Assignment 3, Part 1

Xiner Ning

3/28/2020

Dr. T's start code:

```
#######################################################
# Class: Anly-601
# Script: Starter code for creating boosted models
# Author: Joshuah Touyz
# Version: 0.1
# Last updated: 03/19/20
#######################################################

####################################
##### Loading libraries & data #####
####################################
library(tidyverse)
```

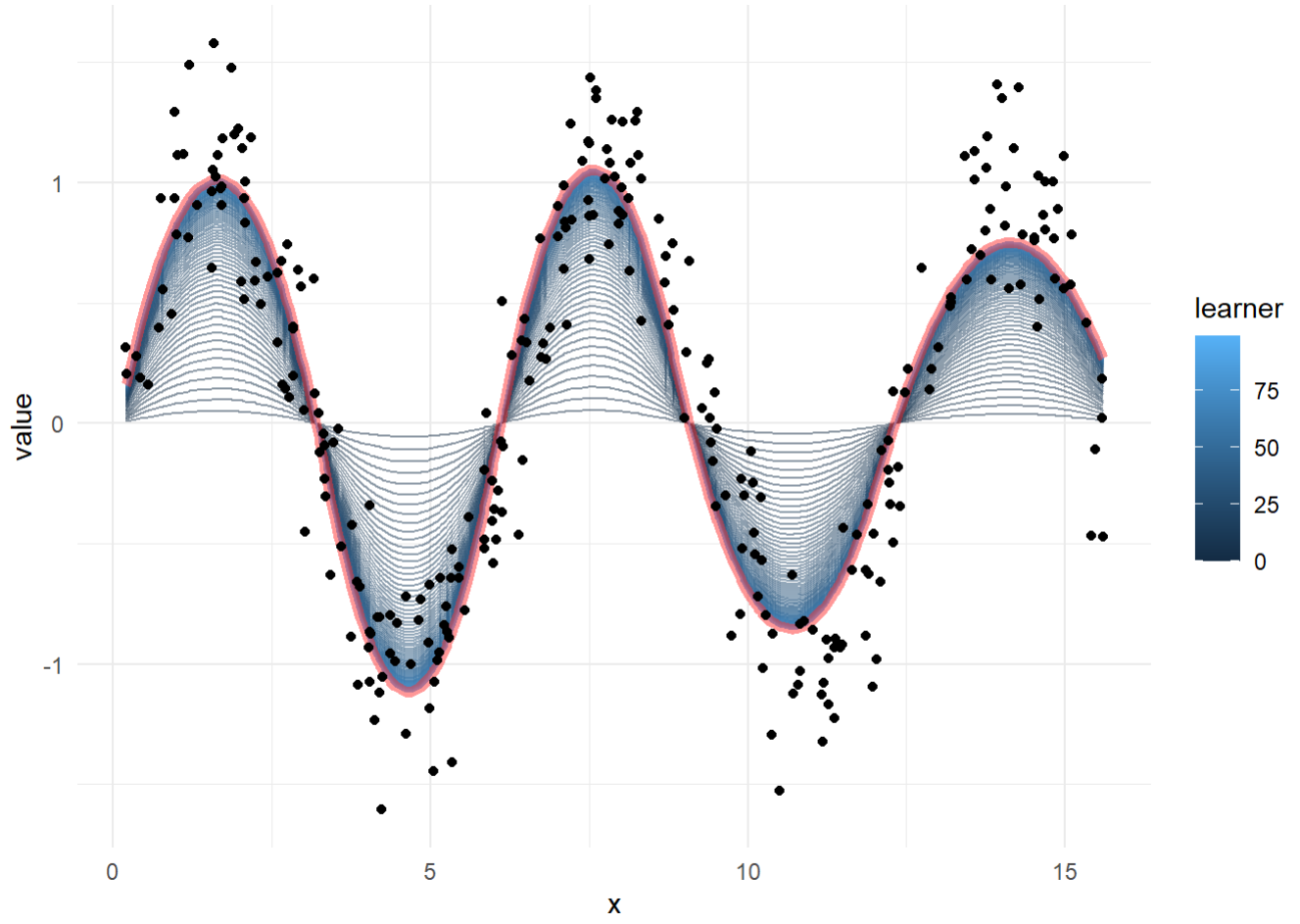
```
## -- Attaching packages ----------------------------------------------------------- tidyve
rse 1.3.0 --
```

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

```
## -- Conflicts -------------------------------------------------------------- tidyverse_co
nflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(splines)

# Generating sample data
n=300
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)


# Setting up parameters
v=.05
number_of_weak_learners = 100
number_of_knots_split = 6
polynomial_degree = 2

# Fit round 1
fit=lm(y~bs(x,degree=2,df=6),data=df)
yp = predict(fit,newdata=df)
df$yr = df$y - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = lm(yr ~ bs(x,
                   degree=polynomial_degree,
                   df=number_of_knots_split),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)
}

# Re-arrange sequences to get pseudo residuals
plot_wl = df %>% select(-y,-yr) %>%
  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()
```




```
###################################
##### Predicting on new data #####
###################################
```

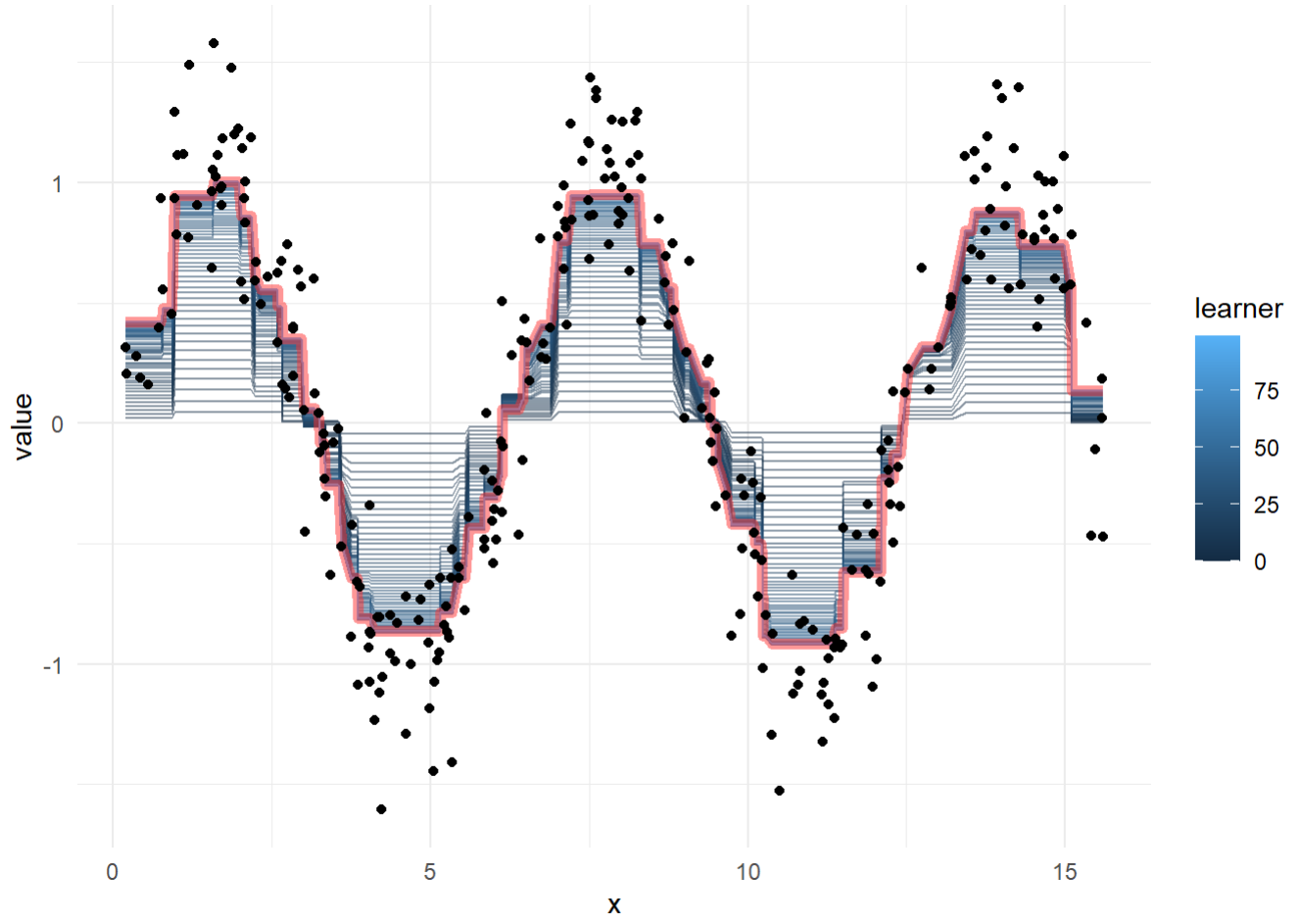
Question 1 (Boosting)

Part 1 In this question you will build a gradient boosted tree using the code in boosting.R

Q0: Then using rpart build a gradient boosted tree with v=0.05 and no changes to the default rpart parameters

```
library(tidyverse)
library(splines)
library(rpart)

n=300
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)


# Setting up parameters
v=.05
number_of_weak_learners = 100
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(y~x,data=df,parms = v)
yp = predict(fit,newdata=df)
df$yr = df$y - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(yr~x,data=df,parms = v)

  # 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]
```

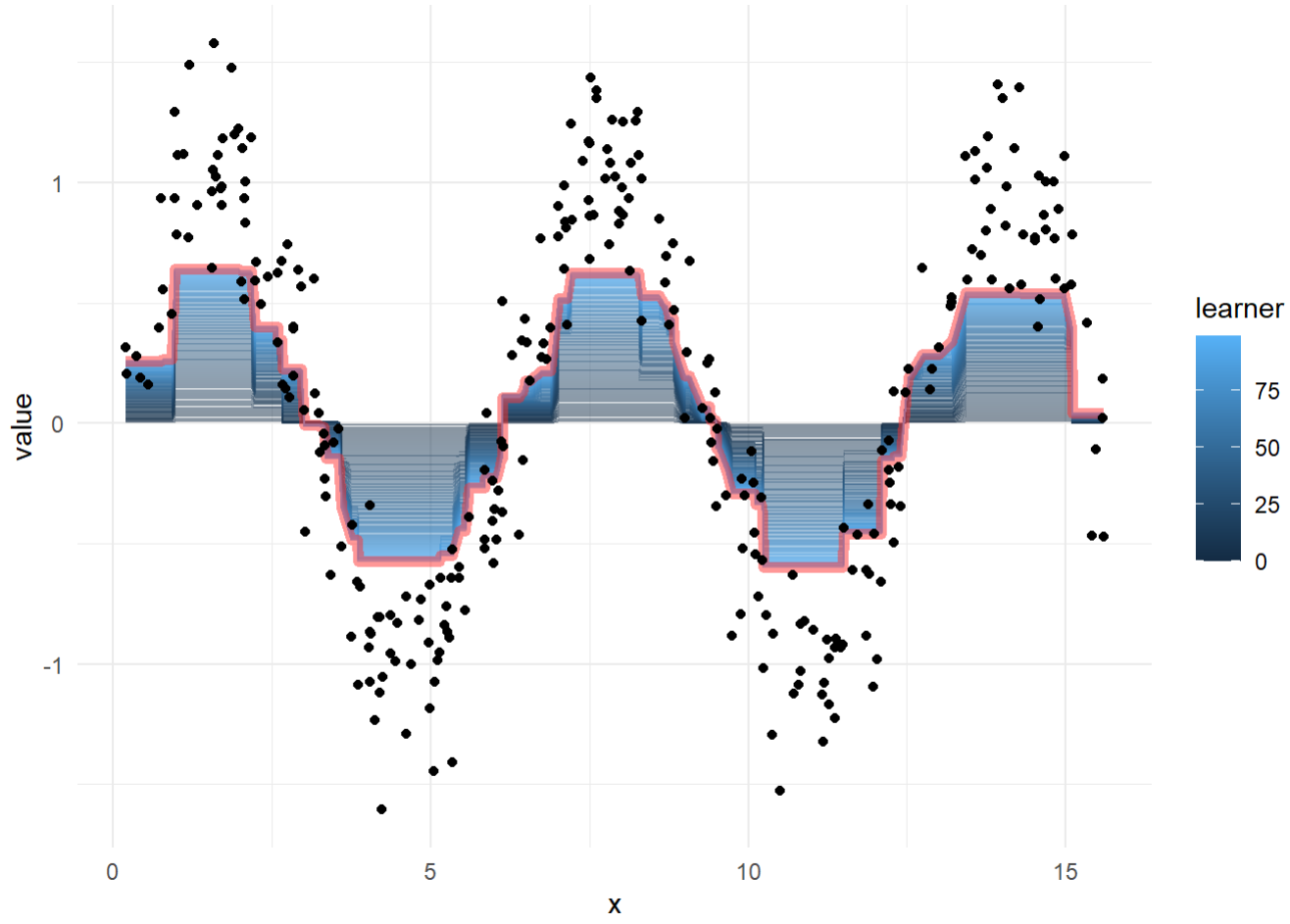
```
  }else{yp_i=apply(YP[,1:i],1,sum) #<- strong learner
  }

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

# Re-arrange sequences to get pseudo residuals
plot_wl = df %>% select(-y,-yr) %>%
  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()
```



Q1: What happens when you change: v the “learning parameter”, show the fitted plots v=0.01,0.05,0.125

