

# HW3

Code ▼

Arshia Singh

March 31, 2020

Collaborators: Xiner, Norman, Jack

## Part 1

### Question 1 - Boosting

Part 1

Q0

Hide

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

```
package <U+393C><U+3E31>tidyverse<U+393C><U+3E32> was built under R version 3.5.3  
m--  [1mAttaching packages[22m ----- tidyverse 1.3.  
0 --[39m  
[30m[32mv[30m [34mggplot2[30m 3.3.0      [32mv[30m [34mpurrr  [30m 0.3.3  
[32mv[30m [34mtibble  [30m 3.0.0      [32mv[30m [34mdplyr  [30m 0.8.5  
[32mv[30m [34mtidyr   [30m 1.0.2      [32mv[30m [34mstringr[30m 1.4.0  
[32mv[30m [34mreadr   [30m 1.3.1      [32mv[30m [34mforcats[30m 0.5.0[39m  
package <U+393C><U+3E31>ggplot2<U+393C><U+3E32> was built under R version 3.5.3package  
<U+393C><U+3E31>tibble<U+393C><U+3E32> was built under R version 3.5.3package <U+393C>  
<U+3E31>tidyr<U+393C><U+3E32> was built under R version 3.5.3package <U+393C><U+3E31>r  
eadr<U+393C><U+3E32> was built under R version 3.5.3package <U+393C><U+3E31>purrr<U+39  
3C><U+3E32> was built under R version 3.5.3package <U+393C><U+3E31>dplyr<U+393C><U+3E3  
2> was built under R version 3.5.3package <U+393C><U+3E31>stringr<U+393C><U+3E32> was  
built under R version 3.5.3package <U+393C><U+3E31>forcats<U+393C><U+3E32> was built u  
nder R version 3.5.3[30m--  [1mConflicts[22m -----  
--- tidyverse_conflicts() --  
[31mx[30m [34mdplyr[30m::[32mfilter()[30m masks [34mstats[30m::filter()  
[31mx[30m [34mdplyr[30m::[32mlag()[30m masks [34mstats[30m::lag()[39m
```

Hide

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

package `rpart` was built under R version 3.5.3

Hide

```

# 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
runboost <- function(v){
  number_of_weak_learners = 100
  number_of_knots_split = 6
  polynomial_degree = 2

  # Fit round 1
  fit=rpart(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 = rpart(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
plot1 <- 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()
print(plot1)

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

new_data = tibble(x = sample(seq(0,4*3,0.001),size = 100,replace = T))

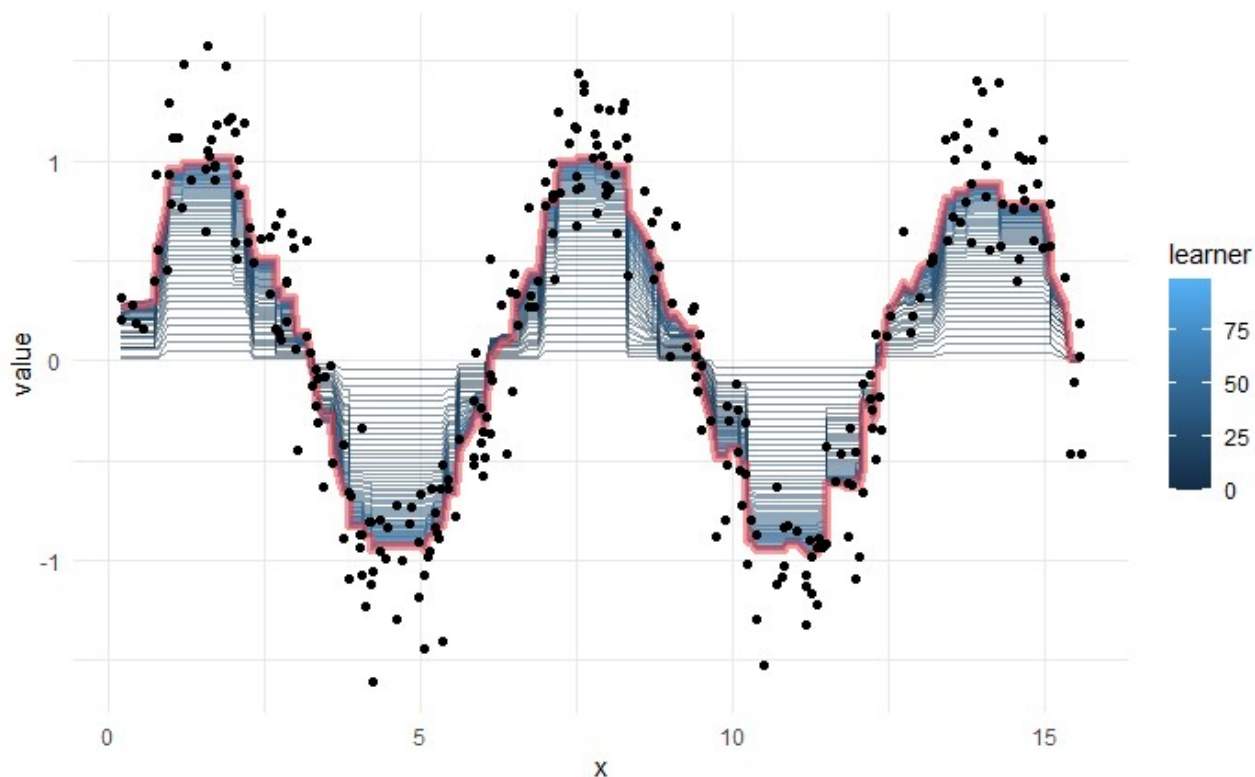
for (i in 1:number_of_weak_learners){
  weak_learner_i = list_of_weak_learners[[i]]

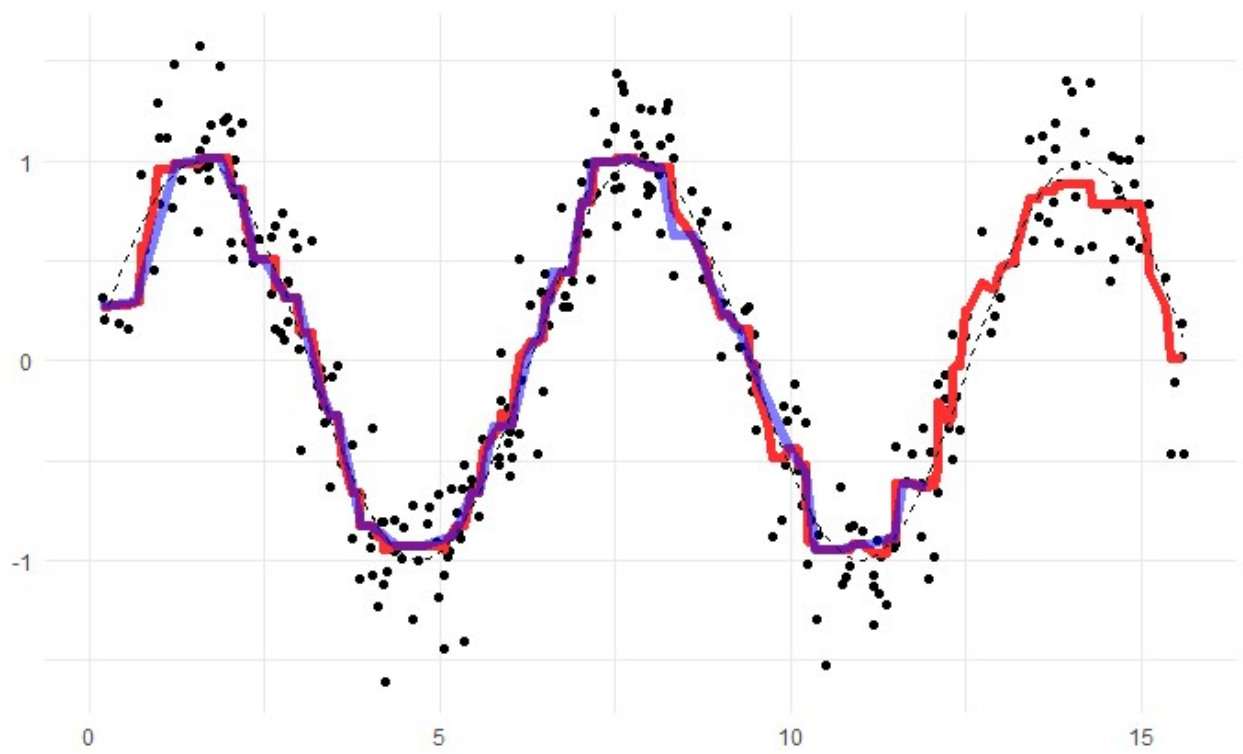
  if (i==1){pred = v*predict(weak_learner_i,new_data)}
  else{pred =pred + v*predict(weak_learner_i,new_data)}

  if(i==number_of_weak_learners){
    new_data = new_data %>% bind_cols(yp=pred)
  }
}

