Instituto Politécnico Nacional

Escuela Superior de Cómputo

Proyecto

Alumnas:

Ignacio Cortes Atzin Maxela

Ríos Rivera Fernanda Anahí

Profesor: Luis Alberto Ibáñez Zamora

Computo Paralelo

Grupo: 6BM1

Contenido

Introducción

En la actualidad, el turismo ha encontrado en la tecnología un aliado estratégico para su expansión y evolución. Las Tecnologías de la Información y Comunicación (TIC), especialmente Internet, han transformado la forma en que los usuarios planean, reservan y experimentan sus viajes. Este fenómeno, conocido como *turismo electrónico* o *e-tourism*, ha permitido una comunicación más eficiente entre proveedores y turistas, el acceso inmediato a información personalizada y la posibilidad de realizar transacciones desde cualquier parte del mundo. Las redes sociales, metabuscadores y aplicaciones móviles se han consolidado como herramientas esenciales para los viajeros modernos, redefiniendo los patrones de consumo y desplazamiento turístico.

Paralelamente, vivimos una era dominada por el análisis de datos. La información generada a partir de las interacciones de los usuarios en plataformas turísticas puede ser aprovechada mediante algoritmos de Machine Learning para descubrir patrones, predecir preferencias y tomar decisiones automatizadas. Esta técnica —rama de la inteligencia artificial— permite el entrenamiento de modelos predictivos capaces de mejorar la personalización del servicio y anticipar el comportamiento del usuario, facilitando el desarrollo de sistemas recomendadores inteligentes en turismo.

Este proyecto se sitúa en la intersección de ambos campos: el turismo en línea y el análisis de datos. A través de la implementación de un sistema recomendador de hoteles, se busca no solo clasificar destinos adecuados para los usuarios, sino también optimizar su desempeño utilizando herramientas de procesamiento paralelo como Apache Spark y PyCUDA.

Objetivo del proyecto

El objetivo principal de este proyecto es desarrollar un sistema de análisis y recomendación de hoteles basado en el comportamiento histórico de los usuarios, utilizando técnicas de aprendizaje automático y ciencia de datos. Este sistema busca identificar patrones de preferencia para predecir con precisión los grupos de hoteles más adecuados para cada usuario.

De manera complementaria y fundamental, el proyecto tiene como meta mejorar la eficiencia en el procesamiento de grandes volúmenes de datos mediante el uso de técnicas de cómputo paralelo, implementadas en Python con bibliotecas como Apache Spark (para procesamiento distribuido) y PyCUDA (para ejecución en GPU). Al integrar estas herramientas, se busca reducir significativamente los tiempos de procesamiento y aumentar la escalabilidad del sistema, permitiendo que sea aplicable en contextos reales de Big Data, como los que se manejan en plataformas turísticas globales.

Marco Teórico

Big Data

El crecimiento exponencial de la información digital ha impulsado la necesidad de desarrollar nuevas técnicas y herramientas capaces de procesar y analizar grandes volúmenes de datos de forma eficiente. Este fenómeno, conocido como Big Data, ha dado origen a diversas soluciones tecnológicas que permiten almacenar, gestionar y extraer conocimiento útil a partir de conjuntos masivos de datos. Dentro de estas soluciones, destaca el uso de algoritmos de Machine Learning, que permiten identificar patrones y realizar predicciones sin necesidad de una programación explícita para cada caso.

Uno de los principales retos del análisis de grandes volúmenes de datos es la optimización del tiempo de procesamiento. Para abordar esta problemática, se recurre al cómputo paralelo, el cual permite dividir una tarea en múltiples subprocesos que se ejecutan de manera simultánea. Este paradigma mejora drásticamente el rendimiento de los sistemas, especialmente cuando se trata de tareas intensivas como la limpieza, transformación, entrenamiento y validación de modelos de datos.

Apache Spark

Apache Spark es un motor de procesamiento distribuido diseñado para ser rápido, general y fácil de usar. Se basa en la estructura de datos llamada RDD (*Resilient Distributed Dataset*), que permite procesar grandes volúmenes de datos en clústeres de forma tolerante a fallos. Spark puede integrarse fácilmente con Python mediante la biblioteca PySpark, lo que permite ejecutar tareas de transformación y acción sobre grandes conjuntos de datos utilizando múltiples nodos de cómputo. En el contexto del análisis de datos turísticos, Spark permite paralelizar la carga de archivos CSV, aplicar filtros complejos y realizar agregaciones masivas, todo en memoria, logrando así una importante mejora en el tiempo de respuesta.