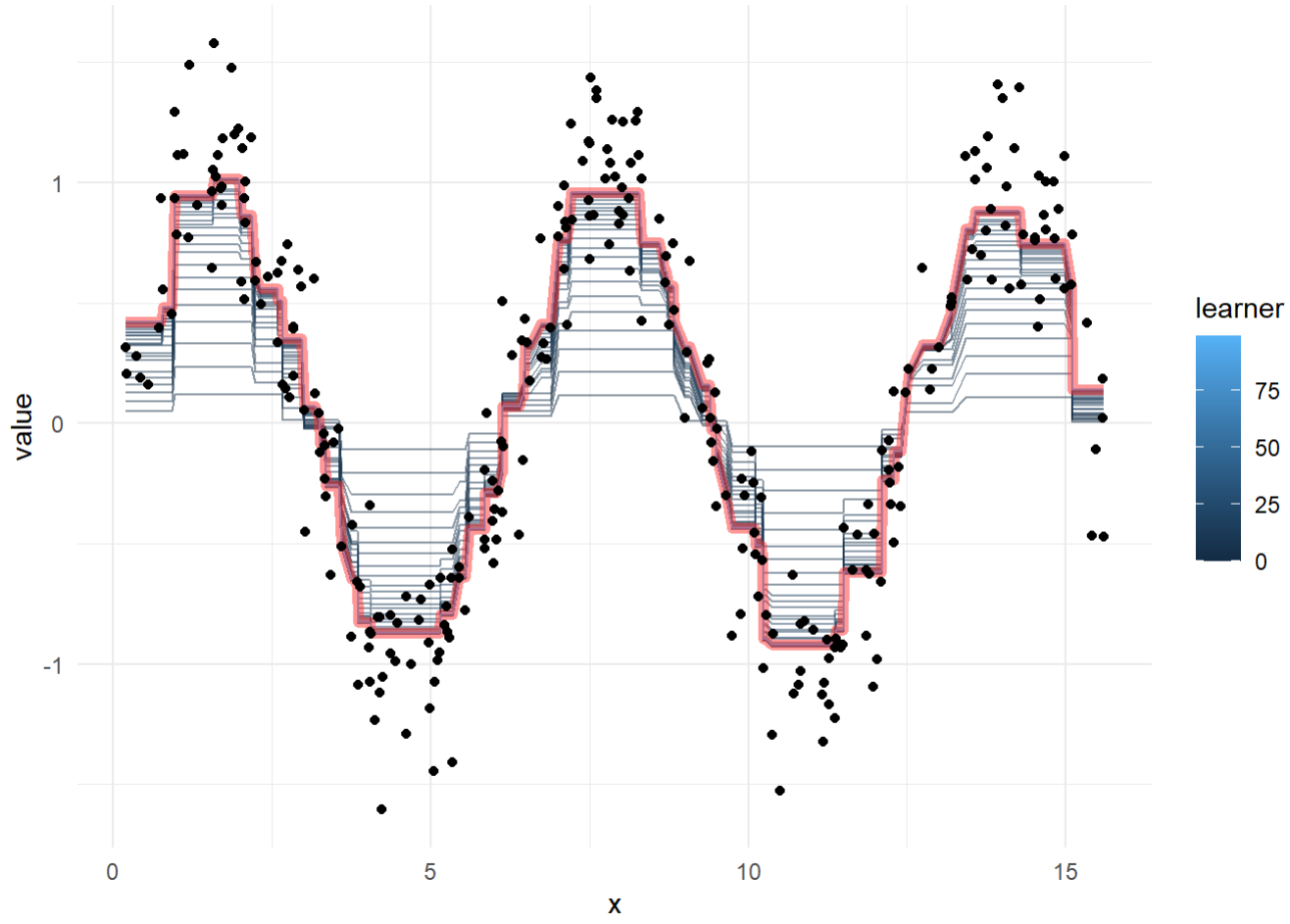
```
n=300
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)


# Setting up parameters
v=.01
number_of_weak_learners = 100
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(y~x,data=df,parms = v)
yp = predict(fit,newdata=df)
df$yr = df$y - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(yr~x,data=df,parms = v)

  # 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)
}

# Re-arrange sequences to get pseudo residuals
plot_wl = df %>% select(-y,-yr) %>%
  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()
```



```
n=300
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)


# Setting up parameters
v=.125
number_of_weak_learners = 100
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(y~x,data=df,parms = v)
yp = predict(fit,newdata=df)
df$yr = df$y - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(yr~x,data=df,parms = v)

  # 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)
}

# Re-arrange sequences to get pseudo residuals
plot_wl = df %>% select(-y,-yr) %>%
  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()
```



Q2: Using a validation and test set

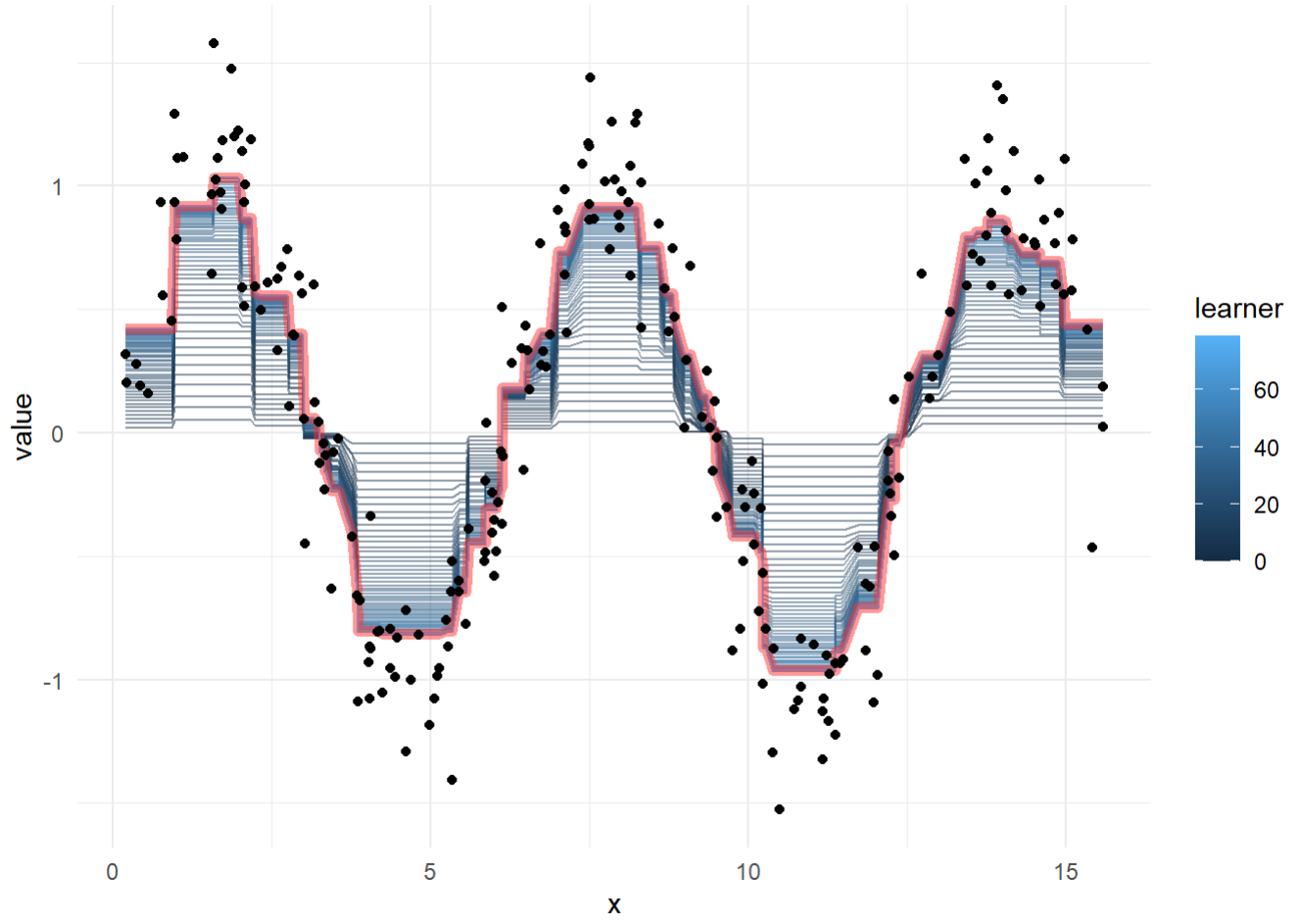
A. Develop a heuristic approach for determining when to stop training your gradient boosted tree using v=0.05 and the defualt setting.

Set the validation and test ratio as 0.8.

The learning stop when the largest v*yp among all data points is less than 0.0001.

```
n=300
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)

## split vlidation and test set
### Random sample indexes
validation_index <- sample(1:nrow(df), 0.8 * nrow(df))
test_index <- setdiff(1:nrow(df), validation_index)

validation_df = df[validation_index,]
test_df = df[test_index,]

# Setting up parameters
v=.05
number_of_weak_learners = 100
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(y~x,data=validation_df,parms = v)
yp = predict(fit,newdata=validation_df)
validation_df$yr = validation_df$y - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){

  if((max(v*yp)) < .0001){
    break
  }

  # Fit linear spline
  fit = rpart(yr~x,data=validation_df,parms = v)

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

  # Update residuals
  validation_df$yr=validation_df$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:dim(YP)[2]){
  # 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)
  validation_df = validation_df %>% bind_cols(yp=yp_i)
}

# Re-arrange sequences to get pseudo residuals
plot_wl = validation_df %>% select(-y,-yr) %>%
  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 == (dim(YP)[2]-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 = validation_df)+ # true values
  theme_minimal()
```



B. How many trees did you include?

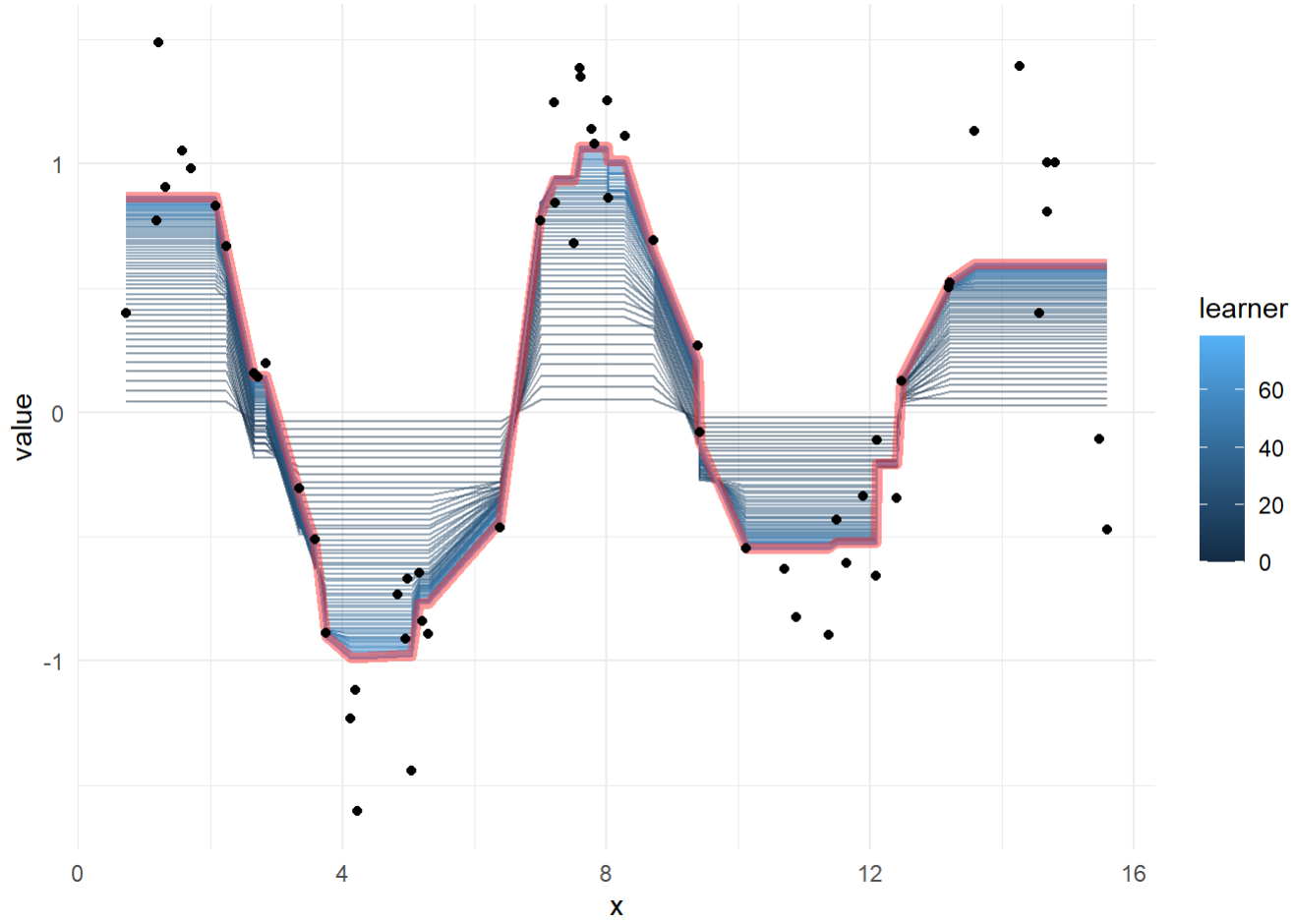
The learning stops at iteration 80. So there are 80 trees being included.

C. What is your performance on the test set for the set of selected parameters (use RMSE= root mean squared error, to assess performance)?

Use 80 as number_of_weak_learners in test_df.

```
# Setting up parameters
v=.05
number_of_weak_learners = 80
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(y~x,data=test_df,parms = v)
yp = predict(fit,newdata=test_df)
test_df$yr = test_df$y - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(yr~x,data=test_df,parms = v)

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

  # Update residuals
  test_df$yr=test_df$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)
  test_df = test_df %>% bind_cols(yp=yp_i)
}

# Re-arrange sequences to get pseudo residuals
plot_wl = test_df %>% select(-y,-yr) %>%
  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 = test_df)+ # true values
  theme_minimal()
```



Calculate RMSE for test set.

```
RMSE = function(m, o){
  sqrt(mean((m - o)^2))
}

RMSE(final_learner$value,test_df$y)
```

```
## [1] 0.2955941
```

Q3: Next you are going to tune you trees, use grid search to assess different values for minsplit,cp,maxdepth. What is the best set of parameters as measured by RMSE?

Define hyperparameter combinations using grid search:

```
gs <- list(minsplit = c(2, 5, 10),
           maxdepth = c(1, 3, 5),
           cp = c(0.01,0.02,0.03)) %>%
  cross_d() # Convert to data frame grid
```

```
## Warning: `cross_d()` is deprecated; please use `cross_df()` instead.
```

```
gs
```

```
## # A tibble: 27 x 3
##    minsplit maxdepth    cp
##       <dbl>    <dbl> <dbl>
##  1        2        1  0.01
##  2        5        1  0.01
##  3       10        1  0.01
##  4        2        3  0.01
##  5        5        3  0.01
##  6       10        3  0.01
##  7        2        5  0.01
##  8        5        5  0.01
##  9       10        5  0.01
## 10        2        1  0.02
## # ... with 17 more rows
```

Build boosted tree in rpart with gs parameters:

```
mod <- function(...) {
  rpart(y ~ x, data = df, control = rpart.control(...))
}

fit = pmap(gs, mod)
```

Calculate RMSE for model with different parameters and find the one with the smallest RMSE:

```
RMSE <- rep(NA,27)
for (i in 1:27){
  predicted <- predict(fit[[i]], data=data.frame(df$x))
  RMSE[i] <- sqrt(mean(df$y-predicted)^2)
}

data.frame(gs)[which(RMSE == min(RMSE)),]
```

```
##    minsplit maxdepth   cp
## 1         2        1 0.01
## 2         5        1 0.01
## 3        10        1 0.01
## 10        2        1 0.02
## 11        5        1 0.02
## 12       10        1 0.02
## 19        2        1 0.03
## 20        5        1 0.03
## 21       10        1 0.03
```

So we see that model with maxdepth=1 has the best performance.