```

```
#####
##### Visualizing boosted vs predicted models #####
#####
plot2 <- ggplot(aes(x=x, y=y),data = tibble(x = df$x, y = df$y))+
  xlab('')+ylab('')+
  geom_point()+
  # Final learner from training data
  geom_line(aes(x = x, y = value, group = learner, color =learner), data = final_learner , color = 'firebrick1',size = 2) +
  # True value
  geom_line(aes(x=x,y=y),data = tibble(x = u,y = sin(u)), color='black',linetype = 'dashed')+ # true values
  # Prediction on new data
  geom_line(aes(x=x,y=yp),data = new_data, color='blue',size = 2,alpha = 0.5)+ # predicted values
  theme_minimal()
print(plot2)
}
runboost(0.05)
```

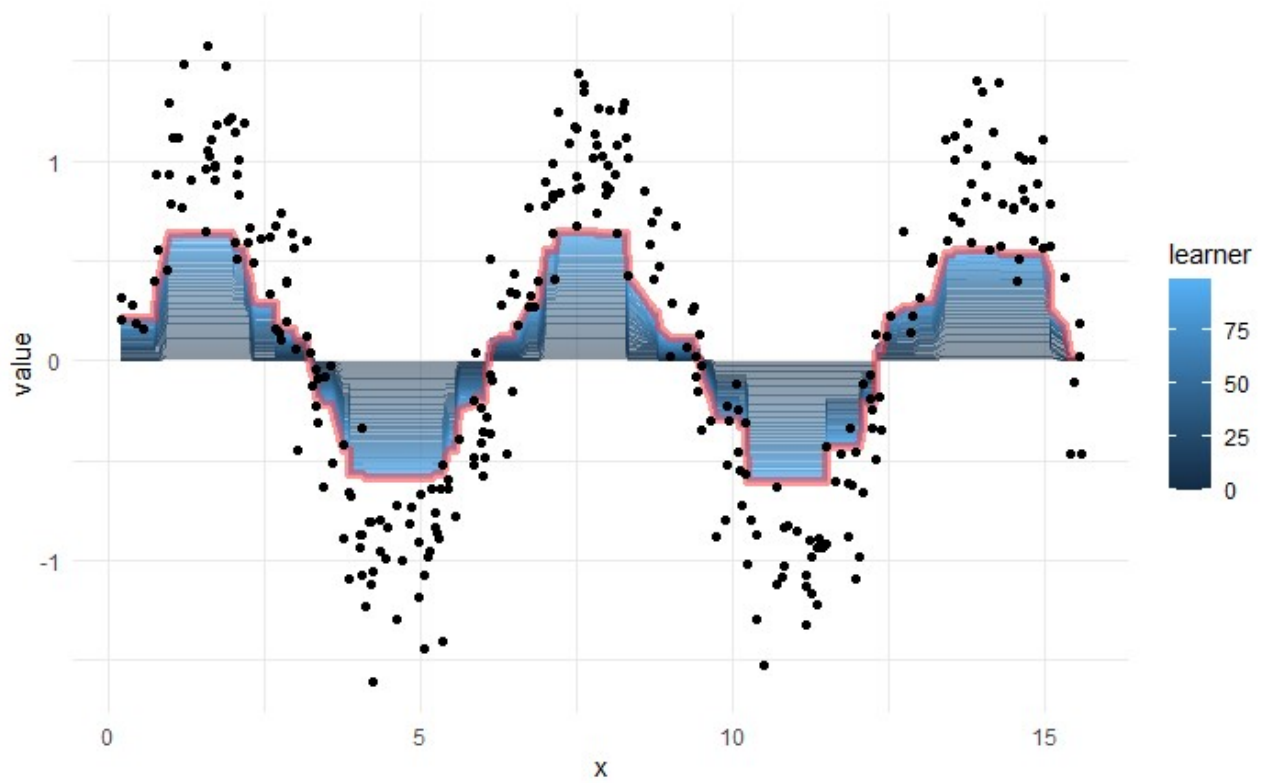


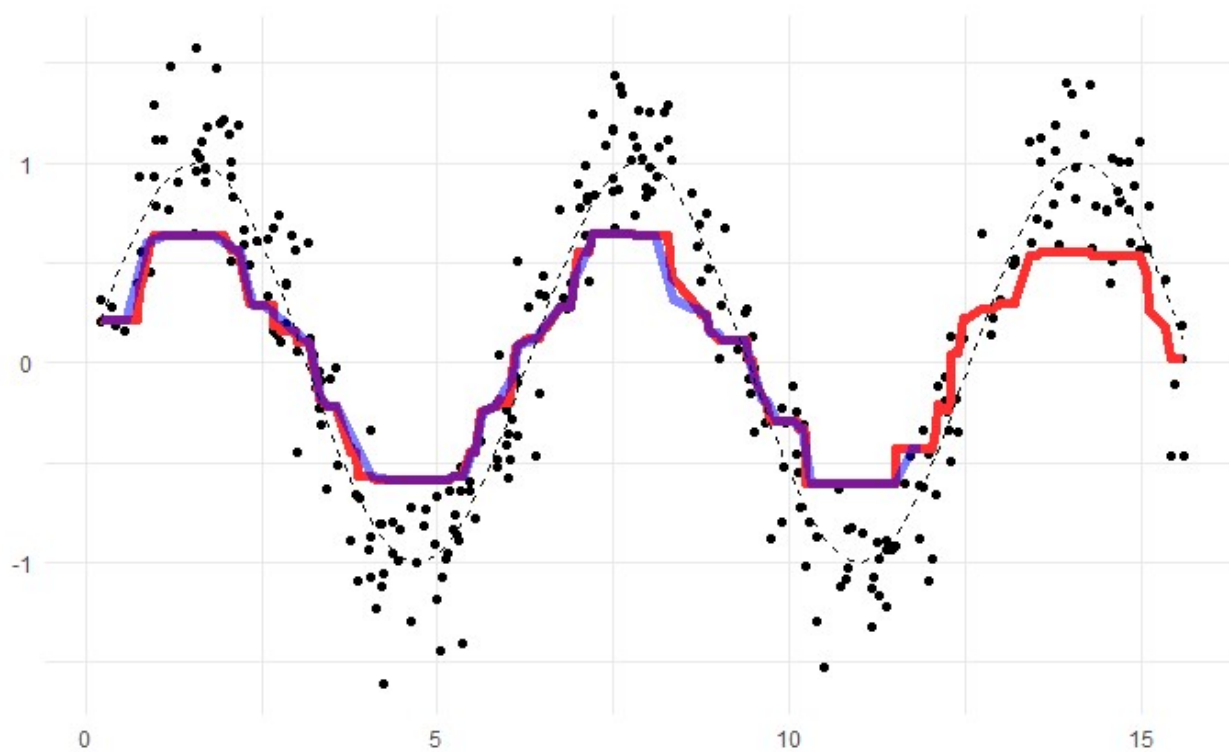


Q1

Hide

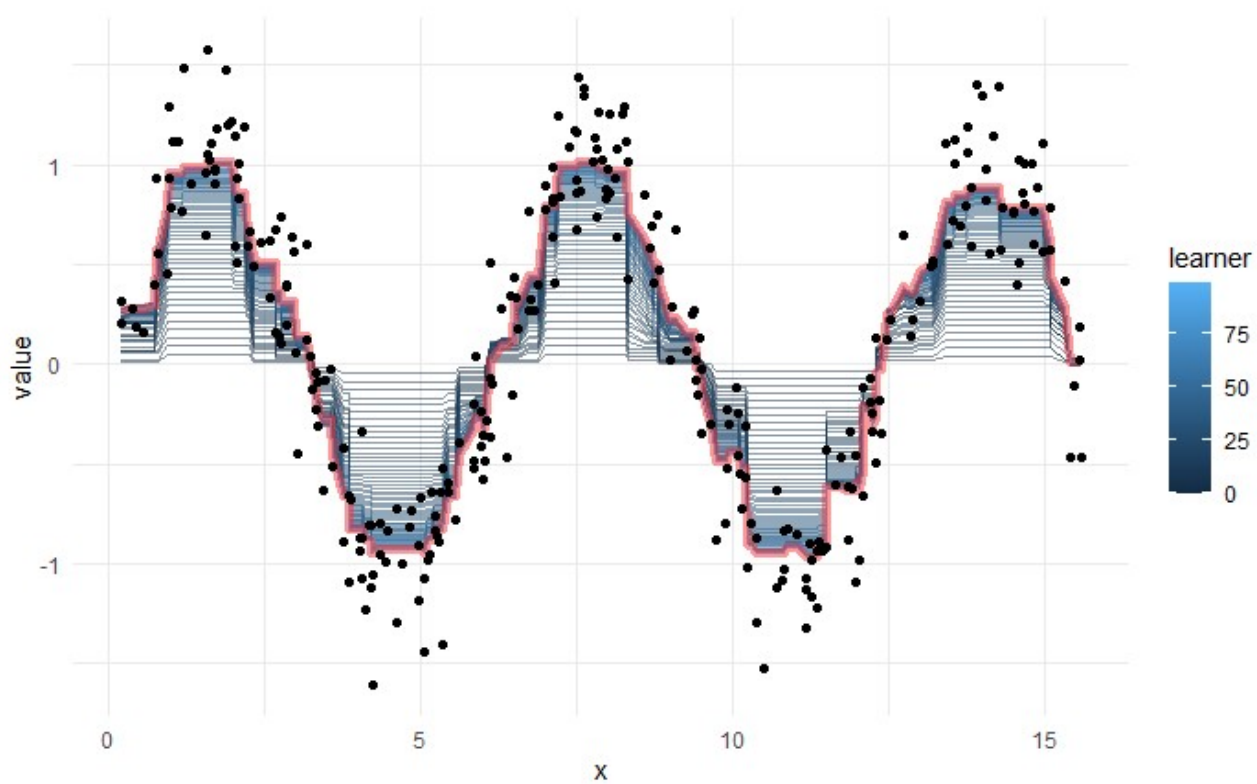
```
runboost(0.01)
