DOCUMENTACIÓN

CAMILO SOTO GERMAN
JUAN LOZANO VERGARA
LUDIS ÁLVAREZ SOLIPAZ
JUAN PALOMO GONZÁLEZ

UNIVERSIDAD DE CÓRDOBA
FACULTAD DE EDUCACIÓN Y CIENCIAS HUMANAS
LICENCIATURA EN INFORMÁTICA
DOCENTE: ALEXANDER TOSCANO

MONTERÍA - CÓRDOBA
ABRIL, 2025

TEMAS

1. **Fundamentos de los árboles de decisión**

- Definición y estructura de un árbol de decisión.
- Ventajas y desventajas en comparación con otros modelos.

2. **Aplicaciones de los árboles de decisión**

- Clasificación y predicción.
- Casos prácticos: salud, educación, negocios, etc.

3. **Configuración del widget "Tree"**

- Parámetros principales: profundidad máxima, número mínimo de instancias, criterios de impureza.
- Cómo ajustar el modelo para evitar sobreajuste.

4. **Conexiones funcionales en Orange**

Uso del widget "Tree" en conjunto con:

- "File" (carga de datos).
- "Test & Score" (evaluación del modelo).
- "Tree Viewer" (visualización del árbol).

5. **Interpretación de los resultados**

- Lectura e interpretación de la estructura del árbol.
- Identificación de nodos, hojas y reglas de decisión.

6. Comparación con otros algoritmos de clasificación

- Comparación de desempeño con modelos como k-NN, Naive Bayes o Logistic Regression.

7. Limitaciones y buenas prácticas

- Cómo detectar y evitar el sobreajuste.
- Selección adecuada de parámetros.

Descripción General

Este proyecto es una aplicación web construida con Vue.js 3 (usando la Composition API con `<script setup>`) y Pinia para la gestión del estado. La aplicación parece estar estructurada en varias secciones/páginas, cada una con su propio store de Pinia para manejar los datos.

Estructura de Stores (Pinia)

2.1 Home Store (`home.ts`)

- **Propósito**: Gestiona los contenidos de la página principal.
- **Estado**:
  - `homeContents`: Array de contenidos dinámicos (títulos, párrafos, bloques).
  - `loading`: Booleano para estados de carga.
  - `error`: Mensaje de error (string o `null`).
- **Acciones**:
  - `fetchHomeContents()`: Obtiene datos de `/api/home`.

2.2 Activities Store (`activities.ts`)

- **Propósito**: Maneja información de las actividades.
- **Endpoint**: `/api/activities`.
- **Estructura similar a Home Store**.

2.3 Contents Store (`contents.ts`)

- **Propósito**: Contenidos del ova.
- **Endpoint**: `/api/contents`.

2.4 Credits Store (`credits.ts`)

- **Propósito**: Datos de los creadores.
- **Endpoint**: `/api/credits`.

2.5 Questions Store (`questions.ts`)

- **Propósito**: Contiene la evalución.
- **Endpoint**: `/api/questions`.

---

3. Componente Principal

Nombre: `Página Principal - Treethner`
Funcionalidad:

- Renderiza contenidos dinámicos según el store `home`.
- Maneja tres estados:
    1. **Cargando**: Muestra "Cargando...".
    2. **Error**: Muestra el mensaje de error.
    3. **Contenido**: Muestra los datos obtenidos.

Estructura del Contenido:

- **Títulos**: `<h1>`, `<h2>`.
- **Párrafos**: `<p>`, `<p2>`.
- **Bloques**: Array de objetos con:
  `{ h3: "Subtítulo", p: "Texto descriptivo" }`

---

4. Endpoints de la API

Store	Endpoint
Home	`/api/home`
Activities	`/api/activities`

Store	Endpoint
Contents	`/api/contents`
Credits	`/api/credits`
Questions	`/api/questions`

---

5. Flujo de Datos

1. **Inicialización**:
    - Los componentes llaman a acciones del store (ej: `fetchHomeContents()`).
2. **Fetch**:
    - Se realiza una petición GET al endpoint correspondiente.
3. **Respuesta**:
    - Éxito: Los datos se almacenan en el estado (ej: `homeContents`).
    - Error: Se captura y guarda en `error`