***SVD (Singular Value Decomposition) :
==============================

***https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/

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.

```
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, 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, 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, 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.

```
# 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.

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]]

[[-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]]
```

```
# 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)
```

==
==============

https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75

```
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import svds

A = csc_matrix([[1, 0, 0], [5, 0, 2], [0, -1, 0], [0, 0, 3]], dtype=float)
u, s, vt = svds(A, k=2) # k is the number of factors

print(s)
array([ 2.75193379,  5.6059665 ])
```

==
===============

*** Best one :
============

https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system#Collaborative-Filtering

```
from surprise import Reader, Dataset, SVD, evaluate
reader = Reader()
ratings = pd.read_csv('../input/the-movies-dataset/ratings_small.csv')
ratings.head()


data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
data.split(n_folds=5)


svd = SVD()
evaluate(svd, data, measures=['RMSE', 'MAE'])


trainset = data.build_full_trainset()
svd.fit(trainset)

ratings[ratings['userId'] == 1]

svd.predict(1, 302, 3)
```

==

https://medium.com/analytics-vidhya/master-dimensionality-reduction-with-these-5-must-know-applications-of-singular-value-777299940b89

Applications of Singular Value Decomposition (SVD)
==

We are going to follow a top-down approach here and discuss the applications first. I have explained the math behind SVD after the applications for those interested in how it works underneath.

You just need to know four things to understand the applications:

SVD is the decomposition of a matrix A into 3 matrices — U, S, and V
S is the diagonal matrix of singular values. Think of singular values as the importance values of different features in the matrix
The rank of a matrix is a measure of the unique information stored in a matrix. Higher the rank, more the information
Eigenvectors of a matrix are directions of maximum spread or variance of data

SVD for Image Compression
==========================

```
# get the image from "https://cdn.pixabay.com/photo/2017/03/27/16/50/
beach-2179624_960_720.jpg"
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2

# read image in grayscale
img = cv2.imread('beach-2179624_960_720.jpg', 0)

# obtain svd
U, S, V = np.linalg.svd(img)

# inspect shapes of the matrices
print(U.shape, S.shape, V.shape)

# plot images with different number of components
comps = [638, 500, 400, 300, 200, 100]

plt.figure(figsize = (16, 8))
for i in range(6):
  low_rank = U[:, :comps[i]] @ np.diag(S[:comps[i]]) @ V[:comps[i], :]
  if(i  == 0):
     plt.subplot(2, 3, i+1), plt.imshow(low_rank, cmap = 'gray'), plt.axis('off'),
plt.title("Original Image with n_components =" + str(comps[i]))
  else:
     plt.subplot(2, 3, i+1), plt.imshow(low_rank, cmap = 'gray'), plt.axis('off'),
```

```
plt.title("n_components =" + str(comps[i]))
```

SVD in NumPy
=============

NumPy is the fundamental package for Scientific Computing in Python. It has useful Linear Algebra capabilities along with other applications.

You can obtain the complete matrices U, S, and V using SVD in numpy.linalg. Note that S is a diagonal matrix which means that most of its entries are zeros. This is called a sparse matrix. To save space, S is returned as a 1D array of singular values instead of the complete 2D matrix.

==

```
import numpy as np
from numpy.linalg import svd

# define your matrix as a 2D numpy array
A = np.array([[4, 0], [3, -5]])

U, S, VT = svd(A)

print("Original Matrix:")
print(A)

print("Left Singular Vectors:")
print(U)

print("Singular Values:")
print(np.diag(S))

print("Right Singular Vectors:")
print(VT)

# check that this is an exact decomposition
# @ is used for matrix multiplication in Py3, use np.matmul with Py2

print("Matrix after multiplying U,S and VT")
print(U @ np.diag(S) @ VT)
```

Result :
=========
Original Matrix:
[[4 0]
 [3 -5]]

Left Singular Vectors:
[[-0.4472136 -0.89442719]
 [-0.89442719 0.4472136]]

Singular Values:
[[6.32455532 0.   ]
 [0.   3.16227766]]

Right Singular Vectors:
[[-0.70710678 0.70710678]
 [-0.70710678 -0.70710678]]

Matrix after multiplying U,S and VT
[[4.00000000e+00 -1.11271234e-15]
 [3.00000000e+00 -5.00000000e+00]]

==

Truncated SVD in scikit-learn
================================

```
import numpy as np
from sklearn.decomposition import TruncatedSVD

A = np.array([[-1, 2, 0], [2, 0, -2], [0, -2, 1]])
print("Original Matrix:")
print(A)

svd =  TruncatedSVD(n_components = 2)
A_transf = svd.fit_transform(A)

print("Singular values:")
print(svd.singular_values_)

print("Transformed Matrix after reducing to 2 features:")
```

```
print(A_transf)
```

```
Result:
========
Original Matrix:
[[-1  2  0]
 [ 2  0 -2]
 [ 0 -2  1]]
Singular values:
[3. 3.]
Transformed Matrix after reducing to 2 features:
[[ 0.70611358  2.12165115]
 [ 2.26353595 -1.69599675]
 [-1.83788155 -1.27365278]]
```

==
=====================

Randomized SVD in scikit-learn
==============================
Randomized SVD gives the same results as Truncated SVD and has a faster computation time. While Truncated SVD uses an exact solver ARPACK, Randomized SVD uses approximation techniques.

```
import numpy as np
from sklearn.utils.extmath import randomized_svd

A = np.array([[-1, 2, 0], [2, 0, -2], [0, -2, 1]])
u, s, vt = randomized_svd(A, n_components = 2)

print("Left Singular Vectors:")
print(u)

print("Singular Values:")
print(np.diag(s))

print("Right Singular Vectors:")
print(vt)
```

==

===================