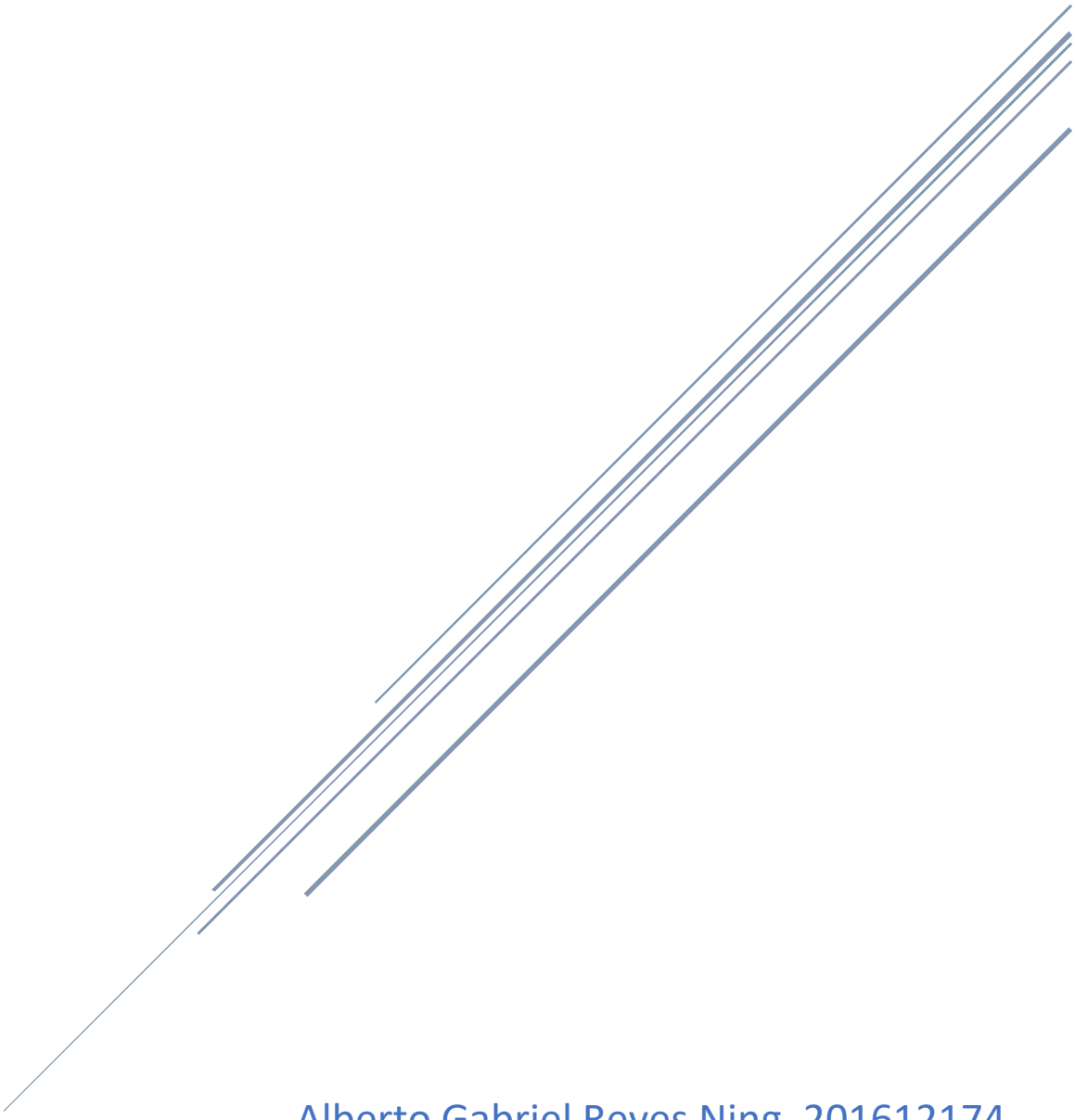
MANUAL TECNICO

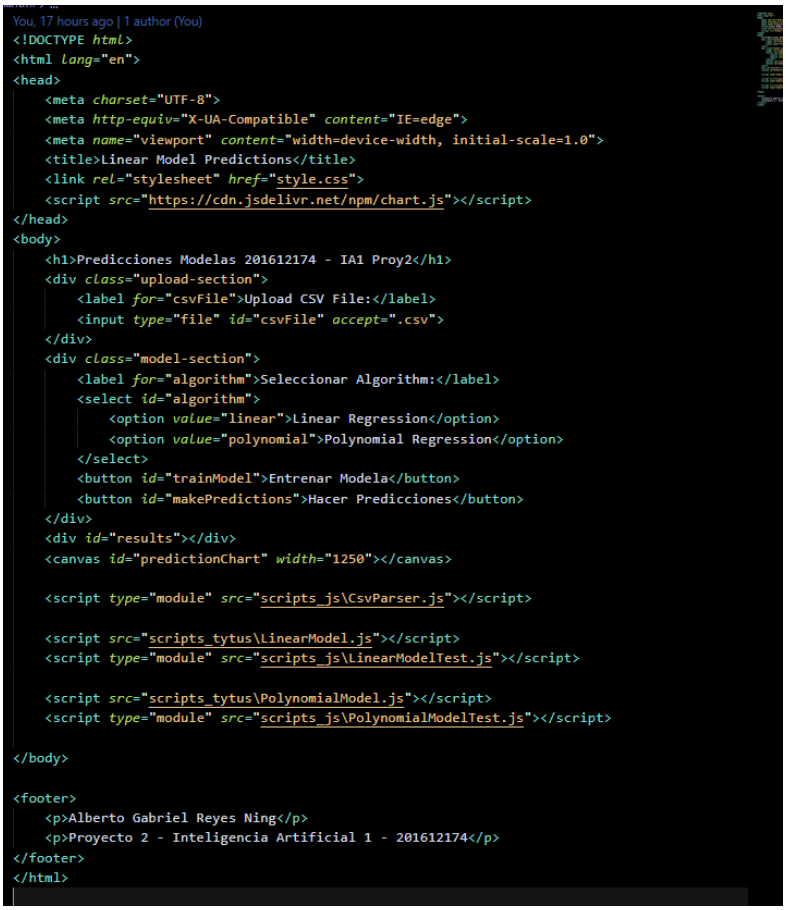


Alberto Gabriel Reyes Ning, 201612174
LAB IA1 2S24 PROY2

Introducción:

En el presente manual se describe cada una de las funciones que tiene el programa. Se explica paso a paso la forma correcta del uso del programa para un funcionamiento optimo del mismo. El programa consta de varias opciones, las cuales sirven para interactuar con el mismo.

Archivo index.html que contiene el homepage



```
You, 17 hours ago | 1 author (You)
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Linear Model Predictions</title>
    <link rel="stylesheet" href="style.css">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <h1>Predicciones Modelas 201612174 - IA1 Proy2</h1>
    <div class="upload-section">
        <label for="csvFile">Upload CSV File:</label>
        <input type="file" id="csvFile" accept=".csv">
    </div>
    <div class="model-section">
        <label for="algorithm">Seleccionar Algorithm:</label>
        <select id="algorithm">
            <option value="linear">Linear Regression</option>
            <option value="polynomial">Polynomial Regression</option>
        </select>
        <button id="trainModel">Entrenar Modela</button>
        <button id="makePredictions">Hacer Predicciones</button>
    </div>
    <div id="results"></div>
    <canvas id="predictionChart" width="1250"></canvas>

    <script type="module" src="scripts_js\CsvParser.js"></script>

    <script src="scripts_tytus\LinearModel.js"></script>
    <script type="module" src="scripts_js\LinearModelTest.js"></script>

    <script src="scripts_tytus\PolynomialModel.js"></script>
    <script type="module" src="scripts_js\PolynomialModelTest.js"></script>

</body>

<footer>
    <p>Alberto Gabriel Reyes Ning</p>
    <p>Proyecto 2 - Inteligencia Artificial 1 - 201612174</p>
</footer>
</html>
```

