



زانكۆی سه‌لاحه‌دین - هه‌ولێر
Salahaddin University-Erbil

On Some Mathematical Applications In Artificial intelligence

Research Project

Submitted to the department of (Mathematic) in partial fulfillment
of

the requirements for the degree of BSc. in (forth)

By:

Rekan Ramadan Salim

Supervised by:

Dr. Hogir Mohammed Yaseen

May– 2024

Certification of the Supervisors

I certify that this report was prepared under my supervision at the Department of Mathematics / College of Education / Salahaddin University-Erbil in partial fulfillment of the requirements for the degree of Bachelor of philosophy of Science in Mathematics.

Signature:



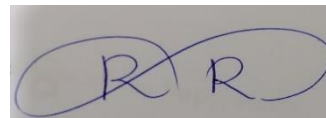
Supervisor: Dr. Hogir Mohammed Yaseen

Scientific grade: Lecturer

Date: 2 / 4 / 2024

In view of the available recommendations, I forward this report for debate by the examining committee.

Signature:



Name: Assistant Prof. **Dr. Rashad Rashid Haji**

Scientific grade: Professor

Chairman of the Mathematics Department

ACKNOWLEDGMENTS

In the Name of Allah, I must acknowledge my limitless thanks to Allah, the Ever-Thankful, for His helps and bless. I am totally sure that this work would have never become truth, without His guidance.

My deepest gratitude goes to my supervisor Dr. Hogir , who's worked hard with me from the beginning till the completion of the present research. A special thanks to Dr. Rashad the head mathematic Department for his continuous help during this study. I would like to take this opportunity to say warm thanks to all my friends, who have been so supportive along the way of doing my research, colleagues for their advice on various topics, and other people who are not mentioned here. I also would like to express my wholehearted thanks to my family for their generous support they provided me throughout my entire life and particularly through the process of pursuing the BSc. degree.

Abstract

In this work we study on some mathematical applications in Artificial intelligence. First, we study mathematical applications in computer programming and algorithms. We show how mathematical concepts applied thier, such as algebra, geometry, and calculus, can be used to solve complex problems and optimize algorithms in various programming languages, with a focus on C++. In addition, we show how programming languages use mathematical concepts in their structure. Moreover, we study mathematical applications in neighbor algorithm, quadratic function, mathematical logic , game theory and binary system. At the end, we study mathematical applications in simple linear regression, k-nearest neighborhood and newton's forward interpolation formula. Furthermore, we solve many examples that illustrate the applications.

Table of Contents

Acknowledgments	ii
Abstract	iii
Introduction	iv
Chapter one : Mathematical applications in computer programming	1
1.1History	2
1.2Neighbor Algorithms K-Nearest:	3
1.3Quadratic functio :	10
1.4Mathematical Logic in c++ 1:	12
1.5Game Theory :	13
1.6Binary System :	15
Chapter two Simple linear regression and Newton's forward interpolation	16
2.1 Simple Linear Regression :	17
2.2K-Nearest Neighborhood :	20
2.3Newton's Forward Interpolation Formula :	24
پوخته	28
References	27

Introduction

Artificial intelligence (AI) relies heavily on mathematics, which provides the theoretical framework for many AI methods. Calculus, linear algebra, probability theory, and statistics are only a few of the math subjects that are essential for comprehending and creating AI systems. There is a significant relationship between mathematics and computer science. Math is used by computer scientists in a variety of professional settings. Mathematics provides the theoretical foundation for several subfields of computer science, and computer scientists use specialized mathematical themes to solve specific computing challenges. In machine learning, calculus is used to optimize functions so that algorithms can learn from data and get better over time. For manipulating high-dimensional data, such text and images, as well as for carrying out operations in neural networks and crucial AI technology linear algebra plays a key role. In AI systems, probabilistic graphical models and Bayesian inference are two examples of how probability theory and statistics are used to explain uncertainty and make judgments. In general, math is the foundation of artificial intelligence (AI), enabling scientists and engineers to build intelligent systems.

In this work we study on some mathematical applications in artificial intelligence and computer science. This work consists of three chapters and is organized as follows. In chapter one we study mathematical applications in neighbor algorithm, quadratic function, mathematical logic, game theory and binary system. In chapter two we study the applications in simple linear regression and k-nearest neighborhood. Furthermore, we study the applications of newton's forward interpolation formula. In addition, we solve many examples that illustrate the applications.

Chapter one

Mathematical applications in computer programming

At the heart of AI, there exists a mathematical framework that underpins its every operation. AI algorithms, whether they are used for image recognition, speech synthesis, or autonomous driving, all rely on mathematical principles. It is through these principles that AI algorithms speak their unique language:

AI algorithms use mathematical formulas and equations to process data and make decisions. These formulas can be simple or incredibly complex, depending on the task at hand. For example, in machine learning, algorithms use mathematical functions to fit data to models and make predictions.

Discrete mathematics is the foundation of computer science, and it teaches the use of algorithms. Furthermore, mathematics gives the analytical skills required in computer science. Applying math in programming involves using mathematical concepts to solve problems efficiently. This includes algorithms that explore neighboring elements, quadratic functions for modeling, logical reasoning, game theory for strategic decision-making, and binary systems for representing data in computers.

