جامعة نيويورك أبوظبي

NYU | ABU DHABI

Lossless Compression of Large Traffic Datasets Using Feature-Based Selective Sampling

Under the kind guidance of
Professor Saif Eddin Jabari

Aman Sunesh
Damiane Kapanadze
Seoyoon Jung

Course Name:
Introduction to Data Analysis for Engineers
Section 001
Course Code: ENGR-UH 2027
Fall 2024

New York University Abu Dhabi
December 12, 2024

Contents

Abstract

This project explores the compression of urban traffic datasets while preserving the ability for perfect decompression. The study implements advanced matrix completion techniques, sampling strategies, and efficient encoding/decoding methods to achieve optimal compression. Results demonstrate the utility of leverage score sampling and rank-reduction techniques for sparse data matrices.

1 Introduction

The growth of urban areas has led to massive traffic datasets that require efficient handling for real-time decision-making. Lossless compression ensures no data loss during reconstruction, enabling accurate analyses and predictions. This project aims to investigate the limits of compressing such datasets while retaining perfect decompression capabilities.

2 Methodology

2.1 Problem Definition

Given a large data matrix $D \in \mathbb{R}^{m \times n}$, the goal is to find a reduced representation $\hat{D}$ with fewer non-zero entries while satisfying:

$$\min_{\hat{D}} \operatorname{rank}(\hat{D}) \quad \text{subject to} \quad \|\hat{D} - D\|_F = 0,$$

where $\|A\|_F$ denotes the Frobenius norm. The problem is reformulated into a convex optimization problem using nuclear norm minimization.

2.2 Dataset Description

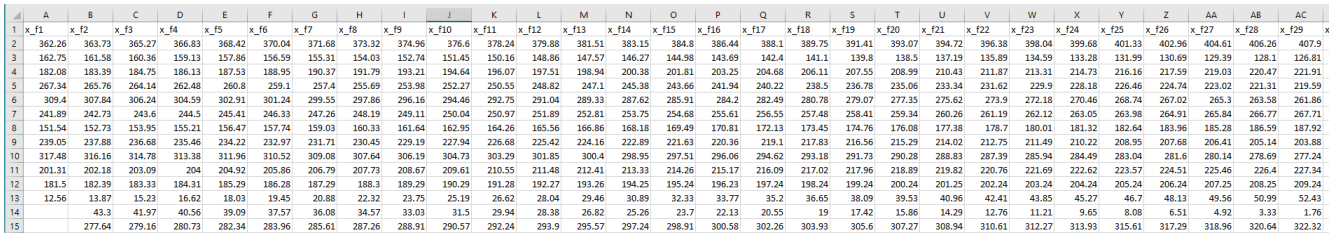
The dataset used is **pNeuma**, collected by 10 drones over 5 days, with five flight sessions per day lasting 2.5 hours each. The dataset spans observations like vehicle trajectories and includes detailed metadata. Specifically, the dataset is in CSV format with the following structure:

- **First 10 Column labels**: `track_id`, `type`, `traveled_d`, `avg_speed`, `lat`, `lon`, `speed`, `lon_acc`, `lat_acc`, `time`.
- **Subsequent Column labels**:
  - `track_id`, `type`, `traveled_d`, `avg_speed`, followed by repeated updates for `(lat, lon, speed, lon_acc, lat_acc, time)`.

For this project, the focus is on compressing the spatiotemporal data represented in this matrix structure. Specifically, compression encoding and decoding is performed on the pNeuma dataset of Oct 30, 2018, 8:30-9:00, Drone 1.

2.3 Data Preprocessing for Compression

In order to start the data compression, we had to modify the initial data. Instead of taking bulks of different data at one timeframe, we reorganized the dataset by taking vehicles in rows and extracting all data of the latitude across all timeframes. This was followed by longitude, longitudinal acceleration, and lateral acceleration. This restructuring allows for better decompression since the related data are grouped together.