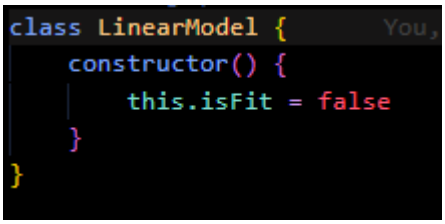
CSV Parser

Función utilizado para parsear los archivos CSV y guardarlo para los modelos

```
let xTrain = [];
let yTrain = [];

export function parseCSV(file, algorithm) {
    const reader = new FileReader();
    reader.onload = function(e) {
        const csvData = e.target.result;
        const lines = csvData.split('\n');
        xTrain = [];
        yTrain = [];

        lines.forEach(line => {
            const [x, y] = line.split(',').map(Number);
            if (!isNaN(x) && !isNaN(y)) {
                xTrain.push(x);
                yTrain.push(y);
            }
        });

        console.log(`Parsed Data for ${algorithm}:`, xTrain, yTrain);
    };

    reader.readAsText(file);
}

export function getXTrain() {
    return xTrain;
}

export function getYTrain() {
    return yTrain;
}
```

Clase Modelo Linear (LinearModel.js, obtenido de tytus)

Clase de Modelo Linear



```
class LinearModel {
    constructor() {
        this.isFit = false
    }
}
```

función fit utilizado para entrenar el modelo

```
fit(xTrain, yTrain) {
    var sumX = 0
    var sumY = 0
    var sumXY = 0
    var sumXX = 0

    //Se agrego la validación de que los datos de entrenamiento sean de la misma longitu
    if (xTrain.length != yTrain.length) {
        throw new Error('Los parametros para entrenar no tienen la misma longitud!');
    }

    //Validación ya que no se cuenta con manejo de errores en caso se llegara a implemen
    if (xTrain.length === 0) {
        return [ [], [] ];
    }

    for(var i = 0; i < xTrain.length; i++) {
        sumX += xTrain[i]
        sumY += yTrain[i]
        sumXY += xTrain[i] * yTrain[i]
        sumXX += xTrain[i] * xTrain[i]
    }
    this.m = (xTrain.length * sumXY - sumX * sumY) / (xTrain.length * sumXX - Math.pow(M
    this.b = (sumY * sumXX - sumX * sumXY) / (xTrain.length * sumXX - Math.pow(Math.abs(
    this.isFit = true
}
```

función predict utilizado para obtener los resultados

```
predict(xTest) {
    var yPredict = []
    if (this.isFit) {
        for(var i = 0; i < xTest.length; i++) {
            yPredict.push(this.m * xTest[i] + this.b)
        }
    }
    return yPredict
}
```

Función mserror utilizado para manejar errores

```
mserror(yTrain, yPredict) {
    var mse = 0
    for(var i = 0; i < yTrain.length; i++) {
        mse += Math.pow(yTrain[i]-yPredict[i],2)
    }
    return mse / yTrain.length
}
```

Clase Modelo Polinomial (PolynomialModel.js, obtenido de tytus)

Clase de Modelo Polinomial

```
class PolynomialModel {
    //Polinomial model that will be inherited by Polynomial Regression
    constructor() {
        this.isFit = false;
    }
}
```