1.1 History of Artificial Intelligence:

Artificial intelligence's historical evolution from the "Dark Ages" to knowledge based systems The field of artificial intelligence was established by three generations of scientists. The following lists some of the most significant occasions and figures from each generation:

The 'Dark Ages', or the birth of artificial intelligence (1943–1956). In 1943, Warren McCulloch and Walter Pitts delivered the first piece of work officially recognized as artificial intelligence (AI). McCulloch, who graduated from Columbia University with degrees in philosophy and medicine, was appointed director of the Basic Research Laboratory in the University of Illinois Department of Psychiatry. His studies on the central nervous system led to the development of the first significant contribution to artificial intelligence: a brain

neuron model In 1943, McCulloch and his colleague Walter Pitts, a young mathematician, presented a model of artificial neural networks in which every neuron was assumed to be in a binary state, meaning that it was either on or off. They proved that the Turing machine was actually equal to their neural network model. shown that a network of interconnected neurons could calculate any computable function. McCulloch and Pitts also demonstrated the learning capabilities of basic network architectures The neural network concept sparked research into simulating the brain in the lab, both theoretically and experimentally. But investigations made it abundantly evident that the binary model of neurons was incorrect. Actually, a Neurons are extremely non-linear devices and cannot be thought of as simple two state devices. The principles of neural computing and artificial intelligence (AI) were created by McCulloch, the second "founding father" of AI after Alan Turing. The field of ANN saw a resurgence in the late 1980s following a decrease in the 1970s. The great mathematician John von Neumann, who was born in Hungary, was the third person to found AI. He began teaching mathematical physics at Princeton University in 1930. He was Alan Turing's friend and coworker. Von Neumann was instrumental in the Manhattan Project, which produced the nuclear weapon, during World War II. Additionally, he joined the University of Pennsylvania's Electronic Numerical Integrator and Calculator (ENIAC) advisory board and contributed to the creation of the Electronic Discrete neural networks made artificially (ANN). The field of ANN saw a resurgence in the late 198 Variable Automatic Computer (EDVAC), a stored program computer, following a slump in the 1970s. The neural network model developed by McCulloch and Pitts had an impact on him. Von Neumann sponsored and encouraged two graduate students in the Princeton mathematics department, Marvin Minsky and Dean Edmonds, when they constructed the first neural network computer in 1951. Claude Shannon was another member of the first generation of researchers. In 1941, after earning his degree from Massachusetts Institute of Technology (MIT), he started working at Bell Telephone Laboratories. Shannon discussed the

possibilities of machine intelligence with Alan Turing. He noted that a normal chess game involved roughly 10120 potential moves in a paper he released in 1950 on chess-playing machines (Shannon, 1950). It would take 3×10^{106} years for the new von Neumann-type computer to make its initial move, even if it could analyze one move each microsecond. Shannon thus illustrated the necessity of employing heuristics when looking for a solution. Another AI creator, John McCarthy, attended Princeton University. He persuaded Claude Shannon and Martin Minsky to plan a summer party (Negnevitsky, 2011)

Mathematics plays a critical role in artificial intelligence. Artificial intelligence (AI) has evolved as a disruptive technology, changing many parts of our existence. Mathematics plays a fundamental part in the astounding achievements and capabilities of artificial intelligence. Mathematics provides a framework for AI systems to learn, reason, and make intelligent decisions. In this post, we will look at how mathematics is used in AI and how important it is. Mathematics serves as the foundation for AI algorithms and models, allowing machines to process, analyze, and understand massive volumes of data. Machine learning algorithms require concepts from linear algebra, calculus, probability theory, and statistics. These algorithms utilize mathematical equations and functions to recognize patterns, forecast outcomes, and categorize data. Linear algebra, for example, is crucial in the creation of neural networks. Potential applications and benefits of integrating mathematics with AI in domains like healthcare, finance, and robotics include:

- **Healthcare:** Mathematicians contribute to AI-powered medical imaging techniques, disease diagnosis models, and personalized treatment optimization algorithms, leading to improved patient outcomes and more efficient healthcare delivery.
- **Finance:** By leveraging mathematical models and AI techniques, mathematicians contribute to areas such as algorithmic trading, fraud detection, risk assessment, and portfolio optimization, enhancing financial decision-making and market efficiency.

- **Robotics:** Mathematicians play a crucial role in developing algorithms for robot perception, motion planning, and control, enabling robots to navigate complex environments, perform precise tasks, and effectively collaborate with humans.

Mathematics has been at the core of AI since its inception, with mathematicians playing a pivotal role in shaping the field. They have made substantial advancements in fields like linear algebra, optimization theory, and deep learning. However, challenges persist, and applied mathematicians have a unique opportunity to contribute to ongoing advancements in AI.

To truly appreciate the synergy between AI and mathematics, we need to recognize that mathematics is the language that enables AI to operate efficiently and effectively. Here are a few key mathematical principles that underpin AI:

Statistics: Statistics is the science of collecting, analyzing, and interpreting data. In the world of AI, statistical methods are crucial for understanding uncertainty, estimating probabilities, and making data-driven decisions. Techniques like regression analysis, classification, and hypothesis testing are statistical tools used to build AI models and assess their accuracy.

Linear Algebra: Linear algebra deals with vector spaces and linear equations. It may sound abstract, but in AI, it's a fundamental tool for representing and processing data. Matrices and tensors are used to perform operations in neural networks, image processing, and data transformations.

Calculus: Calculus is the mathematical framework for understanding how things change. It plays a critical role in optimization tasks, which are central to training machine learning models. Gradient descent, a calculus-based algorithm, is the driving force behind adjusting model parameters to minimize errors and improve predictions.

Artificial intelligence (AI) has evolved as a disruptive technology, changing many parts of our existence. Mathematics plays a fundamental part in the astounding achievements and capabilities of artificial intelligence. Mathematics provides a

framework for AI systems to learn, reason, and make intelligent decisions. In this post, we will look at how mathematics is used in AI and how important it is. The connection between AI and mathematics is not just theoretical; it's the practical foundation on which AI systems are built. The algorithms used in AI are essentially mathematical formulas, and a deep understanding of these mathematical principles is the key to unlocking AI's potential. As we move forward in this exploration of the mathematical underpinnings of AI, we'll delve deeper into how these principles are put into practice in various AI applications and models.

1.2 Neighbor Algorithms K-Nearest:

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It is one of the popular and simplest classification and regression classifiers used in machine learning today.