x_f1	x_f2	x_f3	x_f4	x_f5	x_f6	x_f7	x_f8	x_f9	x_f10	x_f11	x_f12	x_f13	x_f14	x_f15	x_f16	x_f17	x_f18	x_f19	x_f20	x_f21	x_f22	x_f23	x_f24	x_f25	x_f26	x_f27	x_f28	x_f29
362.26	363.73	365.27	366.83	368.42	370.04	371.68	373.32	374.96	376.6	378.24	379.88	381.51	383.15	384.8	386.44	388.1	389.75	391.41	393.07	394.72	396.38	398.04	399.68	401.33	402.96	404.61	406.26	407.9
162.75	161.58	160.36	159.13	157.86	156.59	155.31	154.03	152.74	151.45	150.16	148.86	147.57	146.27	144.98	143.69	142.4	141.1	139.8	138.5	137.19	135.89	134.59	133.28	131.99	130.69	129.39	128.1	126.81
182.08	183.39	184.75	186.13	187.53	188.95	190.37	191.79	193.21	194.64	196.07	197.51	198.94	200.38	201.81	203.25	204.68	206.11	207.55	208.99	210.43	211.87	213.31	214.73	216.16	217.59	219.03	220.47	221.91
267.34	265.76	264.14	262.48	260.8	259.1	257.4	255.69	253.98	252.27	250.55	248.82	247.1	245.38	243.66	241.94	240.22	238.5	236.78	235.06	233.34	231.62	229.9	228.18	226.46	224.74	223.02	221.31	219.59
309.4	307.84	306.24	304.59	302.91	301.24	299.55	297.86	296.16	294.46	292.75	291.04	289.33	287.62	285.91	284.2	282.49	280.78	279.07	277.35	275.62	273.9	272.18	270.46	268.74	267.02	265.3	263.58	261.86
241.89	242.73	243.6	244.5	245.41	246.33	247.26	248.19	249.11	250.04	250.97	251.89	252.81	253.75	254.68	255.61	256.55	257.48	258.41	259.34	260.26	261.19	262.12	263.05	263.98	264.91	265.84	266.77	267.71
151.54	152.73	153.95	155.21	156.47	157.74	159.03	160.33	161.64	162.95	164.26	165.56	166.86	168.18	169.49	170.81	172.13	173.45	174.76	176.08	177.38	178.7	180.01	181.32	182.64	183.96	185.28	186.59	187.92
239.05	237.88	236.68	235.46	234.22	232.97	231.71	230.45	229.19	227.94	226.68	225.42	224.16	222.89	221.63	220.36	219.1	217.83	216.56	215.29	214.02	212.75	211.49	210.22	208.95	207.68	206.41	205.14	203.88
317.48	316.16	314.78	313.38	311.96	310.52	309.08	307.64	306.19	304.73	303.29	301.85	300.4	298.95	297.51	296.06	294.62	293.18	291.73	290.28	288.83	287.39	285.94	284.49	283.04	281.6	280.14	278.69	277.24
201.31	202.18	203.09	204	204.92	205.86	206.79	207.73	208.67	209.61	210.55	211.48	212.41	213.33	214.26	215.17	216.09	217.02	217.96	218.89	219.82	220.76	221.69	222.62	223.57	224.51	225.46	226.4	227.34
181.5	182.39	183.33	184.31	185.29	186.28	187.29	188.3	189.29	190.29	191.28	192.27	193.26	194.25	195.24	196.23	197.24	198.24	199.24	200.24	201.25	202.24	203.24	204.24	205.24	206.24	207.25	208.25	209.24
12.56	13.87	15.23	16.62	18.03	19.45	20.88	22.32	23.75	25.19	26.62	28.04	29.46	30.89	32.33	33.77	35.2	36.65	38.09	39.53	40.96	42.41	43.85	45.27	46.7	48.13	49.56	50.99	52.43
	43.3	41.97	40.56	39.09	37.57	36.08	34.57	33.03	31.5	29.94	28.38	26.82	25.26	23.7	22.13	20.55	19	17.42	15.86	14.29	12.76	11.21	9.65	8.08	6.51	4.92	3.33	1.76
	277.64	279.16	280.73	282.34	283.96	285.61	287.26	288.91	290.57	292.24	293.9	295.57	297.24	298.91	300.58	302.26	303.93	305.6	307.27	308.94	310.61	312.27	313.93	315.61	317.29	318.96	320.64	322.32

Figure 1: Snippet of Transformed data

3 Encoding

The primary objective of this section of the project is to develop an encoding scheme that leverages feature-based selective sampling to compress urban traffic data effectively while making sure we can still recover all the important information later. This is crucial for analyzing traffic in real-time and making quick decisions based on the data.

One significant challenge was handling the high dimensionality of the traffic dataset, which led to computational inefficiencies during matrix completion. To mitigate this, we implemented sparse matrix representations using the CSR format, significantly reducing memory usage and speeding up computations. Additionally, ensuring the selection of a representative subset for encoding was critical for lossless decompression. We addressed this by incorporating feature-based selective sampling, which prioritizes data points based on feature variability, thereby enhancing the representativeness of the sampled subset.

3.0.1 Software and Tools

- **Programming Language:** We used Python to implement our code, leveraging its powerful libraries for numerical operations and data manipulation. And unlike Matlab, it is open source, so anyone with a laptop and access to the Internet can run our program.
- **Libraries:**
  - **NumPy** for numerical operations, essential for efficient data manipulation and matrix operations.
  - **Pandas** for data manipulation and reading/writing data, pivotal for preprocessing and handling structured data.
  - **Matplotlib** for visualizing data and results, crucial for analyzing trends and outcomes graphically.

