cPCA

Robin

9/4/2019

Implementing cPCA: contrastive PCA

(Author's written function in Python)["https://github.com/rtud2/contrastive/blob/master/contrastive/___init___.py"]

```
import numpy as np
from numpy import linalg as LA

def cpca_alpha(self, n_components = 2, alpha=1, return_eigenvectors=False):
        #return None, self.cpca.cpca_alpha(dataset=self.active,alpha=alpha), None
        sigma = self.active_cov - alpha*self.bg_cov
        w, v = LA.eig(sigma)
        eig_idx = np.argpartition(w, -n_components)[-n_components:]
        eig_idx = eig_idx[np.argsort(-w[eig_idx])]
        v_top = v[:,eig_idx]
        reduced_foreground = self.active.dot(v_top)
        reduced_background = self.bg.dot(v_top)

        reduced_foreground[:,0] = reduced_foreground[:,0]*np.sign(reduced_foreground[0,0])
        reduced_foreground[:,1] = reduced_foreground[:,1]*np.sign(reduced_foreground[0,1])


        if (return_eigenvectors):
            #return eig_idx, reduced_foreground, reduced_background, v_top
            print("WHat?")
            return None
        else:
            #return eig_idx, reduced_foreground, reduced_background
            return None, reduced_foreground, None
```

cPCA function:

- inputs:
  - target: Target dataset
  - bg: Background dataset
  - n_components: number of Principal components to use
  - alpha: tuning parameter for how hard to subtract the background data
  - return_eigenvectors: (logical) whether eigenvectors should be returned # Not implemented yet
- output:
  - Data projected on the contrastive principal components

Here are some helper functions to standardize, clear out zeros, etc.

```
## Center variables
center = function(mat){
  return(mat - colMeans(mat, na.rm = T))
  }

## Standardize variables
```

```
standardized = function(mat){
  return(center(mat)/apply(mat,2, sd))
}

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

Some crude tests to make sure my helper functions work as they should

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

# make sure values are close to zero
round(colMeans(center(test_mat)))
```

```
## [1] 0 0 0 0
```

```
# make sure near 1
apply(standardized(test_mat), 2, sd)
```

```
## [1] 1.0094324 1.0075943 0.9797486 1.0048733
```

cPCA and PCA functions I quickly wrote myself:

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

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

  if(standardize){
    transformed_target = standardized(target);
    transform_bg = standardized(bg);
  }else{
    transformed_target = center(target);
    transformed_bg = center(bg)
  }

  target_cov = cov(target);
  bg_cov = crossprod(transformed_bg);

  sigma = target_cov - alpha * bg_cov

  eigen_decomp <- eigen(sigma)
  v_top <- eigen_decomp$vectors[, 1:n_components]
  reduced_target <- target %*% v_top
  reduced_bg <- bg %*% v_top

  return(list("reduced_target" = reduced_target, "reduced_bg" = reduced_bg))
}
```

```
PCA = function(mat, n_components = 2, standardize = F, return_eigenvectors = F){
    if(!is.matrix(mat)){
    mat <- as.matrix(mat)
    }
  if(standardize){
    transform_mat = standardized(mat);
  }else{
    transformed_mat = center(mat);
  }

  v_top <- eigen(cov(transformed_mat))$vectors[, 1:n_components]
  reduced_mat <- mat %*% v_top
  return(reduced_mat)
}
```

Authors Data Analysis

Replicating Figure 3a. in the paper

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
```

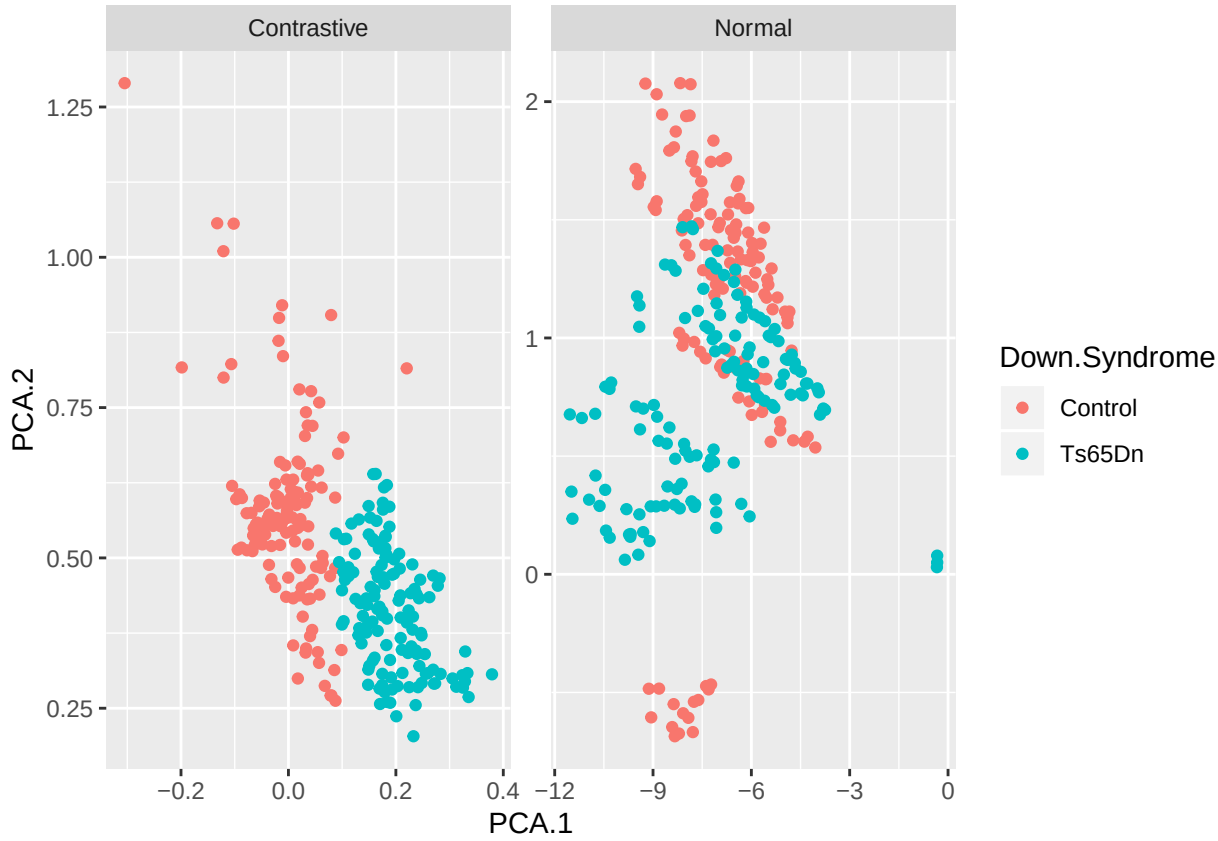
```
results_cPCA <- cPCA(target = targ[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     bg = background[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")],
     alpha = .9)

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

normal_pca <- data.table(PCA(mat = targ[, .SD, .SDcols = -c("Genotype", "Treatment", "Behavior")]))
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")
```



