Book Recommendation System Using Hadoop, Apache Spark, and ML

~ An Efficient System for Personalized Book Recommendations

Problem Statement & Objective

Challenges in Book Recommendation:

- Traditional recommendation systems struggle with large volumes of unstructured data.
- Difficulty in processing and analyzing data efficiently.
- Limited scalability and performance issues with conventional tools.

Objective:

To build a personalized book recommendation system using Hadoop, Apache Spark , and machine learning models.

Focus Areas:

- Distributed data processing
- Machine learning for personalized recommendations
- Data visualization using Plotly
- Web UI for user interaction

Dataset

Dataset Overview

Meets Big Data Criteria: **VOLUME**

•**Volume**: 5.2 GB dataset, over 3 million reviews, and 212,404 unique books.

•**Rich Metadata**: Includes structured (ratings, genres) and semi-structured (textual reviews) data.

•**Link:**https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews

Proposed Solution

Approach:

- Scalable and cost-effective system using distributed processing with Apache Spark.
- Utilize Machine Learning models for personalized recommendations
- Seamless, engaging user experience

Why Hadoop and Apache Spark?

- Cloud limitations are avoided by using Hadoop for scalable storage (HDFS) and distributed processing.
- Apache Spark provides distributed data processing, faster computations, and efficient handling of large datasets.
- Cost-effective and economical, reducing the need for cloud-based solutions.
- A user-friendly web UI and real-time recommendations.

Tools Used & System Architecture

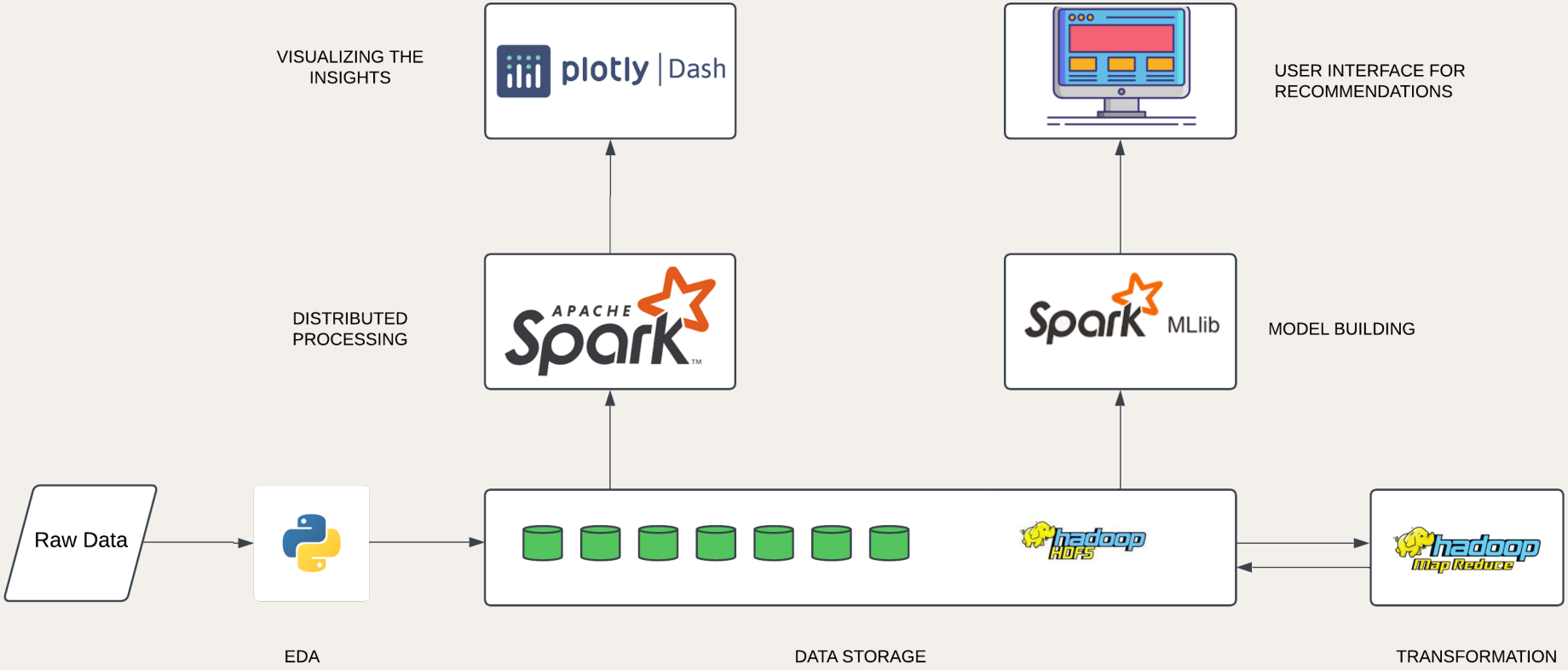
Tools:

- **Hadoop (HDFS):** For storing large datasets.
- **Apache Spark:** For distributed processing using PySpark.
- **ML Models (MLlib):** For generating personalized recommendations.
- **Plotly & Dash:** For data visualization and dashboard creation.
- **Flask:** For Web Development

System Architecture Diagram:

- Data is stored in HDFS and processed by Apache Spark.
- Machine learning models generate recommendations.
- Results are visualized using Plotly and Dash.
- Users interact with the system via the web UI.