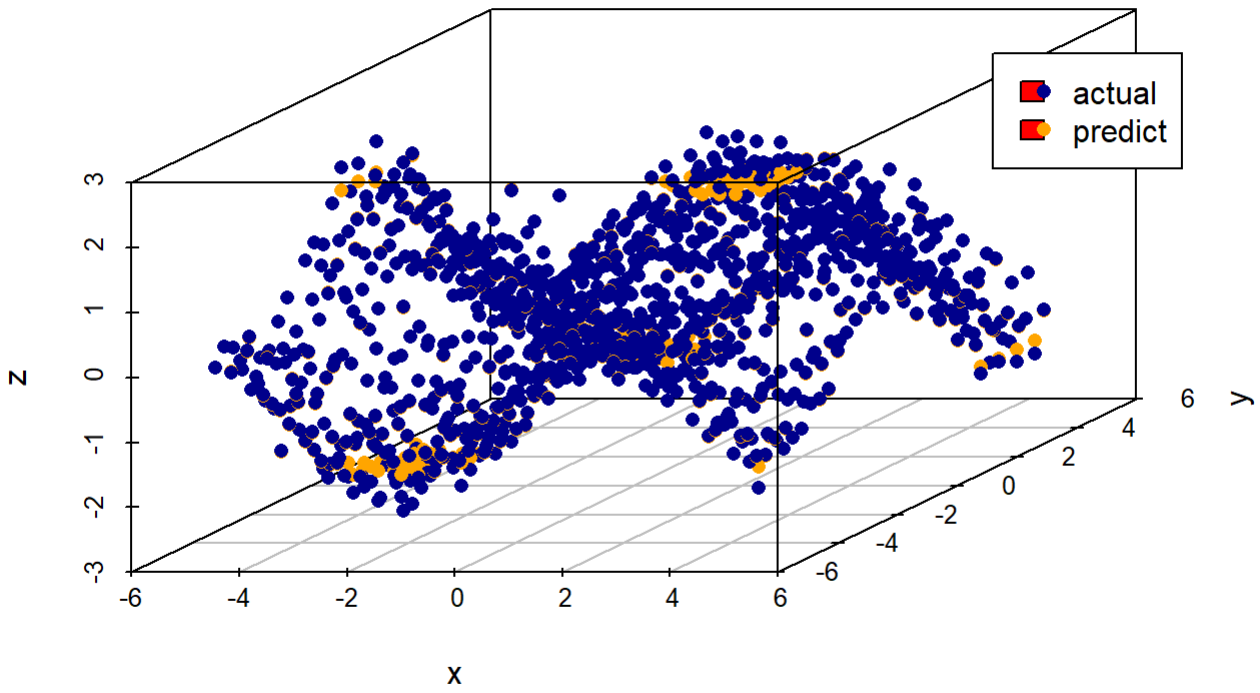
Part 2: The approach above can be extended to multiple dimensions. Repeat Q1-Q3 for the data set kernel_regression_2.csv from Assignment 2.

Q1: the fitted plots v=0.01,0.05,0.125

When v=0.01

```
library(scatterplot3d)
df <- read.csv('kernel_regression_2.csv')

# Setting up parameters
v=.01
number_of_weak_learners = 100
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(z~.,data=df,parms = v)
yp = predict(fit,newdata=df)
df$yr = df$z - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(z~.,data=df,parms = v)

  # 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)
}
```

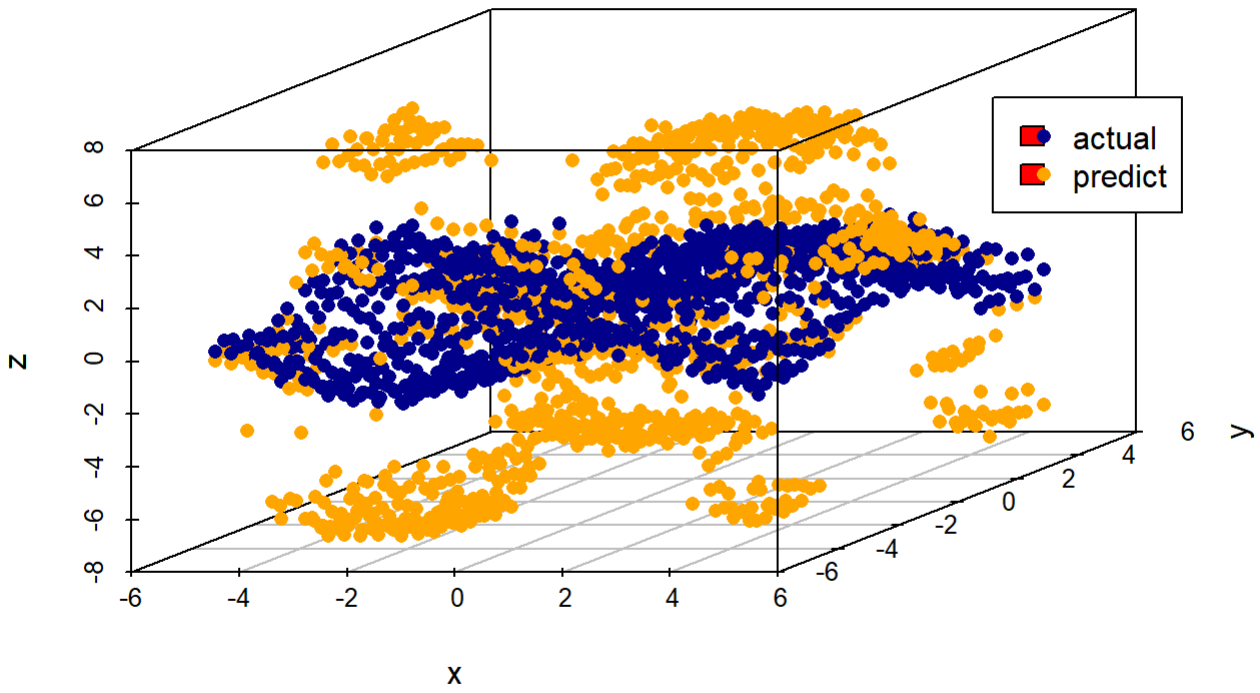
```
# Re-arrange sequences to get pseudo residuals
plot_wl = df %>% select(-z,-yr) %>%
  pivot_longer(cols = starts_with("yp")) %>%
  mutate(learner = str_match(name,"[0-9]+")) %>%
  mutate(learner = as.integer(ifelse(is.na(learner),0,learner)))

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


# put x y, z, z_predict in one dataframe and make 3D plot

myplot_df1 <- data.frame(x=df$x,y=df$y,z=df$z,group='actual')
myplot_df2 <- data.frame(x=df$x,y=df$y,z=final_learner$value,group='predict')
myplot_df <- rbind(myplot_df1,myplot_df2)

cols <- c("darkblue", "orange")


with(myplot_df,
     scatterplot3d(x,
                   y,
                   z,
                   main="3D Plot: actual VS predicted values",
                   xlab = "x",
                   ylab = "y",
                   zlab = "z",
                   pch = 16, color=cols[as.numeric(myplot_df$group)]))
legend(5,5,2,legend = levels(myplot_df$group),
      col =  c("darkblue", "orange"), pch = 16)
```


3D Plot: actual VS predicted values
actual
predict
z
x
y
-6
-4
-2
0
2
4
6
-3
-2
-1
0
1
2
3

When v=0.05

```
df <- read.csv('kernel_regression_2.csv')

# Setting up parameters
v=.05
number_of_weak_learners = 100
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(z~.,data=df,parms = v)
yp = predict(fit,newdata=df)
df$yr = df$z - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(z~.,data=df,parms = v)

  # 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)
}

# Re-arrange sequences to get pseudo residuals
```

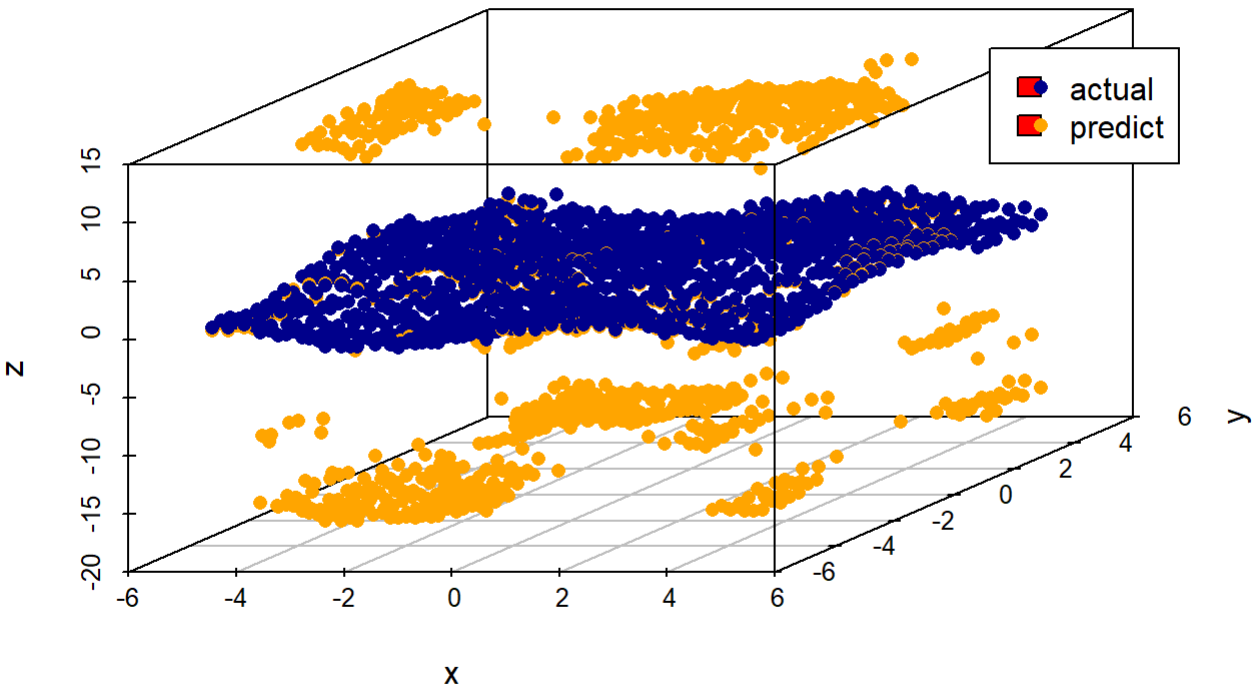
```
plot_wl = df %>% select(-z,-yr) %>%
  pivot_longer(cols = starts_with("yp")) %>%
  mutate(learner = str_match(name,"[0-9]+")) %>%
  mutate(learner = as.integer(ifelse(is.na(learner),0,learner)))

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


# put x y, z, z_predict in one dataframe and make 3D plot

myplot_df1 <- data.frame(x=df$x,y=df$y,z=df$z,group='actual')
myplot_df2 <- data.frame(x=df$x,y=df$y,z=final_learner$value,group='predict')
myplot_df <- rbind(myplot_df1,myplot_df2)

cols <- c("darkblue", "orange")


with(myplot_df,
     scatterplot3d(x,
                   y,
                   z,
                   main="3D Plot: actual VS predicted values",
                   xlab = "x",
                   ylab = "y",
                   zlab = "z",
                   pch = 16, color=cols[as.numeric(myplot_df$group)]))
legend(5,5,2,legend = levels(myplot_df$group),
      col =  c("darkblue", "orange"), pch = 16)
```


3D Plot: actual VS predicted values
actual
predict
z
x
y
-8
-6
-4
-2
0
2
4
6
8
-6
-4
-2
0
2
4
6
-6
-4
-2
0
2
4
6

When v=0.125

```
df <- read.csv('kernel_regression_2.csv')

# Setting up parameters
v=.125
number_of_weak_learners = 100
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(z~.,data=df,parms = v)
yp = predict(fit,newdata=df)
df$yr = df$z - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

##################################
##### Boosting with Splines #####
##################################
for(t in 2:number_of_weak_learners){
  # Fit linear spline
  fit = rpart(z~.,data=df,parms = v)

  # 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)
}

# Re-arrange sequences to get pseudo residuals
```

```
plot_wl = df %>% select(-z,-yr) %>%
  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))


# put x y, z, z_predict in one dataframe and make 3D plot

myplot_df1 <- data.frame(x=df$x,y=df$y,z=df$z,group='actual')
myplot_df2 <- data.frame(x=df$x,y=df$y,z=final_learner$value,group='predict')
myplot_df <- rbind(myplot_df1,myplot_df2)

cols <- c("darkblue", "orange")


with(myplot_df,
     scatterplot3d(x,
                   y,
                   z,
                   main="3D Plot: actual VS predicted values",
                   xlab = "x",
                   ylab = "y",
                   zlab = "z",
                   pch = 16, color=cols[as.numeric(myplot_df$group)]))
legend(5,5,2,legend = levels(myplot_df$group),
      col =  c("darkblue", "orange"), pch = 16)
```



Q2

Set the validation and test ratio as 0.8.
The learning stop when the largest v*yp among all data points is less than 0.0001.

```
df <- read.csv('kernel_regression_2.csv')

## split vlidation and test set
### Random sample indexes
validation_index <- sample(1:nrow(df), 0.8 * nrow(df))
test_index <- setdiff(1:nrow(df), validation_index)

validation_df = df[validation_index,]
test_df = df[test_index,]

# Setting up parameters
v=.05
number_of_weak_learners = 100
number_of_knots_split = 6
#polynomial_degree = 2

# Fit round 1
fit=rpart(z~.,data=validation_df,parms = v)
yp = predict(fit,newdata=validation_df)
validation_df$yr = validation_df$z - v*yp
YP = v*yp ############
list_of_weak_learners = list(fit)

###################################
##### Boosting with Splines #####
###################################
for(t in 2:number_of_weak_learners){

  if((max(v*yp)) < .01){
    break
  }

  # Fit linear spline
  fit = rpart(z~.,data=validation_df,parms = v)

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

  # Update residuals
  validation_df$yr=validation_df$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:dim(YP)[2]){
  # 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)
  validation_df = validation_df %>% bind_cols(yp=yp_i)
}

# Re-arrange sequences to get pseudo residuals
plot_wl = validation_df %>% select(-z,-yr) %>%
  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 == (dim(YP)[2]-1))
```

Question 2 (TSNE)

Part 1: Read the article ‘How to use t-SNE effectively’

– a. Do the distances between points in tSNE matter?
Distances between well-separated clusters in a t-SNE plot may mean nothing.