A. the crisp K-NN Algorithm

Let $W=(x_1,x_2,\dots,x_n)$ be the set of n labeled samples . the algorithm is as follows

BEGIN

Input y, of unknown classification

Set k , $1 \leq k \leq n$

Initialize i=1

DO UNTIL (k-nearest neighbor found)

Compute distance from y to x_i

IF ($i \leq k$) THEN

Include x_i in the set of K-nearest neighbors

ELSE IF (x_i is closer to y than any previous nearest neighbor) then Delete farthest in the set of k-nearest neighbors.

END IF

Increment i (Keller, 1985)

C++ software shows how the k-nearest neighbors (KNN) algorithm for classification can be implemented simply. A dataset of objects with characteristics (size and color) and associated labels (such as "apple" or "orange") is included in

the software. The objective is to categorize a new object in the dataset using its features and the labels of its k-nearest neighbors.

Example:1.1:

```
#include<iostream>
#include<cmath>
Using namespace std;
// define a simple representation of subjects with features ( e.g.. size and color)
Struct Object{ Double size; String
color;
};
// define a dataset of objects with corresponding labels
Struct DatasetElement{
Object object;
String label;
}'
DatasetElement dataset[] = {
{{3.0,"red"},"apple"},
{{4.0,"orange"},"oereng"}, {{2.5,"green"},"grape"},
{{5.0,"orange"},"orange"}, {{3.5,"red"},"apple"},
// additional objects and labels can be added
};
// function to calculate the Euclidean distance between two objects
Doule calculateDistance(const Object& obj1, const Object& obj2){
Double sizeDifference = obj1.size – obj2.size;
Double colorDifertence = (obj1.color==obj2.color) 0.0 : 1.0;// assuming color
dissimilarity as 1.0
Return sqrt ( pow(sizeDiference, 2)+ pow(colorDiference, 2));
// k-nearest neighbors classification
String classifyObject (const Object& newObject , int k){
// calculate distance between the new objects and all objects in the dataset
Pair<double , string > distance [sizeof(datset) / sizeof(dataset[0])]; //
pair : (distance label )
For( int i=0 i<=sioeof(dataset) / sizeof(dataset[0]) ; i++){
Double distance = calculateDistance(newObject, dataset[i].object );
Distance[i]={distance, dataset[i].label};}
// implement a simple version of the sort function (select sort)
For(int i=0; i<=sizeof(dataset) / sizeof(dataset[0]);i++){ int minIndex = i;
For(int j=i+1;j<sizeof(dataset) /sizeof(dataset[0];++j){
If(distances[j].first < distances[minIndex].first){ minIndex=j;
}}
Swap(distances[i].distances[minIndex]);} // count the votes
from the k-nearest neighborts
Unordered_map<string , int>>voteCount;
For(int i=0;i<k'i++){
String label = distances[i].second; voteCount[label]++I;
```

```

}
// find the label from maximum votes
String predictedLabel;
Int maxVotes=0;
For(const auto& entry : voteCount){
If(entry.second > maxVotes){
predictedLabel = entry.first;}} return
predictedLabel; } int main() { //new
object to be classified by robot
Object newObjcr = {3.8,"red"}; / number of nearest neighbors to consider
Int k=3; // classify the object using k-nearest neighbors
String predictedLabel = classifyObject( newObjcr , k);
//Output the predicted label
Cout<<"predicted Label :"<<predictedLabel<<endl; return 0;

```

Example 1.2:

This C++ application controls a simple robot by allowing the user to enter commands to move and rotate the robot. It encapsulates many actions, such as forward and backward movement and left and right turning, using functions. The primary function consists of a loop that asks the user for input repeatedly until 'q' is entered, at which point the user can operate the robot [2].

```

#include<iostream>
Using namespace std;
// function to move robot forward
Void moveForward() {
Cout<<"robot moving forward "<<endl;}
// function to move robot backward Void
moveBackward{
Cout<<"robot moving backward"<<endl;
}
Function to turn robot right
Void turnRight(){
Cout<<"robot turning right"<<endl;}
// function turnLeft(){
Cout<<"robot turning left";}

int main(){
// user input for robot movement
Char userInput;
Cout<<"control the robot ( f: forward , b: backward , l:left , r:right, q:quit):";
//continue taking user input until 'q' is entered
While(true){
Cin> userInput;

```

```

// perform the selected operation  Switch(userInput){
Case
'f':
moveForward();
break; Case 'b': movebackward(); break; Case
'l': turnLeft(); break; Case 'r': turnRight(); break;
case 'q':
cout<<"existing the robot control program,"<< endl; return
0; default:
cout<<"invalid input, try again "<<endl;} //
prompt for the next user input
Cout<<"control the robot ( f: forward , b: backward , l:left , r:right, q:quit):"; }return
0;}

```

Example 1.2:

Using a class called LoginSystem, this C++ program implements a basic login system. Private member variables for the right password and username are contained in the class. The authenticateUser method, which verifies that the password and username entered match the right credentials, is the primary functionality. An instance of login system is created in the main function, where the user is prompted to enter their password and username. The authenticate user function of the is then called with the entered credentials. login system object, and the application outputs the success or failure of the login

```

#include<iostream>
#include<string>
Using namespace std;
Class LoginSystem{ Private:
Const string correctUsername = "yourUsername";
Const string correctPassword = "password123"; Public:
// function to check if the provided user name and password are correct bool
authenticateUser(const string&
enteredUsername,const sting enteredPassowrd)}
}; int main(){
// create a login system
LoginSystem myLoginSystem;
// input username and password from the user
String eterdUsername,enteredPassowrd;
Cout<<"enter your username :";
Cin>>enteredUsername; Cout<<"enter your
password :";
Cin>>enteredPassword;
// authenticate the user using the login system

