A One-Stop Shop for Principal Component Analysis

Matt Brems Apr 17, 2017 · 15 min read

At the beginning of the textbook I used for my graduate stat theory class, the authors (George Casella and Roger Berger) explained in the preface why they chose to write a textbook:

> *"When someone discovers that you are writing a textbook, one or both of two questions will be asked. The first is "Why are you writing a book?" and the second is "How is your book different from what's out there?" The first question is fairly easy to answer. You are writing a book because you are not entirely satisfied with the available texts."*

I apply the authors' logic here. Principal component analysis (PCA) is an important technique to understand in the fields of statistics and data science… but when putting a lesson together for my General Assembly students, I found that the resources online were too technical, didn't fully address our needs, and/or provided conflicting information. It's safe to say that I'm not "entirely satisfied with the available texts" here.

As a result, I wanted to put together the "What," "When," "How," and "Why" of PCA as well as links to some of the resources that can help to further explain this topic. Specifically, I want to present the rationale for this method, the math under the hood, some best practices, and potential drawbacks to the method.

While I want to make PCA as accessible as possible, the algorithm we'll cover is pretty technical. Being familiar with some or all of the following will make this article and PCA as a method easier to understand: matrix operations/linear algebra (matrix multiplication, matrix transposition, matrix inverses, matrix decomposition, eigenvectors/eigenvalues) and statistics/machine learning (standardization, variance, covariance, independence, linear regression, feature selection). I've embedded links to illustrations of these topics throughout the article, but hopefully these will serve as a reminder rather than required reading to get through the article.

What is PCA?

Let's say that you want to predict what the gross domestic product (GDP) of the United States will be for 2017. You have lots of information available: the U.S. GDP for the first quarter of 2017, the U.S. GDP for the entirety of 2016, 2015, and so on. You have any publicly-available economic indicator, like the unemployment rate, inflation rate, and so on. You have U.S. Census data from 2010 estimating how many Americans work in each industry and American Community Survey data updating those estimates in between each census. You know how many members of the House and Senate belong to each political party. You could gather stock price data, the number of IPOs occurring in a year, and how many CEOs seem to be mounting a bid for public office. Despite being an overwhelming number of variables to consider, this *just scratches the surface*.

TL;DR — you have *a lot* of variables to consider.

If you've worked with a lot of variables before, you know this can present problems. Do you understand the relationships between each variable? Do you have so many variables that you are in danger of overfitting your model to your data or that you might be violating assumptions of whichever modeling tactic you're using?

You might ask the question, "How do I take all of the variables I've collected and focus on only a few of them?" In technical terms, you want to "reduce the dimension of your feature space." By reducing the dimension of your feature space, you have fewer relationships between variables to consider and you are less likely to overfit your model.

(Note: This doesn't immediately mean that overfitting, etc. are no longer concerns — but we're moving in the right direction!)

Somewhat unsurprisingly, ***reducing*** the ***dimension*** of the feature space is called "***dimensionality reduction***." There are many ways to achieve dimensionality reduction, but most of these techniques fall into one of two classes:

- Feature Elimination
- Feature Extraction

Feature elimination is what it sounds like: we reduce the feature space by eliminating features. In the GDP example above, instead of considering every single variable, we might drop all variables except the three we think will best predict what the U.S.'s gross domestic product will look like. Advantages of feature elimination methods include simplicity and maintaining interpretability of your variables.

As a disadvantage, though, you gain no information from those variables you've dropped. If we only use last year's GDP, the proportion of the population in manufacturing jobs per the most recent American Community Survey numbers, and unemployment rate to predict this year's GDP, we're missing out on whatever the dropped variables could contribute to our model. By eliminating features, we've also entirely eliminated any benefits those dropped variables would bring.

Feature extraction, however, doesn't run into this problem. Say we have ten independent variables. In feature extraction, we create ten "new" independent variables, where each "new" independent variable is a combination of each of the ten "old" independent variables. However, we create these new independent variables in a specific way and order these new variables by how well they predict our dependent variable.