Función fit utilizado para entrenar el modelo

```
class PolynomialRegression extends PolynomialModel {
    }

    //Method that trains the model in order to create the regression
    fit(xArray, yArray, degree) {
        //Equation matrix size based on the degree and number of elements
        let equationSize = degree + 1;
        let nElements = degree + 2;

        //Equation matrix to be solved
        let equations = new Array(equationSize);
        for (let i = 0; i < equationSize; i++) {
            equations[i] = new Array(nElements);
        }

        //Building equation matrix
        for (let i = 0; i < equationSize; i++) {
            for (let j = 0; j < nElements; j++) {
                let sum = 0;
                if (i == 0 && j == 0) {
                    sum = xArray.length;
                }
                else if (j == nElements - 1) {
                    for (let k = 0; k < xArray.length; k++) {
                        sum += Math.pow(xArray[k], i) * yArray[k];
                    }
                }
                else {
                    for (let k = 0; k < xArray.length; k++) {
                        sum += Math.pow(xArray[k], (j + i));
                    }
                }
                equations[i][j] = sum;
            }
        }

        //Staggering matrix
        for (let i = 1; i < equationSize; i++) {
            for (let j = 0; j <= i - 1; j++) {
                let factor = equations[i][j] / equations[j][j];
                for (let k = j; k < nElements; k++) {
                    equations[i][k] = equations[i][k] - factor * equations[j][k];
                }
            }
        }

        //Solving matrix
        for (let i = equationSize - 1; i > -1; i--) {
            for (let j = equationSize - 1; j > -1; j--) {
                if (i == j) {
                    equations[i][nElements - 1] = equations[i][nElements - 1] / equations[i][j];
                }
                else if (equations[i][j] != 0) {
                    equations[i][nElements - 1] -= equations[i][j] * equations[j][nElements - 1];
                }
            }
        }

        //Storing solutions
        this.solutions = new Array(equationSize);
        for (let i = 0; i < equationSize; i++) {
            this.solutions[i] = equations[i][nElements - 1];
        }

        //Setting Model as trained
        this.isFit = true;

        //Setting error
        this.calculateR2(xArray, yArray);
    }
```

función predict utilizado para obtener los resultados

```
//Function that creates a prediction based in the regression model
predict(xArray) {
    let yArray = [];
    //Checking if the model is already trained
    if (this.isFit) {
        //Generating the predictions based in the input and solutions
        for (let i = 0; i < xArray.length; i++) {
            let yprediction = 0;
            for (let j = 0; j < this.solutions.length; j++) {
                yprediction += this.solutions[j] * Math.pow(xArray[i], j);
            }
            yArray.push(yprediction);
        }
    }

    //Returning Prediction
    return yArray;
}
```

Función calculateR2 utilizado para guardar errores para el array entrenado

```
//Method that stores error for the trained array
calculateR2(xArray, yArray) {
    //Setting error array and predictions
    let errors = new Array(xArray.length);
    let prediction = this.predict(xArray);
    let sumY = 0;

    //Calculating errors
    for (let i = 0; i < xArray.length; i++) {
        sumY += yArray[i];
        errors[i] = Math.pow(yArray[i] - prediction[i], 2);
    }

    let sr = 0;
    let st = 0;
    for (let i = 0; i < xArray.length; i++) {
        sr += errors[i];
        st += Math.pow(yArray[i] - (sumY / xArray.length), 2);
    }
    let r2 = (st - sr) / st;
    this.error = r2;
}

getError() {
    return this.error;
}
```

Linear Model Script para manejar los datos y mostrar resultados

```
import { parseCSV, getXTrain, getYTrain } from './CsvParser.js';

document.getElementById('csvFile').addEventListener('change', (event) => {
    const algorithm = document.getElementById('algorithm').value;
    parseCSV(event.target.files[0], algorithm);
});

document.getElementById('trainModel').addEventListener('click', () => {
    const algorithm = document.getElementById('algorithm').value;
    if (algorithm !== 'linear') return;

    const xTrain = getXTrain();
    const yTrain = getYTrain();

    if (xTrain.length === 0 || yTrain.length === 0) {
        alert("Please upload a valid CSV file and try again.");
        return;
    }

    const model = new LinearRegression();
    model.fit(xTrain, yTrain);
    document.getElementById('results').innerText = `Linear Model trained with m = ${model.m} and b = ${model.b}`;
});

document.getElementById('makePredictions').addEventListener('click', () => {
    const algorithm = document.getElementById('algorithm').value;
    if (algorithm !== 'linear') return;

    const xTrain = getXTrain();
    const yTrain = getYTrain();

    if (xTrain.length === 0) {
        alert("Please train the model before making predictions.");
        return;
    }


    const model = new LinearRegression();
    model.fit(xTrain, yTrain);
    const xTest = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    const predictions = model.predict(xTest);

    document.getElementById('results').innerText += `\nPredictions:\n${predictions.map((value, index) => `${index + 1} --- ${value}`).join('

    const ctx = document.getElementById('predictionChart').getContext('2d');
    new Chart(ctx, {
        type: 'line',
        data: {
            labels: xTest,
            datasets: [
                {
                    label: `Linear Regression Predictions`,
                    data: predictions,
                    borderColor: 'rgba(75, 192, 192, 1)',
                    backgroundColor: 'rgba(75, 192, 192, 0.2)',
                    fill: true,
                }
            ]
        },
        options: {
            scales: {
                x: { title: { display: true, text: 'X Values' } },
                y: { title: { display: true, text: 'Predicted Y Values' } }
            }
        }
    });
});
```

