Comparative Analysis of Dimensionality Reduction Algorithms

Data Science Project

Bhumrapee Soonjun

MUIC

July 10, 2023

Contents

Introduction

Dimensionality Reduction is a technique used to transform the input data from some high dimensional space into a lower one. In today's presentation, we are concerned with algorithms that take in numerical data representable as a vector of real numbers. That is, the dimensionality reduction algorithms of interest is a map F such that,

$$F : \mathbb{R}^n \to \mathbb{R}^d$$

where $n > d$.

Moreover, we will be concerned with only the linear map which is a subset of dimensionality reduction algorithms that linearly project the data.

We might do this to visualize or speed up the training process of our ML model.

Research Question

However, these algorithms often have long computation times. For example, PCA and SVD require spectral decomposition which is very cost-heavy $\sim O(n^3)$. Therefore, alternatives are continually researched and developed.

The motivation for this research is to compare the performance between the classical dimensionality reduction algorithms, specifically the PCA and its variants and the Johnson-Lidenstrauss transform and its variants. Both are linear algorithms.

The research aims to answer the following question: in which situation the algorithm performs the best in term of running time, training time, and errors?

Johnson-Lidenstrauss Lemma

Lemma (Johnson-Lidenstrauss Lemma)

For $0 < \epsilon < \frac{1}{2}$, *given any set of points* $X = \{x_1, x_2, ..., x_n\} \in \mathbb{R}^D$, *there exists a linear mapping* $A : \mathbb{R}^D \to \mathbb{R}^k$ *with* $k = O(\frac{1}{\epsilon^2} \log n)$ *s.t.* $\forall i, j \in \{1, ..., n\}$ *and* $i \neq j$

$$(1 - \epsilon)A(x_i) - A(x_j)_2^2 \leq x_i - x_j{}_2^2 \leq (1 + \epsilon)A(x_i) - A(x_j)_2^2$$

Distributional Johnson-Lidenstrauss Lemma

Lemma (Distributional Johnson-Lidenstrauss Lemma)

For $0 < \epsilon, \delta < \frac{1}{2}, k = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$, $x \in \mathbb{R}^D$ is a unit vector, then there exists a distribution over $R^{k \times D}$ from which the matrix A is drawn such that

$$\textbf{\textit{Pr}}[|A(x)_2^2 - x_2^2| \leq \epsilon x_2^2] \geq 1 - \delta$$

This lemma can then be used to prove the first lemma (the Johnson-Lindenstrauss lemma). Then it follows from the proof of this lemma that the construction of the linear map A can be random values sampled from the standard Gaussian curve. Moreover, if $\delta \sim \frac{4}{n^2} \rightarrow k = O(\frac{1}{\epsilon^2} \log n)$ and it follows from the lemma that there exists such a linear mapping w.p. $1 - \frac{4}{n^2} \geq \frac{1}{2}$; hence, the expected number of trials is 2.

Johnson-Lidenstrauss Dimensionality Reduction

From the two lemmas above, it is implied that there is a mapping that can be sampled randomly such that it will preserve the structure of the data. The most basic form and the first to exist, is to simply hit the data with random normal vectors. That is,

$$M = \begin{bmatrix} \mathcal{N}(0,1) & \mathcal{N}(0,1) & \cdots & \mathcal{N}(0,1) \\ \mathcal{N}(0,1) & \mathcal{N}(0,1) & \cdots & \mathcal{N}(0,1) \\ \vdots & \cdots & \ddots & \vdots \\ \mathcal{N}(0,1) & \mathcal{N}(0,1) & \cdots & \mathcal{N}(0,1) \end{bmatrix}_{n \times d}$$

$$F(X) = M(X)$$

We shall refer to this as the JL transform. There are other more complicated variants that we will be testing, but will not be discussed here. But the idea is similar. The later and more powerful one relies on hashing as well so the detail will be left out to preserve sanity.

JL Transform Variants

The variants of the JL transform that we will test are listed here

1. JL Transform (JLT)
2. Sparse JL Transform (SJLT)
3. Fast JL Transform (FJLT)
4. Extremely Sparse JL Transform (ESE-JLT)

Initial Testing Algorithms

In this research, we initially test the following algorithms.

1. PCA
2. Randomized PCA
3. SVD -> PCA
4. JL Transform
5. Sparse JL Transform
6. Fast JL Transform
7. Extremely Sparse JL Transform

Dropped Algorithms

However, after some testing, we've found out that some of the algorithms are incomparable to the others in terms of speed and even numerical stability, hence it is dropped. Also some are the same thing as other, hence it is dropped as well. The dropped algorithms are listed below.

