DOI 10.51231/2667-9507-2024-007-03-76-79

Optimizing Memory Usage in Python (Pandas)

Giorgi Kuchava[1,2], Maia Mantskava[2,3], Nana Momtselidze[2,4]

[1]*Business And Technology University, Tbilisi, Georgia*

[2]*Ivane Beritashvili Center of Experimental Biomedicine, Tbilisi, Georgia*

[3]*European University, Tbilisi, Georgia*

[4]*Kutaisi University, Kutaisi, Georgia*

Abstract

This article explores Python's prominence in Data Science, Data Analytics, and Machine Learning, attributing its widespread adoption to its user-friendly nature, robust online community, and powerful data-centric libraries such as Pandas, NumPy, and Matplotlib. It delves into the challenges of managing extensive datasets and emphasizes the importance of memory utilization in navigating substantial data. The Pandas library's info() and memory_usage() methods are discussed as essential tools for assessing and optimizing dataframe memory consumption. The article demonstrates how changing data types, particularly for object columns, to the category datatype significantly reduces memory usage without altering the dataframe's appearance. The strategic adjustment of numerical column data types based on value range, illustrated with the age column as an example, is explored as a means of achieving precision and memory efficiency. The article highlights the considerable reduction in memory requirements by transitioning from float64 to float16 for columns containing floating-point numbers. Overall, this comprehensive exploration provides valuable insights into effective strategies for memory optimization in Pandas dataframes, catering to both categorical and numerical data, contributing to enhanced computational efficiency and significant memory savings.

KEYWORDS: Python Programming; Pandas Library; Memory Optimization; Data Science; Data Analytics; Machine Learning

Introduction

Python stands out as a highly prevalent programming language in the realms of Data Science, Data Analytics, and Machine Learning. Its widespread adoption can be attributed to its beginner-friendly nature, a robust online community of learners, and the presence of powerful data-centric libraries such as Pandas, NumPy, and Matplotlib. These libraries facilitate the seamless management and manipulation of extensive datasets. As a result, Python has emerged as the preferred language for professionals in the fields of Data Science and Data Analytics.

The Pandas library in Python provides a powerful capability for storing tabular data through a specialized data structure known as a dataframe. With a pandas dataframe, managing substantial amounts of tabular data becomes effortlessly accessible, thanks to its intuitive handling of row and column indices. This flexible structure accommodates data storage with hundreds of columns (fields) and thousands of rows (records).

Navigating substantial datasets requires a thoughtful approach to memory utilization. The challenge arises when confronted with an extensive volume of data, often leading to memory scarcity. Exhausting the available RAM may result in program crashes accompanied by the elusive MemoryError, posing a formidable challenge. Prioritizing memory management becomes paramount in such scenarios. Beyond the mitigation of potential crashes, streamlined memory usage enhances computational efficiency, ultimately saving valuable time.

Methods

The Pandas **info()** method serves as a quick gauge of a dataframe's memory consumption, offering a total memory overview. Simply by setting the **memory_usage** argument to "deep" within the **info()** method, we obtain the comprehensive memory allocation for the entire Pandas dataframe.

Nevertheless, when a more granular insight into memory usage per column is desired, the **memory_usage()** method comes to the rescue. This method delves into the specifics, providing a detailed breakdown of the memory consumed by each column in the dataframe. Executing the **memory_usage()** method returns a Pandas series, furnishing the memory space occupied by individual columns in bytes. Enriching this

with the deep argument set to True unravels the complete memory panorama of the dataframe's columns.

Typically, columns with an object datatype, such as gender, occupation, and zip code in our dataset, tend to consume substantial memory due to storing strings. Strings inherently occupy more space than numerical data types like integers and floating-point numbers, thereby inflating memory usage significantly.

A strategic solution involves converting select object columns to the category datatype. Consider the gender column, which inherently accommodates only two values, either "M" or "F". Switching the datatype from object to category optimizes storage by representing gender records as integer codes instead of strings. This simple alteration results in a remarkable reduction in memory usage—from 58,466 bytes to a mere 1,147 bytes, constituting a noteworthy 98% space-saving.

Extending this approach to other pertinent object columns in the dataframe yields substantial memory savings and acts as a preventive measure against potential MemoryError occurrences.

For numerical columns, an additional tactic involves tailoring the data type based on the value range. Take the age column in our dataset, where values span from 7 to 73. Recognizing that this range aligns with an 8-bit binary representation, we can efficiently store age as an 8-bit integer instead of the default 64-bit integer in newer Pandas versions. This strategic adjustment minimizes the required bits, resulting in a notable decrease in memory usage, particularly beneficial when dealing with large datasets.

Results

When transitioning the data type of the age column from int64 to int8, the space occupied by the column experiences a remarkable decrease-from 7544 bytes to 943 bytes, marking an impressive 87.5% reduction in space.

Furthermore, exploring alternative integer data types presents opportunities for optimization. Int16, accommodating a range of – 32,768 to +32,767, and int32, supporting a more extensive range from – 2147483648 to +2147483647, offer flexibility based on the required value range. The choice between int8, int16, or int32 hinges on the specific range of values within the dataset.

The following table outlines the complete spectrum of values representable by different integer data types:

Data Type	Range of Values
int8	-128 to +127
int16	-32,768 to +32,767
int32	-2,147,483,648 to +2,147,483,647
int64	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

By aligning the data type with the appropriate range, we strike a balance between precision and memory efficiency, contributing to optimized storage in the age column.

Likewise, optimizing memory usage extends to columns containing floating-point numbers. Shifting from float64 to float16 as the data type introduces a substantial reduction in space requirements.

Discussion

This article delves into two key methods within Pandas for gauging dataframe memory consumption: the **info()** method and the **memory_usage()** method. Expanding on memory optimization strategies, we explored two effective approaches. The first involves transforming the data type of object columns to the category when dealing with categorical data. Despite maintaining the dataframe's appearance, this modification yields a noteworthy reduction in memory usage.

The second strategy revolves around adjusting the data type of numerical columns based on the value range, applicable to both integers and floating-point numbers. This proactive measure significantly contributes to memory efficiency.

References

1. Matthew Rosch, Learning Pandas 2.0: A Comprehensive Guide to Data Manipulation and Analysis for Data Scientists and Machine Learning Professionals, GitforGits, 2023, 175 p.
2. https://www.analyticsvidhya.com/blog/2021/04/how-to-reduce-memory-usage-in-python-pandas/ Last Checked-06.02.2024
3. Wes McKinney, Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter 3rd Edition, O'Reilly Media, 2022, 579 p.