System Architecture



Preprocessing & EDA

Preprocessing Steps:

•Cleaning and transforming raw data to handle missing values.
•Data imputation and removal of null values.

EDA (Exploratory Data Analysis):

•Analyzing the dataset for patterns, trends, and relationships.
•Identifying important features for recommendations (authors, categories, reviewscore).

Storage (HDFS)

Data Storage:

HDFS:

- A distributed file system used to store large datasets
- Raw data is uploaded to HDFS for efficient access and processing across multiple nodes.
- Using PySpark, the data is read from HDFS for further transformations and analysis.

Transformations & MR Jobs

MapReduce Jobs:

•**Purpose:** Process large datasets in parallel across nodes.

•**Data Extraction:** Use MapReduce jobs to extract relevant features (e.g., book titles, authors).

•**Transformation:** Apply transformations like data cleaning and feature selection.

• **MapReduce Flow:**

• Map: Process and transform the data.
• Reduce: Aggregate and summarize the results.

MR Job Example

1.Identify the top 20 categories with the most books in a dataset using MapReduce.

•**Mapper:**

- Processes the category of each book.
- Emits key-value pairs:
  - **Key:** Book category (e.g., "Science Fiction", "Romance")
  - **Value:** 1 (indicating the presence of the book in that category).

•**Reducer:**

- Groups the categories and sums the values to count books in each category.
- Returns a sorted list of categories by total count, showing the most popular ones.

Insights:

•**Category Popularity:** Identifies the top 20 categories with the most books.

•**Trends in Reading Interests:** Highlights popular trends or shifts in genres like "Fantasy" or "Self-Help".

MR Job Execution

Text Editor — Dec 4 21:04

Open — mapper5.py — ~/Group04/mr5 — Save

```python
#!/usr/bin/env python3
import sys
import csv
import ast  # To safely parse list strings

# Read input line by line
for i, line in enumerate(sys.stdin):
    try:
        # Use csv.reader to handle structured CSV rows
        reader = csv.reader([line])
        for row in reader:
            # On the first line (header), find the index for 'categories'
            if i == 0:
                header = row
                categories_index = header.index("categories")
                continue

            # Process subsequent rows
            categories = row[categories_index]
            genres = ast.literal_eval(categories)  # Convert string to list
            if isinstance(genres, list):
                for genre in genres:
                    print(f"{genre}\t1")
    except Exception as e:
        continue
```

Open — *reducer5.py — ~/Group04/mr5 — Save

```python

#!/usr/bin/env python3
import sys
from collections import defaultdict

genre_count = defaultdict(int)

# Aggregate counts for each genre
for line in sys.stdin:
    genre, count = line.strip().split("\t")
    genre_count[genre] += int(count)

# Sort genres by count in descending order and take the top 20
top_genres = sorted(genre_count.items(), key=lambda x: x[1], reverse=True)[:20]

# Output the results
for genre, count in top_genres:
    print(f"{genre}\t{count}")
```

Open

```
Fiction 21474
Book burning    4932
Juvenile Fiction        4268
History 3110
Education       2138
Biography & Autobiography       1951
Business & Economics    1625
Religion        1450
American fiction        1353
Predation (Biology)     950
Sports & Recreation     870
Body, Mind & Spirit     803
Cooking 697
Poetry  677
Authors, American       619
Health & Fitness        571
Science 513
Psychology      506
Children's literature   487
Computers       485
```

ies — Terminal — Dec 4 20:54

nimmi@nimmi-VirtualBox: ~/Group04/mr5

```
nimmi@nimmi-VirtualBox:~$ cd Group04/mr5
nimmi@nimmi-VirtualBox:~/Group04/mr5$ ls
mapper5.py  reducer5.py
nimmi@nimmi-VirtualBox:~/Group04/mr5$ chmod 777 mapper5.py
nimmi@nimmi-VirtualBox:~/Group04/mr5$ chmod 777 reducer5.py
nimmi@nimmi-VirtualBox:~/Group04/mr5$ mapred streaming -input /user/Group04/Books.csv -output /user/Group04/mr5 -mapper 'python3 mapper5.py' -reducer 'python3 reducer5.py' -file /home/nimmi/Group04/mr5/mapper5.py -file /home/nimmi/Group04/mr5/reducer5.py
2024-12-04 20:54:33,445 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [/home/nimmi/Group04/mr5/mapper5.py, /home/nimmi/Group04/mr5/reducer5.py] [/home/nimmi/hadoop-3.2.3/share/hadoop/tools/lib/hadoop-streaming-3.2.3.jar] /tmp/streamjob6532013909518345767.jar tmpDir=null
```

Hadoop — Overview — Datanodes — Datanode Volume Failures — Snapshot — Startup Progress — Utilities

Browse Directory

/user/Group04 — Go!

