t_sne

Luis Angel Rodriguez Garcia

3/5/2022

Introduction

We are going to play with the Iris dataset:

```
X <- iris %>% dplyr::select(-Species) %>% as.matrix()
n <- nrow(X)
p <- ncol(X)
```

Euclidean distance

The way to calculate the Euclidean distance:

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i'\mathbf{x}_j$$

where $\|\mathbf{x}_i\| = \sqrt{x_{i,1}^2 + \cdots + x_{i,p}^2}$ and $\mathbf{x}_k^2 = \mathbf{x}_k'\mathbf{x}_k = x_{k1}^2 + ... + x_{kp}^2$.

The following method apply the formula above:

```
x_diff <- function(X) {
  n <- nrow(X)
  sum_x <- apply(X^2, MARGIN=1, FUN=sum)
  sum_x_m <- t(matrix(replicate(n, sum_x), byrow=T, nrow=n))
  cross_times_minus_2 <- -2 * (X %*% t(X))
  D <- t(cross_times_minus_2 + sum_x_m) + sum_x_m
  D <- round(D, digits=4)
}
```

Perplexity

$$Perp_i = 2^{H_i}$$

Where H_i is the Shannon entropy in the point x_i of the conditional probability:

$$\begin{aligned}
H_i &= -\sum_{j\neq i} p_{j|i}\log(p_{j|i}) \\
&= -\sum_{j\neq i} p_{j|i}\log(\frac{p_{j,i}}{p_i}) \\
&= -\sum_{j\neq i} p_{j|i}(\log(p_{j,i}) - \log(p_i)) \\
&= -\sum_{j\neq i} p_{j|i}(\log(e^{-||x_i-x_j||^2/2\sigma^2}) - \log(\sum_{k\neq i} e^{-||x_i-x_j||^2/2\sigma^2})) \\
&= -\sum_{j\neq i} p_{j|i}((-||x_i-x_j||^2/2\sigma^2) - \log(\sum_{k\neq i} e^{-||x_i-x_j||^2/2\sigma^2})) \\
&= \sum_{j\neq i} p_{j|i}(\log(S_i) + ||x_i-x_j||^2\frac{1}{2\sigma^2}) \\
&= \log(S_i)\sum_{j\neq i} p_{j|i} + \frac{1}{2\sigma^2}\sum_{j\neq i} p_{j|i}||x_i-x_j||^2) \\
&= \log(S_i) + \frac{1}{2\sigma^2}\sum_{j\neq i} p_{j|i}||x_i-x_j||^2
\end{aligned} \tag{1}$$

Where $S_i = \sum_{k\neq i} e^{-||x_i-x_j||^2/2\sigma^2}$ and $\sum_{j\neq i} p_{j|i} = 1$

In order to proceed with the optimization of the variance, we are going to define this term $\frac{1}{2\sigma^2}$ as the parameter β.

```
entropy_beta <- function(D_i, beta=1) {
  P_i <- exp(-D_i * beta)
  sum_p_i <- sum(P_i)
  H_i <- log(sum_p_i) + (beta * sum(D_i * P_i) /sum_p_i)
  P_i <- P_i / sum_p_i
  return(list(entropy=H_i, probs=P_i))
}
```

The goal is to adjust the variability so that the perplexity at each point is the same. The perplexity is a way to measure the effective number of neighbors of a point. We are going to perform a binary search to get the probabilities in such a way that the conditional Gaussian has the same perplexity.

```
index_except_i <- function(i, n) {
  index <- c(seq(1,i-1),seq(i+1,n))
  if (i == 1) {
    index <- 2:n
  } else if (i == n) {
    index <- 1:(n-1)
  }
  return(index)
}

binary_search <- function(h_diff, beta, i, beta_min, beta_max) {
  if(h_diff > 0) {
    beta_min = beta[i]
    if(beta_max == -Inf || beta_max == Inf) {
      beta[i] <- beta[i] * 2
    } else {
      beta[i] <- (beta[i] + beta_max) / 2
    }
  } else {
    beta_max = beta[i]
    if(beta_min == -Inf || beta_min == Inf) {
      beta[i] <- beta[i] / 2
    } else {
      beta[i] <- (beta[i] + beta_min) / 2
    }
  }
  return(list(beta=beta, min=beta_min, max=beta_max))
}

binary_search_optimization <- function(D_i, i, beta, h_star, prob_star, log_perp,
                                       tolerance=1e-5) {
  beta_min <- -Inf
  beta_max <- Inf
  tries <- 0
  h_diff <- h_star - log_perp

  while(abs(h_diff) > tolerance && tries < 50) {
    beta_opt <- binary_search(h_diff, beta, i, beta_min, beta_max)
    beta <- beta_opt$beta; beta_min <- beta_opt$min; beta_max <- beta_opt$max

    res_loop <- entropy_beta(D_i, beta[i])
    h_star <- res_loop$entropy; prob_star <- res_loop$probs

    h_diff <- h_star - log_perp
    tries <- tries + 1
  }
  return(list(probs=prob_star, beta=beta))
}
```