– b. What does the parameter value “perplexity” mean? Typical value of perplexity is 5-50. Lossely speaking, perplexity is the number of points in a neighborhood that are being preserved. It has a complex effect on the final results and it balances attention between local and global aspect of your data,

– c. What effect does the number of steps have on the final outcome of embedding?
As step size increases, final oucome of embedding becomes more stable.

– d. Explain why you may need more than one plot to explain topological information using tSNE
We need more than one plot with different hyperparameter to find the one that best desribe the original data.

Part 2: Using the code in exploring_tsne.R you are going to explore tSNE with the MNIST data set

```
####################################
##### Loading libraries & data #####
####################################
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')
```

– a. Plot the PCA plot

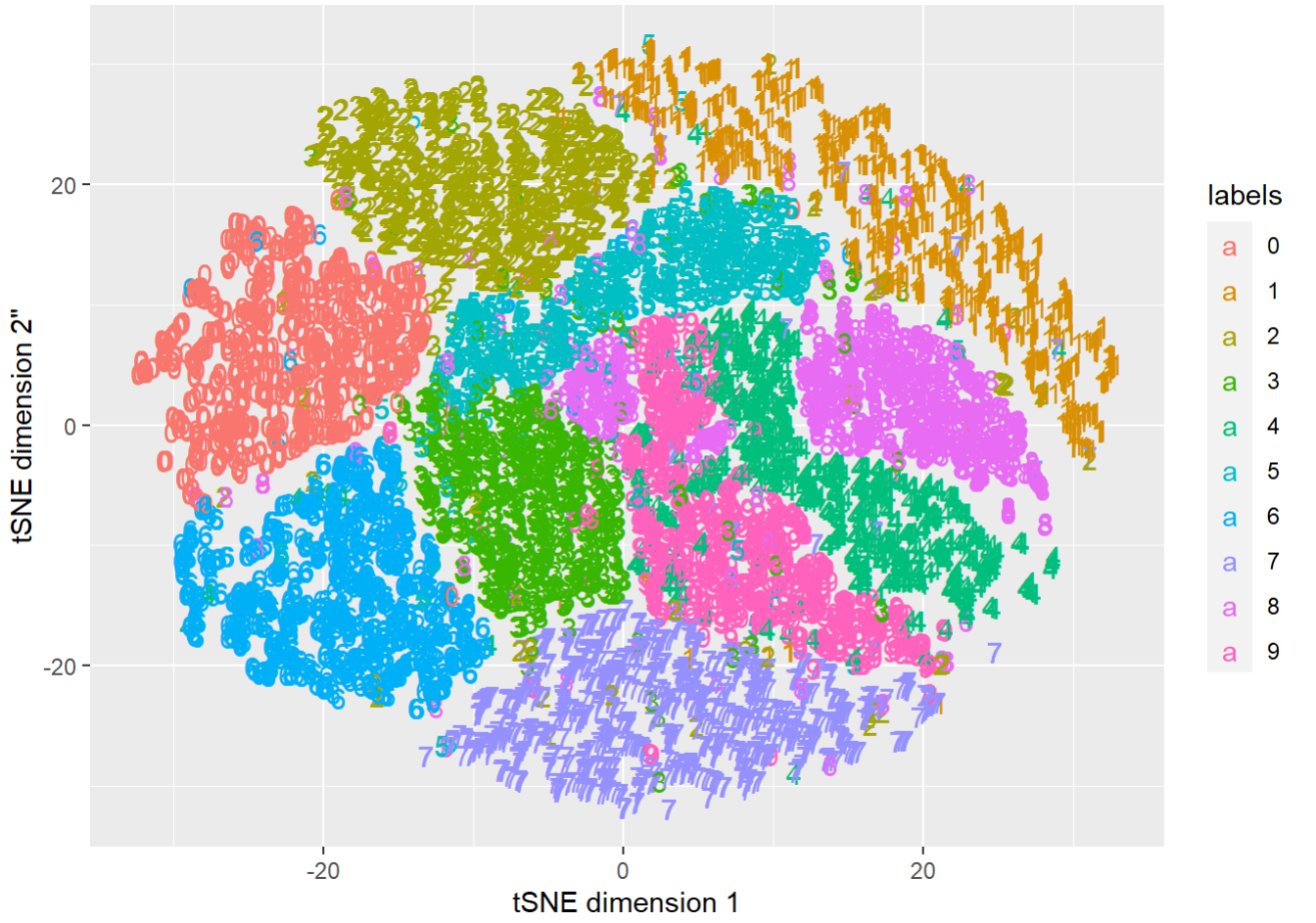
```
################################################
##### Visualizing the PCA decomposition  #####
################################################
pca = princomp(first_10k_samples)$scores[,1:2]
pca_plot = tibble(x = pca[,1], y =pca[,2], labels = as.character(first_10k_samples_labels))
ggplot(aes(x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

– b. Plot the TSNE embedding for perplexity = 5 use 500 iterations.

```
################################################
#####     Running the TSNE emebdding     #####
################################################
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 24.51 seconds (sparsity = 0.002104)!
## Learning embedding...
## Iteration 50: error is 118.296616 (50 iterations in 5.11 seconds)
## Iteration 100: error is 107.305307 (50 iterations in 5.36 seconds)
## Iteration 150: error is 100.409048 (50 iterations in 4.53 seconds)
## Iteration 200: error is 97.672636 (50 iterations in 5.05 seconds)
## Iteration 250: error is 96.062188 (50 iterations in 5.30 seconds)
## Iteration 300: error is 4.394599 (50 iterations in 4.73 seconds)
## Iteration 350: error is 3.820457 (50 iterations in 3.94 seconds)
## Iteration 400: error is 3.451762 (50 iterations in 4.44 seconds)
## Iteration 450: error is 3.185335 (50 iterations in 4.57 seconds)
## Iteration 500: error is 2.980598 (50 iterations in 4.74 seconds)
## Fitting performed in 47.77 seconds.
```

```
# 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"')
```

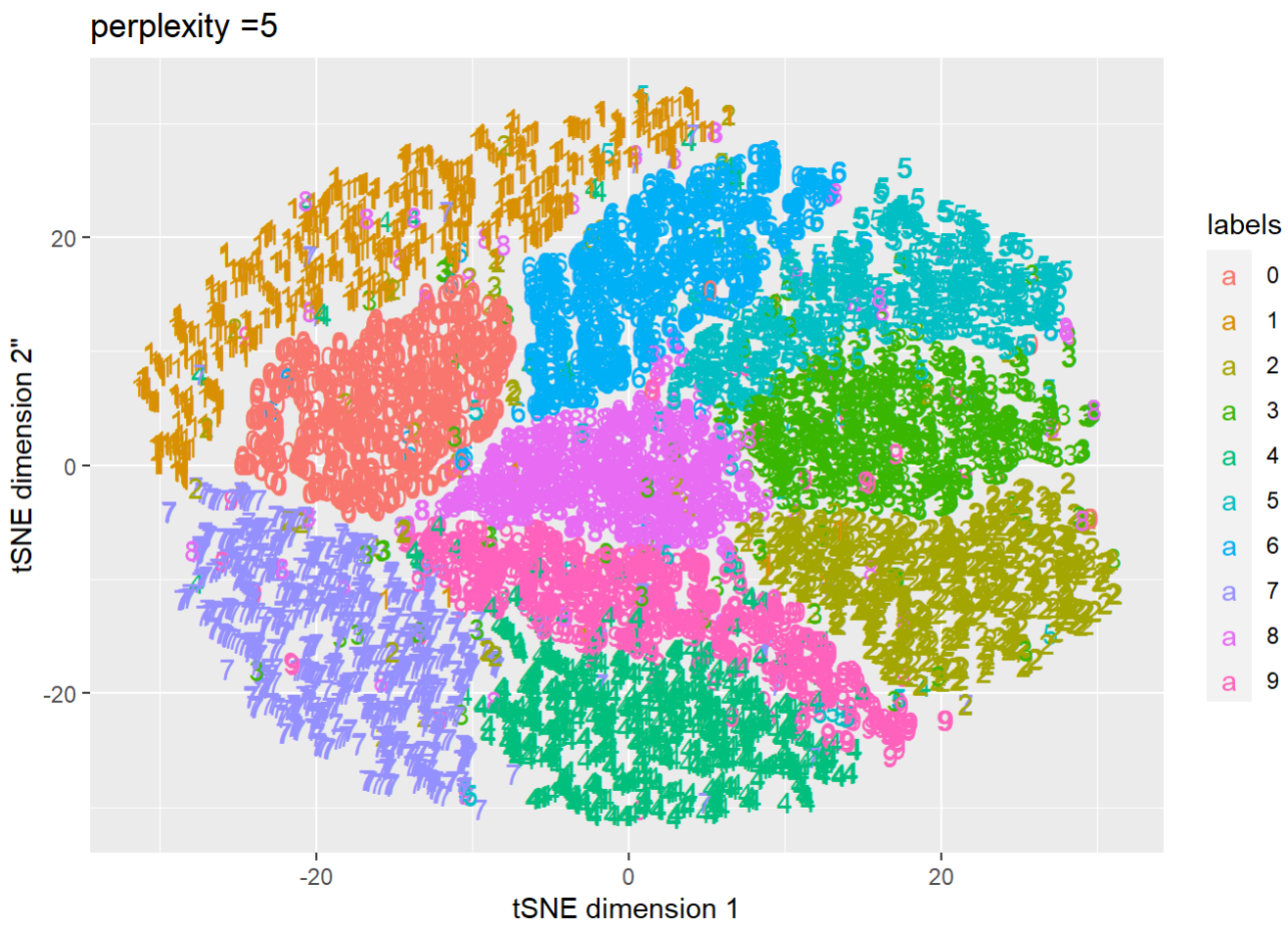


– c. Plot the TSNE embedding for perplexity = 5,20,60,100,125,160, what do you notice?

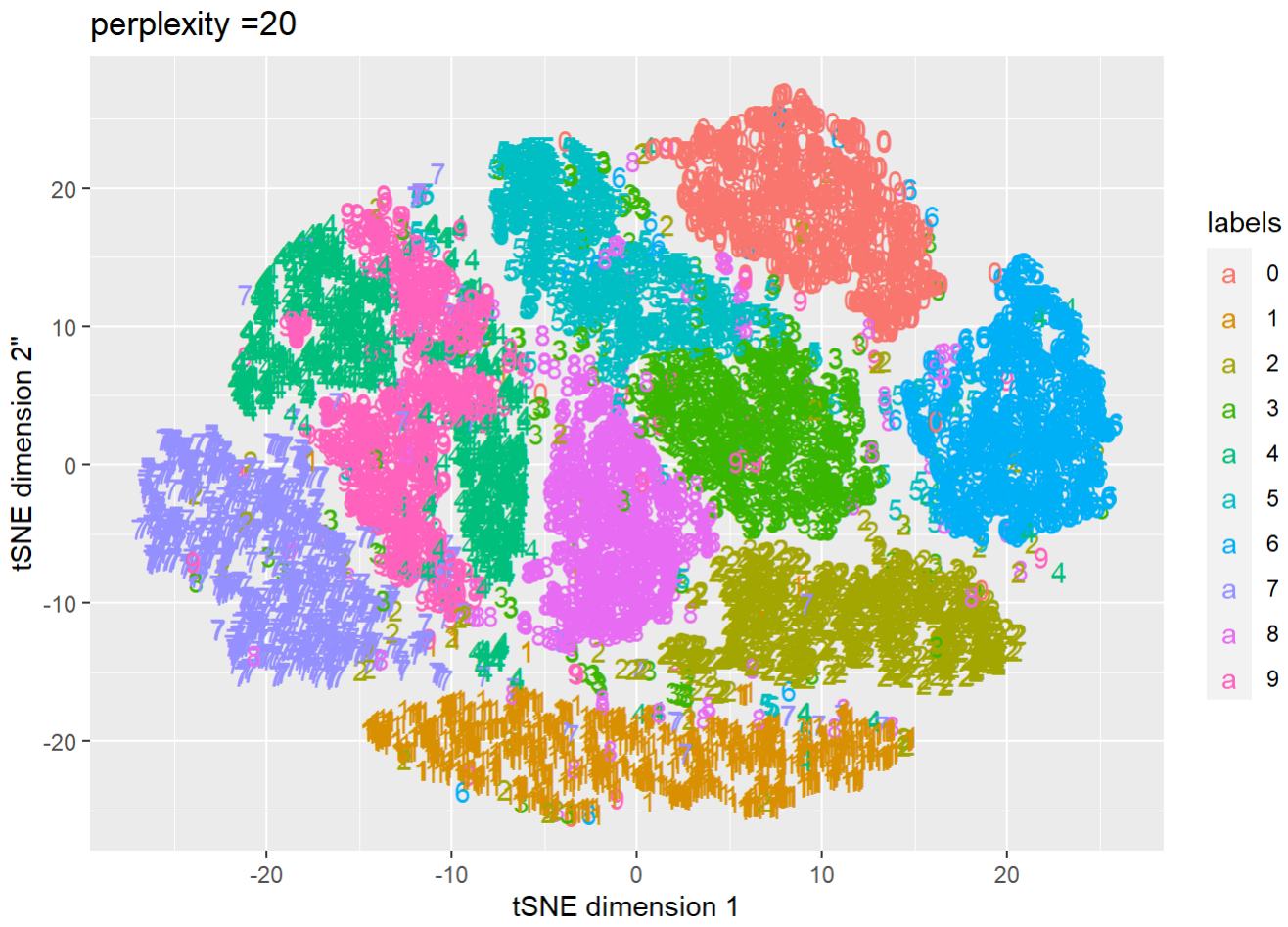
```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = 5,
                  theta = 0.5,
                  eta = 200,
                  pca = TRUE, verbose = TRUE,
                  max_iter = 500)
```

