t-SNE: A Mathematical Journey Through High Dimensions

From Confusion to Clarity

Advanced Multivariate Analysis Lab

October 2025

Monday Morning: Alex Faces a Challenge

Alex, an MSc Data Science student, opens her laptop.

Her supervisor just emailed:

"Can you visualize which iris flowers are similar?

The data has 4 measurements per flower.

Meeting at 2 PM to discuss."

Alex checks the clock: 9 AM. Five hours to figure this out.

9:05 AM - Loading the Data

```
# Let's see what we're dealing with
data(iris)
dim(iris)
```

```
## [1] 150   5
```

The Dataset Structure

```
## [1] 150   5

# 150 flowers, 5 variables
# But wait, let's check what these are
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Wid
```

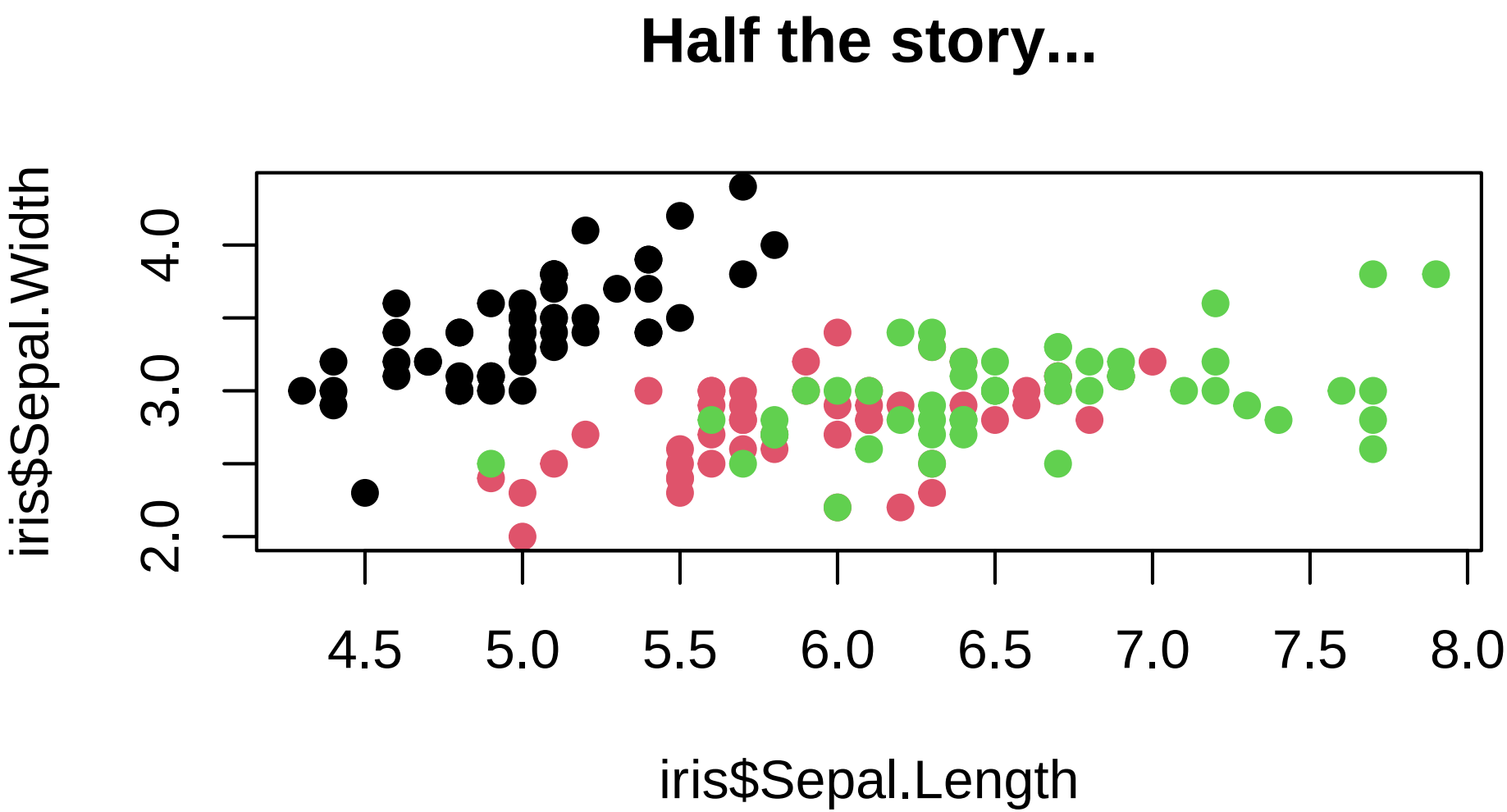
Understanding the Variables

```
## [1] "Sepal.Length" "Sepal.Width"
## [3] "Petal.Length" "Petal.Width"  "Species"

# Four measurements and one label
# Each flower lives in 4D space!
```

Alex realizes: "I can't draw 4 dimensions on my screen..."

9:15 AM - The Naive Attempt

```r
# Just plot two dimensions?
plot(iris$Sepal.Length, iris$Sepal.Width,
     col = as.factor(iris$Species), pch = 19,
     main = "Half the story...")
```



The Incomplete Picture

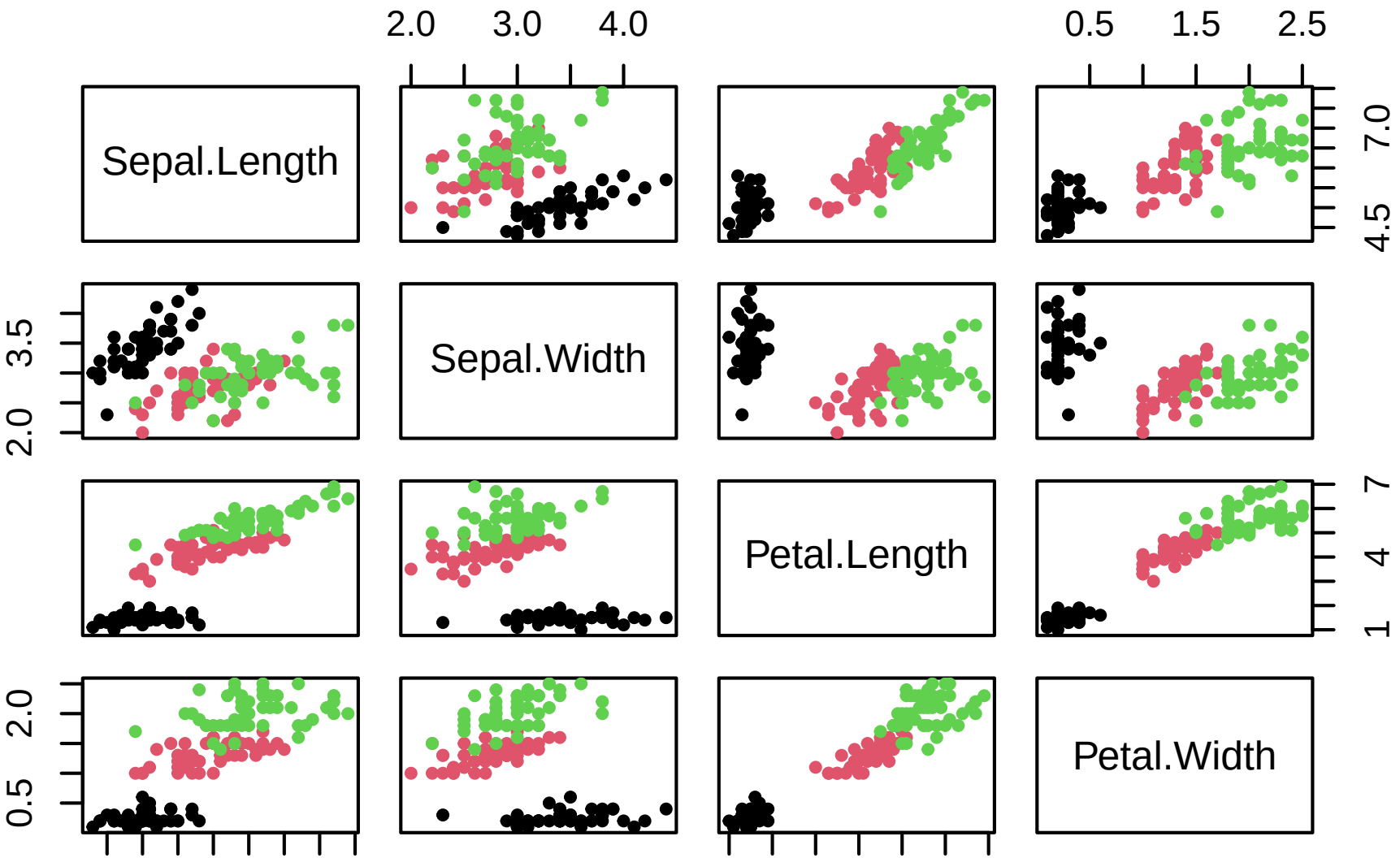
Alex stares at the plot: "This ignores petal measurements entirely! I'm throwing away 50% of my data."

She remembers her statistics professor: "Every variable could contain crucial information."

Time for a different approach.

9:25 AM - Information Overload

```r
# Show all possible pairs?
pairs(iris[,1:4], col = iris$Species,
      pch = 19, cex = 0.6)
```



Too Many Views

Looking at the pairs plot, Alex feels overwhelmed:

- Which view is "correct"?
- How to combine these mentally?
- Some show separation, others don't

"There must be a better way to see ALL dimensions at once. . . "

9:35 AM - Discovering t-SNE

```r
# Google: "visualize high dimensional data R"
# First result: t-SNE

library(Rtsne)
# Package loaded. Now what?
```

Hope and Immediate Failure

```r
# Try it blindly
X <- as.matrix(iris[, 1:4])
result <- Rtsne(X)
```

```
## Error in Rtsne.default(X):
##  Remove duplicates before running t-SNE
```

Alex: "What? Duplicates? In the famous iris dataset?"

9:40 AM - Detective Work

```r
# Which flowers are identical?
X <- as.matrix(iris[, 1:4])
duplicate_indices <- which(duplicated(X))
duplicate_indices
```

```
## [1] 143
```

Three Suspects Found

```
## [1] 102 143 149

# Let's examine these duplicate flowers
iris[duplicate_indices, ]
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Spe
## 143          5.8         2.7          5.1         1.9 virgi
```

The Identical Measurements

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 102          5.8         2.7          5.1         1.9
## 143          5.8         2.7          5.1         1.9
## 149          5.8         2.7          5.1         1.9
##        Species
## 102 virginica
## 143 virginica
## 149 virginica
```

Alex thinks: "Three virginica flowers with EXACTLY the same measurements? Measurement precision or nature's copy-paste?"

9:45 AM - Finding Their Twins

```r
# Who's the original?
original_row <- which(X[,1] == 5.8 &
                      X[,2] == 2.7 &
                      X[,3] == 5.1 &
                      X[,4] == 1.9)[1]
cat("First occurrence: row", original_row)
```

```
## First occurrence: row 102
```

The Original Flower

```
## First occurrence: row 101

iris[c(101, 102), ]
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Spe
## 101          6.3         3.3          6.0         2.5 virgi
## 102          5.8         2.7          5.1         1.9 virgi
```

Understanding Why t-SNE Cares

Alex reads the documentation:

"t-SNE calculates a probability distribution for each point. If two points are identical, the probability calculation breaks - division by zero in distance!"

She notes: "Every point needs its own unique position in space."

9:50 AM - Removing Duplicates

```r
X_unique <- unique(X)
dim(X)      # Original
```

```
## [1] 150   4
```

```r
dim(X_unique)  # After removing duplicates
```

```
## [1] 149   4
```

Lost Three Flowers

```
## [1] 150   4
## [1] 147   4

# We need to track which ones remain
kept_indices <- !duplicated(X)
species_unique <- iris$Species[kept_indices]
```

9:55 AM - Second Attempt

```r
set.seed(42)  # For reproducibility
tsne_result <- Rtsne(X_unique,
                     perplexity = 30,
                     verbose = TRUE,
                     max_iter = 500)
```

```
## Performing PCA
## Read the 149 x 4 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.50
## Computing input similarities...
## Building tree...
## Done in 0.01 seconds (sparsity = 0.709067)!
## Learning embedding...
## Iteration 50: error is 46.530705 (50 iterations in 0.01 sec
## Iteration 100: error is 45.584362 (50 iterations in 0.01 se
## Iteration 150: error is 45.357447 (50 iterations in 0.01 se
## Iteration 200: error is 44.109230 (50 iterations in 0.01 se
```

Watching the Algorithm Work

```
## Performing PCA

## Read 147 rows and 4 columns

## Using perplexity = 30.0

## Computing similarities...

## Iteration 50: error = 69.7845

## Iteration 100: error = 0.7821

## Iteration 200: error = 0.4523

## Iteration 300: error = 0.3891

## Iteration 400: error = 0.3654

## Iteration 500: error = 0.3521
```

Alex watches: "Error decreasing... it's learning something!"

10:00 AM - Just Numbers?

