UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

INTELIGENCIA ARTIFICIAL 1

ING. Luis Fernando Espino

ESTUDIANTE

Christian Alessander Blanco González

202000173

Manual Técnico del Proyecto de Machine Learning

Índice

Requisitos del Sistema

Hardware Recomendado:

1. **Procesador**: Intel i5 o superior / AMD Ryzen 5 o superior (se recomienda un procesador de 4 núcleos o más para manejar múltiples scripts y archivos grandes).
2. **RAM**: 8 GB mínimo (idealmente 16 GB o más para optimizar el procesamiento de datos y el rendimiento en aplicaciones gráficas).
3. **Almacenamiento**: 500 MB de espacio libre como mínimo (para scripts y dependencias), aunque se recomienda un SSD con al menos 10 GB para agilizar el acceso y lectura de archivos grandes.
4. **Tarjeta gráfica**: No es obligatoria, pero una GPU compatible puede acelerar el procesamiento de gráficos si se utilizan grandes cantidades de datos.

Software Requerido:

1. **Sistema Operativo**: Windows 10/11, macOS 10.15 o superior, o cualquier distribución de Linux compatible con Node.js y navegadores modernos.

2. **Node.js**: Para ejecutar cualquier servidor local, si es necesario, y para gestionar las dependencias de JavaScript.
3. **Navegador Moderno**: Google Chrome, Firefox, Safari, o Microsoft Edge para visualizar el proyecto y los gráficos generados.
4. **Editor de Texto**: En tu caso, has utilizado **Visual Studio Code** para el desarrollo del proyecto, que permite la edición de HTML, CSS, y JavaScript con soporte de extensiones para facilitar el desarrollo.

Aplicaciones Utilizadas

1. **Visual Studio Code (VS Code)**:
    - **Extensiones Recomendadas**:
        - **Live Server**: Para visualizar cambios en tiempo real y servir el proyecto en un servidor local.
        - **Prettier** o **ESLint**: Para mantener el formato y consistencia del código JavaScript.
        - **GitLens** (si usas Git): Ayuda a gestionar versiones de código dentro de VS Code.
2. **Google Charts**: Utilizado en el código para la visualización de datos de los modelos de regresión lineal y polinomial.
3. **Vis.js**: Usado para visualizar el árbol de decisión ID3 en formato gráfico.
4. **Bibliotecas y Herramientas de JavaScript (si se utilizan en `tytus.js`)**:
    - `math.js` (opcional): Para operaciones matemáticas avanzadas en el caso de cálculos polinomiales y regresiones.

---

1. `index.html`

Este archivo HTML actúa como una página de inicio que permite la navegación hacia los modelos de Machine Learning. Cada botón redirige a una página específica del modelo, como regresión lineal o polinomial, árboles de decisión, KMeans, y KNN.

2. `linearRegression.html`

Esta página se centra en la Regresión Lineal, permitiendo al usuario cargar datos, entrenar el modelo y visualizar los resultados en un gráfico.

Código JavaScript Explicado

```
// Variables de entrenamiento
let dataset = [];
let m = 0, b = 0;
```

Aquí se declaran dos variables principales para el modelo de regresión lineal:

- `dataset`: Almacena los datos cargados del archivo CSV.

- `m` y `b`: Representan la pendiente y la intersección de la línea de regresión.

```
document.getElementById('csvFile').addEventListener('change', function(event) {
    const file = event.target.files[0];
    if (file) {
        const reader = new FileReader();
        reader.onload = function(e) {
            const lines = e.target.result.split('\n');
            dataset = lines.map(line => line.split(',').map(Number));
            trainModel();
        };
        reader.readAsText(file);
    }
});
```

En este bloque:

