PCA2

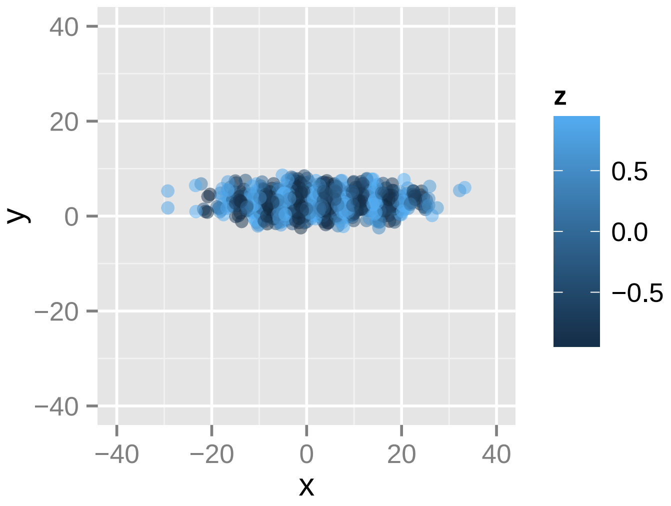
Bob Horton

March 20, 2015

Principle component analysis uses linear algebra to reduce the number of dimensions in a data set. Consider this collection of (x,y) values:

```r
library("knitr")
opts_chunk$set( fig.height=4.0, fig.width=4.0 )
```

```r
library(magrittr)

N <- 1000
x <- rnorm(N, mean=2, sd=10)
y <- rnorm(N, mean=3, sd=2)

plot_xyz <- function(df){
    library(ggplot2)
    ggplot(df, aes(x=x, y=y, col=z)) +
        geom_point( shape=19, alpha=0.5) +
        coord_fixed(ratio = 1) +
        ylim(c(-40,40)) + xlim(c(-40, 40))
}

xyz_df <- data.frame( x, y, z=sin(x) )

plot_xyz(xyz_df)
```



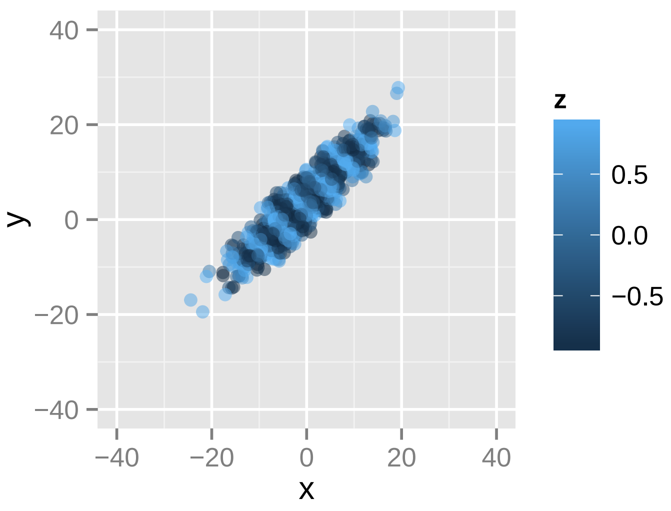
Note that the outcome `z` depends only on variable `x`, and not on variable `y` (I made it a sin function so the data would have nice stripes). We can predict the outcome pretty well if we are given a value for `x`, but it doesn't help to know about `y`.

Now let's take the same data and rotate it 45 degrees. The outcome `z` for each point remains the same, but the mapping of the original `x` and `y` columns of the dataframe to the axes of the plot have been changed by the rotation.

```r
rotate2d <- function(theta)
    matrix(c(cos(theta), -sin(theta), sin(theta), cos(theta)), ncol=2)

#' rotate the x, y columns of a data frame by theta degrees
rotate_xy <- function(df, theta){
    library(dplyr)
    xy_rotated <- df %>% select(x, y) %>%
        as.matrix %>%
        (function(M) M %*% rotate2d(theta))

    df$x <- xy_rotated[,1]
    df$y <- xy_rotated[,2]
    df
}

xyz_df2 <- xyz_df %>% rotate_xy(pi/4)
```

```
##
## Attaching package: 'dplyr'
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
plot_xyz(xyz_df2)
```



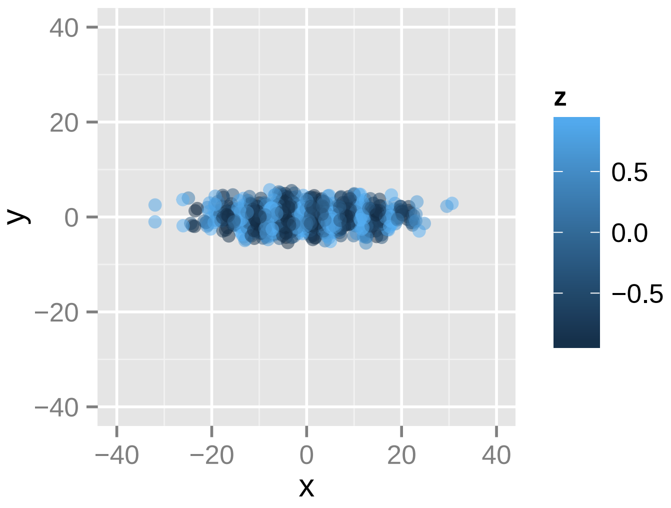
The rotated data has the same elliptical shape as the original data, but now it varies on both the x and y axes. We know that z can be expressed as a function of a single input, but the rotation obscures that relationship.

Note that we used a matrix operation to perform the rotation; it should not come as a big surprise that we can use a matrix operation to change it back. Rather that multiplying by a rotation matrix, though, we will use a Principle Components Analysis (PCA).

```r
pca <- princomp(xyz_df2[1:2])

summary(pca)
```

```
## Importance of components:
##                           Comp.1     Comp.2
## Standard deviation     9.8981396 2.02750177
## Proportion of Variance 0.9597315 0.04026847
## Cumulative Proportion  0.9597315 1.00000000
```

```r
pca_xyz <- as.data.frame(cbind(pca$scores, xyz_df$z))
names(pca_xyz) <- c('x', 'y', 'z')
plot_xyz(pca_xyz)
```



Note that the original standard deviations are very close to the standard deviations of the principle components.

PCA only looked at the x and y values, not at z. It is entirely based on correlations between input vectors. In fact, you don't even need the original data to perform PCA, you can use the correlation (or covariance) matrix instead of raw data.

In practice, PCA is generally applied to scaled values. In our case the x and y values were in the same units (say millimeters)