```

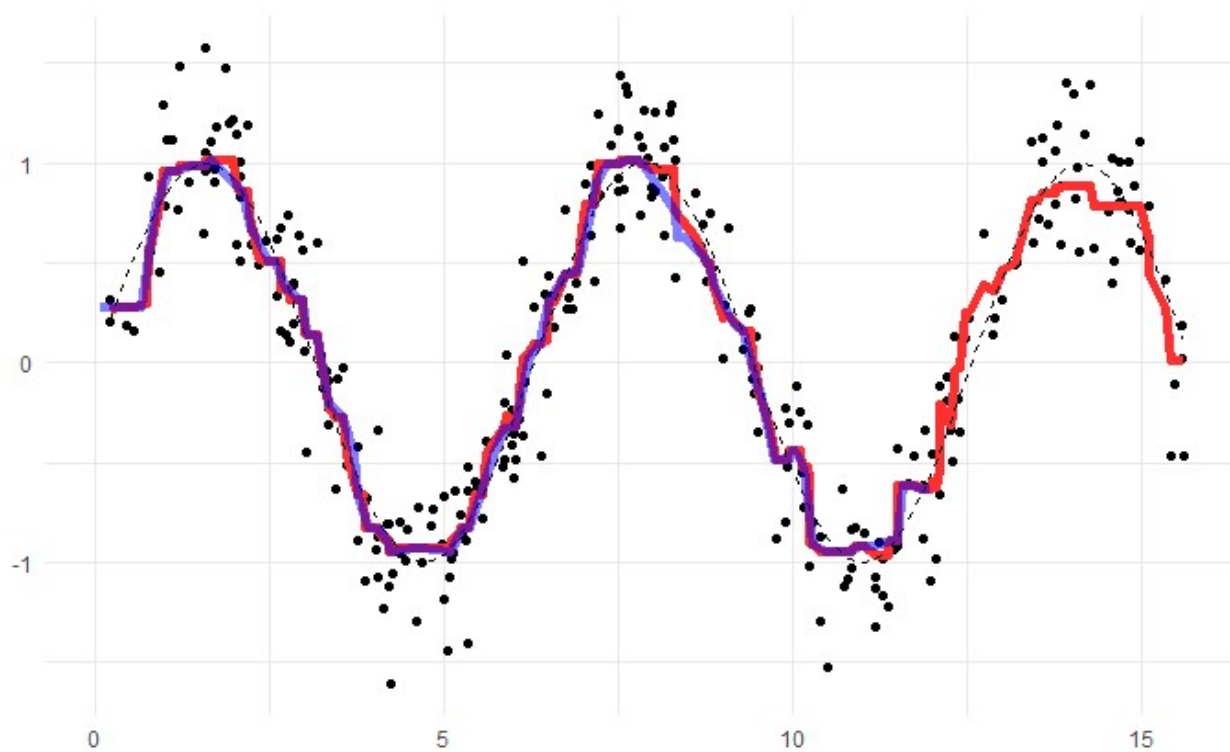




Hide

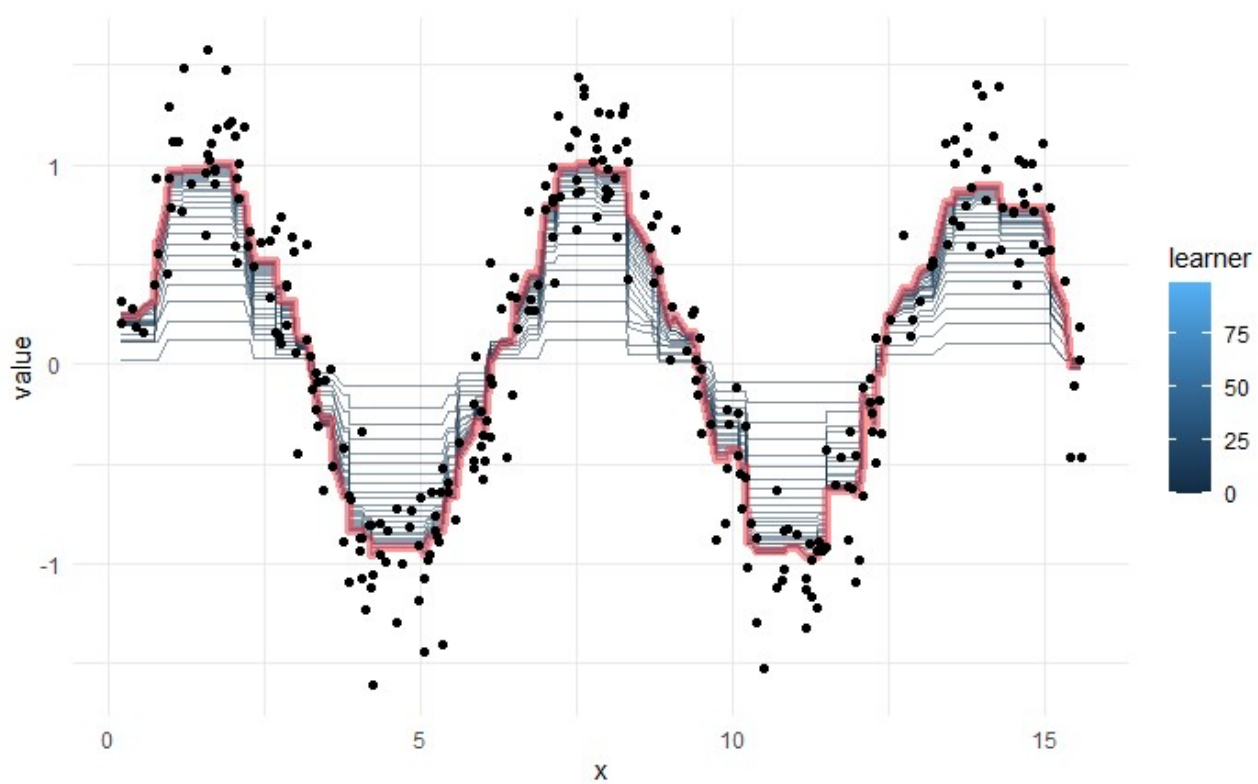
```
runboost(0.05)
```



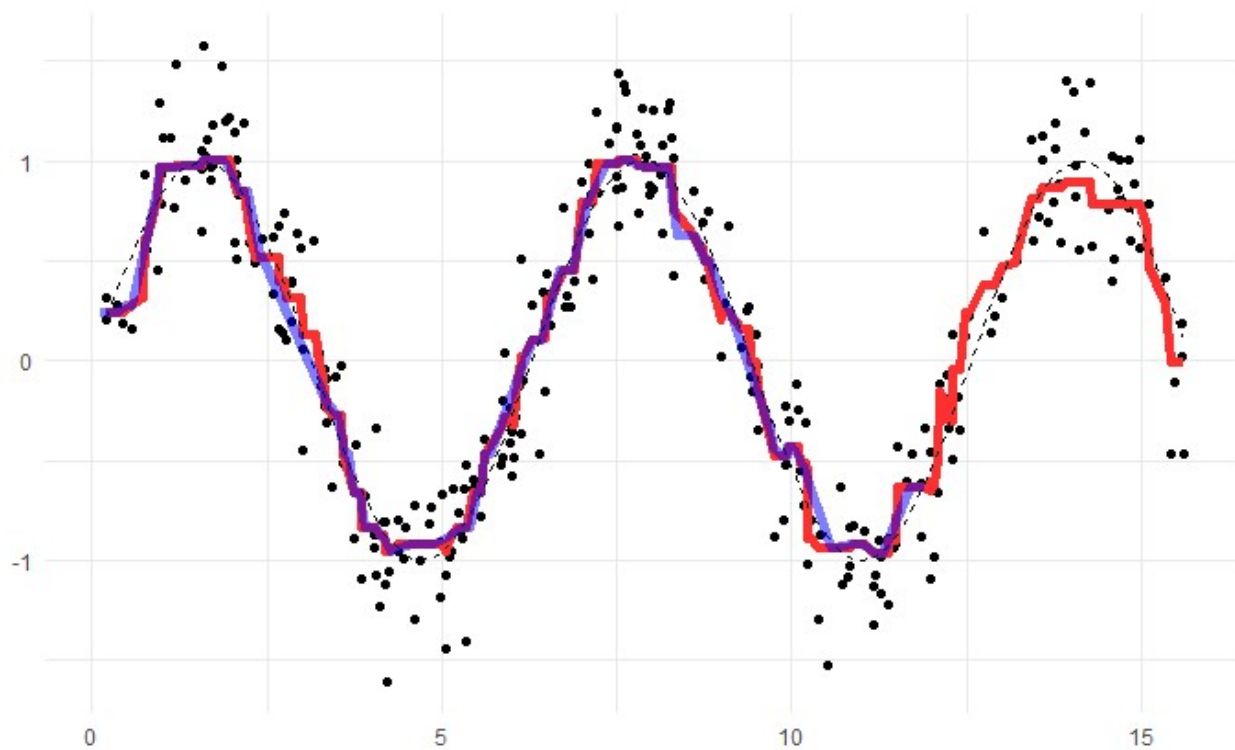


Hide

```
runboost(0.125)
```







It looks like increasing the learning parameter from 0.01 to 0.05 increased the accuracy of the fit significantly, but further increases in the learning parameter did not lead to a better fit.