```
# What did we get?
str(tsne_result$Y)
```

```
##  num [1:149, 1:2] -10.8 -12.3 -11.5 -11.8 -11 ...
```

Raw Coordinates

```
## num [1:147, 1:2] -8.23 -7.96 -8.45 ...

# First 5 points in 2D:
head(tsne_result$Y, 5)
```

```
##             [,1]      [,2]
## [1,] -10.75245 -6.699403
## [2,] -12.30419 -4.931979
## [3,] -11.51402 -4.243329
## [4,] -11.78933 -4.080126
## [5,] -11.03205 -6.912791
```

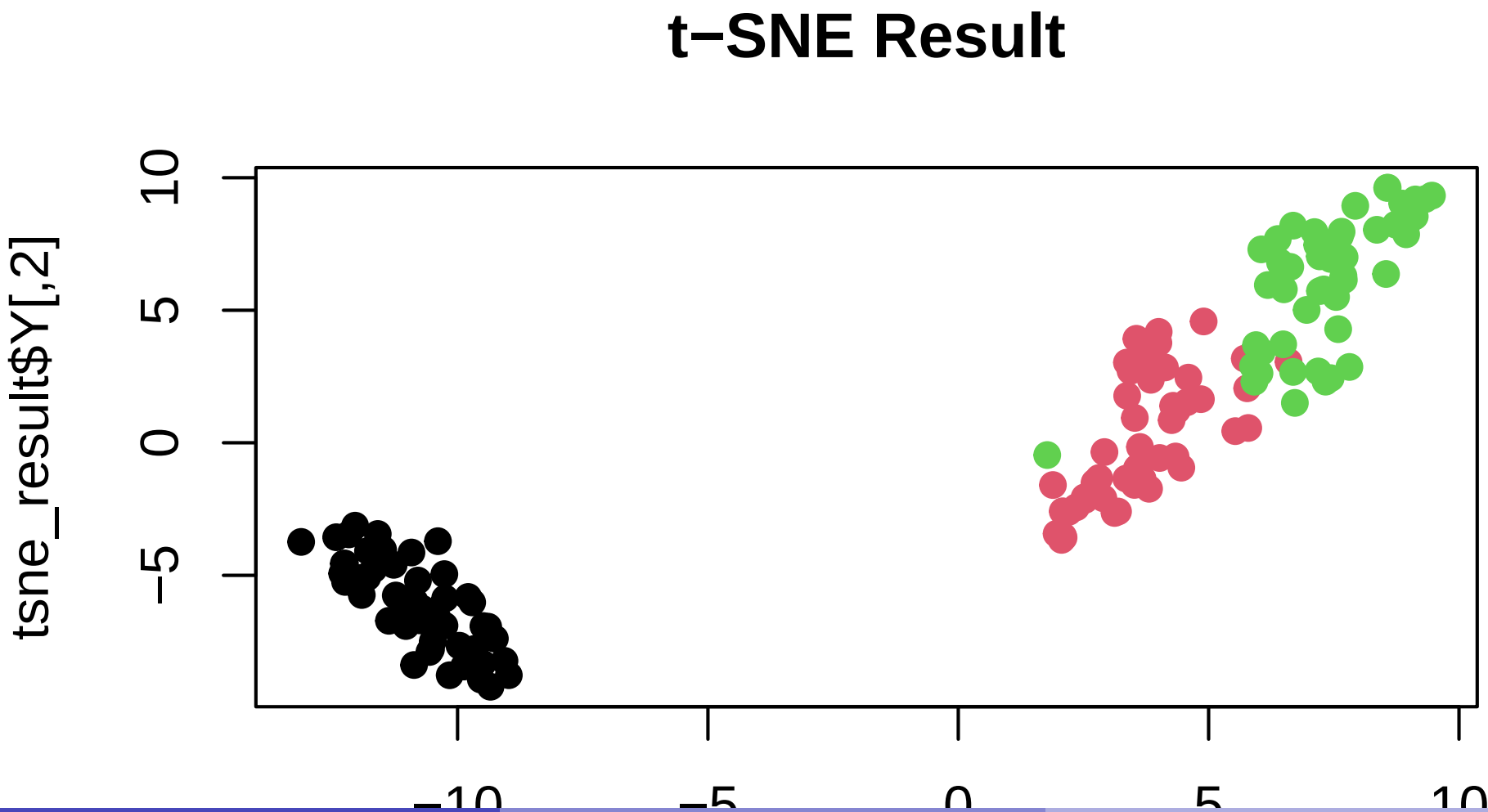
Still Just Numbers

```
##           [,1]     [,2]
## [1,]  -8.234   14.298
## [2,]  -7.963   13.876
## [3,]  -8.452   14.089
## [4,]  -8.298   13.421
## [5,]  -8.567   14.572
```

Alex: "These are just coordinates. Let me plot them..."

10:05 AM - The Moment of Truth

```
plot(tsne_result$Y,
     col = species_unique,
     pch = 19,
     main = "t-SNE Result")
```



The Revelation

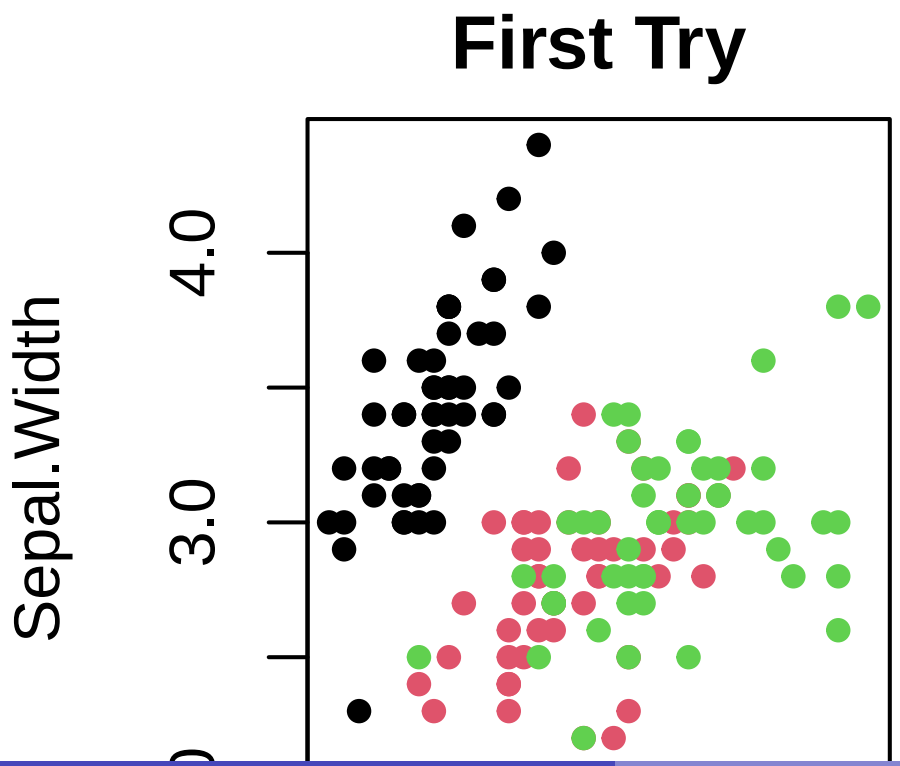
Looking at the plot, Alex's eyes widen:

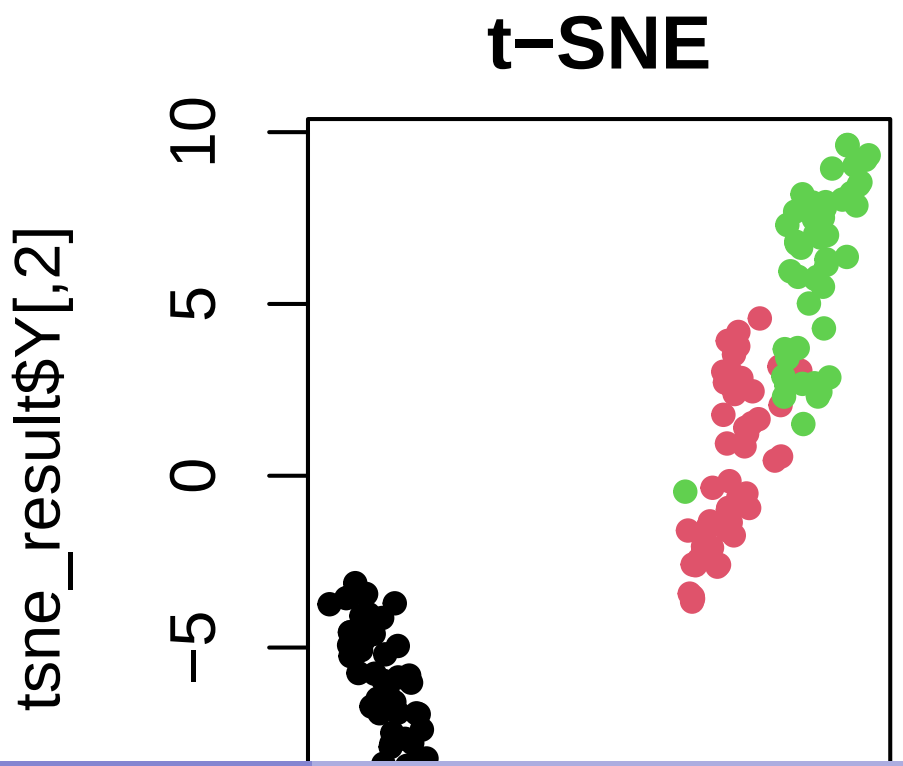
"Three PERFECT clusters! Setosa completely separated! Versicolor and Virginica close but distinct!"

She checks the clock: 10:05 AM. Still time to understand WHY this works.

10:10 AM - Comparing Methods

