PCA, cPCA, occPCA

Robin Tu

10/1/2019

Recap of SVD and some nice properties:

SVD decomposes a $n \times p$ matrix $X = UDV^T$:

- U is a left unitary $n \times n$ matrix. U is the orthonormal eigenvectors for $XX^T = UDV^T(UDV^T)^T = UD^2U^T$
- D is a rectangular diagonal $n \times p$ matrix
- V^T is a right unitary $p \times p$ matrix. V are the orthonormal eigenvectors for $X^TX = (UDV^T)^TUDV^T = VD^2V^T$.

Since U and V are unitary

- $U^T = U^{-1} \Rightarrow U^TU = I_n$
- $V^T = V^{-1} \Rightarrow VV^T = V^TV = I_p$

This decomposition can be thought of as rotation, scaling, and rotation again. SVD also allows you to compute the eigenvectors of the covariance matrix without calculating the covariance matrix and doing an eigen/spectral decomposition on the $p \times p$ matrix.

Why can the diagonal elements of D^2 be described as the variance explained by each dimension??

Helper Functions:

Turning NA's to Zeros.

```
## convert NA values to zero
NAtoZero = function(mat){
  mat[is.na(mat)] <- 0
  return(mat)
}
```

Ploting diagnostics for Principal Components

```
#screeplot for an SVD object
scree = function(svd_obj){
  var_explained <- svd_obj$d^2
  var_explained_plot <- data.table("Components" = 1:length(var_explained), "Value"=cumsum(var_explained)
  scree_plot <- data.table("Components" = 1:length(var_explained), "Value"=var_explained, "Analysis"="V

  plot_dat <- rbind(var_explained_plot, scree_plot)

  ggplot(data = plot_dat)+
    geom_point(aes(x = Components, y = Value))+
    geom_line(aes(x = Components, y = Value), color = "dodgerblue3")+
    facet_grid(Analysis~., scales = "free")+
    ggtitle("Diagnostic Plots")
  }
```

Crude Tests

make sure my helper functions work as they should

```
test_mat <- matrix(rnorm(1000*4, mean = 100, sd = 20), ncol = 4)

centered_scaled_obj <- scale(test_mat)
scaled_obj <- scale(test_mat, center = F, scale = apply(test_mat, 2, sd, na.rm=T))

# make sure the column means are close to zero (ie less than machine epsilon)
expect_equivalent(apply(centered_scaled_obj, 2, mean),expected = rep(0, 4))
# make sure variances are 1
expect_equivalent(apply(centered_scaled_obj, 2, var), expected = rep(1, 4))

# make sure the column means do not change
expect_equal(apply(scaled_obj, 2, mean), expected = colMeans(scaled_obj))
# make sure variances are 1
expect_equivalent(apply(scaled_obj, 2, var), expected = rep(1, 4))
```

PCA, cPCA, occPCA functions:

PCA: Principal Component Analysis

PCA seeks to find the direction v that maximizes the variance/covariance of Xv.

$$cov(Xv) = v^T C_x v$$

In other words, $v = \text{argmax}_{\|v\|_2=1} v^T C_x v$ where C_x is the covariance matrix of X. This could be found using eigen/spectral decomposition, or using the matrix V from the SVD decomposition, $X = UDV^T$. V gives the eigenvectors corresponding to the rotation explaining the largest variance to the smallest variance.

Another way to look at PCA is one of minimizing the reconstruction error of a generative model. Suppose $x_i \approx f(\lambda_i)$, where the function

$$f(\lambda) = \mu + V_r \lambda$$

parameterizes an affine set of dimension r. $\mu \in \mathbb{R}^p$, $V_r \in \mathbb{R}^{p \times r}$ is an matrix with orthonormal columns and $\lambda \in \mathbb{R}^r$ is a parameter vector over samples. Finding the V_r that minimizes:

