CSE/ECE-471: Statistical Methods in AI **Spring 2020**

Lecture 13: Principal Component Analysis

Prepared by: Animesh(20161302), Pratikkumar(2019201074), Utsav(2019202006)

1 The Problem of Feature Selection

Usually we have about 10s of 1000s of features that we have to work with, and large amount of data (about a few Gigabytes) that needs to be processed. And it's not like every feature is important - some are just redundant, like date, time, name and the like. ”Feature Selection” means selecting only a subset of these 10s of 1000s of features and considering them as the final input feature set.

For a d-dimensional dataset, it means taking a subset of the d features - the ones most important to the data - as the final input feature set.

How do you decide what's important? Let's see some rudimentary approaches here:

1. Look at covariance. If covariance is high between two features, then drop one of the features.
2. Start with zero features. Then add the features which add a lot to the overall accuracy of the system.
3. Converse: start with all features, then throw away the ones that lead to the least performance drop.

But these approaches work only on small feature sets - the ones where you can remember the features on your fingertips. And any pendejo not taking this course can come up with these approaches. Trust us when we say we can do better, which is why we need Principal Component Analysis.

2 A Quick Jog of Memory

Consider the dataset X,Y. Then,

$$X = \begin{bmatrix} X_1 & X_2 & X_3 & ... & X_N \end{bmatrix}$$

is the input data, where each of the N d-dimensional columns of X can be represented as

$$X_i = \begin{bmatrix} x_i^1 \\ x_i^2 \\ x_i^3 \\ .. \\ .. \\ .. \\ x_i^d \end{bmatrix}$$

Similarly, even the mean can be defined as

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ .. \\ .. \\ .. \\ \mu_d \end{bmatrix}$$

Define the matrix M as the mean column vector repeated N times.

$$M = \begin{bmatrix} \mu & \mu & \mu & .. & .. & \mu \end{bmatrix}$$

Then we can define X - M as:

$$\tilde{X} = X - M = \begin{bmatrix} X_1 - \mu & X_2 - \mu & X_3 - \mu & .. & .. & X_N - \mu \end{bmatrix}$$

Then, the covariance between all the features can be written as $C = E(\tilde{X}.\tilde{X}^T)$ where the covariance between the i^{th} and j^{th} feature is given by $C_{ij} = E((X_i - \mu)^T(X_j - \mu))$.

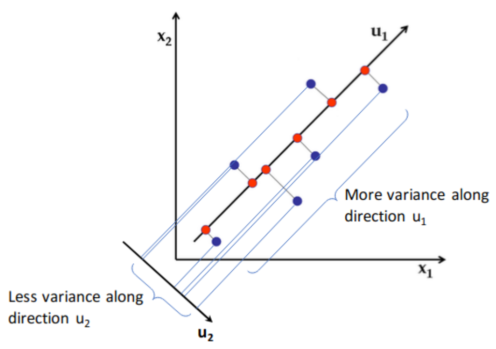
Pearson correlation coefficient for two features, therefore, is $C_{ij}/\sqrt{C_{ii}.C_{jj}}$.

3 Principal Component Analysis - PCA

Now that we have seen those informal (read:rudimentary) approaches, here's a more formal approach. The performance drop for removal of large number of features is very, very, very low. So low that you might have to take just 10 features out of a whopping 10,000. What's the catch? It works only on linear data.

3.1 The idea

Pick a direction vector v over which the data points are spread out the most. This is your first principal component. The next vector you get is perpendicular to v and is your second principal component.


x2
u1
More variance along direction u1
x1
Less variance along direction u2
u2

3.2 The Method

Consider a linear transformation of the dataset matrix X as $U_1^T.X$, where U_1 is the transformation vector. Then we are faced with the optimization problem: $\max_{U_1}(U_1^T.X)$ subject to $U_1^T.U_1 = 1$.

Now, assume that $E(X) = \mu$, then $E(U_1^T.X) = U_1^T.E(X) = U_1^T.\mu$.

