

Dimension Reduction

Adam Richards

Galvanize, Inc

Last updated: 2. April 2018

- 1 Morning
- 2 PCA
- 3 PCA-related
- 4 SVD
- 5 Naive Bayes

Objectives

Morning

- Dimension reduction and feature selection
- Covariance matrix
- PCA
- t-SNE

Afternoon

- SVD

EXTRA DIMENSIONS



The big parallel here is how to connect weak nuclear forces, strong nuclear forces, electromagnetism and gravity? ... just add some dimensions

<http://www.phdcomics.com/comics.php?f=1522>

Why reduce the dimensions?

In our reality dimensions are **features** or **predictors**.

Why reduce the dimensions?

In our reality dimensions are **features** or **predictors**.

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

What tools do we already have?

- AIC/BIC in the context of GLM/GLMMs
- LASSO regression
 - 1 do LASSO regression
 - 2 throw away unused features
 - 3 do OLS regression
- Neural networks
 - 1 For labeled data: feature extraction via upper-layer outputs
 - 2 For labeled or unlabeled data: autoencoders
- Feature selection ([sklearn page](#))
 - 1 `from sklearn.feature_selection import VarianceThreshold`
 - 2 `from sklearn.feature_selection import SelectKBest`
 - For regression: `f_regression`, `mutual_info_regression`
 - For classification: `chi2`, `f_classif`, `mutual_info_classif`
 - [recursive feature elimination](#)

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?

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

$$= \left[\sum_{x,y \in S_X, S_Y} \text{or} \int \right] (x - E[X])(y - E[Y]) P(X = x, Y = y) [dx dy] \quad (2)$$

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

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

PCA

The ideal covariance 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

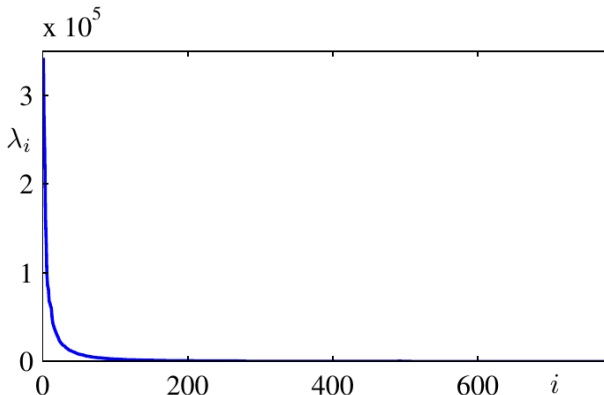
- 1 Create the design matrix
- 2 Standardize your matrix
- 3 Compute $\frac{1}{N}X^T X$
- 4 The principal components are the eigenvectors of the covariance matrix

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

MNIST

The MNIST dataset has digits stored as 28×28 pixel images. This means there are 784 features. We can use PCA to reduce the dimensions.



This is the scree plot. We often want to keep 80-90% of the variance

What is the curse of dimensionality?

Scree plot

displays the eigenvalues associated with a component or factor in descending order versus the number of the component or factor. You can use scree plots in [principal components analysis](#) and [factor analysis](#) to visually assess which components or factors explain most of the variability in the data.

Other related techniques:

- **Factor Analysis** - A technique related to PCA used to reduce the number of features and to detect structure in the relationships between features.
- **Structural Equation Modeling** - A variant of factor analysis is that enables hypothesis testing.
- **Canonical-correlation analysis** - Uses cross-variance matrices to produce a model equation which relates two sets of variables

The aim of principal component analysis is to explain the variance while factor analysis explains the covariance between the variables

t-SNE

t-distributed Stochastic Neighbor Embedding

It converts similarities between data points to joint probabilities and tries to minimize the **Kullback-Leibler divergence** between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.

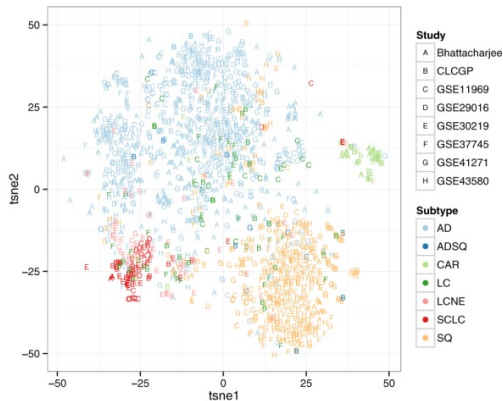
We may want to use another dimension reduction technique first...

- If the data are very dense → PCA to 50 dimensions
- If the data are very sparse → TruncatedSVD to 50 dimensions

[Laurens van der Maaten t-SNE site](#)

(van der Maaten and Hinton, 2008)

Lung cancer subtypes



(Hughey and Butte, 2015)

Discussion

Explain **if and how** PCA/tSNE might be used in the following scenerios.

