CMPT 353 Data Compression Analysis

Michael Goldenberg - 301398854
Gabriel Cheng - 301305668

The Problem:

Our project explores various different compression and decompression algorithms offered in Python. Mainly we looked into gzip, zlib, bz2, lzma, zipfile, and tarfile compression algorithms. We wanted to determine what method to use for different file sizes and file types to achieve optimal compression ratio, compression speed, and decompression speed. Additionally, we compared how HDD, SSD, and NVMe drives compare for compression and decompression.

The Data:

This project required two different categories of data. For the first, we collected a bunch of data files with different sizes and formats. These files were downloaded from the CMPT353 data sets website. We used the weather dataset for CSV files, NASA logs and page counts for type “File”, Reddit and XYZ data for JSON, and finally wordcount for TXT. Altogether this was about 2.2GB worth of data. It is worth noting that the actual information contained in the data sets was not relevant, we simply needed files to use for generating the second data set.

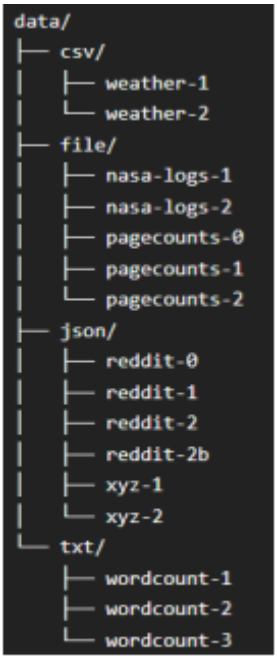

Figure 1: File Tree

For the second category of data, we had to generate it ourselves. The time_compression function was used to generate the compression_times.csv file which is the data we used for analysis. The function starts by getting all subdirectories containing the data described above. The file tree (figure 1) was organized into subdirectories to keep track of file types. The next step is to iterate through all of the files within the subdirectories. For each file, we collect the compression time, decompression time, compression ratio, compression type, file size, storage type, and file_type. This is done for each of the compression/decompression methods mentioned prior. This data is then put into a pandas data frame and exported as a CSV. After a long wait, we came out with nearly 7000 data points.

Data Analysis:

The first question we wanted to answer was if the different compression types had different compression speeds and compression ratios. For this, we wanted to see if there is a difference between the various compression methods' mean compression time and mean compression ratio. However, in our data set, we have a lot of values to compare, for example, there is compression time, compression ratio, and decompression time, but we have those values for 6 different compression methods, 4 different file types, and on 3 different types of storage. So, we

can not simply run a T-test to check for different means of all the combinations as we would likely encounter P-value hacking. Instead, we ran Anova tests, and if the p-value was small enough we performed posthoc analysis to determine which specific means are different. Seeing as we still performed a significant amount of ANOVA tests we decided to use a Bonferroni correction (0.05/3) to further lower the chance of p-value hacking.

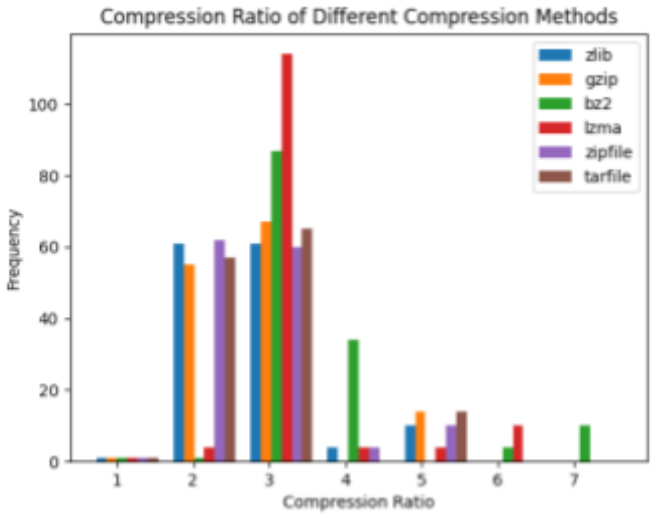

Figure 2: Compression Ratio Histogram

For the first Anova test, we compared the mean compression ratio of the different compression methods. Before doing the ANOVA test we plotted the mean compression ratios(figure 2) to check for a close-to-normal distribution, we did not need to perform the “stats.normaltest” function as we have enough data points. We can now proceed with the ANOVA test, the “compression_analysis_anova.py” file returns a p-value of “3.87e-197” so we are able to continue with Tukey's HSD post hoc analysis. We can see the results of the post hoc analysis in the “pairwise_tukeyhsd output”(figure 3).

We ran a similar test as above to compare the means of compression times and found a p-value of 2.51e-83. The post hoc results showed that there was a difference in means between lzma and the rest of the algorithms.

