Principal Components Analysis

Wednesday Bushong

4/28/2017

The Data & Problem

In this lab, we're going to be looking at the relationship between events and positive mood. There are five categories of events:

- positive academic
- negative academic
- positive relational
- negative relational
- negative financial

Throwing all of these factors into a single model seems ill-advised because they overlap conceptually (and numerically) *a lot*. Instead, it could be more informative to reduce these to a few dimensions using principal components analysis (PCA), and then using scores based on the most informative components to predict our dependent variables using linear regression. Let's load in our libraries and data and take a look:

```
library(foreign)
library(psych)
library(paramap)
library(ggplot2)

d <- read.spss("PCA Data.sav", to.data.frame = TRUE)
head(d)
```

```
##   posmood negmood totmood posacd negacd   posrel   negrel negfin
## 1     4.8     4.8     0.0   6.50    2.6 5.000000 4.000000    4.0
## 2     5.6     2.8     2.8   6.25    3.0 5.500000 1.666667    1.0
## 3     5.4     1.6     3.8   4.50    2.8 3.666667 1.833333    1.5
## 4     4.2     3.2     1.0   6.00    3.4 4.166667 4.500000    3.5
## 5     4.2     1.8     2.4   4.75    2.6 3.833333 2.500000    1.5
## 6     6.2     2.0     4.2   4.75    3.2 3.000000 1.666667    1.0
```

Conceptual Background

The idea of PCA is to take some high-dimensional (here, 5-dimensional) space and reduce it to its most informative dimensions. We do this by computing the **eigenvectors** of the **covariance matrix** of our factors of interest. Our number of eigenvectors is equal to our number of original dimensions because all these do is re-arrange our values onto new axes (where the new axes are the eigenvectors). What's cool about eigenvectors are that they have associated **eigenvalues** that correspond to the amount of variance in the data that a single eigenvector explains. Then what we can do is take the first couple of eigenvectors that explain most of the variance and throw away the ones that don't give us much. Then we've reduced the space quite a bit!

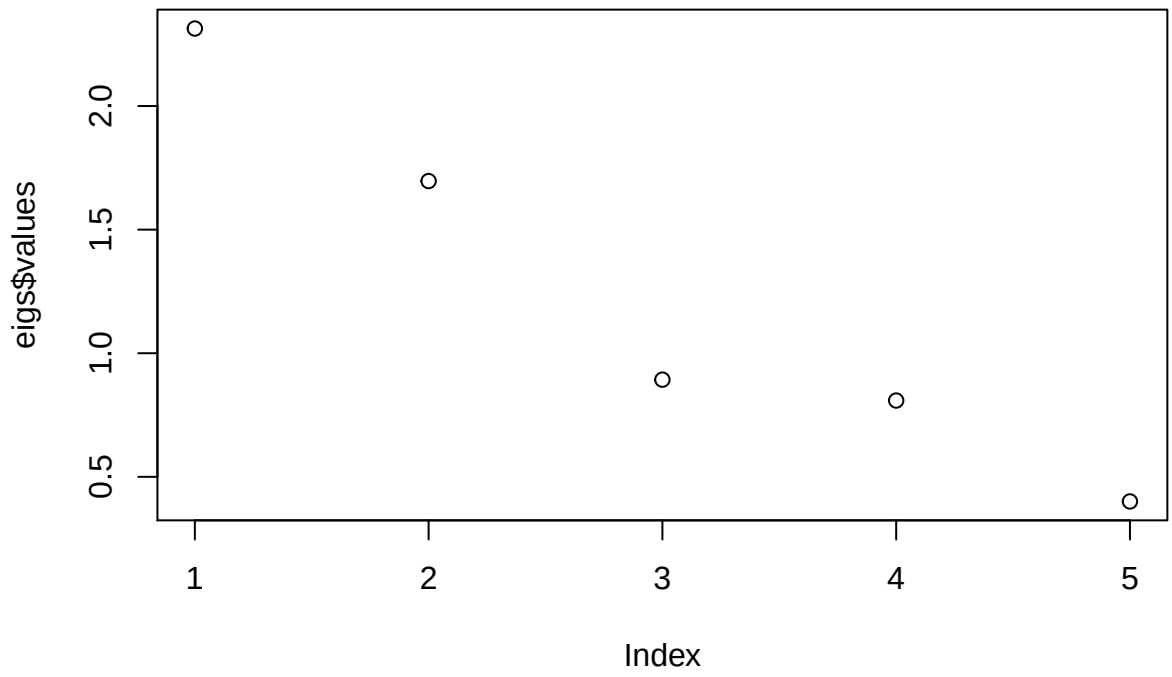
Visualizing selecting number of PCs to use

