

Assignment3

Mengtong Zhang (Collaborate with Yunjia Zeng and Shaoyu Feng)

4/1/2020

Question 1

Part 1

First we generate data for Part1.

```
#generate data
library(rpart)
library(tidyverse)
```

```
## — Attaching packages tidyverse
1.3.0 —
```

```
## ✓ ggplot2 3.3.0      ✓ purrr   0.3.3
## ✓ tibble  3.0.0       ✓ dplyr    0.8.5
## ✓ tidyr   1.0.2       ✓ stringr 1.4.0
## ✓ readr   1.3.1       ✓ forcats 0.5.0
```

```
## Warning: package 'tibble' was built under R version 3.6.2
```

```
## — Conflicts tidyverse_conflicts() —
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()
```

```
library(splines)
n=300
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)
number_of_weak_learners = 100
number_of_knots_split = 6
polynomial_degree = 2
fit = rpart(y~x,data=df)
```

For Q0 and Q1, define a function called plot_fitted.

```

plot_fitted <- function(v,fit)
{number_of_weak_learners = 100
yp = predict(fit,newdata=df)
df$yr = df$y - v*yp
YP = v*yp
list_of_weak_learners = list(fit)
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(yr~x,data=df)

  # Generate new prediction
  yp=predict(fit,newdata=df)

  # Update residuals
  df$yr=df$yr - v*yp

  # Bind to new data point
  YP = cbind(YP,v*yp)

  # Store fitted model in list
  list_of_weak_learners[[t]] = fit
}

#####
##### Getting predictions for each boost #####
#####
for (i in 1:number_of_weak_learners){
  # Calculating performance of first i weak_learners

  # Summing weak learner residuals
  if(i==1){yp_i = YP[,1:i]
}else{yp_i=apply(YP[,1:i],1,sum) #<- strong learner
}

  # Binds new cols
  col_name = paste0('yp_',i)
  df = df %>% bind_cols(yp=yp_i)
}

df111 = df[,c(-2,-3)]
# Re-arrange sequences to get pseudo residuals
plot_wl = df111 %>%
  pivot_longer(cols = starts_with("yp")) %>%
  mutate(learner = str_match(name,"[0-9]+")) %>%
  mutate(learner = as.integer(ifelse(is.na(learner),0,learner)))

# Plot final learner
final_learner = plot_wl %>% filter(learner == (number_of_weak_learners-1))

# Plot progression of learner
ggplot() +
  # Visualizing all learners

```

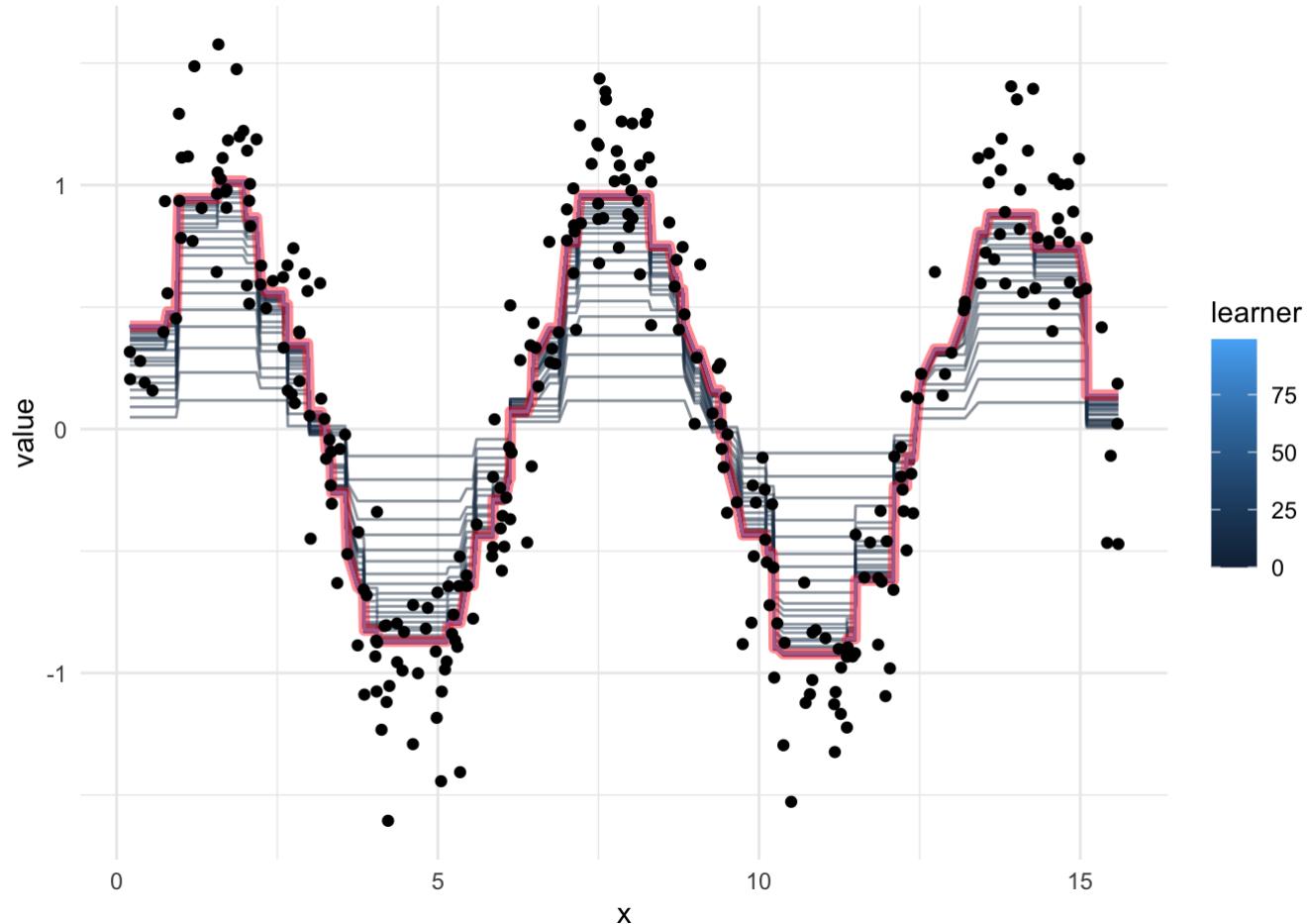
```

geom_line(aes(x = x, y = value, group = learner, color =learner),
          data = plot_wl,alpha=0.5) +
# Final learner
geom_line(aes(x = x, y = value, group = learner, color =learner),
          data = final_learner,alpha=0.5,color = 'firebrick1',size = 2) +
geom_point(aes(x = x, y= y),data = df)+ # true values
theme_minimal()
}