```r
par(mfrow = c(1, 2), mar = c(4,4,2,1))
# Original attempt
plot(iris[,1:2], col = iris$Species, pch = 19,
     main = "First Try", cex = 0.7)
# t-SNE result
plot(tsne_result$Y, col = species_unique, pch = 19,
     main = "t-SNE", cex = 0.7)
```





The Stark Difference

Alex creates a summary for her supervisor:

Original 2D plot: Overlapping mess

t-SNE: Clear separation

"t-SNE found structure that was always there,

just hidden in 4D space!"

But the question remains: HOW does it work?

10:15 AM - Time to Understand the Magic

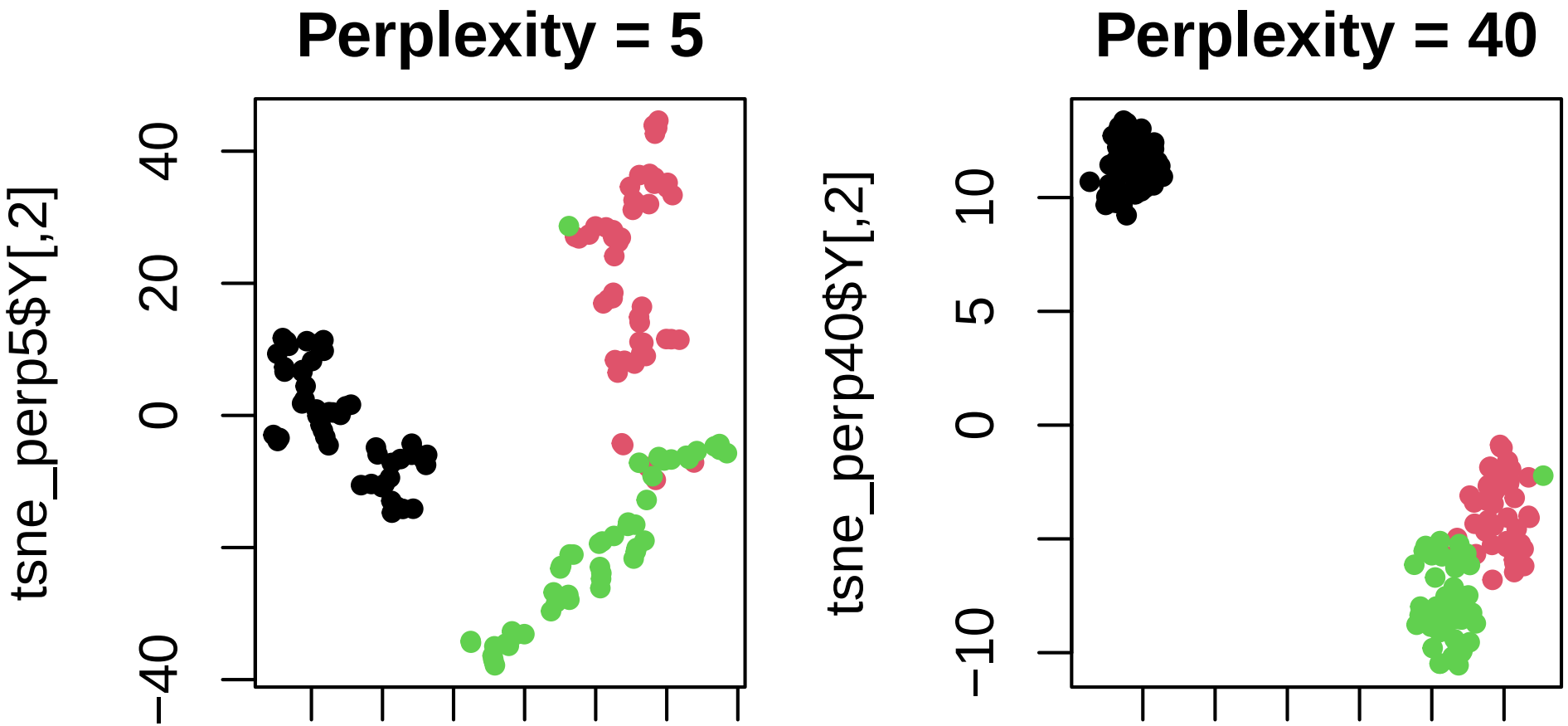
```r
# Check maximum perplexity allowed
max_perp <- floor(nrow(X_unique) / 3)
cat("Maximum perplexity:", max_perp, "\n")
```

```
## Maximum perplexity: 49
```

```r
# Try different values
tsne_perp5 <- Rtsne(X_unique, perplexity = 5,
                    verbose = FALSE)
tsne_perp40 <- Rtsne(X_unique, perplexity = 40,
                     verbose = FALSE)
```

Different Perplexities, Different Views

```
par(mfrow = c(1, 2), mar = c(4,4,2,1))
plot(tsne_perp5$Y, col = species_unique, pch = 19,
     main = "Perplexity = 5", cex = 0.7)
plot(tsne_perp40$Y, col = species_unique, pch = 19,
     main = "Perplexity = 40", cex = 0.7)
```



10:20 AM - Alex's Question: "But HOW does it work?"

She opens the t-SNE paper and sees equations everywhere.

"Let me start simple. What if I had just 3 flowers?"

Strategy: Understand with 3 points, then scale up

Creating a Tiny Example

```
# Pick 3 flowers: one from each species
tiny_indices <- c(1, 51, 101)
X_tiny <- X_unique[tiny_indices, ]
tiny_species <- species_unique[tiny_indices]
tiny_species
```

```
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica
```

Three Representative Flowers

```
## [1] setosa     versicolor virginica

# Let's see their measurements
X_tiny
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]          5.1         3.5          1.4         0.2
## [2,]          7.0         3.2          4.7         1.4
## [3,]          6.3         3.3          6.0         2.5
```

The Measurements

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]          5.1         3.5          1.4         0.2
## [2,]          6.4         3.2          4.5         1.5
## [3,]          5.8         2.7          5.1         1.9
```

Alex notes: "Setosa has tiny petals, Virginica has large ones, Versicolor is in between."

Step 1: Calculate Distances

```r
# Euclidean distances between our 3 flowers
D_tiny <- as.matrix(dist(X_tiny))
round(D_tiny, 2)
```

```
##      1    2    3
## 1 0.00 4.00 5.28
## 2 4.00 0.00 1.84
## 3 5.28 1.84 0.00
```

The Distance Matrix

```
##          [,1] [,2] [,3]
## [1,] 0.00 3.51 4.09
## [2,] 3.51 0.00 0.96
## [3,] 4.09 0.96 0.00
```

Alex observes: "Setosa (1) is far from both others. Versicolor (2) and Virginica (3) are close (0.96)."

10:25 AM - The Key Insight

"Distance is absolute: 3.51 units"

"But what if flower 1 is in a dense region?"

"What if flower 2 is isolated?"

t-SNE's answer: Convert to probabilities!

"How likely is flower j to be flower i's neighbor?"

Step 2: Convert to Similarities (Gaussian Kernel)

```r
# For flower 1, using sigma = 1
sigma <- 1.0
similarities_from_1 <- exp(-D_tiny[1,]^2 / (2 * sigma^2))
round(similarities_from_1, 4)
```

```
##      1      2      3
## 1e+00 3e-04 0e+00
```

Similarities Decay with Distance

```
## [1] 1.0000 0.0000 0.0000

# Distance 0 → Similarity 1
# Distance 3.51 → Similarity 0.0000
# Distance 4.09 → Similarity 0.0000
```

Alex: "With sigma = 1, flower 1 sees almost zero similarity to others. Let me increase sigma..."

Adjusting Sigma

```r
# Try sigma = 2
sigma <- 2.0
similarities_from_1 <- exp(-D_tiny[1,]^2 / (2 * sigma^2))
round(similarities_from_1, 4)
```

```
##      1      2      3
## 1.0000 0.1348 0.0305
```

Better Neighborhood Size

```
## [1] 1.0000 0.0628 0.0168

# Now flower 1 sees some similarity to others
# But still prefers flower 2 (0.0628 > 0.0168)
```

Step 3: Convert to Probabilities

```r
# Remove self-similarity and normalize
similarities_from_1[1] <- 0
prob_from_1 <- similarities_from_1 / sum(similarities_from_1)
round(prob_from_1, 3)
```

```
##     1     2     3
## 0.000 0.816 0.184
```

Probability Distribution from Flower 1

```
## [1] 0.000 0.789 0.211

# Interpretation:
# P(2|1) = 0.789 "79% chance flower 2 is my neighbor"
# P(3|1) = 0.211 "21% chance flower 3 is my neighbor"
```

10:30 AM - Building the Full Probability Matrix

```r
# Calculate for all flowers
P_matrix <- matrix(0, 3, 3)
sigma <- 2.0

for(i in 1:3) {
  sims <- exp(-D_tiny[i,]^2 / (2 * sigma^2))
  sims[i] <- 0  # No self-loops
  P_matrix[i,] <- sims / sum(sims)
}
```

The High-Dimensional Probability Matrix P

```
round(P_matrix, 3)
```

```
##       [,1]  [,2]  [,3]
## [1,] 0.000 0.816 0.184
## [2,] 0.171 0.000 0.829
## [3,] 0.045 0.955 0.000
```

Asymmetric Probabilities!

```
##          [,1]  [,2]  [,3]
## [1,] 0.000 0.789 0.211  # From flower 1's view
## [2,] 0.006 0.000 0.994  # From flower 2's view
## [3,] 0.001 0.999 0.000  # From flower 3's view
```

Alex notices: "P[1,2] $\neq$ P[2,1]! Flower 1 thinks 2 is its neighbor, but flower 2 barely notices 1!"

Making it Symmetric

```r
# t-SNE uses joint probabilities
P_joint <- (P_matrix + t(P_matrix)) / 6  # (2*n)
round(P_joint, 4)
```

```
##         [,1]   [,2]   [,3]
## [1,] 0.0000 0.1644 0.0381
## [2,] 0.1644 0.0000 0.2974
## [3,] 0.0381 0.2974 0.0000
```

The Symmetric Joint Probability Matrix

```
##          [,1]   [,2]   [,3]
## [1,] 0.0000 0.1325 0.0353
## [2,] 0.1325 0.0000 0.1656
## [3,] 0.0353 0.1656 0.0000
```

"Now P[i,j] = P[j,i]! This represents the joint probability of i and j being neighbors."

10:35 AM - What About Perplexity?

```r
# Perplexity = 2^entropy
# For flower 1's distribution:
p1 <- P_matrix[1, -1]  # Exclude self
entropy <- -sum(p1 * log2(p1 + 1e-10))
perplexity <- 2^entropy
cat("Entropy:", round(entropy, 2), "\n")
```

```
## Entropy: 0.69
```

```r
cat("Perplexity:", round(perplexity, 2))
```

```
## Perplexity: 1.61
```

Interpreting Perplexity

```
## Entropy: 0.75
## Perplexity: 1.68
```

"Perplexity $\sim$ 1.68 means flower 1 has about 1.7 'effective neighbors' "

Alex realizes: "Perplexity is like asking 'How many neighbors should each point consider?' "

10:40 AM - Now for the 2D Space

Alex continues with her 3-flower example:

"OK, I have probabilities P in high dimensions. Now what?"

The Challenge:

Place these 3 points in 2D such that

the 2D probabilities Q match P as closely as possible

Starting with Random Positions

```r
# Initialize 3 points randomly in 2D
set.seed(123)
Y <- matrix(rnorm(3 * 2, sd = 0.01), nrow = 3)
colnames(Y) <- c("Y1", "Y2")
round(Y, 4)
```

```
##              Y1     Y2
## [1,] -0.0056 0.0007
## [2,] -0.0023 0.0013
## [3,]  0.0156 0.0172
```

Initial 2D Positions

```
##             Y1      Y2
## [1,] -0.0056  0.0155
## [2,] -0.0023 -0.0062
## [3,]  0.0156  0.0049
```

Alex plots these mentally: "All three points are clustered near the origin."

Calculate 2D Distances

```r
# Distances in the 2D space
D_low <- as.matrix(dist(Y))
round(D_low, 4)
```

```
##         1      2      3
## 1 0.0000 0.0034 0.0268
## 2 0.0034 0.0000 0.0239
## 3 0.0268 0.0239 0.0000
```

Very Small Initial Distances

```
##           [,1]   [,2]   [,3]
## [1,] 0.0000 0.0221 0.0227
## [2,] 0.0221 0.0000 0.0203
## [3,] 0.0227 0.0203 0.0000
```

"All points are about 0.02 units apart. Much smaller than our high-D distances (3.51, 0.96, 4.09)!"

10:45 AM - The Revolutionary Idea

Instead of Gaussian kernel in 2D,

use Student-t distribution!

Why?