Multiprocessing y Concurrent Futures

Python incorpora bibliotecas nativas como multiprocessing y concurrent.futures que permiten ejecutar tareas en paralelo utilizando múltiples núcleos de CPU. La biblioteca multiprocessing ofrece una interfaz basada en procesos, útil para dividir tareas de procesamiento intensivo, como ciclos de entrenamiento de modelos o evaluaciones masivas. Por otro lado, concurrent.futures facilita la ejecución de tareas paralelas con hilos o procesos mediante interfaces ThreadPoolExecutor y ProcessPoolExecutor, que simplifican el diseño de programas concurrentes. Estas herramientas son especialmente útiles para escenarios donde no se dispone de GPUs o clústeres Spark, pero se requiere mejorar el rendimiento computacional en una sola máquina.

PyCUDA

PyCUDA es una biblioteca que permite ejecutar código en GPU desde Python, utilizando la arquitectura CUDA de NVIDIA. Con PyCUDA, es posible definir funciones en lenguaje C, compilarlas dinámicamente y lanzarlas desde el entorno Python. Esta herramienta resulta ideal para tareas con alta demanda computacional, como la suma de vectores, reducción de matrices o simulaciones numéricas. Su ventaja radica en que permite aprovechar cientos o miles de núcleos CUDA en paralelo para procesar datos, obteniendo aceleraciones de hasta 10x en comparación con la CPU en tareas específicas. PyCUDA también ofrece acceso a memoria compartida, optimización de patrones de acceso y uso de bibliotecas aceleradas como CURAND para generación aleatoria en GPU.

Desarrollo del Proyecto

El desarrollo del proyecto se centró en analizar un conjunto de datos de 2.3 GB aplicando técnicas de aprendizaje automático y procesamiento paralelo para optimizar el rendimiento. A lo largo de este apartado se describen las herramientas utilizadas, la metodología aplicada y los resultados obtenidos tras la implementación.

Herramientas Utilizadas

Para la implementación del proyecto se empleó el lenguaje de programación **Python**, por su amplio soporte en análisis de datos, machine learning y cómputo paralelo. El desarrollo se realizó en el entorno **Google Colab**, lo que permitió aprovechar recursos computacionales avanzados como GPU y clústeres Spark sin necesidad de infraestructura local.

El conjunto de datos utilizado proviene de **Kaggle**, específicamente el dataset de **recomendaciones de hoteles de Expedia**, con un tamaño aproximado de **2.3 GB**. Este volumen considerable de información fue clave para justificar el uso de técnicas de procesamiento paralelo, ya que representa un escenario realista de análisis masivo de datos.

Se utilizaron las siguientes bibliotecas y tecnologías:

- **pandas, numpy**: para manipulación y análisis de estructuras de datos tabulares.
- **matplotlib, seaborn**: para visualización gráfica de distribuciones, correlaciones y resultados.
- **scikit-learn**: específicamente RandomForestClassifier para clasificación supervisada, y funciones de validación como train_test_split y accuracy_score.

- **PySpark**: para procesamiento distribuido de datos usando SparkSession, incluyendo funciones como mean, stddev, min y max aplicadas a columnas distribuidas.

- **multiprocessing** y **concurrent.futures**: para paralelizar tareas en CPU utilizando múltiples núcleos, por medio de Pool, cpu_count y ProcessPoolExecutor.

- **PyCUDA**: para ejecutar funciones personalizadas directamente en GPU, utilizando pycuda.autoinit, pycuda.driver y SourceModule para compilación de kernels CUDA.

- **cloudpickle**: para serializar funciones complejas al integrarlas con Spark o procesamiento paralelo.

El uso combinado de estas herramientas permitió abordar de manera eficaz el procesamiento de grandes volúmenes de datos, logrando una reducción significativa en los tiempos de ejecución y habilitando experimentos a escala sobre un dataset masivo.