You might say, "Where does the dimensionality reduction come into play?" Well, we keep as many of the new independent variables as we want, but we drop the "least important ones." Because we ordered the new variables by how well they predict our dependent variable, we know which variable is the most important and least important. But — and here's the kicker — because these new independent variables are

combinations of our old ones, we're still keeping the most valuable parts of our old variables, even when we drop one or more of these "new" variables!

Principal component analysis is a technique for *feature extraction* — so it combines our input variables in a specific way, then we can drop the "least important" variables while still retaining the most valuable parts of all of the variables! *As an added benefit, each of the "new" variables after PCA are all independent of one another.* This is a benefit because the assumptions of a linear model require our independent variables to be independent of one another. If we decide to fit a linear regression model with these "new" variables (see "principal component regression" below), this assumption will necessarily be satisfied.

When should I use PCA?

1. Do you want to reduce the number of variables, but aren't able to identify variables to completely remove from consideration?
2. Do you want to ensure your variables are independent of one another?
3. Are you comfortable making your independent variables less interpretable?

If you answered "yes" to all three questions, then PCA is a good method to use. If you answered "no" to question 3, you **should not** use PCA.

How does PCA work?

The section after this discusses *why* PCA works, but providing a brief summary before jumping into the algorithm may be helpful for context:

- We are going to calculate a matrix that summarizes how our variables all relate to one another.
- We'll then break this matrix down into two separate components: direction and magnitude. We can then understand the "directions" of our data and its "magnitude" (or how "important" each direction is). The screenshot below, from the setosa.io applet, displays the two main directions in this data: the "red direction" and the "green direction." In this case, the "red direction" is the more important one. We'll get into why this is the case later, but given how the dots are arranged, can you

see why the "red direction" looks more important than the "green direction?" (*Hint: What would fitting a line of best fit to this data look like?*)

Our original data in the xy-plane. (Source.)

- We will transform our original data to align with these important directions (which are combinations of our original variables). The screenshot below (again from setosa.io) is the same exact data as above, but transformed so that the x- and y-axes are now the "red direction" and "green direction." What would the line of best fit look like here?

Our original data transformed by PCA. (Source.)

- While the visual example here is two-dimensional (and thus we have two "directions"), think about a case where our data has more dimensions. By identifying which "directions" are most "important," we can compress or project our data into a smaller space by dropping the "directions" that are the "least important." **By projecting our data into a smaller space, we're reducing the dimensionality of our feature space… but because we've transformed our data in these different "directions," we've made sure to keep all original variables in our model!**

Here, I walk through an algorithm for conducting PCA. I try to avoid being too technical, but it's impossible to ignore the details here, so my goal is to walk through things as explicitly as possible. A deeper intuition of *why* the algorithm works is presented in the next section.

Before starting, you should have tabular data organized with n rows and likely $p+1$ columns, where one column corresponds to your dependent variable (usually denoted $\boldsymbol{Y}$) and p columns where each corresponds to an independent variable (the matrix of which is usually denoted $\boldsymbol{X}$).

1. If a $\boldsymbol{Y}$ variable exists and is part of your data, then separate your data into $\boldsymbol{Y}$ and $\boldsymbol{X}$, as defined above — we'll mostly be working with $\boldsymbol{X}$. (Note: if there exists no column for $\mathbf{Y}$, that's okay — skip to the next point!)

2. Take the matrix of independent variables $\boldsymbol{X}$ and, for each column, subtract the mean of that column from each entry. (This ensures that each column has a mean of zero.)

3. Decide whether or not to standardize. Given the columns of $\boldsymbol{X}$, are features with higher variance more important than features with lower variance, or is the importance of features independent of the variance? (In this case, importance means how well that feature predicts $\boldsymbol{Y}$.) **If the importance of features is independent of the variance of the features, then divide each observation in a**

column by that column's standard deviation. (This, combined with step 2, standardizes each column of X to make sure each column has mean zero and standard deviation 1.) Call the centered (and possibly standardized) matrix $\boldsymbol{Z}$.