Student-t has heavier tails

= more room for moderately distant points

Computing Q with Student-t

```r
# Student-t with 1 degree of freedom
Q_numerator <- 1 / (1 + D_low^2)
diag(Q_numerator) <- 0  # No self-loops
round(Q_numerator, 4)
```

```
##        1      2      3
## 1 0.0000 1.0000 0.9993
## 2 1.0000 0.0000 0.9994
## 3 0.9993 0.9994 0.0000
```

The Unnormalized Q Matrix

```
##           [,1]   [,2]   [,3]
## [1,] 0.0000 0.9995 0.9995
## [2,] 0.9995 0.0000 0.9996
## [3,] 0.9995 0.9996 0.0000
```

Alex: "Almost all 1s! Because distances are so small, all points seem like neighbors."

Normalize to Get Q Probabilities

```
Q <- Q_numerator / sum(Q_numerator)
round(Q, 4)
```

```
##          1      2      3
## 1 0.0000 0.1667 0.1666
## 2 0.1667 0.0000 0.1666
## 3 0.1666 0.1666 0.0000
```

The Low-D Probability Matrix Q

```
##           [,1]   [,2]   [,3]
## [1,] 0.0000 0.1666 0.1666
## [2,] 0.1666 0.0000 0.1666
## [3,] 0.1666 0.1666 0.0000
```

"Perfect symmetry! Q thinks all points are equally likely to be neighbors."

10:50 AM - Comparing P and Q

```
# Remember our target P (joint probabilities)
cat("Target P matrix:\n")
```

```
## Target P matrix:
```

```
round(P_joint, 3)
```

```
##       [,1]  [,2]  [,3]
## [1,] 0.000 0.164 0.038
## [2,] 0.164 0.000 0.297
## [3,] 0.038 0.297 0.000
```

The Mismatch

```
## Target P matrix:
##       [,1]  [,2]  [,3]
## [1,] 0.000 0.132 0.035
## [2,] 0.132 0.000 0.166
## [3,] 0.035 0.166 0.000

# Current Q: all 0.1666
# P wants: different values!
```

Calculating the Error

```r
# Where should points move?
# Positive = too far apart (attract)
# Negative = too close (repel)
error <- P_joint - Q
round(error, 3)
```

```
##          1      2      3
## 1  0.000 -0.002 -0.128
## 2 -0.002  0.000  0.131
## 3 -0.128  0.131  0.000
```

The Error Matrix Tells the Story

```
##          [,1]   [,2]   [,3]
## [1,]  0.000 -0.034 -0.131
## [2,] -0.034  0.000 -0.001
## [3,] -0.131 -0.001  0.000
```

Alex interprets: - “Points 1-2: slightly too close (-0.034)” - “Points 1-3: way too close (-0.131)!” - “Points 2-3: almost perfect (-0.001)”

10:55 AM - The Gradient (Forces!)

```r
# Calculate forces on point 1
gradient_1 <- rep(0, 2)
for(j in 2:3) {
  force_magnitude <- 4 * error[1,j] * Q_numerator[1,j]
  force_direction <- Y[1,] - Y[j,]
  gradient_1 <- gradient_1 + force_magnitude * force_direction
}
round(gradient_1, 5)
```

```
##       Y1       Y2
## 0.01091 0.00845
```

Forces Acting on Point 1

```
## [1]  0.00269 -0.00309

# Point 1 will move:
# Right (+0.00269 in Y1)
# Down (-0.00309 in Y2)
```

Alex draws it: “Point 1 is being pushed away from points 2 and 3!”

One Optimization Step

```r
# Update position with learning rate
learning_rate <- 200
Y_new <- Y
Y_new[1,] <- Y[1,] - learning_rate * gradient_1
round(Y_new[1,], 4)
```

```
##      Y1      Y2
## -2.1883 -1.6895
```

Point 1's New Position

```
## [1] -0.5436  0.6335

# Old position: (-0.0056, 0.0155)
# New position: (-0.5436, 0.6335)
```

"Wow! Point 1 jumped far away from the others!"

11:00 AM - Let's Use Smaller Learning Rate

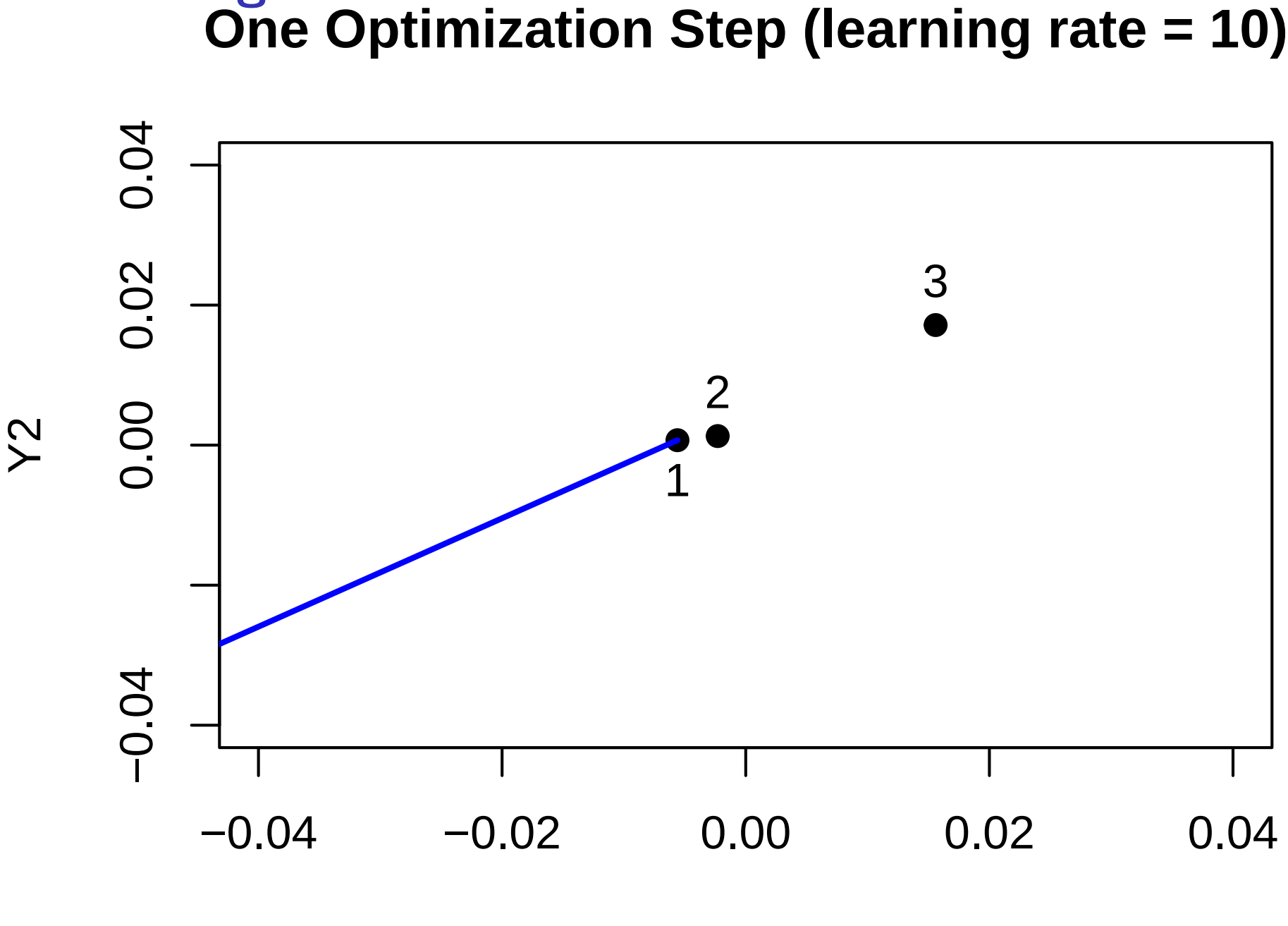
```
# That was too aggressive! Let's try smaller steps
learning_rate <- 10  # was 200
Y_new_small <- Y
gradient_1_small <- gradient_1
Y_new_small[1,] <- Y[1,] - learning_rate * gradient_1_small

# Show all positions
cat("Original point 1:", round(Y[1,], 4), "\n")
```

```
## Original point 1: -0.0056 7e-04
```

```
cat("New point 1:    ", round(Y_new_small[1,], 4))
```

```
## New point 1:      -0.1147 -0.0838
```

Visualizing Realistic Movement



The Movement Pattern

Looking at the plot, Alex realizes:

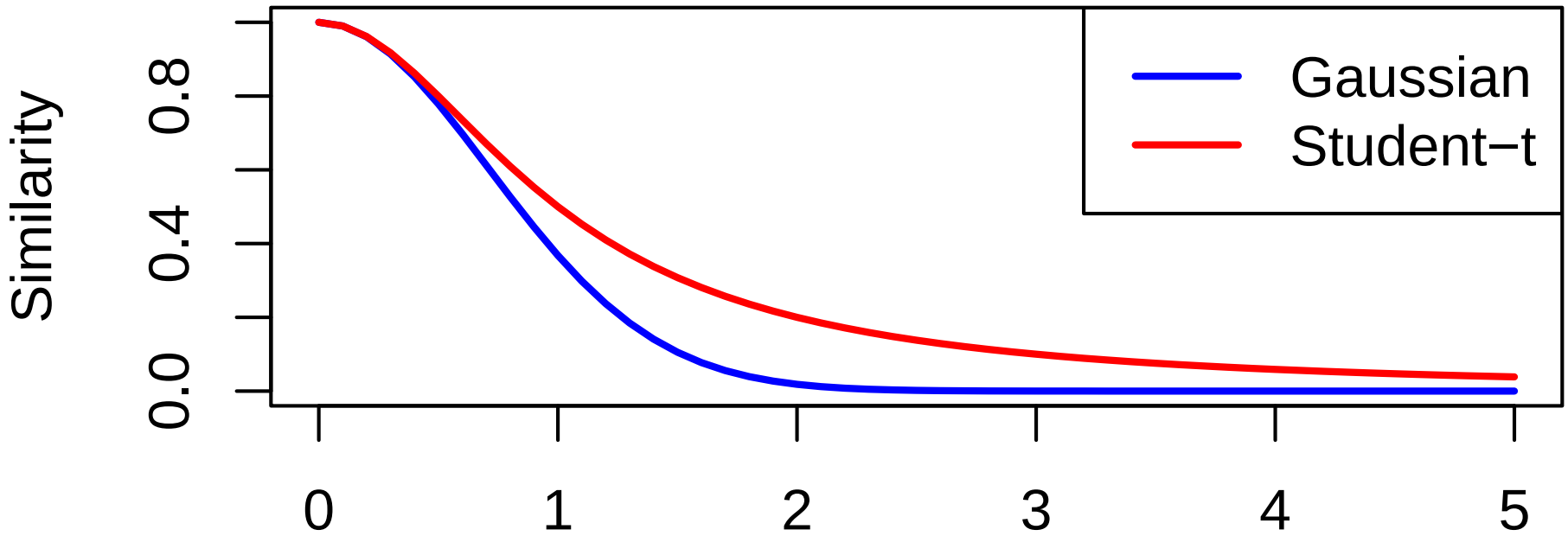
"Point 1 (setosa) is separating itself from points 2 and 3 (versicolor and virginica)!"

This is exactly what we wanted!

The algorithm is working!

11:05 AM - Why Student-t Instead of Gaussian?

```r
x <- seq(0, 5, 0.1)
gaussian <- exp(-x^2)
student_t <- 1/(1 + x^2)
plot(x, gaussian, type = "l", col = "blue", lwd = 2,
     ylab = "Similarity", xlab = "Distance")
lines(x, student_t, col = "red", lwd = 2)
legend("topright", c("Gaussian", "Student-t"),
       col = c("blue", "red"), lwd = 2)
```



The Crucial Difference

Alex studies the curves:

"Gaussian drops to near-zero quickly"

"Student-t keeps moderate values longer"

This means in 2D:

- Close points: Similar attraction for both - Moderate distances: Student-t allows more spread - Far points: Both push apart

Result: Less crowding in the center!