Looking at variance: $\text{var}(U_1^T.X) = E(U_1^T.(X-\mu).(X-\mu)^T.U_1) = U_1^T S U_1$ where

$$S = E((X-\mu).(X-\mu)^T)$$

This means our optimization problem is: $\max_{U_1} U_1^T.S.U_1$ subject to $U_1^T.U_1 = 1$.

This calls for a Lagrangian transformation then. We maximise $U_1^T.S.U_1 - \lambda(U_1^T.U_1 - 1)$.

Differentiating w.r.t U_1 gives $SU_1 - \lambda.U_1 = 0$ or $S.U_1 = \lambda.U_1$.

What do we have here?!? $S.U_1 = \lambda.U_1$, which means U_1 has to be an eigenvector of S and λ has to be the eigenvalue. This then changes the optimization problem to $\max(\lambda)$.

What does this mean? We have a hint here. (No shit, Sherlock!) The maximisation changed to $\max(\lambda)$ - clearly it wants the top dogs. Now you gotta do what you gotta do!

The idea, therefore, is to take the eigenvectors corresponding to the top r out of the d eigenvalues you get. These eigenvectors are your "Principal Components" and will be orthogonal as well. These principal components together make the transformation matrix U_r which then transforms the dataset matrix X from d-dimensional subspace to a lower r-dimensional subspace.

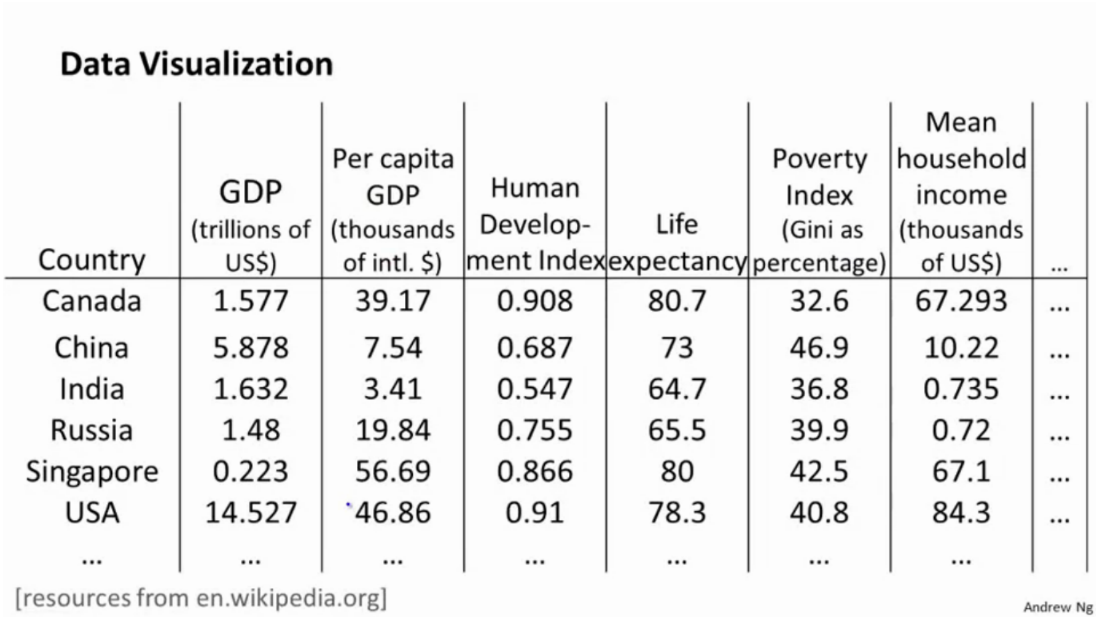
3.3 How to decide r?

A good estimate of r is that the chosen eigenvalues should account for 95% of the variance. This, in math, means $\frac{\sum_1^r \lambda_i}{\sum_1^d \lambda_i} \geq 0.95$

3.4 Applications

1. Data compression: Because of the property of dimensional reduction of the Principal Component Analysis, it can be used to compress the input dataset, making it easy for the system to be deployed immediately after training. Let's say you have 10,000 features in a dataset of 1 million datapoints. Then, applying PCA will help with transforming the dataset to a lower 10-dimensional subspace. This reduces the size of the input dataset, thus easing calculations.