```

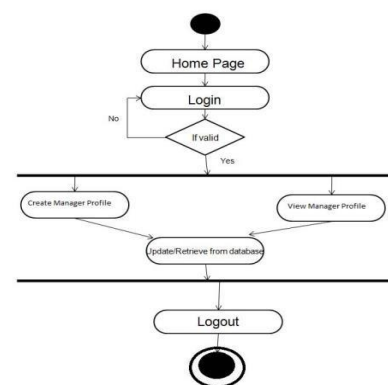


figure2.3

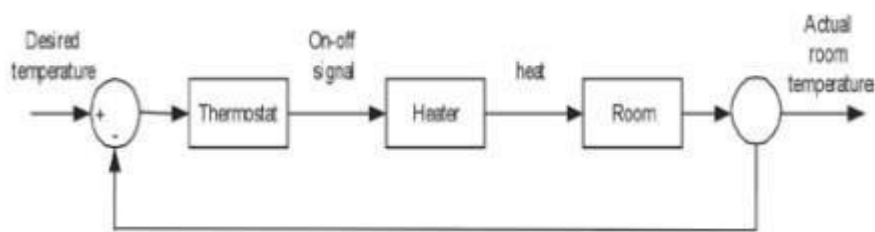
```

If(myLoginSystem.authenticateUser(enteredUsername, enteredPassowrd)){
  Cout<<"login successful. Welcome, "<<enteredUsername<<"!"<<endl; }else{ Cout<<"
  login failed . incorrect username or password. "<<endl;
}
Return 0;}

```

Example 1.3:

An explanation of fuzzy logic systems Four primary components make up the fundamental design of the fuzzy logic system: the fuzzy rule base, fuzzy inference engine, fuzzifier, and defuzzifier. A set of fuzzy IF-THEN rules make up the fuzzy rule base. An example of a fuzzy rule that describes the link between a heating power and a room's temperature trend may be. The temperature will rise quickly if the heating output is high (Kobersi, 2013)



FUZZY LOGIC 2.4:

We have a straightforward fuzzy logic system that uses the current temperature to control heating. The control heating function combines the fuzzy sets "cold," "warm," and "hot," which are defined with triangular membership functions, to calculate the heating amount. Next, a percentage representation of the output is shown. This is an explanation of the code.

```

#include<iostream>
// fuzzy sets for temperature
Double cold(double temperature)
{
  If ( temperature <=20){
Return 1.0;
  } else if (20 < temperature && temperature <= 25){
  Return (20 – temperature / 5.0)
} else{
return 0.0;
}
}
double warm(double temperature){ if(20 <
temperature && temperature <= 25 ){ return ( temperature
- 20 / 5.0 ;
}
else if (25 <temperature && temperature <= 30 )
{ return (30 – temperature / 5.0 );} else{ return 0.0; }
}
double hot(double temperature){ if(temperature <=

```

```

25 ){
    return 0.0 ;
}     else if (25 < temperature && temperature <= 30
) {   return (temperature – 25 ) / 5.0;
}     else {
    Return 1.0;
    }
}
// fuzzy rules for heating control
Double control_heating (double temperature ){
    Double heating_level=0.0;
// rule 1 : if it's cold , increase heating
heating_level=std::max(heating_level,cold(temperature));
// rule 2 : if it's warm , maintain heating
    Heating_level=stdmax(heating_level,warm(temperature));
// rule 3 : if it's hot ,decrease heating
heatint_level=std::max(heating_level,hot(temperature));
    Return heating_level; }

```

The heating level is a measurement of how much heating should be adjusted, taking into consideration "warm" and "hot." The actual rules and fuzzy sets can be adjusted based on specific heating control requirements. In this example, if the current temperature is, for example, 22C⁰, the program determines the heating level based on the fuzzy rules and presents it as a percentage.

1.3 Quadratic Function : A polynomial function of degree 2 is a quadratic

function, meaning that its highest power term is squared. The quadratic function's

generic form is $f(x) = ax^2 + bx + c$ Here:

- x is the variable .
- a,b and c constant with $a \neq 0$
- ax^2 represents the quadratic term .
- bx represents the linear term .
- c is the constant term .

A quadratic function's graph is a parabola, or U-shaped curve.

The sign of the coefficient a determines whether the parabola will open upward or downward.

Let's look at a practical illustration of a quadratic function. Suppose you are working on a physics assignment. Specifically, the following quadratic function can be used to simulate the motion of a projectile shot vertically:

$h(t) = -4.9t^2 + v_0t + h_0$ where : $h(t)$ is the height of the projectile at t , t is the time in second. v_0 is the initial velocity of the projectile . h_0 is the initial height , now , let's create a c++ program to calculate the height of the projectile at a given time , given specific initial condition :

Example 1.4:

```
#include<iostream>
double projectileHeight (double t , double v0, double h0)
{ Const double gravity = 9.8; // acceleration due gravity in m/s^2 return -4.9 * t * t + v0 * t +
h0 ;
}
Int main(){
    // initial condition for the projectile      double initialVelocity = 20.0; //
in m/s      double initialHeight = 10.0; // in meters      // example : calculate
height at time = 2 seconds      double time = 2.0;
    double height = projectileHeight( time , initialVelocity , initialHeight );
    std::cout << " height of the projectile at t= " << time << " seconds : " <<
height
<< " meters/n "; return 0;
```

In this example, the quadratic function is used by the projectile Height function to determine the projectile's height at a particular time. The application then shows how to use this, given an initial condition, to find the height at a particular time. This type of modeling is often used in engineering and physics to forecast projectile trajectory.

1.4 Mathematical logic in c++ :

the rational process We would like to mention three more basic types of things when working with boolean values (true or false values). Let A and B be two bool variables. Statements like "If A and B are both true then this," "While either A or B is true then this," or "If A and not B are both true then this" will eventually need to be made. C++ offers a logical and operator (&&), a logical inclusive or operator (||), and a logical not operator (!) to help in expressing these concepts in code. the rational process The operators "&&" and "||" act on two operands because they are binary operators. The "!" operator, on the other hand, acts on a single operand and is a unary operator. Keep in mind that the operand(s)' truth-value determines the truth-value of these logical procedures. The truth-values of these logical operators for various truth combinations of A and B are displayed in the truth

tables that follow, where "T" stands for true and "F" for false. The truth table, logical AND (&&). Observe that an AND operation is true if and only if both of its operands are true based on the four conceivable combinations.

