Machine Learning

Unit I: BASIC MATHEMATICS FOR MACHINE LEARNING:

Regression Correlation and Regression, types of correlation – Pearson's, Spearman's correlations –Ordinary Least Squares, Fitting a regression line, logistic regression, Rank Correlation Partial and Multiple correlation-Multiple regression, multicollinearity. Gradient descent methods, Newton method, interior point methods, active set, proximity methods, accelerated gradient methods, coordinate descent, cutting planes, stochastic gradient descent. Discriminant analysis, Principal component analysis, Factor analysis, k means.

—------ —------ —------ —------ —------ —------ —------

Regression_and_Correlation.pdf:(You can download the PDF file)

Correlation and regression are the two most commonly used techniques for investigating the relationship between quantitative variables. Here regression refers to linear regression. Correlation is used to give the relationship between the variables whereas linear regression uses an equation to express this relationship.

What are Correlation and Regression?

- Correlation and regression are statistical measurements that are used to give a relationship between two variables.
- For example, suppose a person is driving an expensive car then it is assumed that she must be financially well. To numerically quantify this relationship, correlation and regression are used.

Define Correlation

- Correlation can be defined as a measurement that is used to quantify the relationship between variables. If an increase (or decrease) in one variable causes a corresponding increase (or decrease) in another then the two variables are said to be directly correlated.
- Similarly, if an increase in one causes a decrease in another or vice versa, then the variables are said to be indirectly correlated. If a change in an independent variable does not cause a change in the dependent variable then they are uncorrelated.
- Thus, correlation can be positive (direct correlation), negative (indirect correlation), or zero. This relationship is given by the correlation coefficient.

Regression Definition

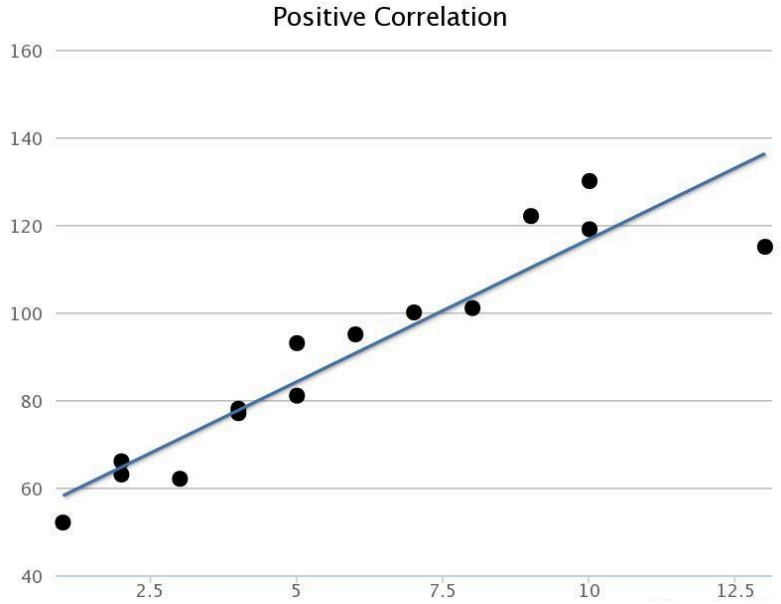
- Regression can be defined as a measurement that is used to quantify how the change in one variable will affect another variable.
- Regression is used to find the cause and effect between two variables.
- Linear regression is the most commonly used type of regression because it is easier to analyze as compared to the rest.

- Linear regression is used to find the line that is the best fit to establish a relationship between variables.

Types of correlation:

1. Positive Linear Correlation

   There is a *positive linear correlation* when the variable on the x-axis increases as the variable on the y-axis increases. This is shown by an upwards sloping straight regression line.



Worked Example

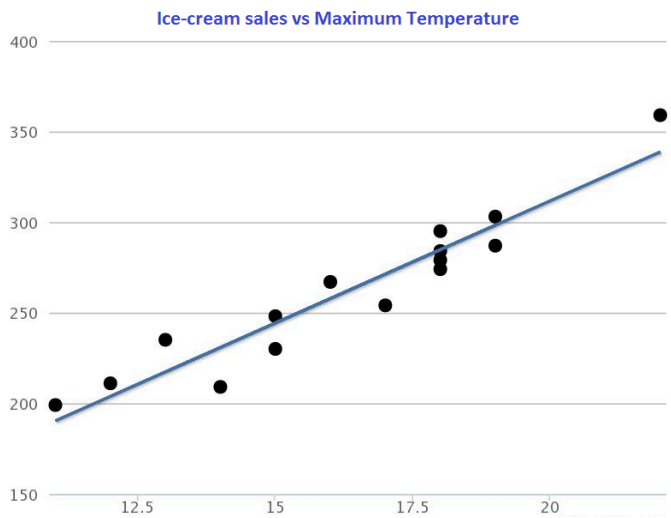
The local ice-cream shop have kept track of how much ice-cream they sell and the maximum temperature on that day. The data that they obtained during the last 15 days is as follows:

Temperature (°c)	Ice-cream Sales (£)
12.5	211
15.8	230
22.1	359
18.9	284
17.7	254
19.3	287
15.3	248
19.2	303
13.4	235
14.1	209
16.7	267
18.6	295
11.9	199
18.4	274
18.9	279

Determine the type of correlation between the number of ice-cream sales and the maximum temperature of the day.

Solution

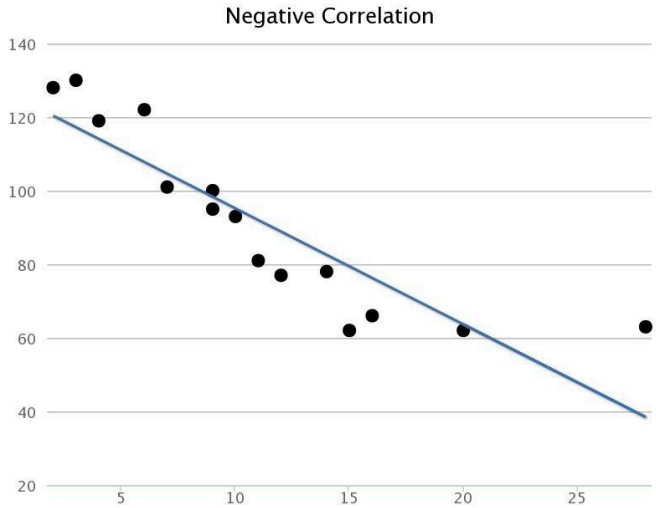
Firstly draw a scatter diagram with the given data.



This shows that there is a strong positive linear correlation between ice-cream sales and maximum temperature. However, it is not always as easy to tell just by looking at the scatter graph, instead we quantify it using a numeric value known as the correlation coefficient.