- 1 Impressively large matrix of features, but we do not have the knowledge to interpret the features (metabolites and disease)
- 2 Impressively large matrix of features, but we really care only about which ones are most predictive of outcome?
- 3 Want to perform clustering on data set with say 100 dimensions?
- 4 Want to perform clustering on data set with say 4 dimensions?
- 5 Want to remove correlation among our features in a data set?
- 6 Want to investigate our target space or find some structure in your targets?
- 7 Want to identifier observation outliers?

Objectives

Morning

- ✓ Dimension reduction and feature selection
- ✓ Covariance matrix
- ✓ PCA
- ✓ t-SNE

Afternoon

- SVD

Singular Value Decomposition (SVD)

- So we can use a technique called SVD for more efficient 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))
```

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

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

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \approx \begin{bmatrix} -0.14 & -0.02 \\ -0.41 & -0.07 \\ -0.55 & -0.09 \\ -0.69 & -0.12 \\ -0.15 & 0.59 \\ -0.07 & 0.73 \\ -0.08 & 0.3 \end{bmatrix} \begin{bmatrix} 12.48 & 0.0 \\ 0.0 & 9.51 \end{bmatrix} \begin{bmatrix} -0.56 & -0.59 & -0.56 & -0.09 & -0.09 \\ -0.13 & 0.03 & -0.13 & 0.7 & 0.7 \end{bmatrix} \\
 = \begin{bmatrix} 0.99 & 1.01 & 0.99 & -0.0 & -0.0 \\ 2.98 & 3.04 & 2.98 & -0.0 & -0.0 \\ 3.98 & 4.05 & 3.98 & -0.01 & -0.01 \\ 4.97 & 5.06 & 4.97 & -0.01 & -0.01 \\ 0.36 & 1.29 & 0.36 & 4.08 & 4.08 \\ -0.37 & 0.73 & -0.37 & 4.92 & 4.92 \\ 0.18 & 0.65 & 0.18 & 2.04 & 2.04 \end{bmatrix}$$

Resources

- PCA tutorial
- Laurens van der Maaten t-SNE site

Bayes Theorem (again)

Bayes' Theorem allows us to switch around the events X and Y in a $P(X|Y)$ situation, provided we know certain other probabilities.

Derivation:

$$P(A, B) = P(B, A) \quad (4)$$

$$P(A|B) * P(B) = P(B|A) * P(A) \quad (5)$$

$$P(A|B) * P(B) = P(A, B) \quad (6)$$

$$P(A|B) = P(B|A) * P(A) / P(B) \quad (7)$$

$$(8)$$

In English:

$$\text{Posterior} = \frac{\text{Prior} * \text{Likelihood}}{\text{Evidence normalizing constant}} \quad (9)$$

Naive Bayes classifiers

Naive Bayes classifiers are considered naive because we assume that **all words in the string are assumed independent from one another**

While this clearly isn't true, they still perform remarkably well and historically were deployed as spam classifiers in the 90's. Naive Bayes handles cases where our number of features vastly outnumber our data points (i.e. we have more words than documents). These methods are also computationally efficient in that we just have to calculate sums.

Let's say we have some arbitrary document come in, (w_1, \dots, w_n) , and we would like to calculate the probability that it was from the sports section. In other words we would like to calculate...

$$P(y_c | w_1, \dots, w_n) = P(y_c) \prod_{i=1}^n P(w_i | y_c) \quad (10)$$

where...

$$P(y_c) = \frac{\sum y == y_c}{|D|} \quad (11)$$

$$P(w_i | y_c) = \frac{\text{count}(w_{D,i} | y_c) + 1}{\sum_{w \in V} [\text{count}(w | y_c) + 1]} \quad (12)$$

$$= \frac{\text{count}(w_{D,i} | y_c) + 1}{\sum_{w \in V} [\text{count}(w | y_c)] + |V|} \quad (13)$$

Laplace Smoothing

$$P(y_c | w_{d,1}, \dots, w_{d,n}) = P(y_c) \prod_{i=1}^n P(w_{d,i} | y_c) \quad (14)$$

$$\log(P(y_c | w_{d,1}, \dots, w_{d,n})) = \log(P(y_c)) + \sum_{i=1}^n \log(P(w_{d,i} | y_c)) \quad (15)$$

Why do we add 1 to the numerator and denominator?

This is called **Laplace Smoothing** and serves to remove the possibility of having a 0 in the denominator or the numerator, both of which would break our calculation.

Helpful video explaining Laplace transformation

<https://www.youtube.com/watch?v=evtCdmjcZ4I>

Summary

- Unknown words \rightarrow Laplace Smoothing
- Useful for Online learning
- Load of extensions and variants out there

Pros:

- Good with *wide data* ($p \gg n$)
- Good if n is small or n is quite big
- Fast to train
- Good at online learning, streaming data
- Simple to implement, not necessarily memory-bound (DB implementations)
- Multi-class

Cons:

- Naive assumption means correlated features are not actually treated right
- Sometimes outperformed by other models

References I

- Hughey, J. J. and Butte, A. J. (2015). Robust meta-analysis of gene expression using the elastic net. *Nucleic acids research*, 43(12):e79.
- van der Maaten, L. and Hinton, G. E. (2008). Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.