```
## 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 32.74 seconds (sparsity = 0.002104)!
## Learning embedding...
## Iteration 50: error is 118.296636 (50 iterations in 7.92 seconds)
## Iteration 100: error is 107.175691 (50 iterations in 4.59 seconds)
## Iteration 150: error is 99.468863 (50 iterations in 5.34 seconds)
## Iteration 200: error is 96.667057 (50 iterations in 5.42 seconds)
## Iteration 250: error is 95.112440 (50 iterations in 5.43 seconds)
## Iteration 300: error is 4.372788 (50 iterations in 4.72 seconds)
## Iteration 350: error is 3.809535 (50 iterations in 3.94 seconds)
## Iteration 400: error is 3.441738 (50 iterations in 3.81 seconds)
## Iteration 450: error is 3.176274 (50 iterations in 4.33 seconds)
## Iteration 500: error is 2.971315 (50 iterations in 4.69 seconds)
## Fitting performed in 50.19 seconds.
```

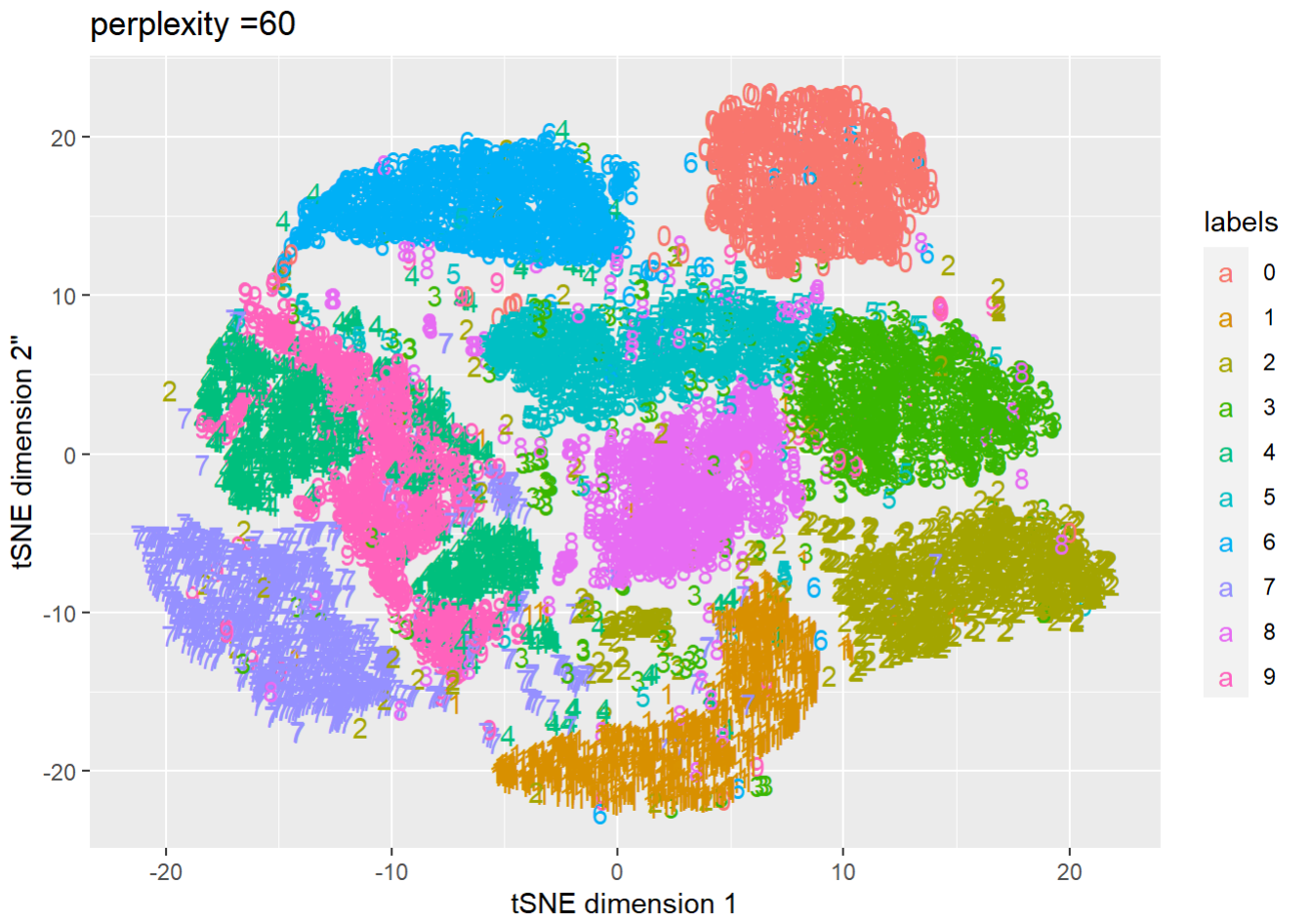
```
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2"')+ggtitle(   ('perplexity =5')
)
```



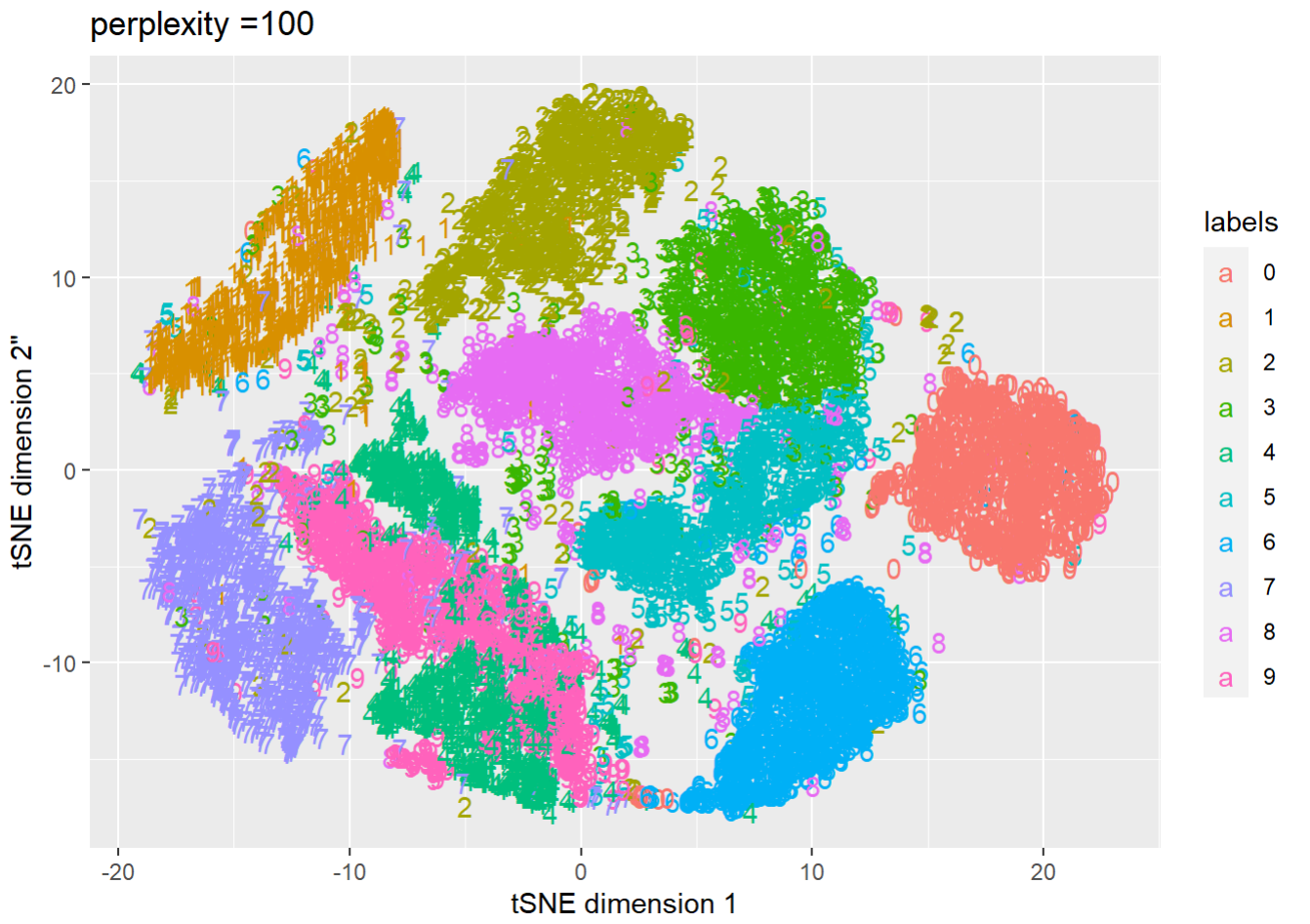

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 20.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 32.36 seconds (sparsity = 0.008217)!
## Learning embedding...
## Iteration 50: error is 102.343944 (50 iterations in 4.86 seconds)
## Iteration 100: error is 92.625350 (50 iterations in 5.39 seconds)
## Iteration 150: error is 88.464427 (50 iterations in 4.36 seconds)
## Iteration 200: error is 87.760235 (50 iterations in 4.26 seconds)
## Iteration 250: error is 87.561547 (50 iterations in 4.09 seconds)
## Iteration 300: error is 3.399365 (50 iterations in 4.20 seconds)
## Iteration 350: error is 2.950062 (50 iterations in 4.22 seconds)
## Iteration 400: error is 2.696896 (50 iterations in 3.83 seconds)
## Iteration 450: error is 2.529727 (50 iterations in 5.09 seconds)
## Iteration 500: error is 2.407325 (50 iterations in 4.83 seconds)
## Fitting performed in 45.12 seconds.
```