In the rest of the lab we'll be using convenient functions that do the annoying computations for us. But first, let's manually compute the eigenvectors and eigenvalues ourselves and plot them. Remember that the principal components are the eigenvectors of the covariance matrix. We can compute the covariance matrix with the `cov()` function, and then get the eigenvectors and values with the `eigen()` function:

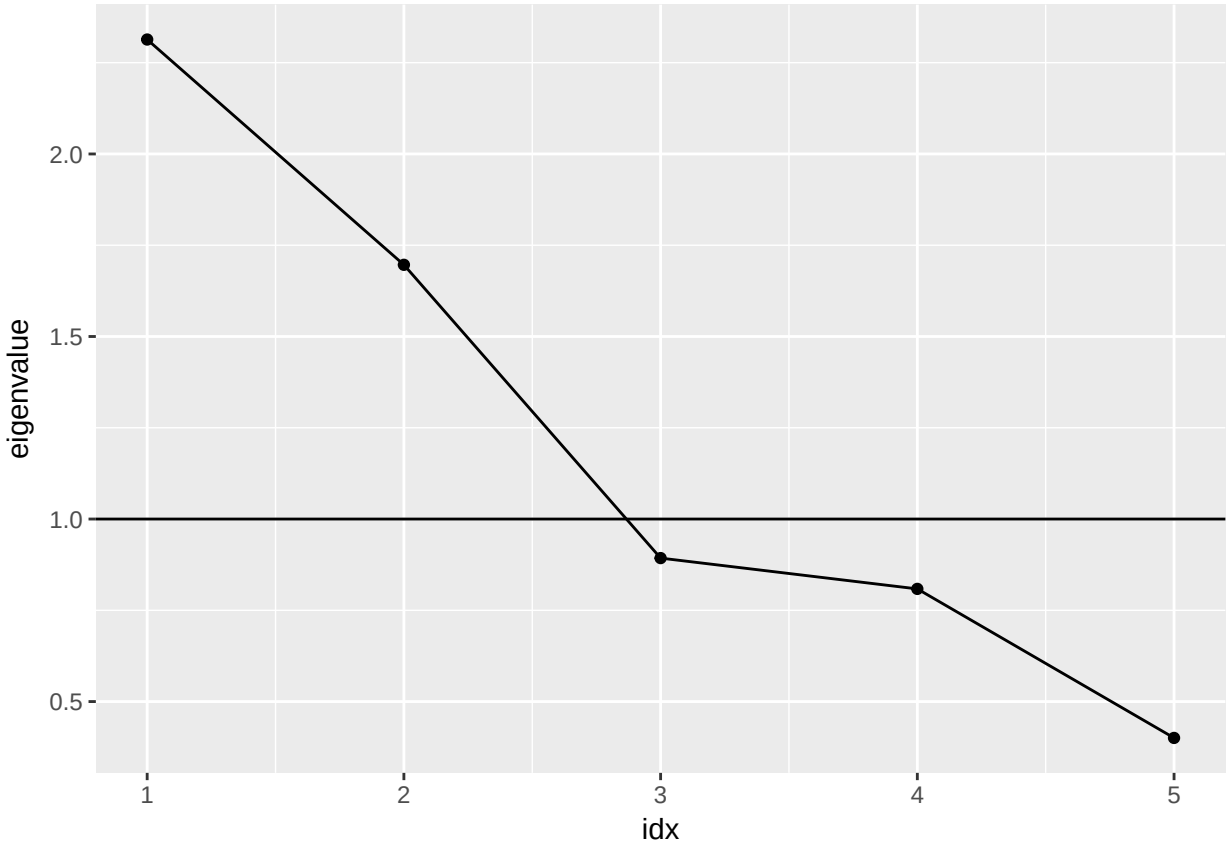
```
covmat <- cov(d[, c("posacd", "negacd", "posrel", "negrel", "negfin")])
eigs <- eigen(covmat)
```

Notice that this function returns a list with both the eigenvalues and the eigenvectors. Let's make a simple plot of the eigenvalues to see how many principal components we may want to utilize:

```
# simple plot version
plot(eigs$values)
```




```
# fancy ggplot version that's definitely overkill for this
eigvals <- data.frame(idx = 1:length(eigs$values),
                      eigenvalue= eigs$values)
ggplot(eigvals, aes(x = idx, y = eigenvalue)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = 1)
```



There are two rule-of-thumb cutoffs for determining how many principal components to use. One is to just use all of the principal components that have an eigenvalue above 1; here this would correspond to using the first two PCs. The other is to look at the plot and see where it "turns" – that is, see at which point you don't gain much more information about the data. Here, this would correspond to using the first 3 PCs, because the slope becomes flat after that.

