Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Inteligencia Artificial 1
Sección A

Manual Técnico

Juan Pablo García Ceballos
201901598

Lugar y Fecha: Guatemala, Sacatepéquez 31/10/2024

REGRESIÓN LINEAL

La **regresión lineal** es un modelo estadístico que permite analizar cómo una variable independiente (X) afecta a una variable dependiente (Y).

- **X**: Variable independiente o exógena.
- **Y**: Variable dependiente o endógena.

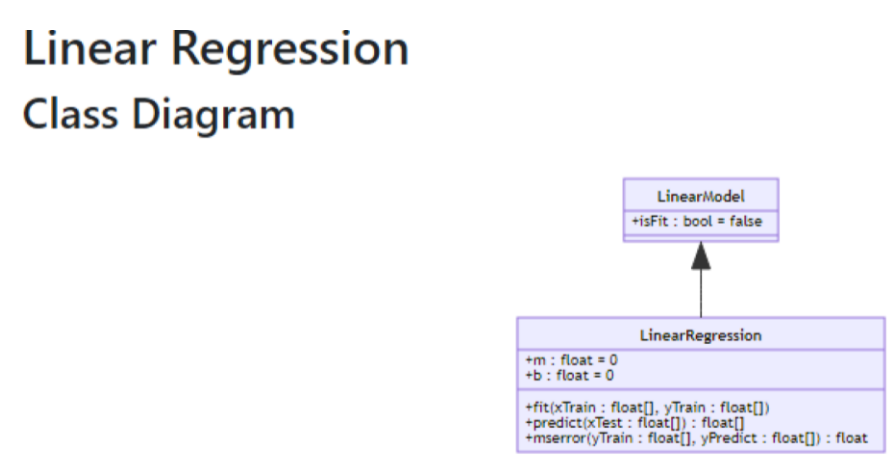
El propósito de este modelo es proporcionar estimaciones razonables para Y en función de distintos valores de X, usando una muestra de pares de datos (x_1, y_1), …, (xn, yn).

Ejemplos prácticos de aplicación de la regresión lineal:

- Estudiar cómo la altura del padre puede influir en la altura de los hijos.
- Calcular el precio de una vivienda basado en su tamaño.
- Relacionar la tasa de mortalidad con el consumo medio de cigarrillos.
- Analizar la abundancia de un parásito según la temperatura.
- Aproximar la nota en una materia a partir del tiempo semanal de estudio.
- Estimar el tiempo de ejecución de un programa según la velocidad del procesador.

Objetivos al aprender y comprender la regresión lineal:

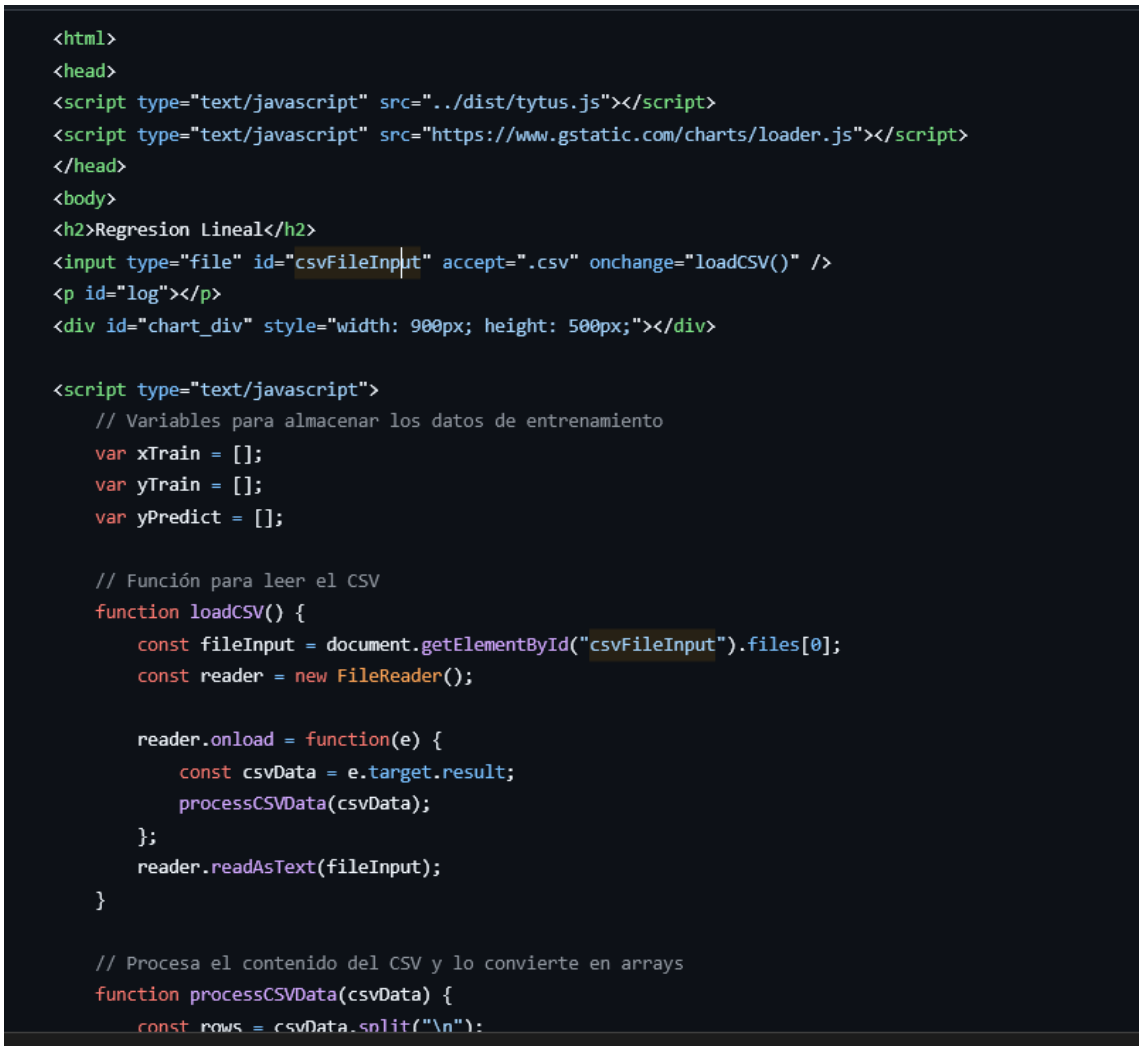
- Construir un modelo de regresión lineal simple para describir cómo X influye en Y.
- Obtener estimaciones puntuales de los parámetros de un modelo de regresión.
- Calcular el valor promedio de Y para cualquier valor dado de X.
- Predecir valores futuros de la variable dependiente Y.