Once we have defined these two methods, we are able to obtain the high dimensional properties:

```
high_dimension_probs <- function(X=matrix(), tolerance=1e-5, perplexity=30) {
  n <- nrow(X)
  p <- ncol(X)

  D <- x_diff(X)

  P <- matrix(0, nrow=150, ncol=150)
  beta <- rep(1, n)
  log_perp <- log(perplexity)

  for(i in seq_len(n)) {
    column_index <- index_except_i(i, n)
    D_i <- D[i, column_index]

    res <- entropy_beta(D_i, beta[i])
    h_star <- res$entropy
    prob_star <- res$probs

    h_diff <- h_star - log_perp
    res_opt = binary_search_optimization(D_i, i, beta, h_star, prob_star,
                                           log_perp, tolerance)
    prob_star = res_opt$probs; beta = res_opt$beta

    P[i, column_index] <- prob_star
  }
  print("Values of sigma for each x_i")
  print(sqrt(1/beta))
  print(sprintf("Mean value of sigma: %01.2f", mean(sqrt(1/beta))))
  return(P)
}
```

The version of the t-SNE is the symmetric one, that has the property that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji} \quad \forall i, j$. Therefore, we define $p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$.

```
symmetric_probs <- function(P) {
  P = (P + t(P)) / (2*nrow(P))
  return(P)
}
```

In order to initialize the lower dimension probability matrix, we are going to use the method `mvtnorm::rmvnorm` as it is described in the paper: $\mathcal{Y}^0 = \{y_1, ..., y_n\} \sim \mathcal{N}(0, 10^{-4}\mathbf{I}_n)$ which is assigned to $\mathcal{Y}^1$ and $\mathcal{Y}^2$ (the first two initial states).

Gradient Descent

$$\begin{aligned} C &= KL(\mathbf{P} \| \mathbf{Q}) \\ &= \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \\ &= \sum_i \sum_j p_{ij} \left(\log p_{ij} - \log q_{ij}\right) \\ &= \sum_i \sum_j p_{ij} \log p_{ij} - p_{ij} \log q_{ij} \end{aligned} \tag{2}$$

We define these two auxiliary variables $d_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$ and $Z = \sum_{k \neq l} \left(1 + d_{kl}^2\right)^{-1}$

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{y}_i} &= \sum_{j \neq i} \left[\frac{\partial C}{\partial d_{ij}} \frac{\partial d_{ij}}{\partial \mathbf{y}_i} + \frac{\partial C}{\partial d_{ji}} \frac{\partial d_{ji}}{\partial \mathbf{y}_i} \right] \\ &= \sum_{j \neq i} \left[\frac{\partial d_{ij}}{\partial \mathbf{y}_i} \left(\frac{\partial C}{\partial d_{ij}} + \frac{\partial C}{\partial d_{ji}} \right) \right] \end{aligned} \tag{3}$$