2. Visualization: As humans, we can't visualize more than 3 dimensions. Big reason why PCA is needed. An example: Data for all the countries in the world - data for each country is a 50-dimensional vector.



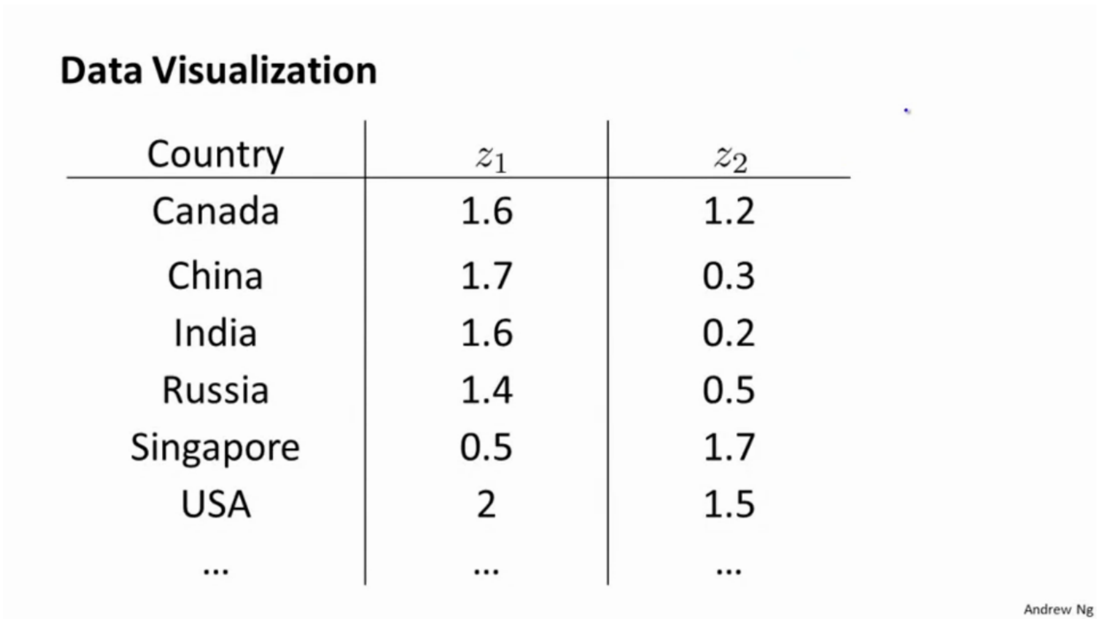
Data Visualization

Country	GDP (trillions of US$)	Per capita GDP (thousands of intl. $)	Human Develop-ment Index	Life expectancy	Poverty Index (Gini as percentage)	Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	

[resources from en.wikipedia.org]

Andrew Ng

Applying PCA reduces it to a 2-dimensional vector denoting its "Per Capita GDP" and "Overall GDP".