Código de Regresión Linea regresión_lineal.html

1. Descripción General

Este código implementa una regresión lineal simple en JavaScript, utilizando una librería externa de regresión (tytus.js) y la API de Google Charts para la visualización gráfica. La aplicación permite al usuario cargar un archivo CSV, procesar los datos de entrenamiento (variables independientes xTrain y dependientes yTrain), calcular una predicción (yPredict) con un modelo de regresión lineal y finalmente visualizar los resultados en un gráfico.



```
<html>
<head>
<script type="text/javascript" src="../dist/tytus.js"></script>
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
</head>
<body>
<h2>Regresion Lineal</h2>
<input type="file" id="csvFileInput" accept=".csv" onchange="loadCSV()" />
<p id="log"></p>
<div id="chart_div" style="width: 900px; height: 500px;"></div>

<script type="text/javascript">
    // Variables para almacenar los datos de entrenamiento
    var xTrain = [];
    var yTrain = [];
    var yPredict = [];

    // Función para leer el CSV
    function loadCSV() {
        const fileInput = document.getElementById("csvFileInput").files[0];
        const reader = new FileReader();

        reader.onload = function(e) {
            const csvData = e.target.result;
            processCSVData(csvData);
        };
        reader.readAsText(fileInput);
    }

    // Procesa el contenido del CSV y lo convierte en arrays
    function processCSVData(csvData) {
        const rows = csvData.split("\n");
```

2. Estructura del Código

1. **HTML Básico**:
    - Define el título "Regresión Lineal" y un selector de archivos (input de tipo file) para cargar el CSV con los datos.
    - Incluye dos scripts externos:
        - tytus.js: Una biblioteca de regresión lineal para crear y entrenar el modelo.

        - loader.js de Google Charts: Permite cargar y visualizar gráficos.

2. **Variables Globales**:
    - xTrain: Almacena los valores de entrada (variable independiente) del CSV.
    - yTrain: Almacena los valores de salida (variable dependiente) del CSV.
    - yPredict: Contiene las predicciones generadas por el modelo de regresión lineal.

3. Funciones Principales

- **loadCSV()**:
    - Se ejecuta al cargar un archivo CSV. Obtiene el archivo y lee su contenido en texto mediante FileReader.
    - Cuando la lectura termina (reader.onload), llama a processCSVData() para procesar los datos.
- **processCSVData(csvData)**:
    - Convierte el contenido del CSV en un arreglo de filas (rows).
    - Itera sobre cada fila (ignorando la primera si tiene encabezados) y extrae los valores de las columnas:
        - La primera columna representa xTrain y la segunda yTrain.
    - Llama a performLinearRegression() una vez que se completan las asignaciones de xTrain y yTrain.
- **performLinearRegression()**:
    - Crea un modelo de regresión lineal utilizando LinearRegression() de la librería tytus.js.
    - Entrena el modelo con los datos de entrenamiento (xTrain, yTrain) mediante fit().
    - Realiza predicciones para los datos de xTrain usando predict() y almacena los resultados en yPredict.
    - Actualiza el elemento HTML log para mostrar los valores de xTrain, yTrain y yPredict.
    - Llama a drawChart() para generar el gráfico.
- **drawChart()**:
    - Usa la API de Google Charts para crear y dibujar un gráfico de dispersión (scatter plot) con una línea de tendencia.

- Convierte los datos de entrenamiento y predicción en un arreglo estructurado (dataArray) adecuado para Google Charts.
- Configura el gráfico como un ComboChart con una dispersión (scatter) y una línea (line) que representa yPredict.
- Dibuja el gráfico en el elemento chart_div.

4. Flujo de Trabajo

1. **Carga de Datos**:
   - El usuario selecciona un archivo CSV. Al detectar el archivo, loadCSV() lo lee y llama a processCSVData().
2. **Procesamiento de Datos**:
   - processCSVData() convierte el contenido del CSV en arreglos xTrain y yTrain, los cuales son pasados al modelo de regresión lineal en performLinearRegression().
3. **Regresión Lineal y Predicción**:
   - performLinearRegression() crea el modelo, ajusta los datos y genera las predicciones yPredict.
4. **Visualización**:
   - drawChart() dibuja un gráfico combinando los valores reales (yTrain) y las predicciones (yPredict) en una misma visualización.

5. Requisitos Previos y Dependencias

- **Librerías externas**:
  - tytus.js: Biblioteca que contiene la clase LinearRegression utilizada para entrenar el modelo y hacer predicciones.
  - https://www.gstatic.com/charts/loader.js: Cargador de Google Charts para visualizar los datos en un gráfico.

REGRESIÓN POLINOMIAL

La **regresión polinomial** es un método de predicción para una variable de respuesta cuantitativa en función de una variable predictora cuantitativa, donde la relación entre ellas se representa mediante una función polinomial de grado nnn.

