.3	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	

	ca	thal	target
0	0	1	1
1	0	2	1
2	0	2	1
3	0	2	1
4	0	2	1

[ ]:
```
# Make a correlation matrix
df.corr()
```

10

[ ]:
```
               age       sex        cp  trestbps      chol       fbs  \
age       1.000000 -0.098447 -0.068653  0.279351  0.213678  0.121308
sex      -0.098447  1.000000 -0.049353 -0.056769 -0.197912  0.045032
cp       -0.068653 -0.049353  1.000000  0.047608 -0.076904  0.094444
trestbps  0.279351 -0.056769  0.047608  1.000000  0.123174  0.177531
chol      0.213678 -0.197912 -0.076904  0.123174  1.000000  0.013294
fbs       0.121308  0.045032  0.094444  0.177531  0.013294  1.000000
restecg  -0.116211 -0.058196  0.044421 -0.114103 -0.151040 -0.084189
thalach  -0.398522 -0.044020  0.295762 -0.046698 -0.009940 -0.008567
exang     0.096801  0.141664 -0.394280  0.067616  0.067023  0.025665
oldpeak   0.210013  0.096093 -0.149230  0.193216  0.053952  0.005747
slope    -0.168814 -0.030711  0.119717 -0.121475 -0.004038 -0.059894
ca        0.276326  0.118261 -0.181053  0.101389  0.070511  0.137979
thal      0.068001  0.210041 -0.161736  0.062210  0.098803 -0.032019
target   -0.225439 -0.280937  0.433798 -0.144931 -0.085239 -0.028046

           restecg   thalach     exang   oldpeak     slope        ca  \
age      -0.116211 -0.398522  0.096801  0.210013 -0.168814  0.276326
sex      -0.058196 -0.044020  0.141664  0.096093 -0.030711  0.118261
cp        0.044421  0.295762 -0.394280 -0.149230  0.119717 -0.181053
trestbps -0.114103 -0.046698  0.067616  0.193216 -0.121475  0.101389
chol     -0.151040 -0.009940  0.067023  0.053952 -0.004038  0.070511
fbs      -0.084189 -0.008567  0.025665  0.005747 -0.059894  0.137979
restecg   1.000000  0.044123 -0.070733 -0.058770  0.093045 -0.072042
thalach   0.044123  1.000000 -0.378812 -0.344187  0.386784 -0.213177
exang    -0.070733 -0.378812  1.000000  0.288223 -0.257748  0.115739
oldpeak  -0.058770 -0.344187  0.288223  1.000000 -0.577537  0.222682
slope     0.093045  0.386784 -0.257748 -0.577537  1.000000 -0.080155
ca       -0.072042 -0.213177  0.115739  0.222682 -0.080155  1.000000
thal     -0.011981 -0.096439  0.206754  0.210244 -0.104764  0.151832
target    0.137230  0.421741 -0.436757 -0.430696  0.345877 -0.391724

              thal    target
age       0.068001 -0.225439
sex       0.210041 -0.280937
cp       -0.161736  0.433798
trestbps  0.062210 -0.144931
chol      0.098803 -0.085239
fbs      -0.032019 -0.028046
restecg  -0.011981  0.137230
thalach  -0.096439  0.421741
exang     0.206754 -0.436757
oldpeak   0.210244 -0.430696
slope    -0.104764  0.345877
ca        0.151832 -0.391724
thal      1.000000 -0.344029
target   -0.344029  1.000000
```

[ ]:
```python
# Let's make our correlation matrix a little prettier
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize=(15, 10))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.5,
                 fmt=".2f",
                 cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[ ]: (14.5, -0.5)