Negative Linear Correlation

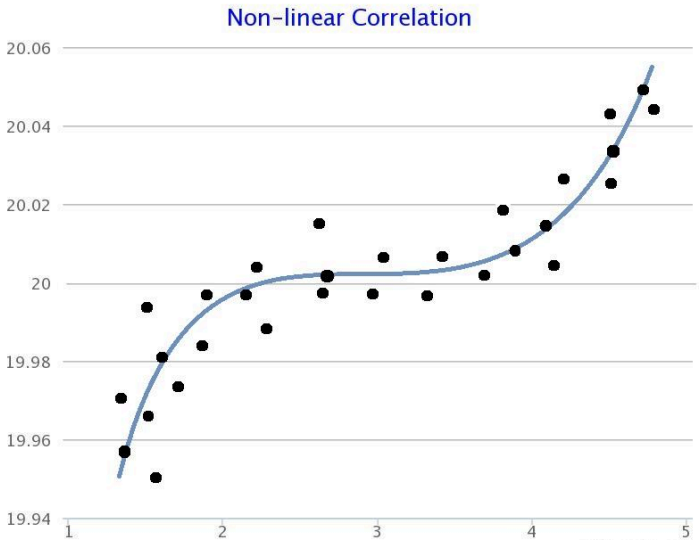
There is a *negative linear correlation* when one variable increases as the other variable decreases. This is shown by a downwards sloping straight regression line.



Non-linear Correlation (known as curvilinear correlation)

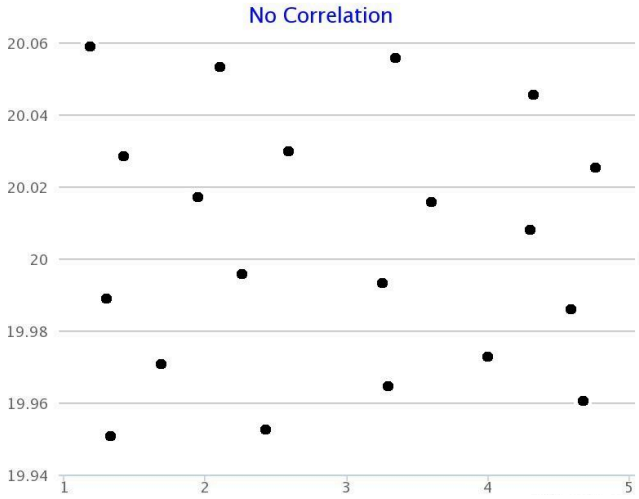
There is a *non-linear correlation* when there is a relationship between variables but the relationship is not linear (straight).

Machine Learning



No Correlation

There is *no correlation* when there is no pattern that can be detected between the variables.



Pearson's correlation & Spearman's correlations :

1. Correlation Coefficients
2. Least Squares Regression

Ordinary Least Squares (OLS) regression is a fundamental concept in machine learning, widely used for predictive modeling. It's a linear regression technique that aims to find the best-fitting straight line through data points.

OLS Regression Works

- OLS minimizes the sum of squared differences between the actual and predicted values (the "least squares" part).

- It finds the slope (coefficient) and intercept of the line that minimize this sum.

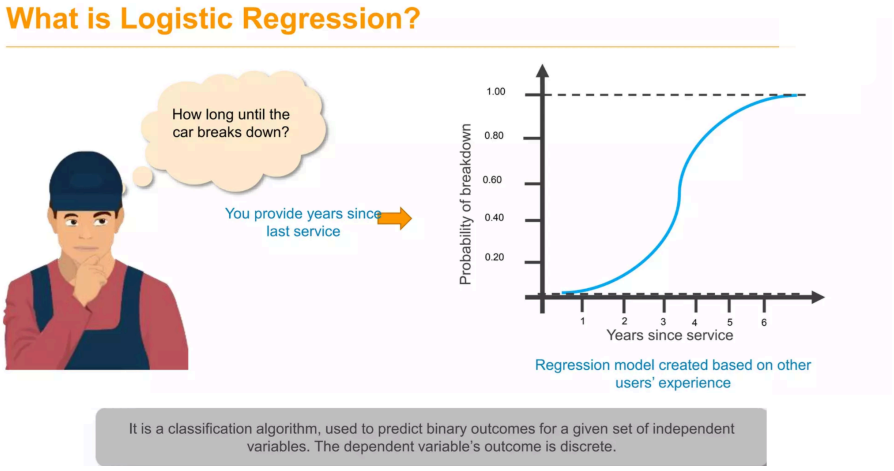
Types of OLS Regression

1. Simple Linear Regression: Used when there's a single independent variable.Formula: $y = b_0 + b_1x_1 + e$Example: Predicting a student's test score based on the number of hours they studied.
2. Multiple Linear Regression: Deals with multiple independent variables. Formula: $y = b_0 + b_1x_1 + b_2x_2 + ... + b\square x\square + e$Example: Predicting house prices based on variables like size, location, and number of bedrooms.
3. Polynomial Regression: Used when relationships between variables are non-linear. Formula: $y = b_0 + b_1x_1 + b_2x_1^2 + ... + b\square x_1^n + e$
   Example: Predicting stock prices that exhibit quadratic behavior.

Real Case Studies Examples:

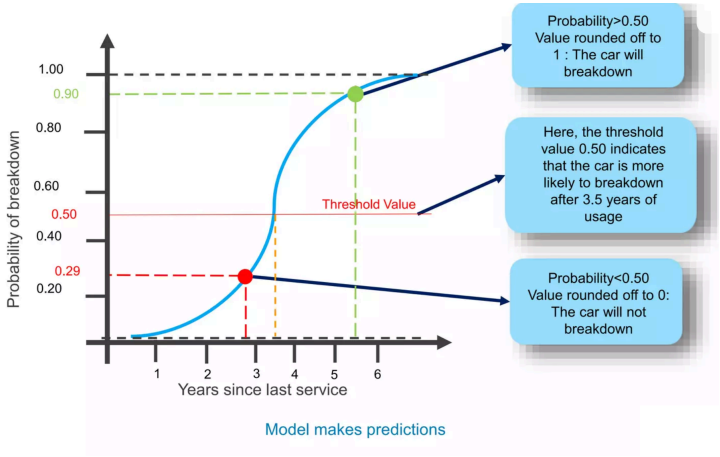
1. **Simple Linear Regression:**
   Example: Predicting a car's fuel efficiency (MPG) based on its weight.
   Case Study: A car manufacturer used simple linear regression to determine how weight impacts fuel efficiency, aiding in designing more fuel-efficient cars.
2. **Multiple Linear Regression:**
   Example: Predicting a patient's hospital length of stay based on age, medical history, and diagnosis.
   Case Study: A hospital used multiple linear regression to optimize resource allocation and improve patient care planning.
3. **Polynomial Regression:**
   Example: Predicting the price of a smartphone based on its features.
   Case Study: A smartphone manufacturer employed polynomial regression to capture complex relationships between features and price, aiding in pricing strategies.

Logistic Regression (Binary Dependent Variable)



Machine Learning

- Define: Logistic regression is a statistical algorithm which analyzes the relationship between two data factors. It takes input as independent variables and produces a probability value between 0 and 1.
- Usage: 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 predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value.
- It can be either Yes or No, 0 or 1, True or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- In Logistic regression, instead of fitting a regression line, we fit an “S” shaped logistic function, which predicts two maximum values (0 or 1).



This Logistic regression technique is similar to linear regression and can be used to predict the Probabilities for classification problems.

Types of 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

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.

```
print ("\nInstance ", i+1, "is", a[i], " and is Negative Instance Hence Ignored")
```

```
    print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")

print("\nThe Maximally specific hypothesis for the training instance is ", hypothesis)
```

Rank Correlation Partial:

Rank Correlation Partial is a statistical technique used to measure the strength and direction of association between two variables while controlling for the effect of one or more other variables. It is a partial correlation analysis method that uses ranks instead of the original values of the variables.