Quantitatively selecting number of PCs to use

A more principled approach to selecting the number of PCs to use comes to us in the form of what's called *parallel analysis*. Here's the general approach:

- Generate a *completely random dataset* that has the same dimensions of our data
- Compute the PCs and their associated eigenvalues for that dataset
- Repeat N (usually 1000) times
- Compute the mean and 95th percentile eigenvalue for random datasets

The idea is that if, on average, the eigenvalue of a particular rank explains more variance in a random dataset than the same rank eigenvalue in our dataset, it's probably not telling us much about our data.

We'll take advantage of a library `paramap` which does parallel analysis for us:

```
random.eigvals <- parallel(Nvars = 5, Ncases = nrow(d),
         extract = "PCA", Ndatasets = 1000, percentile = 95,
         corkind = "pearson", display = "no")
```

```
random.eigvals
```

```
## $eigenvalues
##       Root      Mean   Percentile
## 1        1 1.2141786    1.3136343
## 2        2 1.0903596    1.1593915
## 3        3 0.9958542    1.0483735
## 4        4 0.9037306    0.9615727
## 5        5 0.7958770    0.8753700
```

As we can see, for random datasets with as many variables and observations as our original data, the first principal component on average has an eigenvalue of about 1.2, with the 95th percentile eigenvalue being about 1.3. Remember from our initial plotting above that our first PC had an eigenvalue over 2, which means we definitely want to keep our first PC! Let's paste our original eigenvalues onto the end of the random ones so that we can directly compare:

```
random.eigvals <- as.data.frame(random.eigvals$eigenvalues)
random.eigvals$ourEigvals <- eigs$values
```

Our first two PCs pass the 95th percentile test: they explain more variance in the data than the PCs for random datasets do. However, this breaks down on our third PC: the third-ranked PC in random datasets on average explains *more* variance than our third PC does on in our data. This is a sign that we shouldn't use it!

Now we're in a position to actually do PCA.

Principal Components Analysis

We'll use the `principal` function from the `psych` package to do PCA. Now that we know how many PCs we want to extract, we can specify to extract 2 components. You can also specify a rotation (to avoid the problem of cross-loading; see lecture for more details), which is varimax by default. Type "?principal" to get information about other possible rotations (e.g. "equimax", "oblimin", "none"; note that you will need to install the GPArotation package to be able to use some of these rotations). Varimax is an orthogonal rotation, but there are several non-orthogonal rotations.

```
pca <- principal(d[, c("posacd", "negacd", "posrel", "negrel", "negfin")],
                 nfactors = 2, rotate = "varimax")
```

If we look at what's contained in this object, there's a lot of stuff:

```
names(pca)
```

```
##  [1] "values"       "rotation"     "n.obs"        "communality"
##  [5] "loadings"     "fit"          "fit.off"      "fn"
##  [9] "Call"         "uniquenesses" "complexity"   "chi"
## [13] "EPVAL"        "R2"           "objective"    "residual"
## [17] "rms"          "factors"      "dof"          "null.dof"
## [21] "null.model"   "criteria"     "STATISTIC"    "PVAL"
## [25] "weights"      "r.scores"     "rot.mat"      "Structure"
## [29] "scores"
```

The default print method for this object will give us most of the information we're interested in:

```
pca
```

```
## Principal Components Analysis
## Call: principal(r = d[, c("posacd", "negacd", "posrel", "negrel", "negfin")],
##     nfactors = 2, rotate = "varimax")
## Standardized loadings (pattern matrix) based upon correlation matrix
##          RC1   RC2   h2   u2 com
## posacd -0.19  0.80 0.68 0.32 1.1
## negacd  0.73 -0.41 0.70 0.30 1.6
## posrel  0.17  0.78 0.64 0.36 1.1
## negrel  0.77 -0.01 0.59 0.41 1.0
## negfin  0.73  0.16 0.56 0.44 1.1
##
##                        RC1  RC2
## SS loadings           1.72 1.45
## Proportion Var        0.34 0.29
## Cumulative Var        0.34 0.63
## Proportion Explained  0.54 0.46
## Cumulative Proportion 0.54 1.00
##
## Mean item complexity =  1.2
## Test of the hypothesis that 2 components are sufficient.
##
## The root mean square of the residuals (RMSR) is  0.16
##  with the empirical chi square  89.35  with prob <  3.3e-21
##
## Fit based upon off diagonal values = 0.62
```