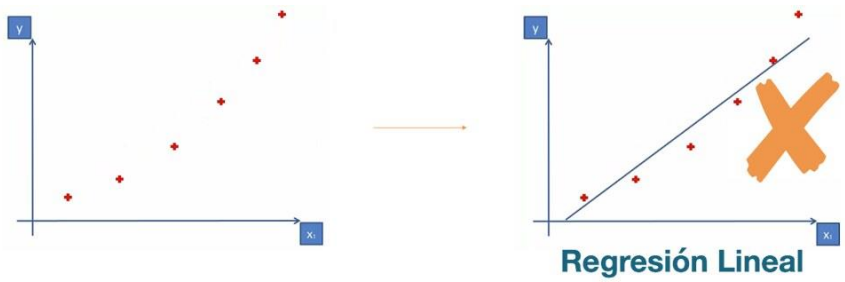
A diferencia de la regresión lineal, este modelo permite captar relaciones más complejas entre las variables mediante términos elevados a potencias mayores que uno (por ejemplo, x^2, x^3, etc.). Para calcular la regresión polinomial, se suelen emplear métodos de mínimos cuadrados, resolviendo incógnitas relacionadas con errores como el error absoluto y el error cuadrático. Esto implica definir una función objetivo adecuada, y se puede implementar en un lenguaje de programación de libre elección para mostrar los polinomios de mínimos cuadrados y sus valores de aproximación.

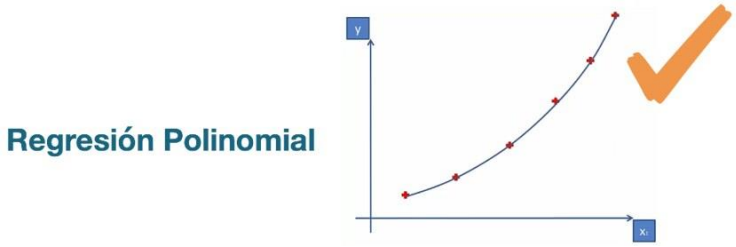
Objetivo: El objetivo es encontrar una función polinómica que se ajuste lo mejor posible a los datos proporcionados, de modo que la curva generada represente bien el comportamiento de la variable de respuesta. En este contexto, el error absoluto se define como la suma de las diferencias al cuadrado entre la función modelada y la función polinómica ajustada, aplicando este cálculo a los valores de la variable independiente.

Ecuación de la regresión polinomial:

$$y=a+bx+cx^2+dx^3+\dots$$

Aquí, los coeficientes a,b,c,d,a, b, c, d,a,b,c,d, etc., representan los pesos de cada término de xxx en el polinomio, que se ajustan para minimizar el error y proporcionar la mejor aproximación posible.

