Matrix decomposition, also known as matrix factorization, involves describing a given matrix using its constituent elements.

Perhaps the most known and widely used matrix decomposition method is the Singular-Value Decomposition, or SVD. All matrices have an SVD, which makes it more stable than other methods, such as the eigendecomposition. As such, it is often used in a wide array of applications including compressing, denoising, and data reduction.

In this tutorial, you will discover the Singular-Value Decomposition method for decomposing a matrix into its constituent elements.

After completing this tutorial, you will know:

* What Singular-value decomposition is and what is involved.
* How to calculate an SVD and reconstruct a rectangular and square matrix from SVD elements.
* How to calculate the pseudoinverse and perform dimensionality reduction using the SVD..

**Tutorial Overview**

This tutorial is divided into 5 parts; they are:

1. Singular-Value Decomposition
2. Calculate Singular-Value Decomposition
3. Reconstruct Matrix from SVD
4. SVD for Pseudoinverse
5. SVD for Dimensionality Reduction

**Singular-Value Decomposition**

The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler.

For the case of simplicity we will focus on the SVD for real-valued matrices and ignore the case for complex numbers.



|  |  |
| --- | --- |
| 1 | A = U . Sigma . V^T |

Where A is the real m x n matrix that we wish to decompose, U is an m x m matrix, Sigma (often represented by the uppercase Greek letter Sigma) is an m x n diagonal matrix, and V^T is the  transpose of an n x n matrix where T is a superscript.

The Singular Value Decomposition is a highlight of linear algebra.

— Page 371, [Introduction to Linear Algebra](http://amzn.to/2AZ7R8j), Fifth Edition, 2016.

The diagonal values in the Sigma matrix are known as the singular values of the original matrix A. The columns of the U matrix are called the left-singular vectors of A, and the columns of V are called the right-singular vectors of A.

The SVD is calculated via iterative numerical methods. We will not go into the details of these methods. Every rectangular matrix has a singular value decomposition, although the resulting matrices may contain complex numbers and the limitations of floating point arithmetic may cause some matrices to fail to decompose neatly.

The singular value decomposition (SVD) provides another way to factorize a matrix, into singular vectors and singular values. The SVD allows us to discover some of the same kind of information as the eigendecomposition. However, the SVD is more generally applicable.

— Pages 44-45, [Deep Learning](http://amzn.to/2B3MsuU), 2016.

The SVD is used widely both in the calculation of other matrix operations, such as matrix inverse, but also as a data reduction method in machine learning. SVD can also be used in least squares linear regression, image compression, and denoising data.

The singular value decomposition (SVD) has numerous applications in statistics, machine learning, and computer science. Applying the SVD to a matrix is like looking inside it with X-ray vision…

— Page 297, [No Bullshit Guide To Linear Algebra](http://amzn.to/2k76D4C), 2017

**Calculate Singular-Value Decomposition**

The SVD can be calculated by calling the svd() function.

The function takes a matrix and returns the U, Sigma and V^T elements. The Sigma diagonal matrix is returned as a vector of singular values. The V matrix is returned in a transposed form, e.g. V.T.

The example below defines a 3×2 matrix and calculates the Singular-value decomposition.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # Singular-value decomposition  from numpy import array  from scipy.linalg import svd  # define a matrix  A = array([[1, 2], [3, 4], [5, 6]])  print(A)  # SVD  U, s, VT = svd(A)  print(U)  print(s)  print(VT) |

Running the example first prints the defined 3×2 matrix, then the 3×3 U matrix, 2 element Sigma vector, and 2×2 V^T matrix elements calculated from the decomposition.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | [[1 2]  [3 4]  [5 6]]    [[-0.2298477   0.88346102  0.40824829]  [-0.52474482  0.24078249 -0.81649658]  [-0.81964194 -0.40189603  0.40824829]]    [ 9.52551809  0.51430058]    [[-0.61962948 -0.78489445]  [-0.78489445  0.61962948]] |

**Reconstruct Matrix from SVD**

The original matrix can be reconstructed from the U, Sigma, and V^T elements.

The U, s, and V elements returned from the svd() cannot be multiplied directly.

The s vector must be converted into a diagonal matrix using the diag() function. By default, this function will create a square matrix that is m x m, relative to our original matrix. This causes a problem as the size of the matrices do not fit the rules of matrix multiplication, where the number of columns in a matrix must match the number of rows in the subsequent matrix.