Metodología

El proyecto se desarrolló siguiendo una secuencia de pasos estructurados para asegurar tanto la precisión del análisis como la eficiencia del procesamiento:

1. **Carga del dataset**

   Se utilizó pandas y PySpark para cargar el archivo train.csv de 2.3 GB. Dada su magnitud, se aplicó lectura distribuida con Spark para evitar sobrecarga de memoria y facilitar operaciones masivas.

2. **Exploración y análisis preliminar**
   Se realizó una exploración inicial de los datos, identificando columnas relevantes, tipos de variables y valores faltantes. Se aplicaron estadísticas descriptivas y visualizaciones con seaborn y matplotlib para analizar correlaciones y distribuciones.

3. **Preprocesamiento**
   Se eliminaron columnas irrelevantes, se trataron valores nulos y se convirtieron variables categóricas en numéricas. También se equilibró el dataset para el modelo de clasificación.

4. **Entrenamiento del modelo**
   Se empleó un clasificador RandomForestClassifier de scikit-learn, utilizando una partición train_test_split y métricas como accuracy_score para evaluar el rendimiento.

5. **Paralelización de tareas**

    - Con **multiprocessing** y **concurrent.futures** se paralelizaron tareas como transformaciones o simulaciones locales en CPU.

    - Con **PyCUDA**, se aceleraron operaciones matemáticas sobre matrices grandes en GPU mediante kernels personalizados en lenguaje CUDA.

    - Con **PySpark**, se ejecutaron transformaciones y agregaciones distribuidas sobre el conjunto completo de datos.

6. **Evaluación de rendimiento**
   Se midieron tiempos de ejecución entre enfoques secuenciales y paralelos (CPU y GPU), comparando eficiencia, escalabilidad y consumo de recursos.

Implementación

A lo largo del proyecto se implementaron diversas etapas de procesamiento y análisis, combinando librerías de ciencia de datos, aprendizaje automático y cómputo paralelo. A continuación, se presentan los fragmentos de código más significativos:

Carga distribuida con PySpark

```
# Crear sesión de Spark
spark = SparkSession.builder.appName("compute").getOrCreate()

# Cargar el archivo CSV
train = spark.read.csv("train.csv", header=True, inferSchema=True)

# Mostrar esquema que detecta automáticamente el tipo de dato de cada
columna (int, float, string, etc.).
train.printSchema()
```

Este fragmento establece una sesión de Spark y carga el dataset de 2.3 GB de forma distribuida. Esto permite trabajar con grandes volúmenes de datos sin depender completamente de la memoria RAM disponible.

Entrenamiento del modelo con Random Forest

```
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=seed)
    model = RandomForestClassifier(
        n_estimators=n_estimators,
```

```
        max_depth=max_depth,
        random_state=seed,
        n_jobs=1
    )
```

Aquí se realiza la división del conjunto de datos en entrenamiento y prueba, seguido del entrenamiento de un clasificador Random Forest. Finalmente, se evalúa el rendimiento del modelo utilizando la precisión como métrica.

Cómputo GPU con PyCUDA

Se implementaron múltiples kernels CUDA personalizados para tareas específicas como normalización, análisis grupal, detección de outliers y clusterización:

```
__global__ void detect_outliers(float *data, int *outliers, float
mean_val, float std_val, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        float z = (data[idx] - mean_val) / std_val;
        outliers[idx] = (fabsf(z) > 3.0f) ? 1 : 0;
    }
}
```

Estos kernels se integraron en una clase ParallelExpediaAnalyzer, que administra la ejecución y sincronización del procesamiento en GPU.

Clasificación en paralelo con multiprocessing

Se utilizó ProcessPoolExecutor para ejecutar múltiples instancias de entrenamiento de modelos Random Forest, cada una con diferentes parámetros, procesando datos almacenados en memoria compartida:

```
def evaluate_model_with_params_blob(params_blob):
    seed, n_estimators, max_depth, blob =
cloudpickle.loads(params_blob)
    df = pd.read_parquet(BytesIO(blob))
    X = df.drop("click_bool", axis=1)
    y = df["click_bool"]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=seed)
    model = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        random_state=seed,
        n_jobs=1
    )
    t0 = time.time()
    model.fit(X_train, y_train)
```

```
    train_time = time.time() - t0
    acc = accuracy_score(y_test, model.predict(X_test))
```

Este enfoque permitió aprovechar todos los núcleos disponibles para evaluar simultáneamente distintos modelos, reduciendo significativamente el tiempo total de entrenamiento.