- **SciPy** for sparse matrix operations and advanced mathematical functions, used extensively in the encoding and decoding processes.
- **Scikit-learn** for preprocessing tools such as scaling and imputing, and for evaluating model performance using metrics like mean squared error.

3.1 Implementation

Step 1: Using Feature-Based Selective Sampling.

Feature-Based Selective Sampling is a method that prioritizes and retains data points based on the calculated importance of each feature within a dataset. Importance is typically assessed by measuring the variability of features, with the assumption that more variable features carry more informative content. This approach allows for the reduction of dataset size while aiming to preserve essential information, making it particularly useful for handling large datasets efficiently.

Before we begin encoding or transforming our data, it is crucial to determine which features (or columns) in our dataset carry the most information. This is where the concept of feature importance comes into play. In this context, feature importance is calculated to decide how much attention or weight each feature should receive during the sampling process:

- **Variability as an Indicator of Importance:** In many datasets, features that vary more contain more information. The basic idea is that if a feature changes a lot across the data, it likely holds valuable insights about the variability of the data points.
- **Range Calculation:** One straightforward method to measure variability is by calculating the range of each feature, defined as the difference between the maximum and minimum values. A larger range suggests higher variability, hence, greater importance.

We could have chosen another metric to determine importance, such as variance, leverage scores, or entropy. However, variance can be skewed by extreme outliers, leverage scores are often complex to compute and interpret, and entropy-based approaches may require additional preprocessing. In contrast, the **range** requires minimal assumptions, making it a practical baseline method for importance assessment.

Here's how we can calculate and use feature importance in our project:

```
# Step 1: Calculate importance scores for each feature (column)
def calculate_feature_importance(X_true):
    """
    Calculate the range (max - min) for each feature.
    Higher range indicates higher variability and importance.
    """
```

```
    max_vals = np.max(X_true, axis=0)
    min_vals = np.min(X_true, axis=0)
    range_vals = max_vals - min_vals

    # Replace NaN ranges with 0
    range_vals = np.nan_to_num(range_vals, nan=0.0)
    return range_vals

importance_scores = calculate_feature_importance(X_true_standardized)
```

This function returns a numeric score for each feature, reflecting its relative importance based on variability.

Step 2: Normalize Importance Scores

After computing the importance scores for each feature, these scores are normalized to form a probability distribution. Each entry's probability in the matrix is proportional to its feature's importance, guiding the sampling process to retain more significant data points preferentially.

```
# Assign importance per entry by replicating importance scores across all IDs
importance_matrix = np.tile(importance_scores, (N, 1))

# Normalize importance scores to create a probability distribution
total_importance = np.sum(importance_matrix)
prob_matrix = importance_matrix / total_importance
```

Step 3: Sampling Based on Scaled Probabilities

Using the normalized importance scores, a scaled probability matrix is created to guide the retention of data entries. This matrix is adjusted by a predefined sparsity level, aimed at reducing the dataset size while retaining critical information.

```
# Define the desired sparsity level (e.g., retain 10% of the data)
retain_fraction = 0.01  # 1% of the total entries

# Calculate the scaled probabilities to achieve the retain_fraction
scaled_prob_matrix = prob_matrix * (retain_fraction * M * N / np.sum(prob_matrix))
```

Here, `retain_fraction` is a parameter that specifies the percentage of the dataset's total entries that should be retained after compression.

- **Increasing `retain_fraction`:** If you increase the `retain_fraction` (say from 10% to 20%), a larger portion of the original dataset is retained. This typically enhances the accuracy of any analyses or models run on the compressed dataset because more information is preserved. However, it also means that the size of the compressed dataset will be larger, potentially reducing the efficiency gains from compression.
- **Decreasing `retain_fraction`:** Lowering this value will result in a smaller compressed dataset, which can significantly increase storage and processing efficiency. However, reducing the amount of data retained can also lead to a loss of important information, potentially decreasing the accuracy or reliability of subsequent analyses.

