Singular Value Decomposition

Taryn Heilman
Adam Richards

galvanize

Learning Objectives

galvanize

- Review PCA and other relevant concepts
- Know why we need SVD
- Compare and contrast PCA and SVD
- Apply numpy's SVD to reduce dimensionality and analyze components

Review relevant concepts

galvanize

- What is the curse of dimensionality?
- What are the steps of computing for performing PCA the eigendecomposition way?
- How would you choose the number of components to keep with PCA?
- Mathematically, what are the principal components?

Motivation

galvanize

Imagine we have 100 images and each image is 500*500 pixels.

For this problem the size of our **covariance matrix** is $6.25*10^{10}$. If each pixel is shown with a double precision number (64bites) then the total needed memory is $4*10^{12}$ bytes (4TB!)

This is very big for a small number of images! Is there any way to reduce the computational efforts without sacrificing the results?

Why SVD?

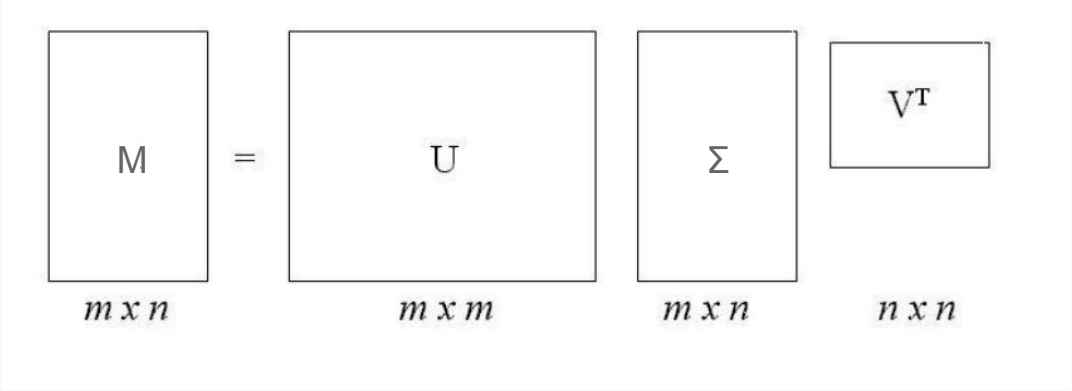
galvanize

Covariance matrix can become very large as the number of components in feature space increases.

SVD can help us to reduce the size of calculations (computational resources) by using matrix M (instead of covariance matrix M^TM) to find the same eigenvectors as PCA and with eigenvalues which are the square of the λ eigenvalues.

SVD

SVD decomposes matrix M_{m*n} into three matrices in which matrix Σ is diagonal.



$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & \sigma_{r+1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & \sigma_n \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}$$

U - examples as linear combinations of features (examples x “latent features”)
Σ - singular values of latent features
V - features x latent features

As matrix Σ is diagonal, some rows or columns will be empty, these are left off in code implementations

How do you do SVD?

galvanize

You don’t really need to know this to complete the lesson

High level walk through:
https://blog.statsbot.co/singular-value-decomposition-tutorial-52c695315254

Lower level, more in depth walkthrough:
http://www.cs.utexas.edu/users/inderjit/public_papers/HLA_SVD.pdf

Learning Objectives

galvanize

- ✓ Review PCA and other relevant concepts
- ✓ Know why we need SVD
- Compare and contrast PCA and SVD
- Apply numpy's SVD to reduce dimensionality and analyze components

Relationship between PCA and SVD

galvanize

In PCA we had

$$M^T M V = V\Lambda$$

where Λ is the diagonal matrix of eigenvalues
According to SVD we have

$$M = U\Sigma V^T$$

$$\begin{aligned} M^T M &= (U\Sigma V^T)^T U\Sigma V^T \\ &= V\Sigma^T U^T U\Sigma V^T \\ &= V\Sigma^2 V^T \end{aligned}$$

This is the same equation as with PCA we just have $\Lambda = \Sigma^2$

Implementations: Numpy

galvanize

Use numpy.linalg.svd, returns 3 np arrays

```
>>> a = np.random.randn(9, 6) + 1j*np.random.randn(9, 6)
>>> u, s, vh = np.linalg.svd(a, full_matrices=True)
>>> u.shape, s.shape, vh.shape
((9, 9), (6,), (6, 6))
>>> np.allclose(a, np.dot(u[:, :6] * s, vh))
True
>>> smat = np.zeros((9, 6), dtype=complex)
>>> smat[:6, :6] = np.diag(s)
>>> np.allclose(a, np.dot(u, np.dot(smat, vh)))
True
```