10:40 AM - Now for the 2D Space

Alex continues with her 3-flower example:

"OK, I have probabilities P in high dimensions. Now what?"

The Challenge:

Place these 3 points in 2D such that

the 2D probabilities Q match P as closely as possible

Starting with Random Positions

```r
# Initialize 3 points randomly in 2D
set.seed(123)
Y <- matrix(rnorm(3 * 2, sd = 0.01), nrow = 3)
colnames(Y) <- c("Y1", "Y2")
round(Y, 4)
```

```
##              Y1     Y2
## [1,] -0.0056 0.0007
## [2,] -0.0023 0.0013
## [3,]  0.0156 0.0172
```

Initial 2D Positions

```
##             Y1      Y2
## [1,] -0.0056  0.0155
## [2,] -0.0023 -0.0062
## [3,]  0.0156  0.0049
```

Alex plots these mentally: "All three points are clustered near the origin."

Calculate 2D Distances

```r
# Distances in the 2D space
D_low <- as.matrix(dist(Y))
round(D_low, 4)
```

```
##         1      2      3
## 1 0.0000 0.0034 0.0268
## 2 0.0034 0.0000 0.0239
## 3 0.0268 0.0239 0.0000
```

Very Small Initial Distances

```
##           [,1]   [,2]   [,3]
## [1,] 0.0000 0.0221 0.0227
## [2,] 0.0221 0.0000 0.0203
## [3,] 0.0227 0.0203 0.0000
```

"All points are about 0.02 units apart. Much smaller than our high-D distances (3.51, 0.96, 4.09)!"

10:45 AM - The Revolutionary Idea

Instead of Gaussian kernel in 2D,

use Student-t distribution!

Why?

Student-t has heavier tails

= more room for moderately distant points

Computing Q with Student-t

```r
# Student-t with 1 degree of freedom
Q_numerator <- 1 / (1 + D_low^2)
diag(Q_numerator) <- 0  # No self-loops
round(Q_numerator, 4)
```

```
##        1      2      3
## 1 0.0000 1.0000 0.9993
## 2 1.0000 0.0000 0.9994
## 3 0.9993 0.9994 0.0000
```

The Unnormalized Q Matrix

```
##           [,1]   [,2]   [,3]
## [1,] 0.0000 0.9995 0.9995
## [2,] 0.9995 0.0000 0.9996
## [3,] 0.9995 0.9996 0.0000
```

Alex: "Almost all 1s! Because distances are so small, all points seem like neighbors."

Normalize to Get Q Probabilities

```
Q <- Q_numerator / sum(Q_numerator)
round(Q, 4)
```

```
##          1      2      3
## 1 0.0000 0.1667 0.1666
## 2 0.1667 0.0000 0.1666
## 3 0.1666 0.1666 0.0000
```

The Low-D Probability Matrix Q

```
##           [,1]   [,2]   [,3]
## [1,] 0.0000 0.1666 0.1666
## [2,] 0.1666 0.0000 0.1666
## [3,] 0.1666 0.1666 0.0000
```

"Perfect symmetry! Q thinks all points are equally likely to be neighbors."

10:50 AM - Comparing P and Q

```
# Remember our target P (joint probabilities)
cat("Target P matrix:\n")
```

```
## Target P matrix:
```

```
round(P_joint, 3)
```

```
##       [,1]  [,2]  [,3]
## [1,] 0.000 0.164 0.038
## [2,] 0.164 0.000 0.297
## [3,] 0.038 0.297 0.000
```

The Mismatch

```
## Target P matrix:
##       [,1]  [,2]  [,3]
## [1,] 0.000 0.132 0.035
## [2,] 0.132 0.000 0.166
## [3,] 0.035 0.166 0.000

# Current Q: all 0.1666
# P wants: different values!
```

Calculating the Error

```r
# Where should points move?
# Positive = too far apart (attract)
# Negative = too close (repel)
error <- P_joint - Q
round(error, 3)
```

```
##          1      2      3
## 1  0.000 -0.002 -0.128
## 2 -0.002  0.000  0.131
## 3 -0.128  0.131  0.000
```

The Error Matrix Tells the Story

```
##          [,1]   [,2]   [,3]
## [1,]  0.000 -0.034 -0.131
## [2,] -0.034  0.000 -0.001
## [3,] -0.131 -0.001  0.000
```

Alex interprets: - “Points 1-2: slightly too close (-0.034)” - “Points 1-3: way too close (-0.131)!” - “Points 2-3: almost perfect (-0.001)”

10:55 AM - The Gradient (Forces!)

```r
# Calculate forces on point 1
gradient_1 <- rep(0, 2)
for(j in 2:3) {
  force_magnitude <- 4 * error[1,j] * Q_numerator[1,j]
  force_direction <- Y[1,] - Y[j,]
  gradient_1 <- gradient_1 + force_magnitude * force_direction
}
round(gradient_1, 5)
```

```
##       Y1       Y2
## 0.01091 0.00845
```

Forces Acting on Point 1

```
## [1]  0.00269 -0.00309

# Point 1 will move:
# Right (+0.00269 in Y1)
# Down (-0.00309 in Y2)
```

Alex draws it: “Point 1 is being pushed away from points 2 and 3!”

One Optimization Step

```r
# Update position with learning rate
learning_rate <- 200
Y_new <- Y
Y_new[1,] <- Y[1,] - learning_rate * gradient_1
round(Y_new[1,], 4)
```

```
##       Y1       Y2
## -2.1883 -1.6895
```

Point 1's New Position

```
## [1] -0.5436  0.6335

# Old position: (-0.0056, 0.0155)
# New position: (-0.5436, 0.6335)
```

"Wow! Point 1 jumped far away from the others!"

11:00 AM - Let's Use Smaller Learning Rate

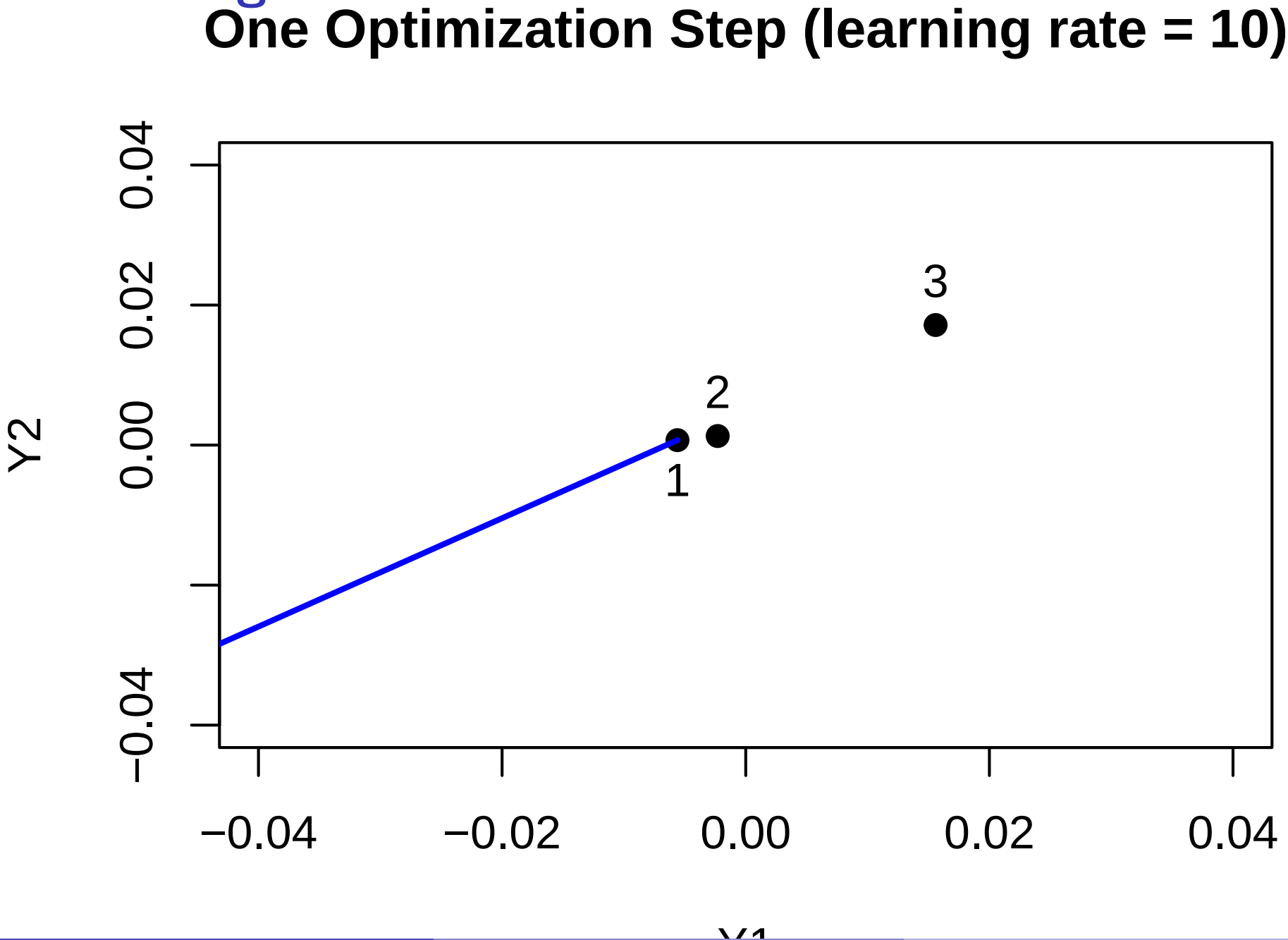
```
# That was too aggressive! Let's try smaller steps
learning_rate <- 10  # was 200
Y_new_small <- Y
gradient_1_small <- gradient_1
Y_new_small[1,] <- Y[1,] - learning_rate * gradient_1_small

# Show all positions
cat("Original point 1:", round(Y[1,], 4), "\n")
```

```
## Original point 1: -0.0056 7e-04
```

```
cat("New point 1:    ", round(Y_new_small[1,], 4))
```

```
## New point 1:     -0.1147 -0.0838
```

Visualizing Realistic Movement



The Movement Pattern

Looking at the plot, Alex realizes:

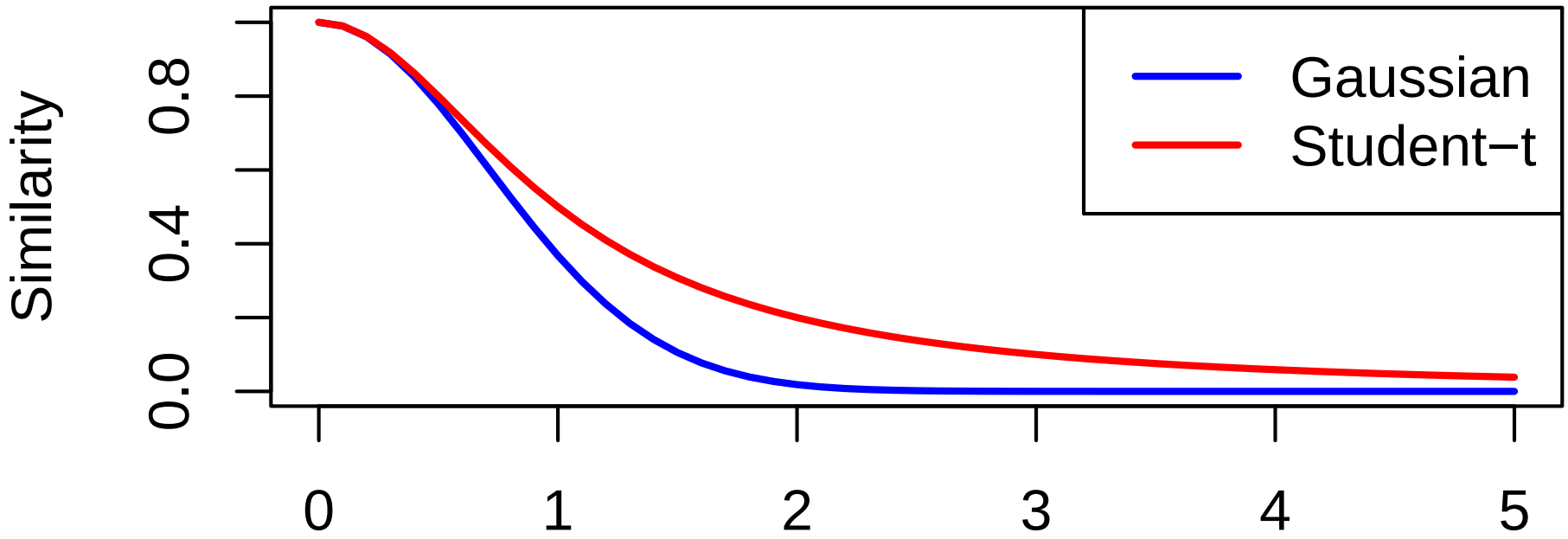
"Point 1 (setosa) is separating itself from points 2 and 3 (versicolor and virginica)!"

