Simple SVD algorithms

Naive ways to calculate SVD

Risto Hinno
Published in Towards Data Science · 6 min read · Jan 31, 2021

Source

Intro

Aim of this post is to show some simple and educational examples how to calculate singular value decomposition using simple methods. If you are interested in industry strength implementations, you might find this useful.

SVD

Singular value decomposition (SVD) is a matrix factorization method that generalizes the eigendecomposition of a square matrix (n x n) to any matrix (n x m) (source).

If you don't know what is eigendecomposition or eigenvectors/eigenvalues, you should google it or read this post. This post assumes that you are familiar with these concepts.

Aman Bhansali
Abinesh Sivakumar

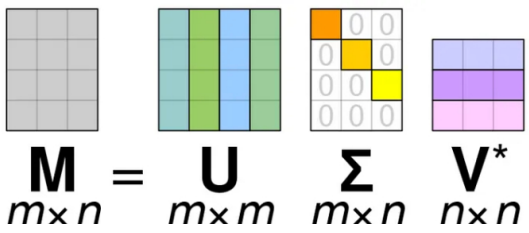
SVD is similar to Principal Component Analysis (PCA), but more general. PCA assumes that input square matrix, SVD doesn't have this assumption. General formula of SVD is:

Ipheiman, E Roodgar, and 1 other

$M=U\Sigma V^t$, where:

Monikamtcse

- **M**-is original matrix we want to decompose
- **U**-is left singular matrix (columns are left singular vectors). **U** columns contain eigenvectors of matrix MM^t
- **Σ**-is a diagonal matrix containing singular (eigen)values
- **V**-is right singular matrix (columns are right singular vectors). **V** columns contain eigenvectors of matrix M^tM



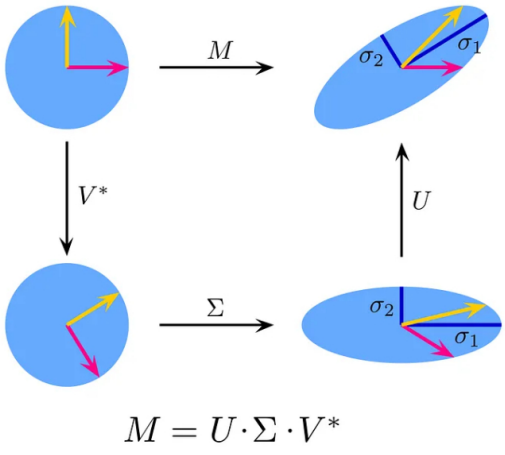
SVD matrices (source)

SVD is more general than PCA. From the previous picture we see that SVD can handle matrices with different number of columns and rows. SVD is similar to PCA. PCA formula is $\mathbf{M}=\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{t}$, which decomposes matrix into orthogonal matrix $\boldsymbol{Q}$ and diagonal matrix $\boldsymbol{\Lambda}$. Simply this could be interpreted as:

- change of the basis from standard basis to basis $\boldsymbol{Q}$ (using $\boldsymbol{Q}^{t}$)
- applying transformation matrix $\boldsymbol{\Lambda}$ which changes length not direction as this is diagonal matrix
- change of the basis from basis $\boldsymbol{Q}$ to standard basis (using $\boldsymbol{Q}$)

SVD does similar things, but it doesn't return to same basis from which we started transformations. It could not do it because our original matrix M isn't square matrix. Following picture shows change of basis and transformations related to SVD.

Screaching Fire and Jason Li



SVD transformations and change of basis. (source)

From the graph we see that SVD does following steps:

- change of the basis from standard basis to basis $\mathbf{V}$ (using $\mathbf{V}^{t}$). Note that in graph this is shown as simple rotation
- apply transformation described by matrix $\mathbf{\Sigma}$. This scales our vector in basis $\mathbf{V}$
- change of the basis from $\mathbf{V}$ to basis $\mathbf{U}$. Because our original matrix M isn't square, matrix $\mathbf{U}$ can't have same dimensions as V and we can't return to our original standard basis (see picture "SVD matrices")

Jason Li

There are numerous variants of SVD and ways to calculate SVD. I'll show just a few of the ways to calculate it.

Ways to calculate SVD

If you want to try coding examples yourself use this notebook which has all the examples used in this post.