Step 4: Generate Sampling Matrix

The matrix Ω is generated by comparing a matrix of uniformly distributed random values (between 0 and 1) to these probabilities. If a random value for a data point is less than its corresponding scaled probability, the data point is retained (1 in Ω); otherwise, it is not retained (0 in Ω.

```
Omega = (np.random.rand(N, M) < scaled_prob_matrix).astype(int)
```

For example,

Probability Distribution Matrix

	x-f1	x-f2	x-f3
ID1	0.8	0.2	0.5
ID2	0.8	0.2	0.5
ID3	0.8	0.2	0.5
ID4	0.8	0.2	0.5
ID5	0.8	0.2	0.5

Random Matrix

	x-f1	x-f2	x-f3
ID1	0.3	0.5	0.1
ID2	0.7	0.1	0.6
ID3	0.6	0.3	0.4
ID4	0.2	0.1	0.7
ID5	0.7	0.4	0.2

Omega Matrix (After Comparison)

	x-f1	x-f2	x-f3
ID1	1	0	1
ID2	1	1	0
ID3	1	0	1
ID4	1	1	0
ID5	1	0	1

Here,

- **High Thresholds (e.g., 0.8):** Important features are assigned higher probabilities in the probability distribution matrix, which increases their likelihood of being retained in the Omega matrix Ω and subsequently in matrix R. Since the majority of random values will be less than 0.8, comparing these values with the probability distribution ensures that these critical data points are preserved, resulting in a 1 in Omega Ω.
- **Low Thresholds (e.g., 0.2):** Features deemed less important are assigned lower probabilities, thereby increasing the likelihood that the corresponding data points will not be retained. As most random values will exceed 0.2, it is more likely that these data points will be excluded, resulting in a 0 in Omega Ω.

Step 5: Sparse Matrix Representation

The next step in our encoding process leverages sparse matrix representation, a powerful technique for efficiently storing and manipulating matrices with many zero or missing values. This approach is particularly effective for our compressed dataset R and sampling matrix Ω.

We utilize the Compressed Sparse Row (CSR) format to store our matrices. This format significantly reduces memory requirements by only recording non-zero elements and their positions. For our matrices R and Ω, the CSR format consists of three arrays:

1. **Data Array**: Stores all the non-zero elements of the matrix, read row-wise.
2. **Indices Array**: Contains the column indices of the non-zero elements, correlating directly to the elements in the Data Array.
3. **Indptr Array**: Holds the index positions in the Data Array where each row starts. This array has one more element than the number of rows in the matrix, where the last element gives the index just beyond the last element of the Data Array.

Consider the matrix R with the following entries:

$$R = \begin{bmatrix} 10 & 0 & 3 \\ 0 & 20 & 0 \\ 0 & 0 & 30 \\ 40 & 5 & 0 \\ 0 & 25 & 0 \end{bmatrix}$$

In a sparse matrix CSR format, only the non-zero values and their respective positions are stored.

- **Data Array:** This array stores all non-zero values from the matrix R. The values are read row-wise from top to bottom.

$$\text{Data Array: } [10, 3, 20, 30, 40, 5, 25]$$

- **Indices Array:** This array stores the column indices corresponding to each non-zero value in the Data Array. It also follows a row-wise storage from top to bottom.

$$\text{Indices Array: } [0, 2, 1, 2, 0, 1, 1]$$

- **Indptr Array:** This array represents the index in the Data Array where each row starts. The last value is one more than the last index of the Data Array, indicating the end of the last row.

$$\text{Indptr Array: } [0, 2, 3, 4, 6, 7]$$

Each entry in the Indptr Array points to the position in the Data Array where the respective row starts. The difference between consecutive entries in the Indptr Array gives the number of non-zero elements in the corresponding row of matrix R.

This representation drastically reduces storage requirements, especially for large, sparse datasets like our traffic data.

```
Omega_sparse = csr_matrix(Omega)
R_sparse = csr_matrix(R)
```

Step 6: Saving the Files

These sparse matrices are then saved using SciPy's `save_npz` function, which preserves the sparse structure:

```
save_npz(Omega_path, Omega_sparse)
save_npz(R_path, R_sparse)
```

This encoding scheme effectively compresses the large traffic dataset by selectively retaining the most important data points based on feature importance and probabilistic sampling. The use of sparse matrices further reduces storage requirements, making it suitable for handling large-scale urban traffic data.

4 Decoding and Results

$$\text{minimize} \quad \mathbf{X} \in \mathbb{R}^{N \times M} \qquad \sum_{(n,m) \in \Omega} (X_{n,m} - R_{n,m})^2 + \lambda \|\mathbf{X}\|_*$$

This formulation minimizes the reconstruction error while penalizing the nuclear norm $\|\mathbf{X}\|_*$, encouraging a low-rank solution.

$$\text{minimize} \quad \mathbf{C} \in \mathbb{R}^{N \times r}, \mathbf{P} \in \mathbb{R}^{M \times r} \qquad \sum_{(n,m) \in \Omega} \left((\mathbf{C}\mathbf{P}^\top)_{n,m} - R_{n,m} \right)^2 + \frac{\lambda}{2} \left(\|\mathbf{C}\|_F^2 + \|\mathbf{P}\|_F^2 \right)$$

This reformulation approximates $\mathbf{X}$ as $\mathbf{C}\mathbf{P}^\top$, with Frobenius norm penalties controlling factor complexity.