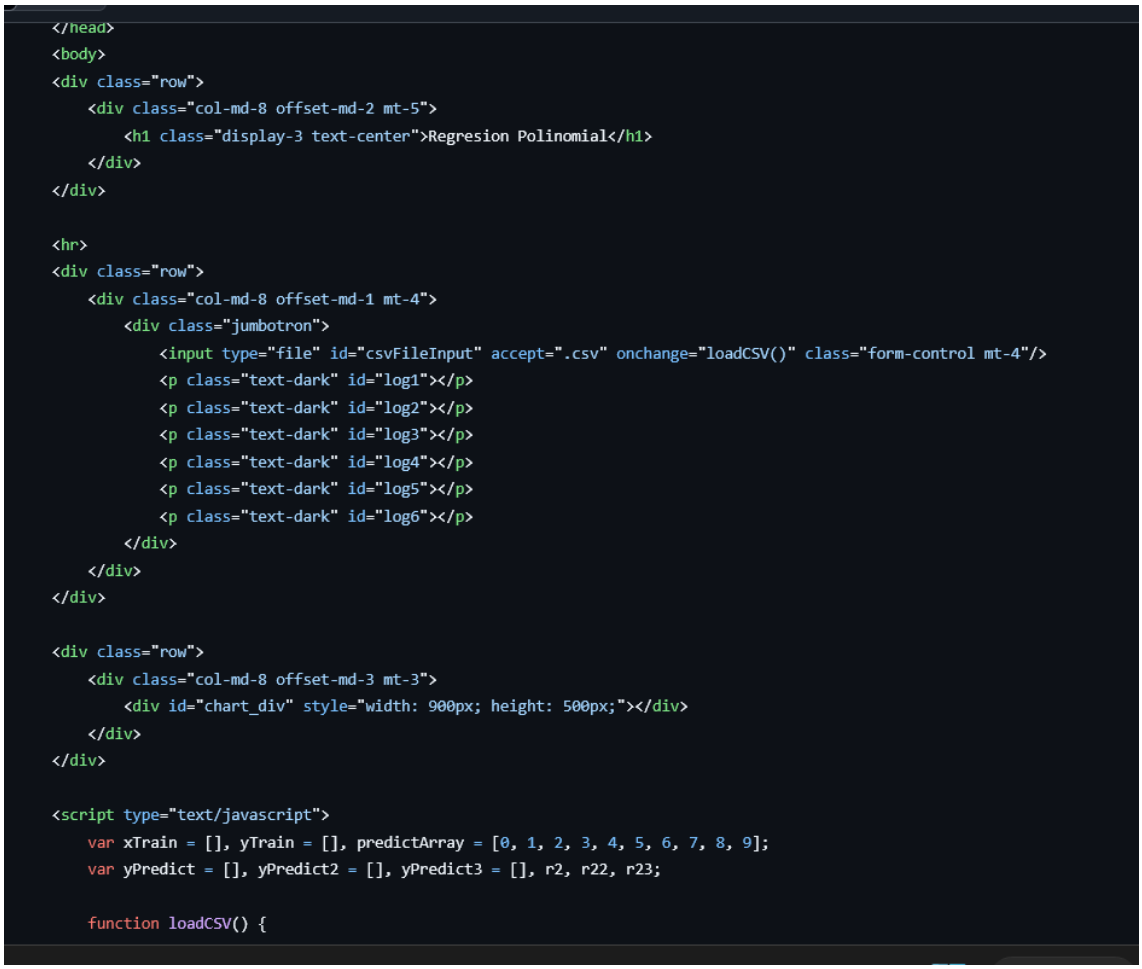




Código de Regresión Polinomial

1. Descripción General

Este código implementa una regresión polinomial en JavaScript para un conjunto de datos cargado desde un archivo CSV. Se utiliza la librería externa tytus.js para crear modelos de regresión polinomial de distintos grados y Google Charts para la visualización de los resultados. Además, incluye Bootstrap para un diseño estructurado y estilizado de la interfaz de usuario.



```
</head>
<body>
<div class="row">
    <div class="col-md-8 offset-md-2 mt-5">
        <h1 class="display-3 text-center">Regresion Polinomial</h1>
    </div>
</div>

<hr>
<div class="row">
    <div class="col-md-8 offset-md-1 mt-4">
        <div class="jumbotron">
            <input type="file" id="csvFileInput" accept=".csv" onchange="loadCSV()" class="form-control mt-4"/>
            <p class="text-dark" id="log1"></p>
            <p class="text-dark" id="log2"></p>
            <p class="text-dark" id="log3"></p>
            <p class="text-dark" id="log4"></p>
            <p class="text-dark" id="log5"></p>
            <p class="text-dark" id="log6"></p>
        </div>
    </div>
</div>

<div class="row">
    <div class="col-md-8 offset-md-3 mt-3">
        <div id="chart_div" style="width: 900px; height: 500px;"></div>
    </div>
</div>

<script type="text/javascript">
    var xTrain = [], yTrain = [], predictArray = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
    var yPredict = [], yPredict2 = [], yPredict3 = [], r2, r22, r23;

    function loadCSV() {
```

2. Estructura del Código

1. **HTML Básico**:
    - Incluye encabezados de estilo y scripts:
        - tytus.js: Proporciona la clase PolynomialRegression.
        - loader.js de Google Charts: Para visualización gráfica.
        - bootstrap.min.css y bootstrap.bundle.min.js: Para aplicar estilos y componentes de interfaz de usuario.
2. **Interfaz de Usuario**:

- Se organiza el contenido en contenedores div de Bootstrap para mostrar el título, el campo de carga de archivo y la visualización de gráficos.
- input de tipo file: Permite al usuario cargar el archivo CSV.
- Varias etiquetas <p> (log1 a log6) para mostrar los resultados de entrenamiento y predicción.

3. **Variables Globales**:
   - xTrain: Valores de la variable independiente (eje X).
   - yTrain: Valores de la variable dependiente (eje Y).
   - predictArray: Valores de X sobre los cuales se realizarán las predicciones.
   - yPredict, yPredict2, yPredict3: Resultados de las predicciones para polinomios de grado 2, 3 y 4.
   - r2, r22, r23: Valores de error R^2 para los modelos de cada grado.

3. Funciones Principales

- **loadCSV()**:
  - Se ejecuta al cargar el archivo CSV. Utiliza FileReader para leer el archivo seleccionado y llama a processCSVData() cuando termina de leer.
- **processCSVData(csvData)**:
  - Convierte el contenido del CSV en arreglos xTrain y yTrain.
  - Itera sobre cada fila del archivo CSV (ignorando la primera si tiene encabezados) y asigna las columnas a xTrain y yTrain.
  - Una vez procesados los datos, llama a performPolynomialRegression().
- **performPolynomialRegression()**:
  - Crea una instancia de PolynomialRegression.
  - Entrena y ajusta el modelo de regresión polinomial para grados 2, 3 y 4:
    - polynomial.fit(xTrain, yTrain, grado): Ajusta el modelo para cada grado.
    - polynomial.predict(predictArray): Genera las predicciones para el conjunto de datos predictArray.
    - polynomial.getError(): Obtiene el error R^2 para el ajuste de cada grado.

- Muestra en la interfaz los resultados de xTrain, yTrain, yPredict, yPredict2, yPredict3 y los valores de R^2 en los elementos log1 a log6.
- Llama a drawChart() para mostrar el gráfico.

- **drawChart()**:
  - Utiliza Google Charts para generar un gráfico de dispersión con líneas de predicción para los distintos grados.
  - Convierte los datos en un formato adecuado (dataArray) para Google Charts.
  - Configura el gráfico como ComboChart con dispersión para yTrain y líneas para yPredict, yPredict2 y yPredict3.
  - Dibuja el gráfico en el elemento chart_div.

4. Flujo de Trabajo

1. **Carga de Datos**:
   - El usuario selecciona un archivo CSV, loadCSV() lo lee y llama a processCSVData() para procesar el contenido.
2. **Procesamiento de Datos**:
   - processCSVData() convierte el CSV en los arreglos xTrain y yTrain, los cuales luego se usan para crear y ajustar el modelo polinomial en performPolynomialRegression().
3. **Regresión Polinomial y Predicción**:
   - performPolynomialRegression() ajusta el modelo polinomial para tres grados (2, 3, y 4), obteniendo los valores de predicción y el error R^2 para cada uno.
4. **Visualización**:
   - drawChart() genera un gráfico que muestra los puntos de entrenamiento (yTrain) y las líneas de predicción para los tres grados de regresión.

5. Requisitos y Dependencias

- **Librerías externas**:
  - tytus.js: Biblioteca que contiene la clase PolynomialRegression para el entrenamiento del modelo y cálculo de errores.
  - https://www.gstatic.com/charts/loader.js: Librería de Google Charts para visualización de datos.
  - Bootstrap: Para estilizar y estructurar la interfaz de usuario

ÁRBOLES DE DECISIÓN

Los **árboles de decisión** son herramientas visuales que representan posibles resultados de una serie de decisiones interrelacionadas, permitiendo comparar opciones según sus costos, probabilidades y beneficios. Este modelo facilita tanto la toma de decisiones en organizaciones como la creación de algoritmos que matemáticamente determinen la opción más favorable.