Power iteration

Power iteration starts with $\mathbf{b}_0$ which might be a random vector. At every iteration this vector is updated using following rule:

* zzmez

$$b_{k+1} = \frac{Ab_k}{||Ab_k||}$$

Source

First we multiply $\mathbf{b}_0$ with original matrix A ($\mathbf{Ab_k}$) and divide result with the norm ($||\mathbf{Ab_k}||$). We'll continue until result has converged (updates are less than threshold).

Power method has few assumptions:

- $\mathbf{b}_0$ has a nonzero component in the direction of an eigenvector associated with the dominant eigenvalue. Initializing $\mathbf{b}_0$ randomly minimizes possibility that this assumption is not fulfilled

- matrix A has dominant eigenvalue which has strictly greater magnitude than other eigenvalues (source).

These assumptions guarantee that algorithm converges to a reasonable result. The smaller is difference between dominant eigenvalue and second eigenvalue, the longer it might take to converge.

Very simple example of power method could be found here. I've made example which also finds eigenvalue. As you can see core of this function is power iteration.

Duy Nguyễn Đỗ Quốc

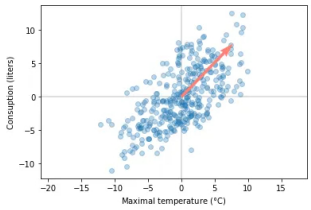
```python
def eigenvalue(A, v):
    # uses property A @ eigenvector = eigenvalue @ eigenvector =>
    # eigenvalue = (A @ eigenvector) / eigenvector
    val = A @ v / v
    return val[0]

def svd_power_iteration(A):
    n, d = A.shape
    #create original normal vector v which is used to approximate dominant eigenvalue
    v = np.ones(d) / np.sqrt(d)
    ev = eigenvalue(A, v)
    while True:
        #these two lines are basically the core of power iteration
        Av = A @ v
        v_new = Av / np.linalg.norm(Av)
        ev_new = eigenvalue(A, v_new)
        #break if new eigenvalue is not much different from previous
        if np.abs(ev - ev_new) < 0.01:
            break
        v = v_new
        ev = ev_new
    return ev_new, v_new
```

power_iteration_simple.py hosted with ❤ by GitHub view raw

For a simple example we use beer dataset (which is available from here). We'll construct covariance matrix and try to determine dominant singular value of the dataset. Full example with data processing is available in the notebook.

```python
#construct data
df=pd.read_csv('data/beer_dataset.csv')
X = np.array([df['Temperatura Maxima (C)'],
              df['Consumo de cerveja (litros)']]).T
C = np.cov(X, rowvar=False)

eigen_value, eigen_vec = svd_power_iteration(C)
```

We can plot dominant eigenvector with original data.



Dominant principle component/eigenvector of beer data

As we can see from the plot, this method really found dominant singular value/eigenvector. It looks like it is working. You can use notebook to see that results are very close to results from svd implementation provided by `numpy` . Next we'll see how to get more than just first dominant singular values.

Power iteration for n singular values

To get more than just most dominant singular value from matrix, we could still use power iteration. We'll implement new function which uses our previous `svd_power_iteration` function. Finding first dominant singular value is easy. We could use previously mentioned function. But how to find second singular value?

We should remove dominant direction from the matrix and repeat finding most dominant singular value (source). To do that we could subtract previous eigenvector(s) component(s) from the original matrix (using singular values and left and right singular vectors we have already calculated):