A	B	A&&B
T	T	T
T	F	F
F	T	F
F	F	F

Table 1.1

Logical OR (||) truth table. Observe from the four possible combinations that an OR operation is true if and only if at least one of its operands is true.

A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F

Table 1.2

Logical NOT (!) truth table. The not operator simply negates the truth-value of a boolean operand -true becomes false when negated and false becomes true when negated. (Luna, 2017)

A	!A
T	F
F	T

Table 1.3

Chapter two Simple linear regression and Newton's forward interpolation

Programming to math equations involves using code to implement mathematical concepts and algorithms. Three commonly used methods for this are simple linear regression, k-nearest neighbor, and Newton's forward interpolation. These techniques are used to analyze data, make predictions, and interpolate values between known data points.

2.1 Simple linear regression :

Simple linear regression lives up to its name: it is a very straightforward simple linear approach for predicting a quantitative response Y on the basis of a simple linear regression predictor variable X . It assumes that there is approximately a linear relationship between X and Y . Mathematically, we can write this linear relationship as

$$Y \approx \beta_0 + \beta_1 X$$

You might read " \approx " as "is approximately modeled as". We will sometimes describe by saying that we are regressing Y on X (or Y onto X) For example, X may represent TV advertising and Y may represent sales Then we can regress sales onto TV by fitting the model

$$sales \approx \beta_0 + \beta_1 \times Tv$$

In equation, β_0 and β_1 , are two unknown constants that represent the intercept and slope terms in the linear model. Together, β_0 and β_1 , are known as the model coefficients or parameters. Once we have used our slope training data to produce estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, for the model coefficients, we can predict future sales on the basis of a particular value of TV advertising parameter by computing

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$
 where \hat{y} indicates a prediction of Y on the basis of $X = x$.

Here we use a hat symbol, to denote the estimated value for an unknown parameter or coefficient, or to denote the predicted value of the response.

Simple linear regression model: There are parameter a_0 , a_1 and e such that for any fixed value of the independent variable x through the dependent variable is a random variable related to x through the model equation : (James, 2013)

$$y = a_0 + a_1x + e \quad \text{where :} \quad a_1 = \frac{\overline{xy} - \bar{x}\bar{y}}{x^2 - (\bar{x})^2} , \quad a_0 = \bar{y} - a_1\bar{x} , \quad \overline{xy} = \frac{\sum_{i=1}^n x_i y_i}{n}$$

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n} \quad \bar{y} = \frac{\sum_{i=1}^n y_i}{n} , \quad \overline{x^2} = \frac{\sum_{i=1}^n x_i^2}{n}$$

y : dependent variable .

- a_0 : y intercept .
- a_1 : slop coefficeint of x .
- x : independent variable .
- e : random error .

Example 2.1: let us consider an example where five week's sales data (in thousand) in given show table :

Week's x_i	Sales(in thousand) y_i
1	1.2
2	1.8
3	2.6
4	3.2
5	3.8

Table 2.1

x_i	y_i	x_i^2	$x_i y_i$
1	1.2	1	1.2
2	1.8	4	3.6
3	2.6	9	7.8
4	3.2	16	12.8
5	3.8	25	19
Σx_i = 15	Σy_i = 12.6	Σx_i^2 = 55	$\Sigma x_i y_i$ = 44.4
\bar{x}_i = 3	\bar{y}_i = 2.52	\bar{x}_i^2 = 11	$\bar{x}_i \bar{y}_i$ = 88.8

Table 2.2 $a_1 = \frac{\bar{xy} - \bar{x}\bar{y}}{\bar{x}^2 - (\bar{x})^2} = \frac{8.88 - 3 \times 2.52}{11 - (3)^2} = 0.66$, $a_0 = \bar{y} - a_1 \bar{x} = 2.52 - 0.66 \times 3 = 0.54$

(linear regression) $y = a_0 + a_1 x = 0.54 + 0.66x$, at $x = 7$ (7 , 12th week) and $x = 7$ $y = 0.54 + 0.66 \times 7 = 5.16$

At $x=12$, $y = 0.54 + 0.66 \times 12 = 8.46$

Here's c++ program to this example :

Example 2.1:

```
#include<iostream> #include<cmath> using
namespace std;
int main(){ while(true){
float x[100] , y[100] yx[100],xx[100];
float sum1=0 , sum2=0,sum3=0; int n; \ \ the
number of data cout<<" Enter the number of data
:"<<endl; cin>>n; for(int i=1;i<=n;i++){
cout<<"x{"<<i<<"}="; cin>>x[i];
cout<<"y{"<<i<<"}=";
cin>>y[i];
yx[i]=x[i]*y[i]; xx[i]=x[i]*x[i]; }
for( int i=1;i<=n;i++){ cout<<"
yx{"<<i<<"}="; cout<<yx[i]<<endl; }
for(int i=1;i<=n;i++){ cout<<"x^2{"<<i<<"}=";
cout<<xx[i]<<endl; } for (int i=1;i<=n;i++){ sum1=sum1+x[i];
sum2=sum2+y[i]; sum3=sum3+yx[i]; sum4=sum4+xx[i]; }
Cout<<"sum of xi :"; Cout<<sum1<<endl;
Cout<<"sum of yi :"; Cout<<sum2<<endl;
```

```

Cout<<" sum of yi*xi :";
Cout<<sum3<<endl;
Cout<<"sum of xi^2 :";
Cout<<sum4<<endl;
Float  avg1  ,      avg2  ,      avg3  ,
        avg4; avg1=(sum1/n);      avg2=(sum2/n);
avg3=(sum3/n);      avg4=(sum4/n);      cout<<"
avarege of xi :"; cout<<avg1<<endl; cout<<" avatage
of yi :"; cout<<"avg2<<endl; cout<<" average of
xi*yi :"; cout<<" avg3 <<endl; cout<<"      average
of      xi^2      :";
cout<<avg4<<endl; float a1,ao,g,m;
a1=(avg3-(avg1*avg2))/(avg4-(avg1^2)); ao=avg2-a1*avg1; cin>>m; \ m is week
y=ao+a1*m;cout<<"y="<<y<<endl; } return 0; }

