PCA: R and Rcpp

November 3, 2020

1 Data

Data Source: a subset of gene expression data associated with different conditions of fungal stress in cotton which is published in Bedre et al., 2015

data source download link: *https* : *//reneshbedre.github.io//assets/posts/pca/cot*$_p$*ca.csv*

2 R

Step 1: Import libraries and read in file.

```
library(rgl)
library(ggplot2)
setwd("path_to_file")
data <- read.csv("cot_pca.csv")
```

Step 2: Normalize Data.
To normalize data: calculate the mean and standard deviation for a variable (column). Then, for each observed value of the variable, subtract the mean and divide by the standard deviation.

```
normalize <- function(x){
    return ((x-mean(x))/sd(x))
}

norm_data <- data %>%
    apply(2, normalize) %>%
as.data.frame
head(norm_data)
```

-	A	B	C	D	E	F
1	0.61918576	0.44794157	-0.2406848	2.45520142	2.3029908	-0.3312387
2	0.34202781	-0.04146802	-0.4283278	-1.21381404	-0.8764883	0.4745708
3	0.60287270	0.49431947	-0.1762513	2.89438077	3.1313612	0.1094798
4	-0.03280001	0.41311071	0.4490437	2.49557475	1.6284758	-1.1709650
5	0.02195115	-0.41113310	-0.4450132	3.76211921	2.9636277	-1.1597403
6	-1.31687291	-0.53550580	0.9188528	0.01653463	1.0068069	1.1618042

Step 3: Calculate correlation matrix.
Has the same variables shown in the rows and columns. The line of 1.00s in the diagonal shows that each variable always perfectly correlates with itself. This matrix is symmetrical, with the same correlation is shown above the main diagonal being a mirror image of those below the main diagonal.

```
cor_mat <- cor(norm_data)
cor_mat
```

-	A	B	C	D	E	F
A	1.00000000	0.5489786	-0.5447870	-0.00024496	-0.1177486	-0.1340140
B	0.54897863	1.00000000	0.38792602	0.05226432	0.02523271	-0.02923557
C	-0.5447870108	0.38792602	1.00000000	0.04694102	0.15416464	0.12428642
D	-0.0002449692	0.05226433	0.04694102	1.00000000	0.65056154	-0.46304741
E	-0.1177486844	0.02523271	0.15416464	0.6505615384	1.00000000	0.36055078
F	-0.13401401	-0.02923557	0.12428642	-0.46304741	0.36055078	1.00000000

Step 4: Find eigenvalues and eigenvectors from the correlation matrix. Then sort the eigenvalues in descending order.

```
eig <- eigen(cor_mat)
eigen_vals <- eig$values
eigen_vec <- eig$vectors
sorted_eigenvals <- sort(eigen_vals,
                         decreasing=T)
sorted_eigenvals
eigen_vec
```

[eigenvalues] 1.787245203 1.648875138 1.390886491 1.157498300 0.008661193 0.006833674

[eigenvectors]

[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
-0.5108978	-0.4522338	0.22735615	-0.32346414	0.614880563	0.008372017
-0.0859082	-0.4011971	0.70855590	0.13278818	-0.558447706	-0.010615577
0.4774770	0.1009940	0.46243668	0.48795135	0.556605227	0.007892690
0.3703182	-0.6114855	-0.30829455	0.05497302	-0.007642200	0.625159163
0.5684909	-0.3001175	-0.01177538	-0.48411477	0.009382374	-0.593425464
0.2080900	0.4004256	0.37044008	-0.63423386	-0.010111495	0.506731846

Step 5: I calculate the variance explained and the cumulative variance

```
var_exp <- sorted_eigenvals/sum(eigen_vals)
cum_sum <- cumsum(var_exp)
cum_sum
```

[1] 0.2978742 0.5726867 0.8045011 0.9974175 0.9988611 1.0000000

Step 6: I decided to use the first 3 principal components because it covers 81% of the data and the first three components have eigenvalues greater than 1. Then, I get the transformation matrix.

```
trans_matrix <- eigen_vec[,1:3]
trans_data <- -as.matrix(norm_data)
                %*% trans_matrix
head(trans_data)
```

```
-       [,1]       [,2]       [,3]
    -0.5108978 -0.4522338 0.22735615
    -0.0859082 -0.4011971 0.70855590
     0.4774770  0.1009940 0.46243668
     0.3703182 -0.6114855 -0.30829455
     0.5684909 -0.3001175 -0.01177538
     0.2080900  0.4004256 0.37044008
```