Un árbol de decisión suele comenzar con un **nodo inicial** y se ramifica en múltiples resultados posibles. Cada resultado puede llevar a otros nodos adicionales que continúan ramificándose, dándole una estructura similar a la de un árbol.

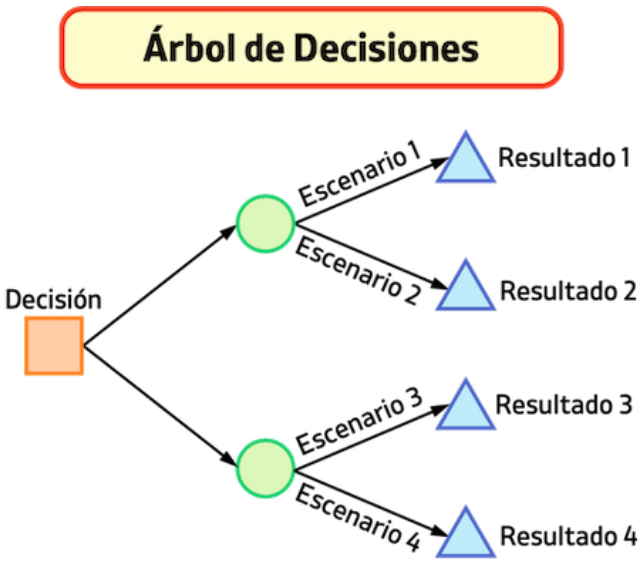
Tipos de Nodos en Árboles de Decisión

1. **Nodo de probabilidad**: Representado con un **círculo** y utilizado para indicar las probabilidades de diferentes resultados en un punto específico.
2. **Nodo de decisión**: Representado con un **cuadrado** e indica un punto en el que se toma una decisión.
3. **Nodo terminal**: Representado al final de cada ruta, muestra el **resultado final** de una secuencia de decisiones.

Simbología de los Árboles de Decisión

- **Círculo**: Nodo de probabilidad.
- **Cuadrado**: Nodo de decisión.
- **Triángulo o Nodo Terminal**: Representa el fin de una rama con el resultado final del análisis o decisión.

Este enfoque visual y estructurado ayuda a evaluar decisiones complejas, presentando todas las alternativas y consecuencias posibles en un formato claro y fácil de seguir.



Código de Árbol de Decisión

1. Descripción General

Este código implementa un árbol de decisión basado en el algoritmo ID3, utilizando JavaScript y la biblioteca tytus.js para el entrenamiento del modelo, y vis-network para visualizar el árbol. La interfaz permite al usuario ingresar un conjunto de datos en formato JSON, entrenar el árbol de decisión, y luego visualizar la estructura del árbol resultante junto con predicciones para datos nuevos.

```
<body>
    <h1 class="display-1 w-100 text-center">Árbol de decisión</h1>
    <div class="container">
        <h3 class="display-6 mt-5">Conjunto de datos utilizados:<br /></h3>
        <textarea id="dataInput" class="form-control" rows="10">[
            ["Outlook", "Temperature", "Humidity", "Windy", "Play Tennis"],
            ["Sunny", "Hot", "High", "Weak", "No"],
            ["Sunny", "Hot", "High", "Strong", "No"],
            ["Overcast", "Hot", "High", "Weak", "Yes"],
            ["Rainy", "Mild", "High", "Weak", "Yes"],
            ["Rainy", "Cool", "Normal", "Weak", "Yes"],
            ["Rainy", "Cool", "Normal", "Strong", "No"],
            ["Overcast", "Cool", "Normal", "Strong", "Yes"],
            ["Sunny", "Mild", "High", "Weak", "No"],
            ["Sunny", "Cool", "Normal", "Weak", "Yes"],
            ["Rainy", "Mild", "Normal", "Weak", "Yes"],
            ["Sunny", "Mild", "Normal", "Strong", "Yes"],
            ["Overcast", "Mild", "High", "Strong", "Yes"],
            ["Overcast", "Hot", "Normal", "Weak", "Yes"],
            ["Rainy", "Mild", "High", "Strong", "No"]
        ]</textarea>

        <!-- Botones para actualizar y mostrar el árbol -->
        <button id="updateButton" class="btn btn-primary mt-3">Actualizar Árbol</button>
        <button id="showButton" class="btn btn-secondary mt-3">Mostrar Árbol</button>

        <h3 class="display-6 mt-5">Datos para predecir:<br /></h3>
        <textarea id="predictInput" class="form-control" rows="5">[
            ["Outlook", "Temperature", "Humidity", "Windy"],
            ["Overcast", "Cool", "Normal", "Strong"]
        ]</textarea>

        <h3 class="display-6 mt-5">Resultado de la predicción:<br /></h3>
        <p id="result" class="bg-light p-5 text-muted" style="border-radius: 10px;">
```

2. Estructura del Código

1. **HTML Básico**:
    - Se incluyen los encabezados para las bibliotecas:
        - vis-network: Biblioteca para visualizar redes y árboles.
        - tytus.js: Librería personalizada para crear el árbol de decisión y el algoritmo ID3.
        - Bootstrap: Para el diseño y estructura de la interfaz.

- Contiene un área para ingresar datos en formato JSON, un área de texto para visualizar el resultado de la predicción, y un contenedor div para la visualización del árbol de decisión.

