PCA and SVD

Adam Richards

Galvanize, Inc

Last updated: May 8, 2019

Dimension Reduction

- Remove multicolinearity
- Deal with the curse of dimensionality
- Remove redundant features
- Interpretation & visualization
- Make computations of algorithms easier
- Identify structure for supervised learning

Standardization

Always start by standardizing the dataset

1. Center the data for each feature at the mean (so we have mean 0)
2. Divide by the standard deviation (so we have std 1)

sklearn.preprocessing

- http://scikit-learn.org/stable/modules/preprocessing.html
- The function `scale` provides a quick and easy way to perform this operation on a single array-like dataset
- the class `StandardScaler` provides further functionality as a `Transformer` (use `fit`)

```
from sklearn import preprocessing
X = preprocessing.scale(X)

scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)
```

Scaling with a train/test split

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

## create some data
X,y = make_classification(n_samples=50, n_features=5)

## make a train test split
X_train, X_test, y_train, y_test = train_test_split(X, y)

## scale using sklearn
scaler = StandardScaler().fit(X_train)
X_train_1 = scaler.transform(X_train)
X_test_1 = scaler.transform(X_test)

## scale without sklearn
X_train_2 = (X_train - X_train.mean(axis=0)) / X_train.std(axis=0)
X_test_2 = (X_test - X_train.mean(axis=0)) / X_train.std(axis=0)
```

https://stats.stackexchange.com/questions/77350/perform-feature-normalization-before-or-within-model-validation

$$\frac{1}{N}X^T X$$

```
import numpy as np
from sklearn import preprocessing

x1 =  np.array([[10,10,40,60,70,100,100]]).T
x2 =  np.array([[3,4,7,6,9,7,8]]).T
X = np.hstack([x1,x2]).astype(float)
n,d = X.shape
X = preprocessing.scale(X)

print(X.mean(axis=0),X.std(axis=0))
print(1.0/(n) * np.dot(X.T,X))
print(np.cov(X.T,bias=True))
```

```
[[ 1.  0.80138769]
 [ 0.80138769  1. ]]
```

1. This tells us that the covariance between feature 1 and feature 2 is 0.801.
2. This intuitively makes sense, since we can tell the two features are correlated
3. The variance of each feature is 1 and this makes sense because we standardized our data first
4. you can set `bias` to 1 if you want

an example

```
x1 =  np.array([[0,1,2,3]]).T
x2 =  np.array([[3,2,1,0]]).T
X = np.hstack([x1,x2]).astype(float)
X = preprocessing.scale(X)
print(np.cov(X.T))
```

What should the signs on the covariance matrix look like?

an example

```
x1 =  np.array([[0,1,2,3]]).T
x2 =  np.array([[3,2,1,0]]).T
X = np.hstack([x1,x2]).astype(float)
X = preprocessing.scale(X)
print(np.cov(X.T,bias=1))
```

What should the signs on the covariance matrix look like?

```
[[ 1. -1.]
 [-1. 1.]]
```

$C_{0,1}$ shows the correlation between x_0 and x_1, which is negative

Why do PCA in the first place?

High dimensional data causes many problems. Here are a few:

1. The Curse of Dimensionality
2. It's hard to visualize anything with more than 3 dimensions.
3. Points are "far away" in high dimensions, and it's easy to overfit small datasets.
4. Often (especially with image/video data) the most relevant features are not explicitly present in the high dimensional (raw) data.
5. Remove Correlation (e.g. neighboring pixels in an image)

PCA

- How is it that correlation and covariance were related again?

PCA

- How is it that correlation and covariance were related again?

$$Cov[X, Y] = E[(x - E[X])(y - E[Y])] \tag{1}$$

$$= \left[\sum or \int_{x,y \in S_X, S_Y}\right] (x - E[X])(y - E[Y])P(X = x, Y = y)\,[dxdy] \tag{2}$$

$$Corr[X, Y] = \frac{E[(x - E[X])(y - E[Y])]}{\sigma_X \sigma_Y} = \frac{Cov[X, Y]}{\sigma_X \sigma_Y} \tag{3}$$

- In say linear regression what might an ideal covariance matrix look like?

PCA

The ideal convariance matrix would look something like this

$$\begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 8 \end{bmatrix}$$

Principal Component Analysis

Usually we will get a covariance matrix with a lot of large values. Our ideal would be one where all the non-diagonal values are 0. This means that there is no relationship between the features. We can do a transformation of this data to make this happen

Behind the PCA

1. Create the design matrix
2. Standardize your matrix
3. Compute $\frac{1}{N} X^T X$
4. The principal components are the eigenvectors of the covariance matrix

The size of each eigenvector's eigenvalue denotes the amount of variance captured by that eigenvector

Ordering the eigenvectors by decreasing corresponding eigenvalues, you get an uncorrelated and orthogonal basis capturing the directions of most-to-least variance in your data

Singular Value Decomposition (SVD)

- So we can use a technique called SVD for more effiecient computation
- It is not always easy to directly compute eigenvalues and eigenvectors
- SVD is also useful for discovering hidden topics or latent features

Every matrix has a unique decomposition in the following form

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

where

- U is column orthogonal: $U^T U = I$
- V is column orthogonal: $V^T V = I$
- Σ is a diagonal matrix of positive values, where the diagonal is ordered in decreasing order

We can reduce the dimensions by sending the smaller of the diagonals to 0.

SVD and PCA

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 $\Lambda = \Sigma^2$

Movie ratings

	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))
```

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

With $M = U\Sigma V^T$, U is the user-to-topic matrix and V is the movie-to-topic matrix.

Science Fiction

- First singular value (12.4)
- 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)

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

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \approx \begin{bmatrix} -0.14 & -0.02 \\ -0.41 & -0.07 \\ -0.55 & -0.09 \\ -0.69 & -0.12 \\ -0.15 & 0.59 \\ -0.07 & 0.73 \\ -0.08 & 0.3 \end{bmatrix} \begin{bmatrix} 12.48 & 0.0 \\ 0.0 & 9.51 \end{bmatrix} \begin{bmatrix} -0.56 & -0.59 & -0.56 & -0.09 & -0.09 \\ -0.13 & 0.03 & -0.13 & 0.7 & 0.7 \end{bmatrix}$$

$$= \begin{bmatrix} 0.99 & 1.01 & 0.99 & -0.0 & -0.0 \\ 2.98 & 3.04 & 2.98 & -0.0 & -0.0 \\ 3.98 & 4.05 & 3.98 & -0.01 & -0.01 \\ 4.97 & 5.06 & 4.97 & -0.01 & -0.01 \\ 0.36 & 1.29 & 0.36 & 4.08 & 4.08 \\ -0.37 & 0.73 & -0.37 & 4.92 & 4.92 \\ 0.18 & 0.65 & 0.18 & 2.04 & 2.04 \end{bmatrix}$$

References I