Issues in Information Systems
Volume 25, Issue 3 pp. 231-237, 2024

DOI: https://doi.org/10.48009/3_iis_2024_118

Dimensionality and data size reduction using singular value decomposition

Ben Kim, *Seattle University, bkim@seattleu.edu*

Abstract

This paper discusses how to use SVD (Singular Value Decomposition) to reduce the data size as a preprocessing method before applying machine learning algorithms. Data reduction can lead to more efficient, and possibly better-performing machine learning models, especially when datasets are large, noisy, or high-dimensional. Specifically, we demonstrate two methods: PCA (Primary Component Analysis) and the data compression technique using SVD. For each method, we explain how it works and show the execution time, memory usage, and data reduction ratio using the random forest classification algorithm. All demonstrations of these methods are implemented in Python and the Python code is provided.

Keywords: data reduction, machine learning, SVD, singular values, eigenvectors, PCA, data compression

Introduction

Applying machine learning (ML) algorithms requires a substantial amount of data. However, in the real world, datasets can be extensive but may contain redundant, random, or noisy data. Such data do not contribute to effective data analysis using ML algorithms. Therefore, it is crucial to eliminate these redundancies and noise before applying ML algorithms. Data reduction techniques such as PCA (Principal Component Analysis) or data compression can be effective methods for this purpose. Data reduction techniques can eliminate noise and irrelevant features, helping to mitigate overfitting and enhance the generalizability of machine learning models. Additionally, datasets with reduced dimensions simplify model interpretation and data visualization. Reduced datasets consume fewer computing resources, leading to more efficient computing overall.

Literature review

Bogaardt et al. (2019) discussed reducing large geo-datasets, which contain autocorrelations and do not necessarily require high resolution. They used rank-deficiency, coarsening, and matrix factorization approaches available in SVD for data reduction, providing a decision tree to guide technique selection. Bydder and Du (2006) used singular value decomposition (SVD) to denoise multiple-echo data sets obtained from gradient- or spin-echo sequences. It identifies and suppresses noise components in the signal variation across echo times using a "minimum variance" filter applied to the smallest singular values. Jaradat et al. (2021) provide an overview of the mathematics behind SVD in a simple way. They noted that SVD technique is used in image compression and in dimensionality reduction as the underlining technique of the PCA and data analysis.

Salem et al. (2019) explained how PCA works, showing the calculation of eigenvalues and eigenvectors, and demonstrated applying PCA to the Iris dataset using MATLAB. They also discussed the relationship between PCA and SVD.

Tanwar et al. (2018) performed dimensionality reduction on big data, comparing SVD and PCA in terms of accuracy and mean squared errors. They recommended SVD over PCA for numerical reasons but preferred PCA for dimensionality reduction when processing image data. Vozalis and Margaritis (2007) explore enhancing Collaborative Filtering (CF) algorithms by integrating Singular Value Decomposition (SVD) with demographic data. It reviews SVD's past applications in Recommender Systems and introduces a method that incorporates demographic information at different stages of the filtering process to improve prediction quality. Testing on User-based and Item-based CF methods shows promising results, addressing known issues in Recommender Systems and boosting prediction accuracy. Wang et al. (2017) explained the use of PCA, KPCA, and SVD for data reduction, demonstrating these methods with a small dataset.

Dataset

We generated the following dataset for experimentation using the random forest classification algorithm in Python:

```
from sklearn.datasets import make_classificatio
X, y = make_classification(n_samples = 10000, n_features = 20,n_informative=12,
        n_repeated = 5 ,n_redundant=3, n_classes =3,flip_y=.2,random_state=1234)
```

This dataset has 20 columns and 10,000 rows. There are 12 informative columns. The repeated columns contain duplicate features. The redundant columns contain random linear combinations of the informative features.

SVD (Singular Value Decomposition)

SVD is a factorization of a matrix into three matrices. Unlike eigendecomposition, SVD can be applied to any matrix, whether square or non-square (rectangle).

Given a matrix X, SVD factorizes it as the product of three matrices as in the following:

$X=U \Sigma V^{-1}$

where V^{-1} is the inverse of V. U and V are orthogonal matrices.
Since V is an orthogonal matrix, $V^{-1} = V^T$ where V^T is the transpose of V.

Thus $\underline{X = U \Sigma V^T}$

$\sum$ is a diagonal matrix of positive singular values on the diagonal. These singular values are arranged in descending order. Each singular value represents the significance in each corresponding column or row of the original matrix.

Figure 1 shows the shapes of X, U, Σ, and V^T. X is an m by n matrix. The left singular vectors U is an *m by m* matrix and can be calculated by computing the eigenvectors of XX^T. U represents the directions of maximum variance in the column space of X. The right singular vectors V is an n by n matrix and can be calculated by computing the eigenvectors of X^TX. V represents the directions of maximum variance in the

row space of X. V^T is the transpose of V. Σ is a diagonal matrix where non-negative singular values on the diagonal are listed in descending order. Each singular value represents the variance or significance of the corresponding eigenvector. The remaining entries below the diagonal matrix are filled with zeros.

