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()
```

Euclidean distance

The way to calculate the Euclidean distance:

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

$$\mathbf{x}_k^2 = \mathbf{x}_k'\mathbf{x}_k = x_{k1}^2 + ... + x_{kp}^2$$

The following method apply the formula above:

```
x_diff <- function(X) {
  sum_x <- apply(X^2, MARGIN=1, FUN=sum)
  sum_x_m <- t(matrix(replicate(150, sum_x), byrow=T, nrow=150))
  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:

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

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.

```r
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.

```r
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)
}
```