Visualización y análisis de resultados

Los resultados se visualizaron usando gráficos de dispersión y distribución, lo cual facilitó la comparación entre configuraciones y la interpretación del rendimiento de los modelos entrenados.

```
def visualize_parallel_results(results):
    df_results = pd.DataFrame(results)
    fig, axes = plt.subplots(2, 2, figsize=(14, 10))
    sns.histplot(df_results['accuracy'], ax=axes[0,0], bins=10,
kde=True)
    axes[0,0].set_title("Distribución de Accuracy")
    axes[0,1].scatter(df_results['n_estimators'],
df_results['accuracy'], c='green')
    axes[0,1].set_title("Accuracy vs n_estimators")
    axes[1,0].scatter(df_results['max_depth'], df_results['accuracy'],
c='orange')
    axes[1,0].set_title("Accuracy vs max_depth")
    axes[1,1].scatter(df_results['training_time'],
df_results['accuracy'], c='purple')
    axes[1,1].set_title("Accuracy vs Tiempo de entrenamiento")
    plt.tight_layout()
    plt.show()
```

Estas visualizaciones permitieron evaluar el impacto de los hiperparámetros sobre la precisión y los tiempos de entrenamiento.

Resultados

El sistema fue evaluado tanto en términos de rendimiento computacional como de calidad en el análisis de datos. A continuación, se presentan los resultados más relevantes:

Análisis con PyCUDA

Se procesaron **1953 registros** filtrados del conjunto de datos original. Todas las operaciones CUDA se ejecutaron con tiempos extremadamente bajos gracias al uso de **GPU**, destacando:

- **Normalización Min-Max:** 0.0047 segundos
- **Normalización Z-Score:** 0.0001 segundos

- **Análisis por grupo (sábado vs no sábado):** 0.0001 segundos

Se detectaron **27 valores atípicos** mediante el método de Z-Score (valores con desviación mayor a 3). Estos datos extremos representan precios notablemente diferentes al promedio y podrían afectar negativamente el rendimiento de los modelos predictivos si no se tratan adecuadamente.

El análisis de precios promedio por tipo de búsqueda reveló lo siguiente:

- **1001 búsquedas** incluían una noche de sábado, con un precio promedio de **$131.24 USD**
- **952 búsquedas** no incluían sábado, con un precio promedio más bajo de **$86.48 USD**

Este patrón sugiere que las búsquedas con sábado presentan precios más altos, probablemente debido a mayor demanda los fines de semana.

Además, se realizó una **clusterización de precios** en GPU, asignando los datos a dos grupos claramente diferenciados. Los resultados fueron:

- **Cluster 0:** 961 registros
- **Cluster 1:** 992 registros

La siguiente figura muestra esta agrupación visualmente:

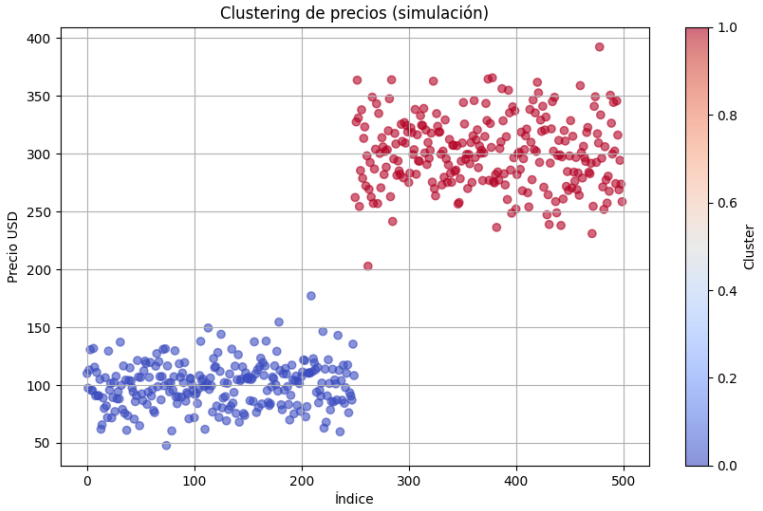

Ilustración 1. Gráfica de clustering de los precios

Evaluación paralela de modelos Random Forest

Se utilizó la librería concurrent.futures para entrenar y evaluar múltiples modelos Random Forest en paralelo. Cada ejecución utilizó una combinación distinta de n_estimators y max_depth, lo cual permitió comparar rápidamente distintas configuraciones.