```

2.2K-Nearest Neighborhood: The biggest advantage to thinking of examples as vectors in a high dimensional space is that it allows us to apply geometric concepts to machine learning. For instance, one of the most basic things that one can do in a vector space is compute distances. In two-dimensional space, the distance between $\langle 2, 3 \rangle$ and $\langle 6, 1 \rangle$ is given by $\sqrt{(2 - 6)^2 + (3 - 1)^2} = \sqrt{18} \approx 4.24$. In general, in D-dimensional space, the Euclidean distance between vectors a and b is given by Eq (3.1) (see Figure 3.2 for geometric intuition in three dimensions)

$$d(a, b) = \left[\sum_{d=1}^D (a_d - b_d)^2 \right]^{\frac{1}{2}}$$

Now that you have access to distances between examples, you can start thinking about what it means to learn again. Consider Figure 3.3. We have a collection of training data consisting of positive examples and negative examples. There is a test point marked by a question mark. Your job is to guess the correct label for that point. Most likely, you decided that the label of this test point is positive. One reason why you might have thought that is that you believe that the label for an example should be similar to the label of nearby points. This is an example of a new form of inductive bias. The nearest neighbor classifier is built upon this insight. In comparison to decision trees, the algorithm is ridiculously simple. At training time, we simply store the entire training set. At test time, we get a test example \hat{x} . To predict its label, we find the training example x that

is most similar to \hat{x} . In particular, we find the training example x that minimizes $d(x, \hat{x})$. Since x is a training example, it has a corresponding label, y . We predict that the label of \hat{x} is also y . Despite its simplicity, this nearest neighbor classifier is incredibly effective. (Some might say frustratingly effective.) However, it is particularly prone to overfitting label noise. Consider the data in Figure 3.4. You would probably want to label the test point positive. Unfortunately, its nearest neighbor happens to be negative. Since the nearest neighbor algorithm only looks at the single nearest neighbor, it cannot consider the “preponderance of evidence” that this point should probably actually be a positive example. It will make an unnecessary error. A solution to this problem is to consider more than just the single nearest neighbor when making a classification decision. We can consider the K -nearest neighbors and let them vote on the correct class for this test point. If you consider the 3-nearest neighbors of the test point in Figure 3.4, you will see that two of them are positive and one is negative. Through voting, positive would win. The full algorithm for K -nearest neighbor classification is given in

Algorithm 3.2. Note that there actually is no “training” phase for K -nearest neighbors. In this algorithm we have introduced five new conventions:

- 1-The training data is denoted by D
- 2-We assume that there are N -many training examples
- 3-These examples are pairs $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$. (Warning: do not confuse x_n , the n th training example, with x_d , the d th feature for example x .)
- 4-We use $[]$ to denote an empty list and $\oplus \cdot$ to append \cdot to that list.
- 5-Our prediction on \hat{x} is called \hat{y}

The first step in this algorithm is to compute distances from the test point to all training points (lines 2-4). The data points are then sorted according to distance. We then apply a clever trick of summing the class labels for each of the K nearest neighbors (lines 6-10) and using the sign of this sum as our prediction.

The big question, of course, is how to choose K. As we've seen, with $K = 1$, we run the risk of overfitting. On the other hand, if K is large (for instance, $K = N$), then KNN-Predict will always predict the majority class. Clearly that is underfitting. So, K is a hyperparameter of the KNN algorithm that allows us to trade-off between overfitting (small value of K) and underfitting (large value of K) (Daumé, 2017)

Example 2.2: let's consider a dataset of house with feature “ square footage “ and “ number of bed room “we want to predict the price of a house based of these feature

House	Sq,ft	Bed room	price
1	1500	3	\$250,000
2	1800	3	\$300,000
3	2000	4	350,000
4	1400	2	200,000

Table 2.3

Let's say we want to predict the price of a house with (1600 sq.ft , 3 bed room)
Assume $k=2$

Calculate the distance between target house and each data point By

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$d = \sqrt{(1600 - 1500)^2 + (3 - 3)^2} = 100$$

$$d = \sqrt{(1600 - 1800)^2 + (3 - 3)^2} = 200$$

$$d = \sqrt{(1600 - 2000)^2 + (3 - 4)^2} = 447.21$$

$$d = \sqrt{(1600 - 1400)^2 + (3 - 2)^2} = 141.42$$

House	Sq.ft	Bed room	distance
1	1500	3	100
2	1800	3	200
4	2000	4	447,21
4	1400	2	141,42

Table 2.4

Select the two nearest neighbourhoods based on distance

H1 (\$250,000) , H2(\$200,000)

And the calculate the average of price

$$(250,000 + 200,000) / 2 = \$225,000$$

Then the new house with (1600 sq.ft , 3 bed rooms) the price is (\$225,000)

Let's write a c++ program to this example :