2. **CSS**:
    - Se utiliza para estilizar los nodos y bordes del árbol (.vis-node y .vis-edge), así como el contenedor del árbol (#tree), con bordes y colores específicos.
3. **Interfaz de Usuario**:
    - La interfaz se organiza con Bootstrap en una disposición centrada.
    - **Área de Datos**:
        - textarea (dataInput): Permite ingresar un conjunto de datos en formato JSON para entrenar el árbol de decisión.
        - **Botones**:
            - Actualizar Árbol: Permite actualizar el árbol de decisión basado en el conjunto de datos ingresado.
            - Mostrar Árbol: Muestra el árbol visualmente.
    - **Predicción**:
        - textarea (predictInput): Permite ingresar los datos de entrada para los que se desea obtener una predicción.
        - p (result): Muestra el resultado de la predicción.
    - **Visualización del Árbol**:
        - div (tree): Contenedor donde se dibuja el árbol de decisión.

3. Funciones Principales

- **drawTree()**:
    - Obtiene los datos de entrenamiento (dataInput) y los datos de predicción (predictInput) del árbol.
    - Convierte los datos de texto JSON en objetos utilizables en JavaScript mediante JSON.parse.
    - Crea una instancia de DecisionTreeID3 utilizando los datos de entrenamiento (dtSt).
    - Entrena el árbol con el método train y obtiene el nodo raíz (root).
    - Utiliza el árbol entrenado para predecir el resultado con los datos de predicción (predictData).
    - Actualiza el elemento result para mostrar el resultado de la predicción.

- Genera el árbol visual:
    - Crea una cadena en formato DOT con generateDotString(root).
    - Utiliza vis-network para convertir la cadena DOT en un formato de red (data) y configura las opciones de visualización.
    - Dibuja el árbol en el contenedor tree con la estructura jerárquica configurada.

- **Eventos de Botón**:
    - **Botón "Mostrar Árbol"** (showButton):
        - Llama a drawTree() para visualizar el árbol sin modificar los datos.

4. Flujo de Trabajo

1. **Ingreso de Datos**:
    - El usuario ingresa el conjunto de datos en formato JSON en el campo dataInput.
    - Puede ingresar los datos de predicción en predictInput.
2. **Entrenamiento del Árbol**:
    - Al hacer clic en "Actualizar Árbol", se entrena el árbol de decisión con los datos proporcionados.
    - El árbol utiliza el algoritmo ID3 para construir nodos basados en los atributos y clases del conjunto de datos.
3. **Predicción y Visualización**:
    - La predicción se realiza utilizando los datos en predictInput.
    - drawTree() dibuja el árbol utilizando vis-network para mostrar la estructura de nodos y decisiones.

5. Requisitos y Dependencias

- **Librerías externas**:
    - vis-network: Para la visualización gráfica del árbol de decisión.
    - tytus.js: Proporciona la clase DecisionTreeID3 y el método train para construir el árbol de decisión.
    - Bootstrap: Para el diseño visual de la interfaz y el estilo de los elementos.

K-means

El **algoritmo K-means** es un método de agrupamiento no supervisado que organiza datos en **K grupos o clústeres** en función de sus características, de modo que los datos dentro de cada grupo sean lo más similares posible entre sí, y lo más diferentes posibles de los datos de otros grupos. Es ampliamente utilizado en análisis de datos y aprendizaje automático para identificar patrones en conjuntos de datos grandes y sin etiquetar.

Funcionamiento del Algoritmo K-means

1. **Selección del número de clústeres (K)**: Se define el número de grupos en los que se quiere dividir el conjunto de datos.
2. **Inicialización de centroides**: Se eligen aleatoriamente K puntos en el espacio de datos, que actuarán como los centroides iniciales de cada clúster.
3. **Asignación de puntos a clústeres**: Cada punto de datos se asigna al clúster cuyo centroide esté más cercano, basándose en la distancia (generalmente la distancia euclidiana).
4. **Reubicación de centroides**: Después de asignar todos los puntos, se recalcula el centroide de cada clúster como el promedio de todos los puntos asignados a él.
5. **Iteración**: Los pasos de asignación y reubicación se repiten hasta que los centroides de los clústeres ya no cambien significativamente o hasta alcanzar un número máximo de iteraciones.

Objetivo

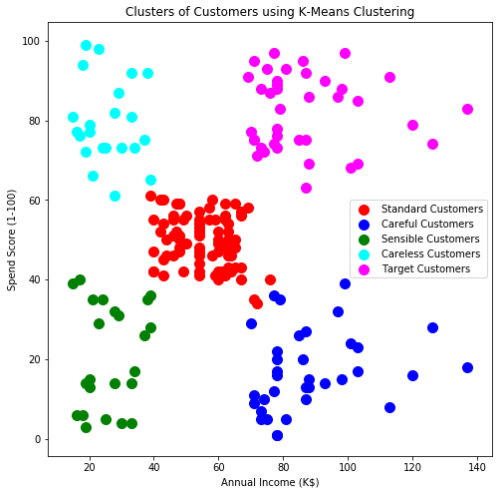
Minimizar la suma de las distancias cuadradas entre los puntos y su centroide asignado en cada clúster, también conocida como **inercia** o **suma de los errores al cuadrado** (SSE). Esto hace que los puntos dentro de cada grupo estén lo más cerca posible del centroide del grupo.

Aplicaciones de K-means

- Segmentación de clientes en marketing.
- Agrupamiento de imágenes en categorías.
- Análisis de patrones de comportamiento en redes sociales.
- Compresión de datos mediante reducción de dimensionalidad.

Consideraciones

La elección de K y la inicialización de los centroides son críticos para el rendimiento del algoritmo, y pueden afectar la estabilidad y precisión del agrupamiento.