Reconstruction based on reduced SVD, 2D case:

```
>>> u, s, vh = np.linalg.svd(a, full_matrices=False)
>>> u.shape, s.shape, vh.shape
((9, 6), (6,), (6, 6))
>>> np.allclose(a, np.dot(u * s, vh))
True
>>> smat = np.diag(s)
>>> np.allclose(a, np.dot(u, np.dot(smat, vh)))
True
```

https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.svd.html

Implementations: Scikit-learn Truncated SVD

galvanize

Use sklearn.decomposition.**TruncatedSVD**

- Fits to a subset of the number of features
- **fit** method assigns singular values, explained variance/ratios, and components (shape (n_components, n_features)) attributes to the class
- **transform** method will transform an input matrix by reducing the number of features from n_features to n_components

```
>>> from sklearn.decomposition import TruncatedSVD
>>> from sklearn.random_projection import sparse_random_matrix
>>> X = sparse_random_matrix(100, 100, density=0.01, random_state=42)
>>> svd = TruncatedSVD(n_components=5, n_iter=7, random_state=42)
>>> svd.fit(X)
TruncatedSVD(algorithm='randomized', n_components=5, n_iter=7,
        random_state=42, tol=0.0)
>>> print(svd.explained_variance_ratio_)
[0.0606... 0.0584... 0.0497... 0.0434... 0.0372...]
>>> print(svd.explained_variance_ratio_.sum())
0.249...
>>> print(svd.singular_values_)
[2.5841... 2.5245... 2.3201... 2.1753... 2.0443...]
```

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

Implementations

galvanize

TruncatedSVD:

- object oriented nature makes it useful for pipelines that might use data reduction….
- But it's not true SVD, kind of a misnomer. (It's an approximation and it uses a different algorithm)
- tends to be used more in things like topic modeling/matrix factorization for recommenders.

np.linalg.svd:

- "true" svd
- What we will be using today

Example: Movie Ratings

galvanize

	Matrix	Alien	Serenity	Casablanca	Amelie
Alice	1	1	1	0	0
Bob	3	3	3	0	0
Cindy	4	4	4	0	0
Dan	5	5	5	0	0
Emily	0	2	0	4	4
Frank	0	0	0	5	5
Greg	0	1	0	2	2

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

M = np.array(
    [[1, 1, 1, 0, 0],
    [3, 3, 3, 0, 0],
    [4, 4, 4, 0, 0],
    [5, 5, 5, 0, 0],
    [0, 2, 0, 4, 4],
    [0, 0, 0, 5, 5],
    [0, 1, 0, 2, 2]]
)

u, e, v = svd(M)
print(M)
print("=")
print(np.around(u, 2))
print(np.around(e, 2))
print(np.around(v, 2))
```

Movie Ratings con't

	Matrix	Alien	Serenity	Casablanca	Amelie
Alice	1	1	1	0	0
Bob	3	3	3	0	0
Cindy	4	4	4	0	0
Dan	5	5	5	0	0
Emily	0	2	0	4	4
Frank	0	0	0	5	5
Greg	0	1	0	2	2

=

```
[[-0.14  0.02  0.01  0.99 -0.   -0.    0.  ]
 [-0.41  0.07  0.03 -0.06 -0.89  0.19  0.  ]
 [-0.55  0.09  0.04 -0.08  0.42  0.71  0.  ]
 [-0.69  0.12  0.05 -0.1   0.19 -0.68  0.  ]
 [-0.15 -0.59 -0.65 -0.    0.   -0.   -0.45]
 [-0.07 -0.73  0.68  0.   -0.    0.    0.  ]
 [-0.08 -0.3  -0.33 -0.   -0.   -0.    0.89]]

[[12.48  0.    0.    0.    0.  ]
 [ 0.    9.51  0.    0.    0.  ]
 [ 0.    0.    1.35  0.    0.  ]
 [ 0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.  ]]