Exmpl 2.3:

```
#include<iostream>
#include<cmath>
#include<vector>      struct
House { int sq; int bd; int pr;
}; double calculateDistance( int sq1, int bd1 , int sq2 , int bd2){
return sqrt(power(sqr – sqr2 ,2) + pow(bd1 – bd2 , 2));    }
Int main() {
// define the house  House
houses[] = {
{1500 , 3 , 250000 }
           {1800 , 3 , 300000}
           {2000, 4 , 350000}
{1400 , 2 200000 }
};
// house to find the price for
Int targetsq = 1600;
Int targetbd = 3;
// set k value
Int k=2;
// calculate distance and find k-nearest neighbors
Std::vector<std::pair<int,double>>distances;
For(int i=0 ; i< sizeof(houses) / sizeof(houses[0]); ++i
```

```

    {
    Double distance = calculateDistance(targetsq , targetbd , houses[i].sq, houses[i]. bd
    );
    Distance.push_back({i,distance});
    }
    // sort distances
    Std::sort(distances.begin(),distances.end(), []( const auto &a, const auto &b){ return
    a.second<b.second;
    });
    // calculate average price of k-nearest neighbors  int
    totalprice = 0; for (int i=0 ; i<k ; ++i){ totalprice+=
    hoses[distances[i].first].pr;
    }
    // calculate average
    Int averageprice = totalprice / k;
    // display the result
    Std::cout <<" Estimate price a house with "<< targetsq<<" sq and "<< tergerbd <<" bd
    using k-nearest neighbors ( k="<< k <<" ) is $"<< avarageprice << std:: endl; return 0;
    }

```

2.3 Newton's forward interpolation formula :

$$y_p = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!} \Delta^2 y_0 + \frac{p(p-1)(p-2)}{3!} \Delta^3 y_0 + \dots \frac{p(p-1)\dots(p-(n-1))}{n!} \Delta^n y_0 \text{ (Burden, 2010)}$$

Example 2.4:

Suppose you have a dataset of weather temperature reading with missing values at certain time points , and you want fill in this missing value using interpolation , the data might look this :

Time(hours) x	Temperature(c ⁰) y
0	20
1	22
2	24
3	25
4	26

Table 2.4

To find Δy , Δy^2 , Δy^3 and Δy^4

x	y	Δy	Δy^2	Δy^3	Δy^4
0	20	2	0	-1	
1	22	2	-1	1	2
2	24	1	0		
3	25	1			
4	26				

Table 2.5

We find the temperature the hour (1.5). If $y(1.5) = 23.109357$

Let's write c++ program :

```
#include<iostream>
Double calculateInterpolation (double x[], double y[] , int n , double target ) {
Double result = y[0];
Double u= ( target - x[0] ) / (x[1] - x[0]); For
(int i=1;i<n;i++){ Double term = 1.0;
For(int j=0;j<i;j++){ term*=(u-j) / (j+1);
} result
+= term * y[i];
}
Return result ;
} int
main(){
// given data
Double x[] = { 0 , 1 , 2 , 3 , 4 }; Double y[] =
{ 20 , 22 , 24 , 25 , 26 }; int n = sizeof(x)
/ sizeof(x[0]); // target value
Double target = 1.5 ;
// calculate interpolation
Double result = calculateInterpolation (x , y , n , target );
// output the result
Std::cout << " interpolation value at x = "<< target <<" is : " << result <<std::endl;
Retur 0; }
```

2.4 Graph theory applications in computer science

Graphs provide a convenient way to represent various kinds of mathematical objects. Essentially, any graph is made up of two sets: 1- A set of vertices 2- A set of edges. Depending on the particular situation, restrictions are imposed on the type of edges we allow. For some problems directed edges are applied and for

other problem undirected edges are applied from one vertex to other. So graphs give us many techniques and flexibility while defining and solving a real life problem.

In this section, we study the role of Graph theory in computer science. Computer scientists use graphs to model problems as diverse as how to detect a deadlock condition in an operating system and how to plan efficient routings for transportation networks. Informally, a graph is a bunch of dots and lines where the lines connect some pairs of dots. The dots are called vertices (or nodes) and the lines are called edges. Graphs are simple but extremely useful mathematical objects; they are ubiquitous in practical applications of computer science. For example:

- In a computer network, we can model how the computers are connected to each other as a graph. The nodes are the individual computers and the edges are the network connections. This graph can then be used, for example, to route messages as quickly as possible.
- In a digitalized map, nodes are cities and edges are roads (or highways).

We may have directed edges (directed graph) to capture one-way streets

- On the internet, nodes are web pages, and (directed) edges are links from one web page to another.
- In a social network, nodes are people, and edges are friendships.

Graphs are used in a wide variety of models. Here we just count some of them

1. Social Networks.
2. Communication Networks (Call Graphs).
3. Information Networks (The Web Graph, Citation Graphs).
4. Software Design Applications.
5. Transportation Networks.

Graphs are regarded as an excellent modeling tool for representing multiple stages of interactions between all physical conditions. Several real-world problems can be depicted using graphs. Here are some key graph applications:

Social Networks:

Graphs are unique network conditions that have only one sort of edge between vertices. Web Graphs: The internet contains a large number of hyperlink references. In other words, the web is another great source of graph data.

Biological Networks: Space (or biological networks) is a major source of realworld graphs. Examples include brain networks, protein communication networks, and nutrition networks.

Information graphs: Geographical information is organized in a graph, and information A is linked to information B when A represents B in a specific way.

Product recommendations.

2.5 Game theory : is a branch of mathematics that deals with the study of strategic interaction between rational decision – makers . it is widely applied in various fields , including economics , political science, biology and computer science .