1. Se activa el evento `change` cuando el usuario selecciona un archivo.
2. **FileReader** lee el archivo CSV y procesa cada línea, convirtiéndola en números que se almacenan en `dataset`.
3. Finalmente, llama a la función `trainModel()` para iniciar el entrenamiento.

```
function trainModel() {
    let sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0;
    const n = dataset.length;

    dataset.forEach(([x, y]) => {
        sumX += x;
        sumY += y;
        sumXY += x * y;
        sumX2 += x * x;
    });

    m = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);
    b = (sumY - m * sumX) / n;
    drawChart();
}
```

Esta función `trainModel` calcula la pendiente (`m`) y la intersección (`b`) utilizando mínimos cuadrados:

- **sumX, sumY, sumXY, sumX2**: Calculan la suma de `X`, `Y`, `X*Y`, y `X²`.
- Los cálculos de `m` y `b` se basan en fórmulas de mínimos cuadrados, definiendo la línea de mejor ajuste para los puntos.

```
function drawChart() {
    const data = new google.visualization.DataTable();
    data.addColumn('number', 'X');
    data.addColumn('number', 'Y');
```

```
    data.addColumn('number', 'Predicción');

    const chartData = dataset.map(([x, y]) => [x, y, m * x + b]);
    data.addRows(chartData);

    const options = {
        title: 'Regresión Lineal',
        hAxis: { title: 'X' },
        vAxis: { title: 'Y' },
        legend: 'none'
    };

    const chart = new
google.visualization.LineChart(document.getElementById('graphContainer'));
    chart.draw(data, options);
}
```

La función `drawChart` utiliza Google Charts para dibujar el gráfico:

- **data.addColumn**: Define las columnas `X`, `Y` y `Predicción`.
- **chartData**: Agrega los puntos `(x, y)` y calcula la predicción `m * x + b`.
- **chart.draw**: Genera el gráfico, configurado para mostrar la línea de regresión y los puntos originales.

---

3. `polynomialRegression.html`

Esta página aplica regresión polinomial, donde los datos cargados son ajustados a una función polinómica.

Código JavaScript Explicado

```
let data = [];
let coefficients = [];
```

Las variables principales son:

- `data`: Almacena los datos cargados.
- `coefficients`: Guardará los coeficientes calculados del polinomio.

```
document.getElementById('csvInput').addEventListener('change', function(event) {
    const file = event.target.files[0];
    if (file) {
        const reader = new FileReader();
        reader.onload = function(e) {
            const lines = e.target.result.split('\n');
            data = lines.map(line => line.split(',').map(Number));
            trainPolynomialRegression();
        };
        reader.readAsText(file);
```

```
    }
});
```

El archivo CSV es procesado y cada línea se convierte en un arreglo de números para ser almacenado en `data`. Luego, se llama a `trainPolynomialRegression()` para entrenar el modelo.

```
function trainPolynomialRegression() {
    coefficients = calculatePolynomialCoefficients(data);
    drawGraph();
}
```

En `trainPolynomialRegression`, los coeficientes se calculan usando una función (asumida como implementada en `tytus.js`) llamada `calculatePolynomialCoefficients`, que devuelve los coeficientes de la función polinómica ajustada a los datos.

```
function drawGraph() {
    const dataTable = new google.visualization.DataTable();
    dataTable.addColumn('number', 'X');
    dataTable.addColumn('number', 'Y');
    dataTable.addColumn('number', 'Predicción');

    data.forEach(([x, y]) => {
        const prediction = coefficients.reduce((acc, coef, power) => acc + coef *
Math.pow(x, power), 0);
        dataTable.addRow([x, y, prediction]);
    });

    const options = {
        title: 'Regresión Polinomial',
        hAxis: { title: 'X' },
        vAxis: { title: 'Y' },
        legend: 'none'
    };

    const chart = new
google.visualization.LineChart(document.getElementById('graphContainer'));
    chart.draw(dataTable, options);
}
```

La función `drawGraph` usa Google Charts para dibujar la curva de regresión polinomial:

- Calcula la predicción para cada `x` usando los coeficientes obtenidos.
- **dataTable.addRow**: Inserta cada punto `(x, y)` y la predicción.
- **chart.draw**: Dibuja el gráfico de los puntos originales y la curva de ajuste.