```

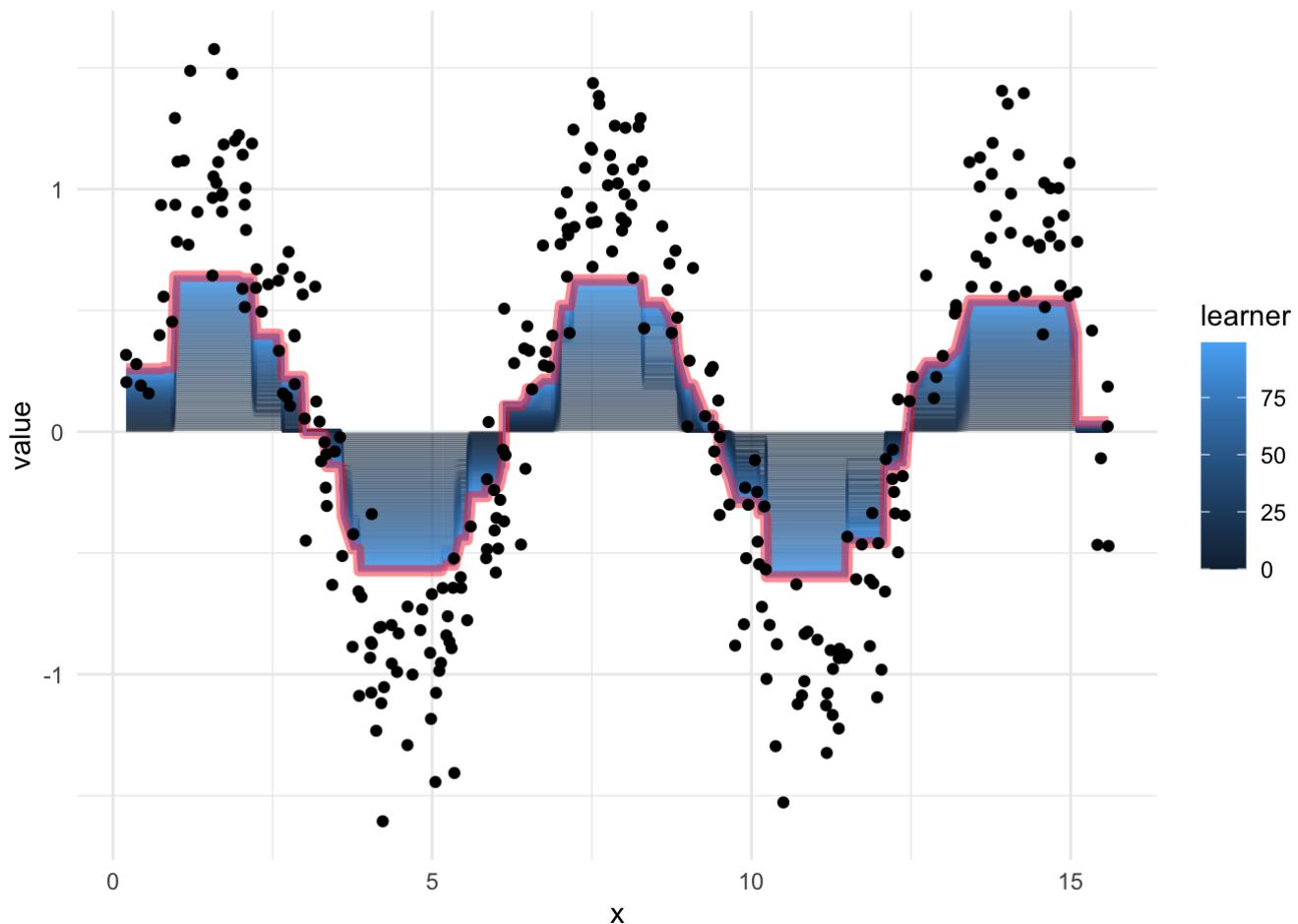
Q0

```
plot_fitted(0.125,fit)
```

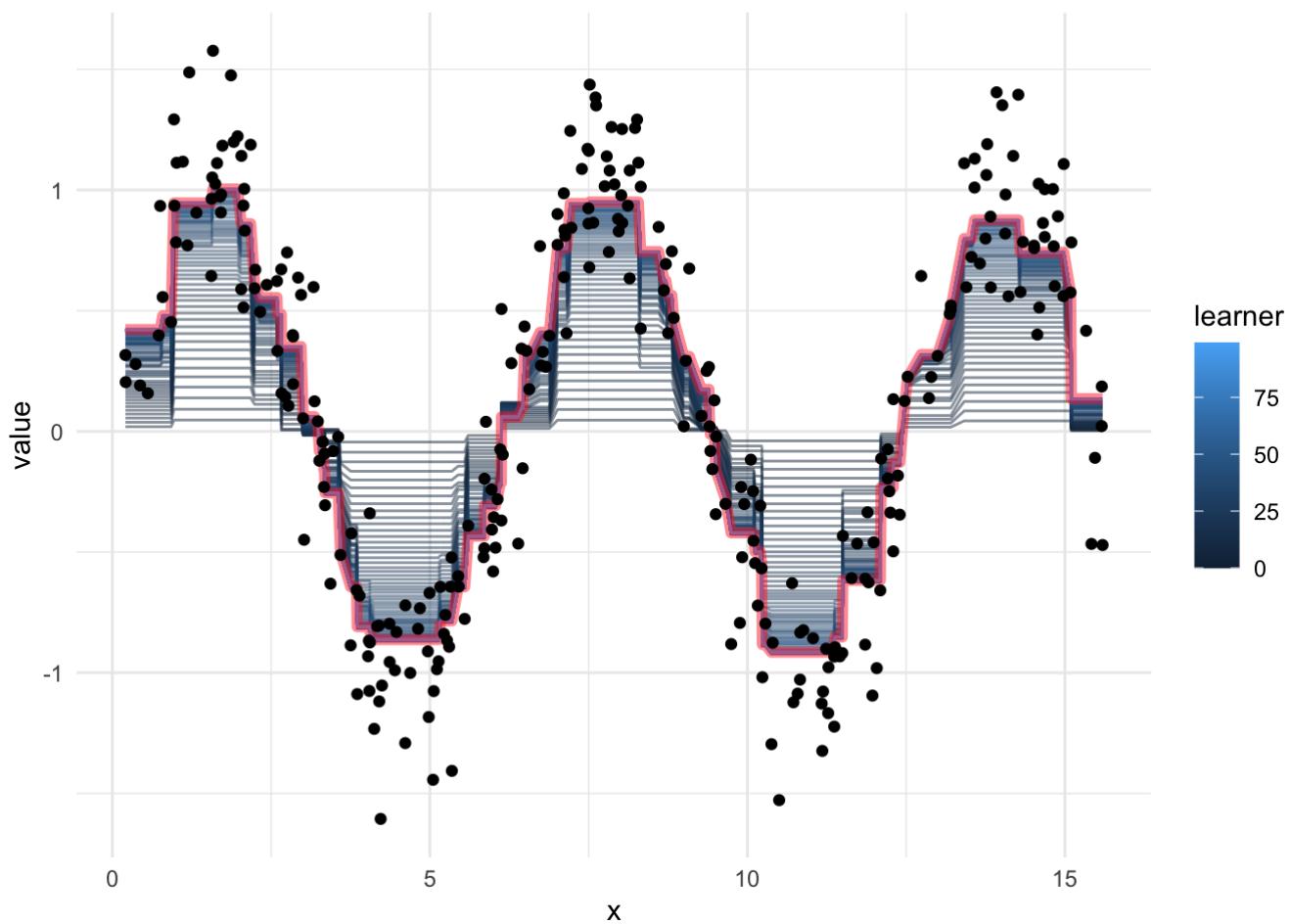
**Q1**

$v=0.125$ is plotted in Q0. Here we plot for $v=0.01$ and $v=0.05$.

```
plot_fitted(0.01,fit)
```



```
plot_fitted(0.05,fit)
```



Q2

A&B

My method is to measure the difference between previous RMSE and current RMSE, if the difference is less than 0.01, then make an early stop.

```

rm(list=ls())
n=500
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)
idx=sample(seq_len(n), size =50)
test <- df[idx, ]
remain <- df[-idx, ]
df<-remain
row.names(df) <- NULL
valid_idx=sample(seq_len(450), size =50)
valid<-df[valid_idx,]
remain<-df[-valid_idx,]
df<-remain
row.names(df) <- NULL

fit_early_stop<-function(model){
diff = 100000000000
t=1
v=.05

yp = predict(model,newdata=df)
df$yr = df$y - v*yp
YP = v*yp
list_of_weak_learners = list(model)
y_valid_p=v*predict(model,newdata=valid)
y_rmse_valid=sqrt(mean(abs(valid$y-y_valid_p)**2))
while (diff>0.01)
{
  t=t+1
  fit = rpart(yr ~ x,data=df)
  yp=predict(fit,newdata=df)
  list_of_weak_learners[[t]] = fit
  df$yr=df$yr - v*yp
  YP = cbind(YP,v*yp)
  y_valid_p=y_valid_p+v*predict(fit,newdata=valid)
  y_rmse_valid_new=sqrt(mean(abs(valid$y-y_valid_p)**2))
  diff=abs(y_rmse_valid_new-y_rmse_valid)
  y_rmse_valid=y_rmse_valid_new

}

t=t-1

#####
##### Getting predictions for each boost #####
#####

for (i in 1:t){
  # Calculating performance of first i weak_learners
}

```

```

# Summing weak learner residuals
if(i==1){yp_i = YP[,1:i]
}else{yp_i=apply(YP[,1:i],1,sum) #<- strong learner
}
# Binds new cols
col_name = paste0('yp_',i)
df = df %>% bind_cols(yp=yp_i)
}
df111 = df[,c(-2,-3)]
# Re-arrange sequences to get pseudo residuals
plot_wl = df111 %>%
  pivot_longer(cols = starts_with("yp")) %>%
  mutate(learner = str_match(name,"[0-9]+")) %>%
  mutate(learner = as.integer(ifelse(is.na(learner),0,learner)))

# Plot final learner
final_learner = plot_wl %>% filter(learner == (t-1))

# Plot progression of learner
ggplot() +
  # Visualizing all learners
  geom_line(aes(x = x, y = value, group = learner, color =learner),
            data = plot_wl,alpha=0.5) +
  # Final learner
  geom_line(aes(x = x, y = value, group = learner, color =learner),
            data = final_learner,alpha=0.5,color = 'firebrick1',size = 2) +
  geom_point(aes(x = x, y= y),data = df)+ # true values
  ggtitle('Plot progression of learner with original learning parameter v=0.05')+
  theme_minimal()
cat('The number of trees is',t)
return(list_of_weak_learners)}

model=rpart(y~x,data=df)
list_of_weak_learners = fit_early_stop(model)

```

```
## The number of trees is 19
```

C

```

v=0.05
t=18
for (i in 1:18){
  weak_learner_i = list_of_weak_learners[[i]]

  if (i==1){pred = v*predict(weak_learner_i,test)}
  else{pred = pred + v*predict(weak_learner_i,test)}

  if(i==t){
    test = test %>% bind_cols(yp=pred)
  }
}
cat('The RMSE is',sqrt(sum((test$yp-test$y)**2))/50)

```

```
## The RMSE is 0.06358244
```

Q3

In this part, I will use grid-search to tune 3 parameters: minsplit,cp and maxdepth.

```

rm(list=ls())
n=500
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)
idx=sample(seq_len(n), size =50)
test <- df[idx, ]
remain <- df[-idx,]
df<-remain
row.names(df) <- NULL
valid_idx=sample(seq_len(450), size =50)
valid<-df[valid_idx,]
remain<-df[-valid_idx,]
df<-remain
row.names(df) <- NULL

fit_grid_search<-function(minsplit,cp,maxdepth){
model=rpart(y~x,data=df,control = rpart.control(minsplit = minsplit,cp = cp,maxdepth = m
axdepth))
list_of_weak_learners = list(model)
v=.05
number_of_weak_learners = 100
yp = predict(model,newdata=df)
df$yr = df$y - v*yp
YP = v*yp
list_of_weak_learners = list(model)
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(yr~x,data=df,control = rpart.control(minsplit = minsplit,cp = cp,maxdepth
= maxdepth))

  # Generate new prediction
  yp=predict(fit,newdata=df,control = rpart.control(minsplit = minsplit,cp = cp,maxdepth
= maxdepth))

  # Update residuals
  df$yr=df$yr - v*yp

  # Bind to new data point
  YP = cbind(YP,v*yp)

  # Store fitted model in list
  list_of_weak_learners[[t]] = fit
}

for (i in 1:t){
  weak_learner_i = list_of_weak_learners[[i]]

  if (i==1){pred = v*predict(weak_learner_i,test)}
  else{pred = pred + v*predict(weak_learner_i,test)}
}

```

```
if(i==t){  
  test = test %>% bind_cols(yp=pred)  
}  
}  
return(sqrt(sum((test$yp-test$y)**2)))  
  
}  
  
  
res = data.frame(minsplit = rep(0,27),cp = rep(0,27),maxdepth = rep(0,27),RMSE = rep(0,27))  
  
i=1  
for (p1 in c(5,20,50)){  
  for (p2 in c(0.1,0.01,0.001)){  
    for (p3 in c(5,10,30)){  
      res[i,] = c(p1,p2,p3,fit_grid_search(p1,p2,p3)/50)  
      i=i+1  
    }  
  }  
}
```

```
res
```

```

##   minsplit    cp maxdepth      RMSE
## 1      5 0.100      5 0.06683252
## 2      5 0.100     10 0.06683252
## 3      5 0.100     30 0.06683252
## 4      5 0.010      5 0.04643182
## 5      5 0.010     10 0.04591451
## 6      5 0.010     30 0.04591451
## 7      5 0.001      5 0.04536585
## 8      5 0.001     10 0.05088259
## 9      5 0.001     30 0.05310488
## 10     20 0.100      5 0.06683252
## 11     20 0.100     10 0.06683252
## 12     20 0.100     30 0.06683252
## 13     20 0.010      5 0.04628380
## 14     20 0.010     10 0.04560248
## 15     20 0.010     30 0.04560248
## 16     20 0.001      5 0.04420488
## 17     20 0.001     10 0.04573664
## 18     20 0.001     30 0.04586075
## 19     50 0.100      5 0.06683252
## 20     50 0.100     10 0.06683252
## 21     50 0.100     30 0.06683252
## 22     50 0.010      5 0.04648577
## 23     50 0.010     10 0.04631201
## 24     50 0.010     30 0.04631201
## 25     50 0.001      5 0.04555101
## 26     50 0.001     10 0.04545742
## 27     50 0.001     30 0.04545742

```

```
cat('Best Parameters are:', '\n')
```

```
## Best Parameters are:
```

```
res[res$RMSE==min(res$RMSE), ]
```

```

##   minsplit    cp maxdepth      RMSE
## 16     20 0.001      5 0.04420488