Recall that $\frac{\partial}{\partial x} \sqrt{g(x)} = \frac{1}{2\sqrt{g(x)}} g'(x)$, $\frac{\partial}{\partial \mathbf{y}} \|\mathbf{y}\|^2 = 2\mathbf{y}$ and $d_{ij} = d_{ji}$

$$
\begin{aligned}
\frac{\partial d_{ij}}{\partial \mathbf{y}_i} &= \frac{\partial}{\partial y_i}\|\mathbf{y}_i - \mathbf{y}_j\| \\
&= \frac{\partial}{\partial \mathbf{y}_i}(\|\mathbf{y}_i\|^2 + \|\mathbf{y}_j\|^2 - 2\mathbf{y}_i'\mathbf{y}_j)^{\frac{1}{2}} \\
&= \frac{1}{2}\frac{1}{d_{ij}}\frac{\partial}{\partial \mathbf{y}_i}(\|\mathbf{y}_i\|^2 + \cancel{\|\mathbf{y}_j\|^2} - 2\mathbf{y}_i'\mathbf{y}_j) \\
&= \frac{1}{2}\frac{1}{d_{ij}}(2\mathbf{y}_i - 2\mathbf{y}_j) \\
&= \frac{(\mathbf{y}_i - \mathbf{y}_j)}{d_{ij}} \\
&= \frac{\partial d_{ji}}{\partial \mathbf{y}_i}
\end{aligned}
\tag{4}
$$

```r
dij.1 <- function(i, j) {
  sqrt(norm(i, type="2")^2 + norm(j, type="2")^2 - 2 * (t(i) %*% j))
}

dij.2 <- function(i, j) {
  norm(i-j, type="2")
}

y <- data.matrix(iris)[, -5]

# For rows 2 and 3
result1 <- as.numeric(round(jacobian(func=dij.2, x=y[2,], j=y[3,]), digits=7))
result2 <- as.numeric((y[2,]-y[3,])/norm(y[2,]-y[3,], type="2"))
all.equal(result2, result1) # checked
```

```
## [1] "Mean relative difference: 6e-08"
```

```r
# For row 3 and 2
result3 <- as.numeric(round(jacobian(func=dij.2, x=y[3,], i=y[2,]), digits=7))
result4 <- as.numeric((y[3,]-y[2,])/norm(y[2,]-y[3,], type="2"))
all.equal(result4, result3) # checked
```

```
## [1] "Mean relative difference: 6e-08"
```

```r
# Checked that both d(d_ij)/dy_i = d(d_ji)/dy_i
```

Recall that $d_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$ and $Z = \sum_{k \neq l}\left(1 + d_{kl}^2\right)^{-1}\big)$:

$$
\begin{aligned}
\frac{\partial C}{\partial d_{ij}} &= \frac{\partial}{\partial d_{ij}} \sum_{k\neq l} p_{kl} \log p_{kl} - p_{kl} \log q_{kl} \\
&= \frac{\partial}{\partial d_{ij}} \sum_{k\neq l} -p_{kl} \log q_{kl} = -\sum_{k\neq l} p_{kl} \frac{\partial(\log q_{kl})}{\partial d_{ij}} \\
&= -\sum_{k\neq l} p_{kl} \frac{\partial(\log \frac{q_{kl}Z}{Z})}{\partial d_{ij}} \\
&= -\sum_{k\neq l} p_{kl} \left[\frac{\partial(\log q_{kl}Z - \log Z)}{\partial d_{ij}} \right] \\
&= -\sum_{k\neq l} p_{kl} \left[\frac{\partial(\log q_{kl}Z)}{\partial d_{ij}} - \frac{\partial \log Z}{\partial d_{ij}} \right] \\
&= -\sum_{k\neq l} p_{kl} \left[\frac{1}{q_{kl}Z} \frac{\partial(q_{kl}Z)}{\partial d_{ij}} - \frac{1}{Z} \frac{\partial Z}{\partial d_{ij}} \right] \\
&= -\sum_{k\neq l} p_{kl} \left[\frac{1}{q_{kl}Z} \frac{\partial(\frac{(1+d_{ij}^2)^{-1}}{\cancel{\sum_{k\neq l}(1+d_{kl}^2)^{-1}}} \cancel{\sum_{k\neq l}(1+d_{kl}^2)^{-1}})}{\partial d_{ij}} - \frac{1}{Z} \frac{\partial(\sum_{k\neq l}(1+d_{kl}^2)^{-1})}{\partial d_{ij}} \right] \\
&= 2\frac{p_{ij}}{q_{ij}Z}(1+d_{ij}^2)^{-2} d_{ij} - 2\sum_{k\neq l} p_{kl} \frac{(1+d_{ij}^2)^{-1}}{Z}(1+d_{ij}^2)^{-1} d_{ij} \\
&= 2\frac{p_{ij}}{\frac{\cancel{(1+d_{ij}^2)^{-1}}}{\cancel{Z}}\cancel{Z}}(1+d_{ij}^2)^{-\cancel{2}1} d_{ij} - 2\cancel{\sum_{k\neq l} p_{kl}}\, q_{ij}(1+d_{ij}^2)^{-1} d_{ij} \\
&= 2p_{ij}(1+d_{ij}^2)^{-1} d_{ij} - 2q_{ij}(1+d_{ij}^2)^{-1} d_{ij} \\
&= 2(p_{ij} - q_{ij})(1+d_{ij}^2)^{-1} d_{ij}
\end{aligned}
\tag{5}
$$