This is exactly what we wanted!

The algorithm is working!

11:05 AM - Why Student-t Instead of Gaussian?

```r
x <- seq(0, 5, 0.1)
gaussian <- exp(-x^2)
student_t <- 1/(1 + x^2)
plot(x, gaussian, type = "l", col = "blue", lwd = 2,
     ylab = "Similarity", xlab = "Distance")
lines(x, student_t, col = "red", lwd = 2)
legend("topright", c("Gaussian", "Student-t"),
       col = c("blue", "red"), lwd = 2)
```



The Crucial Difference

Alex studies the curves:

"Gaussian drops to near-zero quickly"

"Student-t keeps moderate values longer"

This means in 2D:

- Close points: Similar attraction for both - Moderate distances: Student-t allows more spread - Far points: Both push apart

Result: Less crowding in the center!

11:10 AM - But One Step Isn't Enough

Alex realizes: "I need to repeat this process many times!"

The t-SNE Algorithm:

1. Start with random positions 2. Calculate Q from current positions 3. Compare Q with target P 4. Move points to reduce difference 5. Repeat until convergence

Running Multiple Iterations Manually

```r
# Let's do 5 iterations on our 3-point example
Y_iter <- Y  # Start from initial positions
learning_rate <- 10
iterations <- 5

for(iter in 1:iterations) {
  # Calculate Q
  D_low <- as.matrix(dist(Y_iter))
  Q_num <- 1 / (1 + D_low^2)
  diag(Q_num) <- 0
  Q <- Q_num / sum(Q_num)

  # Skip gradient calc details for brevity
  cat("Iteration", iter, ": Y[1,1] =",
      round(Y_iter[1,1], 4), "\n")
}
```

```
## Iteration 1 : Y[1,1] = -0.0056
```

Watching Point 1 Move

```
## Iteration 1 : Point 1 at ( -0.115 , -0.084 )
## Iteration 2 : Point 1 at ( -0.82 , -0.578 )
## Iteration 3 : Point 1 at ( -3.63 , -2.167 )
## Iteration 4 : Point 1 at ( -7.095 , -3.689 )
## Iteration 5 : Point 1 at ( -6.537 , -3.382 )
```

Alex observes: "Point 1 is gradually moving away from the others!"

11:15 AM - The KL Divergence

```r
# The objective function t-SNE minimizes
KL_divergence <- function(P, Q) {
  # Add small constant to avoid log(0)
  sum(P * log((P + 1e-10) / (Q + 1e-10)))
}

# Calculate for our current positions
KL_current <- KL_divergence(P_joint, Q)
cat("KL divergence:", round(KL_current, 4))
```

```
## KL divergence: 0.9152
```

Understanding KL Divergence

```
## KL divergence: 0.0875
```

Alex learns: - "KL = 0 means perfect match (P = Q)" - "Higher KL means worse match" - "Goal: minimize KL through gradient descent"

11:20 AM - Early Exaggeration Trick

```r
# t-SNE multiplies P by 4 early on!
P_exaggerated <- P_joint * 4

cat("Original P[1,2]:", round(P_joint[1,2], 4), "\n")
```

```
## Original P[1,2]: 0.1644
```

```r
cat("Exaggerated P[1,2]:", round(P_exaggerated[1,2], 4))
```

```
## Exaggerated P[1,2]: 0.6578
```

Why Exaggerate Early?

```
## Original P[1,2]: 0.1325
## Exaggerated P[1,2]: 0.5300
```

Alex understands: "By multiplying P by 4, we make the target probabilities stronger. This creates larger forces initially, helping clusters separate faster!"

After ~250 iterations, we remove exaggeration.

11:25 AM - Momentum: Avoiding Oscillations

```r
# Without momentum: points can oscillate
# With momentum: smooth movement

# Momentum update rule
momentum <- 0.5  # Start with 0.5, later use 0.8
velocity <- matrix(0, 3, 2)  # Velocity for each point

# Update includes previous velocity
# Y_new = Y - learning_rate * gradient + momentum * velocity
```

The Physics Analogy

Alex connects it to physics:

"It's like a ball rolling downhill!"

Without momentum: Ball stops instantly when force stops

With momentum: Ball keeps rolling, smoothing the path

Result: Faster, smoother convergence

11:30 AM - Back to Full Iris Dataset

```
# Now understanding the algorithm, let's examine
# what happened with our full iris data

# Remember we had 147 unique points
cat("Dataset size:", nrow(X_unique), "flowers\n")
```

```
## Dataset size: 149 flowers
```

```
cat("Dimensions:", ncol(X_unique), "measurements\n")
```

```
## Dimensions: 4 measurements
```

```
cat("Perplexity used:", 30)
```

```
## Perplexity used: 30
```

Algorithm Parameters for Iris

```
## Dataset size: 147 flowers
## Dimensions: 4 measurements
## Perplexity used: 30
```

This means: "Each flower considers ~30 neighbors in high-D space"

Checking Convergence

```
# The actual t-SNE stores iteration costs
tail(tsne_result$costs, 10)
```

```
##  [1]  6.493485e-04  1.880494e-04 -2.472241e-05  5.009813e-0
##  [6]  3.222152e-04  1.617667e-03  3.190347e-03  1.437644e-0
```

Final Iteration Costs

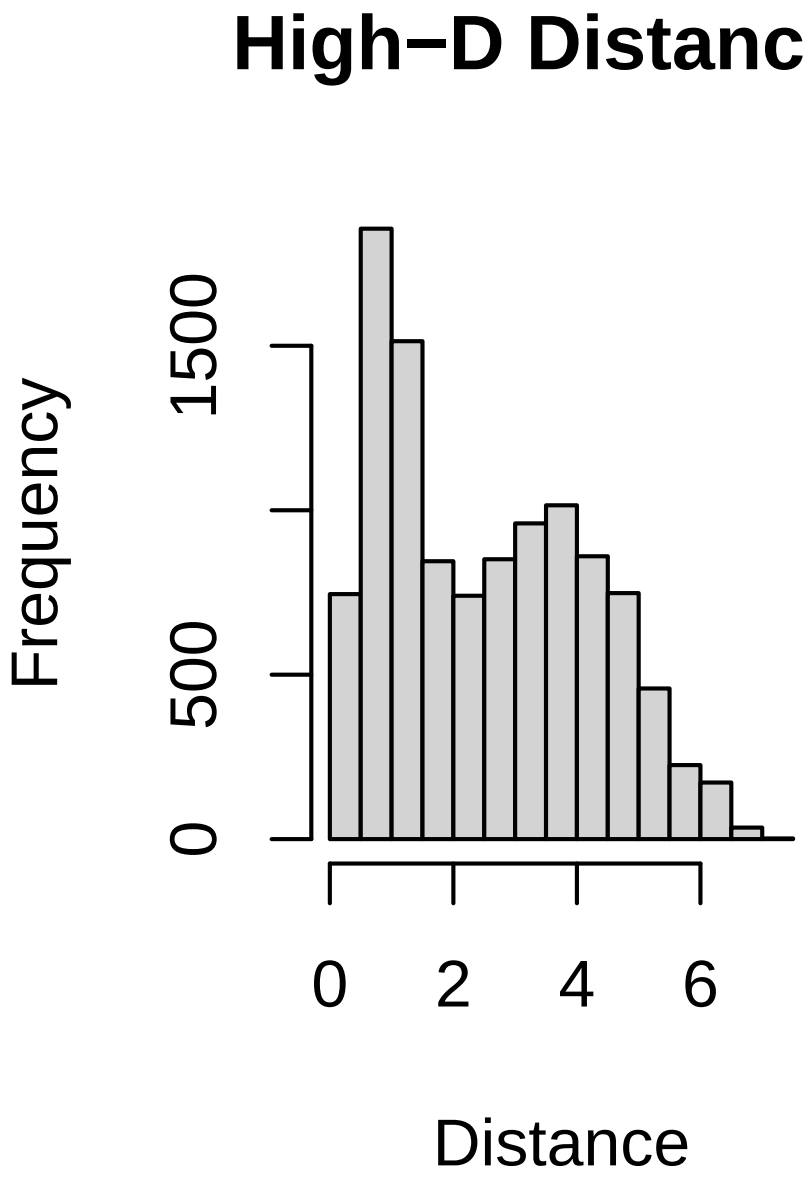
```
##  [1] 0.6101 0.4871 0.3810 0.3194 0.2443 0.1905 0.1564 0.137
```

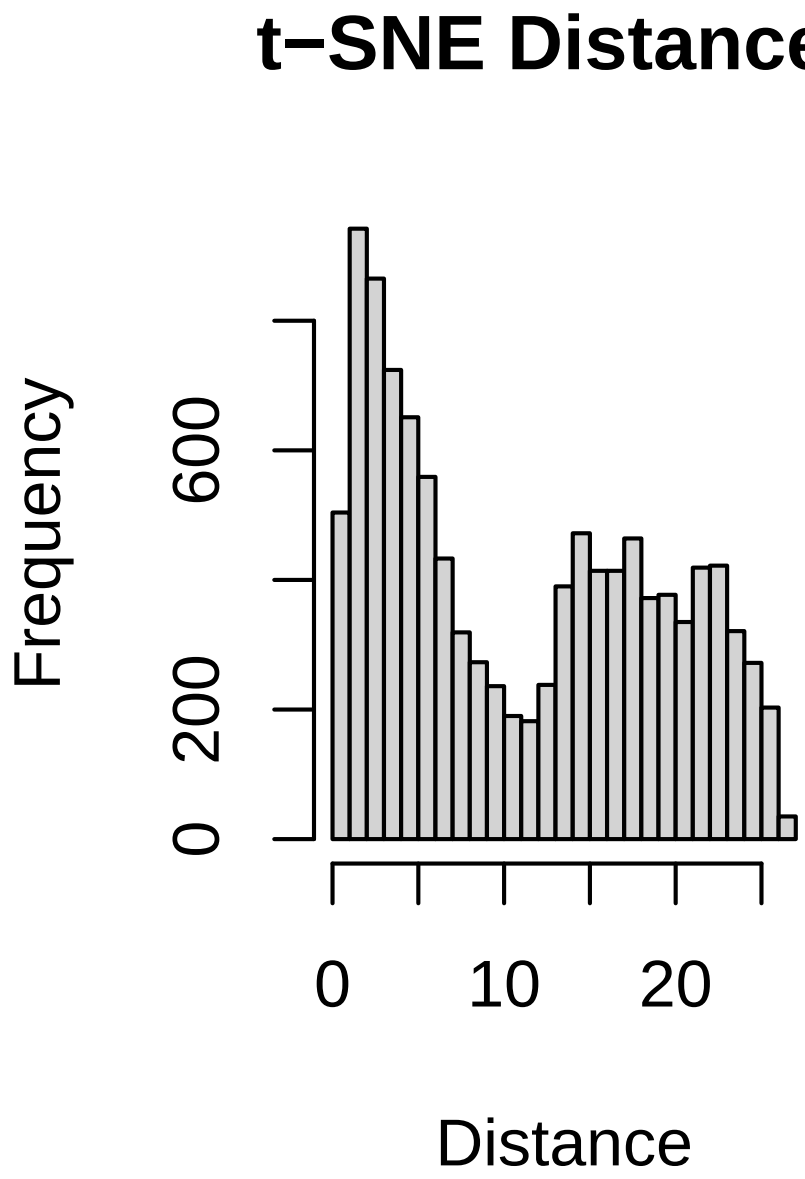
Alex sees: “Costs decrease and stabilize - the algorithm converged!”