Q2

Part A

Hide

```

runboost <- function(v){
  number_of_knots_split = 6
  polynomial_degree = 2

  # Fit round 1
  fit=rpart(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 #####
  #####
  mean_vyp = mean(YP)
  t=1
  while(mean_vyp > 0.0001){
    t=t+1
    # Fit linear spline
    fit = rpart(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
    mean_vyp = mean(v*yp)
  }
  print(t)
  #####
  ##### 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)
}

# 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 == (t-1))

# Plot progression of learner
plot1 <- 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()
print(plot1)

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

new_data = tibble(x = sample(seq(0,4*3,0.001),size = 100,replace = T))

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

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

  if(i==t){
    new_data = new_data %>% bind_cols(yp=pred)
  }
}

#####
##### Visualizing boosted vs predicted models #####
#####
plot2 <- ggplot(aes(x=x, y=y),data = tibble(x = df$x, y = df$y))+

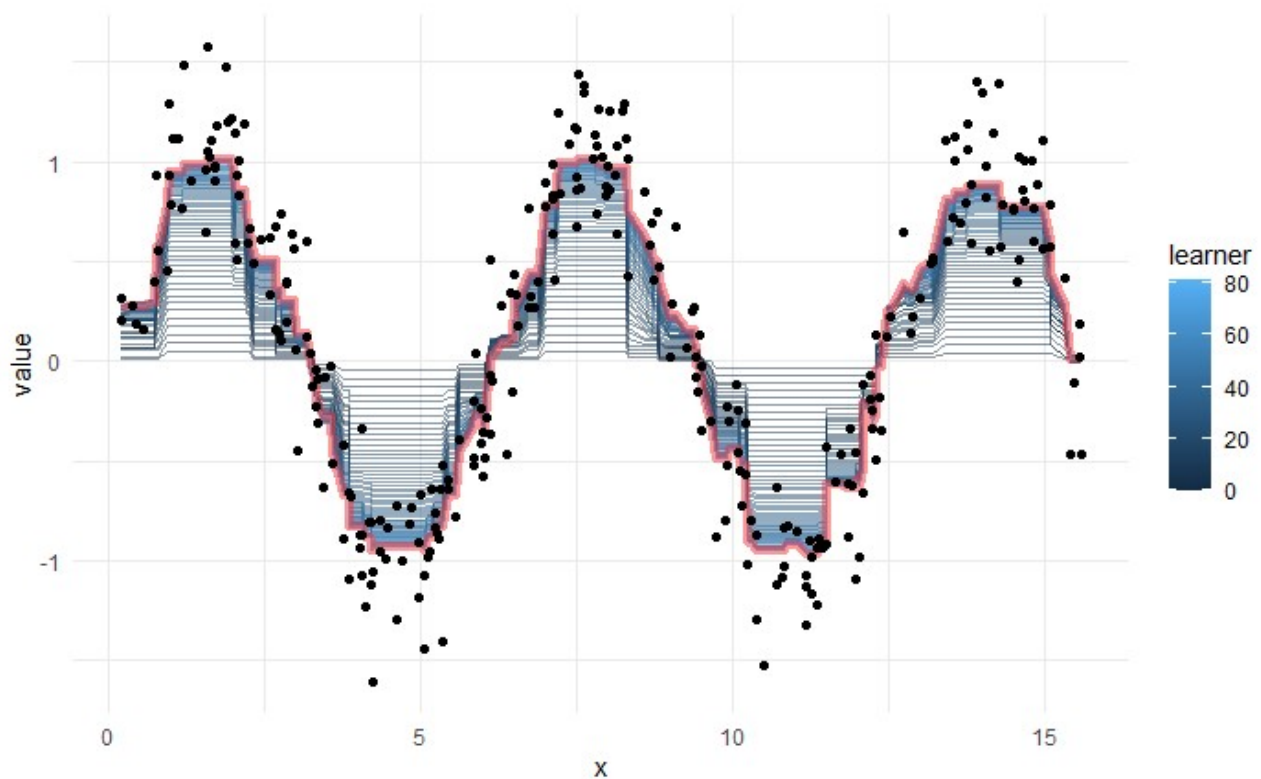
```

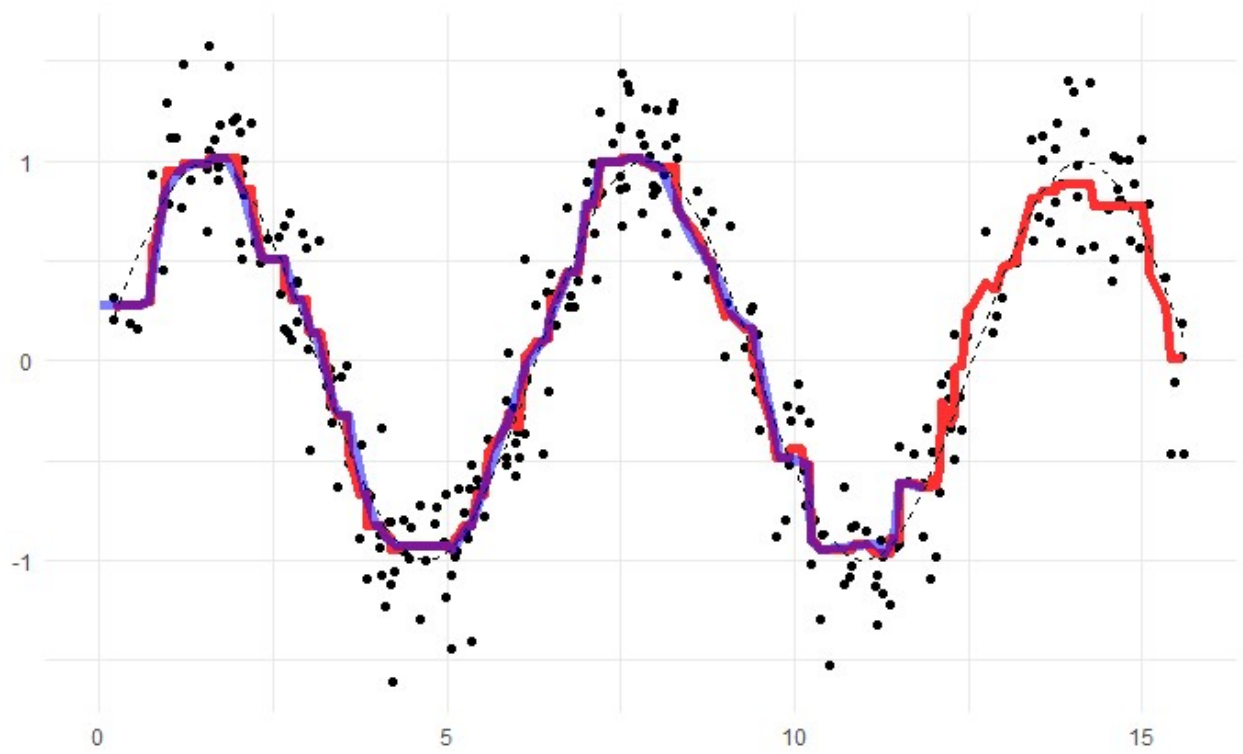
```

xlab('')+ylab('')+
geom_point()+
# Final learner from training data
geom_line(aes(x = x, y = value, group = learner, color = learner), data = final_learner, color = 'firebrick1', size = 2) +
# True value
geom_line(aes(x=x,y=y),data = tibble(x = u,y = sin(u)), color='black',linetype = 'dashed')+ # true values
# Prediction on new data
geom_line(aes(x=x,y=yp),data = new_data, color='blue',size = 2,alpha = 0.5)+ # predicted values
theme_minimal()
print(plot2)
}
runboost(0.05)

```

[1] 82





Part B

There were 82 trees for this run.