Código de K-Means

1. Descripción General

Este código implementa el algoritmo de K-Means para realizar clustering en un conjunto de datos bidimensional. La interfaz permite al usuario ingresar datos en 2D manualmente o cargar un archivo CSV que contiene los parámetros del algoritmo. Utiliza tytus.js para ejecutar el algoritmo de K-Means y Google Charts para visualizar los resultados en un gráfico de dispersión.

```
<h2>KMeans</h2>

<!-- Input para cargar el archivo CSV -->
<input type="file" id="csvFileInput" accept=".csv" onchange="loadCSV()" />

<!-- Inputs para los parámetros de KMeans -->
<label for="cluster_count">Cantidad de Clusters</label>
<input id="cluster_count" type="number" placeholder="Ingrese la cantidad de clusters" value="3">

<label for="linear_data">Datos 2D (x,y)</label>
<input id="linear_data" type="text" placeholder="Ejemplo: [0,0],[0,1],[0,2]" value="[11,6],[4,2],[15,0],[10,6],[7,8],[9,12],[13,0],[5,1],[0,13],[7,5],[6,1],[3,6],[0,10],[14,10],[6,14],[6,4],[4,9],[5,14],[9,9],[13,8]">

<label for="iterations">Iteraciones</label>
<input id="iterations" type="number" placeholder="Ingrese las iteraciones" value="3">

<button id="btnLineal">Calcular KMeans 2D</button>

<div id="chart_div"></div>

<script>
    // Función para cargar y leer el CSV
    function loadCSV() {
        const fileInput = document.getElementById("csvFileInput").files[0];
        const reader = new FileReader();

        reader.onload = function(e) {
            const csvData = e.target.result;
            parseCSVData(csvData);
        };

        reader.readAsText(fileInput);
    }

    // Función para procesar el contenido del CSV
```

2. Estructura del Código

1. **HTML Básico**:
    - Incluye los scripts para tytus.js (que proporciona el algoritmo K-Means) y loader.js de Google Charts (para visualización de datos).
    - Estilos CSS básicos para la estructura y diseño de la página.
    - Entradas (input) para cargar un archivo CSV, especificar la cantidad de clusters (cluster_count), datos en 2D (linear_data), y número de iteraciones (iterations).

- Un botón para iniciar el cálculo de K-Means y un contenedor div (chart_div) donde se visualiza el gráfico.

2. **Interfaz de Usuario**:
   - input[type="file"] (csvFileInput): Permite cargar un archivo CSV con los parámetros para K-Means.
   - input[type="number"] y input[type="text"]: Se usan para definir la cantidad de clusters, los datos en 2D y el número de iteraciones.
   - button (btnLineal): Ejecuta el cálculo de K-Means cuando se hace clic.
   - div (chart_div): Contenedor para mostrar el gráfico de Google Charts con los clusters.

3. Funciones Principales

- **loadCSV()**:
  - Lee el archivo CSV cargado por el usuario usando FileReader.
  - Llama a parseCSVData() para procesar el contenido una vez que la lectura ha finalizado.
- **parseCSVData(csvData)**:
  - Convierte el contenido del archivo CSV en un arreglo de filas (rows).
  - Extrae los valores de cluster_count, linear_data, y iterations desde la primera fila de datos y los asigna a sus respectivos campos en la interfaz.
  - Verifica que el CSV tenga el formato adecuado antes de asignar valores. Si no es así, muestra un mensaje de alerta.
- **Evento de botón btnLineal**:
  - Al hacer clic en el botón, se extraen los valores de linear_data, cluster_count y iterations.
  - Convierte el valor de linear_data de cadena a un arreglo de pares [x, y] utilizando expresiones regulares.
  - Comprueba que la cantidad de datos sea suficiente para el número de clusters deseado.
  - Ejecuta el algoritmo de K-Means:
    - Crea una instancia de _2DKMeans y llama a clusterize() para realizar el clustering.
    - Genera un color aleatorio para cada cluster.

    - Llama a drawChart() para mostrar el gráfico con los datos clusterizados.

- **drawChart(clusters)**:
    - Configura y dibuja el gráfico de dispersión usando Google Charts.
    - Los puntos en el gráfico están coloreados de acuerdo al cluster asignado.
    - Configura las opciones del gráfico, como los títulos de los ejes y el estilo de los puntos.

4. Flujo de Trabajo

1. **Ingreso de Datos**:
    - El usuario puede ingresar manualmente los datos 2D, la cantidad de clusters y el número de iteraciones, o cargar un archivo CSV.
    - Si se carga un archivo CSV, se extraen los valores de cluster_count, linear_data, y iterations y se muestran en la interfaz.
2. **Ejecución del Algoritmo K-Means**:
    - Al hacer clic en el botón "Calcular KMeans 2D", el código valida y transforma los datos de entrada.
    - Ejecuta el algoritmo de K-Means y asigna colores aleatorios a cada cluster.
3. **Visualización**:
    - Llama a drawChart() para visualizar los clusters en un gráfico de dispersión, mostrando la agrupación de los puntos en el espacio 2D.

5. Requisitos y Dependencias

- **Librerías externas**:
    - tytus.js: Biblioteca que contiene el algoritmo _2DKMeans para el cálculo de clustering.
    - https://www.gstatic.com/charts/loader.js: Librería de Google Charts para visualizar los clusters.

Teorema de Bayes

El **Teorema de Bayes** permite calcular la probabilidad de que ocurra un evento A, dado que se tiene información sobre otro evento B que afecta o condiciona la probabilidad de A. Es una herramienta fundamental en estadística y probabilidad, ya que permite actualizar las probabilidades a medida que se obtiene nueva información.

A diferencia del **Teorema de la Probabilidad Total**, que evalúa la probabilidad de B a partir de varios eventos A, el Teorema de Bayes calcula la probabilidad de A dado que B ha ocurrido, es decir, la probabilidad de A condicionada a B.