Step 7: Plot data

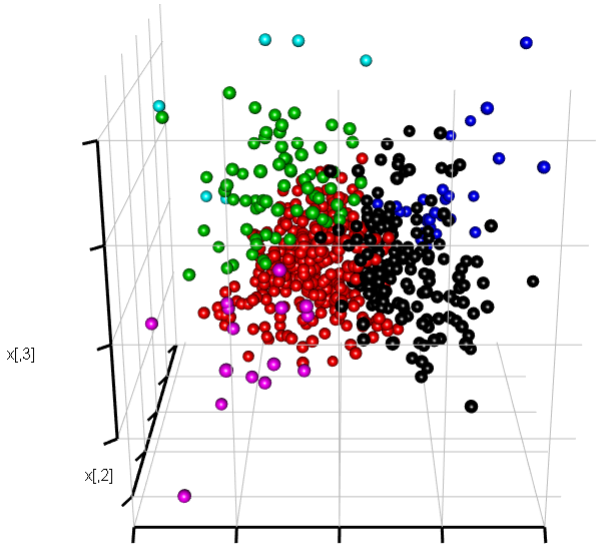
```
fit <- hclust(dist(trans_data[,1:3]),
                method="complete")
groups <- cutree(fit, k=6)

plotPCA <- function(x, nGroup) {
  n <- ncol(x)
  if(!(n %in% c(2,3))) { # check if 2d or 3d
    stop("x must have either 2 or 3 columns")
  }

  fit <- hclust(dist(x), method="complete")
  groups <- cutree(fit, k=nGroup)

  if(n == 3) { # 3d plot
    plot3d(x, col=groups, type="s",
                size=1, axes=F)
    axes3d(edges=c("x--", "y--", "z"),
           lwd=3,axes.len=2, labels=FALSE)
    grid3d("x")
    grid3d("y")
    grid3d("z")
  } else { # 2d plot
    maxes <- apply(abs(x), 2, max)
    rangeX <- c(-maxes[1], maxes[1])
    rangeY <- c(-maxes[2], maxes[2])
    plot(x, col=groups, pch=19,
           xlab=colnames(x)[1]
```

```
                , ylab=colnames(x)[2],
               xlim=rangeX, ylim=rangeY)
    lines(c(0,0), rangeX*2)
    lines(rangeY*2, c(0,0))
  }
}
plotPCA(trans_data[,1:3], 6)
```



3 Rcpp

This is the same method but in Rcpp using armadillo.

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
// [[Rcpp::export]]

DataFrame normalize(DataFrame x) {
  DataFrame ret=x;

  for (int i=0;i<x.size();i++){
    NumericVector A= x[i];
    float mean_=mean(A);
    float sdev=sd(A);
    //Rcout<<mean_<<", "<<sdev<<"\n";
    for (int j=0;j<A.size();j++){
```

```
      NumericVector temp=ret[i];
      temp[j]=(A[j]-mean_)/sdev;

    }

  }
  return ret;

}
// [[Rcpp::export]]
NumericMatrix dfToMatrix(DataFrame x) {
  Function asMatrix("as.matrix");
  return asMatrix(x);
}
// [[Rcpp::export]]
arma::mat convertDataFrame(NumericMatrix nm){
    arma::mat y = as<arma::mat>(nm) ;
    return(y) ;
}
// [[Rcpp::export]]
arma::vec getEigenValues(arma::mat M) {
  arma::vec eigval = sort(arma::eig_sym(M),
                                  "descend");
  return eigval;
}
// [[Rcpp::export]]
arma::mat getEigenVectors(arma::mat M) {
  arma::vec V;
  arma::mat eigvectors;
  arma::eig_sym(V, eigvectors, M);
  arma::mat ret=M;
  unsigned int ctr=M.n_cols-1;

  for(unsigned int i=0;i<M.n_cols;i++){
    ret.col(i)=eigvectors.col(ctr);
    ctr=ctr-1;
  }
  return ret;

}
// [[Rcpp::export]]
arma::vec getVarExp(arma::mat M){
  arma::vec res=M;
  float sum_=sum(res);
  for (unsigned int i=0;i<M.size();i++){
    res[i]=res[i]/sum_;
```

```
  }
  return res;
}
// [[Rcpp::export]]
arma::vec getCumSum(arma::mat M){
  arma::vec res=M;
  float sum_=0;
  for (unsigned int i=0;i<M.size();i++){
    sum_=M[i]+sum_;
    res[i]=sum_;
  }
  return res;
}
// [[Rcpp::export]]
arma::mat getTransMatrix(arma::mat M,
                         arma::mat norm){
  arma::mat res(M.n_rows,3);
  for (unsigned int i=0;i<3;i++){
    res.col(i)=M.col(i);
  }

  return norm*res;
}
/*** R
setwd("C:/Users/meenu/OneDrive/UCM")
dataPath<-"C:/Users/meenu/OneDrive/UCM"
data <- read.csv("cot_pca.csv")
features<-c("A")
normalized<-normalize(data)
norm_matrix<-dfToMatrix(normalized)
nnorm_matrix<-convertDataFrame(norm_matrix)
head(nnorm_matrix)
cor_mat<-cor(normalized)
corr_matrix<-dfToMatrix(cor_mat)

ncorr_matrix<-convertDataFrame(corr_matrix)
ncorr_matrix
eig_values<-getEigenValues(ncorr_matrix)

var_exp<-getVarExp(eig_values)
cum_sum<-getCumSum(var_exp)
eig_vectors<-getEigenVectors(ncorr_matrix)

trans_matrix<-getTransMatrix(eig_vectors,
                             norm_matrix)
head(trans_matrix)
```

```
*/
```

4 Conclusions

Using proc.time() starting from loading in the csv file to plotting the graph in R, I calculated the time it took to run the program. Below are the results measured in seconds.

user	system	elapsed
0.47	0.19	1.39

Then, I did the same for Rcpp. Below are the results measured in seconds.

user	system	elapsed
0.00	0.04	1.06

It is evident that Rcpp is much faster.