Block coordinate descent alternates updates for $\mathbf{C}$ and $\mathbf{P}$, making the optimization faster and suitable for large, sparse datasets.

4.1 Block Coordinate Descent

Block coordinate descent is implemented to iteratively minimize the reconstruction error in matrix completion. Each step alternates between updating specific rows or columns of the matrix while keeping the others fixed. This approach simplifies the optimization process and ensures convergence to a local minimum, as it does not require operating on the entire matrix at one, unlike SVD decomposition. The updates rely on minimizing the loss function over the known entries of the matrix thus making block coordinate descent computationally efficient especially for large, sparse datasets.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	x_f1	x_f2	x_f3	x_f4	x_f5	x_f6	x_f7	x_f8	x_f9	x_f10	x_f11	x_f12	x_f13	x_f14	x_f15	x_f16	x_f17	x_f18	x_f19	x_f20	x_f21	x_f22	x_f23	x_f24	x_f25	x_f26	x_f27	x_f28	x_f29
2	362.26	363.73	365.27	366.83	368.42	370.04	371.68	373.32	374.96	376.6	378.24	379.88	381.51	383.15	384.8	386.44	388.1	389.75	391.41	393.07	394.72	396.38	398.04	399.68	401.33	402.96	404.61	406.26	407.9
3	162.75	161.58	160.36	159.13	157.86	156.59	155.31	154.03	152.74	151.45	150.16	148.86	147.57	146.27	144.98	143.69	142.4	141.1	139.8	138.5	137.19	135.89	134.59	133.28	131.99	130.69	129.39	128.1	126.81
4	182.08	183.39	184.75	186.13	187.53	188.95	190.37	191.79	193.21	194.64	196.07	197.51	198.94	200.38	201.81	203.25	204.68	206.11	207.55	208.99	210.43	211.87	213.31	214.73	216.16	217.59	219.03	220.47	221.91
5	267.34	265.76	264.14	262.48	260.8	259.1	257.4	255.69	253.98	252.27	250.55	248.82	247.1	245.38	243.66	241.94	240.22	238.5	236.78	235.06	233.34	231.62	229.9	228.18	226.46	224.74	223.02	221.31	219.59
6	309.4	307.84	306.24	304.59	302.91	301.24	299.55	297.86	296.16	294.46	292.75	291.04	289.33	287.62	285.91	284.2	282.49	280.78	279.07	277.35	275.62	273.9	272.18	270.46	268.74	267.02	265.3	263.58	261.86
7	241.89	242.73	243.6	244.5	245.41	246.33	247.26	248.19	249.11	250.04	250.97	251.89	252.81	253.75	254.68	255.61	256.55	257.48	258.41	259.34	260.26	261.19	262.12	263.05	263.98	264.91	265.84	266.77	267.71
8	151.54	152.73	153.95	155.21	156.47	157.74	159.03	160.33	161.64	162.95	164.26	165.56	166.86	168.18	169.49	170.81	172.13	173.45	174.76	176.08	177.38	178.7	180.01	181.32	182.64	183.96	185.28	186.59	187.92
9	239.05	237.88	236.68	235.46	234.22	232.97	231.71	230.45	229.19	227.94	226.68	225.42	224.16	222.89	221.63	220.36	219.1	217.83	216.56	215.29	214.02	212.75	211.49	210.22	208.95	207.68	206.41	205.14	203.88
10	317.48	316.16	314.78	313.38	311.96	310.52	309.08	307.64	306.19	304.73	303.29	301.85	300.4	298.95	297.51	296.06	294.62	293.18	291.73	290.28	288.83	287.39	285.94	284.49	283.04	281.6	280.14	278.69	277.24
11	201.31	202.18	203.09	204	204.92	205.86	206.79	207.73	208.67	209.61	210.55	211.48	212.41	213.33	214.26	215.17	216.09	217.02	217.96	218.89	219.82	220.76	221.69	222.62	223.57	224.51	225.46	226.4	227.34
12	181.5	182.39	183.33	184.31	185.29	186.28	187.29	188.3	189.29	190.29	191.28	192.27	193.26	194.25	195.24	196.23	197.24	198.24	199.24	200.24	201.25	202.24	203.24	204.24	205.24	206.24	207.25	208.25	209.24
13	12.56	13.87	15.23	16.62	18.03	19.45	20.88	22.32	23.75	25.19	26.62	28.04	29.46	30.89	32.33	33.77	35.2	36.65	38.09	39.53	40.96	42.41	43.85	45.27	46.7	48.13	49.56	50.99	52.43
14		43.3	41.97	40.56	39.09	37.57	36.08	34.57	33.03	31.5	29.94	28.38	26.82	25.26	23.7	22.13	20.55	19	17.42	15.86	14.29	12.76	11.21	9.65	8.08	6.51	4.92	3.33	1.76
15		277.64	279.16	280.73	282.34	283.96	285.61	287.26	288.91	290.57	292.24	293.9	295.57	297.24	298.91	300.58	302.26	303.93	305.6	307.27	308.94	310.61	312.27	313.93	315.61	317.29	318.96	320.64	322.32