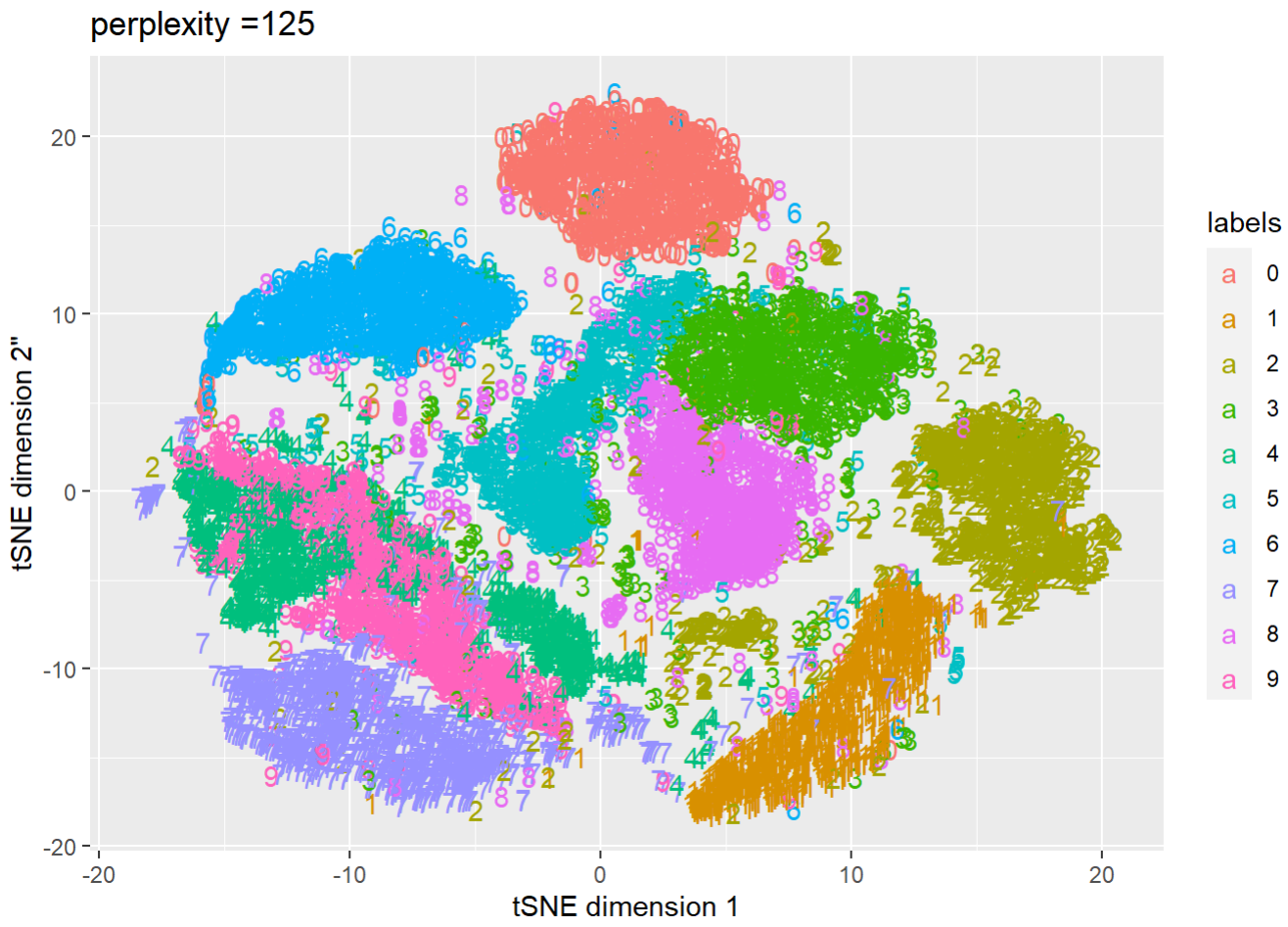

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 60.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 36.06 seconds (sparsity = 0.024328)!
## Learning embedding...
## Iteration 50: error is 89.428527 (50 iterations in 5.61 seconds)
## Iteration 100: error is 85.584534 (50 iterations in 7.30 seconds)
## Iteration 150: error is 81.782225 (50 iterations in 6.28 seconds)
## Iteration 200: error is 81.748666 (50 iterations in 5.69 seconds)
## Iteration 250: error is 81.521236 (50 iterations in 5.51 seconds)
## Iteration 300: error is 2.611150 (50 iterations in 5.32 seconds)
## Iteration 350: error is 2.272051 (50 iterations in 5.36 seconds)
## Iteration 400: error is 2.096188 (50 iterations in 5.47 seconds)
## Iteration 450: error is 1.986804 (50 iterations in 5.88 seconds)
## Iteration 500: error is 1.910685 (50 iterations in 4.89 seconds)
## Fitting performed in 57.29 seconds.
```



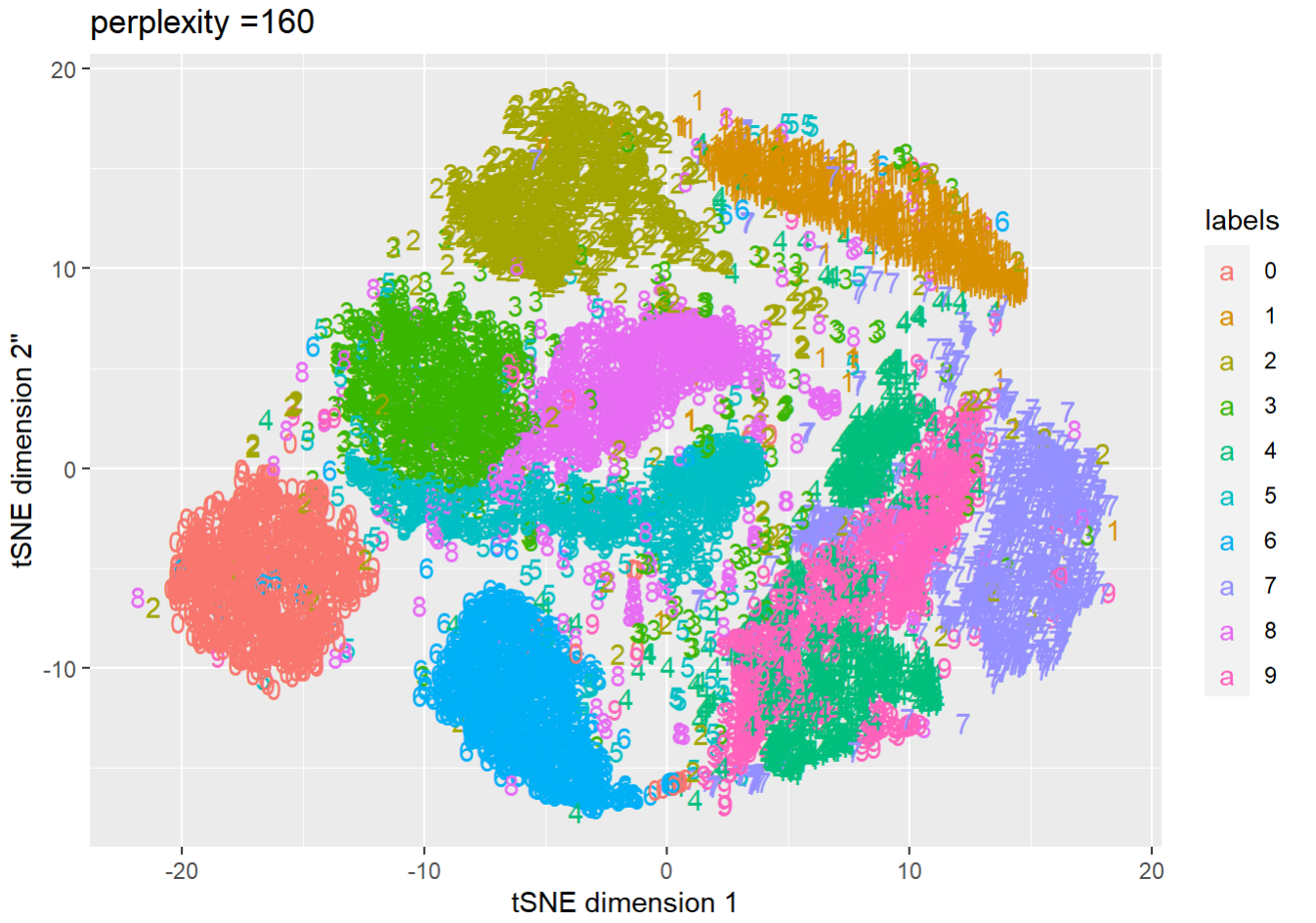

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 100.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 45.81 seconds (sparsity = 0.040571)!
## Learning embedding...
## Iteration 50: error is 83.371128 (50 iterations in 7.14 seconds)
## Iteration 100: error is 81.722562 (50 iterations in 12.38 seconds)
## Iteration 150: error is 78.362527 (50 iterations in 10.49 seconds)
## Iteration 200: error is 78.282697 (50 iterations in 7.93 seconds)
## Iteration 250: error is 78.269868 (50 iterations in 7.45 seconds)
## Iteration 300: error is 2.306024 (50 iterations in 6.49 seconds)
## Iteration 350: error is 1.995891 (50 iterations in 6.32 seconds)
## Iteration 400: error is 1.839198 (50 iterations in 6.24 seconds)
## Iteration 450: error is 1.747433 (50 iterations in 6.02 seconds)
## Iteration 500: error is 1.685629 (50 iterations in 6.65 seconds)
## Fitting performed in 77.14 seconds.
```




```
## 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 62.12 seconds (sparsity = 0.050806)!
## Learning embedding...
## Iteration 50: error is 80.714217 (50 iterations in 17.41 seconds)
## Iteration 100: error is 80.620499 (50 iterations in 20.66 seconds)
## Iteration 150: error is 76.722822 (50 iterations in 16.56 seconds)
## Iteration 200: error is 76.738675 (50 iterations in 7.69 seconds)
## Iteration 250: error is 76.735951 (50 iterations in 8.50 seconds)
## Iteration 300: error is 2.130951 (50 iterations in 4.52 seconds)
## Iteration 350: error is 1.854940 (50 iterations in 6.80 seconds)
## Iteration 400: error is 1.720390 (50 iterations in 7.22 seconds)
## Iteration 450: error is 1.641707 (50 iterations in 8.97 seconds)
## Iteration 500: error is 1.589524 (50 iterations in 10.50 seconds)
## Fitting performed in 108.82 seconds.
```




```
## 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 57.85 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 8.30 seconds)
## Iteration 100: error is 77.749752 (50 iterations in 15.80 seconds)
## Iteration 150: error is 75.221340 (50 iterations in 16.55 seconds)
## Iteration 200: error is 74.933638 (50 iterations in 5.99 seconds)
## Iteration 250: error is 74.921967 (50 iterations in 8.82 seconds)
## Iteration 300: error is 1.997133 (50 iterations in 7.94 seconds)
## Iteration 350: error is 1.734541 (50 iterations in 7.52 seconds)
## Iteration 400: error is 1.610010 (50 iterations in 8.07 seconds)
## Iteration 450: error is 1.535765 (50 iterations in 7.63 seconds)
## Iteration 500: error is 1.486628 (50 iterations in 7.59 seconds)
## Fitting performed in 94.23 seconds.
```



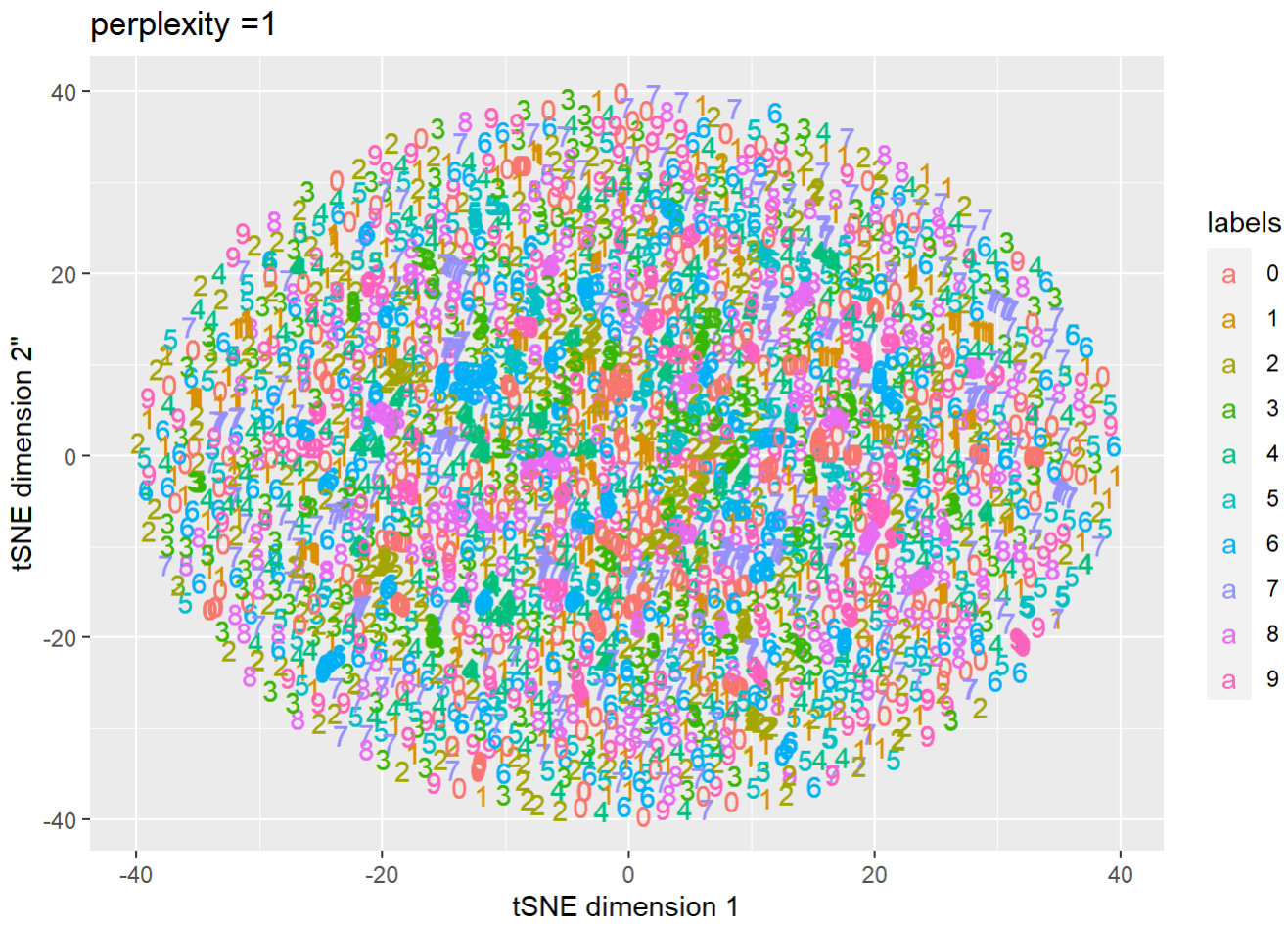
Final embedding is more stable as perplexity increases. But when perplexity is larger than 100, the embedding becomes less dense and deviate from its original shape.

– d. If the perplexity is set to 1 what would the distribution of values look like in 2d, provide an explanation as to why.

```
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = 1,
                  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 = 1.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 14.40 seconds (sparsity = 0.000442)!
## Learning embedding...
## Iteration 50: error is 135.150077 (50 iterations in 4.89 seconds)
## Iteration 100: error is 116.602876 (50 iterations in 3.96 seconds)
## Iteration 150: error is 108.499873 (50 iterations in 5.37 seconds)
## Iteration 200: error is 104.022373 (50 iterations in 7.35 seconds)
## Iteration 250: error is 100.905638 (50 iterations in 6.12 seconds)
## Iteration 300: error is 5.267270 (50 iterations in 5.32 seconds)
## Iteration 350: error is 4.629215 (50 iterations in 6.00 seconds)
## Iteration 400: error is 4.131008 (50 iterations in 8.34 seconds)
## Iteration 450: error is 3.740237 (50 iterations in 7.54 seconds)
## Iteration 500: error is 3.426381 (50 iterations in 8.28 seconds)
## Fitting performed in 63.15 seconds.
```

```
# 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"')+ggtitle(   ('perplexity =1')
)
```



When perplexity is small, local variation dominate so no cluster is formed.

– e. How about if the perplexity is set to 5000 what would the distribution of values look like in 2d, provide an explanation as to why. Note: don't try this on your laptop - Rtsne is not optimized for distance calculations and will take a long time to run.

Number of data points for each number

```
table(mnist_raw$X1)
```

```
##
##    0    1    2    3    4    5    6    7    8    9
## 5923 6742 5958 6131 5842 5421 5918 6265 5851 5949
```

If we set perplexity = 5000, which is close to the number of data points for each number, the final embedding will be 10 dense point.