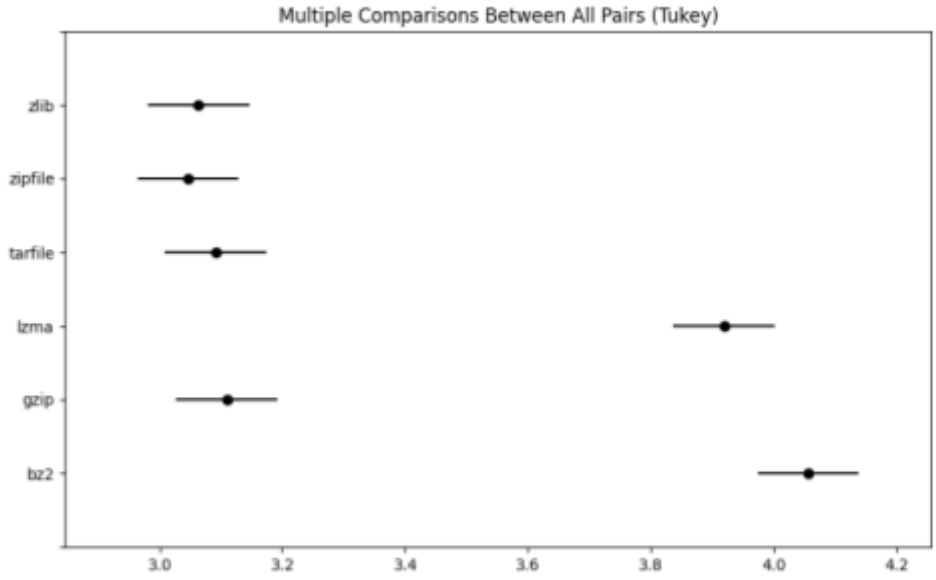

Figure 3: Compression Ratio posthoc

Unfortunately, the decompression times did not seem to be normal enough to run an ANOVA test on them, however, it is worth noting that the decompression times were significantly faster

than the compression times, so they are not as big of a factor when choosing a compression algorithm.

Our final ANOVA test was performed on the mean compression times for different storage types. The goal of this test was to conclude if there is a difference in mean compression time when compressing on an HDD, SSD, or NVMe drive. We once again checked normality and went on with the testing to find a p-value of 1.358e-7. The post hoc results showed that there is a difference in the HDD/SSD and HDD/NVMe but not SSD/NVMe.

From the statistical tests above we can see that the compression method and storage type relate to the compression ratio and speed. The file type is also a factor that would change both, we know this because a txt file with little structure can not be compressed as well as a JSON file that contains repeated labels on every line. So, we will use machine learning techniques, specifically different regression techniques to create a model that can predict the compression method that will yield the best compression ratio, or compression time based on the file size, file type, and storage you are using.

We used three different regression models, K nearest neighbors, random forests, and MLP to try and find the best predictor. For all of the models, we split the data into a training and testing group to then use for scoring the accuracy of the models on trained and unseen data. The X data contained the fields: file size, compression type, file type, and storage type. While the y data contained: compression time, compression ratio, and decompression time. The compression type, storage type, and file type data were non-numerical so we had to use a fit_transform function to have it work with the regression models.

K nearest neighbors performed the best with 7 neighbors, additionally, we created a pipeline to apply a standard scaler to improve performance.

Random forests did best with 50 estimators and a max depth of 15, scaling did not result in a better score so it was not applied

The MLP regressor was harder to tune as it would often not converge or predict poorly but 8 hidden layers, a logistic activation function, and an ‘lbfgs’ solver seemed to do best. We also applied a standard scaler to the MLP regressor. Finally, we capped it at 10000 iterations as it would sometimes take a very long time to converge depending on the test, and train split.

	Training Score	Testing Score
K Nearest Neighbors	0.788	0.795
Random Forest	0.990	0.956
MLP	0.670	0.655

Figure 5

The training and testing scores (figure 5) show that the random forests had (by far) the highest score and should be used as our model for prediction.

Findings and conclusions

Using the Random Forest classifier, we used it to predict what compression methods is the best for minimum compression time and maximum compression ratio. Using the file's type, size, and the storage type it is on, our model predicted and output into a csv:

Filename

Compression method for minimum time,

Time for minimum time compression,

Time for maximum ratio compression,

Compression method for maximum compression ratio,

Maximum compression ratio

With a test folder of different types of files to predict on and a csv for the prediction outputs, we notice that the SSD and NVMe option barely affect the results of the prediction, but when using HDD, the prediction output changes.

We predicted small files less than 10MB, and on HDD, compression type for minimum time was all tarfile, and for maximum compression ratio, all zlib.

Using the SSD option, the minimum time was between tarfile and zlib, with one gzip, while for maximum compression ratio, between bz2, tarfile, and zlib.

FInally with NVMe option, the minimum time was mainly zlib, then tarfile, and one gzip. For maximum compression ratio, it was mainly zlib, then tarfile, and one bz2.

For all 3 storage drives, the common compression type form minimum time was tarfile, and the prediction of SSD and NVMe were similarly predicted to the microseconds, while for HDD it was milliseconds.
For compression ratio, the common of the 3 storages were zlib, but HDD had higher compression ratio than SSD and NVMe, but this could be due to different computers collecting the data for SSD/NVMe and HDD

Limitations

The HDD, SSD, and NVMe data collection were done on 2 separate computer systems, which could affect the data collected during compression and decompression.

The data set used couldn't be a huge variety of different sizes, since we have normal computer systems and it would take hours or days, so we had to reduce the amount of data collection, which resulted in less training data.

More training data would allow for better model training.

Project Experience Summary:

Michael Goldenberg - 301398854

Compression Algorithm Analysis - Computational Data Science, SFU July - August 2023

- Generated a dataset with compression speeds and ratios for various compression methods
- Utilized several ANOVA & Tukey HSD tests to find differences in the performance of compression based on compression type, storage type, etc…
- Trained ML regression models to predict the best compression type for optimal compression speed or compression ratio with up to 96% accuracy score.

Gabriel Cheng - 301305668

Compression Analysis - Computational Data Science, SFU July - Aug 2023

- Researched through video and web resources to understand more about compression