Figure 2: Snippet of Original data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	id	x_f1	x_f2	x_f3	x_f4	x_f5	x_f6	x_f7	x_f8	x_f9	x_f10	x_f11	x_f12	x_f13	x_f14	x_f15	x_f16	x_f17	x_f18	x_f19	x_f20	x_f21	x_f22	x_f23	x_f24	x_f25	x_f26	x_f27	x_f28	x_
2	1	362.26	363.73	365.27	366.83	368.42	370.04	371.68	373.32	374.96	376.6	378.24	379.88	381.51	383.15	384.8	386.44	388.1	389.75	391.41	393.07	394.72	396.38	398.04	399.68	401.33	402.96	404.61	406.26	
3	2	162.75	161.58	160.36	159.13	157.86	156.59	155.31	154.03	152.74	151.45	150.16	148.86	147.57	146.27	144.98	143.69	142.4	141.1	139.8	138.5	137.19	135.89	134.59	133.28	131.99	130.69	129.39	128.1	
4	3	182.08	183.39	184.75	186.13	187.53	188.95	190.37	191.79	193.21	194.64	196.07	197.51	198.94	200.38	201.81	203.25	204.68	206.11	207.55	208.99	210.43	211.87	213.31	214.73	216.16	217.59	219.03	220.47	
5	4	267.34	265.76	264.14	262.48	260.8	259.1	257.4	255.69	253.98	252.27	250.55	248.82	247.1	245.38	243.66	241.94	240.22	238.5	236.78	235.06	233.34	231.62	229.9	228.18	226.46	224.74	223.02	221.31	
6	5	309.4	307.84	306.24	304.59	302.91	301.24	299.55	297.86	296.16	294.46	292.75	291.04	289.33	287.62	285.91	284.2	282.49	280.78	279.07	277.35	275.62	273.9	272.18	270.46	268.74	267.02	265.3	263.58	
7	6	241.89	242.73	243.6	244.5	245.41	246.33	247.26	248.19	249.11	250.04	250.97	251.89	252.81	253.75	254.68	255.61	256.55	257.48	258.41	259.34	260.26	261.19	262.12	263.05	263.98	264.91	265.84	266.77	
8	7	151.54	152.73	153.95	155.21	156.47	157.74	159.03	160.33	161.64	162.95	164.26	165.56	166.86	168.18	169.49	170.81	172.13	173.45	174.76	176.08	177.38	178.7	180.01	181.32	182.64	183.96	185.28	186.59	
9	8	239.05	237.88	236.68	235.46	234.22	232.97	231.71	230.45	229.19	227.94	226.68	225.42	224.16	222.89	221.63	220.36	219.1	217.83	216.56	215.29	214.02	212.75	211.49	210.22	208.95	207.68	206.41	205.14	
10	9	317.48	316.16	314.78	313.38	311.96	310.52	309.08	307.64	306.19	304.73	303.29	301.85	300.4	298.95	297.51	296.06	294.62	293.18	291.73	290.28	288.83	287.39	285.94	284.49	283.04	281.6	280.14	278.69	
11	10	201.31	202.18	203.09	204	204.92	205.86	206.79	207.73	208.67	209.61	210.55	211.48	212.41	213.33	214.26	215.17	216.09	217.02	217.96	218.89	219.82	220.76	221.69	222.62	223.57	224.51	225.46	226.4	
12	11	181.5	182.39	183.33	184.31	185.29	186.28	187.29	188.3	189.29	190.29	191.28	192.27	193.26	194.25	195.24	196.23	197.24	198.24	199.24	200.24	201.25	202.24	203.24	204.24	205.24	206.24	207.25	208.25	
13	12	12.56	13.87	15.23	16.62	18.03	19.45	20.88	22.32	23.75	25.19	26.62	28.04	29.46	30.89	32.33	33.77	35.2	36.65	38.09	39.53	40.96	42.41	43.85	45.27	46.7	48.13	49.56	50.99	
14	13	0	43.3	41.97	40.56	39.09	37.57	36.08	34.57	33.03	31.5	29.94	28.38	26.82	25.26	23.7	22.13	20.55	19	17.42	15.86	14.29	12.76	11.21	9.65	8.08	6.51	4.92	3.33	
15	14	0	277.64	279.16	280.73	282.34	283.96	285.61	287.26	288.91	290.57	292.24	293.9	295.57	297.24	298.91	300.58	302.26	303.93	305.6	307.27	308.94	310.61	312.27	313.93	315.61	317.29	318.96	320.64	