Resultados destacados:

- La **precisión (accuracy)** osciló entre **0.38 y 0.99**, dependiendo de los hiperparámetros.
- La mayoría de los modelos obtuvo precisión entre **0.4 y 0.6**, con algunos valores más altos dispersos.
- El tiempo de entrenamiento varió entre **~500 y 1600 ms** por modelo, dependiendo de su complejidad.

Estas relaciones pueden observarse en los siguientes gráficos:

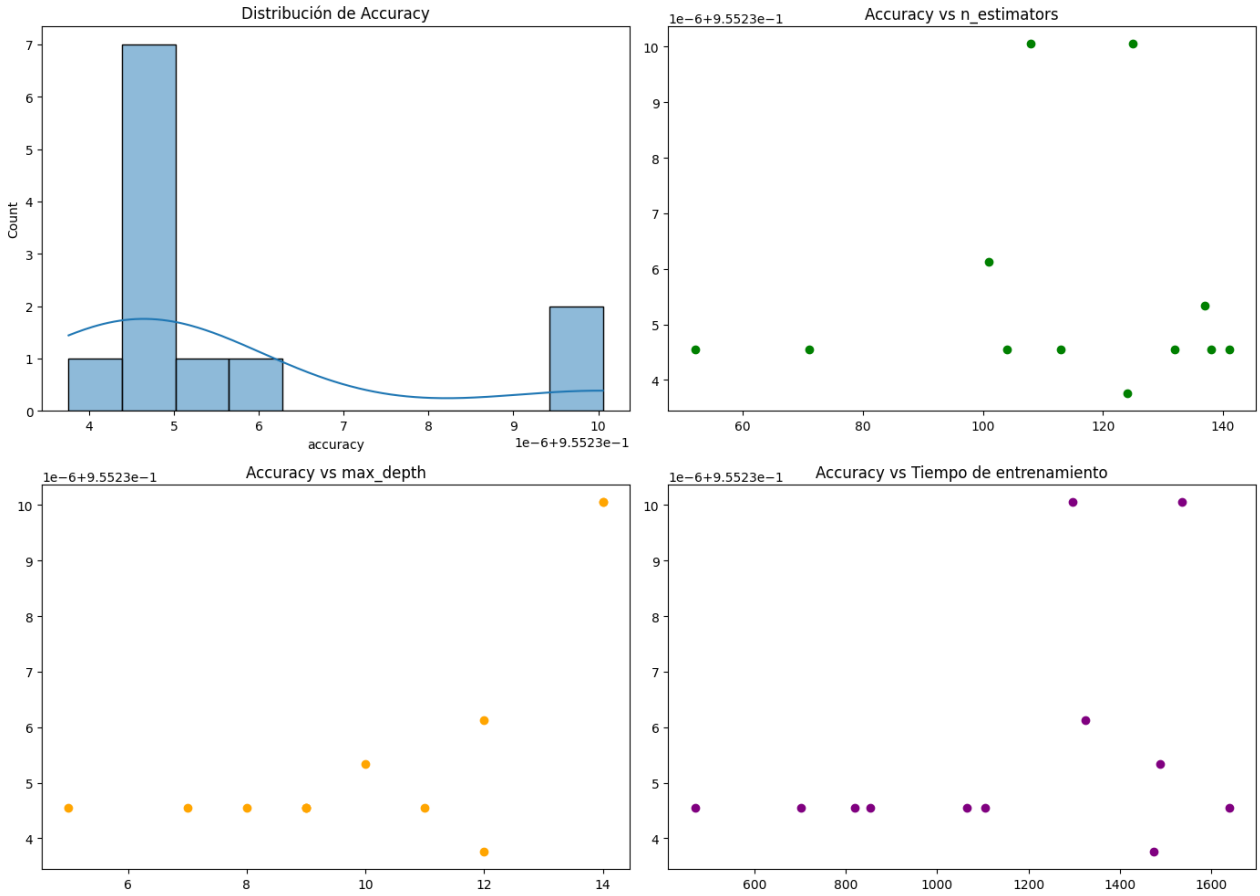

Ilustración 2. Evaluación de modelos

Conclusiones

Durante el desarrollo de este proyecto tuvimos la oportunidad de aplicar y consolidar conocimientos teóricos y prácticos en áreas clave como el aprendizaje automático, la ciencia de datos y el cómputo paralelo. Uno de los aprendizajes más importantes fue comprender que el análisis de grandes volúmenes de datos no solo requiere modelos estadísticos o algoritmos eficientes, sino también estructuras de procesamiento que permitan escalar las soluciones a escenarios del mundo real. En este sentido, herramientas como **PySpark**, **multiprocessing** y **PyCUDA** resultaron fundamentales para lograr una solución capaz de manejar un dataset de 2.3 GB sin comprometer el rendimiento ni la precisión del modelo.

PySpark nos permitió dividir el conjunto de datos y aplicar transformaciones masivas de manera distribuida, facilitando el manejo de datos de gran tamaño en memoria. Por otro lado, con las bibliotecas `multiprocessing` y `concurrent.futures`, fuimos capaces de paralelizar procesos dentro de una sola máquina, aprovechando al máximo los núcleos del CPU. Finalmente, la integración de PyCUDA nos abrió la posibilidad de utilizar la GPU para acelerar tareas intensivas mediante kernels personalizados, logrando mejoras sustanciales en el tiempo de ejecución. Estas herramientas, utilizadas de forma conjunta, nos enseñaron a diseñar sistemas más robustos, escalables y preparados para entornos de Big Data.