After creating the square Sigma diagonal matrix, the sizes of the matrices are relative to the original m x n matrix that we are decomposing, as follows:



|  |  |
| --- | --- |
| 1 | U (m x m) . Sigma (m x m) . V^T (n x n) |

Where, in fact, we require:



|  |  |
| --- | --- |
| 1 | U (m x m) . Sigma (m x n) . V^T (n x n) |

We can achieve this by creating a new Sigma matrix of all zero values that is m x n (e.g. more rows) and populate the first n x n part of the matrix with the square diagonal matrix calculated via diag().



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | # Reconstruct SVD  from numpy import array  from numpy import diag  from numpy import dot  from numpy import zeros  from scipy.linalg import svd  # define a matrix  A = array([[1, 2], [3, 4], [5, 6]])  print(A)  # Singular-value decomposition  U, s, VT = svd(A)  # create m x n Sigma matrix  Sigma = zeros((A.shape[0], A.shape[1]))  # populate Sigma with n x n diagonal matrix  Sigma[:A.shape[1], :A.shape[1]] = diag(s)  # reconstruct matrix  B = U.dot(Sigma.dot(VT))  print(B) |

Running the example first prints the original matrix, then the matrix reconstructed from the SVD elements.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [[1 2]  [3 4]  [5 6]]    [[ 1.  2.]  [ 3.  4.]  [ 5.  6.]] |

The above complication with the Sigma diagonal only exists with the case where m and n are not equal. The diagonal matrix can be used directly when reconstructing a square matrix, as follows.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # Reconstruct SVD  from numpy import array  from numpy import diag  from numpy import dot  from scipy.linalg import svd  # define a matrix  A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  print(A)  # Singular-value decomposition  U, s, VT = svd(A)  # create n x n Sigma matrix  Sigma = diag(s)  # reconstruct matrix  B = U.dot(Sigma.dot(VT))  print(B) |

Running the example prints the original 3×3 matrix and the version reconstructed directly from the SVD elements.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [[1 2 3]  [4 5 6]  [7 8 9]]    [[ 1.  2.  3.]  [ 4.  5.  6.]  [ 7.  8.  9.]] |

**SVD for Pseudoinverse**

The pseudoinverse is the generalization of the matrix inverse for square matrices to rectangular matrices where the number of rows and columns are not equal.

It is also called the the Moore-Penrose Inverse after two independent discoverers of the method or the Generalized Inverse.

Matrix inversion is not defined for matrices that are not square. […] When A has more columns than rows, then solving a linear equation using the pseudoinverse provides one of the many possible solutions.

— Page 46, [Deep Learning](http://amzn.to/2B3MsuU), 2016.

The pseudoinverse is denoted as A^+, where A is the matrix that is being inverted and + is a superscript.

The pseudoinverse is calculated using the singular value decomposition of A:



|  |  |
| --- | --- |
| 1 | A^+ = V . D^+ . U^T |

Or, without the dot notation:



|  |  |
| --- | --- |
| 1 | A^+ = VD^+U^T |

Where A^+ is the pseudoinverse, D^+ is the pseudoinverse of the diagonal matrix Sigma and U^T is the transpose of U.

We can get U and V from the SVD operation.



|  |  |
| --- | --- |
| 1 | A = U . Sigma . V^T |

The D^+ can be calculated by creating a diagonal matrix from Sigma, calculating the reciprocal of each non-zero element in Sigma, and taking the transpose if the original matrix was rectangular.



|  |  |
| --- | --- |
| 1  2  3 | s11,   0,   0  Sigma = (  0, s22,   0)             0,   0, s33 |



|  |  |
| --- | --- |
| 1  2  3 | 1/s11,     0,     0  D^+ = (    0, 1/s22,     0)             0,     0, 1/s33 |

The pseudoinverse provides one way of solving the linear regression equation, specifically when there are more rows than there are columns, which is often the case.

NumPy provides the function pinv() for calculating the pseudoinverse of a rectangular matrix.

The example below defines a 4×2 matrix and calculates the pseudoinverse.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # Pseudoinverse  from numpy import array  from numpy.linalg import pinv  # define matrix  A = array([  [0.1, 0.2],  [0.3, 0.4],  [0.5, 0.6],  [0.7, 0.8]])  print(A)  # calculate pseudoinverse  B = pinv(A)  print(B) |

Running the example first prints the defined matrix, and then the calculated pseudoinverse.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [[ 0.1  0.2]  [ 0.3  0.4]  [ 0.5  0.6]  [ 0.7  0.8]]    [[ -1.00000000e+01  -5.00000000e+00   9.04289323e-15   5.00000000e+00]  [  8.50000000e+00   4.50000000e+00   5.00000000e-01  -3.50000000e+00]] |

We can calculate the pseudoinverse manually via the SVD and compare the results to the pinv() function.