```

Part 2

Q0&Q1

```

df = read.csv('/Users/zmt/Desktop/Assignment3/kernel_regression_2.csv')
colnames(df) = c('x1','x2','y')
plot_fitted <- function(v,fit)
{number_of_weak_learners = 100
yp = predict(fit,newdata=df)
df$yr = df$y - v*yp
YP = v*yp
list_of_weak_learners = list(fit)
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(yr~x1+x2,data=df)

  # Generate new prediction
  yp=predict(fit,newdata=df)

  # Update residuals
  df$yr=df$yr - v*yp

  # Bind to new data point
  YP = cbind(YP,v*yp)

  # Store fitted model in list
  list_of_weak_learners[[t]] = fit
}

#####
##### Getting predictions for each boost #####
#####
for (i in 1:number_of_weak_learners){
  # Calculating performance of first i weak_learners

  # Summing weak learner residuals
  if(i==1){yp_i = YP[,1:i]
  }else{yp_i=apply(YP[,1:i],1,sum) #<- strong learner
  }

  # Binds new cols
  col_name = paste0('yp_',i)
  df = df %>% bind_cols(yp=yp_i)
}

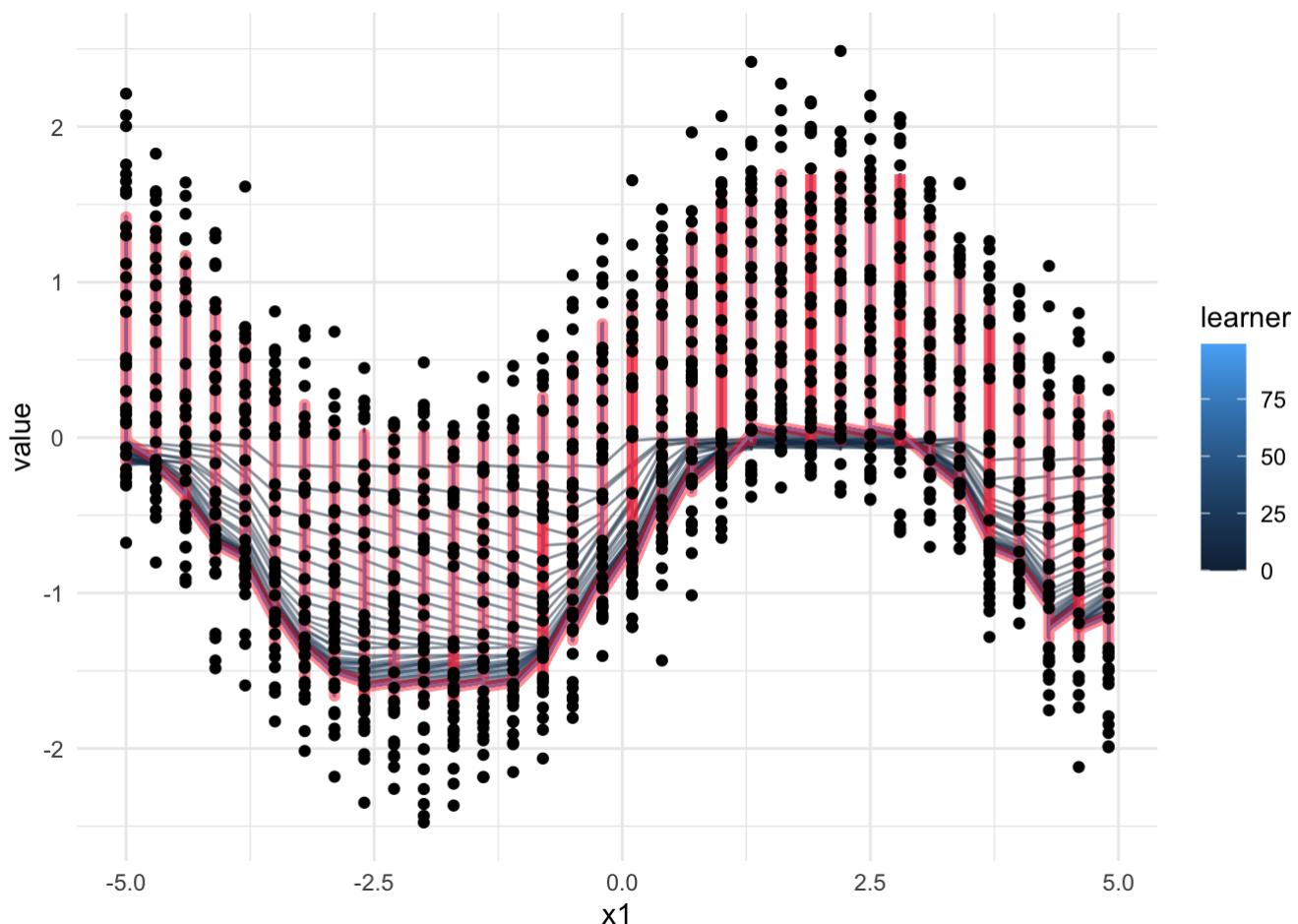
df111 = df[,c(-2,-3)]
# Re-arrange sequences to get pseudo residuals
plot_wl = df111 %>%
  pivot_longer(cols = starts_with("yp")) %>%
  mutate(learner = str_match(name,"[0-9]+")) %>%
  mutate(learner = as.integer(ifelse(is.na(learner),0,learner)))

# Plot final learner
final_learner = plot_wl %>% filter(learner == (number_of_weak_learners-1))

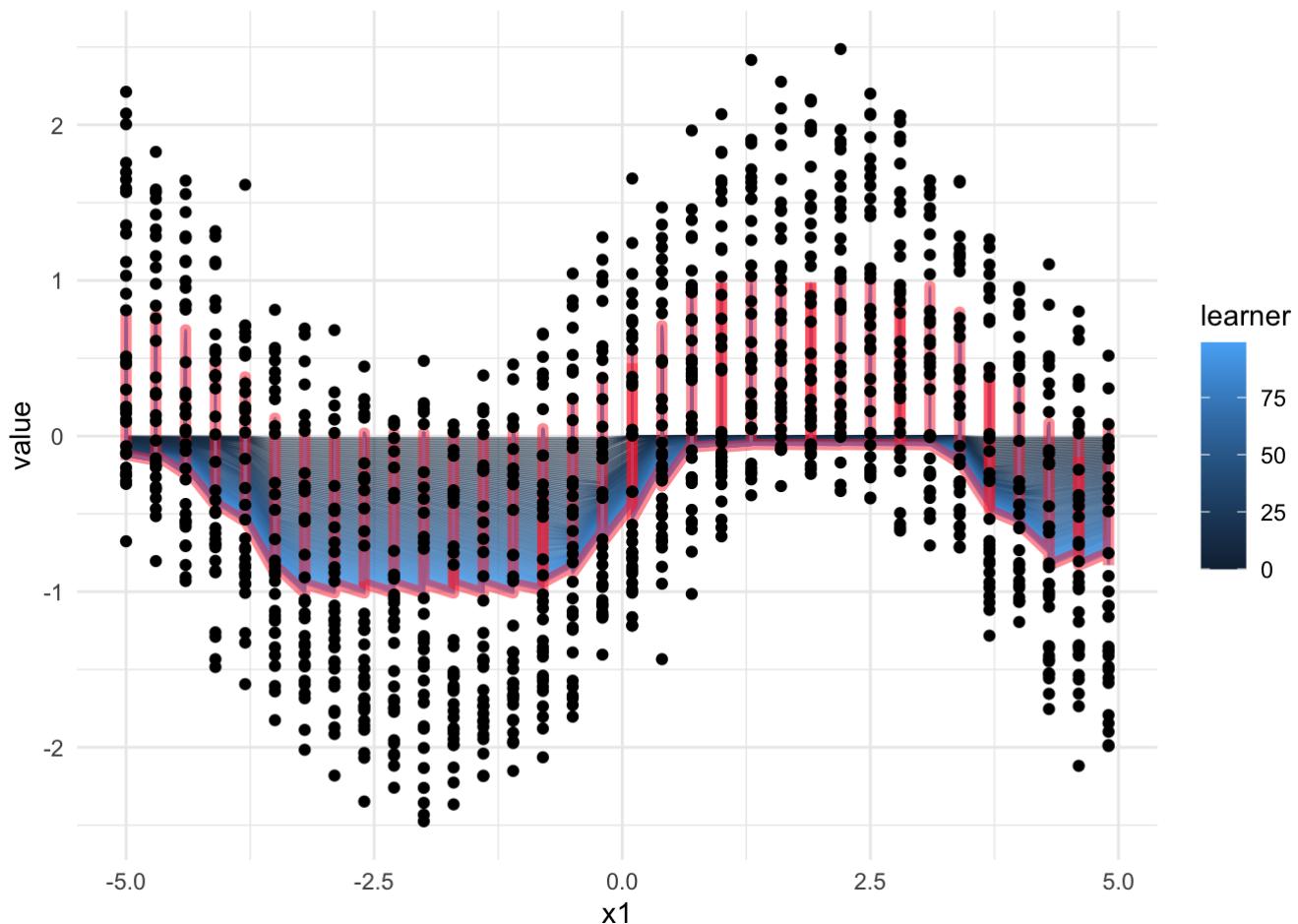
# Plot progression of learner

```

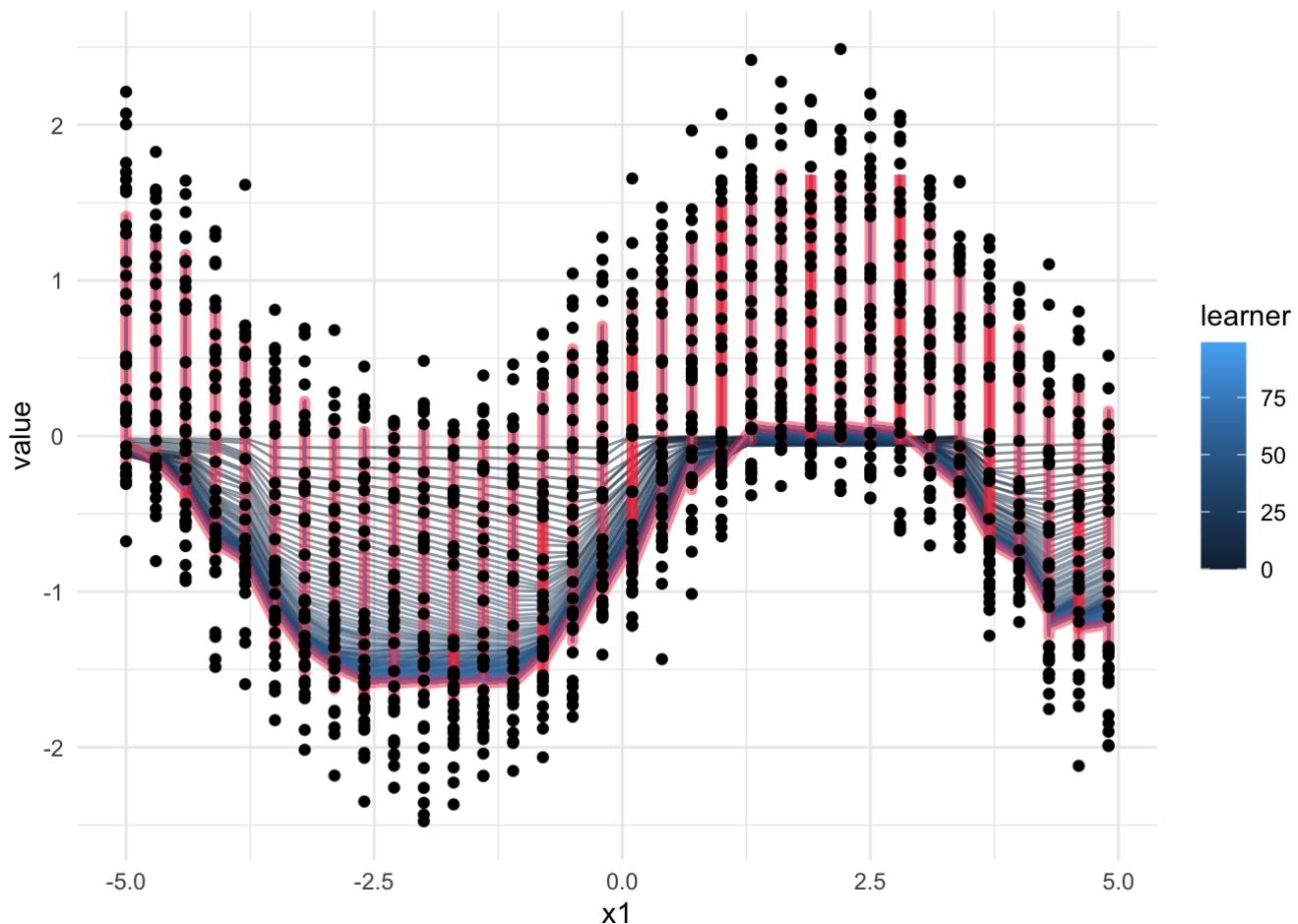
```
ggplot() +  
  # Visualizing all learners  
  geom_line(aes(x = x1, y = value, group = learner, color =learner),  
            data = plot_wl,alpha=0.5) +  
  # Final learner  
  geom_line(aes(x = x1, y = value, group = learner, color =learner),  
            data = final_learner,alpha=0.5,color = 'firebrick1',size = 2) +  
  geom_point(aes(x = x1, y= y),data = df)+ # true values  
  theme_minimal()  
}  
  
model = rpart(y~x1+x2,data=df)  
plot_fitted(0.125,fit=model)
```



```
plot_fitted(0.01,fit=model)
```



```
plot_fitted(0.05,fit=model)
```