1. PCA - Extremely Slow
2. SVD -> PCA - Repeated
3. Fast JL Transform - Extremely Slow

Modified List

Hence, the modified list of algorithms are

1. Randomized PCA
2. JL Transform
3. Sparse JL Transform
4. Extremely Sparse JL Transform

Testing Parameters

We will test the algorithms and compare the following data of interest

1. Dimensionality Reduction Time
2. Training Time after applying DR
3. Errors of prediction (accuracy)
4. Shape of transformed data

Independent Variables

1. Input Data (Both Randomly Generated and Real inputs)
2. Data size
3. Data sparsity
4. Type of Tasks (Regression, Classification)
5. Data Compactness
6. Parameters of DR algorithms

Dependent Variables

1. Running time
2. Training time
3. Accuracy
4. Transformed Shape

Controlled Variables

1. Testing Computer - Cloud Service with guaranteed VM

All Settings - KMeans

1. Range of coefficient [-100, 100], [-500, 500], [1000, 1000]
2. Sparisity [0, 0.33, 0.66, 0.99]
3. std [100, 500, 1000]

All Settings - Regression

1. Range of coefficient [-100, 100], [-500, 500], [1000, 1000]
2. Sparisity [0, 0.33, 0.66, 0.99]
3. std [100, 500, 1000]
4. Correlation [0, 0.1, 0.3, 0.5, 0.9]

All Settings - Real Data

We will also test the algorithm on a real dataset. The dataset that we will be using is the health news dataset for classification model. The dataset contains raw text data so it has to be converted into tfid features first.

Using sklearn, we can perform the transformation and get the numerical data from the raw text. Each column can be thought of the frequency of a word in that particular file.

We will use Kmeans and Decision Tree on this dataset.

Code

Each algorithm is implemented in Python using mostly numpy and sklearn. However, there are helper and wrapper classes that help easily scale the testing. Moreover, the whole pipeline is written to automate the whole testing process.

Procedure

The main loop of the testing pipeline goes as follows

1. A random data is generated according to the defined parameters.
2. For each of the algorithms do:
    1. Apply dimensionality reduction using the algorithm on the data
    2. Train ML models on transformed data
    3. Collect results
3. Write to DataFrame

Data Collection

After defining what algorithms we want to test along with their parameters, we submit the code to a cloud computing service to run the code and collect the results. I've let it run for weeks (literally) before the whole process is completed. Then outputs are then downloaded and stored locally on my computer for analysis.

Method of Analysis

The data generated from the experiments contains a lot of information ranging from the method name, parameters, and accuracy, all the way to associated data's characteristics such as sparsity, correlation, std, mean and so on.

However, the data point of interest as defined in earlier section are the following columns of each method: ["accuracy", "reduction_time", "train_time"].

We will try to classify which algorithm should we use using those three obtained data points, the rest are for providing explanation to why it happens that way.

Score Function

To cope with high dimensional data, we will define another function that will be used to give score to each algorithms. This is due to the fact that we have three variables of interest, hence it will be hard to work in a 3D space and classify which is best.

Therefore, we define the following function that takes in accuracy, reduction_time, train_time, accuracy_weigh, reduction_weight, train_weigh, such that

$$F = accuracy^{accuracy\_weigh} \cdot \left(\frac{reduction\_weight}{reduction\_time} + \frac{train\_weigh}{train\_time} \right)$$

That is, the function will output a scalar that we can work with. Moreover, if we can more about accuracy, we can increase the accuracy weight to penalize algorithms that output lower accuracy. This is because of the $0 \leq$ accuracy ≤ 1. This successive power yields a monotonically decreasing sequence. Similarly, for other variables.

Score Function

We will apply the score function tot every row to get the score of each test. Then, we will simply take max to see which algorithm yields the best score for each test settings.

KMeans Result - Example

filename		name	filename	original_shape	transformed_shape	params	reduction_time	accuracy	train_time	score_series
1000_0.33_(-100, 100)	0	extremely sparse JL transform	1000_0.33_(-100, 100)	(1000, 1000)	(1110, 2351)	{'ep': 0.05, 'de': 0.05}	0.016116	0.97	0.638783	8.715012e+00
	1	extremely sparse JL transform	1000_0.33_(-100, 100)	(1000, 1000)	(1110, 2255)	{'ep': 0.05, 'de': 0.1}	0.036851	0.43	0.621037	2.554084e-03
	2	extremely sparse JL transform	1000_0.33_(-100, 100)	(1000, 1000)	(1110, 1175)	{'ep': 0.1, 'de': 0.05}	0.022616	0.58	0.458997	5.091322e-02
	3	extremely sparse JL transform	1000_0.33_(-100, 100)	(1000, 1000)	(1110, 1127)	{'ep': 0.1, 'de': 0.1}	0.012347	0.63	0.319385	1.164009e-01
	4	extremely sparse JL transform	1000_0.33_(-100, 100)	(1000, 1000)	(1110, 235)	{'ep': 0.5, 'de': 0.05}	0.008589	0.16	0.259311	1.299423e-07