$$
\begin{aligned}
\frac{\partial C}{\partial y_i} &= \sum_{j\neq i} \left[\frac{\partial C}{\partial d_{ij}} \frac{\partial d_{ij}}{\partial y_i} + \frac{\partial C}{\partial d_{ji}} \frac{\partial d_{ji}}{\partial y_i} \right] \\
&= \sum_{j\neq i} \left[\frac{\partial d_{ij}}{\partial y_i} \left(\frac{\partial C}{\partial d_{ij}} + \frac{\partial C}{\partial d_{ij}} \right) \right] \\
&= 2\sum_{j\neq i} \left[\frac{\partial C}{\partial d_{ij}} \right] \frac{\mathbf{y}_i - \mathbf{y}_j}{d_{ij}} \\
&= 2\sum_{j\neq i} \left[2(p_{ij} - q_{ij})(1+d_{ij}^2)^{-1} d_{ij} \right] \frac{\mathbf{y}_i - \mathbf{y}_j}{d_{ij}} \\
&= 4\sum_{j\neq i} (p_{ij} - q_{ij})(1+d_{ij}^2)^{-1} \cancel{d_{ij}} \frac{\mathbf{y}_i - \mathbf{y}_j}{\cancel{d_{ij}}} \\
&= 4\sum_{j\neq i} (p_{ij} - q_{ij})(1+\|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}(\mathbf{y}_i - \mathbf{y}_j)
\end{aligned}
\tag{6}
$$

Cauchy distribution on the sphere

The following formula corresponds with the probability density function of the Cauchy family on the unit sphere:

$$f(\mathbf{y};\boldsymbol{\mu},\rho) = \frac{\Gamma\{(d+1)/2\}}{2\pi^{(d+1)/2}}\left(\frac{1-\rho^2}{1+\rho^2-2\rho\boldsymbol{\mu}'\mathbf{y}}\right)^d$$

where $\mathbf{y} \in S^d$, the location parameter $\boldsymbol{\mu} \in S^d$, the concentration parameter $\rho \in [0,1)$ and the unit sphere in $\mathbb{R}^{d+1}$ denoted by $S^d = \{\mathbf{y} \in \mathbb{R}^{d+1}; \|\mathbf{y}\| = 1\}$. When $d = 1$ the case is the well-known Wrapped Cauchy or circular Cauchy family.

```
dsphcauchy <- function(y, mu, rho, d=2) {
  (gamma(((d+1)/2))/(2*pi^((d+1)/2))*((1-rho^2)/(1+rho^2-2*rho*(t(mu) %*% y)))^d)
}

library(Directional)
library(Rcpp)
library(rotasym)

sunspots_births$X <-
  cbind(cos(sunspots_births$phi) * cos(sunspots_births$theta),
        cos(sunspots_births$phi) * sin(sunspots_births$theta),
        sin(sunspots_births$phi))

theta_params <- spcauchy.mle(sunspots_births$X)
dsphcauchy(sunspots_births$X[1,], theta_params$mu, theta_params$rho)
```

```
##            [,1]
## [1,] 0.07957747
```

```
library(DirStats)
library(ivdoctr)
n <- nrow(sunspots_births$X)
q <- 2
x <- rbind(diag(1, nrow = q + 1), diag(-1, nrow = q + 1))
bw_rot <- bw_dir_rot(sunspots_births$X)

polysphere <- array(NA, dim=c(100,3, 120))
for(i in seq_len(120)) {
  th <- sample(seq(0, pi/2, l=200), size=100)
  ph <- sample(seq(0, 2*pi, l=200), size=100)
  polysphere[,,i] <- to_sph(th, ph)
}

rgl::plot3d(0, 0, 0, xlim = c(-1, 1), ylim = c(-1, 1), zlim = c(-1, 1),
            radius = 1, type = "s", col = "lightblue", alpha = 0.25,
            lit = FALSE)
# dens <- apply(x, MARGIN=1, FUN=dsphcauchy, mu=theta_params$mu, rho=theta_params$rho)
dens <- apply(x, MARGIN=1, FUN=kde_dir, data = sunspots_births$X, h = bw_rot, L = NULL)
map2color <- function(x, pal, limits = range(x)){
  pal[findInterval(x, seq(limits[1], limits[2], length.out = length(pal) + 1),
                   all.inside=TRUE)]
}
rgl::points3d(x, col = map2color(dens, pal=heat.colors(10, alpha=0.8)))
```

