TouristHelper: Predicción de Riesgo Delictivo con IA Híbrida (C++ & React)

Este proyecto implementa un sistema de predicción de seguridad ciudadana basado en datos históricos. Utiliza una arquitectura híbrida de Inteligencia Artificial que combina **Random Forest (ML)** para análisis de patrones estáticos y **LSTM (DL)** para predicción de secuencias temporales.

El núcleo de procesamiento y entrenamiento está escrito en **C++ moderno (C++17/20)** para máximo rendimiento, mientras que la interfaz de usuario es una aplicación web en **React**.

Tabla de Contenidos

🛠 Requisitos y Dependencias

Backend y Entrenamiento (C++)

- **Compilador C++:** Compatible con C++17 o superior (GCC 9+, Clang 10+, MSVC).
- **CMake:** Versión 3.16+.
- **Librerías del Sistema:**
- `mlpack` (Machine Learning en C++).
- `armadillo` (Álgebra lineal, dependencia de mlpack).
- `OpenMP` (Paralelización).
- `LibTorch` (PyTorch C++ frontend - Versión **cxx11 ABI**).

- `pkg-config`.

- **Librerías "Header-only" (Incluidas en el código):**

- `cpp-httplib` (Servidor HTTP).
- `nlohmann/json` (Manejo de JSON).

Frontend (JavaScript)

- **Node.js:** v14 o superior.
- **npm** o **yarn**.

Instalación y Compilación

1. Configuración de LibTorch

Descarga la versión **LibTorch cxx11 ABI** (CPU o CUDA según tu hardware) desde pytorch.org. Descomprímela dentro de la carpeta raíz del proyecto (o ten la ruta a mano).

```
# Estructura esperada
/TouristHelper
  /backend
  /server
  /frontend
  /Dataset
```

2. Compilación del Backend (C++)

El proyecto utiliza CMake para gestionar la construcción.

```
# 1. Crear carpeta de build
mkdir build && cd build

# 2. Configurar CMake
```

```
# Si libtorch está en la raíz del proyecto:
cmake ..
# Si libtorch está en otra ruta:
cmake -DCMAKE_PREFIX_PATH=/ruta/a/libtorch ..

# 3. Compilar (Usa -j para usar todos los núcleos)
cmake --build . --parallel 4
```

Esto generará los ejecutables principales (repetir en la carpeta backend y server):

- `TouristHelper` (Entrenamiento).
- `TouristBackend` (Servidor API).

3. Instalación del Frontend

```
cd frontend
npm install
```

Entrenamiento del Modelo

Antes de iniciar el servidor, es necesario procesar los datos y generar los modelos binarios (Dentro de la carpeta server estan los modelos ya generados).

Comando:

```
cd build
./TouristHelper
```

Flujo del proceso:

1. **Carga de Datos:** Lee el archivo CSV masivo (`datos_202112_202510_v2.csv`).
2. **Preprocesamiento:** Codifica variables categóricas (Provincia, Cantón) a numéricas.
3. **Entrenamiento Random Forest:** Genera `rf_model.bin`.
4. **Entrenamiento LSTM:** Genera `lstm_model.bin` (o usa TorchScript para cargar `.pt`).

Nota: Asegúrate de que el archivo `.csv` esté en la ruta relativa correcta (usualmente `../` desde la carpeta build).

Ejecución del Servidor (Backend)

Una vez entrenados los modelos, levanta el servidor API que atenderá las peticiones del frontend.

Comando:

```
cd build
./TouristBackend
```

Salida esperada:

```
--- Iniciando Backend TouristHelper ---
Modelo cargado exitosamente.
Servidor escuchando en http://localhost:8000
```

El servidor expone el endpoint: `GET /riesgo?provincia=XXX&canton=YYY`.

Ejecución del Frontend (React)

En una nueva terminal:

Comando:

```
cd frontend
npm run start
```

La aplicación se abrirá automáticamente en `http://localhost:3000`.

Explicación Técnica del Entrenamiento

El archivo `main.cpp` (y sus helpers) implementa una estrategia avanzada para manejar "Big Data" en C++ en hardware de consumo.

1. Estrategia de Carga "Two-Pass Loading"

Dado que el dataset puede pesar gigabytes (2.7GB+), cargarlo todo en un `std::vector` y luego convertirlo a matriz colapsaría la RAM.

- **Pase 1 (Scan):** Lee el archivo línea por línea *sin guardar los datos*. Solo construye los diccionarios (Encoders) para saber cuántas provincias y cantones existen y cuenta las filas totales.
- **Pase 2 (Stream):** Vuelve a leer el archivo, pero esta vez inserta los valores numéricos directamente en una matriz pre-asignada de **Armadillo (`arma::fmat`)**. Esto evita la fragmentación de memoria.

2. Codificación Jerárquica (`HierarchicalCantonEncoder`)

Los cantones no tienen IDs únicos globales en los datos crudos (ej. puede haber un cantón con el mismo nombre en otra provincia, o los IDs se repiten).

- El código implementa un encoder personalizado que combina `ID_Provincia` + `Nombre_Canton` para generar un **ID Matemático Único**. Esto es crucial para que el modelo no confunda zonas geográficas distintas.

3. Modelo Híbrido

- **Random Forest (MLPack):**
- Se utiliza para clasificar el riesgo basándose en características estáticas y demográficas.
- Configurado con `MinLeafSize=50` para evitar árboles demasiado profundos que consuman mucha RAM.

- Produce una probabilidad base para cada tipo de incidente.

- **LSTM (Red Neuronal Recurrente):**

- Se encarga de la **Memoria Temporal**.
- Analiza la secuencia histórica de eventos en un cantón específico.
- Entrada: `[Evento_Previo, ID_Canton]`.
- Objetivo: Predecir cuál es el *siguiente* tipo de evento más probable, permitiendo al sistema alertar sobre tendencias (ej. "Después de un robo en esta zona, suele haber violencia").

4. Optimización de Memoria

- Uso estricto de `float` (4 bytes) en lugar de `double` (8 bytes) para las matrices de datos, reduciendo el consumo de RAM en un 50%.
- Uso de `std::string_view` donde es posible para evitar copias innecesarias de texto durante el parseo del CSV.
- Debido a que se uso random forest, utiliza mucha memoria al momento de generar los arboles de decision, por lo cual se tuvo que rentar una maquina virtual con mas o menos 128 gigas de ram (para el dataset completo), para un entrenamiento de prueba probar con un dataset mucho mas liviano.

Librerías Utilizadas

C++ (Backend & Core)

Librería	Propósito
MLPack	Algoritmos de Machine Learning (Random Forest).
Armadillo	Álgebra lineal de alto rendimiento (Matrices/Vectores).
LibTorch	Deep Learning (Carga y ejecución de modelos LSTM de PyTorch).
OpenMP	Procesamiento paralelo multihilo.
cpp-httplib	Creación del servidor REST API ligero.
nlohmann/json	Serialización y deserialización de JSON moderno.

JavaScript (Frontend)

Librería	Propósito
React	Framework de UI.

Librería	Propósito
Leaflet	Mapas interactivos.
react-leaflet	Componentes de React para Leaflet.
leaflet.heat	Plugin para generación de mapas de calor.