4. Take the matrix $\boldsymbol{Z}$, transpose it, and multiply the transposed matrix by $\boldsymbol{Z}$. (Writing this out mathematically, we would write this as $\boldsymbol{Z}^T\boldsymbol{Z}$.) The resulting matrix is the covariance matrix of $\boldsymbol{Z}$, up to a constant.

5. (This is probably the toughest step to follow — stick with me here.) Calculate the eigenvectors and their corresponding eigenvalues of $\boldsymbol{Z}^T\boldsymbol{Z}$. This is quite easily done in most computing packages— in fact, the eigendecomposition of $\boldsymbol{Z}^T\boldsymbol{Z}$ is where we decompose $\boldsymbol{Z}^T\boldsymbol{Z}$ into $\boldsymbol{PDP}^{-1}$, where $\boldsymbol{P}$ is the matrix of eigenvectors and $\boldsymbol{D}$ is the diagonal matrix with eigenvalues on the diagonal and values of zero everywhere else. The eigenvalues on the diagonal of $\boldsymbol{D}$ will be associated with the corresponding column in $\boldsymbol{P}$ — that is, the first element of $\boldsymbol{D}$ is λ_1 and the corresponding eigenvector is the first column of $\boldsymbol{P}$. This holds for all elements in $\boldsymbol{D}$ and their corresponding eigenvectors in $\boldsymbol{P}$. We will always be able to calculate $\boldsymbol{PDP}^{-1}$ in this fashion. (Bonus: for those interested, we can always calculate $\boldsymbol{PDP}^{-1}$ in this fashion because $\boldsymbol{Z}^T\boldsymbol{Z}$ is a symmetric, positive semidefinite matrix.)

6. Take the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_p$ and sort them from largest to smallest. In doing so, sort the eigenvectors in $\boldsymbol{P}$ accordingly. (For example, if λ_2 is the largest eigenvalue, then take the second column of $\boldsymbol{P}$ and place it in the first column position.) Depending on the computing package, this may be done automatically. Call this sorted matrix of eigenvectors $\boldsymbol{P^*}$. (The columns of $\boldsymbol{P^*}$ should be the same as the columns of $\boldsymbol{P}$, but perhaps in a different order.) **Note that these eigenvectors are independent of one another.**

7. Calculate $\boldsymbol{Z^*} = \boldsymbol{ZP^*}$. This new matrix, $\boldsymbol{Z^*}$, is a centered/standardized version of $\boldsymbol{X}$ but now each observation is a combination of the original variables, where the weights are determined by the eigenvector. **As a bonus, because our eigenvectors in $\boldsymbol{P^*}$ are independent of one another, each column of $\boldsymbol{Z^*}$ is also independent of one another!**

An example from setosa.io where we transform five data points using PCA. The left graph is our original data **X**; the right graph would be our transformed data **Z***.

Note two things in this graphic:

- The two charts show the exact same data, but the right graph reflects the original data transformed so that our axes are now the principal components.
- In both graphs, the principal components are perpendicular to one another. **In fact, every principal component will ALWAYS be orthogonal** (a.k.a. official math term for perpendicular) **to every other principal component.** (Don't believe me? Try to break the applet!)

Because our principal components are orthogonal to one another, they are statistically linearly independent of one another… which is why our columns of *Z are linearly independent of one another!**