Q2

A&B

```

rm(list=ls())
df = read.csv('/Users/zmt/Desktop/Assignment3/kernel_regression_2.csv')
n=length(df[,1])
colnames(df) = c('x1','x2','y')
idx=sample(seq_len(n), size =50)
test <- df[idx, ]
remain <- df[-idx,]
df<-remain
row.names(df) <- NULL
valid_idx=sample(seq_len(n-50), size =50)
valid<-df[valid_idx,]
remain<-df[-valid_idx,]
df<-remain
row.names(df) <- NULL
fit_early_stop<-function(model){
diff = 100000000000
t=1
v=.05

yp = predict(model,newdata=df)
df$yr = df$y - v*yp
YP = v*yp
list_of_weak_learners = list(model)
y_valid_p=v*predict(model,newdata=valid)
y_rmse_valid=sqrt(mean(abs(valid$y-y_valid_p)**2))
while (diff>0.01)
{
  t=t+1
  fit = rpart(yr ~ x1+x2,data=df)
  yp=predict(fit,newdata=df)
  list_of_weak_learners[[t]] = fit
  df$yr=df$yr - v*yp
  YP = cbind(YP,v*yp)
  y_valid_p=y_valid_p+v*predict(fit,newdata=valid)
  y_rmse_valid_new=sqrt(mean(abs(valid$y-y_valid_p)**2))
  diff=abs(y_rmse_valid_new-y_rmse_valid)
  y_rmse_valid=y_rmse_valid_new

}
t=t-1
#####
##### Getting predictions for each boost #####
#####
for (i in 1:t){
  # Calculating performance of first i weak_learners

  # Summing weak learner residuals
  if(i==1){yp_i = YP[,1:i]
  }else{yp_i=apply(YP[,1:i],1,sum) #<- strong learner
  }
  # Binds new cols
}

```

```

col_name = paste0('yp_',i)
df = df %>% bind_cols(yp=yp_i)
}
df111 = df[,c(-2,-3)]
# Re-arrange sequences to get pseudo residuals
plot_wl = df111 %>%
  pivot_longer(cols = starts_with("yp")) %>%
  mutate(learner = str_match(name,"[0-9]+")) %>%
  mutate(learner = as.integer(ifelse(is.na(learner),0,learner)))

# Plot final learner
final_learner = plot_wl %>% filter(learner == (t-1))

# Plot progression of learner
ggplot() +
  # Visualizing all learners
  geom_line(aes(x = x, y = value, group = learner, color =learner),
            data = plot_wl,alpha=0.5) +
  # Final learner
  geom_line(aes(x = x, y = value, group = learner, color =learner),
            data = final_learner,alpha=0.5,color = 'firebrick1',size = 2) +
  geom_point(aes(x = x, y= y),data = df)+ # true values
  ggtitle('Plot progression of learner with original learning paremater v=0.05')+ 
  theme_minimal()
cat('The number of trees is',t)
return(list_of_weak_learners)}
```

```

model=rpart(y~x1+x2,data=df)
list_of_weak_learners = fit_early_stop(model)
```

```
## The number of trees is 22
```

C

```

v=0.05
t=18
for (i in 1:18){
  weak_learner_i = list_of_weak_learners[[i]]

  if (i==1){pred = v*predict(weak_learner_i,test)}
  else{pred = pred + v*predict(weak_learner_i,test)}

  if(i==t){
    test = test %>% bind_cols(yp=pred)
  }
}
cat('The RMSE is',sqrt(sum((test$yp-test$y)**2))/100)
```

```
## The RMSE is 0.03939153
```

Q3

```

fit_grid_search<-function(minsplit,cp,maxdepth){
model=rpart(y~x1+x2,data=df,control = rpart.control(minsplit = minsplit,cp = cp,maxdepth
= maxdepth))
list_of_weak_learners = list(model)
v=.05
number_of_weak_learners = 100
yp = predict(model,newdata=df)
df$yr = df$y - v*yp
YP = v*yp
list_of_weak_learners = list(model)
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(yr~x1+x2,data=df,control = rpart.control(minsplit = minsplit,cp = cp,maxde
pth = maxdepth))

  # Generate new prediction
  yp=predict(fit,newdata=df,control = rpart.control(minsplit = minsplit,cp = cp,maxdepth
= maxdepth))

  # Update residuals
  df$yr=df$yr - v*yp

  # Bind to new data point
  YP = cbind(YP,v*yp)

  # Store fitted model in list
  list_of_weak_learners[[t]] = fit
}

for (i in 1:t){
  weak_learner_i = list_of_weak_learners[[i]]

  if (i==1){pred = v*predict(weak_learner_i,test)}
  else{pred = pred + v*predict(weak_learner_i,test)}

  if(i==t){
    test = test %>% bind_cols(yp=pred)
  }
}
return(sqrt(sum((test$yp-test$y)**2))/100)
}

res = data.frame(minsplit = rep(0,27),cp = rep(0,27),maxdepth = rep(0,27),RMSE = rep(0,2
7))

i=1
for (p1 in c(2,5,10)){
  for (p2 in c(0.01,0.1,0.001)){
    for (p3 in c(5,10,30)){

```

```

    res[i,] = c(p1,p2,p3,fit_grid_search(p1,p2,p3))
    i=i+1
}

}

```

```
cat('Best Parameters are:', '\n')
```

```
## Best Parameters are:
```

```
res[res$RMSE==min(res$RMSE),][1,]
```

```
##   minsplit   cp maxdepth      RMSE
## 1        2  0.01        5 0.03939153
```

The best set is minsplit=2,cp=0.01,maxdepth=5

Question 2

Part 1

- a. Yes. The goal of t-SNE is to maintain the local structure of data, so the distance between points after mapping somewhat indicates their distance before mapping.
- b. Perplexity is a guess about the number of close neighbors each point has. It is used to balance attention between local and global aspects of your data.
- c. If the number of steps is too small, then t-SNE will end before it converges. From the article, we know if you see a t-SNE plot with strange “pinched” shapes, chances are the process was stopped too early. Unfortunately, there's no fixed number of steps that yields a stable result, which is because different data requires different numbers of iterations to converge.
- d. There are two examples listed in the article. One of the simplest topological properties is containment. t-SNE with low perplexity value greatly exaggerates the size of the smaller group of points. Another is trefoil knot, low perplexity values failed to show global connectivity.

Part 2

```

library(tidyverse)
library(Rtsne)
library(RColorBrewer)

```

```
# Get MNIST data
mnist_raw <- read_csv("https://pjreddie.com/media/files/mnist_train.csv", col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )
```

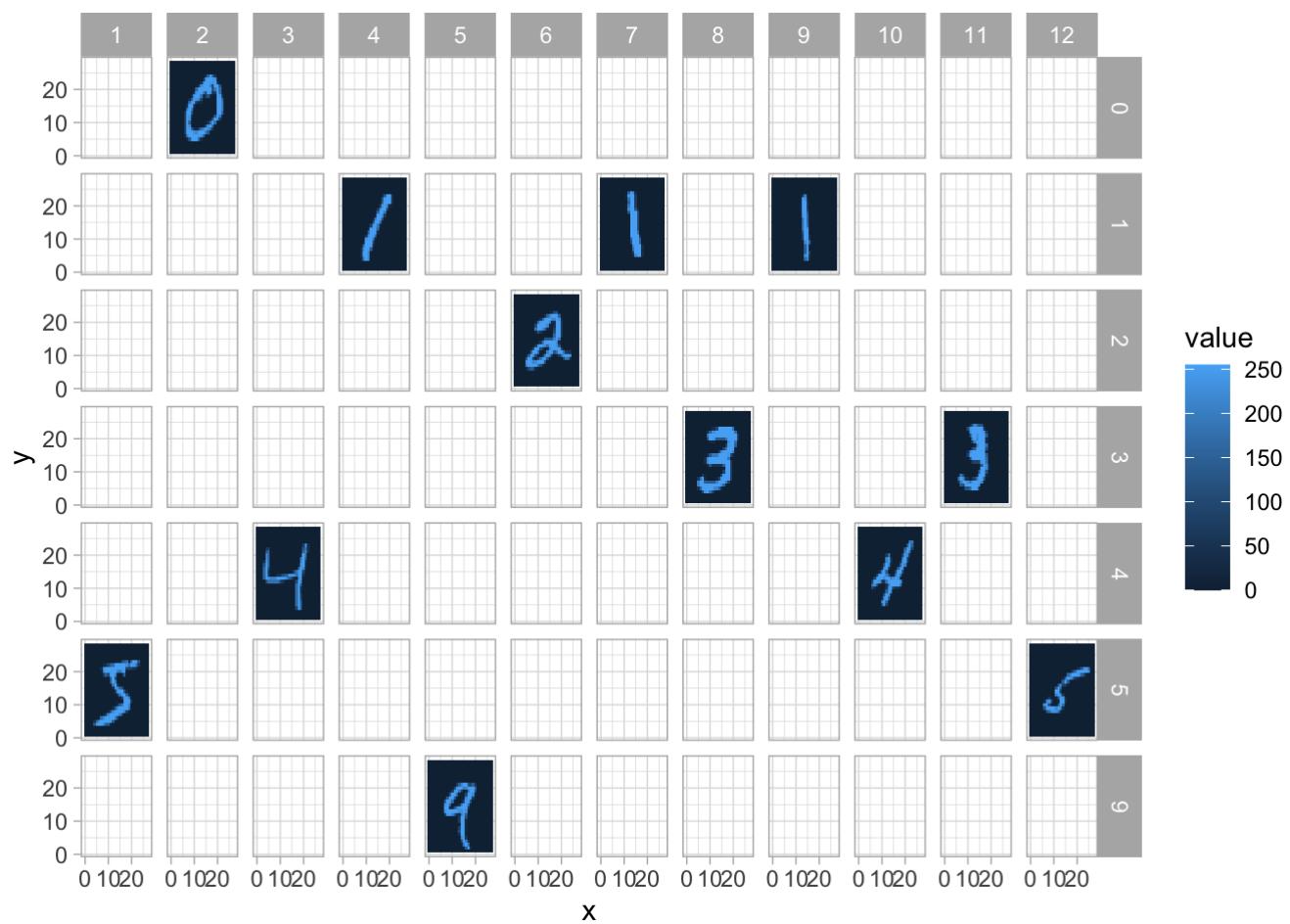
```
## See spec(...) for full column specifications.
```

```
# What is the dimension of the data set  
dim(mnist_raw) # first column is the value, the rest are the pixels
```

```
## [1] 60000 785
```