Jason Li and Sourish Dasgupta

$A\_next = A - (singular\_value_1)(u_1)(v_1)^t$

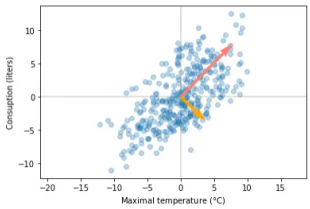
Here is example code (borrowed it from here, made minor modifications) for calculating multiple eigenvalues/eigenvectors.

```python
def svd(A, k=None, epsilon=1e-10):
    """returns k dominant eigenvalues and eigenvectors of matrix A"""
```

```
    A = np.array(A, dtype=float)
    n, m = A.shape

    svd_so_far = []
    if k is None:
        k = min(n, m)

    for i in range(k):
        matrix_for_1d = A.copy()
        #ubtract previous eigenvector(s) component(s)
        for singular_value, u, v in svd_so_far[:i]:
            matrix_for_1d -= singular_value * np.outer(u, v)
        #here also deal with different dimensions,
        #otherwise shapes don't match for matrix calculus
        if n > m:
            _, v = svd_dominant_eigen(matrix_for_1d, epsilon=epsilon)  # next singular vector
            u_unnormalized = A @ v
            sigma = np.linalg.norm(u_unnormalized)  # next singular value
            u = u_unnormalized / sigma
        else:
            _, u = svd_dominant_eigen(matrix_for_1d, epsilon=epsilon)  # next singular vector
            v_unnormalized = A.T @ u
            sigma = np.linalg.norm(v_unnormalized)  # next singular value
            v = v_unnormalized / sigma

        svd_so_far.append((sigma, u, v))

    singular_values, us, vs = [np.array(x) for x in zip(*svd_so_far)]
    return singular_values, us.T, vs
```

svd_n_values_power_iteration.py hosted with ♥ by GitHub view raw

When we apply to our beer dataset we get two eigenvalues and eigenvectors.



Beer data principle components/eigenvectors

Note that this example works also with matrices which have more columns than rows or more rows than columns. Results are comparable to `numpy` svd implementation. Here is one example:

```
mat = np.array([[1,2,3],
                [4,5,6]])
u, s, v = np.linalg.svd(mat, full_matrices=False)
values, left_s, rigth_s = svd(mat)

np.allclose(np.absolute(u), np.absolute(left_s))
#True
np.allclose(np.absolute(s), np.absolute(values))
#True
np.allclose(np.absolute(v), np.absolute(rigth_s))
#True
```

To compare our custom solution results with `numpy` svd implementation we take absolute values because signs in he matrices might be opposite. It means that vectors point opposite directions but are still on the same line and thus are still eigenvectors.

Now that we have found a way to calculate multiple singular values/singular vectors, we might ask could we do it more efficiently?

Block version of the Power Method

This version has also names like simultaneous power iteration or orthogonal iteration. Idea behind this version is pretty straightforward (source): Sourish Dasgupta

- other eigenvectors are orthogonal to the dominant one
- we can use the power method, and force that the second vector is orthogonal to the first one
- algorithm converges to two different eigenvectors
- do this for many vectors, not just two of them

Each step we multiply A not just by just one vector, but by multiple vectors which we put in a matrix Q. At each step we'll normalize the vectors using QR Decomposition. QR Decomposition decomposes matrix into following components:

A=QR, where

- A-original matrix we want to decompose
- Q-orthogonal matrix
- R-upper triangular matrix

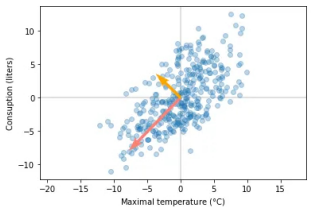
If algorithm converges then Q will be eigenvectors and R eigenvalues. Here is example code:

```python
def svd_simultaneous_power_iteration(A, k, epsilon=0.00001):
    #source http://mlwiki.org/index.php/Power_Iteration
    #adjusted to work with n<m and n>m matrices
    n_orig, m_orig = A.shape
    if k is None:
        k=min(n_orig,m_orig)
    A_orig=A.copy()
    if n_orig > m_orig:
        A = A.T @ A
        n, m = A.shape
    elif n_orig < m_orig:
        A = A @ A.T
        n, m = A.shape
    else:
        n,m=n_orig, m_orig

    Q = np.random.rand(n, k)
    Q, _ = np.linalg.qr(Q)
    Q_prev = Q
    #this part does the block power iteration
    for i in range(1000):
        Z = A @ Q
        Q, R = np.linalg.qr(Z)
        err = ((Q - Q_prev) ** 2).sum()
        Q_prev = Q
        if err < epsilon:
            break

    singular_values=np.sqrt(np.diag(R))
    #deal with different shape input matrices
    if n_orig < m_orig:
        left_vecs=Q.T
        #use property Values @ V = U.T@A => V=inv(Values)@U.T@A
        right_vecs=np.linalg.inv(np.diag(singular_values))@left_vecs.T@A_orig
    elif n_orig==m_orig:
        left_vecs=Q.T
        right_vecs=left_vecs
        singular_values=np.square(singular_values)
    else:
        right_vecs=Q.T
        #use property Values @ V = U.T@A => U=A@V@inv(Values)
        left_vecs=A_orig@ right_vecs.T @np.linalg.inv(np.diag(singular_values))

    return left_vecs, singular_values, right_vecs
```

svd_simultaneous_power_iteration.py hosted with ♥ by GitHub view raw

From the code we could see that calculating singular vectors and values is small part of the code. Much of the code is dedicated to dealing with different shaped matrices.