$$\frac{1}{N} \sum_{i=1}^{N} \|x_i - \mu - V_r \lambda_i\|_2^2$$

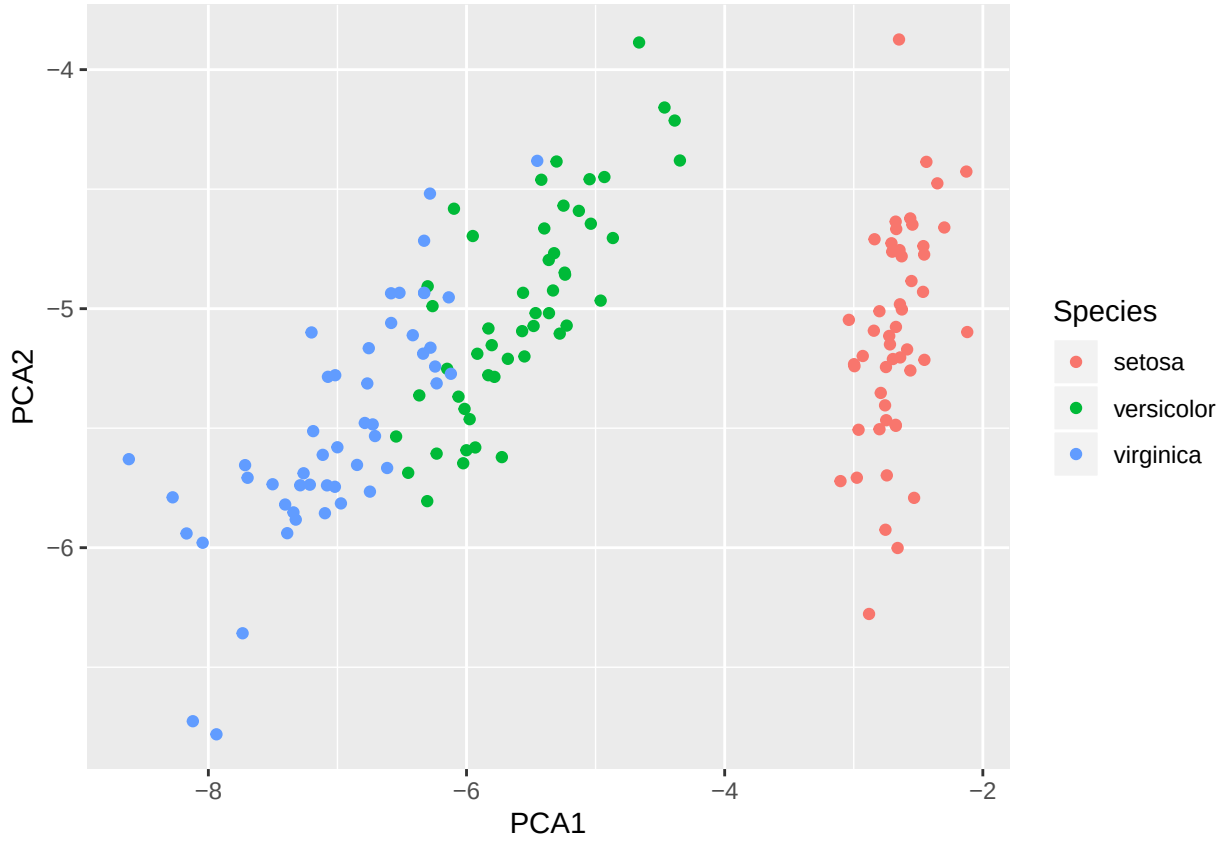
For some μ and λ yields the same solution. If data has been precentered, where $\mu = 0$ this simplifies to:

$$\frac{1}{N} \sum_{i=1}^{N} \|x_i - V_r V_r^T x_i\|_2^2$$

An example of using PCA on the `iris` dataset:

```
data(iris)
iris <- data.table(iris)
scaled_iris <- scale(iris[, .SD, .SDcols = - "Species"])
```

```r
# double checking that the results of eigen and svd are the same up to a sign change
expect_equivalent(abs(svd(cov(scaled_iris))$v), abs(eigen(cov(scaled_iris))$vectors))

iris_pca <- cbind((data.matrix(iris[, .SD, .SDcols = -"Species"]) %*% svd(cov(scaled_iris), nv = 2)$v),
setnames(iris_pca, c("PCA1", "PCA2", "Species"))

ggplot(data = iris_pca)+
  geom_point(aes(x = PCA1, y = PCA2, color = Species))
```



PCA function:

- Function inputs:
  - `target`: Target dataset - dataset of interest
  - `n_components`: number of Principal components to calculate
  - `return_all`: (logical) whether top eigenvectors should be returned
- Function outputs:
  - Data projected on the principal components

```r
PCA = function(target, n_components = 2, standardize = T, return_eigenvectors = F){
  if(!is.matrix(target)){
    target <- as.matrix(target)
  }
  og_target <- target
  if(standardize){
    target = scale(target);
  }
```

```
  target_cov = cov(target);
  v_top <- svd(target_cov, nv = n_components)$v
  reduced_target <- og_target %*% v_top

  if(return_eigenvectors){
    return(list("reduced_target"=reduced_target, "vectors"=v_top))
  }else{
    return(list("reduced_target"=reduced_target))
  }
}
```

cPCA: Contrastive PCA

Author's written function in Python

- Description: Suppose you have two datasets, a `target` and a `background` where the `target` were a dataset containing information on cases and `background` contained information on controls, or uninteresting variation. cPCA works directly on the covariance matrices and seeks to find the directions/rotation (eigenvector) that maximizes the variance explained in the `target` and minimize the variance explained in the `background`. More specifically:

$$\mathbf{v}^* = \text{argmax}_{\mathbf{v} \in \mathbb{R}^d_{unit}} v^T \left(C_{target} - \alpha C_{bg}\right) v$$

Where C_{target} and C_{bg} are the sample covariance matrices.