Calculating Rank Correlation Partial are:

1. Rank the variables involved (the two variables of interest and the control variable(s)) separately.
2. Calculate the Spearman rank correlation coefficient between each pair of variables.
3. Use the formula for partial correlation to calculate the rank correlation between the two variables of interest, controlling for the effect of the other variable(s).

The formula for Rank Correlation Partial is:

$$r_{xy.z} = \frac{r_{xy} - r_{xz} r_{yz}}{\sqrt{(1 - r_{xz}^2)(1 - r_{yz}^2)}}$$

where:

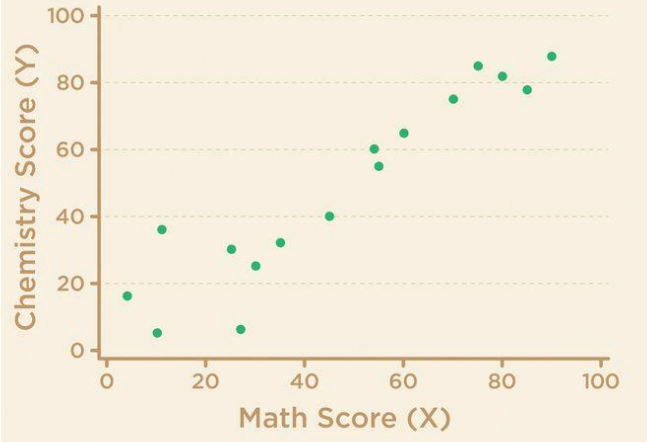
- $r_{xy.z}$ is the rank correlation partial between variables x and y, controlling for z
- r_{xy} is the Spearman rank correlation between x and y
- r_{xz} is the Spearman rank correlation between x and z
- r_{yz} is the Spearman rank correlation between y and z

Rank Correlation Partial is useful when you want to:

- Measure the association between two variables while removing the linear effect of one or more other variables
- Determine if the relationship between two variables is direct or mediated by other variables
- Analyze the strength of association between ranked variables after controlling for the effect of other ranked variables

Spearman rank correlation:

A statistical tool that assesses the strength and direction of a relationship between two quantitative, ranked variables.



Spearman's rank correlation is a flexible statistical tool that assesses the strength and direction of the relationship between two quantitative, ranked variables.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

ρ = Spearman's rank correlation coefficient
d_i = difference between the two ranks of each observation in the dataset
n = number of observations

Spearman correlation test example

Let us suppose a university lecturer is interested in comparing 2 sets of exam scores from their class of 16 students to see if the 2 sets of exam results are correlated. General chemistry and mathematics exams were both scored between 0 and 100.

Step one is to calculate the ranks for each student's exam scores, the differences between the ranks and the differences squared, tabulated as follows (Table 1):

Table 1: Student's math and chemistry exam scores and values required to calculate Spearman's correlation.

Student	Math Score (X)	Rank (X)	Chemistry Score (Y)	Rank (Y)	Difference (d)	d^2
1	85	2	78	4	2	4
2	80	3	82	3	0	0
3	90	1	88	1	0	0
4	70	5	75	5	0	0
5	60	6	65	6	0	0
6	75	4	85	2	2	4

7	54	8	60	7	1	1
8	55	7	55	8	1	1
9	45	9	40	9	0	0
10	35	10	32	11	1	1
11	25	13	30	12	1	1
12	11	14	36	10	4	16
13	4	16	16	14	0	0
14	10	15	5	16	1	1
15	27	12	6	15	3	9
16	30	11	25	13	2	4

It is also always useful to visualize our data using a scatter plot to check the assumption of a monotonic relationship is met (Figure 2):

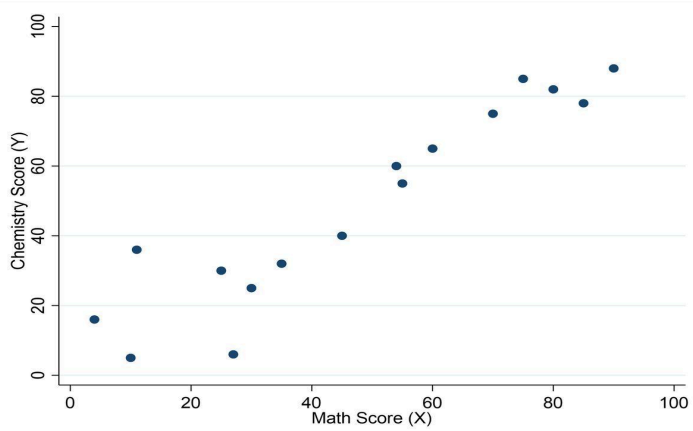


Figure 2: A scatter plot of the exam result data. *Credit: Elliot McClenaghan.*

Step two is to sum the squared differences (d^2), in this case they sum to 42, and calculate the Spearman's rank correlation coefficient using the formula:

$$= 1 - \frac{6 \text{ x } 42}{16(256 - 1)}$$

$$= 1 - 0.06176$$

$$= 0.938$$

Step three is to conduct a hypothesis test for the relationship between the two variables using Spearman's ρ.

Our hypotheses for this test are as follows:

- Null hypothesis (H0) is that the two variables are independent, there is no correlation.
- Alternative hypothesis (H1) is that as one variable increases the other also increases.

Next, we calculate the test statistic (t). In the case of a sample size greater than 10, as in our example, the ρ can approximate to a Normal distribution and the test statistic can be calculated as:

$$t = \rho\sqrt{n-1}$$

$$= 0.938\sqrt{16-1}$$

$$= 3.63$$

The test statistic can then be used to look up the one-sided or two-sided p-value (the latter being the probability of observing data as extreme or more extreme than the observed results, assuming the null hypothesis is true) using a t-distribution table. The t-distribution is a probability distribution that gives the probabilities of the occurrence of different outcomes for an experiment and is commonly used in statistical hypothesis testing.The two-sided p-value is usually more of interest as it gives both directions of the effect and better represents the hypotheses of our test. In this case, we find a two-sided p-value of $p < 0.001$. In practice this can be done, as can the whole calculation of the Spearman's ρ and hypothesis test, using statistical software. Our p-value indicates strong evidence against the null hypothesis and that there is evidence of a correlation between the math scores and chemistry scores.

Multiple correlation:

The multiple correlation coefficient, denoted as R, ranges from 0 to 1, with 0 indicating no linear relationship and 1 indicating a perfect linear relationship. The formula for calculating the multiple correlation coefficient is:

$$R = \sqrt{\frac{\sum_{i=1}^{n}(y_i-\bar{y})^2-\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

where:

- y_i is the observed value of the dependent variable for the ith observation
- $\bar{y}$ is the mean of the dependent variable
- $\hat{y}_i$ is the predicted value of the dependent variable for the ith observation based on the regression equation

The multiple correlation coefficient is closely related to the coefficient of determination, denoted as R^2, which represents the proportion of the variance in the dependent variable

that is explained by the independent variables in the regression model. The coefficient of determination is calculated as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

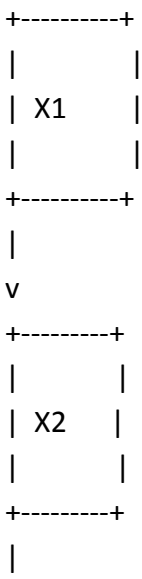
Key points about multiple correlation are:

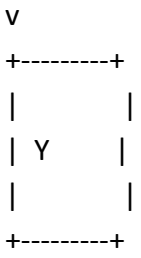
1. It measures the strength of the linear relationship between a dependent variable and a combination of two or more independent variables.
2. It is used in multiple regression analysis to assess the overall fit of the regression model.
3. The multiple correlation coefficient ranges from 0 to 1, with higher values indicating a stronger linear relationship.
4. The coefficient of determination (R^2) represents the proportion of the variance in the dependent variable that is explained by the independent variables in the regression model.
5. Multiple correlation is a useful tool for predicting the value of the dependent variable based on the values of the independent variables.