[[-0.56 -0.59 -0.56 -0.09 -0.09]
 [ 0.13 -0.03  0.13 -0.7  -0.7 ]
 [ 0.41 -0.8   0.41  0.09  0.09]
 [-0.71  0.    0.71 -0.    0.  ]
 [-0.    0.   -0.    0.71 -0.71]]
```

Note that there are only 3 non-zero singular values! Thus we can ignore all but the first 3 columns of U and V (or the first three rows of)

Movie Ratings con't

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} & 0 & 0 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 0 & 0 \\ \mathbf{5} & \mathbf{5} & \mathbf{5} & 0 & 0 \\ 0 & \mathbf{2} & 0 & \mathbf{4} & \mathbf{4} \\ 0 & 0 & 0 & \mathbf{5} & \mathbf{5} \\ 0 & \mathbf{1} & 0 & \mathbf{2} & \mathbf{2} \end{bmatrix} = \begin{bmatrix} -0.14 & -0.02 & -0.01 \\ \mathbf{-0.41} & -0.07 & -0.03 \\ \mathbf{-0.55} & -0.09 & -0.04 \\ \mathbf{-0.69} & -0.12 & -0.05 \\ -0.15 & \mathbf{0.59} & \mathbf{0.65} \\ -0.07 & \mathbf{0.73} & \mathbf{-0.68} \\ -0.08 & \mathbf{0.3} & \mathbf{0.33} \end{bmatrix} \begin{bmatrix} \mathbf{12.48} & 0.0 & 0.0 \\ 0.0 & \mathbf{9.51} & 0.0 \\ 0.0 & 0.0 & \mathbf{1.35} \end{bmatrix} \begin{bmatrix} \mathbf{-0.56} & \mathbf{-0.59} & \mathbf{-0.56} & -0.09 & -0.09 \\ -0.13 & 0.03 & -0.13 & \mathbf{0.7} & \mathbf{0.7} \\ \mathbf{-0.41} & \mathbf{0.8} & \mathbf{-0.41} & -0.09 & -0.09 \end{bmatrix}$$

With , *U* is the user-to-topic matrix and *V* is the movie-to-topic matrix.

Science Fiction

- First singular value (12.48)
- First column of the U matrix (note: the first four users have large values)
- First row of the V matrix (note: the first three movies have large values)

Romance

- Second singular value (9.5)
- Second column of the U matrix (note: last three users have large values)
- Second row of the V matrix (note: the last two movies have large values)

Movie Ratings con't

The third singular value is relatively small, so we can exclude it with little loss of data. Let's try doing that and reconstruct our matrix

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} & 0 & 0 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 0 & 0 \\ \mathbf{5} & \mathbf{5} & \mathbf{5} & 0 & 0 \\ 0 & \mathbf{2} & 0 & \mathbf{4} & \mathbf{4} \\ 0 & 0 & 0 & \mathbf{5} & \mathbf{5} \\ 0 & \mathbf{1} & 0 & \mathbf{2} & \mathbf{2} \end{bmatrix} = \begin{bmatrix} -0.14 & -0.02 & -0.01 \\ \mathbf{-0.41} & -0.07 & -0.03 \\ \mathbf{-0.55} & -0.09 & -0.04 \\ \mathbf{-0.69} & -0.12 & -0.05 \\ -0.15 & \mathbf{0.59} & \mathbf{0.65} \\ -0.07 & \mathbf{0.73} & \mathbf{-0.68} \\ -0.08 & \mathbf{0.3} & \mathbf{0.33} \end{bmatrix} \begin{bmatrix} \mathbf{12.48} & 0.0 & 0.0 \\ 0.0 & \mathbf{9.51} & 0.0 \\ 0.0 & 0.0 & \mathbf{1.35} \end{bmatrix} \begin{bmatrix} \mathbf{-0.56} & \mathbf{-0.59} & \mathbf{-0.56} & -0.09 & -0.09 \\ -0.13 & 0.03 & -0.13 & \mathbf{0.7} & \mathbf{0.7} \\ \mathbf{-0.41} & \mathbf{0.8} & \mathbf{-0.41} & -0.09 & -0.09 \end{bmatrix}$$

```
# take only first two singular values and diagonalize, turn to 7x5
epsilon = np.diag(np.concatenate([e[:2], np.zeros(5)]))[:, :5]

