Microsoft: DAT210x Programming with Python for Data Science

Help

Course › 4. Transforming Data › Lecture: PCA › Video

Video

Bookmark this page

PCA Gotchas!

Start of transcript. Skip to the end.

(Caption will be displayed when you start playing the video.)
0:00 / 1:55
1.25x

We've really already discussed the pros of PCA.

And they are essentially all of the real life use cases

that we just covered.

Now, you should also be aware that PCA has some weaknesses, so

that you know when to avoid it when necessary.

The first is that PCA is sensitive to feature scaling.

Video

Download video file

Transcripts

Download SubRip (.srt) file

Download Text (.txt) file

To use PCA effectively, you should be aware of its weaknesses. The first is that it is sensitive to the scaling of your features. PCA maximizes variability based off of variance (the average squared differences of your samples from the mean), and then projects your original data on these directions of maximal variance. If your has a feature with a large variance and others with small variances, PCA will load on the larger variance feature. Being a linear transformation, it will rotate and reorient your feature space so it diverts as much of the variance of the larger-variance feature (and some of the other features' variances), into the first few principal components.

When it does this, a feature might go from having little impact on your transformation to totally dominating the first principal component, or vice versa. Standardizing your variables ahead of time is the way to free your PCA results of such scaling issues. The cases where you *should not* use standardization are when you know your feature variables, and thus their importance, need to respect the specific scaling you've set up. In such a case, PCA would be used on your raw data.

PCA is extremely fast, but for ***very*** large datasets it might take a while to train. All of the matrix computations required for inversion, eigenvalue decomposition, etc. boggle it down. Since most real-world datasets are typically very large and will have a level of noise in them, razor-sharp, machine-precision matrix operations aren't always necessary. If you're willing to sacrifice a bit of accuracy for computational efficiency, SciKit-Learn offers a sister algorithm called *RandomizedPCA* that applies some approximation techniques to speed up large-scale matrix computation. You can use this for your larger datasets.

NOTE: The sklearn.decomposition.RandomizedPCA() method has since been deprecated. If you would like to perform a randomized singular variable decomposition as the means of estimating PCA, update the svd_solver parameter to equal the string 'randomized'. Whether or not the randomized approximation actually runs faster than regular PCA depends on a number of factors, including but not limited to: if there was any interference from other running sub-processes, the size of your dataset, the data types used in your dataset, and if the logic introduced to run the randomized selection can complete faster than just running through the dataset regularly. Due to this, for small to medium sized datasets, it might occasionally be faster to run regular PCA.

The last issue to keep in mind is that PCA is a linear transformation only! In graphical terms, it can rotate and translate the feature space of your samples, but will not skew them. PCA will only, therefore, be able to capture the underlying *linear* shapes and variance within your data and cannot discern any complex, nonlinear intricacies. For such cases, you will have to make use different dimensionality reduction algorithms, such as Isomap.

English ▾

POWERED BY