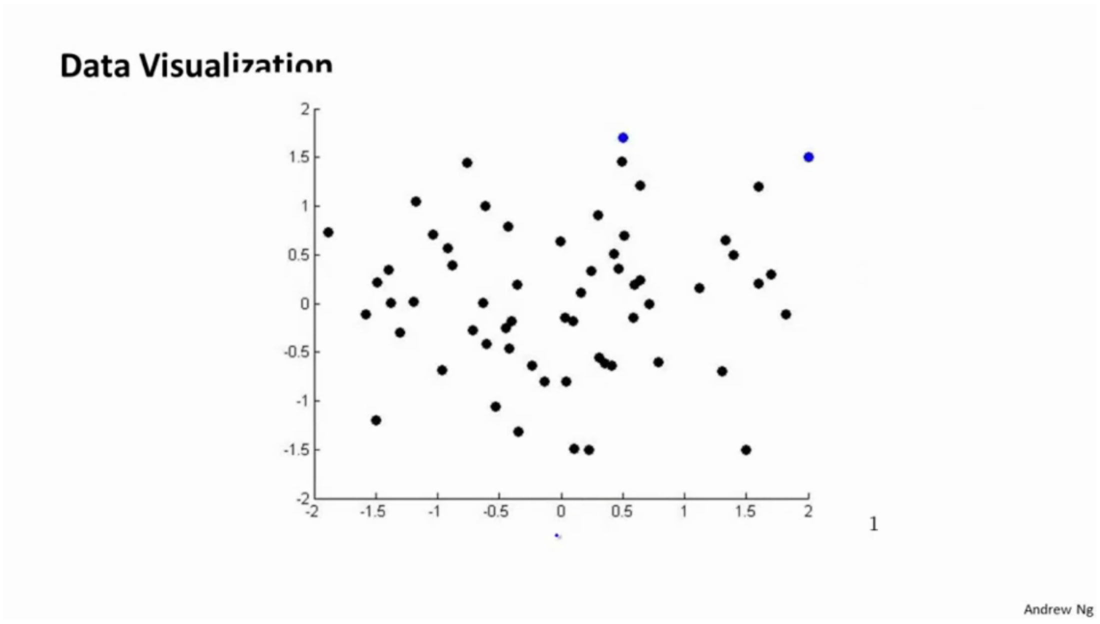


Data Visualization

Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

Andrew Ng

This reduction of dimensions then gives us the following graph.

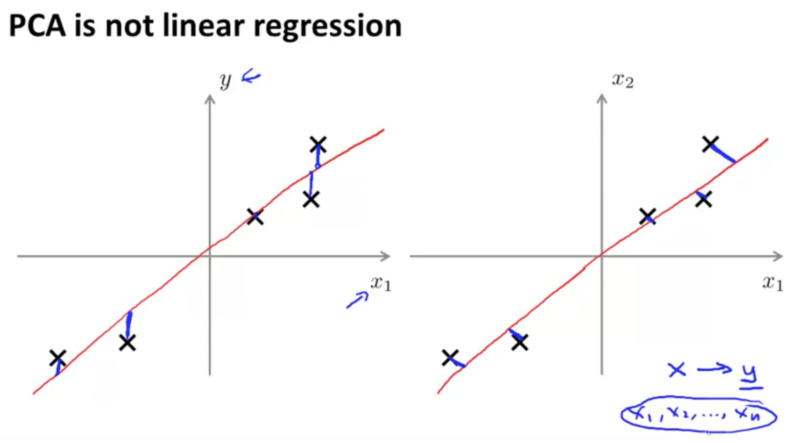


3.5 PCA is NOT Linear Regression!

Yes. PCA is not Linear Regression.

Linear Regression is a form of supervised learning and involves both X_i and y_i. It involves calculating the distance between the target and the ground truth in terms of their y-axis distance.

PCA, on the other hand, is just a method of preprocessing your INPUT, which does NOT involve the use of y_i.



3.6 Finding the eigenvectors - Singular Value Decomposition

The singular value decomposition is a factorisation of a real or complex matrix. The singular value decomposition of a matrix M_{mxn} is a factorisation of the form $M = UDV^T$ where U_{mxm} is a unitary matrix, i.e. $UU^* = U^*U = I_m$, D_{mxn} is a diagonal matrix with non-negative real numbers on the diagonal and V_{nxn} is a unitary matrix, i.e. $VV^* = V^*V = I_n$.

U^* is the conjugate transpose of the matrix U, irrespective of the order.

Note 1: The left singular vectors of M are the eigen-vectors of MM^T, while the right singular vectors of M are the eigen-vectors of M^TM.

Note 2: The non-zero singular values of M (found on the diagonal entries of D) are the square-roots of the non-zero eigen-values of both MM^T and M^TM.

References

[1] Andrew Ng's course on Machine Learning @ Coursera: https://www.coursera.org/lecture/machine-learning/motivation-ii-visualization-t6pYD

[2] The direction vector diagram, modified by Pratik, to show the minimum variance vector alongside the existing max variance vector. Original image (without the extra minimum-variance vector) found here: https://medium.com/@datalesdatales/visualising-football-players-in-two-dimensions-with-pca-92c7bb005ab4