Multicollinearity:

Multicollinearity is a phenomenon in statistical analysis where two or more independent variables in a multiple regression model are highly correlated with each other. This situation can significantly affect the interpretation and reliability of the regression coefficients, making it challenging to determine the unique impact of each independent variable on the dependent variable.

A simple model diagram illustrating multicollinearity would look like this:





In this diagram, the independent variables X1 and X2 are highly correlated with each other, as indicated by the bidirectional arrow between them. This correlation can lead to multicollinearity in the regression model. The consequences of multicollinearity include:

1. Unstable and unreliable estimates of the regression coefficients
2. Large standard errors and wide confidence intervals for the coefficients
3. Difficulty in assessing the individual effects of the predictors on the dependent variable
4. Sensitive estimates to small changes in the data or model specification

To detect and address multicollinearity, several methods can be used, such as:

1. Calculating the variance inflation factor (VIF) for each independent variable
2. Examining the correlation matrix of the predictors
3. Performing principal component analysis (PCA) or ridge regression
4. Removing highly correlated variables from the model or combining them into a single variable

Definition:

- Multicollinearity occurs when independent variables in a regression model are correlated. This can be perfect (exact linear relationship) or imperfect (nearly exact linear relationship)

Effects on Regression Coefficients:

- Multicollinearity can lead to:
    - Broader confidence intervals for the regression coefficients.
    - Decreased precision of the estimated regression coefficients.
    - Increased uncertainty in statistical inferences

Detection Methods:

- **Variance Inflation Factor (VIF)**: Measures the inflation of variance in estimated regression coefficients due to linear relationships between predictor variables. A VIF value above 1 indicates multicollinearity, with higher values indicating stronger correlations

- **Correlation Matrix**: Visualizes the degree of correlation between independent variables. High correlation values indicate multicollinearity
- **Scatter Plot**: Plots independent variables against each other to identify linear relationships

Types of Multicollinearity:

- **Structural Multicollinearity**:

Arises from the model itself, such as when a variable is created by combining other variables. For example, if a model includes both income and expenses, and income is equal to expenses plus savings, there is a structural multicollinearity

- **Data Multicollinearity**:

  Present in the data itself, often due to observational studies where variables are naturally correlated. For example, in a dataset where income, expenses, and savings are all measured, there is data multicollinearity

Mitigation Strategies:

- **Include All Collinear Variables**: Excluding any collinear variable can worsen the coefficient estimates and lead to biased standard errors
- **Transform Variables**: Reduce the correlation by transforming variables, such as taking logarithms or square roots
- **Use Diverse Indicators**: In investment analysis, using different technical indicators can help mitigate multicollinearity by providing independent analytical perspectives

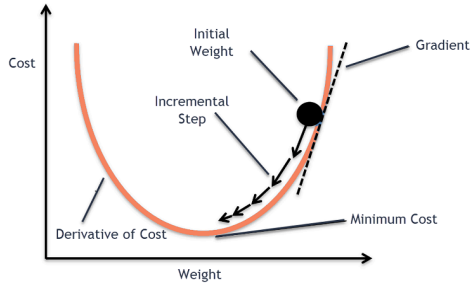
Impact on Investment Analysis:

- Multicollinearity can mislead assumptions about investments by creating similar trends in technical indicators. Using diverse indicators can enhance the robustness of investment analyses

Gradient descent methods:

Gradient descent is an optimization algorithm used in machine learning to minimize the cost function by iteratively adjusting parameters in the direction of the negative gradient, aiming to find the optimal set of parameters.

The cost function represents the discrepancy between the predicted output of the model and the actual output. Gradient descent aims to find the parameters that minimize this discrepancy and improve the model's performance.



The algorithm operates by calculating the gradient of the cost function, which indicates the direction and magnitude of the steepest ascent. However, since the objective is to minimize the cost function, gradient descent moves in the opposite direction of the gradient, known as the negative gradient direction.

By iteratively updating the model's parameters in the negative gradient direction, gradient descent gradually converges towards the optimal set of parameters that yields the lowest cost. The learning rate, a hyperparameter, determines the step size taken in each iteration, influencing the speed and stability of convergence

Example:

The best way is to observe the ground and find where the land descends. From that position, step in the descending direction and iterate this process until we reach the lowest point.

Finding the lowest point in a hilly landscape

Gradient descent is an iterative optimization algorithm for finding the local minimum of a function.

To find the local minimum of a function using gradient descent, we must take steps proportional to the negative of the gradient (move away from the gradient) of the function at the current point. If we take steps proportional to the positive of the gradient (moving towards the gradient), we will approach a local maximum of the function, and the procedure is called **Gradient Ascent.**

Gradient descent algorithm

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$\text{(for } j = 1 \text{ and } j = 0)$$
$$\}$$

How Does Gradient Descent Work?

1. The algorithm optimizes to minimize the model's cost function.
2. The cost function measures how well the model fits the training data and defines the difference between the predicted and actual values.
3. The cost function's gradient is the derivative with respect to the model's parameters and points in the direction of the steepest ascent.
4. The algorithm starts with an initial set of parameters and updates them in small steps to minimize the cost function.
5. In each iteration of the algorithm, it computes the gradient of the cost function with respect to each parameter.
6. The gradient tells us the direction of the steepest ascent, and by moving in the opposite direction, we can find the direction of the steepest descent.
7. The learning rate controls the step size, which determines how quickly the algorithm moves towards the minimum.
8. The process is repeated until the cost function converges to a minimum. Therefore indicating that the model has reached the optimal set of parameters.
9. Different variations of gradient descent include batch gradient descent, stochastic gradient descent, and mini-batch gradient descent, each with advantages and limitations.
10. Efficient implementation of gradient descent is essential for performing well in machine learning tasks. The choice of the learning rate and the number of iterations can significantly impact the algorithm's performance.

Types of Gradient Descent Algorithm

Batch Gradient Descent

- Calculates the gradient using the entire training dataset
- Provides a stable and reliable update to the model parameters
- Can be computationally expensive for large datasets

Stochastic Gradient Descent (SGD)

- Calculates the gradient using a single training example
- Provides noisy but frequent updates to the model parameters

- Can be faster than batch gradient descent for large datasets

Mini-Batch Gradient Descent

- Calculates the gradient using a small subset of the training dataset
- Provides a balance between the stability of batch gradient descent and the speed of SGD
- Commonly used in deep learning due to its efficiency

Momentum-Based Gradient Descent

- Adds a momentum term to the update rule to accelerate convergence
- Helps overcome the zig-zag pattern and slow convergence of standard gradient descent
- Examples include Nesterov Accelerated Gradient and RMSProp

Adaptive Gradient Descent

- Adjusts the learning rate for each parameter individually
- Examples include AdaGrad, AdaDelta, and Adam
- Can be more robust to noisy gradients and sparse data

Advantages and Disadvantages

Advantages

Easy to use: It's like rolling the marble yourself – no fancy tools needed, you just gotta push it in the right direction.