Figure 3: Snippet of Reconstructed data

4.2 Error Calculation

The decoding accuracy is assessed using the mean squared error (MSE) metric:

$$\text{MSE} = \frac{1}{|\Omega'|} \sum_{(i,j)\in\Omega} (D_{ij} - CP^T{}_{ij})^2,$$

where Ω' represents the binary set of observed entries of the pre-processed data that is sorted by columns and not yet have Nan values filled with column means.
D_{ij} the true values, and $CP^T{}_{ij}$ the predicted values.

As the block descent iterations progress, MSE decreases, demonstrating improved reconstruction accuracy. Additionally, increasing the amount of observed data enhances decoding performance, as reflected in reduced error trends.

4.3 Error Trends

The relationship between error and iterations highlights the effectiveness of the block coordinate descent:

- **With more data:** Error decreases due to better initialization and richer constraints on the optimization problem. However, as the amount of data retained in the sample increases, the compression rate decreases.

Number of Cars in Sample Set	Compression Ratio
5.0	15.71
10.0	41.45
20.0	113.86
30.0	199.35
50.0	404.08
75.0	629.04
250.0	783.25
500.0	1950.72
1000.0	1915.93
10000.0	2061.85
348751.0	2061.85

Table 1: Compression Ratios for Different Sample Sizes

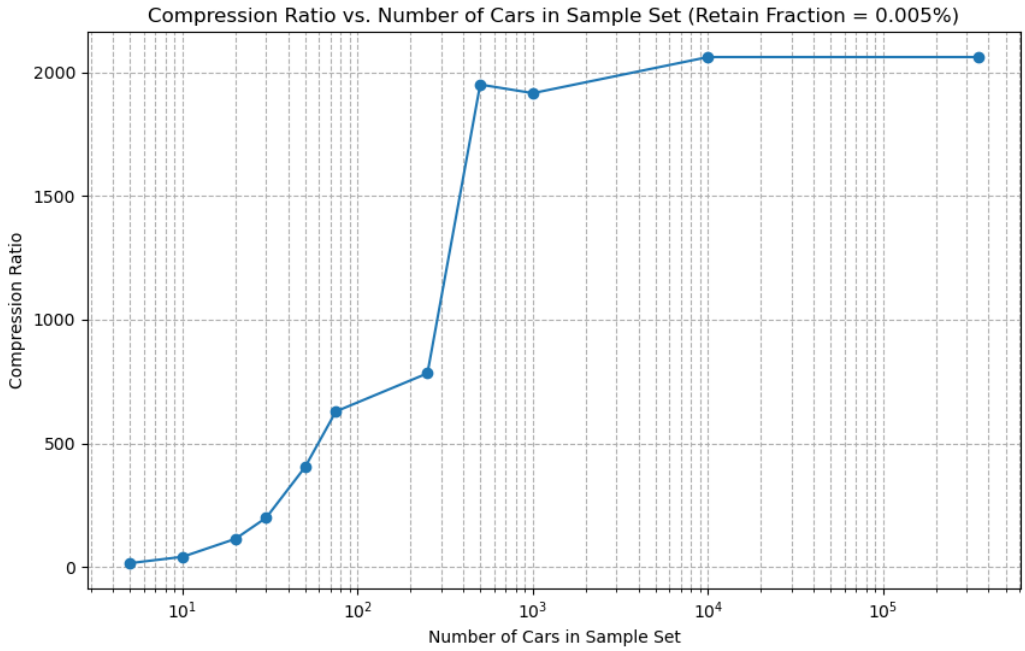


Figure 4: Compression Ratio vs. Number of Cars in the Sample Set

```
Compression Efficiency:
Original size: 2374045 bytes
Encoded size (Omega + R): 121135 bytes
Compression ratio (Original / Encoded): 19.60
```

Figure 5: Compression of 100 Cars with 1% retention

```
Compression Efficiency:
Original size: 2374045 bytes
Encoded size (Omega + R): 3907 bytes
Compression ratio (Original / Encoded): 607.64
```

Figure 6: Error of 100 Cars with 0.01% retention

- **Across iterations:** Early iterations see significant error reduction, while later iterations provide diminishing returns, converging to a stable solution. This is because the C and P matrices are each initialized as the identity matrix of dimensions NxN and MxM, where N and M are the dimensions of the original matrix (NxM). As the algorithm iterates block coordinate descent, the error rate shows a steep decline as it more correctly predicts C and P, which is a trend more prominent in larger datasets.