Playing with the tuning parameter alpha

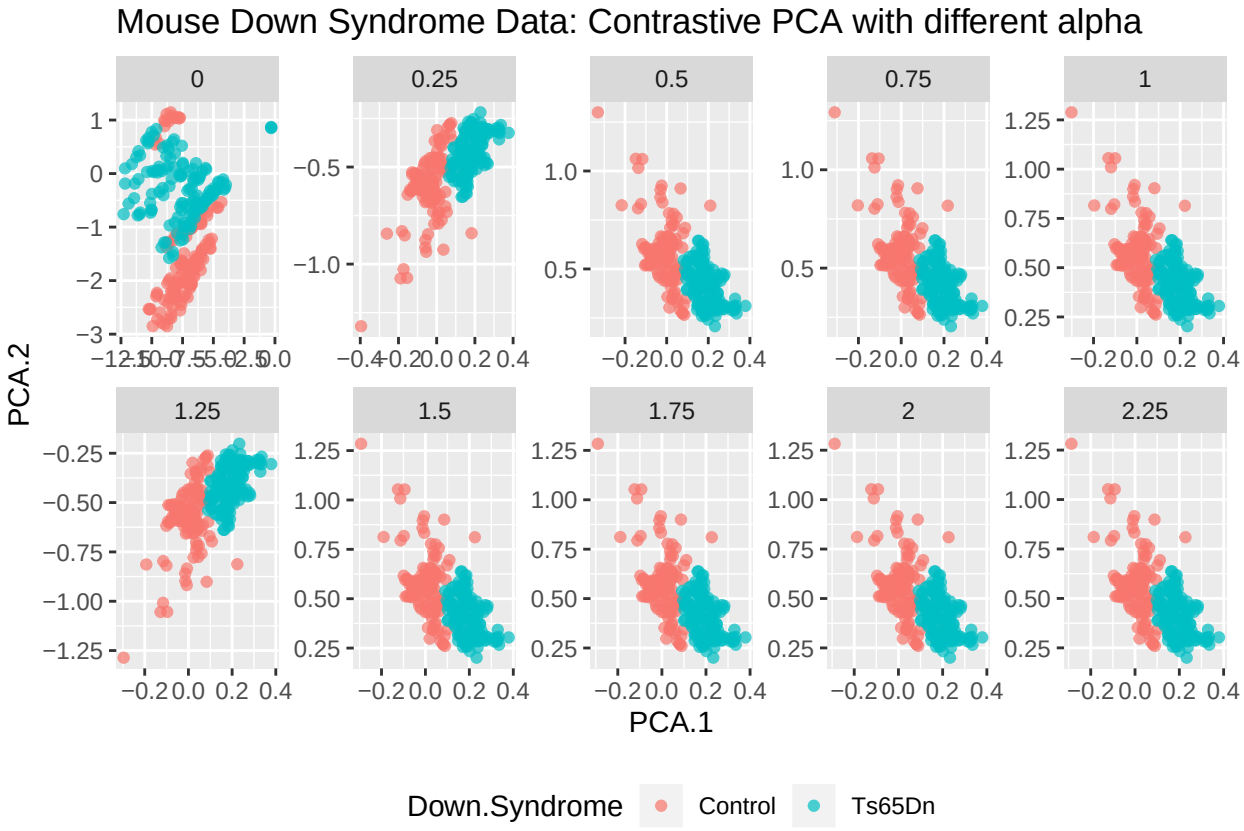
Contrastive PCA seems very sensitive to the tuning parameter α

```
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")
```


Mouse Down Syndrome Data: Contrastive PCA with different alpha
0
0.25
0.5
0.75
1
1.25
1.5
1.75
2
2.25
PCA.2
PCA.1
Down.Syndrome
Control
Ts65Dn