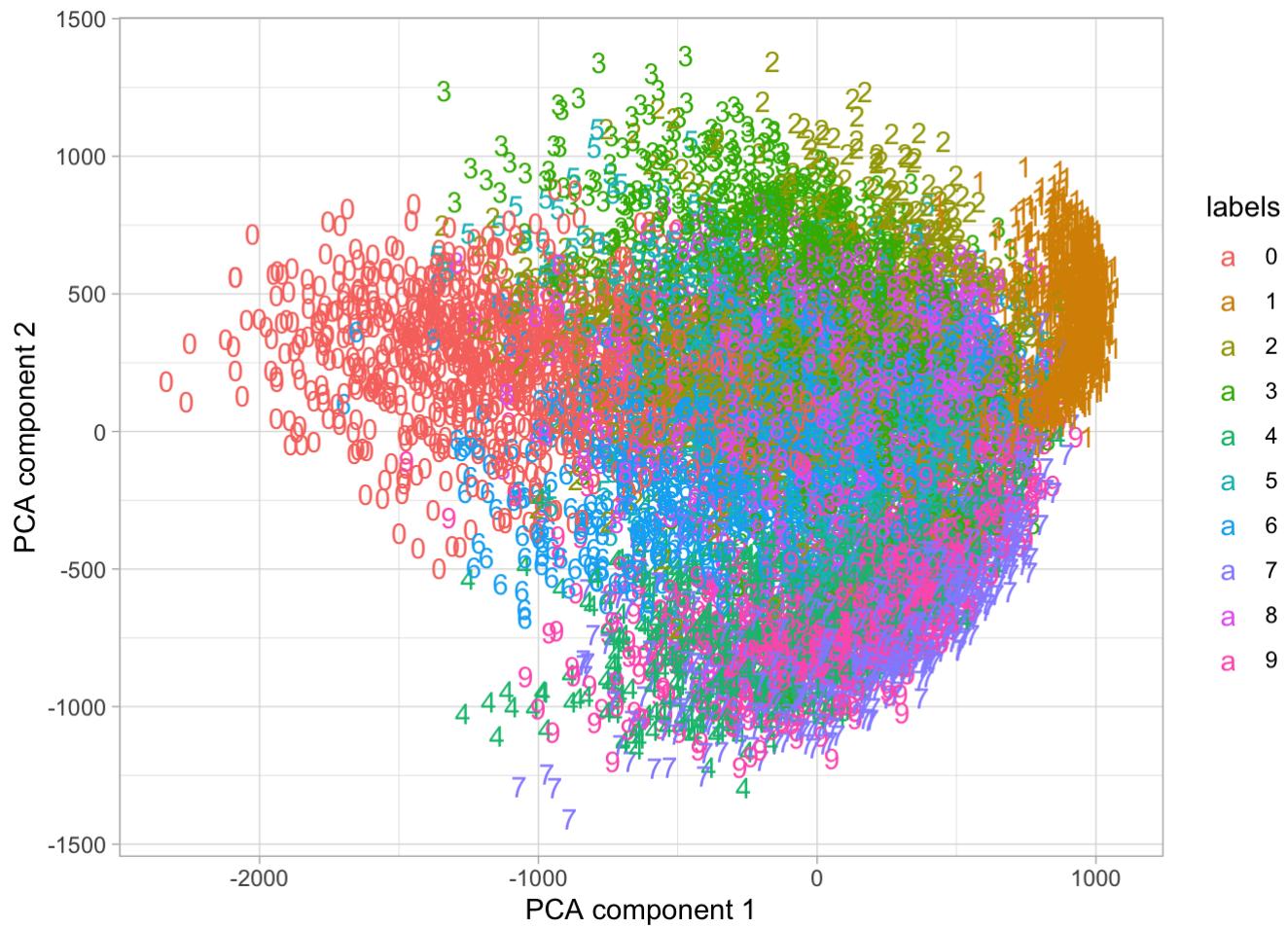
```
# Rearranging the data  
pixels_gathered <- mnist_raw %>% head(10000) %>%  
  rename(label = X1) %>%  
  mutate(instance = row_number()) %>%  
  gather(pixel, value, -label, -instance) %>%  
  extract(pixel, "pixel", "(\\d+)", convert = TRUE) %>%  
  mutate(pixel = pixel - 2,  
         x = pixel %% 28,  
         y = 28 - pixel %/% 28)  
  
first_10k_samples = mnist_raw[1:10000,-1] #>%> as.matrix()  
first_10k_samples_labels = mnist_raw[1:10000,1] %>% unlist(use.names=F)  
colors = brewer.pal(10, 'Spectral')
```

```
# Visualizing the data  
theme_set(theme_light())  
pixels_gathered %>%  
  filter(instance <= 12) %>%  
  ggplot(aes(x, y, fill = value)) +  
  geom_tile() +  
  facet_grid(label ~ instance )
```



a.

```
#####
##### Visualizing the PCA decomposition #####
#####
pca = princomp(first_10k_samples)$scores[,1:2]
pca_plot = tibble(x = pca[,1], y = pca[,2], labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y = y, label = labels, color = labels), data = pca_plot) + geom_text() +
  xlab('PCA component 1') + ylab('PCA component 2')
```



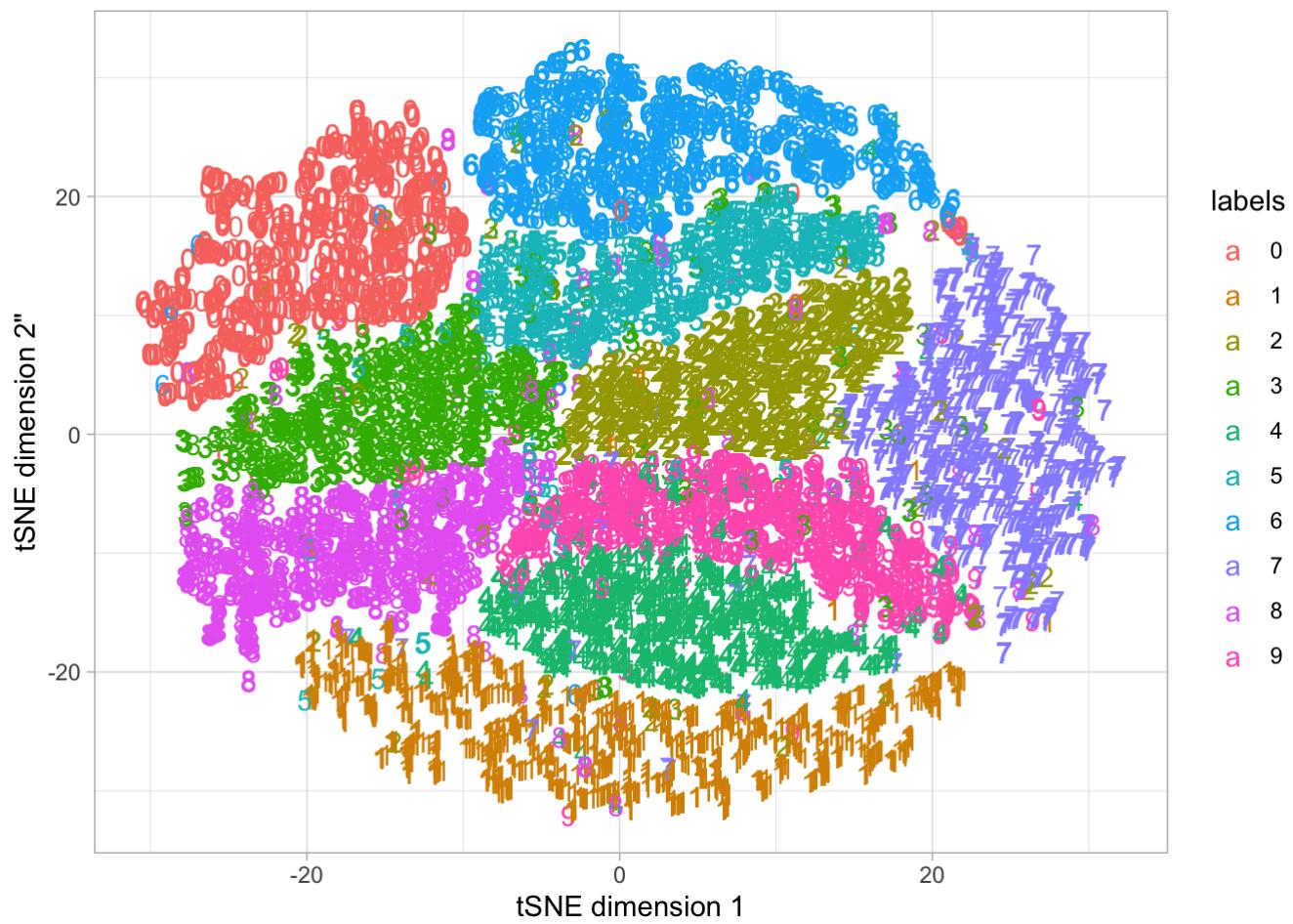
b.

```
#####
#####      Running the TSNE embedding      #####
#####

itercost = c()
embedding = Rtsne(X = first_10k_samples, dims = 2,
                    perplexity = 5,
                    theta = 0.5,
                    eta = 200,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 5.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## - point 10000 of 10000
## Done in 12.44 seconds (sparsity = 0.002104)!
## Learning embedding...
## Iteration 50: error is 118.296633 (50 iterations in 2.11 seconds)
## Iteration 100: error is 107.234275 (50 iterations in 2.58 seconds)
## Iteration 150: error is 99.561972 (50 iterations in 1.96 seconds)
## Iteration 200: error is 96.723478 (50 iterations in 1.80 seconds)
## Iteration 250: error is 95.000359 (50 iterations in 1.73 seconds)
## Iteration 300: error is 4.356011 (50 iterations in 1.64 seconds)
## Iteration 350: error is 3.798177 (50 iterations in 1.62 seconds)
## Iteration 400: error is 3.436557 (50 iterations in 1.65 seconds)
## Iteration 450: error is 3.173032 (50 iterations in 1.77 seconds)
## Iteration 500: error is 2.969045 (50 iterations in 1.75 seconds)
## Fitting performed in 18.61 seconds.
```

```
itercost = c(itercost,embedding$itercosts[10])
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2''')
```