- Function inputs:
  - `target`: Target dataset - dataset of interest
  - `bg`: Background dataset
  - `n_components`: number of Principal components to calculate
  - `alpha`: tuning parameter for how hard to weight minimizing the variance of the rotated background data
  - `return_all`: (logical) whether top eigenvectors should be returned
- Function outputs:
  - Data projected on the contrastive principal components

```
cPCA = function(target, bg, n_components = 2, alpha = 1, standardize = T, return_all = F){

  if(!is.matrix(target) | !is.matrix(bg)){
    target <- as.matrix(target)
    bg <- as.matrix(bg)
  }
  og_target <- target

  if(standardize){
    target = scale(target);
    bg = scale(bg);
  }

  target_cov = cov(target);
  bg_cov = cov(bg);

  sigma = target_cov - alpha * bg_cov
  v_top <- svd(sigma, nv = n_components)$v
```

```
  reduced_target <- og_target %*% v_top
  reduced_bg <- bg %*% v_top

  if(return_all){
    return(list("reduced_target" = reduced_target, "reduced_bg" = reduced_bg, "vectors" = v_top))
  }else{
    return(list("reduced_target" = reduced_target))
  }
}
```

occPCA: Orthogonal Complement cPCA

Description: Finding the principal components of the target data projected onto the space orthogonal to the vectors that explain the most variation in the background data

Assumption:

- target data exists in a lower dimensional subspace spanned by background data.

Advantages:

- You can repeat this procedure on more than 1 background dataset by repeatedly projecting onto the orthogonal complement of the background data.
- Since the objective function is not working on the covariance matrix, you can extend to other methods which have objective functions that work with the reconstruction error. NMF, penalized PCA, Factor Analysis, etc.
- Do not have to tune α, the weight in cPCA, but still need to pick the number of dimensions that captures most of the variation in the background data. Arguably, there are already best practices (total variance explained/screeplot) for this and it's less abstract/abitrary than tuning α in cPCA, and the results don't differ much based on the tuning parameter

Procedure:

1. Calculate the Top Principal Components of the background data
2. Compute the projection onto the Top Principal Components of the background data
   - since the eigenvectors, V_{bg} computed by `SVD` are orthonormal, the projection simplifies to $V_{bg}V_{bg}'$
   - this is because $P_{bg} = V_{bg}(V_{bg}'V_{bg})^{-1}V_{bg}'$, but $(V_{bg}'V_{bg})^{-1} = I_k$
3. Project the target onto the orthogonal complement of the background, $X_{target}P_{bg}$
   - $X_{target}P_{bg}$ projects X_{target} onto the rowspace of P_{bg}
   - $P_{bg} = P_{bg}^T$ means that X_{target} is also projected onto the column space of P_{bg}
4. Find the PCs of the target projected onto the orthogonal complement of the background.
5. Rotate the Target data onto the new PCs

- Function inputs:
  - `target`: Target dataset - dataset of interest
  - `bg`: Background dataset
  - `n_components`: number of Principal components to calculate for the target data, after being projected onto the orthogonal complement of the background
  - `bg_components`: number of background principal components used. Tuning parameter because this affects the span of the Orthogonal Complement
  - `return_all`: (logical) whether to return the background PCs and target projected on the Orthogonal Complement of the background
- Function outputs:
  - Data projected on the Orthogonal Complement contrastive principal components

```
occPCA = function(target, bg, n_components = 2, bg_components = 2, standardize = T, return_all = F){
  if(!is.matrix(target) | !is.matrix(bg)){
```

```
    target <- as.matrix(target)
    bg <- as.matrix(bg)
  }
  og_target <- target

  if(standardize){
    target = scale(target);
    bg = scale(bg);
  }
  # Rotate the background
  bg_svd <-svd(bg, nv = bg_components)

  # Background Projection Matrix
  bg_projection <- tcrossprod(bg_svd$v)

  #projection onto the orthogonal complement
  oc_target <-  target %*% (diag(nrow = nrow(bg_projection)) - bg_projection)

  reduced_target <- og_target %*% svd(oc_target, nv = n_components)$v

  if(return_all){
    return(list("reduced_target" = reduced_target, "bg_svd" = bg_svd, "oc_target" = oc_target))
  }else{
    return(list("reduced_target" = reduced_target))
  }
}
```