KMeans Result - Aggregated 1

filename	name
1000_0.33_(-100, 100)	extremely sparse JL transform
1000_0.33_(-1000, 1000)	extremely sparse JL transform
1000_0.33_(-500, 500)	extremely sparse JL transform
1000_0.66_(-100, 100)	extremely sparse JL transform
1000_0.66_(-1000, 1000)	extremely sparse JL transform
1000_0.66_(-500, 500)	extremely sparse JL transform
1000_0.99_(-100, 100)	extremely sparse JL transform
1000_0.99_(-1000, 1000)	JL transform
1000_0.99_(-500, 500)	extremely sparse JL transform
1000_0_(-100, 100)	PCA
1000_0_(-1000, 1000)	extremely sparse JL transform
1000_0_(-500, 500)	extremely sparse JL transform

KMeans Result - Aggregated 2

500_0.33_(-100, 100)	extremely sparse JL transform
500_0.33_(-1000, 1000)	extremely sparse JL transform
500_0.33_(-500, 500)	extremely sparse JL transform
500_0.66_(-100, 100)	extremely sparse JL transform
500_0.66_(-1000, 1000)	extremely sparse JL transform
500_0.66_(-500, 500)	extremely sparse JL transform
500_0.99_(-100, 100)	extremely sparse JL transform
500_0.99_(-1000, 1000)	sparse JL transform
500_0.99_(-500, 500)	sparse JL transform
500_0_(-100, 100)	extremely sparse JL transform
500_0_(-1000, 1000)	extremely sparse JL transform
500_0_(-500, 500)	extremely sparse JL transform

KMeans Result - Aggregated 3

100_0.33_(-100, 100)	extremely sparse JL transform
100_0.33_(-1000, 1000)	extremely sparse JL transform
100_0.33_(-500, 500)	extremely sparse JL transform
100_0.66_(-100, 100)	extremely sparse JL transform
100_0.66_(-1000, 1000)	extremely sparse JL transform
100_0.66_(-500, 500)	extremely sparse JL transform
100_0.99_(-100, 100)	sparse JL transform
100_0.99_(-1000, 1000)	extremely sparse JL transform
100_0.99_(-500, 500)	extremely sparse JL transform
100_0_(-100, 100)	extremely sparse JL transform
100_0_(-1000, 1000)	extremely sparse JL transform
100_0_(-500, 500)	extremely sparse JL transform

Regression Result - Aggregated

filename	name
1000_0	extremely sparse JL transform
1000_0.1	extremely sparse JL transform
1000_0.3	extremely sparse JL transform
1000_0.5	extremely sparse JL transform
1000_0.9	extremely sparse JL transform
100_0	extremely sparse JL transform
100_0.1	extremely sparse JL transform
100_0.3	extremely sparse JL transform
100_0.5	extremely sparse JL transform
100_0.9	extremely sparse JL transform
500_0	extremely sparse JL transform
500_0.1	JL transform
500_0.3	extremely sparse JL transform
500_0.5	extremely sparse JL transform
500_0.9	extremely sparse JL transform

Real Data - KMeans - Aggregated

filename	name
2	extremely sparse JL transform
3	extremely sparse JL transform
5	extremely sparse JL transform

Real Data - Decision tree - Aggregated

filename	name
2	extremely sparse JL transform
3	sparse JL transform
5	sparse JL transform

Conclusion

For most of the case, the ESE-JL performs the best in terms of accuracy, reduction time, and training time.

This experiment could be repeat for multiple times to generate a more accurate results.

Thank you!

1. Approximate nearest neighbors and the fast ... - princeton university. (n.d.). https://www.cs.princeton.edu/ chazelle/pubs/stoc06.pdf
2. Fandina, O. N., Høgsgaard, M. M., amp; Larsen, K. G. (2022, April 4). The fast johnson-lindenstrauss transform is even faster. arXiv.org. https://arxiv.org/abs/2204.01800
3. Freksen, C. B. (2021, February 28). An introduction to johnson-lindenstrauss transforms. arXiv.org. https://arxiv.org/abs/2103.00564
4. Kane, D. M., amp; Nelson, J. (2014, February 5). Sparser Johnson-Lindenstrauss transforms. arXiv.org. https://arxiv.org/abs/1012.1577
5. Yin, R., Liu, Y., Wang, W., amp; Meng, D. (2020). Extremely sparse Johnson-Lindenstrauss transform: From theory to algorithm. 2020 IEEE International Conference on Data Mining (ICDM). https://doi.org/10.1109/icdm50108.2020.00180