Cauchy-SNE

High Dimension For a poly-sphere $\mathbb{S}^{d,r}$ where $r \geq 1$ and $d \geq 1$:

$$
\begin{aligned}
p_{j|i} &= \frac{p_{ji}}{p_i} \\
&= \frac{\prod_{k=1}^{r} \frac{\Gamma\{(d+1)/2\}}{2\pi^{(d+1)/2}} \left(\frac{1-\rho_k^2}{1+\rho_k^2-2\rho_k \mathbf{x}'_{j_{(k)}} \mathbf{x}_{i_{(k)}}} \right)^d}{\sum_{\ell \neq i} \left[\prod_{s=1}^{r} \frac{\Gamma\{(d+1)/2\}}{2\pi^{(d+1)/2}} \left(\frac{1-\rho_s^2}{1+\rho_s^2-2\rho_s \mathbf{x}'_{\ell_{(s)}} \mathbf{x}_{i_{(s)}}} \right)^d \right]} \\
&= \frac{\left(\frac{\Gamma\{(d+1)/2\}}{2\pi^{(d+1)/2}} \right)^r \prod_{k=1}^{r} \left(\frac{1-\rho_k^2}{1+\rho_k^2-2\rho_k \mathbf{x}'_{j_{(k)}} \mathbf{x}_{i_{(k)}}} \right)^d}{\left(\frac{\Gamma\{(d+1)/2\}}{2\pi^{(d+1)/2}} \right)^r \sum_{\ell \neq i} \left[\prod_{s=1}^{r} \left(\frac{1-\rho_s^2}{1+\rho_s^2-2\rho_s \mathbf{x}'_{\ell_{(s)}} \mathbf{x}_{i_{(s)}}} \right)^d \right]} \\
&= \frac{\prod_{k=1}^{r} \left(\frac{1-\rho_k^2}{1+\rho_k^2-2\rho_k \mathbf{x}'_{j_{(k)}} \mathbf{x}_{i_{(k)}}} \right)^d}{\sum_{\ell \neq i} \left[\prod_{s=1}^{r} \left(\frac{1-\rho_s^2}{1+\rho_s^2-2\rho_s \mathbf{x}'_{\ell_{(s)}} \mathbf{x}_{i_{(s)}}} \right)^d \right]} \\
&= \frac{\left(\prod_{k=1}^{r} (1-\rho_k^2)^d \right) \prod_{k=1}^{r} (1+\rho_k^2 - 2\rho_k \mathbf{x}'_{j_{(k)}} \mathbf{x}_{i_{(k)}})^{-d}}{\sum_{\ell \neq i} \left[\left(\prod_{s=1}^{r} (1-\rho_s^2)^d \right) \prod_{s=1}^{r} (1+\rho_s^2 - 2\rho_s \mathbf{x}'_{\ell_{(s)}} \mathbf{x}_{i_{(s)}})^{-d} \right]} \\
&= \frac{\prod_{k=1}^{r} (1+\rho_k^2 - 2\rho_k \mathbf{x}'_{j_{(k)}} \mathbf{x}_{i_{(k)}})^{-d}}{\sum_{\ell \neq i} \left[\prod_{s=1}^{r} (1+\rho_s^2 - 2\rho_s \mathbf{x}'_{\ell_{(s)}} \mathbf{x}_{i_{(s)}})^{-d} \right]}
\end{aligned}
\tag{7}
$$

where the cosine similarity is denoted by $S_c(\mathbf{x}_i, \mathbf{x}_j) = \cos(\theta) = \frac{\mathbf{x}_i \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$. In this case $\|\mathbf{x}_i\| = \|\mathbf{x}_j\| = 1$.