First we must calculate the SVD. Next we must calculate the reciprocal of each value in the s array. Then the s array can be transformed into a diagonal matrix with an added row of zeros to make it rectangular. Finally, we can calculate the pseudoinverse from the elements.

The specific implementation is:



|  |  |
| --- | --- |
| 1 | A^+ = V . D^+ . U^V |

The full example is listed below.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | # Pseudoinverse via SVD  from numpy import array  from numpy.linalg import svd  from numpy import zeros  from numpy import diag  # define matrix  A = array([  [0.1, 0.2],  [0.3, 0.4],  [0.5, 0.6],  [0.7, 0.8]])  print(A)  # calculate svd  U, s, VT = svd(A)  # reciprocals of s  d = 1.0 / s  # create m x n D matrix  D = zeros(A.shape)  # populate D with n x n diagonal matrix  D[:A.shape[1], :A.shape[1]] = diag(d)  # calculate pseudoinverse  B = VT.T.dot(D.T).dot(U.T)  print(B) |

Running the example first prints the defined rectangular matrix and the pseudoinverse that matches the above results from the pinv() function.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [[ 0.1  0.2]  [ 0.3  0.4]  [ 0.5  0.6]  [ 0.7  0.8]]    [[ -1.00000000e+01  -5.00000000e+00   9.04831765e-15   5.00000000e+00]  [  8.50000000e+00   4.50000000e+00   5.00000000e-01  -3.50000000e+00]] |

**SVD for Dimensionality Reduction**

A popular application of SVD is for dimensionality reduction.

Data with a large number of features, such as more features (columns) than observations (rows) may be reduced to a smaller subset of features that are most relevant to the prediction problem.

The result is a matrix with a lower rank that is said to approximate the original matrix.

To do this we can perform an SVD operation on the original data and select the top k largest singular values in Sigma. These columns can be selected from Sigma and the rows selected from V^T.

An approximate B of the original vector A can then be reconstructed.



|  |  |
| --- | --- |
| 1 | B = U . Sigmak . V^Tk |

In natural language processing, this approach can be used on matrices of word occurrences or word frequencies in documents and is called Latent Semantic Analysis or Latent Semantic Indexing.

In practice, we can retain and work with a descriptive subset of the data called T. This is a dense summary of the matrix or a projection.



|  |  |
| --- | --- |
| 1 | T = U . Sigmak |

Further, this transform can be calculated and applied to the original matrix A as well as other similar matrices.



|  |  |
| --- | --- |
| 1 | T = V^Tk . A |

The example below demonstrates data reduction with the SVD.

First a 3×10 matrix is defined, with more columns than rows. The SVD is calculated and only the first two features are selected. The elements are recombined to give an accurate reproduction of the original matrix. Finally the transform is calculated two different ways.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | from numpy import array  from numpy import diag  from numpy import zeros  from scipy.linalg import svd  # define a matrix  A = array([  [1,2,3,4,5,6,7,8,9,10],  [11,12,13,14,15,16,17,18,19,20],  [21,22,23,24,25,26,27,28,29,30]])  print(A)  # Singular-value decomposition  U, s, VT = svd(A)  # create m x n Sigma matrix  Sigma = zeros((A.shape[0], A.shape[1]))  # populate Sigma with n x n diagonal matrix  Sigma[:A.shape[0], :A.shape[0]] = diag(s)  # select  n\_elements = 2  Sigma = Sigma[:, :n\_elements]  VT = VT[:n\_elements, :]  # reconstruct  B = U.dot(Sigma.dot(VT))  print(B)  # transform  T = U.dot(Sigma)  print(T)  T = A.dot(VT.T)  print(T) |

Running the example first prints the defined matrix then the reconstructed approximation, followed by two equivalent transforms of the original matrix.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | [[ 1  2  3  4  5  6  7  8  9 10]  [11 12 13 14 15 16 17 18 19 20]  [21 22 23 24 25 26 27 28 29 30]]    [[  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]  [ 11.  12.  13.  14.  15.  16.  17.  18.  19.  20.]  [ 21.  22.  23.  24.  25.  26.  27.  28.  29.  30.]]    [[-18.52157747   6.47697214]  [-49.81310011   1.91182038]  [-81.10462276  -2.65333138]]    [[-18.52157747   6.47697214]  [-49.81310011   1.91182038]  [-81.10462276  -2.65333138]] |

The scikit-learn provides a TruncatedSVD class that implements this capability directly.