Example 2.5: in c++ to game theory Example:

```
#include<iostream>
#include<cstdlib>
#include<ctime> using
namespace std;    int main(){ cout<<" *****guessing number
game*****\n\n";
    int Player,computer,times=0;          srand ( time (0) );    computer=(
rand () % 100 )+1; \ a random number from computer    cout<<" Enter
your number from 0 to 100\n";    cin>>player;    while( player !=
computer){    times++;    if ( player < computer ){    cout<<"
your number is small :\n";    cin>>player ;
    }
    If (player > computer ){    cout<<" your number
is big :\n";    cin>>player ;
    } }    Cout<<" you are win after ( "<< times <<" ) times ";    return 0; }
```

Example 2.6: We have an example to create testing calculator in c++ program use mathematical statement in it :

```
#include<iostream>
#include<cstdlib>
Using namespace std;
Int main(){
```

```

While(1){
    Cout<<"***** calculator testing game ***** \n\n";
    Cout<<"***** choose one of the operation do you want to test *****\n";    Char op;
    Cin>>op;
    Double    x,y,x0,x1,z,v;
    Srand(time(0));    int n=0;
    if(op=='+' ){    while(n<=3){
    x0=(rand()
% 20)+1;    x1=(rand() % 20
)+1;    z=x0+x1;
    cout<<x0<<"+"<<y<<"="
; cin>>v;    if(v==z){
    cout<<"correct \n";    }
    if(
v!=z){ cout<<"in
correct\n";    n++;
        }
    }
    Cout<<" you are fail \n";
    }    if(op=='-'){
    while(n<=3){    x0=(rand() %
20)+1;    x1=(rand() % 20
)+1;    z=x0-x1;    cout<<x0<<"-"<<y<<"=";
    cin>>v;    if(v==z) {    cout<<"correct
\n";
        }    if( v!=z){    cout<<"incorrect\n";
    n++;
    }
    }
    Cout<<" you are fail \n";
    }

    if(op=='*'){    while(n<=3){
    x0=(rand()
% 20)+1;    x1=(rand() % 20
)+1;    z=x0*x1;
    cout<<x0<<"*"<<y<<"=";    cin>>v;
    if(v==z){    cout<<"correct \n";    }    if( v!=z){
    cout<<"incorrect\n";    n++;    }
    }
    Cout<<" you are fail \n";
    }    if(op=='/'){    while(n<=3){    x0=(rand()
% 20)+1;    x1=(rand() %
20 )+1;    z=x0/x1;
    cout<<x0<<"/"<<y<<"=";
    cin>>v;    if(v==z){
    cout<<"correct \n";
    }
    }
}

```

```

if( v!=z){    cout<<"incorrect\n"; n++;
}
}
Cout<<" you are fail \n";
} Cout<<endl; } return 0; }

```

2.6 Binary system :

Example 2.7: Let's consider an example where binary numbers are used in a robot's control system to represent sensor data and make decisions. Suppose you have a robot equipped with a line-following sensor that detects whether it's over a black line or a white surface. The sensor provides analog readings, and these readings are converted into binary values for processing.

- Black Line: 1 (binary: 01)
- White Surface: 0 (binary: 00)

The robot's control system continually reads these binary values from the sensor. Let's say the robot's objective is to follow the line. It might have a simple rule. If the sensor reads 01 (black line), turn left. If the sensor reads 00 (white surface), turn right. These rules are implemented in the robot's control algorithm using binary numbers to represent the sensor states. The robot processes this information in real-time, making decisions based on the binary-encoded sensor readings to navigate and follow the line.

Example 2.8:

```

C++ program:
#include<iostream>
Const int BLACK_LINE = 0b01;    // binary values for line-following sensor
Const int WHITE_SURFACE = 0b00;
Void followinLine(int sensorReading){ // robot control function
    Std::cout<<" turn left\n";
    // additional code for left turn action
} else if (sensorReading == WHITE_SURFACE){
    Std::cout<<"turn right\n";
    // additional code for right turn action
} else{
    Std::cout<<"unknown sensor reading \n";
    // additional error handling or default action
} } int main(){ // simulate sensor reading
    Int sensorReading1=BLACK_LINE;    Int sensorReading2=WHITE_SURFACE;
// robot makes decisions based on binary-encoded sensor readings return 0; }

```

References

1. Kobersi, I.S., Finaev, V.I., Almasani, S.A. and Abdo, K.W.A., 2013. Control of the heating system with fuzzy logic. *World Applied Sciences Journal*, 23(11), pp.1441-1447.
2. Lentin, J., 2018. *Robot Operating System for Absolute Beginners*. Apress, Berkeley, CA.
3. Daumé, H., 2017. *A course in machine learning* (pp. 149-155). Hal Daumé III.
4. Keller, J.M., Gray, M.R. and Givens, J.A., 1985. A fuzzy knearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4), pp. 580-585.
5. James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. *An introduction to statistical learning* (Vol. 112, p. 18). New York: springer.
6. Luna, F. (2004). *C++ Programming for Games Module I*. eInstitute Publishing Inc.
7. Burden, R. L., & Faires, J. D. (2010). *Numerical Analysis* (9th ed.). Cengage Learning.
8. Negnevitsky, M. (2011). *Artificial Intelligence: A Guide to Intelligent Systems* (2nd ed.). Pearson Education Limited.
9. Sekharan, S. C., et al. (2017) "Application of session login and one-time password in fund transfer system using RSA algorithm," April, pp. 1-6. DOI: 10.1109/ICECA.2017.8212763.

زیرہکی دستکرد (AI) بواریکی خیرا و پھرہسہندووه کہ زور پشت بہ بنہما بیرکاریہکان دہہستی ت بو دیزاینکردنی سیستہ می زیر ہک کہ توانای ئ ہنجامدانی ئ ہرکہکانی ہہیہ کہ بہ شیوہیہکی گشتی پیویست یان بہ زیرہ کی مرو ف ہہیہ. لہ ئل گوریتہکانی فیربوونی نامیرہوہ تا توره دہمارییہکان، بیرکاری بناغہی AI پیکدہہینی ت بہ دابینکردنی چوارچیوہ ب و مؤدیلکردنی کیشہ ئالوزہکان، شیکردنہوہی داتاگان و بریاردانی ناگادارانہ. ئہم پیشہکیی ہ بہدواداچوون بو ئہو چ ہمکہ بنہرہتیہ بیرکاریانہ دہکات کہ ہ بنہمای AI یہ، ک ہ نیشان دہدات کہ چون بیرکاری رول یکی گرنگ دہگیری ت لہ دارشتنی داہاتووی زیرہکی دستکرد.