---

4. `decisionTree.html`

Esta página implementa un árbol de decisión ID3, permitiendo cargar datos en JSON y clasificar elementos nuevos.

Código JavaScript Explicado

```
let dataset = [];
```

La variable `dataset` almacenará los datos JSON cargados.

```
document.getElementById('jsonFile').addEventListener('change', function(event) {
   const file = event.target.files[0];
   if (file) {
      const reader = new FileReader();
      reader.onload = function(e) {
         dataset = JSON.parse(e.target.result);
         trainDecisionTree();
      };
      reader.readAsText(file);
   }
});
```

Este bloque:

1. Carga el archivo JSON.
2. **JSON.parse** convierte el contenido en un objeto JavaScript y lo almacena en `dataset`.
3. Llama a `trainDecisionTree` para entrenar el árbol.

```
function trainDecisionTree() {
   const tree = new DecisionTreeID3(dataset);
   tree.train();
   visualizeTree(tree);
}
```

Aquí:

- Crea una instancia del árbol `DecisionTreeID3`.
- Llama a `train()` en la instancia del árbol para entrenarlo.
- Luego, `visualizeTree(tree)` renderiza el árbol.

```
function visualizeTree(tree) {
   const dotString = tree.generateDotString();
   renderDot(dotString);
}
```

La función `visualizeTree` utiliza `generateDotString` para convertir el árbol en un formato visualizable (formato Dot para graficación) y `renderDot` muestra el gráfico.

---

5. `kmeans.html`

La página permite usar KMeans para agrupar datos, configurando el número de clusters y entrenando el modelo.

Código JavaScript Explicado

```
let data = [], clusters = 3, kmeans;

document.getElementById('data_file').addEventListener('change', function(event) {
   const file = event.target.files[0];
   if (file) {
      const reader = new FileReader();
      reader.onload = function(e) {
         data = parseCSV(e.target.result);
         trainKMeans

();
      };
      reader.readAsText(file);
   }
});
```

Este código:

1. Carga un archivo y procesa el contenido usando `parseCSV`, transformando el archivo en un arreglo `data`.
2. Llama a `trainKMeans` para realizar el clustering.

```
function trainKMeans() {
   kmeans = new KMeans(clusters, data);
   kmeans.fit();
   visualizeClusters(kmeans);
}
```

En `trainKMeans`:

- Se inicializa `KMeans` con el número de clusters y los datos.
- `fit()` agrupa los datos y `visualizeClusters` muestra los resultados.

---

6. `knn.html`

Esta página implementa K-Nearest Neighbor (KNN), cargando datos y evaluando puntos nuevos.

Código JavaScript Explicado

```
let knnData = [], k = 5;

document.getElementById('jsonFileInput').addEventListener('change',
function(event) {
   const file = event.target.files[0];
   if (file) {
      const reader = new FileReader();
      reader.onload = function(e) {
         knnData = JSON.parse(e.target.result);
      };
      reader.readAsText(file);
   }
});
```

Este fragmento:

1. Carga datos JSON y los almacena en `knnData`.
2. Los datos se procesarán para realizar clasificaciones con KNN.

```
function classifyPoint(point) {
   const knn = new KNearestNeighbor(knnData, k);
   const prediction = knn.predict(point);
   displayResult(prediction);
}
```

La función `classifyPoint`:

1. Crea una instancia de `KNearestNeighbor`.
2. Usa `predict` para clasificar `point` y llama a `displayResult` para mostrar el resultado.

---

7. `tytus.js`

Este archivo contiene las clases y algoritmos, como `DecisionTreeID3`, `KMeans`, y `KNearestNeighbor`, que los modelos utilizan para procesamiento y cálculos de Machine Learning.