Ingesting Data into OpenSearch

Introduction

This document outlines the step-by-step process of ingesting product sales data from the Amazon Products Sales Dataset into OpenSearch. The dataset comprises product details categorized across 142 distinct categories, sourced from Amazon's website (https://www.kaggle.com/datasets/lokeshparab/amazon-products-dataset). Initially stored in CSV format (`Amazon-Products.csv`), each product record includes attributes such as name, main_category, sub_category, image, link, ratings, no of ratings, discount_price, and actual_price.

Data Preprocessing and Cleaning

Before ingesting the data into OpenSearch, the dataset was split into multiple CSV files (`amazon_xaa.csv`, `amazon_xab.csv`, `amazon_xac.csv`) for better manageability. These files were cleaned using Python scripts to remove duplicates,fill missing values,fixed errors,standardized formats,and remove unnecessary fields.

Step-by-Step Guide to Ingesting Data into OpenSearch

1. OpenSearch Index Creation

To manage product data efficiently, we'll create two primary indexes in OpenSearch:

Mapping Definitions:

Mapping for `amazon_data` Index:

```
{
  "mappings": {
    "properties": {
      "actual_price": {
        "type": "double"
      },
      "discount_price": {
        "type": "double"
      },
      "id": {
        "type": "integer"
      },
      "link": {
        "type": "text"
      },
      "main_category": {
        "type": "keyword"
      },
      "name": {
        "type": "search_as_you_type",
```

```
        "fields": {
          "keyword": {
            "type": "keyword"
          }
        }
      },
      "no_of_ratings": {
        "type": "integer"
      },
      "ratings": {
        "type": "double"
      },
      "sub_category": {
        "type": "keyword"
      }
    }
  },
  "settings": {
    "index": {
      "number_of_shards": "1",
      "number_of_replicas": "1"
    }
  }
}
```

Mapping for `amazon_aggregated_index` Index:

```
{
  "mappings": {
    "properties": {
      "main_category": {
        "type": "keyword"
      },
      "sub_category": {
        "type": "keyword"
      }
    }
  }

}
```

2. Steps to Dump Data into Both Indexes

For `amazon_data` Index:

Step A: Convert CSV to JSON:

Assume you have CSV files (`clean_data_xaa.csv`, `clean_data_xab.csv`, `clean_data_xac.csv`) containing product details. Convert them to JSON format (`clean_data_xab_output.json`) using Python:

```
import csv
```

```
import json

input_csv_files = ['clean_data_xaa.csv', 'clean_data_xab.csv', 'clean_data_xac.csv']
output_json_file = 'clean_data_xab_output.json'

for input_csv_file in input_csv_files:
    with open(input_csv_file, 'r') as csvfile:
        csvreader = csv.DictReader(csvfile)
        data = list(csvreader)

    with open(output_json_file, 'a') as jsonfile:
        for record in data:
            create_record = {
                "create": {
                    "_index": "amazon_data",
                    "_id": record['id']
                }
            }
            json.dump(create_record, jsonfile)
            jsonfile.write('\n')

            data_record = {
                "name": record['name'],
                "main_category": record['main_category'],
                "sub_category": record['sub_category'],
                "link": record['link'],
                "ratings": float(record['ratings']),
                "no_of_ratings": int(record['no_of_ratings']),
                "discount_price": float(record['discount_price']),
                "actual_price": float(record['actual_price'])
            }
            json.dump(data_record, jsonfile)
            jsonfile.write('\n')
```

Step B: Bulk Indexing JSON Data into OpenSearch:

Execute the Python script (`bulk_indexing.py`) to upload JSON data to OpenSearch using curl commands or an OpenSearch client library.

For `amazon_aggregated_index` Index:

Step C: Python Script to Load Data into OpenSearch:

Use Python to load data from `Amazon-Products.csv` into the `amazon_aggregated_index` index:

```
import pandas as pd
from opensearchpy import OpenSearch
import numpy as np
```

```
# Load CSV data into a DataFrame
csv_file_path = 'Amazon-Products.csv'
df = pd.read_csv(csv_file_path)

# Clean NaN values
df = df.replace({np.nan: ''})

# Initialize OpenSearch client
client = OpenSearch(
    hosts=[{'host': 'localhost', 'port': 9200}],
    http_auth=('admin', 'admin@123'),  # Change credentials as needed
    use_ssl=False
)

# Index name
index_name = 'amazon_aggregated_index'

# Ensure index exists with mapping
index_mapping = {
    "mappings": {
        "properties": {
            "main_category": {"type": "keyword"},
            "sub_category": {"type": "keyword"}
        }
    }
}

# Create index if it doesn't exist
if not client.indices.exists(index=index_name):
    client.indices.create(index=index_name, body=index_mapping)

# Index documents
for i, row in df.iterrows():
    doc = row.to_dict()
    try:
        response = client.index(
            index=index_name,
            body=doc,
            id=i
        )
        print(response)
    except Exception as e:
        print(f"Error indexing document ID {i}: {e}")

print("Data insertion into amazon_aggregated_index completed.")
```

3. Ingest Pipeline for Reindexing

An ingest pipeline (`ignore_invalid_fields`) is set up to handle potential errors during reindexing:

```
PUT /_ingest/pipeline/ignore_invalid_fields
{
  "description": "Pipeline to ignore invalid field values during reindexing",
  "processors": [
    {
      "script": {
        "lang": "painless",
        "source": """
          if (ctx.containsKey('ratings')) {
            try {
              Double.parseDouble(ctx.ratings.toString());
            } catch (NumberFormatException e) {
              ctx.ratings = 1;
            }
          }
          if (ctx.containsKey('no_of_ratings')) {
            try {
              Integer.parseInt(ctx.no_of_ratings.toString());
            } catch (NumberFormatException e) {
              ctx.no_of_ratings = 0;
            }
          }
        """
      }
    }
  ]
}
```

Explanation:

- **Description:** Describes the purpose of the pipeline to handle invalid field values.
- **Processors:** Uses a Painless script to convert `ratings` to a double and `no_of_ratings` to an integer. If conversion fails due to non-numeric values, default values (1 for ratings and 0 for no_of_ratings) are assigned.

4. Reindexing Data into `amazon_data_v2` Index

Reindex data from `amazon_data` to `amazon_data_v2` using the `ignore_invalid_fields` pipeline:

```
POST /_reindex?timeout=60m
{
  "source": {
    "index": "amazon_data"
  },
  "dest": {
    "index": "amazon_data_v2",
    "pipeline": "ignore_invalid_fields"
  }
}
```

Explanation:

- **Source:** Specifies the source index (`amazon_data`) from which data will be reindexed.
- **Destination (dest):** Specifies the destination index (`amazon_data_v2`) where reindexed data will be stored.
- **Pipeline:** Utilizes the `ignore_invalid_fields` pipeline to preprocess data during reindexing, ensuring that invalid field values for `ratings` and `no_of_ratings` are handled according to the defined script.

Conclusion

By following these steps, you can effectively ingest and manage product sales data from the Amazon Products Sales Dataset into OpenSearch. This approach ensures data integrity, efficient querying, and scalability for handling large datasets. Adjust paths, index names, and credentials as per your environment and requirements.