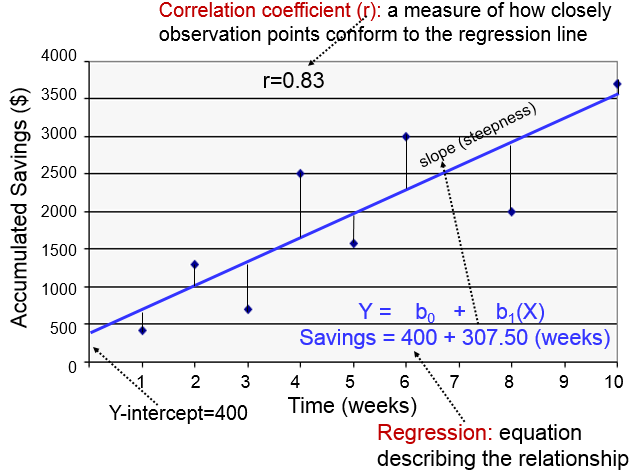
Algoritmo usado: Linear Regresion

Introducción:

Básicamente la regresión linear es un algoritmo de aprendizaje supervisado que se puede utilizar en el machine learning y también en la estadística. Una de sus versiones clásicas es principalmente “dibujar una recta “que nos indicara la tendencia de un conjunto de datos continuos. Adicionalmente en la estadística, la regresión lineal es una aproximación para el modelado de la relación entre una variable escalar dependiente “y” y una o más variables explicativas nombradas “x”.

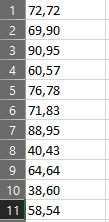


Aplicación:

En el siguiente documente se explicará la implementación del algoritmo programado paralelamente usando correctamente los métodos de sincronización y lectura del dataset.

* DataSet(Students performance in exams)

Link: *https://github.com/DiegoKraenau/TA2Concurrente/raw/main/StudentsPerformance.csv*



*StudentsPerformance.csv*

Formato: MathScore,ReadingScore=x,y

* Leer el csv correspondiente(Se descarga desde un URL publica subida en github)

func leerCSV(c chan []float64) {

    mutex.Lock()

    fileUrl := "https://github.com/DiegoKraenau/TA2Concurrente/raw/main/StudentsPerformance.csv"

    err := DescargarArchivo("StudentsPerformance.csv", fileUrl)

    if err != nil {

        panic(err)

    }

    path := flag.String("f", "./StudentsPerformance.csv", "CSV filepath")

    dataFile, err := os.Open(\*path)

    reader := csv.NewReader(dataFile)

    var x []float64

    var y []float64

    for {

        record, err := reader.Read()

        if err != nil {

            if err == io.EOF {

                break

            }

            fmt.Printf("Error al leer el csv: %v", err)

        }

        // convertir todas las filas a float64

        for i, val := range record {

            float, err := strconv.ParseFloat(val, 64)

            if err != nil {

                fmt.Printf("Error al convertir el valor <%v> afloat64: %v", val, err)

                break

            }

            if i < len(record)-1 {

                x = append(x, float)

            } else {

                y = append(y, float)

            }

        }

    }

    c <- x

    c <- y

}

func DescargarArchivo(filepath string, url string) error {

    // Obtiene la data

    resp, err := http.Get(url)

    if err != nil {

        return err

    }

    defer resp.Body.Close()

    // Crea el archivo

    out, err := os.Create(filepath)

    if err != nil {

        return err

    }

    defer out.Close()

    // Lo escribe al archivo

    \_, err = io.Copy(out, resp.Body)

    return err

}

* Forma de la ecuación de linear regresión

y = mx + b

* Estructura de la ecuación en Golang

type Ecuacion struct {

M, B float64

}

* Como se ve, mantendremos nuestras dos constantes (pendiente [M] e intersección con el eje y [B]) en la estructura de la ecuación. Tener en cuenta que las variables dentro de la estructura de la ecuación están en mayúsculas y por lo tanto son públicas.