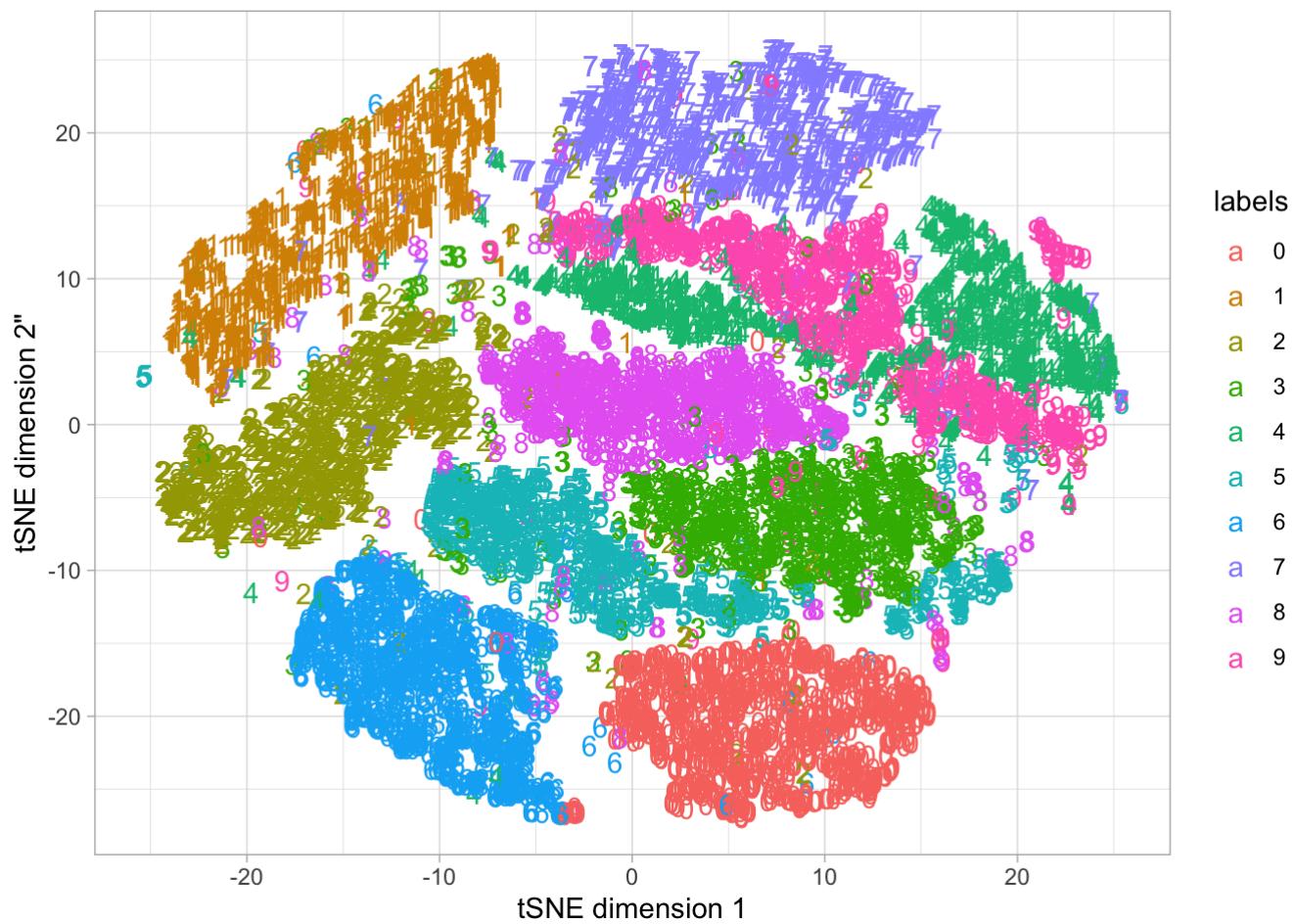


c. As the value of perplexity increases, I can see the distances between clusters also increase, which means the clusters become more dispersed.

```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                    perplexity = 20,
                    theta = 0.5,
                    eta = 200,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 20.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## - point 10000 of 10000
## Done in 16.39 seconds (sparsity = 0.008217)!
## Learning embedding...
## Iteration 50: error is 102.343946 (50 iterations in 2.21 seconds)
## Iteration 100: error is 93.292482 (50 iterations in 2.13 seconds)
## Iteration 150: error is 88.798251 (50 iterations in 2.02 seconds)
## Iteration 200: error is 87.913978 (50 iterations in 2.22 seconds)
## Iteration 250: error is 87.654497 (50 iterations in 2.28 seconds)
## Iteration 300: error is 3.394537 (50 iterations in 1.93 seconds)
## Iteration 350: error is 2.951053 (50 iterations in 1.83 seconds)
## Iteration 400: error is 2.703117 (50 iterations in 1.85 seconds)
## Iteration 450: error is 2.536505 (50 iterations in 1.86 seconds)
## Iteration 500: error is 2.414508 (50 iterations in 1.97 seconds)
## Fitting performed in 20.29 seconds.
```

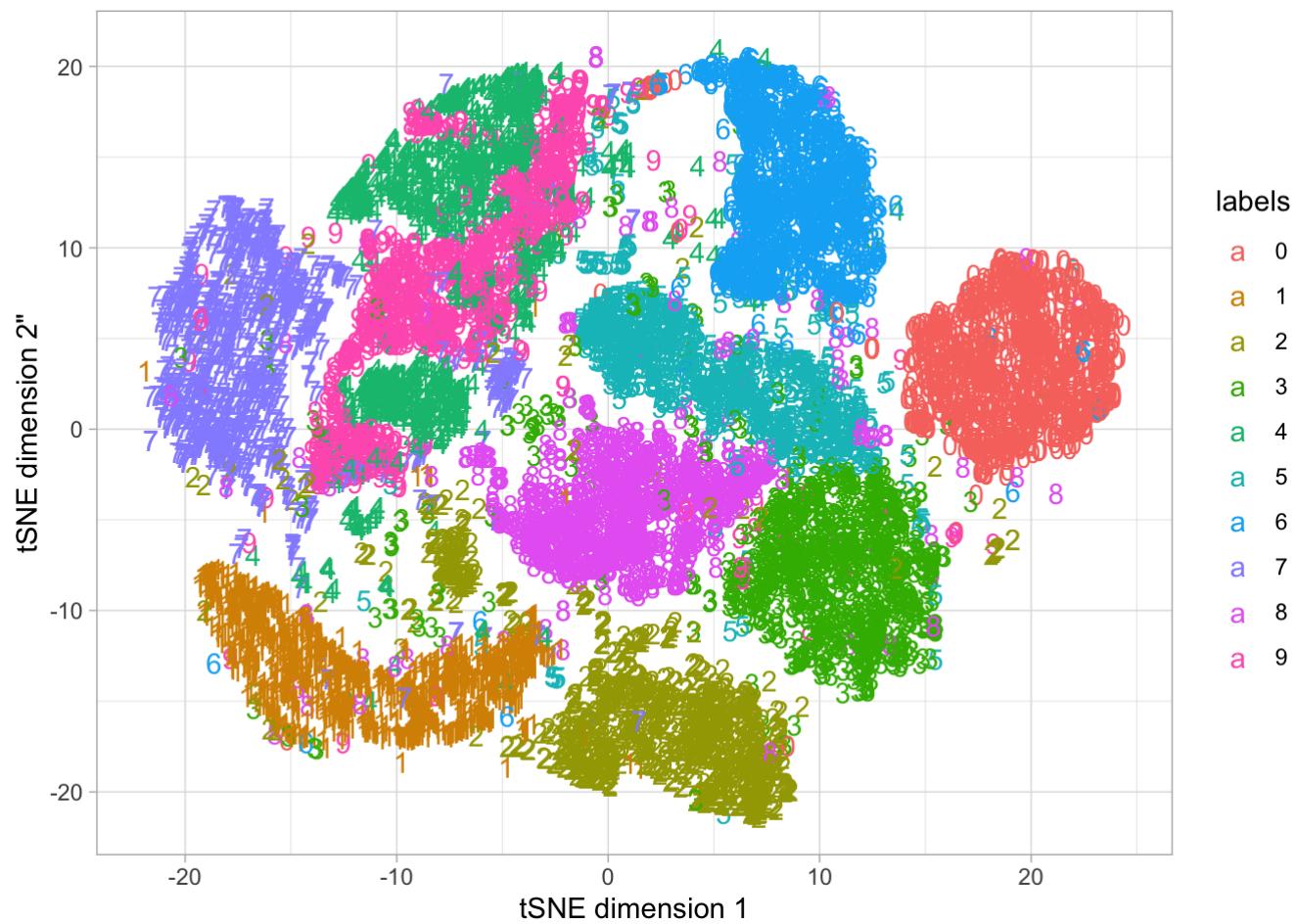
```
itercost = c(itercost,embedding$itercosts[10])
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2''')
```



```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                   perplexity = 60,
                   theta = 0.5,
                   eta = 200,
                   pca = TRUE, verbose = TRUE,
                   max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 60.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## - point 10000 of 10000
## Done in 21.35 seconds (sparsity = 0.024328)!
## Learning embedding...
## Iteration 50: error is 89.428527 (50 iterations in 2.87 seconds)
## Iteration 100: error is 86.712359 (50 iterations in 3.89 seconds)
## Iteration 150: error is 81.658892 (50 iterations in 2.59 seconds)
## Iteration 200: error is 81.469172 (50 iterations in 2.67 seconds)
## Iteration 250: error is 81.450563 (50 iterations in 2.61 seconds)
## Iteration 300: error is 2.627544 (50 iterations in 2.74 seconds)
## Iteration 350: error is 2.282607 (50 iterations in 2.28 seconds)
## Iteration 400: error is 2.107304 (50 iterations in 2.41 seconds)
## Iteration 450: error is 1.997415 (50 iterations in 2.39 seconds)
## Iteration 500: error is 1.920060 (50 iterations in 2.54 seconds)
## Fitting performed in 26.99 seconds.
```