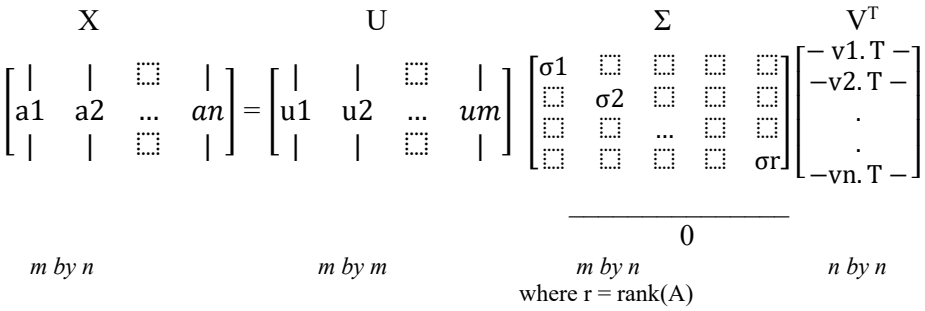

Figure 1: Shapes of X, Σ, U, and V^T:

We can calculate U, Σ, and V^T (transpose of V) using the *numpy* library available on Python as in the following:

```
from numpy.linalg import svd
U, S, VT = svd(X)
```

In Python, we can do the following to fill zeros in Σ:

```
import numpy as np
S_diag = np.diag(S)
# pad zeros in S.
Sigma_full = np.zeros((X.shape[0],X.shape[1]));Sigma_full
Sigma_full[:len(S),:len(S)] = np.diag(S);Sigma_full
```

We can reproduce X by multiplying U, Sigma_full, and VT.

```
U@Sigma_full@VT
```

PCA using SVD

As previously mentioned, singular values indicate the significance (or variance) of their corresponding eigenvectors. By identifying an inflection point within a sorted list of singular values in descending order, we can discern the more valuable or informative columns. The elbow method is a commonly used technique for this objective.

For example, the scree plot in Figure 2 can be generated from the list of singular values obtained earlier. We used the ratios of the singular values. The Python code for this is provided below. As expected from the dataset, the inflection point is shown at 12 components.

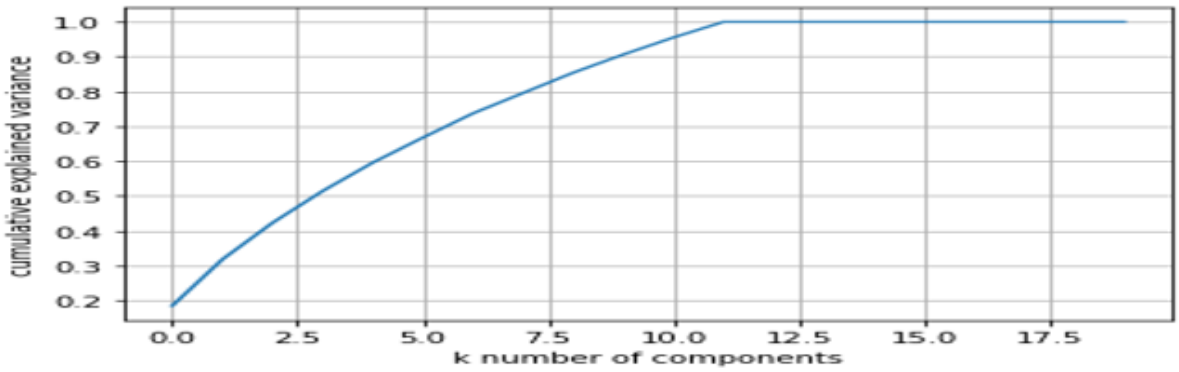

Figure 2: Cumulative Scree Plot

```
from numpy.linalg import svd
import matplotlib.pyplot as plt

U,S,Vt = svd(X)
S # Singular values.
singular_values_ratios = S/np.sum(S)
plt.plot(np.cumsum(singular_values_ratios))
plt.xlabel('k number of components')
plt.ylabel('cumulative explained variance')
plt.grid(True)
plt.show()
```

We can now use the 12 eigenvectors corresponding to the 12 highest singular values as our primary components. Subsequently, we project the original 12 columns of data onto these primary components to transform the original data.

In terms of linear algebra, this involves multiplying the original data by the primary components (eigenvectors) as demonstrated in the Python code below. We will then apply the random forest algorithm to this transformed dataset and discuss the performance measures in the next section.

```
from numpy.linalg import svd
k = 6 # k can be any number less than or equal to the number of columns of X
U,S,VT = svd(X)
V = VT.T # eigenvectors
X_svd = X @ V # Project X to V; X_svd has transformed data.
X_svd_pca = X_svd[:,:k] # We take the first k columns as shown above.
```

X_svd_pca is the transformed data by PCA. We will use it later for evaluations discussed in the next section.

Data compression using SVD

Although data compression is primarily used for image and signal compression, we can also use the data compression method for dimensionality reduction. This method is quite similar to the previously discussed

PCA. In PCA, after identifying the significant singular values, we project the original data onto the principal components (eigenvectors) to transform the data.