– f. Plot iter_cost (KL divergence) for against perplexity, what is the optimal perplexity value from the set of perplexities above, why?

```
REPS =  2; # Number of random starts

#Rtsne(X = first_10k_samples, dims = 2, perplexity = 1, theta = 0.5, eta = 200,pca = TRUE, verbo
se = TRUE,max_iter = 500)

per1 <- sapply(1:REPS, function(u) {set.seed(u);
  Rtsne(X = first_10k_samples, dims = 2, perplexity = 1, theta = 0.5, eta = 200,pca = TRUE, verb
ose = TRUE,max_iter = 500)}, simplify = FALSE)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 1.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 23.84 seconds (sparsity = 0.000442)!
## Learning embedding...
## Iteration 50: error is 135.158513 (50 iterations in 3.69 seconds)
## Iteration 100: error is 116.573121 (50 iterations in 4.82 seconds)
## Iteration 150: error is 108.489790 (50 iterations in 5.41 seconds)
## Iteration 200: error is 104.000924 (50 iterations in 5.65 seconds)
## Iteration 250: error is 100.888704 (50 iterations in 5.53 seconds)
## Iteration 300: error is 5.265234 (50 iterations in 4.80 seconds)
## Iteration 350: error is 4.625787 (50 iterations in 5.91 seconds)
## Iteration 400: error is 4.127706 (50 iterations in 5.01 seconds)
## Iteration 450: error is 3.736811 (50 iterations in 5.23 seconds)
## Iteration 500: error is 3.423765 (50 iterations in 5.39 seconds)
## Fitting performed in 51.43 seconds.
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 1.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 18.98 seconds (sparsity = 0.000442)!
## Learning embedding...
## Iteration 50: error is 135.155969 (50 iterations in 4.19 seconds)
## Iteration 100: error is 116.597244 (50 iterations in 4.08 seconds)
## Iteration 150: error is 108.492218 (50 iterations in 5.29 seconds)
## Iteration 200: error is 104.006162 (50 iterations in 5.89 seconds)
## Iteration 250: error is 100.889950 (50 iterations in 6.21 seconds)
## Iteration 300: error is 5.266257 (50 iterations in 5.57 seconds)
## Iteration 350: error is 4.628365 (50 iterations in 5.69 seconds)
## Iteration 400: error is 4.130963 (50 iterations in 6.41 seconds)
## Iteration 450: error is 3.740446 (50 iterations in 6.74 seconds)
## Iteration 500: error is 3.426703 (50 iterations in 8.48 seconds)
## Fitting performed in 58.56 seconds.
```

```
per20 <- sapply(1:REPS, function(u) {set.seed(u);
  Rtsne(X = first_10k_samples, dims = 2, perplexity = 20, theta = 0.5, eta = 200,pca = TRUE, ver
bose = TRUE,max_iter = 500)}, simplify = FALSE)
```

```
## 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 28.34 seconds (sparsity = 0.008217)!
## Learning embedding...
## Iteration 50: error is 102.343942 (50 iterations in 4.24 seconds)
## Iteration 100: error is 92.830084 (50 iterations in 3.87 seconds)
## Iteration 150: error is 88.371685 (50 iterations in 4.57 seconds)
## Iteration 200: error is 87.697376 (50 iterations in 3.72 seconds)
## Iteration 250: error is 87.512289 (50 iterations in 3.75 seconds)
## Iteration 300: error is 3.401974 (50 iterations in 3.74 seconds)
## Iteration 350: error is 2.951175 (50 iterations in 3.09 seconds)
## Iteration 400: error is 2.700691 (50 iterations in 3.98 seconds)
## Iteration 450: error is 2.533665 (50 iterations in 3.00 seconds)
## Iteration 500: error is 2.412077 (50 iterations in 3.81 seconds)
## Fitting performed in 37.79 seconds.
## 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 21.81 seconds (sparsity = 0.008217)!
## Learning embedding...
## Iteration 50: error is 102.343945 (50 iterations in 4.43 seconds)
## Iteration 100: error is 93.901087 (50 iterations in 3.97 seconds)
## Iteration 150: error is 89.236969 (50 iterations in 3.12 seconds)
## Iteration 200: error is 88.429102 (50 iterations in 3.69 seconds)
## Iteration 250: error is 88.186193 (50 iterations in 3.34 seconds)
## Iteration 300: error is 3.441329 (50 iterations in 3.91 seconds)
## Iteration 350: error is 2.984238 (50 iterations in 3.77 seconds)
## Iteration 400: error is 2.731061 (50 iterations in 3.11 seconds)
## Iteration 450: error is 2.561714 (50 iterations in 3.56 seconds)
## Iteration 500: error is 2.437756 (50 iterations in 2.96 seconds)
## Fitting performed in 35.85 seconds.
```

```
per60 <- sapply(1:REPS, function(u) {set.seed(u);
  Rtsne(X = first_10k_samples, dims = 2, perplexity = 60, theta = 0.5, eta = 200,pca = TRUE, ver
bose = TRUE,max_iter = 500)}, simplify = FALSE)
```

```
## 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 26.73 seconds (sparsity = 0.024328)!
## Learning embedding...
## Iteration 50: error is 89.428526 (50 iterations in 4.89 seconds)
## Iteration 100: error is 84.262063 (50 iterations in 5.63 seconds)
## Iteration 150: error is 81.456988 (50 iterations in 4.72 seconds)
## Iteration 200: error is 81.376962 (50 iterations in 4.47 seconds)
## Iteration 250: error is 81.370881 (50 iterations in 4.43 seconds)
## Iteration 300: error is 2.611131 (50 iterations in 3.99 seconds)
## Iteration 350: error is 2.270123 (50 iterations in 4.02 seconds)
## Iteration 400: error is 2.092954 (50 iterations in 4.11 seconds)
## Iteration 450: error is 1.982012 (50 iterations in 4.50 seconds)
## Iteration 500: error is 1.906667 (50 iterations in 4.54 seconds)
## Fitting performed in 45.30 seconds.
## 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 29.42 seconds (sparsity = 0.024328)!
## Learning embedding...
## Iteration 50: error is 89.428527 (50 iterations in 4.74 seconds)
## Iteration 100: error is 85.005670 (50 iterations in 4.69 seconds)
## Iteration 150: error is 81.667141 (50 iterations in 4.69 seconds)
## Iteration 200: error is 81.590959 (50 iterations in 4.65 seconds)
## Iteration 250: error is 81.573274 (50 iterations in 4.57 seconds)
## Iteration 300: error is 2.623011 (50 iterations in 4.32 seconds)
## Iteration 350: error is 2.282006 (50 iterations in 3.93 seconds)
## Iteration 400: error is 2.107884 (50 iterations in 3.92 seconds)
## Iteration 450: error is 1.997445 (50 iterations in 6.46 seconds)
## Iteration 500: error is 1.920451 (50 iterations in 6.23 seconds)
## Fitting performed in 48.20 seconds.
```

```
per100 <- sapply(1:REPS, function(u) {set.seed(u);
  Rtsne(X = first_10k_samples, dims = 2, perplexity = 100, theta = 0.5, eta = 200,pca = TRUE, ve
rbose = TRUE,max_iter = 500)}, simplify = FALSE)
```

```
## 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 36.08 seconds (sparsity = 0.040571)!
## Learning embedding...
## Iteration 50: error is 83.371128 (50 iterations in 5.61 seconds)
## Iteration 100: error is 80.446025 (50 iterations in 5.18 seconds)
## Iteration 150: error is 78.422442 (50 iterations in 4.93 seconds)
## Iteration 200: error is 78.301383 (50 iterations in 4.64 seconds)
## Iteration 250: error is 78.266912 (50 iterations in 5.15 seconds)
## Iteration 300: error is 2.271216 (50 iterations in 9.81 seconds)
## Iteration 350: error is 1.974392 (50 iterations in 17.34 seconds)
## Iteration 400: error is 1.831235 (50 iterations in 18.92 seconds)
## Iteration 450: error is 1.744442 (50 iterations in 14.30 seconds)
## Iteration 500: error is 1.685932 (50 iterations in 14.38 seconds)
## Fitting performed in 100.25 seconds.
## 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 41.95 seconds (sparsity = 0.040571)!
## Learning embedding...
## Iteration 50: error is 83.371128 (50 iterations in 7.12 seconds)
## Iteration 100: error is 82.556976 (50 iterations in 8.38 seconds)
## Iteration 150: error is 78.350615 (50 iterations in 10.63 seconds)
## Iteration 200: error is 78.283846 (50 iterations in 8.78 seconds)
## Iteration 250: error is 78.252541 (50 iterations in 7.11 seconds)
## Iteration 300: error is 2.265345 (50 iterations in 8.33 seconds)
## Iteration 350: error is 1.974379 (50 iterations in 7.53 seconds)
## Iteration 400: error is 1.834015 (50 iterations in 7.73 seconds)
## Iteration 450: error is 1.747152 (50 iterations in 5.69 seconds)
## Iteration 500: error is 1.688130 (50 iterations in 5.86 seconds)
## Fitting performed in 77.15 seconds.
```

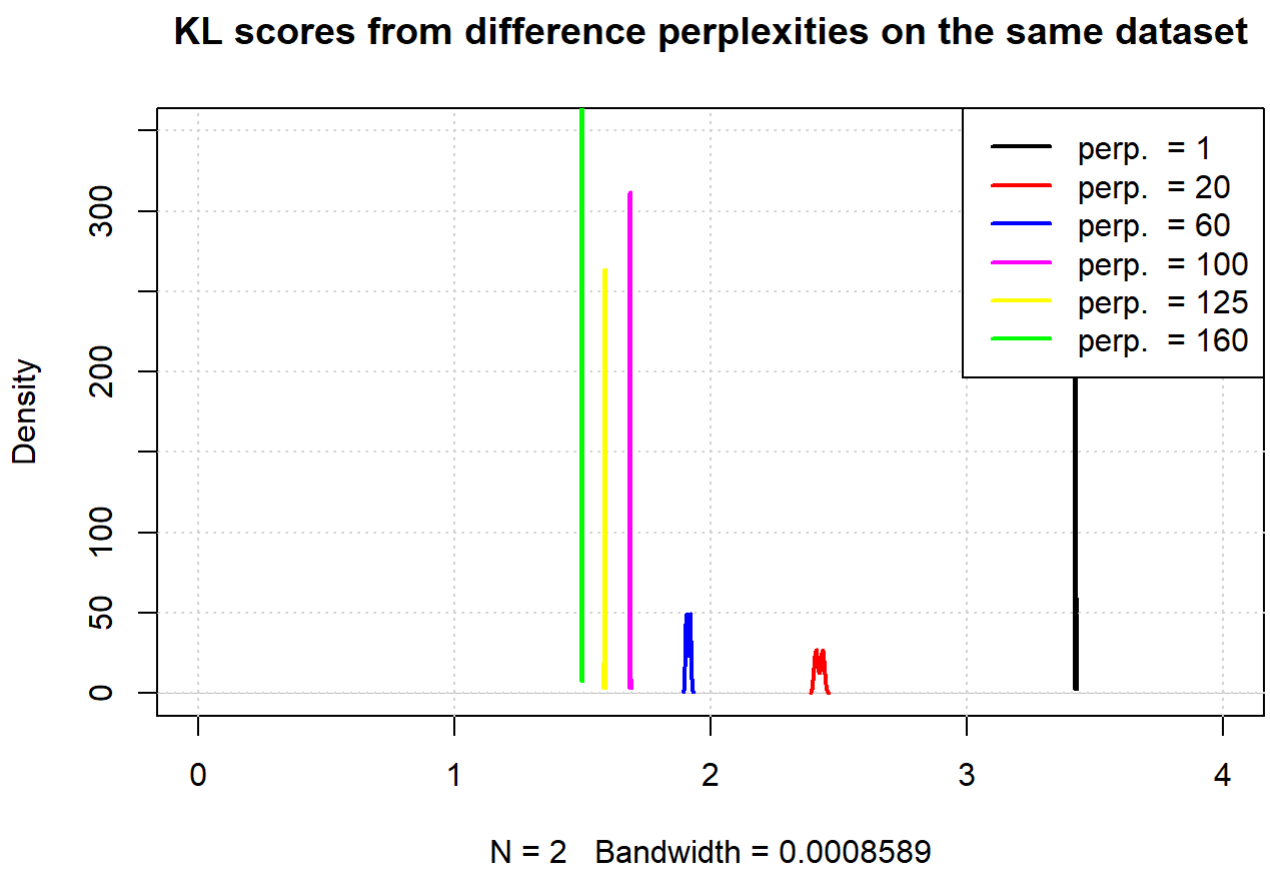
```
per125 <- sapply(1:REPS, function(u) {set.seed(u);
  Rtsne(X = first_10k_samples, dims = 2, perplexity = 125, theta = 0.5, eta = 200,pca = TRUE, ve
rbose = TRUE,max_iter = 500)}, simplify = FALSE)
```

```
## 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 51.09 seconds (sparsity = 0.050806)!
## Learning embedding...
## Iteration 50: error is 80.714217 (50 iterations in 7.28 seconds)
## Iteration 100: error is 79.080034 (50 iterations in 7.94 seconds)
## Iteration 150: error is 76.987311 (50 iterations in 8.20 seconds)
## Iteration 200: error is 76.745694 (50 iterations in 8.25 seconds)
## Iteration 250: error is 76.728442 (50 iterations in 9.73 seconds)
## Iteration 300: error is 2.138554 (50 iterations in 10.37 seconds)
## Iteration 350: error is 1.853870 (50 iterations in 11.66 seconds)
## Iteration 400: error is 1.719228 (50 iterations in 12.20 seconds)
## Iteration 450: error is 1.639136 (50 iterations in 11.53 seconds)
## Iteration 500: error is 1.586096 (50 iterations in 12.23 seconds)
## Fitting performed in 99.38 seconds.
## 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 62.15 seconds (sparsity = 0.050806)!
## Learning embedding...
## Iteration 50: error is 80.714217 (50 iterations in 8.19 seconds)
## Iteration 100: error is 80.600711 (50 iterations in 12.41 seconds)
## Iteration 150: error is 76.783943 (50 iterations in 15.99 seconds)
## Iteration 200: error is 76.715583 (50 iterations in 8.03 seconds)
## Iteration 250: error is 76.709750 (50 iterations in 8.93 seconds)
## Iteration 300: error is 2.141902 (50 iterations in 14.32 seconds)
## Iteration 350: error is 1.859181 (50 iterations in 6.34 seconds)
## Iteration 400: error is 1.721148 (50 iterations in 5.92 seconds)
## Iteration 450: error is 1.640448 (50 iterations in 5.91 seconds)
## Iteration 500: error is 1.588689 (50 iterations in 6.02 seconds)
## Fitting performed in 92.04 seconds.
```

```
per160 <- sapply(1:REPS, function(u) {set.seed(u);
  Rtsne(X = first_10k_samples, dims = 2, perplexity = 160, theta = 0.5, eta = 200,pca = TRUE, ve
rbose = TRUE,max_iter = 500)}, simplify = FALSE)
```

```
## 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 46.36 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 6.65 seconds)
## Iteration 100: error is 77.534004 (50 iterations in 6.87 seconds)
## Iteration 150: error is 75.214318 (50 iterations in 6.52 seconds)
## Iteration 200: error is 75.026680 (50 iterations in 6.35 seconds)
## Iteration 250: error is 74.914691 (50 iterations in 6.15 seconds)
## Iteration 300: error is 2.016431 (50 iterations in 5.95 seconds)
## Iteration 350: error is 1.752646 (50 iterations in 5.92 seconds)
## Iteration 400: error is 1.622257 (50 iterations in 5.85 seconds)
## Iteration 450: error is 1.545680 (50 iterations in 5.72 seconds)
## Iteration 500: error is 1.497488 (50 iterations in 7.30 seconds)
## Fitting performed in 63.27 seconds.
## 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 78.09 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 14.51 seconds)
## Iteration 100: error is 77.763665 (50 iterations in 20.36 seconds)
## Iteration 150: error is 75.294135 (50 iterations in 32.65 seconds)
## Iteration 200: error is 74.935435 (50 iterations in 14.55 seconds)
## Iteration 250: error is 74.919567 (50 iterations in 14.14 seconds)
## Iteration 300: error is 2.010377 (50 iterations in 15.31 seconds)
## Iteration 350: error is 1.748823 (50 iterations in 22.82 seconds)
## Iteration 400: error is 1.623011 (50 iterations in 15.01 seconds)
## Iteration 450: error is 1.548752 (50 iterations in 26.54 seconds)
## Iteration 500: error is 1.498493 (50 iterations in 17.64 seconds)
## Fitting performed in 193.53 seconds.
```

```
costs <- c( sapply(per1, function(u) min(u$itercosts)),
            sapply(per20, function(u) min(u$itercosts)),
            sapply(per60, function(u) min(u$itercosts)),
            sapply(per100, function(u) min(u$itercosts)),
            sapply(per125, function(u) min(u$itercosts)),
            sapply(per160, function(u) min(u$itercosts)))

perplexities <- c( rep(1,REPS), rep(20,REPS), rep(60,REPS), rep(100,REPS),rep(125,REPS),rep(160,
REPS))

plot(density(costs[perplexities == 1]), lwd= 2,xlim= c(0,4), ylim=c(0,350),
     main='KL scores from difference perplexities on the same dataset'); grid()
lines(density(costs[perplexities == 20]), col='red', lwd= 2);
lines(density(costs[perplexities == 60]), col='blue', lwd= 2);
lines(density(costs[perplexities == 100]), col='magenta', lwd= 2);
lines(density(costs[perplexities == 125]), col='yellow', lwd= 2);
lines(density(costs[perplexities == 160]), col='green', lwd= 2)
legend('topright', col=c('black','red','blue','magenta','yellow','green'),
       c('perp.  = 1', 'perp.  = 20', 'perp.  = 60','perp.  = 100','perp.  = 125','perp.  = 160'
), lwd = 2)
```