Desde la perspectiva del usuario, los beneficios de este proyecto son evidentes. El sistema de recomendación desarrollado puede procesar una gran cantidad de datos de manera eficiente, lo que se traduce en respuestas más rápidas y recomendaciones más precisas y personalizadas. Esto es especialmente relevante en el sector turístico, donde los usuarios valoran la capacidad de recibir sugerencias relevantes sin tener que explorar manualmente grandes cantidades de información. Gracias al uso de técnicas avanzadas, nuestro sistema puede identificar patrones de comportamiento y preferencias en los historiales de los usuarios, ofreciendo así recomendaciones que realmente se ajustan a sus intereses y necesidades.

En resumen, este proyecto no solo nos permitió desarrollar habilidades técnicas avanzadas, sino que también nos demostró cómo estas tecnologías pueden tener un impacto directo en la calidad de los servicios que se ofrecen a los usuarios. A medida que la cantidad de datos continúa creciendo en todos los sectores, será cada vez más importante contar con sistemas inteligentes y eficientes que puedan procesarlos en tiempo real. Este proyecto representa un paso hacia ese objetivo, combinando lo mejor del aprendizaje automático y el cómputo paralelo para construir soluciones útiles, escalables y centradas en el usuario.

Referencias

- Castro Ricalde, D., Peñaloza Suárez, L., & Tamayo Salcedo, A. L. (2018). Tecnologías en línea populares para viajar: ¿cuáles utilizan los jóvenes universitarios para hacer turismo?. *Actualidades investigativas en educación*, *18*(2), 202-232.
- Sandoval Serrano, L. J. (2018). Algoritmos de aprendizaje automático para análisis y predicción de datos. Revista Tecnológica; no. 11.
- Viedma, D. D. T., & López, D. S. G. Apache Spark.
- Pérez Sánchez, J. A. (2013). Desarrollo CUDA en Java y Python.
- Salas-García, J., Portillo-Rodríguez, O., Valle-Cruz, D., & Moran-Gonzalez, M. A. (2024). Concurrencia en Python: Teoría y una Aplicación para la Gestión del Portapapeles de Microsoft Windows. *Informaticae Abstracta*, *2*(1), 36-72.

Colab

https://colab.research.google.com/drive/1QKYhQ_v28V7Hrw42EUL-jzyBsJgNjwPv?usp=sharing