Fast updates: Each push (iteration) is quick, you don't have to spend a lot of time figuring out how hard to push.

Memory efficient: You don't need a big backpack to carry around extra information, just the marble and your knowledge of the hill.

Usually finds a good spot: Most of the time, the marble will end up in a pretty flat area, even if it's not the absolute flattest (global minimum).

Disadvantages

Slow for giant hills (large datasets): If the hill is enormous, pushing the marble all the way down each time can be super slow. There are better ways to roll for these giants.

Can get stuck in shallow dips (local minima): The hill might have many dips, and the marble could get stuck in one that isn't the absolute lowest. It depends on where you start pushing it from.

Finding the perfect push (learning rate): You need to figure out how har to push the marble (learning rate). If you push too weakly, it'll take forever to get anywhere. Push too hard, and it might roll right past the flat spot.

Challenges of Gradient Descent Algorithm

1. Local Optima: Gradient descent can converge to local optima instead of the global optimum, especially if the cost function has multiple peaks and valleys.
2. Learning Rate Selection: The choice of learning rate can significantly impact the performance of gradient descent. If the learning rate is too high, the algorithm may

overshoot the minimum, and if it is too low, the algorithm may take too long to converge.

3. Overfitting: Gradient descent can overfit the training data if the model is too complex or the learning rate is too high. This can lead to poor generalization performance on new data.
4. Convergence Rate: The convergence rate of gradient descent can be slow for large datasets or high-dimensional spaces, making the algorithm computationally expensive.
5. Saddle Points: In high-dimensional spaces, saddle points can cause the gradient of the cost function to get stuck in a plateau, preventing gradient descent from converging to a minimum.

Newton method:

Newton's method is a **method for finding the root of a function, rather than its maxima or minima**.

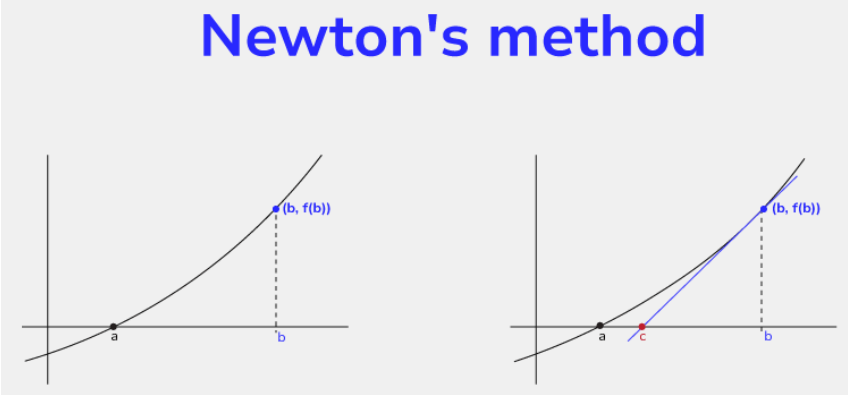
This means that, if the problem satisfies the constraints of Newton's method, we can find x for which f(x)=0 . Not f'(x)=0, as was the case for gradient descent.

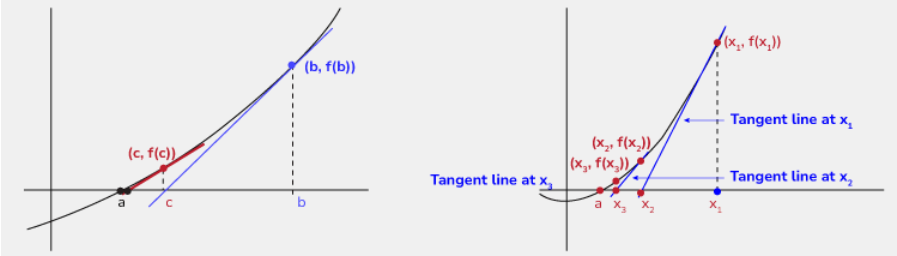
We, therefore, apply Newton's method on the derivative f'(x) of the cost function, not on the cost function itself. This is important because Newton's method requires the analytical form of the derivative of any input function we use, as we'll see shortly. Therefore, **this means that the cost function we use must be differentiable twice**, not just once, as was the case for gradient descent.

Let's define f(x) as the cost function of a model on which we apply Newton's method. The terms f'(x) and f''(x) thus indicate, respectively, the first and second-order derivatives of f(x). If we start from a point x_n that's sufficiently close to the minimum of f, we can then get a better approximation by computing this formula:

$$x_{n+1} = x_n + \frac{f'(x_n)}{f''(x_n)}$$

The term $\frac{f'(x)}{f''(x)}$ here, indicates that we're approximating the function f'(x) with a linear model, in proximity of x_n





- At each iteration, the method updates the current approximation X_n by subtracting the ratio of the gradient $f'(x_n)$and the curvature $f''(x_n)$
- If the curvature is positive (convex function), this ratio decreases the value of x, bringing it closer to the minimum.
- If the curvature is negative (concave function), this ratio increases the value of x, bringing it closer to the maximum.

Interior point methods:

Interior-point methods for linear and (convex) quadratic programming display several features which make them particularly attractive for very large scale optimization. They have an impressive low-degree polynomial worst case complexity.

An interior-point method for optimization relies on three basic ideas:

1. Logarithmic barrier functions are used to “replace” the inequality constraints
2. Duality theory is applied to barrier subproblems to derive the first-order optimality conditions which take the form of a system of nonlinear equations, and
3. Newton’s method is employed to solve this system of nonlinear equations.

Nonlinear Constrained Optimization:

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ s.t. \quad & c(x) = 0 \\ & x \geq 0 \end{aligned}$$

- Linear, quadratic, or general nonlinear objective and constraints
- Convex optimization, local solution possible for non convex problems
- Convert maximization by minimizing negative of the objective
- Convert general inequalities to simple inequalities with slack variables

Slack Variables:

Slack variables are additional variables used in optimization that convert an inequality constraint into an equality constraint, allowing the problem to be solved with standard methods, such as Interior Point Methods.

Slack variables are typically denoted by the letter *s* and the value is always positive. Slack variables convert a constraint of the form "greater than or equal to" into an equation.

For example, if the constraint is 2x+y≥4 then the constraint can be rewritten as 2x+y−4≥0 or with the slack variable (*s*), s=2x+y−4, where s≥0.

Slack variables are defined to transform an inequality expression into an equality expression with the added slack variable. The slack variable is defined by setting a lower bound of zero (≥*0*).

$$\begin{array}{ll} \min\limits_{x\in R^n} & f(x) \\ s.t. & g(x) \geq b \\ & h(x) = 0 \end{array} \quad \Rightarrow \quad \begin{array}{ll} \min\limits_{x\in R^n} & f(x) \\ s.t. & c(x) = 0 \\ & x \geq 0 \end{array}$$

Complete worksheet on slack variables:

$$\begin{array}{ll} \min\limits_{x} & x_1x_4(x_1+x_2+x_3)+x_3 \\ s.t. & x_1x_2x_3x_4 \geq 25 \\ & x_1^2+x_2^2+x_3^2+x_4^2 = 40 \end{array}$$