Part C

Hide

```

RMSE = function(fit, obs){
  sqrt(mean((fit - obs)^2))
}
runboost <- function(v){
  number_of_knots_split = 6
  polynomial_degree = 2

  assignment <- sample(1:3, size = nrow(df), prob=c(0.70, 0.15, 0.15), replace = TRUE)
E)
  df_train <- df[assignment == 1, ]
  df_valid <- df[assignment == 2, ]
  df_test <- df[assignment == 3, ]

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

  #####
  ##### Boosting with Splines #####
  #####
  mean_vyp = mean(YP)
  t=1
  while(mean_vyp > 0.0001){
    t=t+1
    # Fit linear spline
    fit = rpart(yr ~ bs(x,
                        degree=polynomial_degree,
                        df=number_of_knots_split),data=df_train)

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

    # Update residuals
    df_train$yr=df_train$y - v*yp

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

    # Store fitted model in list
    list_of_weak_learners[[t]] = fit
    mean_vyp = mean(v*yp)
  }
  print(t)
  #####
  ##### 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_train = df_train %>% bind_cols(yp=yp_i)
}

# Re-arrange sequences to get pseudo residuals
plot_wl = df_train %>% 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 == (t-1))

# Plot progression of learner
plot1 <- 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_train)+ # true values
  theme_minimal()
print(plot1)

#####
##### Predicting on new data, test, validation set #####
#####

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

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

  if(i==t){
    df_valid = df_valid %>% bind_cols(yp=pred)
  }
}
```

```

}

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

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

  if(i==t){
    df_test = df_test %>% bind_cols(yp=pred)
  }
}

new_data = tibble(x = sample(seq(0,4*3,0.001),size = 100,replace = T))

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

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

  if(i==t){
    new_data = new_data %>% bind_cols(yp=pred)
  }
}

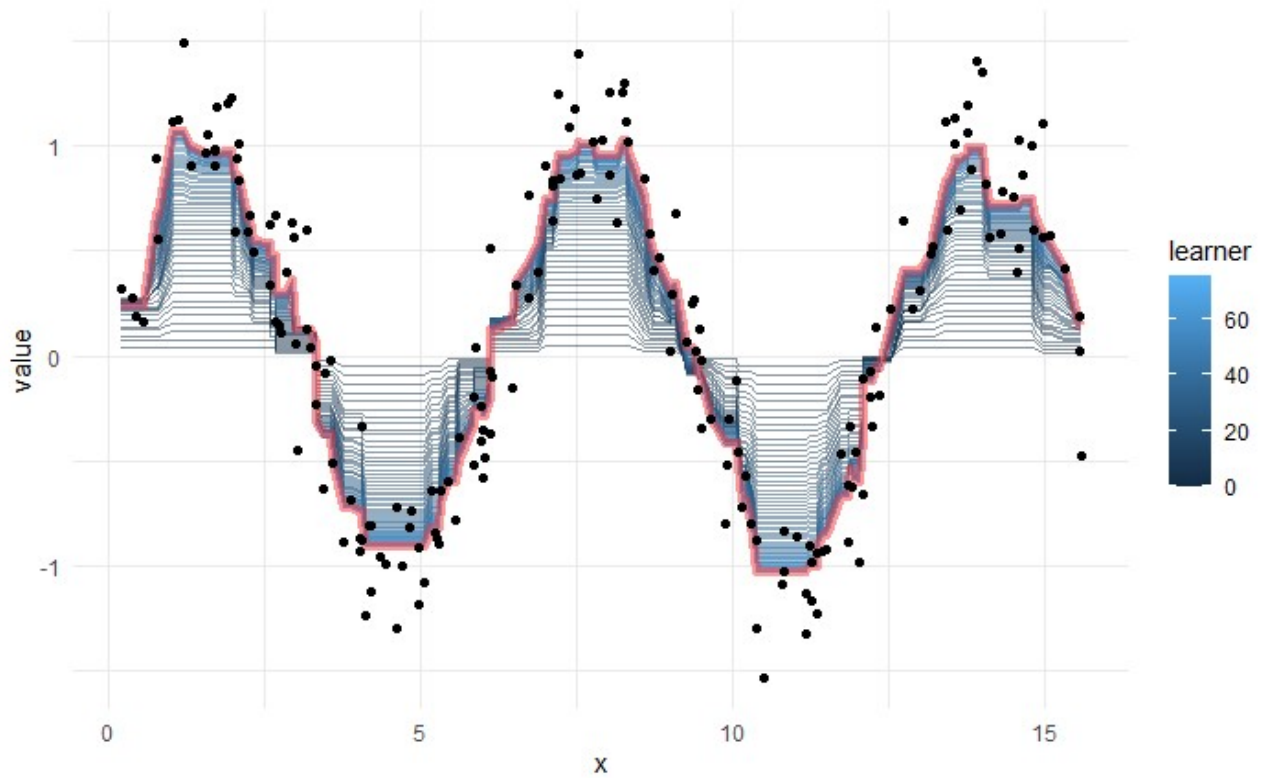
print("train RMSE:")
print(RMSE(df_train$yp, df_train$y))
print("validation RMSE:")
print(RMSE(df_valid$yp, df_valid$y))
print("test RMSE:")
print(RMSE(df_test$yp, df_test$y))
#####
##### Visualizing boosted vs predicted models #####
#####
plot2 <- ggplot(aes(x=x, y=y),data = tibble(x = df$x, y = df$y))+
  xlab('')+ylab('')+
  geom_point()+
  # Final learner from training data
  geom_line(aes(x = x, y = value, group = learner, color =learner), data = final_learner , color = 'firebrick1',size = 2) +
  # True value
  geom_line(aes(x=x,y=y),data = tibble(x = u,y = sin(u)), color='black',linetype = 'dashed')+ # true values
  # Prediction on new data
  geom_line(aes(x=x,y=yp),data = new_data, color='blue',size = 2,alpha = 0.5)+ # predicted (test) values
  geom_line(aes(x=x,y=yp),data = df_valid, color='green',size = 2,alpha = 0.5)+ # validation set
  theme_minimal()

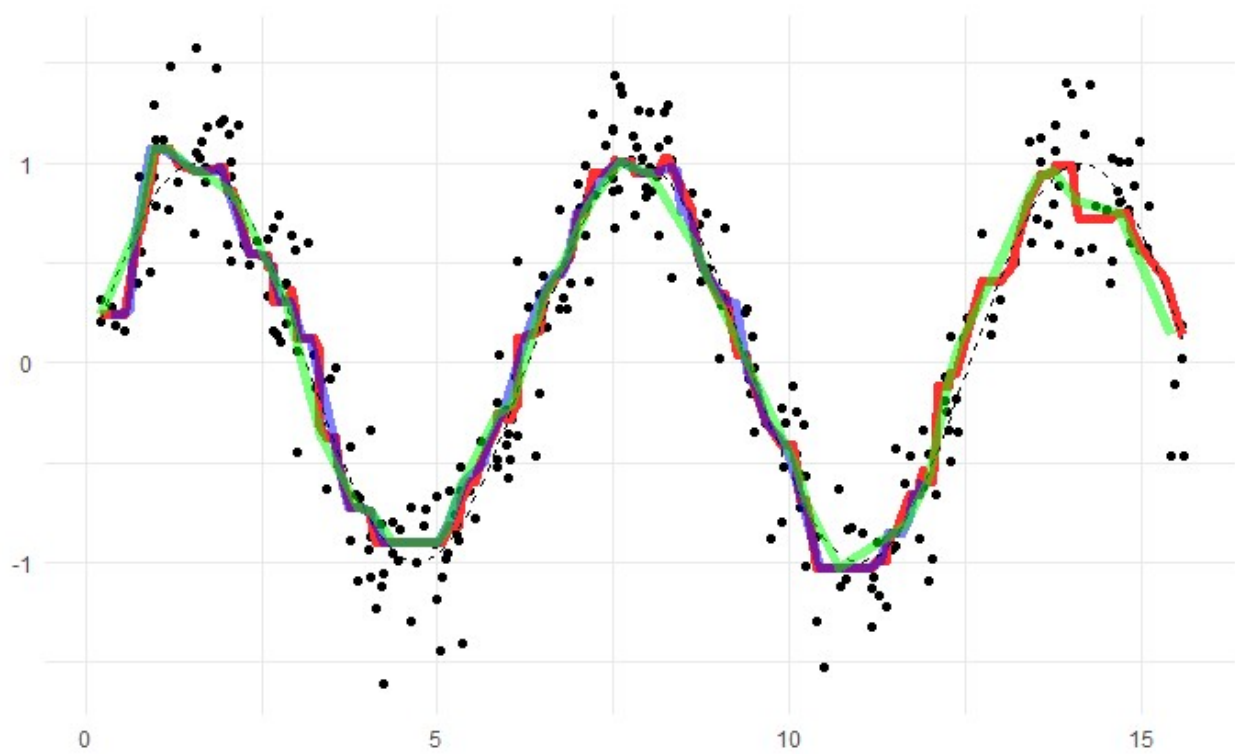
```



```
print(plot2)
}  
runboost(0.05)
```

```
[1] 76  
[1] "train RMSE:"  
[1] 0.7362656  
[1] "validation RMSE:"  
[1] 0.3094331  
[1] "test RMSE:"  
[1] 0.2989204
```