We must adapt the configuration of each parameter to have the same perplexity fixed at the beginning, in the same way we had done with the Gaussian case.

$$\text{Perp}_i = 2^{H_i}$$

$$H_i = -\sum_{j \neq i} p_{j|i} \log p_{j|i}$$

```
simple_dspcauchy <- function(x, l, i, rho, k, d) {
  ((1 + rho^2 - 2 * rho * t(x[l,,k]) %*% x[i,,k])^(-d))
}

prob_i_spcauchy <- function(x, i, rho, d){
  r <<- dim(x)[3]
  n <<- nrow(x)
  probi_k_spcauchy <- function(j) {
    prod(sapply(X=seq_len(r), FUN=simple_dspcauchy, x=x, i=i, l=j, rho=rho, d=d))
  }
  sum(sapply(seq_len(n)[-i], probi_k_spcauchy))
}

jcondi_spcauchy <- function(x, i, j, rho, d, prob_is=NULL) {
  r <- dim(x)[3]
  if(!is.null(prob_is) && !is.null(prob_is[i])) {
    prob_i <<- prob_is[i]
  } else {
    prob_i <<- prob_i_spcauchy(x, i, rho, d)
  }
  (prod(sapply(1:r, simple_dspcauchy, x=x, i=i, l=j, rho=rho, d=d)) / prob_i)
}

to_perplexity <- function(X, i, rho, d=2, prob_is=NULL) {
  entropy <- function(j) {
    jcondi_value <- jcondi_spcauchy(X, i, j, rho, d, prob_is)
    (jcondi_value * log2(jcondi_value))
  }
  return(2^(-1*sum(sapply(seq_len(nrow(X))[-i], entropy))))
}
```

Verifying formulas

We are going to check the following partial derivatives with respect a the vector $\mathbf{y}_i$ and the scalar s_{ij}.

Verifying $\frac{\partial s_{ij}}{\partial \mathbf{y}_i}$

$$\frac{\partial s_{ij}}{\partial \mathbf{y}_i} = \frac{\partial}{\partial y_i}\mathbf{y}_i'\mathbf{y}_j = \mathbf{y}_j' = \frac{\partial s_{ji}}{\partial \mathbf{y}_i}$$

```
library(numDeriv)

n <- 25
p <- 3
d <- 2
Y <- r_unif_sphere(n, p)

s.ji <- function(i, j) {
    t(i) %*% j
}

y_j <- jacobian(func=s.ji, x=Y[2,], j=Y[3,])

all.equal(y_j, rbind(Y[3,]))
```

```
## [1] TRUE
```