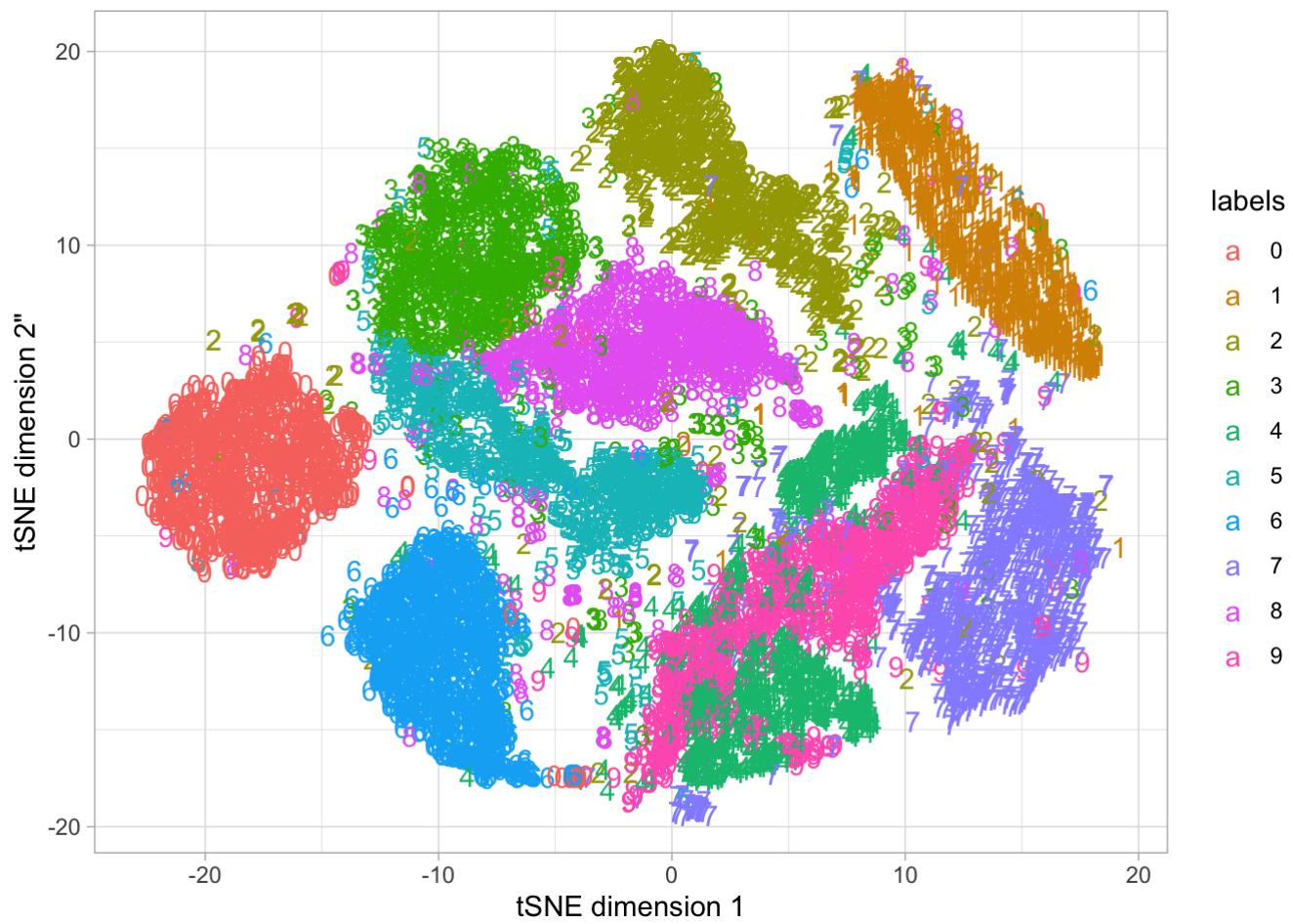
```
itercost = c(itercost,embedding$itercosts[10])
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2''')
```



```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                    perplexity = 100,
                    theta = 0.5,
                    eta = 200,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 100.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## - point 10000 of 10000
## Done in 22.85 seconds (sparsity = 0.040571)!
## Learning embedding...
## Iteration 50: error is 83.371128 (50 iterations in 3.43 seconds)
## Iteration 100: error is 81.959169 (50 iterations in 5.31 seconds)
## Iteration 150: error is 78.550870 (50 iterations in 4.26 seconds)
## Iteration 200: error is 78.344096 (50 iterations in 3.37 seconds)
## Iteration 250: error is 78.296656 (50 iterations in 3.30 seconds)
## Iteration 300: error is 2.261678 (50 iterations in 2.77 seconds)
## Iteration 350: error is 1.968606 (50 iterations in 2.89 seconds)
## Iteration 400: error is 1.825864 (50 iterations in 2.86 seconds)
## Iteration 450: error is 1.738897 (50 iterations in 3.10 seconds)
## Iteration 500: error is 1.680204 (50 iterations in 2.87 seconds)
## Fitting performed in 34.16 seconds.
```

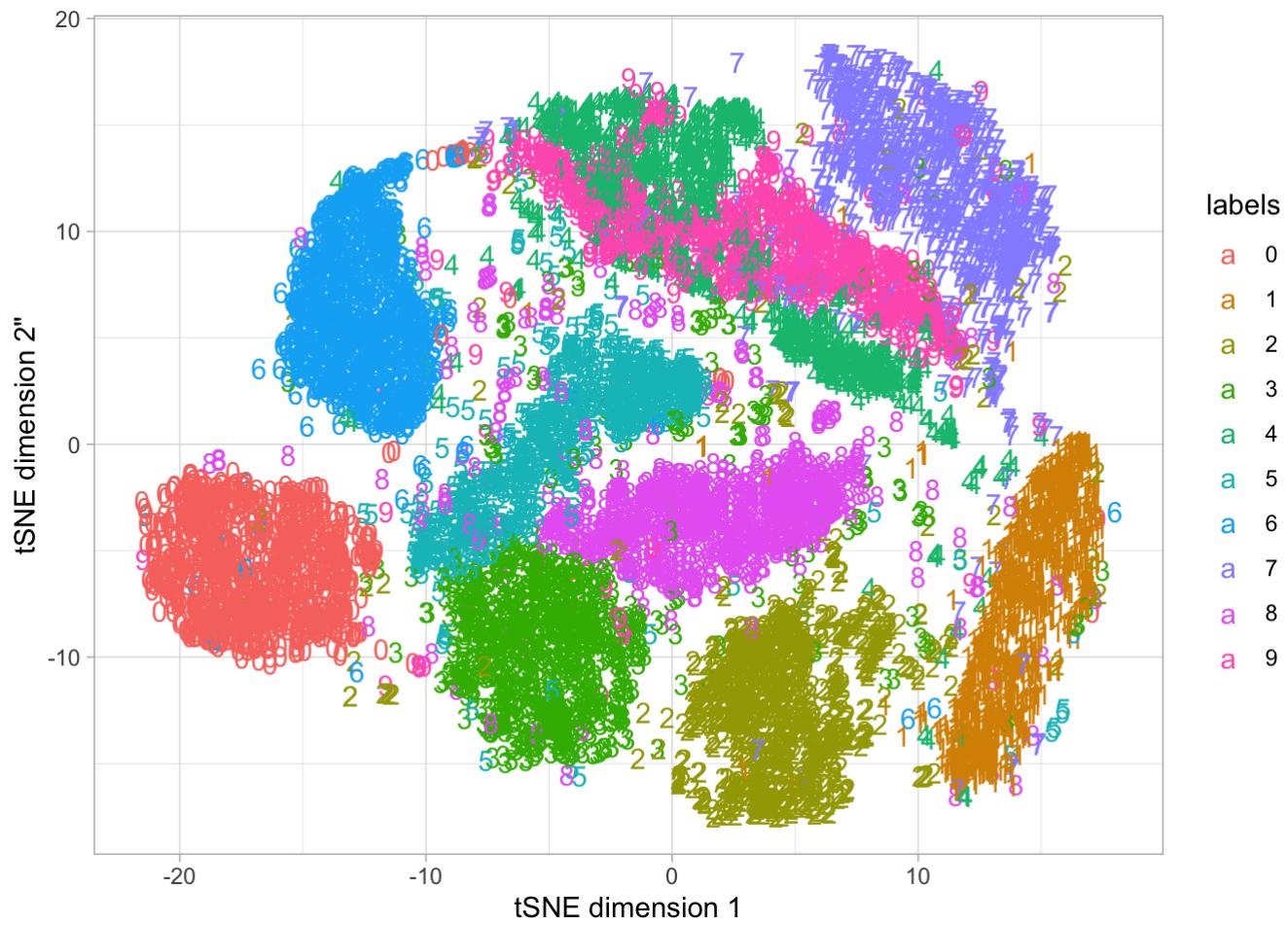
```
itercost = c(itercost,embedding$itercosts[10])
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2''')
```



```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                   perplexity = 125,
                   theta = 0.5,
                   eta = 200,
                   pca = TRUE, verbose = TRUE,
                   max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 125.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## - point 10000 of 10000
## Done in 27.66 seconds (sparsity = 0.050806)!
## Learning embedding...
## Iteration 50: error is 80.714217 (50 iterations in 3.31 seconds)
## Iteration 100: error is 80.363285 (50 iterations in 3.28 seconds)
## Iteration 150: error is 76.794677 (50 iterations in 3.13 seconds)
## Iteration 200: error is 76.731696 (50 iterations in 3.10 seconds)
## Iteration 250: error is 76.727369 (50 iterations in 3.14 seconds)
## Iteration 300: error is 2.153940 (50 iterations in 2.93 seconds)
## Iteration 350: error is 1.865270 (50 iterations in 2.95 seconds)
## Iteration 400: error is 1.727065 (50 iterations in 2.93 seconds)
## Iteration 450: error is 1.645474 (50 iterations in 3.39 seconds)
## Iteration 500: error is 1.591008 (50 iterations in 3.70 seconds)
## Fitting performed in 31.86 seconds.
```

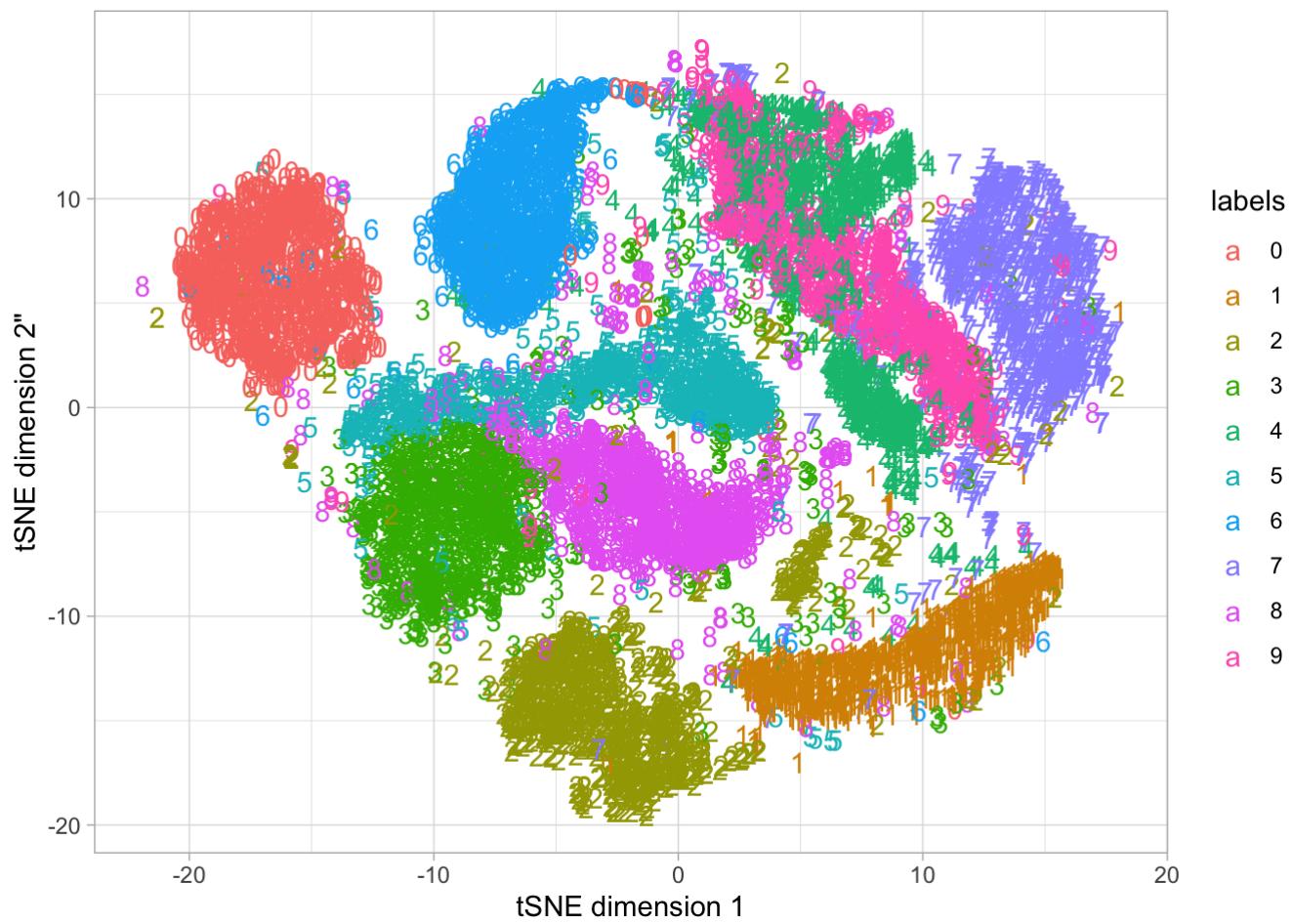
```
itercost = c(itercost,embedding$itercosts[10])
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2''')
```



```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                   perplexity = 160,
                   theta = 0.5,
                   eta = 200,
                   pca = TRUE, verbose = TRUE,
                   max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 160.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## - point 10000 of 10000
## Done in 34.02 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 5.74 seconds)
## Iteration 100: error is 77.748853 (50 iterations in 6.70 seconds)
## Iteration 150: error is 75.081994 (50 iterations in 7.91 seconds)
## Iteration 200: error is 74.927615 (50 iterations in 5.41 seconds)
## Iteration 250: error is 74.924031 (50 iterations in 5.49 seconds)
## Iteration 300: error is 1.997740 (50 iterations in 4.39 seconds)
## Iteration 350: error is 1.739551 (50 iterations in 4.42 seconds)
## Iteration 400: error is 1.611347 (50 iterations in 4.34 seconds)
## Iteration 450: error is 1.537428 (50 iterations in 4.09 seconds)
## Iteration 500: error is 1.490170 (50 iterations in 3.58 seconds)
## Fitting performed in 52.06 seconds.
```