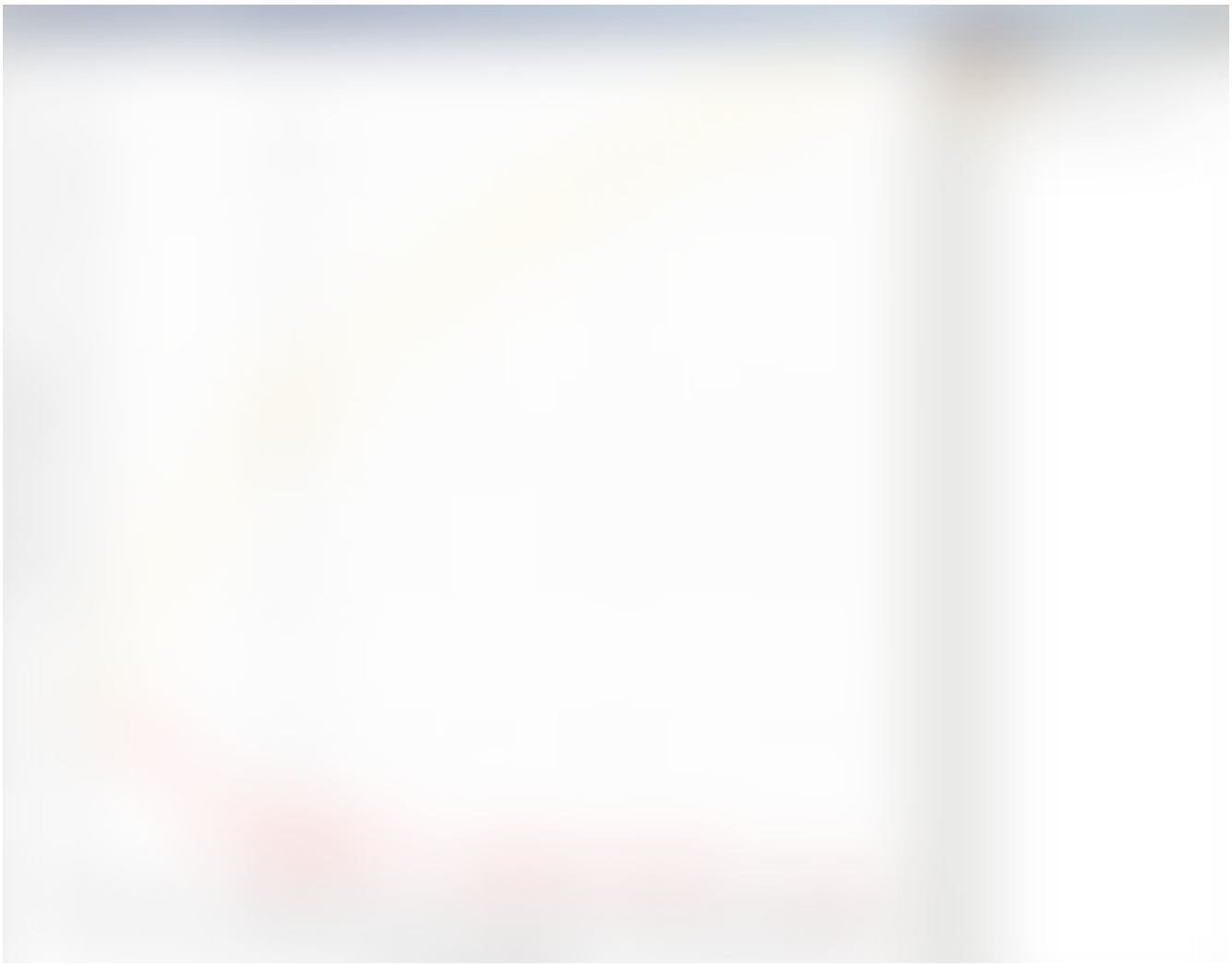
8. Finally, we need to determine how many features to keep versus how many to drop. There are three common methods to determine this, discussed below and followed by an explicit example:

- **Method 1**: We arbitrarily select how many dimensions we want to keep. Perhaps I want to visually represent things in two dimensions, so I may only keep two features. This is use-case dependent and there isn't a hard-and-fast rule for how many features I should pick.
- **Method 2**: Calculate the proportion of variance explained (briefly explained below) for each feature, pick a threshold, and add features until you hit that threshold. (For example, if you want to explain 80% of the total variability possibly explained by

your model, add features with the largest explained proportion of variance until your proportion of variance explained hits or exceeds 80%.)

- **Method 3**: This is closely related to Method 2. Calculate the proportion of variance explained for each feature, sort features by proportion of variance explained and plot the cumulative proportion of variance explained as you keep more features. (This plot is called a scree plot, shown below.) One can pick how many features to include by identifying the point where adding a new feature has a significant drop in variance explained relative to the previous feature, and choosing features up until that point. (I call this the "find the elbow" method, as looking at the "bend" or "elbow" in the scree plot determines where the biggest drop in proportion of variance explained occurs.)