Verifying of $\frac{\partial C}{\partial s_{ij}}$

$$\frac{\partial C}{\partial s_{ij}} = -\sum_{\substack{k,\ell=1\\ k\neq l}}^{n} p_{k\ell}\left[\frac{\partial}{\partial s_{ij}}\log q_{k\ell}Z - \frac{\partial}{\partial s_{ij}}\log Z\right] = \frac{2d\rho}{1-\rho^2-2\rho s_{ij}}\left(q_{ij}-p_{ij}\right)$$

```
library(numDeriv)
library(rotasym)
source("perplexity.R")
source("prob_distribution.R")
source("high_dimension.R")
source("low_dimension.R")
```

```
## Loading required package: SnowballC
```

```
cos_sim_ij <- function(i, j) {
  (t(i) %*% j)[1]
}

simple_dspcauchy_sim <- function(s_ij, rho, d) {
  ((1 + rho^2 - 2 * rho * s_ij)^(-d))
}

simple_dspcauchy_ld <- function(Y, i, j, rho, d) {
  ((1 + rho^2 - 2 * rho * t(Y[i,]) %*% Y[j,])^(-d))
}

q_i_not_j <- function(Y, i, j, rho, d) {
  n <- nrow(Y)
  qijs <- sapply(1:n, function(k){
    sapply(1:n, function(l) {
      return(simple_dspcauchy_ld(Y, l, k, rho, d))
    })
  }, simplify = 'array')
  diag(qijs) <- 0
  qijs[i, j] <- 0
  sum(qijs)
}

q_ij <- function(Y, s_ij, i, j, rho, d, has_deriv_num) {
  n <- nrow(Y)
  qij <- simple_dspcauchy_ld(Y, i, j, rho, d)
  if(has_deriv_num)
    qij <- simple_dspcauchy_sim(s_ij, rho, d)
  return(qij / Z(Y, s_ij, i, j, rho, d))
}

Z <- function(Y, s_ij, i, j, rho, d) {
  qij <- simple_dspcauchy_sim(s_ij, rho, d)
  q_inotj <- q_i_not_j(Y, i, j, rho, d)
  return(qij+q_inotj)
}

log_q_ij <- function(Y, s_ij, i, j, rho, d, has_deriv_num){
  log(q_ij(Y, s_ij, i, j, rho, d, has_deriv_num))
}

log_q_ij_prod_Z <- function(Y, s_ij, i, j, rho, d, has_deriv_num){
  log(q_ij(Y, s_ij, i, j, rho, d, has_deriv_num)*Z(Y, s_ij, i, j, rho, d))
}

log_Z <- function(Y, s_ij, i, j, rho, d){
  log(Z(Y, s_ij, i, j, rho, d))
}

## check diff/diff(s_ij) * log(q_kl * Z)
library(rotasym)
```

```r
i <- 2
j <- 1
rho <- .5
s_ij <- cos_sim_ij(Y[i, ], Y[j, ])

log_q_ij_prod_Z(Y, s_ij, i, j, rho, d, TRUE)
```

```
## [1] -0.3422469
```

```r
s_ij <- 0
eval(D(expression((1 + rho^2 - 2 * rho * s_ij)^(-d)), 's_ij'))
```

```
## [1] 1.024
```

```r
grad(simple_dspcauchy_sim, x=0, rho=.5, d=2)
```

```
## [1] 1.024
```

```r
s_ij <- cos_sim_ij(Y[i, ], Y[j, ])
q_2_1 <- q_ij(Y, s_ij, i, j, rho, d, TRUE)*Z(Y, s_ij, i, j, rho, d)
sim_dspcauchy_2_1 <- simple_dspcauchy_sim(s_ij, rho, -(-d-1))
num_d_2_rho <- -d*-2*rho
Z_2_1 <- Z(Y, s_ij, i, j, rho, d)


first_term <- grad(func=log_q_ij_prod_Z, x=s_ij, Y=Y, i=i, j=j, rho=rho, d=d, has_deriv_num=TRUE)
first_term_analytically <- (num_d_2_rho/q_2_1*sim_dspcauchy_2_1)
print(round(first_term,8) == round(first_term_analytically, 8))
```

```
## [1] TRUE
```

```r
## check diff/diff(s_ij) * log(Z)

second_term <- grad(func=log_Z, x=s_ij, Y=Y, i=i, j=j, rho=rho, d=d)
second_term_analytically <- (num_d_2_rho/Z_2_1*sim_dspcauchy_2_1)
print(round(second_term, 8) == round(second_term_analytically, 8))
```

```
## [1] TRUE
```

```r
## check diff/diff(s_ij)
diff_Z <- grad(func=Z, x=s_ij, Y=Y, i=i, j=j, rho=rho, d=d)
diff_Z_analytically <- 2*d*rho*sim_dspcauchy_2_1
print(round(diff_Z, 6) == round(diff_Z_analytically, 6))
```

```
## [1] TRUE
```

```r
# some random value for p_kl
r <- 4
polysphere <- gen_polysphere(n, d, r)
perp_value <- 15
rho_list <- rho_optimize(polysphere, perp_value)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Time difference of 5.418934 secs
```

```
P <- high_dimension(polysphere, rho_list)
P <- symmetric_probs(P)

diff_C_wr_sij <- function(Y, i, j, rho, d, P) {
  s_ij <- cos_sim_ij(Y[i, ], Y[j, ])
  n <- nrow(Y)
  result <- -sum(
  sapply(1:n, function(k) {
    sapply(1:n, function(l) {
      if(k!=l) {
        has_deriv_numerator <- FALSE
        if(k == i && l == j) {
          has_deriv_numerator <- TRUE
        }
        return(P[k,l] *
               (grad(func=log_q_ij_prod_Z, x=s_ij, Y=Y, i=k, j=l, rho=rho, d=d, has_deriv_num=has_deriv_
                - grad(func=log_Z, x=s_ij, Y=Y, i=k, j=l, rho=rho, d=d)))
      } else {
        return(0)
      }
    }, simplify = 'array')
  }, simplify = 'array'))
  return(result)
}

result_should_be <- diff_C_wr_sij(Y, i, j, rho, d, P)

## calculated value of the analytical expression

Q <- low_dimension_Q(Y, d, rho)

numerator <- 2*d*rho
denominator <- (1+rho^2-2*rho*s_ij)
difference <- (Q[i,j]-P[i,j])

result <- (numerator/denominator) * (difference)

## with 5 digits of significance, with 6 it fails
print((trunc(result_should_be*1e4)/1e4)==(trunc(result*1e4)/1e4))
```

```
## [1] TRUE
```