In the SVD-based data compression method, however, instead of projecting the original data onto the principal components, we truncate the original matrix by reconstructing the reduced versions of U, Σ, and V using only the components corresponding to the largest singular values. This process is demonstrated as follows:

Let's say k is the number of largest singular values chosen. *k* determines the level of approximation of the original data and consequently the compression ratio.

To truncate the original matrix, create the reduced versions of U, Σ, and V based on k as in the following:

$A_k = U_k \Sigma_k V_k^T$

where $U_k = U[:,:k]$, $\Sigma_k = \Sigma[:k]$, $V_k^T = V^T[:k,:]$

```
k = 3 # k can be any number less than or equal to the number of columns of X.
U_k = U[:,:k]
VT_k = VT[:k,:]
# S_k = np.diag(S[:k])
Sigma_full = np.zeros((X.shape[0],X.shape[1]))
Sigma_full[:len(S),:len(S)] = np.diag(S)
X_svd_comp = U_k@Sigma_full[:k,:k]@VT_k
```

X_svd_pca is the transformed data by data compression. We use it later for evaluations discussed in the next section.

Discussion

We used the random forest classification algorithm to evaluate the performances of the following three datasets:

- *X*:_original dataset
- *X_svd_pca*: dataset transformed by PCA
- *X_svd_comp*: dataset transformed by data compression

Also, we used three k values (6,12,18) as the number of singular values for PCA and data compression. In the original dataset, there are 12 informative columns.

The data reduction ratios are calculated as in the following:

> For X_svd_pca, the ratio is the matrix size with k number of columns divided by the size of the original data matrix. k is the number of singular values used.
>
> For X_svd_comp, the ratio is the sum of (m x k) + (k x k) + (k x n) divided by the size of the original data matrix where m and n are the numbers of columns and rows of X, respectively. k is the number of singular values used.

As can be seen in Table 3, there are no differences in data reduction ratios between PCA and the data compression method.

Table 3: Data Reduction Ratios

Method	k = 6	k = 12	k = 18
PCA (*X_svd_pca*)	.30	.60	.90
Data compression (*X_svd_comp*)	.30	.60	.90

As shown in Table 4, there are no discernible differences between the PCA and data compression methods in accuracy. Also, when we use only the 12 columns selected by both methods, they achieve the same accuracy as when using the original data with 20 columns.

Table 4: Accuracy

Method	k = 6	k = 12	k = 18
PCA (*X_svd_pca*)	.68	.79	.79
Data compression (*X_svd_comp*)	.69	.79	.79

Original Data (X): .79

Regarding execution time, PCA takes slightly longer than the data compression method (see Figure 5).

Table 5: Execution Time in Seconds

Method	k = 6	k = 12	k = 18
PCA (*X_svd_pca*)	7.71	7.67	7.75
Data compression (*X_svd_comp*)	7.23	7.21	7.25

Original Data(X): 8.65 seconds

However, both methods are faster than processing the original dataset. In terms of main memory usage, PCA also consumes slightly more space than the data compression method (see Table 6).

Table 6: Memory Usage in MB (megabytes)

Method	k = 6	k = 12	k = 18
PCA (*X_svd_pca*)	22.95	22.02	21.54
Data compression (*X_svd_comp*)	22.53	19.99	21.90

Original Data(X): 22.21 MB

Conclusion

In this paper, we explained PCA and the data compression method for data reduction. We demonstrated their performances in terms of execution time, memory usage, and data reduction ratios as well as the accuracy of predictions using the random forest classification algorithm. We found that both methods are effective and comparable to each other. In the next phase of this study, we plan to experiment with other machine learning models, such as neural networks, regression algorithms, and more.

References

Bogaardt, L., Goncalves, R., Zurita-Milla, R., & Izquierdo-Verdiguier, E. (2019). Dataset reduction techniques to speed up svd analyses on big geo-datasets. *ISPRS international journal of geo-information*, *8*(2), 55.

Bydder, M., & Du, J. (2006). Noise reduction in multiple-echo data sets using singular value decomposition. *Magnetic resonance imaging*, 24(7), 849-856.

Jaradat, Y., Masoud, M., Jannoud, I., Manasrah, A., & Alia, M. (2021, July). A tutorial on singular value decomposition with applications on image compression and dimensionality reduction. In 2021 *international conference on information technology* (ICIT) (pp. 769-772). IEEE.

Salem, N., & Hussein, S. (2019). Data dimensional reduction and principal components analysis. *Procedia Computer Science*, *163*, 292-299.

Tanwar, S., Ramani, T., & Tyagi, S. (2018). Dimensionality reduction using PCA and SVD in big data: A comparative case study. In *Future Internet Technologies and Trends: First International Conference, ICFITT 2017, Surat, India, August 31-September 2, 2017, Proceedings 1* (pp. 116-125). Springer International Publishing.

Vozalis, M. G., & Margaritis, K. G. (2007). Using SVD and demographic data for the enhancement of generalized collaborative filtering. *Information Sciences*, 177(15), 3017-3037.

Wang, Y., & Zhu, L. (2017, May). Research and implementation of SVD in machine learning. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)* (pp. 471-475). IEEE.