Replicating Figure 3a. in the paper

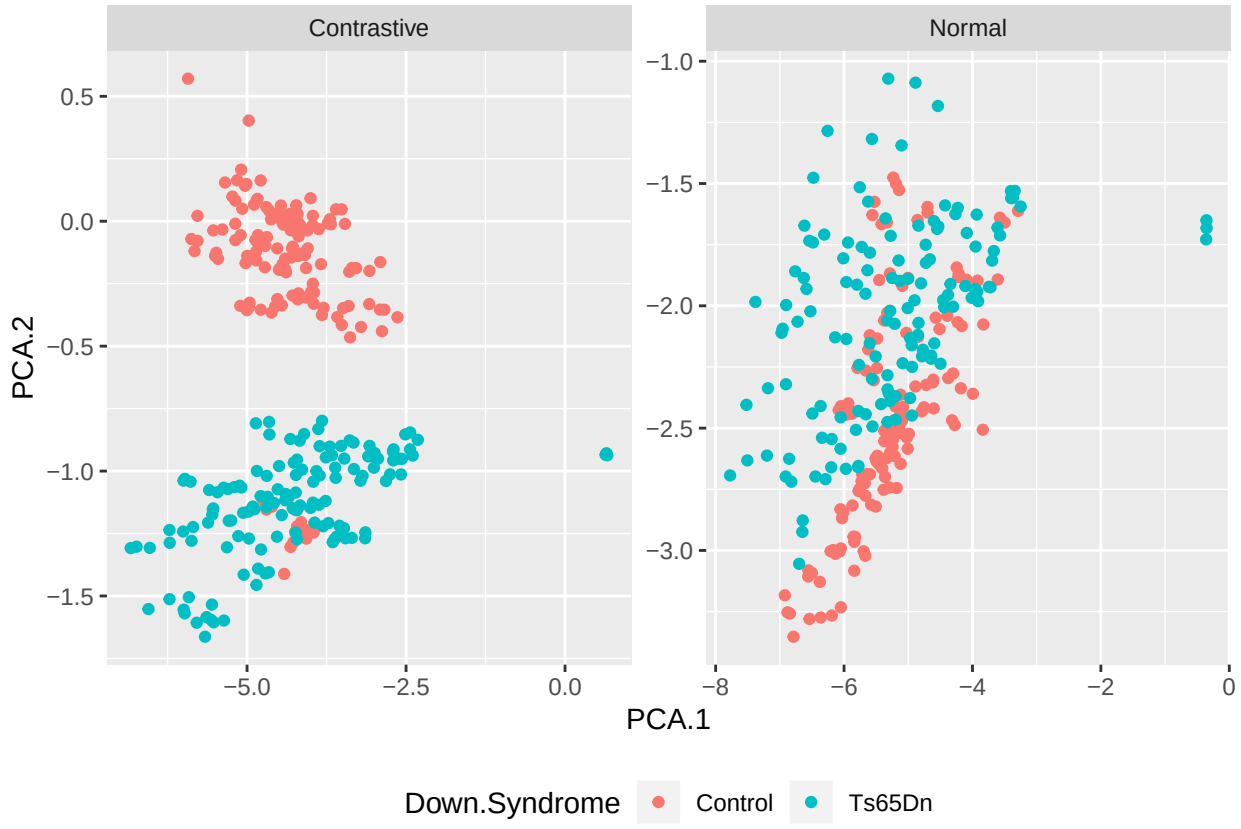
Authors Data Analysis

The original data had some missing data in it. From the Author's anaylsis, missing values were turned to zero.

```
mouse <- fread('../contrastive/experiments/datasets/Data_Cortex_Nuclear.csv')
mouse <- NAtoZero(mouse)

targ <- mouse[Behavior == "S/C" & Treatment == "Saline" & Genotype %in% c("Control", "Ts65Dn"), .SD, .SI
background <- mouse[Behavior == "S/C" & Treatment == "Saline" & Genotype == "Ts65Dn", .SD, .SDcols = -c
```

```
results_cPCA <- cPCA(target = targ[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     bg = background[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     alpha = 1)

reduced_target <- data.table(results_cPCA[[1]])
reduced_target <- cbind(reduced_target,  targ[, .SD, .SDcols = c("Genotype")], "Contrastive")

normal_pca <- data.table(PCA(target = targ[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")])[[1
normal_pca <- cbind(normal_pca, targ[, .SD, .SDcols = c("Genotype")], "Normal")

first_pass_plot <- rbind(reduced_target, normal_pca)
setnames(first_pass_plot, c("PCA.1", "PCA.2", "Down.Syndrome", "Method"))

ggplot(data = first_pass_plot)+
```