M = (total sum y)\*(total sum x^2)-(total sum x)\*(total sum x \* y)/n(total sum x^2)-(total sum x)^2

B = n(total sum x \* y)-(total sum x)\*(total sum y)/n(total sum x^2)-(total sum x)^2

En Golang:

type LinearRegression struct {

    x, y []float64

    xTotal, yTotal, xSqTotal, xYTotal, numerator, denominator,

    slope, yInt float64

}

* Construcción de la función que toma las matrices de datos “x” y “y” la cual devuelve la estructura de la ecuación correspondiente

func (e \*Ecuacion) ReturnEcuacionParts(c chan []float64,num int) {

    var l LinearRegression

    l.LinearRegressionInit(c)

    e.M = l.slope

    e.B = l.yInt

    sayEcuation(e.M, e.B,num)

}

* Función que calcula los totales de la regresión lineal

func (l \*LinearRegression) LinearRegressionInit(c chan []float64) {

    l.x = <-c

    l.y = <-c

    //fmt.Println(len(l.X))

    for i, \_ := range l.x {

        l.xTotal = l.xTotal + l.x[i]

        l.yTotal = l.yTotal + l.y[i]

        l.xSqTotal = l.xSqTotal + math.Pow(l.x[i], 2)

        l.xYTotal = l.xYTotal + (l.x[i] \* l.y[i])

    }

    l.numerator = (float64(len(l.x)) \* l.xYTotal) - (l.xTotal \* l.yTotal)

    l.denominator = (float64(len(l.x)) \* l.xSqTotal) - (math.Pow(l.xTotal, 2))

    yIntNumerator := (l.yTotal \* l.xSqTotal) - (l.xTotal \* l.xYTotal)

    l.yInt = yIntNumerator / l.denominator

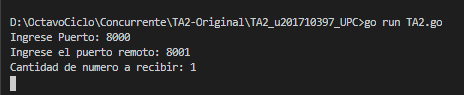
    l.slope = l.numerator / l.denominator

    fmt.Println("Datos para MathScore:", l.x)

    fmt.Println("Datos para WritingScore", l.y)

}

* Se ejecuta el servidor que contiene la lógica ingresando los puertos que se usaran:



* Luego se ejecuta el lanzador.go el cual simula al cliente:

*Se ingresa en enviar(54) por lo que se interpreta que se está enviando el MathScore que es igual a 54 para que el algoritmo determine el posible resultado del WritingScore.*

package main

import (

    "bufio"

    "fmt"

    "net"

    "os"

    "strings"

)

var remotehost string

func main() {

    //host remoto y puerto

    rIn := bufio.NewReader(os.Stdin)

    fmt.Print("Ingrese el puerto de destino: ")

    port, \_ := rIn.ReadString('\n')

    port = strings.TrimSpace(port)

    remotehost = fmt.Sprintf("localhost:%s", port)

    enviar(54)

}

func enviar(num int) {

    conn, \_ := net.Dial("tcp", remotehost)