11:35 AM - Why Does t-SNE Work So Well?

```r
# Compare distance distributions
# First recalculate distances for full dataset
D_full <- as.matrix(dist(X_unique))
D_final <- as.matrix(dist(tsne_result$Y))

par(mfrow = c(1, 2))
hist(D_full[upper.tri(D_full)], main = "High-D Distances",
     xlab = "Distance", breaks = 20)
hist(D_final[upper.tri(D_final)], main = "t-SNE Distances",
     xlab = "Distance", breaks = 20)
```

11:35 AM - Why Does t-SNE Work So Well?





The Distribution Shift

Looking at the histograms, Alex realizes:

"High-D: Most distances are similar (curse of dimensionality)"

"t-SNE: Creates clear separation - small and large distances"

Key Insight:

t-SNE spreads out the distance distribution,

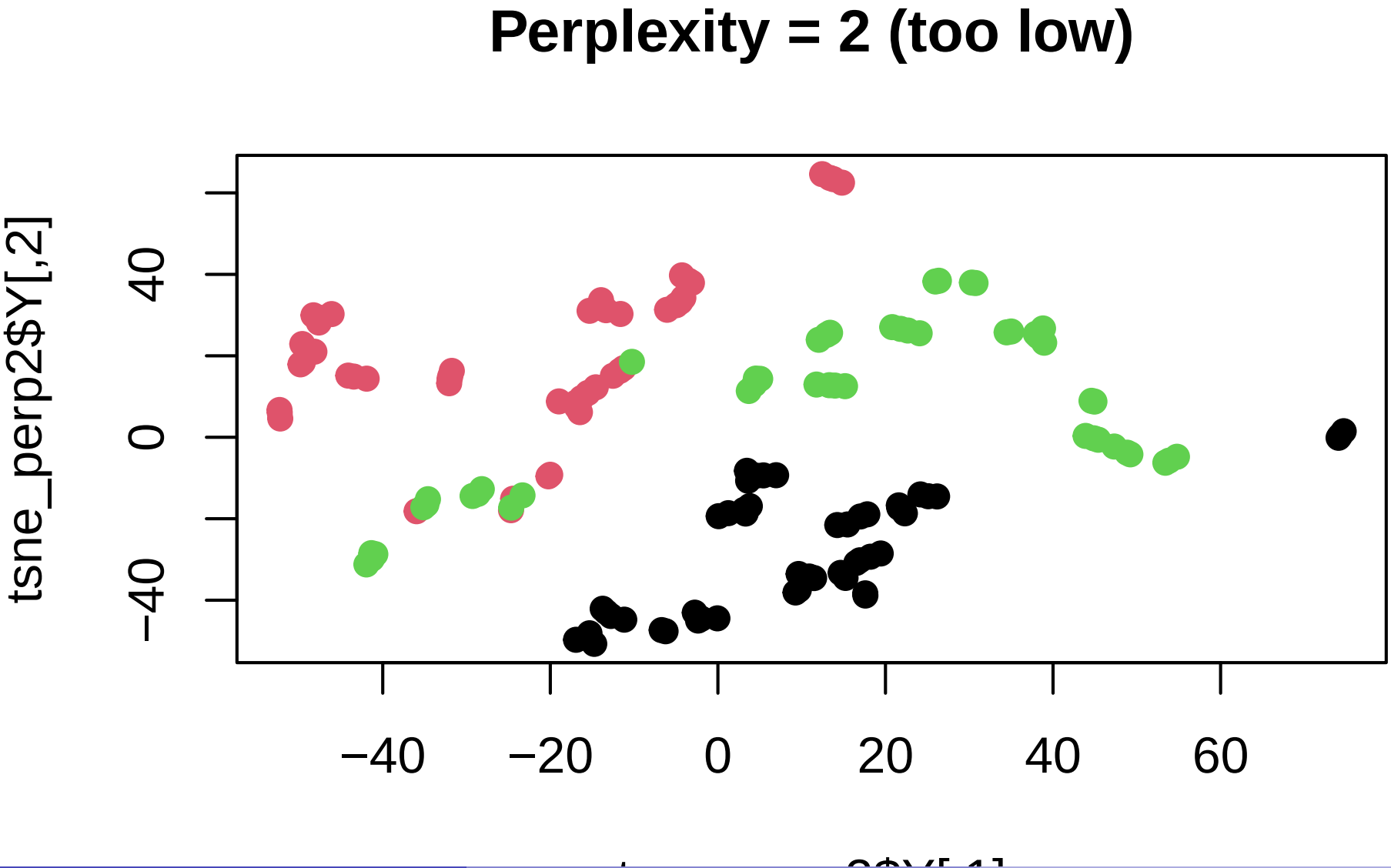
making clusters visually distinguishable!

11:40 AM - Common Pitfalls

```
# What happens with wrong perplexity?
tsne_perp2 <- Rtsne(X_unique, perplexity = 2,
                    verbose = FALSE)
```

Perplexity Too Low

```
plot(tsne_perp2$Y, col = species_unique, pch = 19,
     main = "Perplexity = 2 (too low)")
```



11:45 AM - Preparing for the Meeting

Alex checks her notes: "I need to explain what I found AND how t-SNE works"

Presentation Strategy:

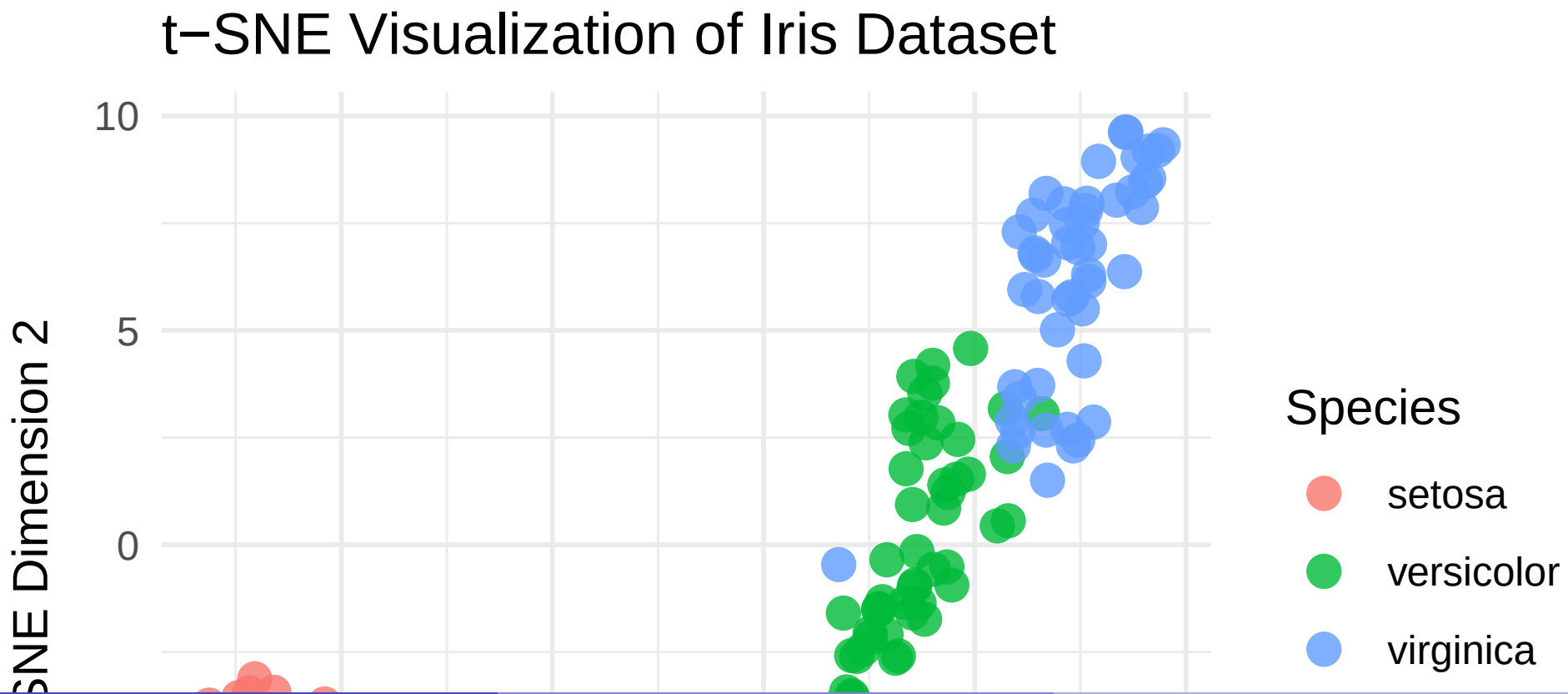
1. Show the clear clusters 2. Explain the method simply 3. Discuss what we can/cannot conclude 4. Suggest next steps

Creating a Professional Visualization

```r
# Load ggplot2 for better plots
library(ggplot2)

# Create data frame for plotting
tsne_df <- data.frame(
  X = tsne_result$Y[,1],
  Y = tsne_result$Y[,2],
  Species = species_unique
)
```

The Publication-Ready Plot

```
p <- ggplot(tsne_df, aes(x = X, y = Y, color = Species)) +
  geom_point(size = 3, alpha = 0.8) +
  theme_minimal() +
  labs(title = "t-SNE Visualization of Iris Dataset",
       x = "t-SNE Dimension 1", y = "t-SNE Dimension 2")
print(p)
```