Verifying $\frac{\partial C}{\partial \mathbf{y}_i}$

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4d\rho \sum_{\substack{j=1 \\ j \neq i}}^{n} \left[\frac{\mathbf{y}_j}{1+\rho^2-2\rho\mathbf{y}_i'\mathbf{y}_j} (q_{ij}-p_{ij}) \right]$$

```
simple_dspcauchy_yi <- function(Y, yi, j, rho, d) {
  ((1 + rho^2 - 2 * rho * t(yi) %*% Y[j,])^(-d))
}

simple_dspcauchy_yj <- function(Y, yi, i, rho, d) {
  ((1 + rho^2 - 2 * rho * t(Y[i, ]) %*% yi)^(-d))
```

```r
}

Z_yi <- function(Y, rho, d, yi, diff_i) {
  n <- nrow(Y)
  qijs <- sapply(1:n, function(k){
    sapply(1:n, function(l) {
      if(k==l) {
        return(0)
      } else if(k == diff_i) {
        return(simple_dspcauchy_yi(Y, yi, l, rho, d))
      } else if(l == diff_i) {
        return(simple_dspcauchy_yj(Y, yi, k, rho, d))
      } else {
        return(simple_dspcauchy_ld(Y, k, l, rho, d))
      }
    })
  }, simplify = 'array')
  return(sum(qijs))
}

q_ij_yi <- function(Y, yi, i, j, rho, d, diff_i) {
  qij <- NULL
  if(diff_i == i) {
    qij <- simple_dspcauchy_yi(Y, yi, j, rho, d)
  } else if(diff_i == j) {
    qij <- simple_dspcauchy_yj(Y, yi, i, rho, d)
  } else {
    qij <- simple_dspcauchy_ld(Y, i, j, rho, d)
  }
  return(qij / Z_yi(Y, rho, d, yi, diff_i))
}

kl_cost_i <- function(Y, P, rho, d, yi, ii) {
  n <- nrow(Y)
  total <- sum(sapply(1:n, function(i){
    sapply((1:n), function(j) {
      if(i==j) {
        return(0)
      } else {
        (P[i,j]*log(P[i,j])-P[i,j] * log(q_ij_yi(Y, yi, i, j, rho, d, ii)))
      }
    })
  }))
  return(total)
}

kl_cost_analytic <- function(Y, i,  rho, d, P, Q) {
  (4*d*rho*colSums(t(sapply((1:nrow(Y))[-i], function(j) {
    s_ij <- cos_sim_ij(Y[i,], Y[j,])
    diff_Q_P <- (Q[i,j]-P[i,j])
    div <- (1+rho^2-2*rho*s_ij)
    return(Y[j,]/div*diff_Q_P)
  }))))
```

```
}

ii <- 1
yi <- Y[ii,]
d <- 2
rho <- 0.5
cost_result_expected <- jacobian(kl_cost_i, Y=Y, P=P, rho=rho, d=d, x=yi, i=ii)

cost_result <- kl_cost_analytic(Y, ii, rho, d, P, Q)

print(all.equal(cost_result_expected[1,], cost_result))
```

```
## [1] TRUE
```