Because each eigenvalue is roughly the importance of its corresponding eigenvector, the proportion of variance explained is the sum of the eigenvalues of the features you kept divided by the sum of the eigenvalues of all features.



Scree Plot for Genetic Data. (Source.)

Consider this scree plot for genetic data. (Source: here.) The red line indicates the proportion of variance explained by each feature, which is calculated by taking that principal component's eigenvalue divided by the sum of all eigenvalues. The proportion of variance explained by including only principal component 1 is $\lambda_1/(\lambda_1 + \lambda_2 + \ldots + \lambda_p)$, which is about 23%. The proportion of variance explained by including only principal component 2 is $\lambda_2/(\lambda_1 + \lambda_2 + \ldots + \lambda_p)$, or about 19%.

The proportion of variance explained by including both principal components 1 and 2 is $(\lambda_1 + \lambda_2)/(\lambda_1 + \lambda_2 + \ldots + \lambda_p)$, which is about 42%. This is where the yellow line comes in; the yellow line indicates the cumulative proportion of variance explained if you included all principal components up to that point. For example, the yellow dot above PC2 indicates that including principal components 1 and 2 will explain about 42% of the total variance in the model.

Now let's go through some examples:

- Method 1: We arbitrarily select a number of principal components to include. Suppose I wanted to keep five principal components in my model. In the genetic data case above, these five principal components explains about 66% of the total variability that would be explained by including all 13 principal components.
- Method 2: Suppose I wanted to include enough principal components to explain 90% of the total variability explained by all 13 principal components. In the genetic data case above, I would include the first 10 principal components and drop the final three variables from $\boldsymbol{Z}^*$.
- Method 3: Here, we want to "find the elbow." In the scree plot above, we see there's a big drop in proportion of variability explained between principal component 2 and principal component 3. In this case, we'd likely include the first two features and drop the remaining features. As you can see, this method is a bit subjective as "elbow" doesn't have a mathematically precise definition and, in this case, we'd include a model that explains only about 42% of the total variability.

(Note: Some scree plots will have the size of eigenvectors on the Y axis rather than the proportion of variance. This leads to equivalent results, but requires the user to manually calculate the proportion of variance. An example of this can be seen here.)

Once we've dropped the transformed variables we want to drop, we're done! That's PCA.

But, like, *why* does PCA work?

While PCA is a very technical method relying on in-depth linear algebra algorithms, it's a relatively intuitive method when you think about it.

- First, the covariance matrix $\boldsymbol{Z}^T\boldsymbol{Z}$ is a matrix that contains estimates of how every variable in $\boldsymbol{Z}$ relates to every other variable in $\boldsymbol{Z}$. Understanding how one variable is associated with another is quite powerful.
- Second, eigenvalues and eigenvectors are important. Eigenvectors represent directions. Think of plotting your data on a multidimensional scatterplot. Then one can think of an individual eigenvector as a particular "direction" in your scatterplot of data. Eigenvalues represent magnitude, or importance. Bigger eigenvalues correlate with more important directions.
- Finally, we make an assumption that more variability in a particular direction correlates with explaining the behavior of the dependent variable. Lots of variability usually indicates signal, whereas little variability usually indicates noise. Thus, the more variability there is in a particular direction is, theoretically, indicative of something important we want to detect. (The setosa.io PCA applet is a great way to play around with data and convince yourself why it makes sense.)

Thus, PCA is a method that brings together:

1. A measure of how each variable is associated with one another. (Covariance matrix.)
2. The directions in which our data are dispersed. (Eigenvectors.)
3. The relative importance of these different directions. (Eigenvalues.)

PCA combines our predictors and allows us to drop the eigenvectors that are relatively unimportant.

Are there extensions to PCA?

Yes, more than I can address here in a reasonable amount of space. The one I've most frequently seen is principal component regression, where we take our untransformed $\boldsymbol{Y}$ and regress it on the subset of $\boldsymbol{Z^*}$ that we didn't drop. (This is where the independence of the columns of $\boldsymbol{Z^*}$ comes in; by regressing $\boldsymbol{Y}$ on $\boldsymbol{Z^*}$, we know that the required independence of independent variables will necessarily be satisfied. However, we will need to still check our other assumptions.)