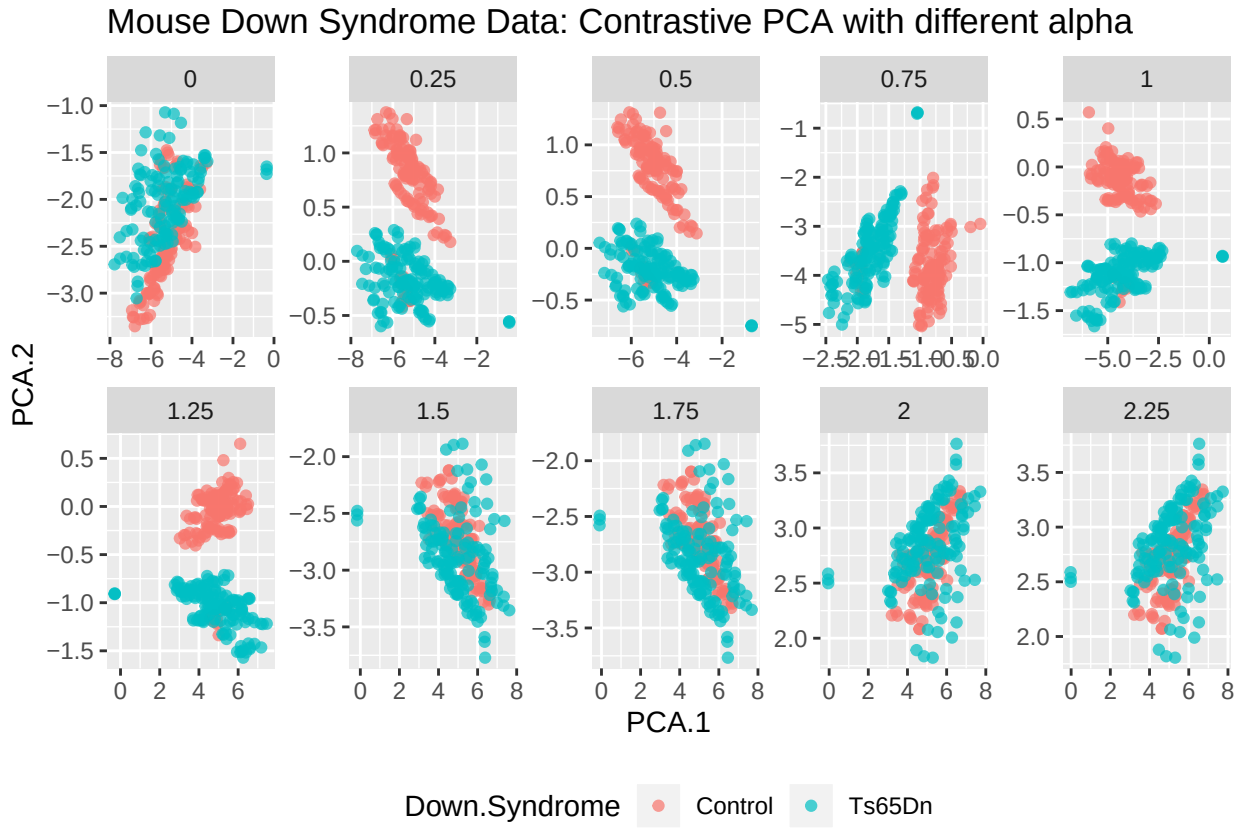
```
  geom_point(aes(x = PCA.1, y=PCA.2, color = Down.Syndrome))+
  facet_wrap(~Method, scale = "free")+
  theme(legend.position = "bottom")
```



Playing with the tuning parameter alpha

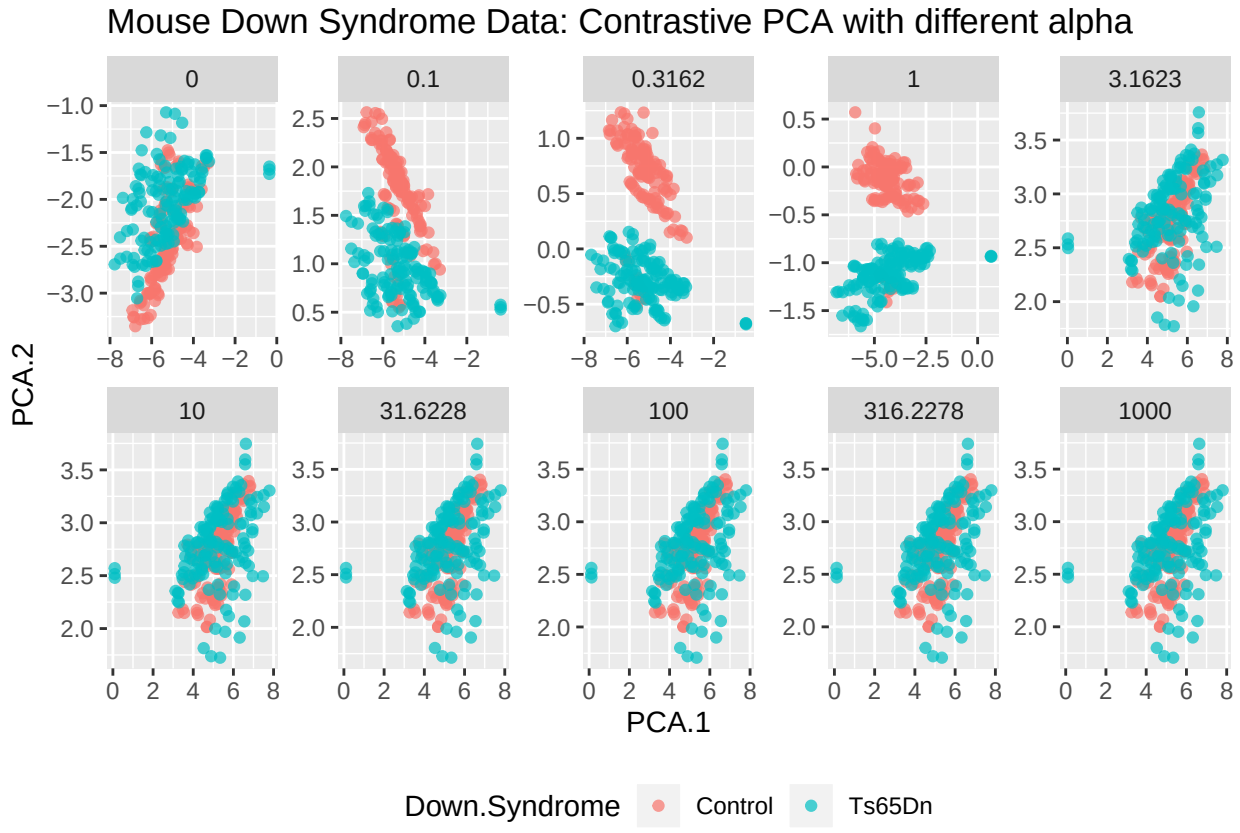
Contrastive PCA seems very sensitive to the tuning parameter α

```
search_grid <- seq(0, 2.25, by = 0.25)

tuning_alpha <- lapply(search_grid, function(xx) cPCA(target = targ[, .SD, .SDcols = -c("Genotype", "Tr
     bg = background[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     alpha = xx))

tuning_alpha_plot <- data.table(do.call(rbind, lapply(tuning_alpha, "[[", 1)),
                                sort(rep(search_grid, nrow(targ))),
                                unlist(rep(targ[, "Genotype"],length(search_grid))))
setnames(tuning_alpha_plot, c("PCA.1", "PCA.2", "alpha", "Down.Syndrome"))

ggplot(data = tuning_alpha_plot)+
  geom_point(aes(x = PCA.1, y = PCA.2, color = Down.Syndrome), alpha = 0.7)+
  facet_wrap(~alpha, scales = "free", nrow = 2)+
  theme(legend.position = "bottom")+
  labs(title = "Mouse Down Syndrome Data: Contrastive PCA with different alpha")
```