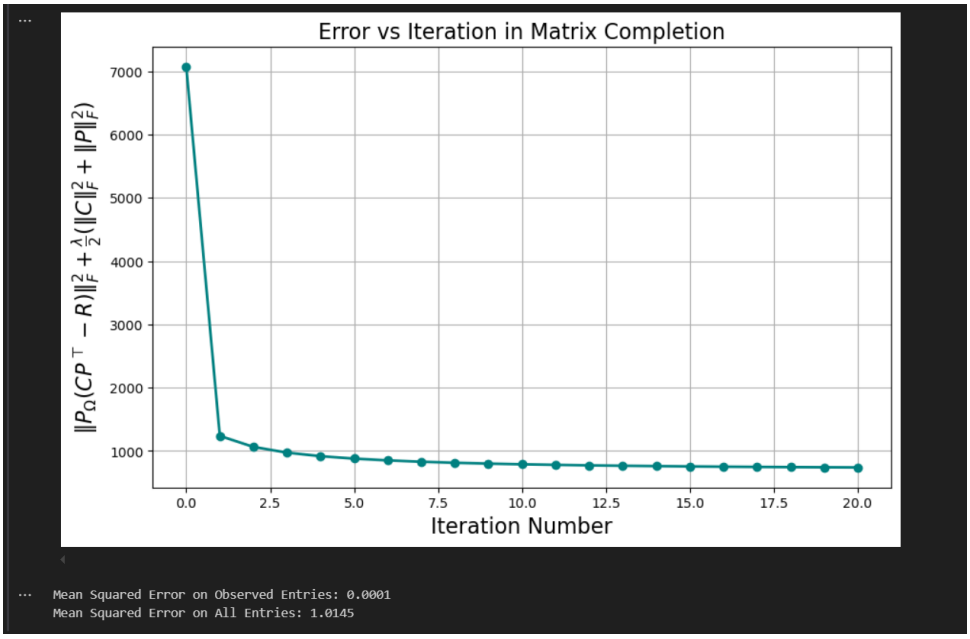


Figure 7: MSE Complete Dataset (348751 Cars)

The reduced matrix R retains the primary patterns in D, demonstrating the effectiveness of feature-based selective sampling for sparse traffic data. Future work could explore adaptive sampling strategies for datasets with higher dimensionality.

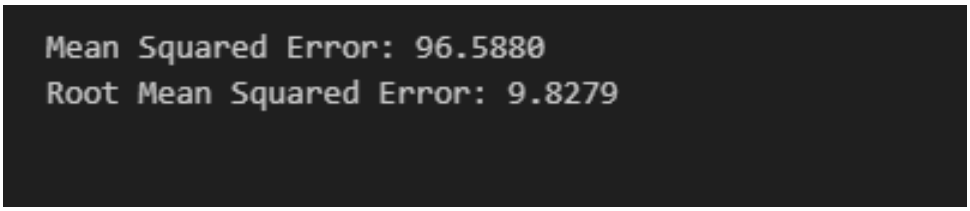


Figure 8: Error of 10000 Cars with 1% retention

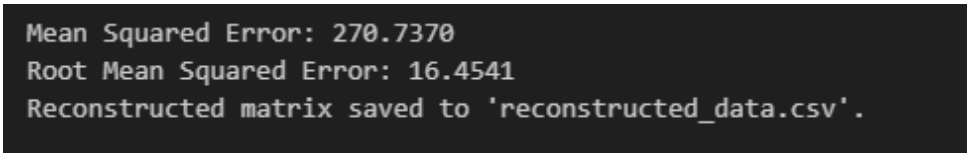


Figure 9: Error of 10000 Cars with 0.01% retention

As the retention rate increases, the error decreases. However, our current sampling method does not handle errors well when dealing with small datasets. As the dataset size grows, the approach performs better. This can be considered a limitation of our code. For future research, we would suggest exploring adaptive sampling strategies. These strategies would dynamically adjust feature retention probabilities based on intermediate reconstruction accuracy, rather than relying solely on feature-based selective sampling.

5 Conclusion

This project successfully applied feature-based selective sampling and rank-reduction techniques for the lossless compression of traffic datasets. The approach provides a foundation for handling large-scale data in real-time applications while preserving accuracy.

References

1. Candès, E., & Recht, B. (2012). Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6), 111-119.

2. Liu, H., Motoda, H., & Yu, L. (2004). A selective sampling approach to active feature selection. *Artificial Intelligence*, 159(1–2), 49-74. `https://doi.org/10.1016/j.artint.2004.05.009`

3. Saif Eddin Jabari, ENGR-UH 2027 Course Materials.