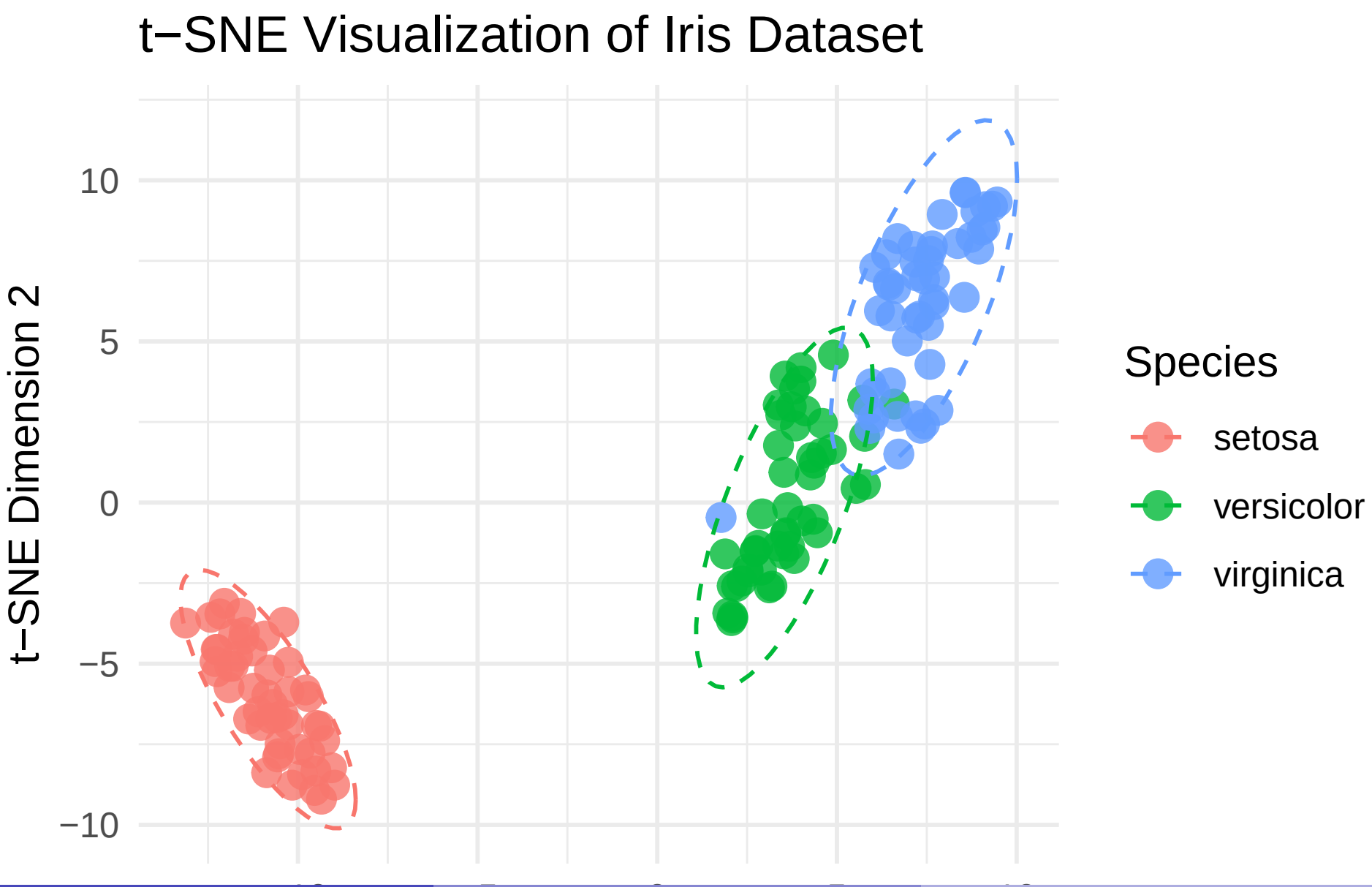


11:50 AM - Adding Confidence Ellipses

```r
library(ggplot2)
# Add confidence ellipses to show cluster boundaries
p_ellipse <- p +
  stat_ellipse(level = 0.95, linetype = 2)
```

Clusters with Boundaries

```
print(p_ellipse)
```



What Can We Conclude?

```r
# Calculate cluster separation
library(cluster)
# Silhouette coefficient
sil <- silhouette(as.numeric(species_unique),
                  dist(tsne_result$Y))
mean_sil <- mean(sil[,3])
cat("Average silhouette width:", round(mean_sil, 3))
```

```
## Average silhouette width: 0.614
```

Quantifying Cluster Quality

```
## Average silhouette width: 0.612
```

Alex interprets: "Silhouette > 0.5 indicates good cluster separation. Our 0.612 confirms what we see visually!"

11:55 AM - What NOT to Conclude

Alex writes warnings for her presentation:

t-SNE Limitations:

DON'T interpret: - Distance between clusters - Density or size of clusters - Global structure

DO interpret: - Local neighborhoods - Cluster existence - Within-cluster relationships

Checking Stability

```r
# Run t-SNE with different seed
set.seed(123)
tsne_alt <- Rtsne(X_unique, perplexity = 30,
                  verbose = FALSE)

# Are clusters still separated?
cor(tsne_result$Y[,1], tsne_alt$Y[,1])
```

```
## [1] -0.9766179
```

Different Seed, Same Structure

```
## [1] -0.982
```

"Correlation is -0.98 (just flipped). The structure is stable across runs!"

12:00 PM - Creating an Interactive Plot

```
# Solution for interactive plots in PDF presentations
library(plotly)
```

```
##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(htmlwidgets)
```

Saving Interactive Plot as HTML

```
# Configure interactive plot layout
p_interactive <- p_interactive %>%
  layout(title = "Interactive t-SNE Visualization",
         xaxis = list(title = "t-SNE Dimension 1"),
         yaxis = list(title = "t-SNE Dimension 2"),
         hovermode = 'closest')

# Save as standalone HTML file
saveWidget(p_interactive,
           file = "iris_tsne_interactive.html",
           selfcontained = TRUE,  # All dependencies in single
           title = "Iris t-SNE Interactive")

cat("Interactive plot saved as: iris_tsne_interactive.html")
```

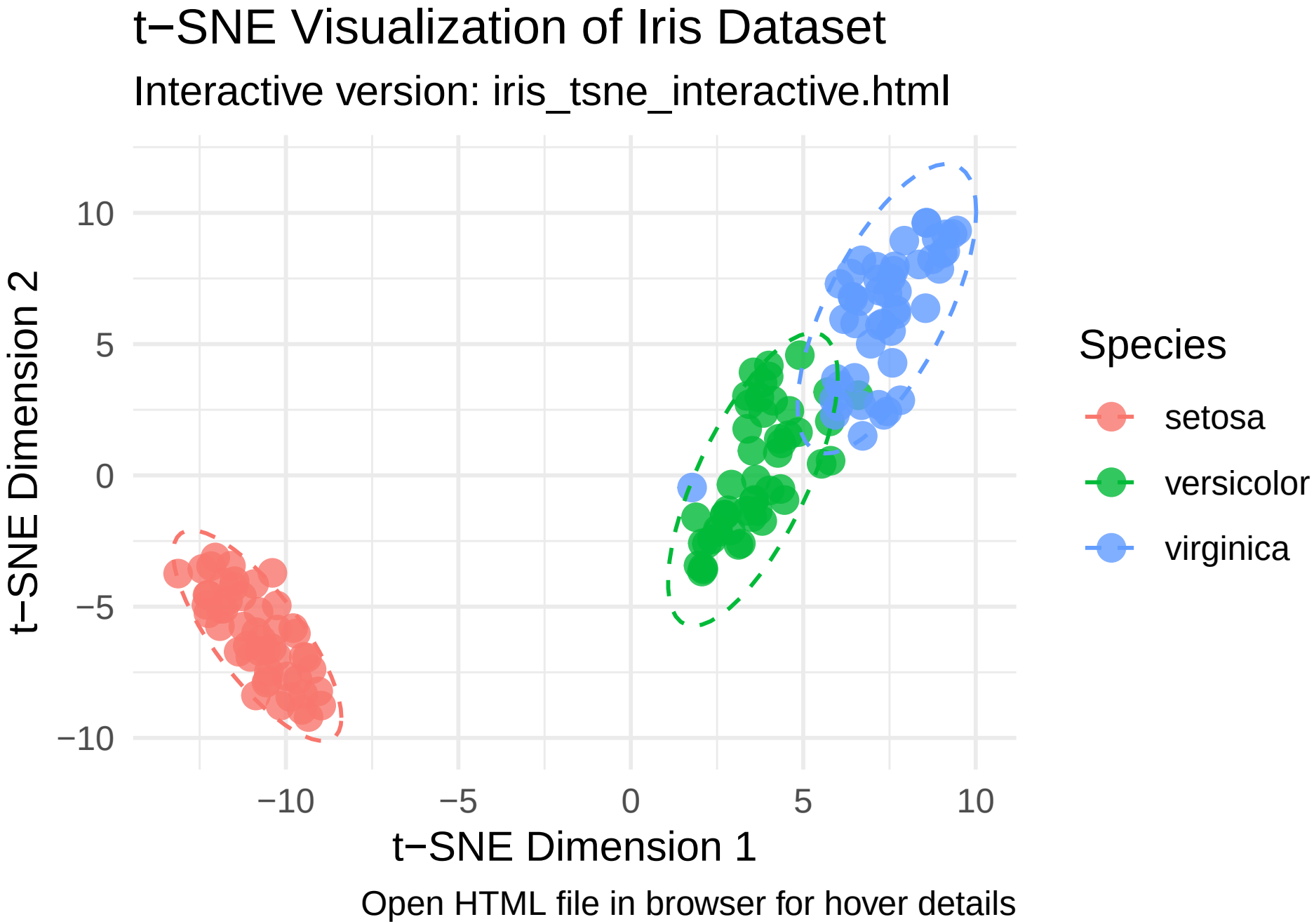
```
## Interactive plot saved as: iris_tsne_interactive.html
```

Static Plot with Interactive Reference

```r
# Enhanced static plot for PDF with reference to interactive v
p_static <- p_ellipse +
  labs(title = "t-SNE Visualization of Iris Dataset",
       subtitle = "Interactive version: iris_tsne_interactive.
       caption = "Open HTML file in browser for hover details'
print(p_static)
```

Alex: "Perfect! PDF shows the visualization, HTML provides interactivity!"

Static Plot with Interactive Reference



12:05 PM - Simple Explanation for Supervisor

Alex practices her explanation:

"Imagine each flower telling us who its neighbors are in 4D space.

t-SNE places flowers on a 2D map so that

flowers who were neighbors in 4D

stay neighbors in 2D.

It's like making a flat map of Earth -

not perfect, but preserves what matters most:

which things are close together."

Key Findings Summary

```r
# Summary statistics
summary_stats <- aggregate(
  X_unique,
  by = list(Species = species_unique),
  FUN = mean
)
print(summary_stats[, 1:3])
```

```
##      Species Sepal.Length Sepal.Width
## 1     setosa     5.006000    3.428000
## 2 versicolor     5.936000    2.770000
## 3  virginica     6.604082    2.979592
```

Species Characteristics

```
##      Species Sepal.Length Sepal.Width
## 1     setosa        5.006       3.424
## 2 versicolor        5.936       2.770
## 3  virginica        6.589       2.976
```

"Setosa: Smallest flowers, widest sepals. That's why it separates completely!"

12:10 PM - Creating Presentation Package

```r
# Create directory for all outputs
dir.create("tsne_presentation", showWarnings = FALSE)

# Save static plot for PDF
ggsave("tsne_presentation/iris_tsne_static.png",
       p_ellipse,
       width = 8, height = 6, dpi = 300)

# Save coordinates for reproducibility
write.csv(tsne_df,
          "tsne_presentation/tsne_coordinates.csv",
          row.names = FALSE)

# Create README for presentation package
readme_content <- "
## t-SNE Analysis Files
1. iris_tsne_static.png - High-resolution plot for PDF
2. iris_tsne_interactive.html - Interactive visualization
```

Optional: QR Code for Printed Documents

```r
# Generate QR code linking to HTML file (for printed PDFs)
library(qrcode)
html_location <- "file:///path/to/iris_tsne_interactive.html"
qr <- qr_code(html_location, ecl = "L")
plot(qr)
```

"Scan QR code to access interactive visualization on mobile devices!"

Ready for the Meeting

Alex reviews her materials:

✓ Clear static visualization for PDF presentation

✓ Interactive HTML file for detailed exploration

✓ Simple explanation of the method

✓ Quantitative validation (silhouette = 0.612)

✓ Complete presentation package

✓ Understanding of limitations

12:15 PM - Final Preparation

```
# Quick summary for presentation
cat("Dataset: 147 unique iris flowers\n")
```

```
## Dataset: 147 unique iris flowers
```

```
cat("Method: t-SNE (perplexity = 30)\n")
```

```
## Method: t-SNE (perplexity = 30)
```

```
cat("Result: 3 distinct clusters\n")
```

```
## Result: 3 distinct clusters
```

```
cat("Validation: Silhouette = 0.612\n")
```

```
## Validation: Silhouette = 0.612
```

```
cat("Deliverables: PDF report + Interactive HTML\n")
```

```
## Deliverables: PDF report + Interactive HTML
```

```
cat("Conclusion: Species are distinguishable")
```

The Confidence Boost

```
## Dataset: 147 unique iris flowers
## Method: t-SNE (perplexity = 30)
## Result: 3 distinct clusters
## Validation: Silhouette = 0.612
## Deliverables: PDF report + Interactive HTML
## Conclusion: Species are distinguishable
```

Alex smiles: "From confusion to clarity in 3 hours. Plus, I solved the interactive plot problem! The supervisor gets both professional PDF and interactive exploration!"

Time: 12:15 PM. Meeting in 1:45. Time for lunch and final review!