Using the log spaced values for the tuning parameter α... They chose the best α based on Spectral Clustering

```
search_grid2 <- c(0, 10^(seq(-1, log10(1000), length.out = 9)))

tuning_alpha <- lapply(search_grid2, function(xx) cPCA(target = targ[, .SD, .SDcols = -c("Genotype", "T
     bg = background[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     alpha = xx))

tuning_alpha_plot <- data.table(do.call(rbind, lapply(tuning_alpha, "[[", 1)),
                                sort(rep(round(search_grid2, 4), nrow(targ))),
                                unlist(rep(targ[, "Genotype"],length(search_grid2))))
setnames(tuning_alpha_plot, c("PCA.1", "PCA.2", "alpha", "Down.Syndrome"))

ggplot(data = tuning_alpha_plot)+
  geom_point(aes(x = PCA.1, y = PCA.2, color = Down.Syndrome), alpha = 0.7)+
  facet_wrap(~alpha, scales = "free", nrow = 2)+
  theme(legend.position = "bottom")+
  labs(title = "Mouse Down Syndrome Data: Contrastive PCA with different alpha")
```

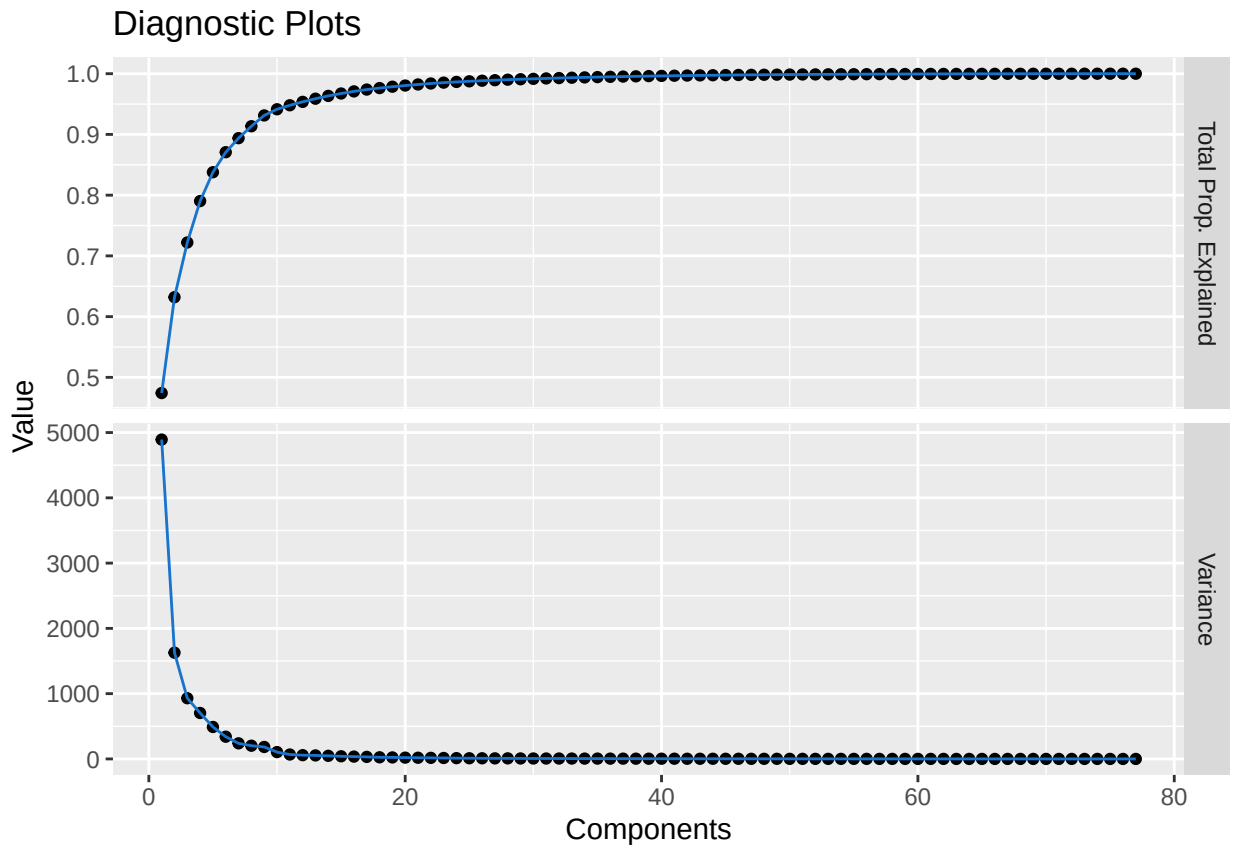


occPCA

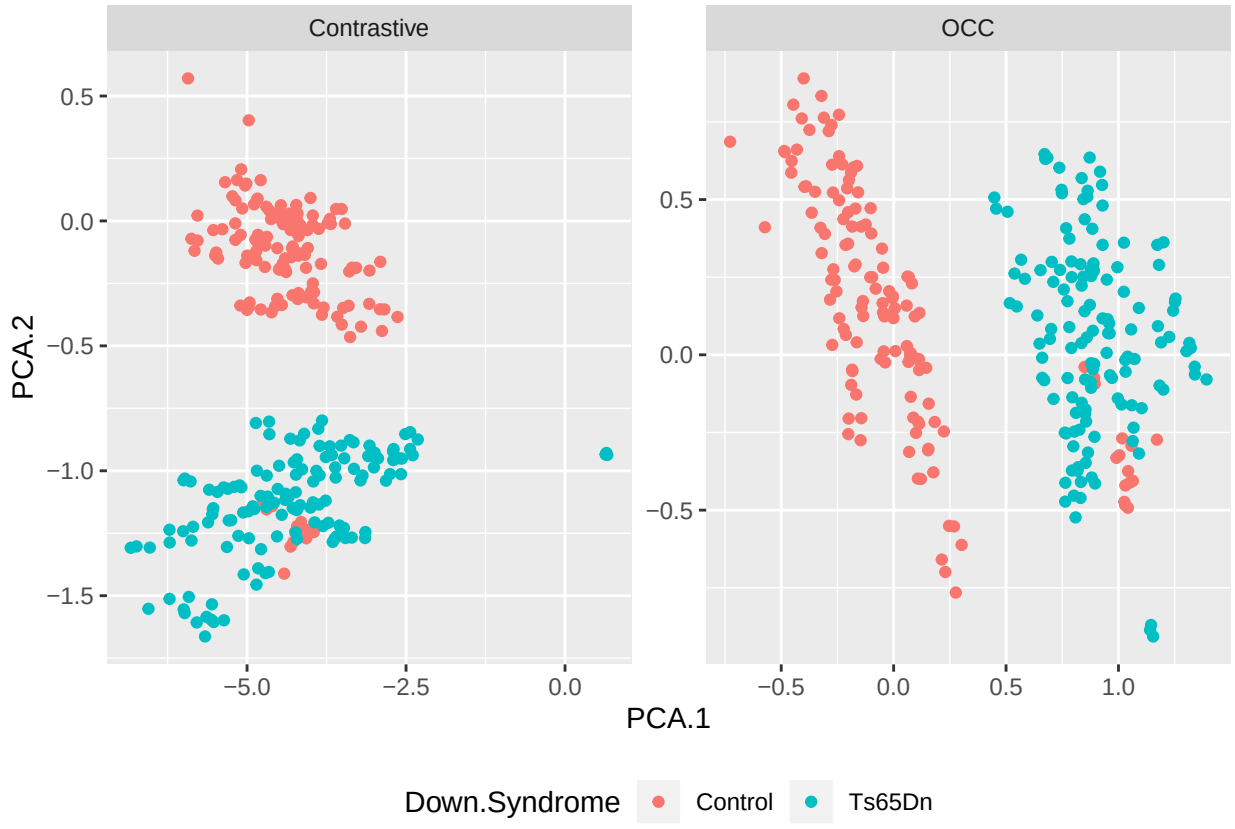
```
results_occPCA <- occPCA(target = targ[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     bg = background[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     bg_components = 3,
     return_all = T)

occ_reduced_target <- data.table(results_occPCA[[1]])
occ_reduced_target <- cbind(occ_reduced_target,  targ[, .SD, .SDcols = c("Genotype")], "OCC")
setnames(occ_reduced_target, c("PCA.1", "PCA.2", "Down.Syndrome", "Method"))

second_pass_plot <- rbind(first_pass_plot[Method == "Contrastive"], occ_reduced_target)
```

diagnostic plots for the background data

```
#screeplot
scree(results_occPCA$bg_svd)
```