The TruncatedSVD class can be created in which you must specify the number of desirable features or components to select, e.g. 2. Once created, you can fit the transform (e.g. calculate V^Tk) by calling the fit() function, then apply it to the original matrix by calling the transform() function. The result is the transform of A called T above.

The example below demonstrates the TruncatedSVD class.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | from numpy import array  from sklearn.decomposition import TruncatedSVD  # define array  A = array([  [1,2,3,4,5,6,7,8,9,10],  [11,12,13,14,15,16,17,18,19,20],  [21,22,23,24,25,26,27,28,29,30]])  print(A)  # svd  svd = TruncatedSVD(n\_components=2)  svd.fit(A)  result = svd.transform(A)  print(result) |

Running the example first prints the defined matrix, followed by the transformed version of the matrix.

We can see that the values match those calculated manually above, except for the sign on some values. We can expect there to be some instability when it comes to the sign given the nature of the calculations involved and the differences in the underlying libraries and methods used. This instability of sign should not be a problem in practice as long as the transform is trained for reuse.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [[ 1  2  3  4  5  6  7  8  9 10]  [11 12 13 14 15 16 17 18 19 20]  [21 22 23 24 25 26 27 28 29 30]]    [[ 18.52157747   6.47697214]  [ 49.81310011   1.91182038]  [ 81.10462276  -2.65333138]] |

**Extensions**

This section lists some ideas for extending the tutorial that you may wish to explore.

* Experiment with the SVD method on your own data.
* Research and list 10 applications of SVD in machine learning.
* Apply SVD as a data reduction technique on a tabular dataset.

**Further Reading**

This section provides more resources on the topic if you are looking to go deeper.

**Books**

* Chapter 12, Singular-Value and Jordan Decompositions, [Linear Algebra and Matrix Analysis for Statistics](http://amzn.to/2A9ceNv), 2014.
* Chapter 4, The Singular Value Decomposition and Chapter 5, More on the SVD, [Numerical Linear Algebra](http://amzn.to/2kjEF4S), 1997.
* Section 2.4 The Singular Value Decomposition, [Matrix Computations](http://amzn.to/2B9xnLD), 2012.
* Chapter 7 The Singular Value Decomposition (SVD), [Introduction to Linear Algebra](http://amzn.to/2AZ7R8j), Fifth Edition, 2016.
* Section 2.8 Singular Value Decomposition, [Deep Learning](http://amzn.to/2B3MsuU), 2016.
* Section 7.D Polar Decomposition and Singular Value Decomposition, [Linear Algebra Done Right](http://amzn.to/2BGuEqI), Third Edition, 2015.
* Lecture 3 The Singular Value Decomposition, [Numerical Linear Algebra](http://amzn.to/2BI9kRH), 1997.
* Section 2.6 Singular Value Decomposition, [Numerical Recipes: The Art of Scientific Computing](http://amzn.to/2BezVEE), Third Edition, 2007.
* Section 2.9 The Moore-Penrose Pseudoinverse, [Deep Learning](http://amzn.to/2B3MsuU), 2016.

**API**

* [numpy.linalg.svd() API](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.svd.html)
* [numpy.matrix.H API](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.matrix.H.html)
* [numpy.diag() API](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.diag.html)
* [numpy.linalg.pinv() API](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.pinv.html).
* [sklearn.decomposition.TruncatedSVD API](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html)

**Articles**

* [Matrix decomposition on Wikipedia](https://en.wikipedia.org/wiki/Matrix_decomposition)
* [Singular-value decomposition on Wikipedia](https://en.wikipedia.org/wiki/Singular-value_decomposition)
* [Singular value on Wikipedia](https://en.wikipedia.org/wiki/Singular_value)
* [Moore-Penrose inverse on Wikipedia](https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse)
* [Latent semantic analysis on Wikipedia](https://en.wikipedia.org/wiki/Latent_semantic_analysis)

**Summary**

In this tutorial, you discovered the Singular-value decomposition method for decomposing a matrix into its constituent elements.

Specifically, you learned:

* What Singular-value decomposition is and what is involved.
* How to calculate an SVD and reconstruct a rectangular and square matrix from SVD elements.
* How to calculate the pseudoinverse and perform dimensionality reduction using the SVD.