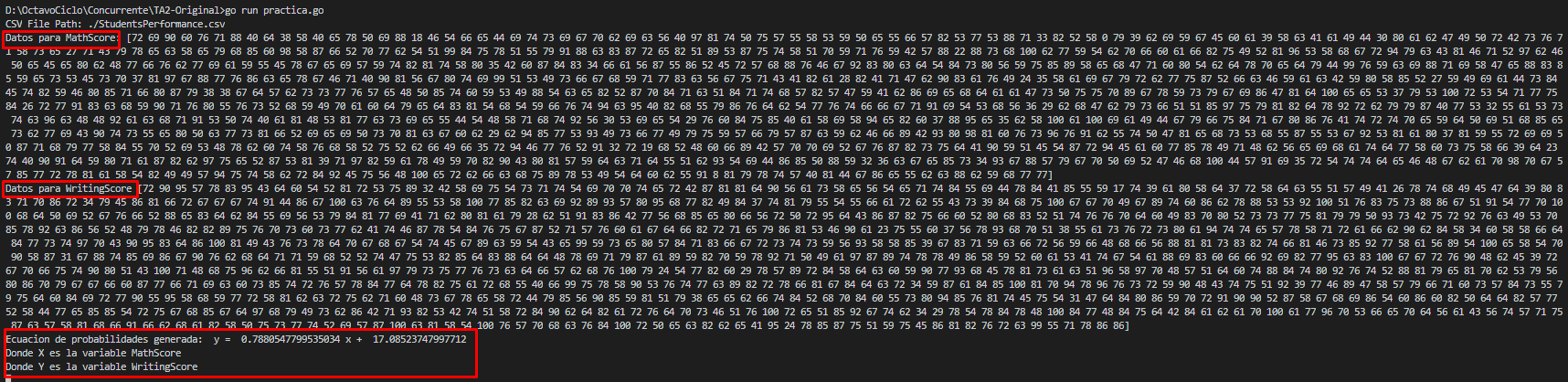
    defer conn.Close()

    fmt.Fprintf(conn, "%d\n", num)

}



* Finalmente por parte del nodo del cliente se visualiza la implementación del algoritmo y el mismo resultado probable del WritingScore



*En la consola de deberá mostrar los datos leídos del MathScore y WritingScore, adicionalmente se genera la ecuación de probabilidad la cual básicamente acepta 2 parámetros: MathScore y WritingScore.*

Ejemplo:

Ecuacion: y = 0.7880547799535034 x + 17.08523747997712

Donde X es la variable MathScore

Donde Y es la variable WritingScore

¿Predecir cuál es el resultado del WritingScore de un alumno el cual tiene como MathScore =54?

y = 0.7880547799535034 (54) + 17.08523747997712

y=59.6401956

La predicción del WritingScore básicamente según el algoritmo seria 59.64 el cual redondeado seria 60.

**¿Qué se utilizó?**

**Algunos elementos básicos que se utilizaron fueron:El lenguaje de programación golang, programación paralela, buffering channels, encoding/csv,net,etc.**

Datos Estadísticos:

¿Predecir cuál es el resultado del WritingScore de un alumno el cual tiene como MathScore =54?

Donde X es la variable MathScore

Donde Y es la variable WritingScore

* 100 datos:

y = 0.8779841313304338 x + 11.21368418036284

y=58.62482727220543

* 300 datos:

y = 0.8322254121990779 x + 13.648294271089876

y=58.58846652984001

* 600 datos:

y = 0.8077240008548033 x + 15.599260057068756

y=59.21635610322808

* 900 datos:

y = 0.7901510965194769 x + 16.888174119478908

y=59.55633333153065

* 1000 datos:

y = 0.7880547799535034 (54) + 17.08523747997712

y=59.6401956

En esta sección se puede ver con exactitud los datos estadísticos generados con diferentes datos de aprendizaje por parte del algoritmo, como se puede analizar podemos concluir que hay una ligera variación de error en la respuesta según los datos ingresados para que aprenda el algoritmo.

Conclusiones:

En conclusión, al finalizar este proyecto, se pudo aplicar correctamente el algoritmo supervisado para machine learning el cual es Linear Regresión. Adicionalmente, con la ayuda de la programación paralela en la implementación y consumo de un dataset público se pudo predecir la ecuación que da respuesta a la estimación de probabilidad relacionadas con factores “X” y “Y”. Finalmente, este trabajo de implementación dio grandes conocimientos de cómo usar la programación paralela con los usos de channels, golotines, etc.

Bibliografía:

Tusell, F. (2003). Estadística Matemática. 154 p., notas de clase.

William Mendenhall y Dennos D. Wackerly. ESTADÍSTICA MATEMÁTICA CON APLICACIONES, Segunda Edición. 1994 Editorial Iberoamericana.