Data Analytics 1: Assignment 2 Report

Team 41

September 5, 2024

1 Introduction

This report explores data cube extraction techniques using the **BUC (Bottom-Up Cube) algorithm** and **Attribute-Oriented Induction (AOI)** on a dataset of electric vehicles. The dataset contains various attributes like *City*, *Model Year*, *Electric Vehicle Type*, and others. The goal is to apply BUC to generate interesting cubes and analyze the efficiency of different implementations (in-memory and out-of-memory). We will also explore the generalization of data using AOI and compare the two techniques.

2 Attribute-Oriented Induction (AOI) Implementation

AOI is a data generalization technique that simplifies data by removing or generalizing certain attributes. In our implementation, we focused on the generalization of attributes such as *Electric Utility* and *Make* by removing them to reduce complexity and highlight core patterns.

AOI Code Example:

```
def attribute_oriented_induction(data, attributes_to_remove):
    generalized_data = data.drop(columns=attributes_to_remove)
    return generalized_data
```

The result is a more streamlined dataset that emphasizes core features while removing redundant details.

3 BUC Algorithm Implementation

3.1 In-Memory BUC

The in-memory implementation of BUC assumes that all the data can fit into memory. The algorithm builds cubes by progressively aggregating data based on different combinations of dimensions, starting from a single dimension to multiple combinations.

3.2 Out-of-Memory BUC

In the out-of-memory BUC implementation, we simulate a scenario where the data exceeds memory limits by partitioning the dataset into smaller chunks, processing them individually, and combining the results.

Key Functions:

- **Partitioning Data**: We divide the dataset into smaller chunks that can fit into memory.
- **Recursive Cube Building**: After partitioning, the BUC algorithm processes each chunk and generates cubes.

Code Snippet:

```
def partition_data_to_disk(df, chunk_size, base_filename):
    num_chunks = len(df) // chunk_size + 1
    for i in range(num_chunks):
        chunk = df[i*chunk_size:(i+1)*chunk_size]
        chunk.to_csv(f'{base_filename}_part_{i}.csv', index=False)
```

4 Performance Analysis

For performance analysis, we measured the runtime of the BUC algorithm while varying two parameters:

- **minsup (Minimum Support)** vs. **Runtime** (Fixed Allotted Memory).
- **Allotted Memory** vs. **Runtime** (Fixed minsup).

4.1 a. Plot: minsup vs. Runtime (Fixed Memory)

As minsup increases, we observed a decrease in runtime due to fewer groupings being processed, as lower minsup requires more data to be aggregated.

4.2 b. Plot: Allotted Memory vs. Runtime (Fixed minsup)

With larger memory (chunk size), the runtime decreased significantly, as more data can be processed in a single pass, reducing the number of chunks that need to be written to disk and read back.

Example Plot Code:

```
plt.plot(minsup_values, runtime_minsup, marker='o')
plt.title('minsup vs Runtime')
plt.xlabel('minsup')
plt.ylabel('Runtime (seconds)')
```

5 Optimization Technique

The optimization technique used for BUC is **Iceberg Cubing**. This technique involves pruning cubes that do not meet a specified minimum support (minsup). By ignoring low-support groupings, we significantly reduce the number of cubes generated, improving both memory usage and runtime.

Impact of Optimization: The iceberg cubing method led to faster execution times, especially for large datasets with many potential groupings. This is particularly evident when the minsup is set to a higher value, as it drastically reduces the number of computations.

6 Comparison of BUC and AOI

- **Purpose and Use Cases**:
  - **BUC Algorithm**: The BUC algorithm is used to compute data cubes efficiently, primarily for **multidimensional aggregation**. It is ideal for exploratory data analysis, especially in OLAP (Online Analytical Processing) systems.
  - **AOI**: Attribute-Oriented Induction is used for **data generalization** and the discovery of characteristic rules. It reduces data complexity by removing or summarizing attributes.
- **Types of Insights**:
  - **BUC Algorithm**: It discovers **aggregate patterns** across multiple dimensions, making it useful for spotting trends and outliers in large, multidimensional datasets.
  - **AOI**: AOI is best suited for identifying **characteristic rules** by generalizing the dataset.
- **Computational Efficiency and Scalability**:
  - **BUC Algorithm**: BUC can be computationally expensive for high-dimensional data. The out-of-memory implementation adds complexity when datasets do not fit into memory.
  - **AOI**: AOI is generally more efficient for smaller datasets because it focuses on **data reduction**.
- **Interpretability**:
  - **BUC Algorithm**: The output of BUC, such as data cubes, is typically large and complex, requiring further analysis to interpret.
  - **AOI**: AOI produces generalized summaries that are easier to interpret.
- **Scenarios of Preference**:

- **BUC Algorithm**: Preferred when detailed, multidimensional aggregations are needed, such as in business intelligence.
- **AOI**: Suitable for exploratory data analysis where reducing cognitive load is important.

7 Conclusion

In this assignment, we implemented and analyzed two key techniques: the **BUC algorithm** and **Attribute-Oriented Induction**. Through performance analysis, we demonstrated that BUC is highly efficient for processing large datasets, especially when optimized with iceberg cubing. AOI, while useful for data simplification, is less suited for detailed, multidimensional analysis. Both methods offer unique strengths depending on the use case.