We can notice a few things here:

- The "Standardized Loadings" section shows us how much
- The *Cumulative Var* row shows us that the two PCs combined explain 63% of the variance in our data

One piece of additional information we are interested in is the *communalities* of each variable in our dataset. Communalities represent the amount of variance in a single variable accounted for by our PCs (it's like the cumulative variance we saw above, but separately for each original factor in our data).

```
pca$communality
```

```
##    posacd    negacd    posrel    negrel    negfin
## 0.6813504 0.7006666 0.6362619 0.5896944 0.5574119
```

This isn't the case here, but if some factor was not well captured by the PCs we might consider dropping it and entering it into our final regression model separately. A reasonable cutoff is ~0.2-0.3.

Assigning Variables to Components

Now, we need to decide how to transform our original variables into values on our PCs so that we can use them as predictors in a regression model. This is the point where we really go from 5 dimensions to 2. We have two broad options of how to do this:

Unit Scores

In this scenario, we look at the loadings of each variable on the PCs and choose which variables are "represented" by that PC. Then, we take the relevant variables in our dataset and for each row we average their scores. If any of the variables load negatively, we first "reverse-score" them (see below) before adding them to the averaging process. Notice that this loses all of the detail contained in the original PCs; we draw a binary line of whether a variable contributes to a PC or not, and then treat them all as equal.

First, we need to decide which variables each PC represents. This is commonly done by taking all variables whose absolute value of their loading is above 0.5:

```
pca$loadings
```

```
##
## Loadings:
##        RC1    RC2
## posacd -0.195  0.802
## negacd  0.729 -0.411
## posrel  0.166  0.780
## negrel  0.768
## negfin  0.730  0.158
##
##                  RC1   RC2
## SS loadings    1.719 1.446
## Proportion Var 0.344 0.289
## Cumulative Var 0.344 0.633
```

Here, we'll choose:

- PC1: `negacd`, `negrel`, `negfin`. Notice that these conceptually go nicely together as well – they're all related to negative events
- PC2: `posacd`, `posrel`. Again, this nicely corresponds to positive events!

Now we'll compute the unit scores. First we want to make sure that all of our variables are on the same scale by z-scoring them:

```
d$posacd.z <- scale(d$posacd)
d$negacd.z <- scale(d$negacd)
d$posrel.z <- scale(d$posrel)
d$negrel.z <- scale(d$negrel)
d$negfin.z <- scale(d$negfin)
```

Now, we create our new unit score by averaging the variables that we decided on above:

```
d$PC1.unit.score <- rowMeans(d[, c("negacd.z", "negrel.z", "negfin.z")])
d$PC2.unit.score <- rowMeans(d[, c("posacd.z", "posrel.z")])
```

Notice that because we didn't divide up our variables gradiently, there's a possibility that there is still some overlap in the information in the two scores we've created. We can check this by looking at the correlation between scores:

```r
cor(d[, c("PC1.unit.score", "PC2.unit.score")])
```

```
##                 PC1.unit.score PC2.unit.score
## PC1.unit.score       1.0000000     -0.1341011
## PC2.unit.score      -0.1341011      1.0000000
```

However, units scores have the distinct advantage of being *very* interpretable, since we can say that it's a composite of the variables we selected.

Reverse-Scoring

We didn't have any factors that loaded negatively on our PCs, but if we did, we would want to reverse-score them before we average them together! To do this, you can just take the (unscaled) variable and subtract it from the maximum. You could code this like `data$var.reverse <- max(data$var) - data$var`. Then, you'll want to z-score that new variable: `data$var.reverse.z <- scale(data$var.reverse)`.

Exact Scores

In this scenario, instead of making a binary decision about which variables go with which PC, we just project our original values onto the axis defined by the PCs and use those values as our scores. Luckily for us, this information is already contained in our `pca` object:

```r
d$PC1.exact.score <- pca$scores[, "RC1"] # scores on 1st PC
d$PC2.exact.score <- pca$scores[, "RC2"]
```

Notice that since we divided up the score gradiently, our scores now don't really correlate with each other, meaning that they'll each contribute unique information in our final regression model:

```r
cor(d[, c("PC1.exact.score", "PC2.exact.score")])
```

```
##                  PC1.exact.score PC2.exact.score
## PC1.exact.score    1.000000e+00    1.020658e-15
## PC2.exact.score    1.020658e-15    1.000000e+00
```

However, exact scores have the disadvantage of being less interpretable since they to some extent carry information about all of the variables in our dataset. The primary benefit of exact scores is to reduce the covariance of predictors in your regression models, but since we only have two variables here this is unlikely to be a problem.