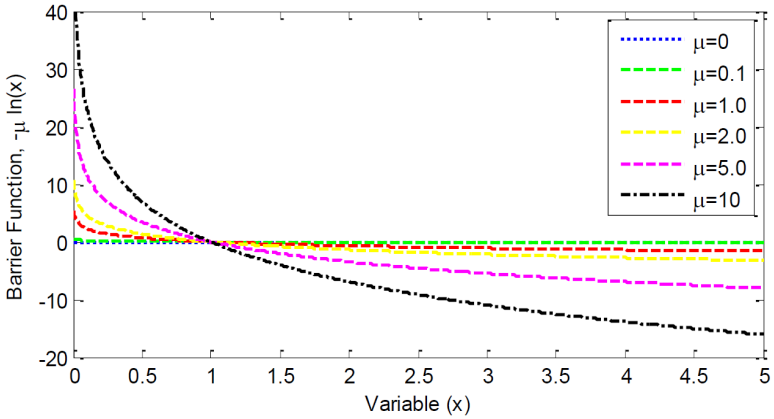
Barrier function:

Once the given problem is converted to the standard form we will now try to remove the inequality constraint x ⩾ 0 using the barrier term. We will introduce natural log barrier term for inequality constraints as shown in Equation set 3

$$\min f(x) - \mu \textstyle\sum_i ln(x_i)$$
$$s.t.\, c(x) = 0$$

The point l n (xi) is not defined for xi ⩽ 0

Our goal is to solve this barrier problem for a given ′ μ ′. If ′ μ ′ is decreased at the right rate and the approximate solutions are good enough then this method will converge to an optimal solution of the nonlinear optimization problem. We want to only search in the interior feasible space.



KKT Condition for Barrier Problem

KKT conditions for the barrier problem of the form:

$\min f(x) - \mu \sum_{i=1}^{n} ln(x_i) \Rightarrow \nabla f(x) + \nabla c(x)\lambda - \mu \sum_{i=1}^{n} \frac{1}{x_i}$

$s.t.\, c(x) = 0 \;\; c(x) = 0$

We can define now define $z_i = \frac{\mu}{x_i}$ and solve the modified version of the KKT conditions:

$\nabla f(x) + \nabla c(x)\lambda - z = 0$

$c(x) = 0$

$XZe - \mu e = 0\,.$

The matrix $'e'$ is simply a column vector of ones i.e. $e = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ to satisfy the rules of matrix operations.

KKT solution with Newton-Raphson method:

$\nabla f(x) + \nabla c(x)\lambda - z = 0$

$c(x) = 0$

$XZe - \mu e = 0$

$$\Rightarrow \begin{bmatrix} W_k & \nabla c(x_k) & -I \\ \nabla c(x_k)^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \\ d_k^z \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + \nabla c(x_k)\lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu_j e \end{pmatrix}$$

Here, d_k^z , d_k^x and d_k^λ are the search directions for the x, z, λ and W_k is the second gradient of the Lagrangian.

Thus, $W_k = \nabla_{xx}{}^2 L(x_k, \lambda_k, z_k) = \nabla_{xx}{}^2 (f(x_k) + c(x_k)^T \lambda_k - z_k)$ and $Z_k = \begin{bmatrix} z_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & z_n \end{bmatrix}$, $X_k = \begin{bmatrix} x_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & x_n \end{bmatrix}$

The above system can be arranged into symmetric linear system as follows:

$$\begin{bmatrix} W_k + \Sigma_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + \nabla c(x_k)\lambda_k \\ c(x_k) \end{pmatrix} \text{ where } \Sigma_k = X_k^{-1} Z_k \ldots\ldots\ldots Equation(*)$$

We can then solve for d_k^z after finding the linear solution to d_k^x and d_k^λ with the following form of the explicit solution:

$d_k^z = \mu_k X_k^{-1} e - z_k - \Sigma_k d_k^x \ldots\ldots\ldots Equation(**)$

Step size (α):

Decreasing the merit function: Merit function is defined as the sum of the objective function and the absolute values of the constraint violations multiplied by a scalar quantity

i.e. $merit = f(x) + v \sum |c(x)|$

so that it determines if it's a better step or not.

Filter methods use a combination of objective function f(x) and constraint violations |c(x)| to determine whether to accept the step size or not.

Once the step size has been determined we can find out the values of

$x_{k+1}\,, z_{k+1}\,, \lambda_{k+1}$

$$x_{k+1} = x_k + \alpha d_k^x$$

$$z_{k+1} = z_k + \alpha d_k^z$$

$$\lambda_{k+1} = \lambda_k + \alpha d_k^\lambda$$

Convergence Criteria

Convergence criteria can be defined when KKT conditions are satisfied within a tolerance:

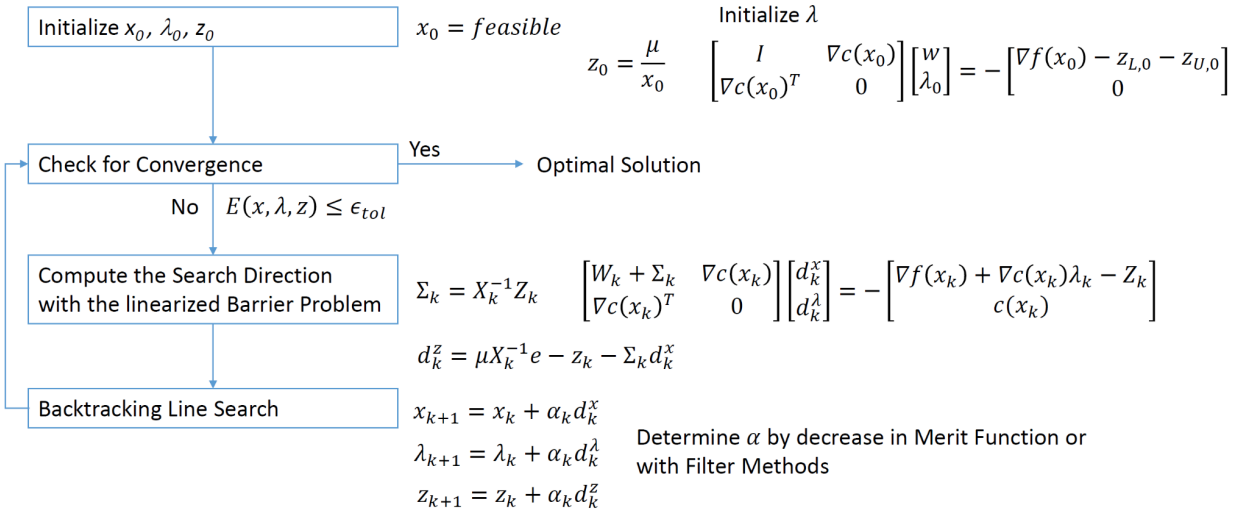
$$\max |\nabla f(x) + \nabla c(x)\lambda - z| \leqslant \epsilon_{tol}$$

$$\max |c(x)| \leq \epsilon_{tol}$$

$$\max |XZe - \mu e| \leq \epsilon_{tol}$$

Algorithm:

- Choose a feasible guess value of x_0 and initialize λ and Z_0 by solving the above described equation.
- Once we have initial values then check for convergence, if it is within the 'tolerance limit' then we have reached the optimal solution. If not then, move to the next step.
- Compute the search direction using the Equation(*) and Equation(**) as mentioned above.
- Perform backtracking line search by progressively decreasing the value of α using the merit function of Filter methods.
- Check for convergence again and repeat steps if the solution is not within the tolerance limit.



---

active set, proximity methods, accelerated gradient methods, coordinate descent, cutting planes, stochastic gradient descent. Discriminant analysis, Principal component analysis, Factor analysis, k means.