Show 25 entries — Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	nimmi	supergroup	1.87 GB	Nov 23 15:02	3	128 MB	Books.csv
drwxr-xr-x	nimmi	supergroup	0 B	Nov 21 12:48	0	0 B	mr1
drwxr-xr-x	nimmi	supergroup	0 B	Nov 23 18:42	0	0 B	mr2
drwxr-xr-x	nimmi	supergroup	0 B	Nov 21 00:51	0	0 B	mr3
drwxr-xr-x	nimmi	supergroup	0 B	Nov 23 21:08	0	0 B	mr4
drwxr-xr-x	nimmi	supergroup	0 B	Nov 23 19:24	0	0 B	mr5
drwxr-xr-x	nimmi	supergroup	0 B	Nov 23 19:57	0	0 B	mr6
drwxr-xr-x	nimmi	supergroup	0 B	Nov 23 18:25	0	0 B	mr7
drwxr-xr-x	nimmi	supergroup	0 B	Nov 21 15:54	0	0 B	mr8

Showing 1 to 9 of 9 entries — Previous — 1 — Next

Hadoop, 2022.

Apache Spark for Distributed Processing

Why Apache Spark?

•**Speed:** Spark processes data faster due to in-memory computation.

•**Scalability:** Can handle large datasets efficiently across clusters.

•**Ease of use:** PySpark provides a Python interface to work with Spark for distributed data processing.

•**Distributed Processing:**

•Data is loaded from HDFS directory and processed in parallel using Spark.

•Spark performs various data transformations and prepares it for machine learning algorithms.

Plotly for Visualizing Insights

Dashboard Link-Book Dashboard

Visualization with Plotly-Dash: A web framework built on Plotly for creating interactive data visualizations.

•**Purpose:** Visualize key insights from the processed data.

•**Features:**

- Interactive plots for user-driven analysis.
- Real-time data updates.
- Easy integration with the Flask web application.

ML Models

Machine Learning Models:

- **Collaborative Filtering:** Uses user ratings and preferences to recommend books.
- **Content-based Filtering:** Recommends books based on attributes like genre, author, and keywords.
- **Apache Spark MLlib:** Used to implement and train machine learning models efficiently.
- **Recommendation Generation:** Personalized book recommendations based on user behavior and book characteristics.

Collaborative Model

The collaborative filtering model recommends books to users based on their interactions and ratings from similar users. It focuses on user preference patterns rather than explicit book content, leveraging latent factors (hidden characteristics) to predict ratings and suggest books.

Process:

- Preprocessing ensures data quality by filtering users with ratings and books with ratings. StringIndexer converts categorical identifiers to numerical indices for ALS compatibility.
- The ALS algorithm learns to identify relationships between users and books, predicting ratings for unseen books using patterns from similar users.

Benefits:

- **Personalization:** Recommends books based on preferences of similar users, introducing users to new, highly rated books.
- **Enhanced User Experience:** Broadens user choices and improves satisfaction by aligning recommendations with their tastes.

Content Based Model

This system recommends books by analyzing textual features such as titles, descriptions, authors, and categories.

Process:

- **Data Preparation:** Cleans dataset, removes duplicates, handles missing values, and consolidates textual fields.
- **Feature Engineering:** Text is tokenized, stop words are removed, and features are converted into vectors using CountVectorizer.
- **Similarity Detection:** MinHashLSH computes approximate Jaccard similarity to identify top N similar books, stored in a CSV file with book IDs, titles, and recommendations.

Advantages:

- **Scalable:** Spark enables efficient handling of large datasets.
- **Efficient:** MinHashLSH reduces computation time without accuracy loss.

UI Page Designing

UI Link-Book Recommendation System

Web Interface Design:

•Built using **Flask**.
•Dynamic pages using Jinja2 templating engine.

•**Main Pages:**
    •**Home page:** Displays popular books.
    •**Book Details:** Shows detailed information about selected books.
    •**Recommendations:** Personalized recommendations for the user.

•**Search functionality** for easy exploration of books.

•**Responsive Design:** Ensures the UI adapts well across different devices.

Future Steps

Improving Model Accuracy:

•Incorporating more advanced algorithms like neural networks for better recommendations.

Enhancing User Experience:

- Adding features like book reviews and ratings.
- Implementing social media integrations for sharing recommendations.

Scaling the System:

- Expanding the recommendation system to handle larger datasets and more users.

Conclusion

Summary:

- The project successfully implemented a scalable and efficient book recommendation system using Hadoop, Apache Spark, and machine learning.
- The system offers personalized recommendations and visual insights.
- The solution is cost-effective and suitable for large-scale data processing.

Key Achievements:

- Built a robust distributed system for real-time book recommendations.
- Enabled users to explore data insights through interactive visualizations.
- Delivered a user-friendly web interface for seamless interaction.

THANK YOU !!!