Polinomial Model Script para manejar datos y retornar resultados

```
import { parseCSV, getXTrain, getYTrain } from './CsvParser.js';

document.getElementById('csvFile').addEventListener('change', (event) => {
    const algorithm = document.getElementById('algorithm').value;
    parseCSV(event.target.files[0], algorithm);
});

document.getElementById('trainModel').addEventListener('click', () => {
    const algorithm = document.getElementById('algorithm').value;
    if (algorithm !== 'polynomial') return;
    const xTrain = getXTrain();
    const yTrain = getYTrain();

    if (xTrain.length === 0 || yTrain.length === 0) {
        alert("Please upload a valid CSV file and try again.");
        return;
    }

    const degree = prompt("Enter the degree for the Polynomial Regression:", "2");
    const model = new PolynomialRegression();
    model.fit(xTrain, yTrain, parseInt(degree));
    document.getElementById('results').innerText = `Polynomial Model trained with R² = ${model.getError()}`;
});

document.getElementById('makePredictions').addEventListener('click', () => {
    const algorithm = document.getElementById('algorithm').value;
    if (algorithm !== 'polynomial') return;

    const xTrain = getXTrain();
    const yTrain = getYTrain();

    if (xTrain.length === 0) {
        alert("Please train the model before making predictions.");
        return;
    }

    const degree = prompt("Enter the degree for the Polynomial Regression (used in training):", "2");
    const model = new PolynomialRegression();
    model.fit(xTrain, yTrain, parseInt(degree));
    const xTest = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    const predictions = model.predict(xTest);

    document.getElementById('results').innerText += `\nPredictions:\n${predictions.map((value, index) => `${index + 1} --- ${value}`).join('

    const ctx = document.getElementById('predictionChart').getContext('2d');
    new Chart(ctx, {
        type: 'line',
        data: {
            labels: xTest,
            datasets: [
                {
                    label: `Polynomial Regression Predictions`,
                    data: predictions,
                    borderColor: 'rgba(153, 102, 255, 1)',
                    backgroundColor: 'rgba(153, 102, 255, 0.2)',
                    fill: true,
                }
            ]
        },
        options: {
            scales: {
                x: { title: { display: true, text: 'X Values' } },
                y: { title: { display: true, text: 'Predicted Y Values' } }
            }
        }
    });
});
```

Conclusión

El proyecto se estructuró y acopló de manera eficiente a los requerimientos establecidos, logrando una solución sólida y funcional. Gracias a la implementación del enfoque modular y la correcta separación de responsabilidades, se obtuvo un código más organizado y fácil de mantener, permitiendo una gestión más eficaz de las funciones principales, como la carga de datos, el entrenamiento de modelos y la visualización de resultados. La utilización de bibliotecas como Chart.js contribuyó significativamente a la representación gráfica de las predicciones, proporcionando un análisis visual claro y dinámico de los datos procesados. En conjunto, este enfoque garantiza un mejor entendimiento y manejo del proyecto, optimizando la experiencia del usuario y facilitando futuras ampliaciones del sistema.