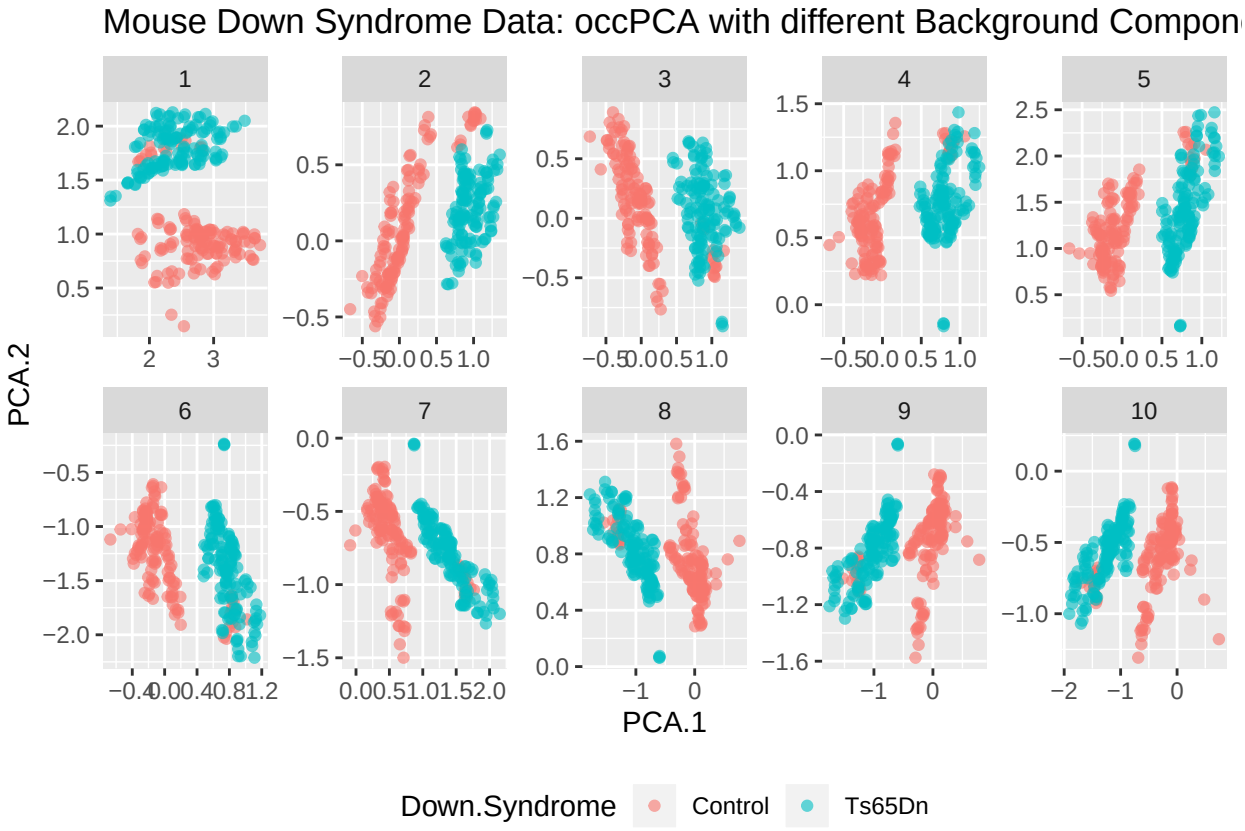

```
ggplot(data = second_pass_plot)+
  geom_point(aes(x = PCA.1, y=PCA.2, color = Down.Syndrome))+
  facet_wrap(~Method, scale = "free")+
  theme(legend.position = "bottom")
```



Playing with the tuning bg_components

The results are robust to first k number of background principal components, unlike choosing the α parameter in cPCA. Clean seperation occurs at every k background principal components.

```
search_grid <- 1:10

tuning_bg <- lapply(search_grid, function(xx) occPCA(target = targ[, .SD, .SDcols = -c("Genotype", "Trea
     bg = background[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     bg_components = xx))

tuning_bg_plot <- data.table(do.call(rbind, lapply(tuning_bg, "[[", 1)),
                             sort(rep(search_grid, nrow(targ))),
                             unlist(rep(targ[, "Genotype"],length(search_grid))))
setnames(tuning_bg_plot, c("PCA.1", "PCA.2", "alpha", "Down.Syndrome"))

ggplot(data = tuning_bg_plot)+
  geom_point(aes(x = PCA.1, y = PCA.2, color = Down.Syndrome), alpha = 0.6)+
  facet_wrap(~alpha, scales = "free", nrow = 2)+
  theme(legend.position = "bottom")+
  labs(title = "Mouse Down Syndrome Data: occPCA with different Background Components")
```


Mouse Down Syndrome Data: occPCA with different Background Compon
1
2
3
4
5
6
7
8
9
10
PCA.2
PCA.1
Down.Syndrome
Control
Ts65Dn