UNIT-III

UNIT-IV

UNIT-V

---

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("data.csv")
print(data,"n")

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
   for i, val in enumerate(t):
      if val == "Yes":
         specific_hypothesis = c[i].copy()
         break

   for i, val in enumerate(c):
      if t[i] == "Yes":
         for x in range(len(specific_hypothesis)):
```

```
            if val[x] != specific_hypothesis[x]:
               specific_hypothesis[x] = '?'
            else:
               pass

   return specific_hypothesis

#obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))

—----------------------------------------------------------
#download 1.csv
import csv
a = []
with open('1.csv', 'r') as csvfile:
         for row in csv.reader(csvfile):
         a.append(row)
         print(a)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1

hypothesis = ['0']*num_attribute
print("\n The initial hypothesis is : \n", hypothesis)

for i in range(0, len(a)):
         if a[i][num_attribute] == 'yes':
         for j in range(0, num_attribute):
         if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
         hypothesis[j] = a[i][j]
         else:
         hypothesis[j] = '?'

         print("\n The hypothesis for the training instance {} is: \n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instances is: \n",hypothesis )
```

Machine Learning

2. For a given set of training data examples stored in a csv file, implement and demonstrate the candidate-elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

2.csv

```
import numpy as np
import pandas as pd
data = pd.read_csv('2.csv')
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
        specific_h = concepts[0].copy()
        print("initialization of specific_h \n",specific_h)
        general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
        print("initialization of general_h \n", general_h)
        for i, h in enumerate(concepts):
        if target[i] == "yes":
        print("If instance is Positive ")
        for x in range(len(specific_h)):
        if h[x]!= specific_h[x]:
                specific_h[x] ='?'
                general_h[x][x] ='?'

        if target[i] == "no":
        print("If instance is Negative ")
        for x in range(len(specific_h)):
        if h[x]!= specific_h[x]:
                general_h[x][x] = specific_h[x]
        else:
                general_h[x][x] = '?'

        print(" step {}".format(i+1))
        print(specific_h)
        print(general_h)
        print("\n")
        print("\n")
        indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
        for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
        return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

---

3. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

DataSet Pgm3.csv

```
import numpy as np
import math
import csv

def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute

def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
```

```
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict

def entropy(S):
    items = np.unique(S)

    if items.size == 1:
        return 0

    counts = np.zeros((items.shape[0], 1))
    sums = 0

    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums

def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))
```

```
    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy / iv

def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    gains = np.zeros((data.shape[1] - 1, 1))

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)

    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)

    items, dict = subtables(data, split, delete=True)

    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node
```

```
def empty(size):
   s = ""
   for x in range(size):
      s += "   "
   return s

def print_tree(node, level):
   if node.answer != "":
      print(empty(level), node.answer)
      return
   print(empty(level), node.attribute)
   for value, n in node.children:
      print(empty(level + 1), value)
      print_tree(n, level + 2)

metadata, traindata = read_data("pgm3.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

```
Outlook
   Overcast
     b'Yes'
   Rainy
     Windy
       b'False'
         b'Yes'
       b'True'
         b'No'
   Sunny
     Humidity
       b'High'
         b'No'
       b'Normal'
         b'Yes'
```

—---------- ---------- ---------- ----------

4. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)     # X = (hours sleeping, hours studying)
y = np.array(([92], [86], [89]), dtype=float)           # y = score on test

# scale units
X = X/np.amax(X, axis=0)       # maximum of X array
y = y/100                      # max test score is 100

class Neural_Network(object):
   def __init__(self):
                        # Parameters
      self.inputSize = 2
      self.outputSize = 1
      self.hiddenSize = 3
                        # Weights
      self.W1 = np.random.randn(self.inputSize, self.hiddenSize)
                        # (3x2) weight matrix from input to hidden layer
      self.W2 = np.random.randn(self.hiddenSize, self.outputSize)
                        # (3x1) weight matrix from hidden to output layer

   def forward(self, X):
                        #forward propagation through our network
      self.z = np.dot(X, self.W1)             # dot product of X (input) and first set of 3x2 weights
      self.z2 = self.sigmoid(self.z)          # activation function
      self.z3 = np.dot(self.z2, self.W2)       # dot product of hidden layer (z2) and second set of 3x1
weights
      o = self.sigmoid(self.z3)               # final activation function
      return o

   def sigmoid(self, s):
      return 1/(1+np.exp(-s))     # activation function

   def sigmoidPrime(self, s):
      return s * (1 - s)         # derivative of sigmoid
```

```
  def backward(self, X, y, o):
                        # backward propgate through the network
    self.o_error = y - o       # error in output
    self.o_delta = self.o_error*self.sigmoidPrime(o) # applying derivative of sigmoid to
    self.z2_error = self.o_delta.dot(self.W2.T)
                # z2 error: how much our hidden layer weights contributed to output error
    self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2)
                 # applying derivative of sigmoid to z2 error
    self.W1 += X.T.dot(self.z2_delta)      # adjusting first set (input --> hidden) weights
    self.W2 += self.z2.T.dot(self.o_delta)  # adjusting second set (hidden --> output) weights

  def train (self, X, y):
    o = self.forward(X)
    self.backward(X, y, o)
—--------------------------------
NN = Neural_Network()
for i in range(1000): # trains the NN 1,000 times
  print ("\nInput: \n" + str(X))
  print ("\nActual Output: \n" + str(y))
  print ("\nPredicted Output: \n" + str(NN.forward(X)))
  print ("\nLoss: \n" + str(np.mean(np.square(y - NN.forward(X))))    # mean sum squared loss)
  NN.train(X, y)
—----------
```

SAMPLE OUTPUT:

```
Input:
[[0.66666667 1.       ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
[[0.47212874]
 [0.42728946]
 [0.40891365]]
```

Loss:
0.20642371917499927

Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]

Actual Output:
[[0.92]
[0.86]
[0.89]]

Predicted Output:
[[0.55398066]
[0.49831918]
[0.50254468]]

Loss:
0.13830159742519685
.
.
.
—----------

5. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Dataset prog5.csv

```
# import necessary libarities
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# load data from CSV
data = pd.read_csv('prog5.csv')
print("THe first 5 values of data is :\n",data.head())
```

Output:

THe first 5 values of data is :

	Outlook	Temperature	Humidity	Windy	PlayTennis
0	**Sunny**	**Hot**	**High**	**False**	**No**
1	**Sunny**	**Hot**	**High**	**True**	**No**

2 Overcast Hot High False Yes
3 Rainy Mild High False Yes
4 Rainy Cool Normal False Yes
—-------

```
# obtain Train data and Train output
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is\n",X.head())
```

Output:
The First 5 values of train data is
Outlook Temperature Humidity Windy
0 Sunny Hot High False
1 Sunny Hot High True
2 Overcast Hot High False
3 Rainy Mild High False
4 Rainy Cool Normal False
—----------

```
y = data.iloc[:,-1]
print("\nThe first 5 values of Train output is\n",y.head())
```

output:
The first 5 values of Train output is
0 No
1 No
2 Yes
3 Yes
4 Yes
Name: PlayTennis, dtype: object
—--------

```
# Convert then in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
```

```
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the Train data is :\n",X.head())
```

Output:

Now the Train data is :

	Outlook	Temperature	Humidity	Windy
0	2	1	0	0
1	2	1	0	1
2	0	1	0	0
3	1	2	0	0
4	1	0	1	0