Q3

Hide

```

v=0.05
out <- data.frame(0,0,0,0,0,0)
colnames(out) <- c("mins", "cp", "maxd", "train", "test", "valid")

mins <- 10:30
cp <- seq(0.01, 0.1, 0.01)
maxd <- 20:40

for (m in mins){
  for (c in cp){
    for (md in maxd){
      row=1
      number_of_knots_split = 6
      polynomial_degree = 2

      assignment <- sample(1:3, size = nrow(df), prob=c(0.70, 0.15, 0.15), replace = TRUE)
E)

      df_train <- df[assignment == 1, ]
      df_valid <- df[assignment == 2, ]
      df_test <- df[assignment == 3, ]

      # Fit round 1
      fit=rpart(y~bs(x,degree=2,df=6),data=df_train, control = list(
        minsplit = m, minbucket = round(m/3), cp = c, maxdepth = md))
      yp = predict(fit,newdata=df_train)
      df_train$yr = df_train$y - v*yp
      YP = v*yp
      list_of_weak_learners = list(fit)

      #####
      ##### Boosting with Splines #####
      #####
      mean_vyp = mean(YP)
      t=1
      while(mean_vyp > 0.0001){
        t=t+1
        # Fit linear spline
        fit = rpart(yr ~ bs(x,
                           degree=polynomial_degree,
                           df=number_of_knots_split),data=df_train)

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

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

```

```

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

# Store fitted model in list
list_of_weak_learners[[t]] = fit
mean_vyp = mean(v*yp)
}
print(t)
#####
##### 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_train = df_train %>% bind_cols(yp=yp_i)
}

#####
##### Predicting on new data, test, validation set #####
#####

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

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

  if(i==t){
    df_valid = df_valid %>% bind_cols(yp=pred)
  }
}

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

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

  if(i==t){
    df_test = df_test %>% bind_cols(yp=pred)
  }
}

```

```

    }
  }

  new_data = tibble(x = sample(seq(0,4*3,0.001),size = 100,replace = T))

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

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

    if(i==t){
      new_data = new_data %>% bind_cols(yp=pred)
    }
  }

  out[row,] <- c(m, c, md, RMSE(df_train$yp, df_train$y), RMSE(df_test$yp, df_test
$y), RMSE(df_valid$yp, df_valid$y))
  row=row+1
}
}
}

```

## Part 2

### Question 2 - TSNE

#### Part 1

A

In most respects, the distance between points in tSNE does not matter, but in a few cases it can be meaningful. The distance is a low dimensional (2D) representation of points that initially exist in a higher dimensional space, and does reflect a transformation of these points. However, tSNE has a tendency to expand clusters that are initially dense, while contracting clusters that are initially expansive. As a result, distances between points in a cluster are meaningless, because they reflect a notion of regional and global distances depending on the level of perplexity applied. Distances between clusters tend to be more meaningful at higher levels of perplexity, because high levels of perplexity better preserve global distances between clusters.

B

Perplexity is a value that usually ranges between 5 and 50 and refers to the balance between local and global differences in the data, somewhat like setting the number of nearest neighbors we can expect each point to have. Low levels of perplexity emphasize local variations, while high levels of perplexity emphasize differences between larger clusters of data. The paper recommends examining plots of a few different perplexity values for a given set of data, and making sure that the perplexity value is smaller than the number of points, otherwise we risk unexpected behavior from the algorithm.

## C

The number of steps is important in ensuring that the representation is stable. The paper states that there is no one number of steps that lead to stability, but that there are some warning signs that the representation is unstable. For one, if you see pointlike or pinched shapes as the clusters, it is possible that not enough steps have been run to achieve convergence.

## D

In order to uncover topological information in an embedding like tSNE, we may need to make multiple plots at different perplexity levels. This is because depending on the number of points and the density of local and global clusters, a different perplexity value may be needed to uncover patterns in the original data. Producing only one graph at a specific perplexity may lead us to make false conclusions about the nature of the higher dimensional data, particularly when we are unable to graph it, or know exactly how it inhabits its higher dimensional space.

## Part 2

Hide

```
library(tidyverse)
library(Rtsne)
```

```
package 'Rtsne' was built under R version 3.5.3
```

Hide

```
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.
```

```
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    1 MB
|==
| 1%    2 MB
|==
| 1%    2 MB
|==
| 1%    2 MB
|==
| 1%    2 MB
|==
| 2%    2 MB
|==
| 2%    2 MB
|==
| 2%    2 MB
|==
| 2%    2 MB
|==
| 2%    2 MB
|==
| 2%    2 MB
|==
```

[illegible]



[illegible]

[illegible]







[illegible]



[illegible]





[illegible]



[illegible]



[illegible]

[illegible]

[illegible]





















[illegible]

[illegible]

[illegible]

16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
16%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	17 MB
=====	
17%	18 MB
=====	
17%	18 MB

[illegible]

```
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 17% 18 MB
|=====
| 18% 18 MB
|=====
| 18% 18 MB
|=====
| 18% 18 MB
|=====
| 18% 18 MB
|=====
| 18% 18 MB
|=====
| 18% 18 MB
|=====
| 18% 18 MB
|=====
| 18% 18 MB
|=====
| 18% 19 MB
|=====
| 18% 19 MB
|=====
| 18% 19 MB
|=====
| 18% 19 MB
|=====
| 18% 19 MB
|=====
| 18% 19 MB
|=====
```

