Dimensionality Reduction

* 1. Principal Component Analysis (PCA)
     1. PCA Intuition

What is the true purpose of PCA?

The true purpose is mainly to decrease the complexity of the model. It is to simplify the model while keeping relevance and performance. Sometimes you can have datasets with hundreds of features so in that case you just want to extract much fewer independent variables that explain the most the variance.

What is the difference between PCA and Factor Analysis?

Principal component analysis involves extracting linear composites of observed variables.

Factor analysis is based on a formal model predicting observed variables from theoretical latent factors. PCA is meant to maximize the total variance to look for distinguishable patterns, and Factor analysis looks to maximize the shared variance for latent constructs or variables.

Should I apply PCA if my dataset has categorical variables?

You could try PCA, but I would be really careful, because categorical values can have high variances by default and will usually be unstable to matrix inversion.

Apply PCA and do cross validation to see if it can generalize better than the actual data. If it does, then PCA is good for your model. (Your training matrix is numerically stable). However, I am certain that in most cases, PCA does not work well in datasets that only contain categorical data. Vanilla PCA is designed based on capturing the covariance in continuous variables. There are other data reduction methods you can try to compress the data like multiple correspondence analysis and categorical PCA etc.

What is the best extra resource on PCA?

Check out this [video](https://www.youtube.com/watch?v=FgakZw6K1QQ) that has an amazing explanation of PCA and studies it in more depth.

* + 1. PCA in Python

What does the fit\_transform do here? Why do we apply fit\_transform to the training set, and only transform to the test set?

In the fit\_transform method there is fit and transform.

The fit part is used to analyze the data on which we apply the object (getting the eigen values and the eigen vectors of the covariance matrix, etc.) in order to get the required information to apply the PCA transformation, that is, extracting some top features that explain the most the variance.

Then once the object gets these informations thanks to the fit method, the transform part is used to apply the PCA transformation.

And since the test set and the training set have very similar structures, we don’t need to create a new object that we fit to the test set and then use to transform the test set, we can directly use the object already created and fitted to the training set, to transform the test set.

How do you find out which ones of the independent variables are the top 2 components?

PCA is a feature extraction technique so the components are not one ones of the original independent variables. These are new ones, like some sort of transformations of the original ones.

It’s only with feature selection that you end up with ones of the original independent variables, like with Backward Elimination.

Is it better to use Feature Extraction or Feature Selection, or both? If both, in which order? Feature Extraction and Feature Selection are two great dimensionality reduction techniques, and therefore you should always consider both.

What I recommend is doing first Feature Selection to only keep the relevant features, and then apply Feature Extraction on these selected relevant features to reduce even more the dimensionality of your dataset while keeping enough variance.

How much total variance ratio do we need to use? Is there any threshold for good total variance ratio?

Generally a good threshold is 50%. But 60% is more recommended.

Is it more common to use exactly 2 independent variables to build a classifier, or do people typically use more than that?

In general people just extract a number of independent variables that explain a sufficient proportion of the variance (typically 60%). So it’s not always two. It can be more. And if it’s two that is great because then you can visualize better.

Is there a Python method or attribute that can help provide the underlying components and signs of the two principal components PC1 and PC2?

Sure you can access them with the components\_ attributes:

components = pca.components\_

* + 1. PCA in R

If I wanted to see the actual values for all the columns, how would I unscale the data which was feature scaled?

Say that your scaled vector was y\_train, then to unscale your result y\_pred do the following:

y\_pred\_unscaled = y\_pred\*attr(y\_train,’scaled:scale’) + attr(y\_train, ’scaled:center’)

How can I see the principal components in R?

You can run the following code:

pca$rotation

We got an accuracy of 100%, shouldn’t we be worried of overfitting?

If you look at the data, you’ll see that it’s almost perfectly separable by the 3 lines drawn, which means that the separability is a characteristic of the data rather than an overfitting problem.