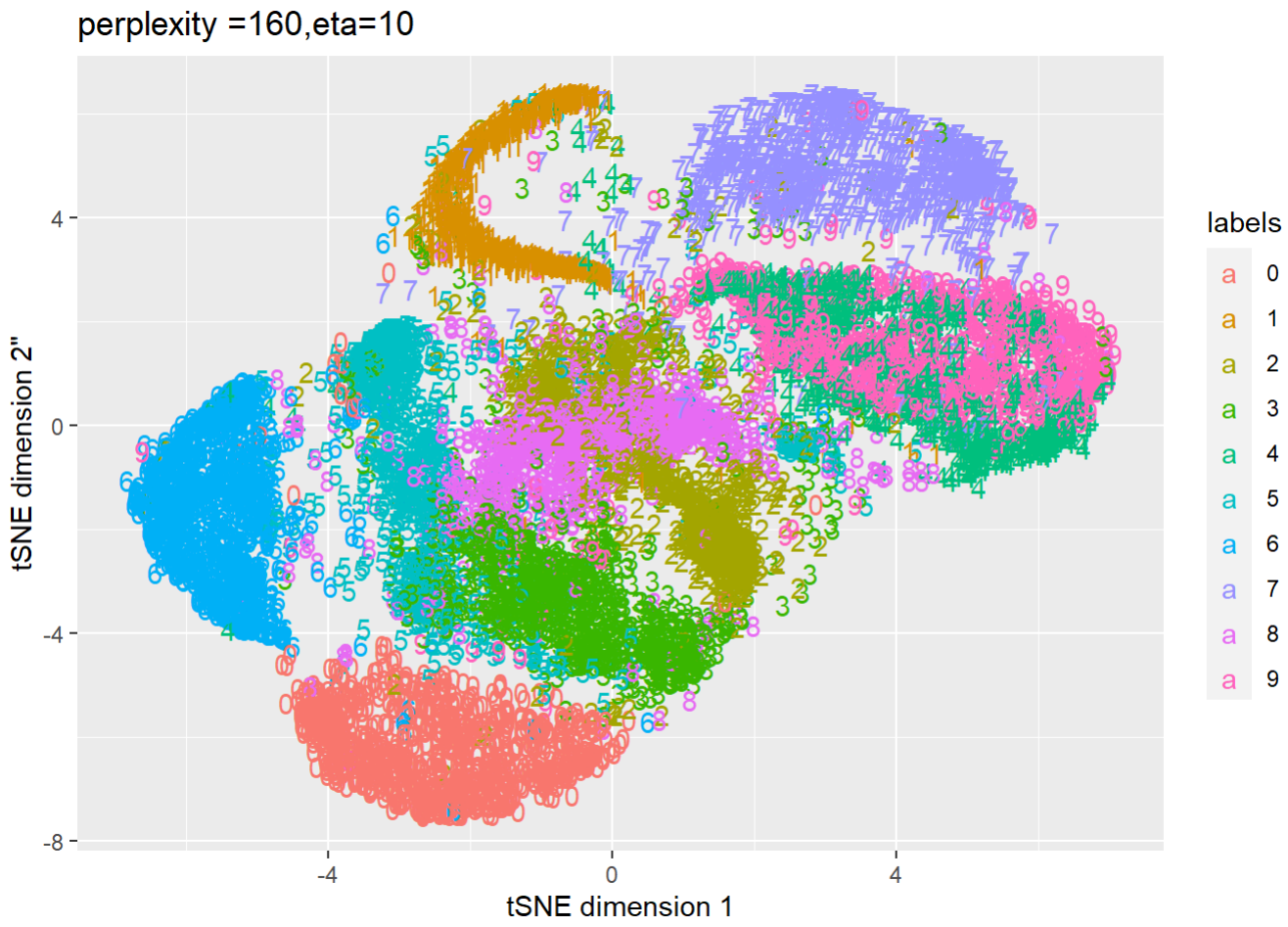
KL Score is the highest when perplexity=160

– g. Plot the embeddings for eta=(10,100,200) while keeping max_iter and your optimal perplexity value selected above constant. What do you notice?

```
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 102.60 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 12.63 seconds)
## Iteration 100: error is 77.767491 (50 iterations in 7.74 seconds)
## Iteration 150: error is 77.767491 (50 iterations in 8.39 seconds)
## Iteration 200: error is 77.767491 (50 iterations in 8.57 seconds)
## Iteration 250: error is 77.767491 (50 iterations in 10.10 seconds)
## Iteration 300: error is 3.959960 (50 iterations in 8.29 seconds)
## Iteration 350: error is 2.870271 (50 iterations in 8.65 seconds)
## Iteration 400: error is 2.459827 (50 iterations in 7.20 seconds)
## Iteration 450: error is 2.244453 (50 iterations in 7.72 seconds)
## Iteration 500: error is 2.104838 (50 iterations in 10.76 seconds)
## Fitting performed in 90.06 seconds.
```

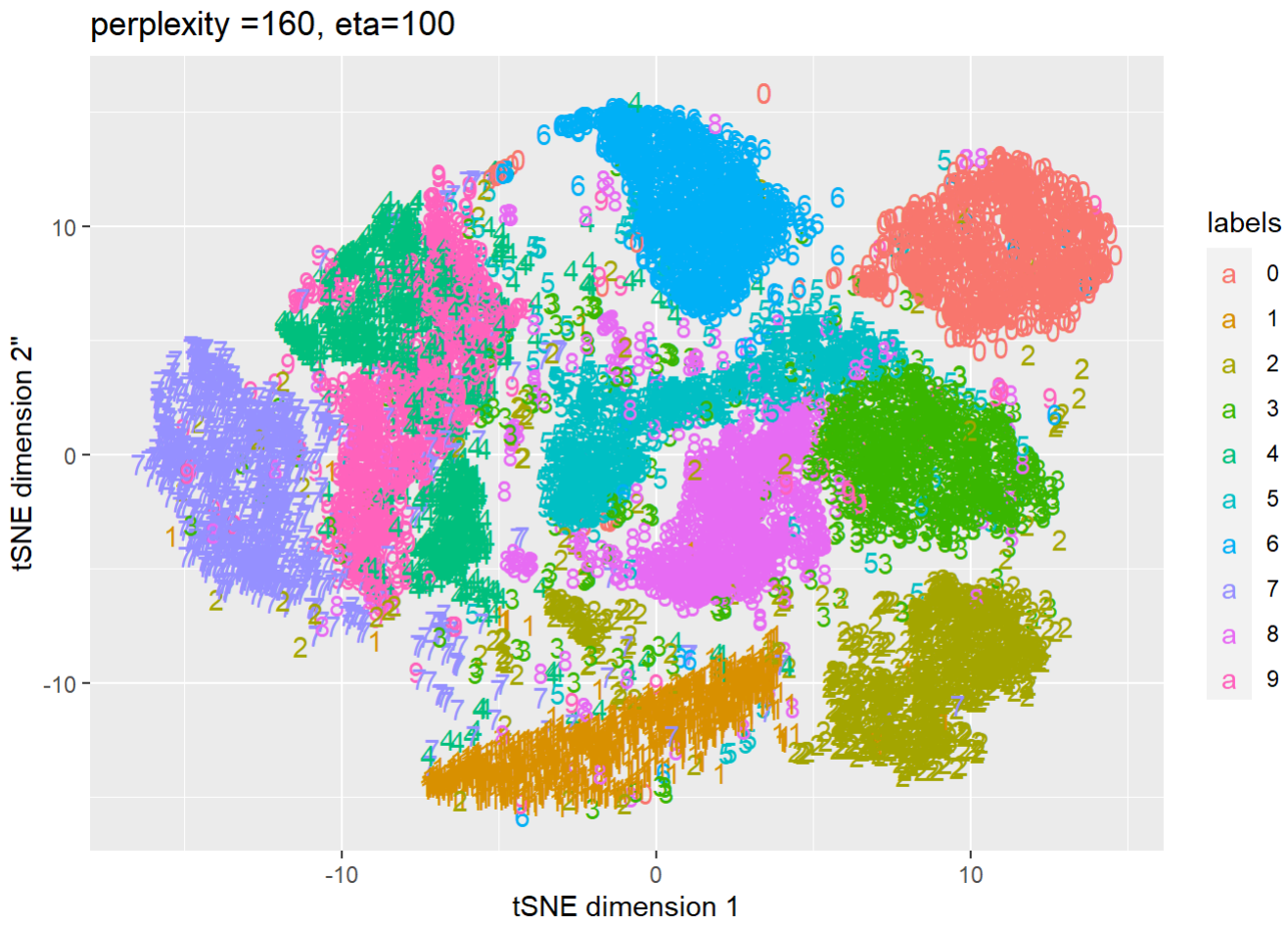
```
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2"')+ggtitle(   ('perplexity =160,
eta=10')   )
```




```
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 51.44 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 8.00 seconds)
## Iteration 100: error is 77.767482 (50 iterations in 7.49 seconds)
## Iteration 150: error is 75.842977 (50 iterations in 9.13 seconds)
## Iteration 200: error is 74.944037 (50 iterations in 8.04 seconds)
## Iteration 250: error is 74.918170 (50 iterations in 9.20 seconds)
## Iteration 300: error is 2.067613 (50 iterations in 6.85 seconds)
## Iteration 350: error is 1.810103 (50 iterations in 6.86 seconds)
## Iteration 400: error is 1.682254 (50 iterations in 6.71 seconds)
## Iteration 450: error is 1.603559 (50 iterations in 7.11 seconds)
## Iteration 500: error is 1.551170 (50 iterations in 6.68 seconds)
## Fitting performed in 76.07 seconds.
```

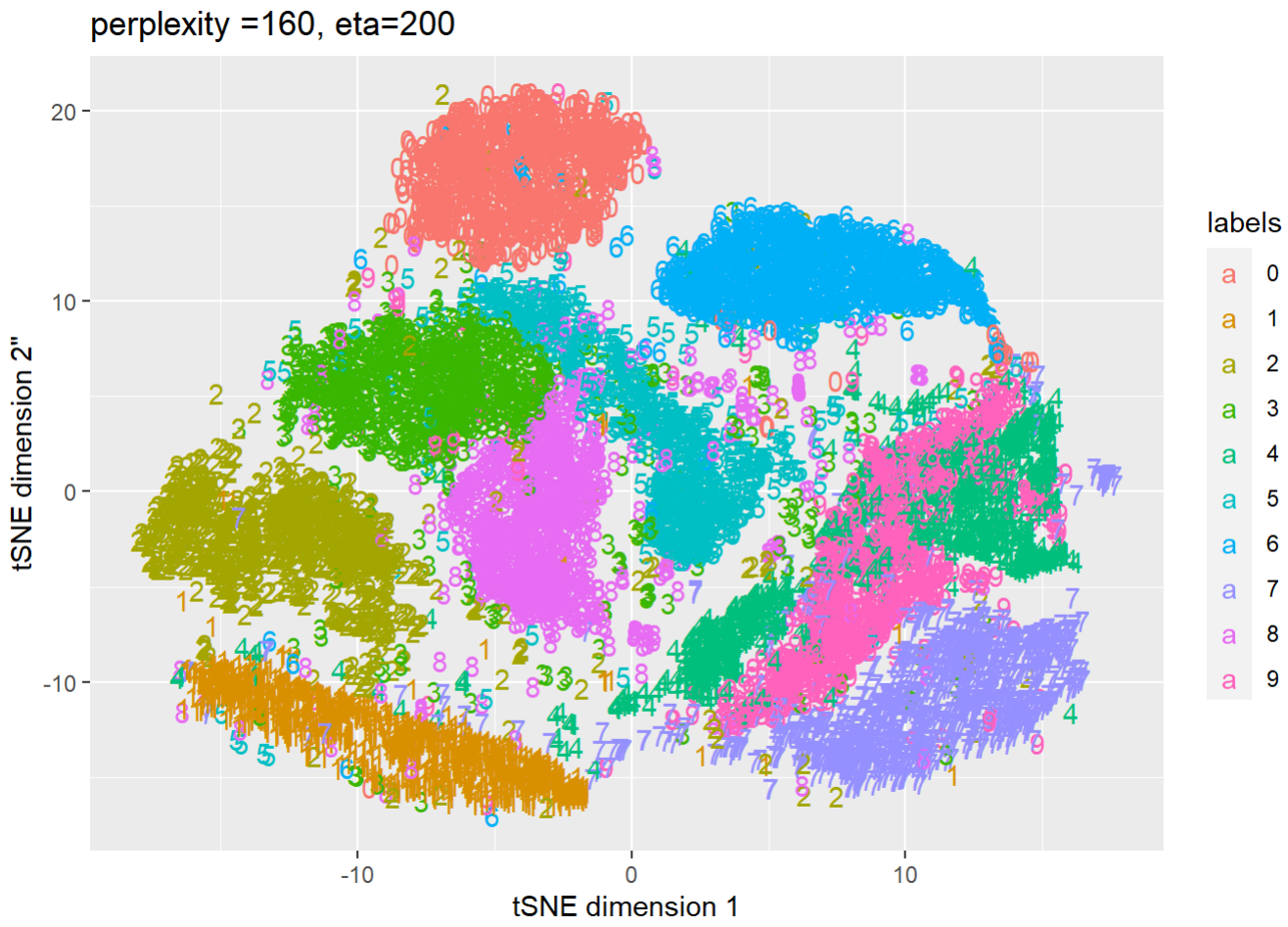
```
# Visualizing TSNE output
embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                        labels = as.character(first_10k_samples_labels))
ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2"')+ggtitle(   ('perplexity =160,
eta=100')   )
```




```
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 52.58 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 7.46 seconds)
## Iteration 100: error is 77.704365 (50 iterations in 7.83 seconds)
## Iteration 150: error is 75.221281 (50 iterations in 8.30 seconds)
## Iteration 200: error is 74.944234 (50 iterations in 10.79 seconds)
## Iteration 250: error is 74.919850 (50 iterations in 7.73 seconds)
## Iteration 300: error is 2.021763 (50 iterations in 7.47 seconds)
## Iteration 350: error is 1.742705 (50 iterations in 7.32 seconds)
## Iteration 400: error is 1.613922 (50 iterations in 14.62 seconds)
## Iteration 450: error is 1.538521 (50 iterations in 12.96 seconds)
## Iteration 500: error is 1.488992 (50 iterations in 9.68 seconds)
## Fitting performed in 94.17 seconds.
```

```
# 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"')+ggtitle(   ('perplexity =160,
eta=200')   )
```



As eta (learning rate) increases, running time for get embedding results decreases.