The other commonly-seen variant I've seen is kernel PCA.

Conclusion

I hope you found this article helpful! Check out some of the resources below for more in-depth discussions of PCA. Let me know what you think, especially if there are suggestions for improvement.

I've been told that a Chinese translation of this article has been made available here. (Thanks, Jakukyo Friel!)

I want to offer many thanks to my friends Ritika Bhasker, Joseph Nelson, and Corey Smith for their suggestions and edits. You should check Ritika and Joseph out on Medium — their posts are far more entertaining than mine. (Corey is too focused on not getting his Ph.D. research scooped to have a Medium presence.)

I also want to give a **huge** h/t to the setosa.io applet for its visual and intuitive display of PCA.

Edit: Thanks to Michael Matthews for noticing a typo in the formula for $\boldsymbol{Z^*}$ in Step 7 above. He correctly pointed out that $\boldsymbol{Z^* = ZP^*}$, not $\boldsymbol{Z^TP^*}$. Thanks also to Chienlung Cheung for noticing another typo in Step 8 above and noted that I had conflated "eigenvector" with "eigenvalue" in one line.

Resources You Should Check Out:

This is a list of resources I used to compile this PCA article as well as other resources I've generally found helpful to understand PCA. **If you know of any resources that would be a good inclusion to this list, please leave a comment and I'll add them.**

Non-Academic Articles and Resources

- Setosa.io's PCA applet. (An applet that allows you to visualize what principal components are and how your data affect the principal components.)
- A semi-academic walkthrough of building blocks to the PCA algorithm and the algorithm itself.
- The top answer to this StackExchange question is, in a word, outstanding.
- A CrossValidated question and answer discussing whether there are parametric assumptions to PCA. (Spoiler alert: PCA itself is a nonparametric method, but regression or hypothesis testing after using PCA might require parametric assumptions.)
- A resource list would hardly be complete without the Wikipedia link, right? (Despite Wikipedia being low-hanging fruit, it has an solid list of additional links and resources at the bottom of the page.)

Coding Resources

- Python Documentation for PCA within the sklearn library. (This link includes examples!)
- PCA Explanation on AnalyticsVidhya. (This link includes Python and R.)
- Implementing PCA in Python with a few cool plots.
- Comparison of methods for implementing PCA in R.

Academic Textbooks and Articles

- An Introduction to Statistical Learning, 6th printing, by James, Witten, Hastie, and Tibshirani. (PCA is covered extensively in chapters 6.3, 6.7, and 10.2. This book assumes knowledge of linear regression but is pretty accessible, all things considered.)

- Notes from Penn State's STAT 505 (Applied Multivariate Statistical Analysis) Course. (I've found Penn State's online statistics course notes to be incredible, and the PCA section here is particularly helpful.)
- Linear Algebra and Its Applications, 4th edition, by David Lay. (PCA is covered in chapter 7.5.)
- A Tutorial on Principal Components Analysis, by Jonathon Shlens at Google Research.
- A draft chapter on Principal Component Analysis from Cosma Shalizi of Carnegie Mellon University.
- A chapter on data preprocessing from Applied Predictive Modeling includes an introductory discussion of principal component analysis (with visuals!) in Section 3.3. (h/t to Jay Lucas for the recommendation!)
- Elements of Statistical Learning, 10th printing, by Hastie, Tibshirani, and Friedman. (PCA is covered extensively in chapters 3.5, 14.5, and 18.6. This book assumes knowledge of linear regression, matrix algebra, and calculus and is significantly more technical than *An Introduction to Statistical Learning*, but the two follow a similar structure given the common authors.)

Tangential Resources

- Essence of Linear Algebra YouTube Series (Including one video on Eigenvectors and Eigenvalues that is especially relevant to PCA; h/t to Tim Book for making me aware of this incredible resource.)


Download on the
App Store


GET IT ON
Google Play