If we apply this function to beer dataset we should get similar results as we did with previous approach.



Beer data principle components/eigenvectors from svd_simultaneous_power_iteration

Eigenvectors point opposite directions compared to previous version, but they are on the same (with some small error) line and thus are the same eigenvectors. In the notebook I have examples which compares output with `numpy` svd implementation.

Conclusion

To calculate dominant singular value and singular vector we could start from * power iteration method. This method could be adjusted for calculating n-dominant singular values and vectors. For simultaneous singular value decomposition we could use block version of Power Iteration. These methods are not fastest and most stabile methods but are great sources for learning.

References

Beer Consumption — Sao Paulo, Kaggle

Eigenvalues and Eigenvectors, Risto Hinno

How to Compute the SVD

Power Iteration, ML Wiki

Power Iteration, Wikipedia

QR decomposition, Wikipedia

Singular value decomposition, Wikipedia

Singular Value Decomposition Part 2: Theorem, Proof, Algorithm, Jeremy Kun

Singular Values | Svd | Eigenvectors | Eigenvalue | Linear Algebra

118 1

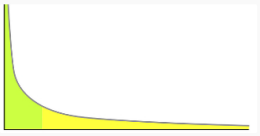
Written by Risto Hinno

Edit profile

126 Followers · Writer for Towards Data Science

Highly trained monkey

More from Risto Hinno and Towards Data Science

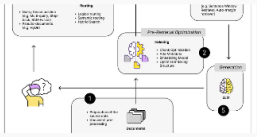


Risto Hinno

Intro to Power Law

Extremes are more frequent than we expect

Jan 10, 2023 59



Dominik Polzer in Towards Data Science

17 (Advanced) RAG Techniques to Turn Your LLM App Prototype int...

A collection of RAG techniques to help you develop your RAG app into something robus...

Jun 26 1.6K 16

Almog Baku in Towards Data Science

Building LLM Apps: A Clear Step-By-Step Guide

Comprehensive Steps for Building LLM-Native Apps: From Initial Idea to...

Jun 10 6K 16

Risto Hinno

PLDA

How to train and inference with Probabilistic Liner Discriminant Analysis

May 23, 2021 57 1

See all from Risto Hinno | See all from Towards Data Science

Recommended from Medium

Rukshan Pramoditha in Towards Data Science

Non-Negative Matrix Factorization (NMF) for Dimensionality...

Discussing theory and implementation with Python and Scikit-learn

May 6, 2023 404 2

Renda Zhang

Matrix Decomposition Series: 8 — The Principles and Methods of...

Matrix decomposition, a cornerstone in linear algebra and data science, plays a pivotal rol...

Jan 16

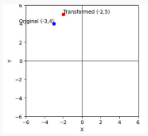
Lists

Staff Picks
684 stories · 1124 saves

Stories to Help You Level-Up at Work
19 stories · 685 saves

Self-Improvement 101
20 stories · 2273 saves

Productivity 101
20 stories · 2004 saves

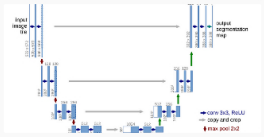


James Koh, PhD in MITB For All

Singular Value Decomposition — Explained step by step with code

What it is intuitively, with numbers to illustrate

Feb 6 · 81

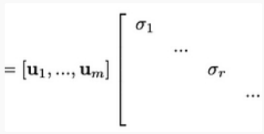


Alejandro Ito Aramendia

The U-Net : A Complete Guide

Everything you Need to Know

Feb 1 · 164

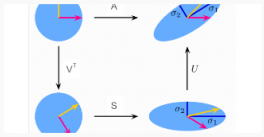


Izhangstat

Understanding Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a fundamental technique in linear algebra for...

Jun 5 · 5



Yuki Shizuya in Intuition

The mathematical and geographic understanding of Singular Value...

Introduction

Jan 18 · 122

See more recommendations