—---------

```
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

Output:

Now the Train output is

[0 0 1 1 1 0 1 0 1 1 1 1 1 0]

—---------

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
classifier = GaussianNB()
classifier.fit(X_train,y_train)
from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

Output:

Accuracy is: 0.6666666666666666

---

6. Assuming a set of documents that need to be classified, use the Naive Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set

Dataset Prog6.csv

```
import pandas as pd
msg = pd.read_csv('prog6.csv', names=['message', 'label'])
print("Total Instances of Dataset: ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
```

Output:

Total Instances of Dataset: 18

```
X = msg.message
y = msg.labelnum
from ś_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
from sklearn.feature_extraction.text import CountVectorizer
count_v = CountVectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)
df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names_out()) #
get_feature_names()
print(df[0:5])
```

Output:

```
   am  amazing  an  and  awesome  bad  can  deal  do  enemy  ...  this  tired  \
0   1        0   0    1        0    0    0     0   0      0  ...     1      1
1   0        0   1    0        1    0    0     0   0      0  ...     1      0
2   0        0   0    0        0    0    0     0   0      0  ...     1      0
3   0        0   0    0        0    0    0     0   0      1  ...     0      0
4   0        0   0    0        0    0    0     0   0      0  ...     0      0

   to  today  tomorrow  we  went  what  will  with
0   0      0         0   0     0     0     0     0
1   0      0         0   0     0     0     0     0
2   0      0         0   0     0     0     0     0
3   1      1         0   0     1     0     0     0
4   0      0         1   1     0     0     1     0

[5 rows x 45 columns]
```

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
pred = clf.predict(Xtest_dm)

for doc, p in zip(Xtrain, pred):
        p = 'pos' if p == 1 else 'neg'
        print("%s -> %s" % (doc, p))
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
print('Accuracy Metrics: \n')
print('Accuracy: ', accuracy_score(ytest, pred))
print('Recall: ', recall_score(ytest, pred))
print('Precision: ', precision_score(ytest, pred))
print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```

Output:

I am sick and tired of this place -> pos
This is an awesome place -> neg
I love this sandwich -> pos
I went to my enemy's house today -> neg
We will have good fun tomorrow -> neg
Accuracy Metrics:

Accuracy: 0.6
Recall: 0.5
Precision: 1.0
Confusion Matrix:
[[1 0]
[2 2]]

---

7. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set. You can use Java/Python ML library classes/API

#dataset Prog7.csv
#conda update jupyter
#To install API
#On the Jupyter notebook interface, go to “New” > “Terminal”
#conda install -c ankurankan pgmpy

```
from pgmpy.models import BayesianModel
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
data = pd.read_csv("pgm7.csv")
heart_disease = pd.DataFrame(data)
```

```
model=BayesianModel([
('age','Lifestyle'),
('Gender','Lifestyle'),
('Family','heartdisease'),
('diet','cholestrol'),
('Lifestyle','diet'),
('cholestrol','heartdisease'),
('diet','cholestrol')
])

from pgmpy.estimators import MaximumLikelihoodEstimator
model.fit(data, estimator=MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)

print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')
print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')
print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
   'age':int(input('Enter age :')),
   'Gender':int(input('Enter Gender :')),
   'Family':int(input('Enter Family history :')),
   'diet':int(input('Enter diet :')),
   'Lifestyle':int(input('Enter Lifestyle :')),
   'cholestrol':int(input('Enter cholestrol :'))
         })
print(q)
```

---

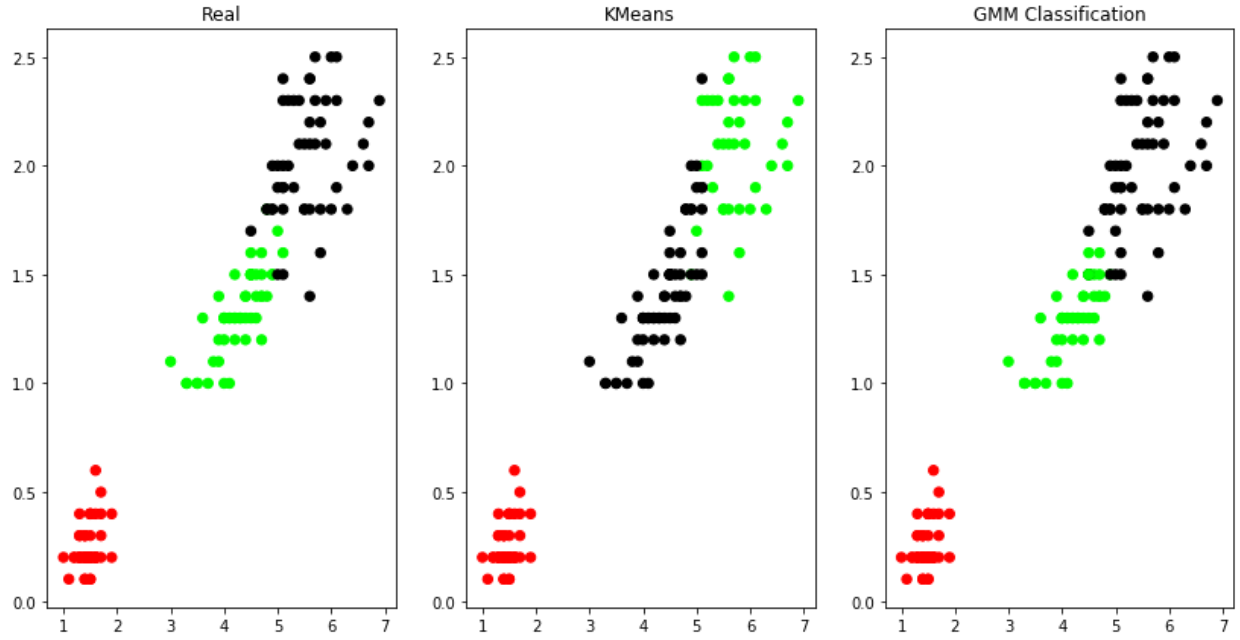
8. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
```

```
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset=load_iris()
# print(dataset)

X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
```

```
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

Output: #(following output will separate)
Text(0.5, 1.0, 'GMM Classification')



OR

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset=load_iris()
# print(dataset)

X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
```

```
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

---

9. Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.model_selection import train_test_split
import numpy as np

dataset=load_iris()
print(dataset)
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0
)

kn=KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train,y_train)

#output:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
        metric_params=None, n_jobs=None, n_neighbors=1, p=2,
        weights='uniform')

for i in range(len(X_test)):
   x=X_test[i]
   x_new=np.array([x])
   prediction=kn.predict(x_new)

print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,dataset["
target_names"][prediction])
print(kn.score(X_test,y_test))

TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
```

```
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

---

10. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

```
from math import ceil
import numpy as np
from scipy import linalg

def lowess(x, y, f, iterations):
   n = len(x)
```

```
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:, i]
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
                        A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x),
np.sum(weights * x * x)]])
            beta = linalg.solve(A, b)
            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta ** 2) ** 2

    return yest

import math
n = 100
x = np.linspace(0, 2 * math.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f =0.25
iterations=3
yest = lowess(x, y, f, iterations)

import matplotlib.pyplot as plt
plt.plot(x,y,"r.")
plt.plot(x,yest,"b-")
```

Output:

```
[<matplotlib.lines.Line2D at 0x1856f8b87a0>]
```

Machine Learning