reconstructed = u.dot(epsilon).dot(v)
print(np.round(reconstructed, 1))
```

```
[[ 1.   1.   1.  -0.  -0. ]
 [ 3.   3.   3.  -0.  -0. ]
 [ 4.   4.   4.  -0.  -0. ]
 [ 5.   5.1  5.  -0.  -0. ]
 [ 0.4  1.3  0.4  4.1  4.1]
 [-0.4  0.7 -0.4  4.9  4.9]
 [ 0.2  0.6  0.2  2.   2. ]]
```

Example: Image Compression

galvanize

Plot Image and convert it to a numpy array:

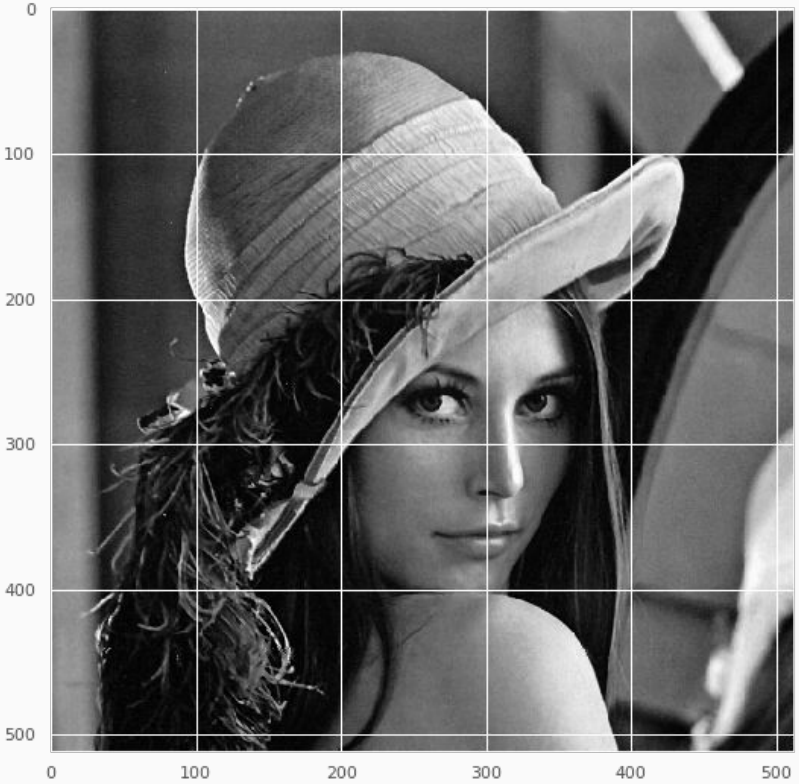
```
import matplotlib.pyplot as plt
import numpy as np
import time

from PIL import Image

img = Image.open('lena1.png')

print (img.size)
imggray = img.convert('LA')
plt.figure(figsize=(9, 9))
plt.imshow(imggray);

imgmat = np.array(list(imggray.getdata(band=0)), float)
imgmat.shape = (imggray.size[1], imggray.size[0])
imgmat = np.matrix(imgmat)
plt.figure(figsize=(9,9))
plt.imshow(imgmat, cmap='gray')
```



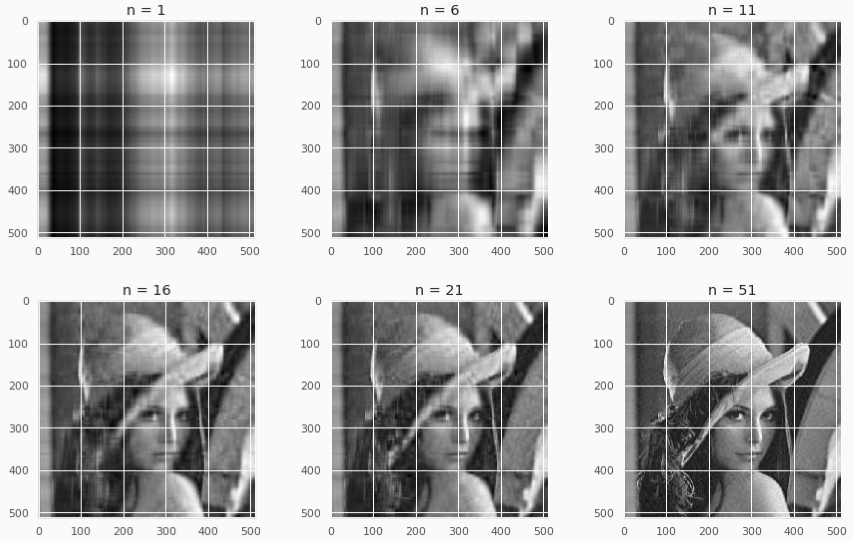
Example: Image Compression con't

galvanize

Apply SVD

```
U, sigma, V = np.linalg.svd(imgmat)

for i in range(1, 101, 5):
    reconstimg = np.matrix(U[:, :i]) * np.diag(sigma[:i]) * np.matrix(V[:i, :])
    plt.imshow(reconstimg, cmap='gray')
    title = "n = %s" % i
    plt.title(title)
    plt.show()
```



Check in Questions

galvanize

- Describe the outputs of the SVD
- What is the relationship between the eigenvalues of PCA and those of SVD?
- Which is more computationally efficient, PCA or SVD? Why?
- If you wanted to reduce dimensionality in your model, but wanted to maintain interpretability, should you use PCA or SVD? Why or why not?

Recap: Learning Objectives

galvanize

- ✓ Review PCA and other relevant concepts
- ✓ Know why we need SVD
- ✓ Compare and contrast PCA and SVD
- ✓ Apply numpy's SVD to reduce dimensionality and analyze components