Weighted Scores

With weighted scores, you get the best of both worlds: the conceptual cleanliness of assigning each variable to only one PC, but also the gradiency of how much each factor contributes to the PC.

First, we decide on which factors will go with which PCs. We already did this above. Recall:

- PC1: `negacd`, `negrel`, `negfin`
- PC2: `posacd`, `posrel`

Now we need to compute the weights on each factor, which we do by normalizing the loadings and using them as weights:

```
PC1.weights <- pca$loadings[c("negacd", "negrel", "negfin"), "RC1"]
PC1.weights <- PC1.weights/sum(PC1.weights)
d$PC1.weighted.score <- d$negacd.z*PC1.weights["negacd"] + d$negrel.z*PC1.weights["negrel"] +
  d$negfin.z*PC1.weights["negfin"]

PC2.weights <- pca$loadings[c("posacd", "posrel"), "RC2"]
PC2.weights <- PC2.weights/sum(PC2.weights)
d$PC2.weighted.score <- d$posacd.z*PC2.weights["posacd"] + d$posrel.z*PC2.weights["posrel"]
```

I also want to point out that these measures really aren't that different... look at the correlations between the three different measures:

```
cor(d[, c("PC1.weighted.score", "PC1.unit.score", "PC1.exact.score")])
```

```
##                     PC1.weighted.score PC1.unit.score PC1.exact.score
## PC1.weighted.score          1.0000000      0.9998977       0.9860933
## PC1.unit.score              0.9998977      1.0000000       0.9861914
## PC1.exact.score             0.9860933      0.9861914       1.0000000
```

```
cor(d[, c("PC2.weighted.score", "PC2.unit.score", "PC2.exact.score")])
```

```
##                     PC2.weighted.score PC2.unit.score PC2.exact.score
## PC2.weighted.score          1.0000000      0.9999500       0.9739873
## PC2.unit.score              0.9999500      1.0000000       0.9738469
## PC2.exact.score             0.9739873      0.9738469       1.0000000
```

The real difference between these comes out in datasets where factors load very differently on PCs.

Regression using PCs as predictors

Now we are finally in a place to use our PC scores as predictors of mood in a linear regression! We have the option of either using the unit or exact scores in our analysis – here I'll show you both, but in real life you will want to choose one a priori because the more models you fit to try to find an effect the higher your false positive rate will be! Most of the time, unit scores are more clear, but exact scores can be useful if you're fitting a large model with many predictors and covariance between predictors is a concern.

Unit Scores

```
m.unit <- lm(posmood ~ PC1.unit.score + PC2.unit.score, d)
summary(m.unit)
```

```
##
## Call:
## lm(formula = posmood ~ PC1.unit.score + PC2.unit.score, data = d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.23541 -0.55882  0.07303  0.60128  2.09875
```

```
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     5.04091    0.06928  72.766  < 2e-16 ***
## PC1.unit.score -0.33641    0.09314  -3.612 0.000398 ***
## PC2.unit.score  0.64855    0.08630   7.516 2.92e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.919 on 173 degrees of freedom
## Multiple R-squared:  0.3113, Adjusted R-squared:  0.3034
## F-statistic: 39.11 on 2 and 173 DF,  p-value: 9.706e-15
```

Recall that PC1 was the concept of negative events and PC2 was positive events. In that context, our results make a lot of sense! Positive events are a positive predictor of positive mood, and negative events are a negative predictor of positive mood.

Exact Scores

```
m.exact <- lm(posmood ~ PC1.exact.score + PC2.exact.score, d)
summary(m.exact)
```

```
##
## Call:
## lm(formula = posmood ~ PC1.exact.score + PC2.exact.score, data = d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.46859 -0.53426  0.07632  0.64135  2.18367
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.04091    0.07097  71.028  < 2e-16 ***
## PC1.exact.score -0.24866    0.07117  -3.494 0.000605 ***
## PC2.exact.score  0.52374    0.07117   7.359 7.18e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9415 on 173 degrees of freedom
## Multiple R-squared:  0.2772, Adjusted R-squared:  0.2689
## F-statistic: 33.18 on 2 and 173 DF,  p-value: 6.361e-13
```

Here, we get the same results but a bit weaker, since as we saw above these scores incorporate all of the original variables, some of which go against what seems to be the underlying concept of the PC (e.g., incorporating some of the information from negative events into the PC that we thought seemed like the "positive event" concept).