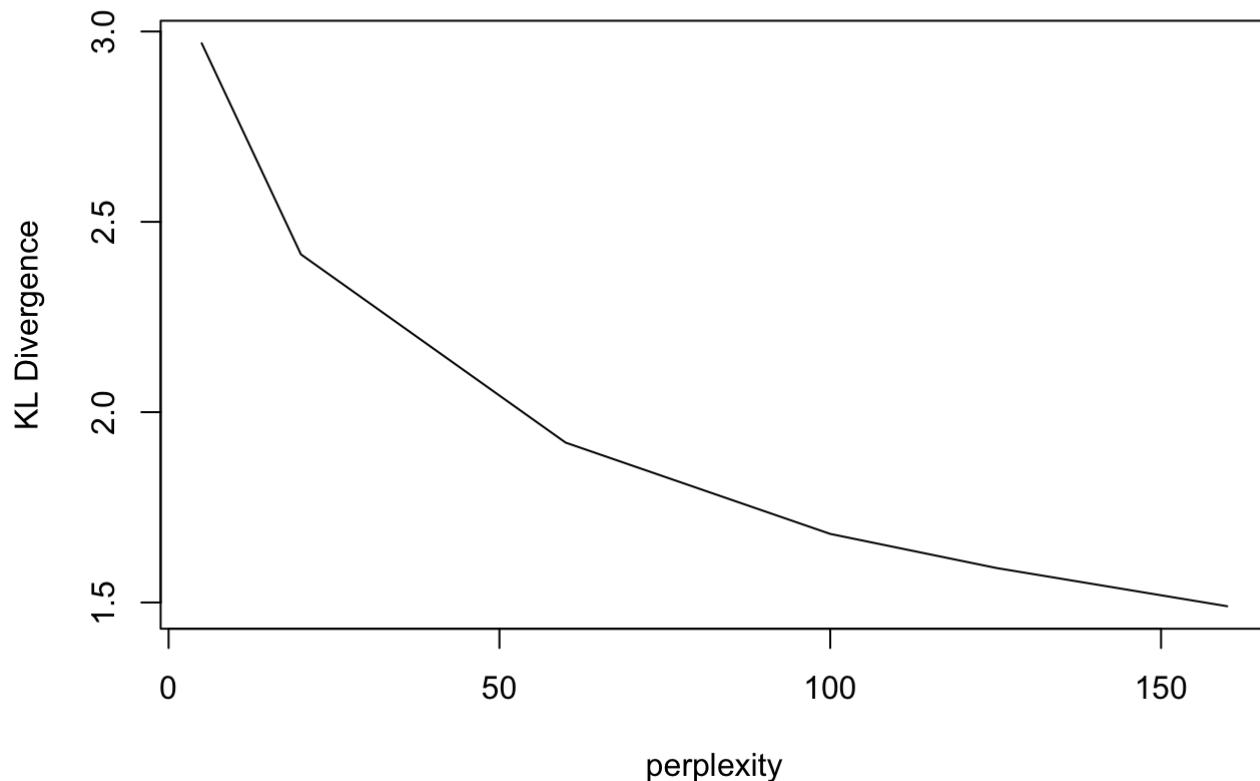
```
itercost = c(itercost,embedding$itercosts[10])
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2''')
```



e. When perplexity is as large as 5000, the clusters will become very dispersed.

f. The optimal perplexity is 160.

```
plot(c(5,20,60,100,125,160),itercost,type='l',xlab='perplexity',ylab='KL Divergence')
```

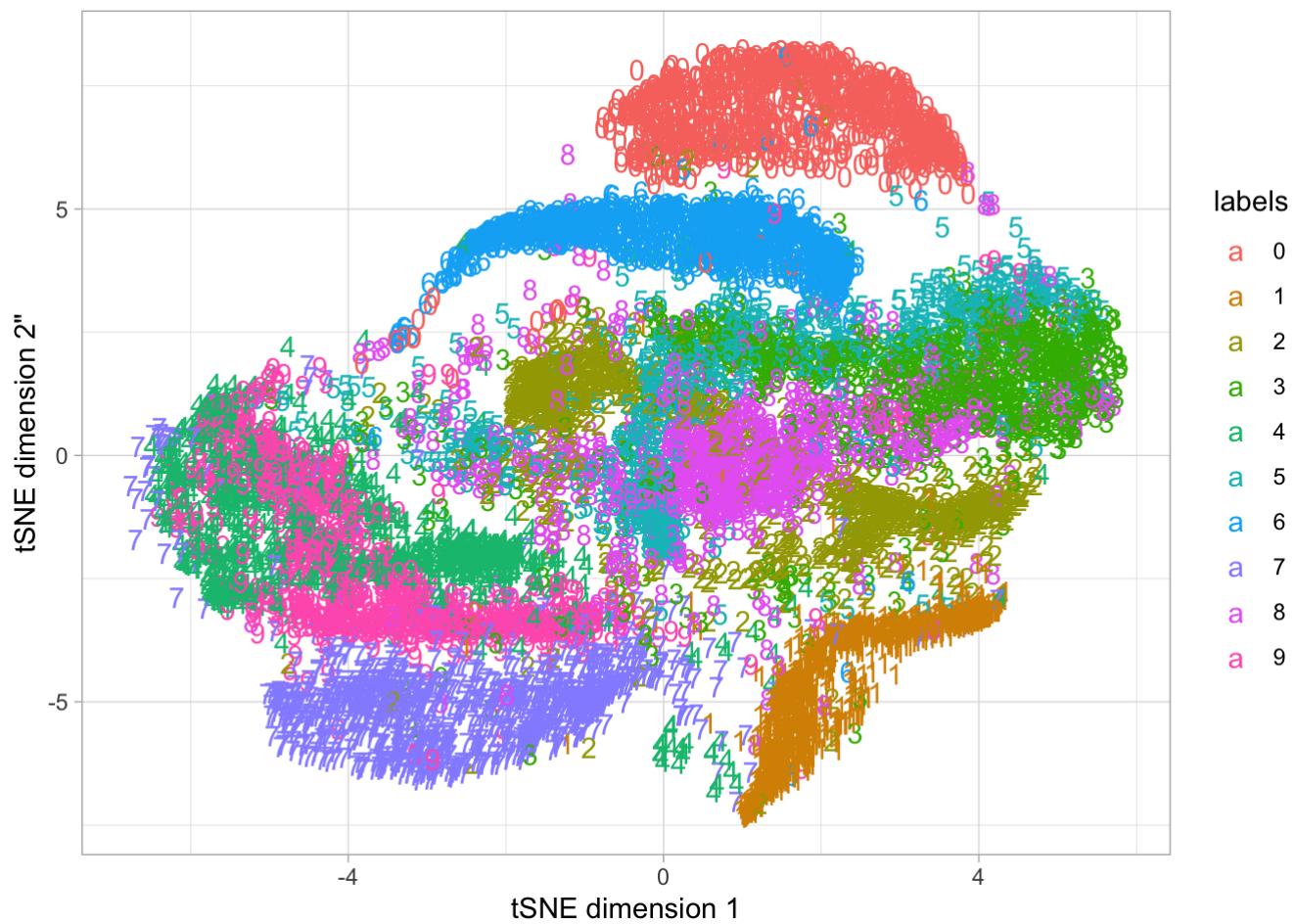


- e. Here we run eta=10 and eta=100 at perplexity=160. (since eta=200 is run before). Larger learning rate will lead to more dispersed clustering result for this dataset.

```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                    perplexity = 160,
                    theta = 0.5,
                    eta = 10,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 160.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## - point 10000 of 10000
## Done in 34.66 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 5.21 seconds)
## Iteration 100: error is 77.767491 (50 iterations in 5.13 seconds)
## Iteration 150: error is 77.767491 (50 iterations in 5.69 seconds)
## Iteration 200: error is 77.767491 (50 iterations in 6.11 seconds)
## Iteration 250: error is 77.767490 (50 iterations in 6.81 seconds)
## Iteration 300: error is 3.955333 (50 iterations in 7.01 seconds)
## Iteration 350: error is 2.846720 (50 iterations in 6.23 seconds)
## Iteration 400: error is 2.442388 (50 iterations in 5.16 seconds)
## Iteration 450: error is 2.236288 (50 iterations in 4.96 seconds)
## Iteration 500: error is 2.099385 (50 iterations in 4.72 seconds)
## Fitting performed in 57.02 seconds.
```

```
itercost = c(itercost,embedding$itercosts[10])
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2''')
```



```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                   perplexity = 160,
                   theta = 0.5,
                   eta = 100,
                   pca = TRUE, verbose = TRUE,
                   max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 160.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## - point 10000 of 10000
## Done in 30.25 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 3.98 seconds)
## Iteration 100: error is 77.767490 (50 iterations in 4.07 seconds)
## Iteration 150: error is 76.618588 (50 iterations in 4.33 seconds)
## Iteration 200: error is 75.284082 (50 iterations in 4.42 seconds)
## Iteration 250: error is 74.980694 (50 iterations in 4.50 seconds)
## Iteration 300: error is 2.119916 (50 iterations in 3.68 seconds)
## Iteration 350: error is 1.849361 (50 iterations in 3.46 seconds)
## Iteration 400: error is 1.712394 (50 iterations in 3.54 seconds)
## Iteration 450: error is 1.629139 (50 iterations in 4.08 seconds)
## Iteration 500: error is 1.573809 (50 iterations in 4.99 seconds)
## Fitting performed in 41.04 seconds.
```

```
itercost = c(itercost,embedding$itercosts[10])
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2''')
```