Fórmula del Teorema de Bayes

La fórmula del Teorema de Bayes se expresa matemáticamente como:

$$P\ (A/B) = \frac{P(B/A) * P(A)}{P(B)}$$

donde:

- P(A|B) es la probabilidad de que ocurra A dado que B ha ocurrido.
- P(B|A) es la probabilidad de que ocurra B dado que A ha ocurrido.
- P(A) es la probabilidad a priori de A (la probabilidad de A antes de saber que B ocurrió).
- P(B) es la probabilidad total de B, calculada como la suma ponderada de P(B|A) ·P(A) para todos los posibles eventos A.

Código de predicción de Naive Bayes

1. Descripción General

Este código implementa una interfaz de usuario en HTML para realizar predicciones climáticas usando el método de Naive Bayes. Se simulan dos métricas de distancia, Manhattan y Euclidiana, para determinar la similitud de los datos ingresados con semanas anteriores, y predecir si el clima del viernes será "LLUVIA", "SOLEADO" o "NUBLADO". Los datos de semanas anteriores se muestran en una tabla, y los resultados de la predicción en otra.

2. Estructura del Código

1. **HTML Básico**:

- Incluye los encabezados para Bootstrap (para los estilos) y estilos personalizados en CSS.
- Estructura la interfaz en una cuadrícula con dos secciones principales:
  - **Tabla de Datos Anteriores**: Muestra las condiciones climáticas históricas de las semanas previas.
  - **Formulario de Datos de Entrada**: Permite al usuario ingresar los porcentajes de lluvia del lunes al jueves para realizar una predicción para el viernes.
- Una tercera sección muestra los resultados de las predicciones utilizando las distancias Manhattan y Euclidiana.

```
<h1 class="display-1 text-center">Predicción Naive Bayes</h1>
<div class="container">
    <div class="row">
        <div class="col-md-6">
            <div class="shadow-lg p-4 bg-white rounded">
                <h3 class="text-center">Tabla de Valores Anteriores</h3>
                <table id="tabla" class="table table-striped">
                </table>
            </div>
        </div>
        <div class="col-md-6">
            <div class="shadow-lg p-4 bg-white rounded">
                <h3 class="text-center">Ingresa Valores Deseados</h3>
                <form action="#">
                    <div class="input-group">
                        <label for="lunes" class="form-label">Porcentaje Lunes:</label>
                        <input id="lunes" type="text" class="form-control" placeholder="Porcentaje Lunes" required>
                    </div>
                    <div class="input-group">
                        <label for="martes" class="form-label">Porcentaje Martes:</label>
                        <input id="martes" type="text" class="form-control" placeholder="Porcentaje Martes" required>
                    </div>
                    <div class="input-group">
                        <label for="miercoles" class="form-label">Porcentaje Miércoles:</label>
                        <input id="miercoles" type="text" class="form-control" placeholder="Porcentaje Miércoles" required>
                    </div>
                    <div class="input-group">
                        <label for="jueves" class="form-label">Porcentaje Jueves:</label>
                        <input id="jueves" type="text" class="form-control" placeholder="Porcentaje Jueves" required>
                    </div>
                    <button type="button" class="btn btn-primary" id="btn">Calcular Predicción</button>
                </form>
            </div>
```

2. **Estilos CSS**:
   - Define el diseño básico de la página, el estilo de los encabezados, contenedores, tablas, y del botón de predicción.
3. **Interfaz de Usuario**:
   - **Tabla de Valores Anteriores** (tabla): Muestra los porcentajes de lluvia de lunes a jueves, y el estado climático del viernes para las semanas anteriores.
   - **Formulario de Entrada de Datos**:

        - Cuatro campos de entrada (lunes, martes, miercoles, jueves) permiten al usuario ingresar los porcentajes de lluvia.
        - Botón Calcular Predicción: Ejecuta la predicción cuando se hace clic.
    - **Resultado de la Predicción**:
        - Tabla (tabla2) muestra los resultados de la predicción usando las métricas Manhattan y Euclidiana.

3. Funciones Principales

- **Carga de Tabla Inicial**:
    - Los datos históricos se definen en la variable datos, que contiene el porcentaje de lluvia de lunes a jueves y el clima del viernes.
    - Se genera una tabla HTML en tabla utilizando un bucle que recorre los datos y los presenta en filas.
- **Evento del Botón Calcular Predicción**:
    - Al hacer clic, se verifica que todos los campos de entrada tengan valores.
    - Crea un nuevo arreglo newIndividuo con los valores ingresados, representando los datos de la semana 8.
    - Llama a la función predecirCluster con las métricas Manhattan y Euclidiana, y presenta los resultados en la tabla tabla2.
- **predecirCluster(metrica, datos, nuevo)**:
    - Calcula las distancias entre los datos ingresados y cada semana anterior en datos.
    - Si metrica es 1, utiliza la distancia Manhattan (calcularManhattan), y si es 2, utiliza la distancia Euclidiana (calcularEuclidiana).
    - Ordena las distancias en orden ascendente y selecciona la clase del elemento más cercano, que representa la predicción.
- **Funciones de Distancia**:
    - **calcularManhattan(p1, p2)**: Calcula la distancia Manhattan entre dos puntos p1 y p2.
    - **calcularEuclidiana(p1, p2)**: Calcula la distancia Euclidiana entre dos puntos p1 y p2.

4. Flujo de Trabajo

1. **Ingreso de Datos**:

- El usuario observa la tabla de datos históricos en la sección "Tabla de Valores Anteriores".
- Ingrese los porcentajes de lluvia de lunes a jueves en los campos correspondientes y haga clic en el botón "Calcular Predicción".

2. **Predicción**:
    - La función predecirCluster calcula la distancia entre los datos ingresados y los datos de las semanas anteriores usando Manhattan y Euclidiana.
    - La predicción para el clima del viernes se muestra en tabla2, comparando ambas métricas.

5. Requisitos y Dependencias

- **Librerías externas**:
    - **Bootstrap**: Para el diseño visual y la estructura de los elementos de la interfaz.