If you had something like 50 lines and 100% accuracy (i.e., each section would capture precisely a small amount of points, guaranteeing that they are rightly classified), than it would probably be an overfitting issue, but here that is clearly not the case.

Besides we obtained 100% accuracy only on the test set. There were some mis-classified points in the training set. And overfitting is rather the opposite: an almost perfect accuracy on the training set and a poor one on the test set.

How can we know which are the two variables taken for plotting?

The two variables are not among your original independent variables, since they were extracted (Feature Extraction), as opposed to being selected (Feature Selection). The two new variables are the directions where there is the most variance, that is the directions where the data is most spread out.

* 1. Linear Discriminant Analysis (LDA)
     1. LDA Intuition

Could you please explain in a more simpler way the difference between PCA and LDA?

A simple way of viewing the difference between PCA and LDA is that PCA treats the entire data set as a whole while LDA attempts to model the differences between classes within the data. Also, PCA extracts some components that explain the most the variance, while LDA extracts some components that maximize class separability.

Feature Selection or Feature Extraction?

You would rather choose feature selection if you want to keep all the interpretation of your problem, your dataset and your model results. But if you don’t care about the interpretation and only car about getting accurate predictions, then you can try both, separately or together, and compare the performance results.

So yes feature selection and feature extraction can be applied simultaneously in a given problem.

Can we use LDA for Regression?

LDA is Linear Discriminant Analysis. It is a generalization of Fisher’s linear discriminant, a method used in statistics, pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification. However, for regression, we have to use ANOVA, a variation of LDA. LDA is also closely related to principal component analysis (PCA) and factor analysis in that they both look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data. PCA on the other hand does not take into account any difference in class, and factor analysis builds the feature combinations based on differences rather than similarities. Discriminant analysis is also different from factor analysis in that it is not an interdependence technique: a distinction between independent variables and dependent variables (also called criterion variables) must be made. LDA works when the measurements made on independent variables for each observation are continuous quantities. When dealing with categorical independent variables, the equivalent technique is discriminant correspondence analysis.

* + 1. LDA in Python

Which independent variables are found after applying LDA?

The two independent variables that you see, indexed by 0 and 1, are new independent variables that are not among your 12 original independent variables. These are totally new independent variables that were extracted through LDA, and that’s why we call LDA Feature Extraction, as opposed to Feature Selection where you keep some of your original independent variables.

How to decide the LDA n\_component parameter in order to find the most accurate result?

You can run:

LDA(n\\_components = None)

and it should give you automatically the ideal n\_components.

How can I get the two Linear Discriminants LD1 and LD2 in Python?

You can get them by running the following line of code:

lda.scalings\_

* 1. Kernel PCA
     1. Kernel PCA Intuition

Should Kernel PCA be used to convert non-linearly separable data into linearly separable data?

That’s right, but you don’t need to use Kernel PCA with a non linear classifier since the data will be linearly separable after applying Kernel PCA, and therefore a linear classifier will be sufficient.

When should we use PCA vs Kernel PCA?

You should start with PCA. Then if you get poor results, try Kernel PCA.

* + 1. Kernel PCA in Python

How do I know if my data is linearly separable or not?

A good trick is to train a Logistic Regression model on it first. If you get a really good accuracy, it should be (almost) linearly separable.

Is there a huge difference and what is better to use between Kernel PCA + SVM vs PCA + Kernel SVM?

Yes there is a difference.

Use Kernel PCA + SVM when you can transform your data into a non-linear low dimensional manifold where the points are separable.

Use PCA + Kernel SVM when you need to transform your data through a linear transformation into a low dimensional manifold, using these points to be transformed into a non-linear space where they are separable.

How do we decide which kernel is best for Kernel PCA?

The RBF Kernel is a great kernel, and is the best option in general. But the best way to figure out what kernel you need to apply is to do some Parameter Tuning with Grid Search and k-Fold Cross Validation.

We will see that in Part 10 - Model Selection.