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	25 MB
	24%	26 MB
	24%	26 MB
	24%	26 MB
	24%	26 MB
	24%	26 MB
	25%	26 MB
	25%	26 MB
	25%	26 MB
	25%	26 MB

[illegible]

```
|=====
| 25%    26 MB
|=====
| 25%    26 MB
|=====
| 25%    26 MB
|=====
| 25%    26 MB
|=====
| 25%    26 MB
|=====
| 25%    26 MB
|=====
| 25%    26 MB
|=====
| 25%    26 MB
|=====
| 25%    26 MB
|=====
| 25%    27 MB
|=====
| 25%    27 MB
|=====
| 25%    27 MB
|=====
| 25%    27 MB
|=====
| 25%    27 MB
|=====
| 25%    27 MB
|=====
| 25%    27 MB
|=====
| 26%    27 MB
|=====
| 26%    27 MB
|=====
| 26%    27 MB
|=====
| 26%    27 MB
|=====
| 26%    27 MB
|=====
```

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]



[illegible]



[illegible]

[illegible]





[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

	34%	36 MB
	34%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	36 MB
	35%	37 MB
	35%	37 MB
	35%	37 MB

[illegible]



[illegible]

	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	36%	38 MB
=====		
	37%	38 MB
=====		
	37%	38 MB

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]



[illegible]





[illegible]







```
=====
| 49%    51 MB
|=====
| 49%    51 MB
|=====
| 49%    51 MB
|=====
| 49%    51 MB
|=====
| 49%    51 MB
|=====
| 49%    51 MB
|=====
| 49%    51 MB
|=====
| 49%    51 MB
|=====
| 49%    51 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 49%    52 MB
|=====
| 50%    52 MB
|=====
| 50%    52 MB
|=====
```



[illegible]

[illegible]

[illegible]

[illegible]



[illegible]



[illegible]



[illegible]

[illegible]

[illegible]

[illegible]



[illegible]



[illegible]

[illegible]

[illegible]



[illegible]







[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

		67%	70 MB
=====			
==		67%	70 MB
=====			
==		67%	70 MB
=====			
==		67%	70 MB
=====			
==		67%	70 MB
=====			
==		67%	70 MB
=====			
==		67%	70 MB
=====			
==		67%	70 MB
=====			
==		67%	70 MB
=====			
==		67%	71 MB
=====			
==		67%	71 MB
=====			
==		67%	71 MB
=====			
==		68%	71 MB
=====			
==		68%	71 MB
=====			
==		68%	71 MB
=====			
==		68%	71 MB
=====			
==		68%	71 MB
=====			
==		68%	71 MB
=====			
===		68%	71 MB
=====			
===		68%	71 MB

[illegible]

[illegible]







=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 73 MB
=====		
=====		70% 74 MB
=====		



[illegible]





[illegible]



[illegible]

[illegible]



[illegible]











[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

```
|=====
=====
| 83% 87 MB
|=====
=====
| 83% 87 MB
|=====
=====
| 83% 87 MB
|=====
=====
| 83% 87 MB
|=====
=====
| 83% 87 MB
|=====
=====
| 83% 87 MB
|=====
=====
| 84% 87 MB
|=====
=====
| 84% 87 MB
|=====
=====
| 84% 87 MB
|=====
=====
| 84% 87 MB
|=====
=====
| 84% 87 MB
|=====
=====
| 84% 87 MB
|=====
=====
| 84% 87 MB
|=====
=====
| 84% 88 MB
|=====
=====
| 84% 88 MB
|=====
=====
| 84% 88 MB
|=====
=====
| 84% 88 MB
|=====
=====
| 84% 88 MB
|=====
=====
| 84% 88 MB
|=====
```

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	94 MB
		90%	95 MB
		90%	95 MB
		90%	95 MB
		90%	95 MB
		90%	95 MB
		91%	95 MB
		91%	95 MB
		91%	95 MB
		91%	95 MB
		91%	95 MB

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

```
=====| 99% 104 MB
|=====
=====| 99% 104 MB
|=====
=====| 100% 104 MB
|=====
=====| 100% 104 MB
```

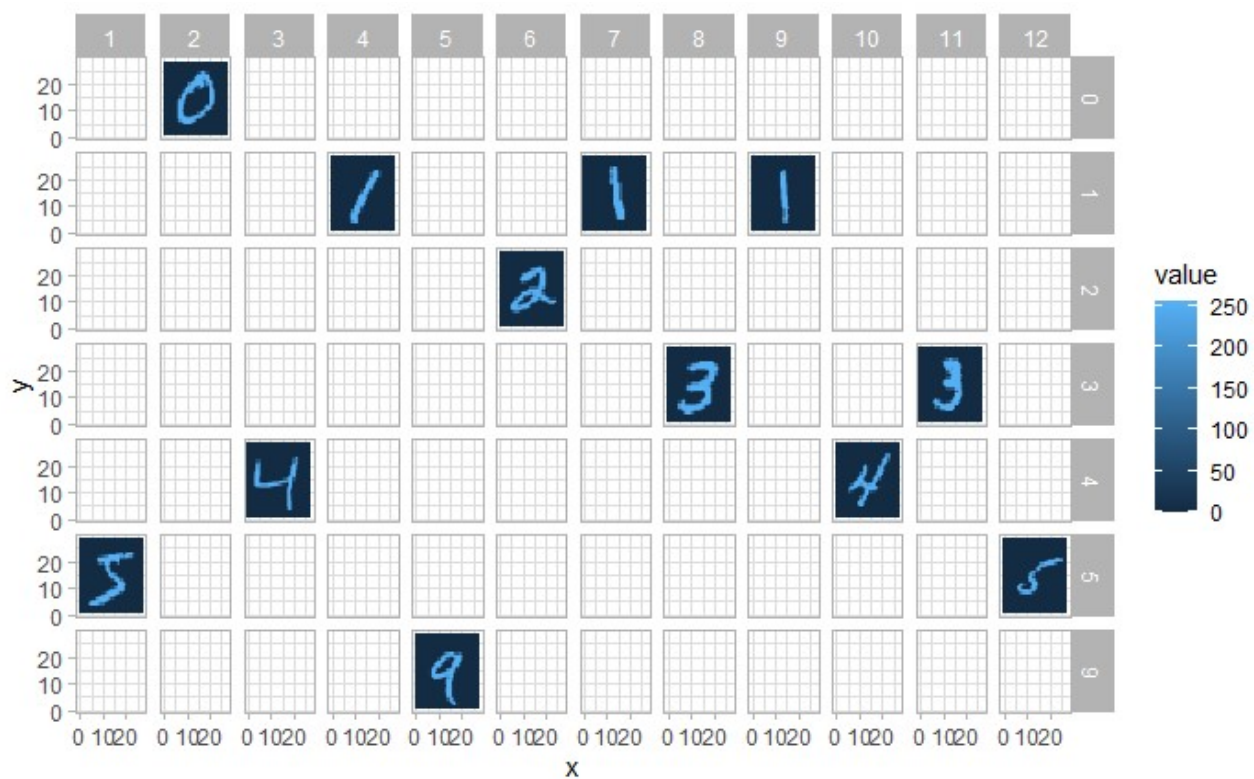
Hide

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

```
[1] 60000 785
```

Hide

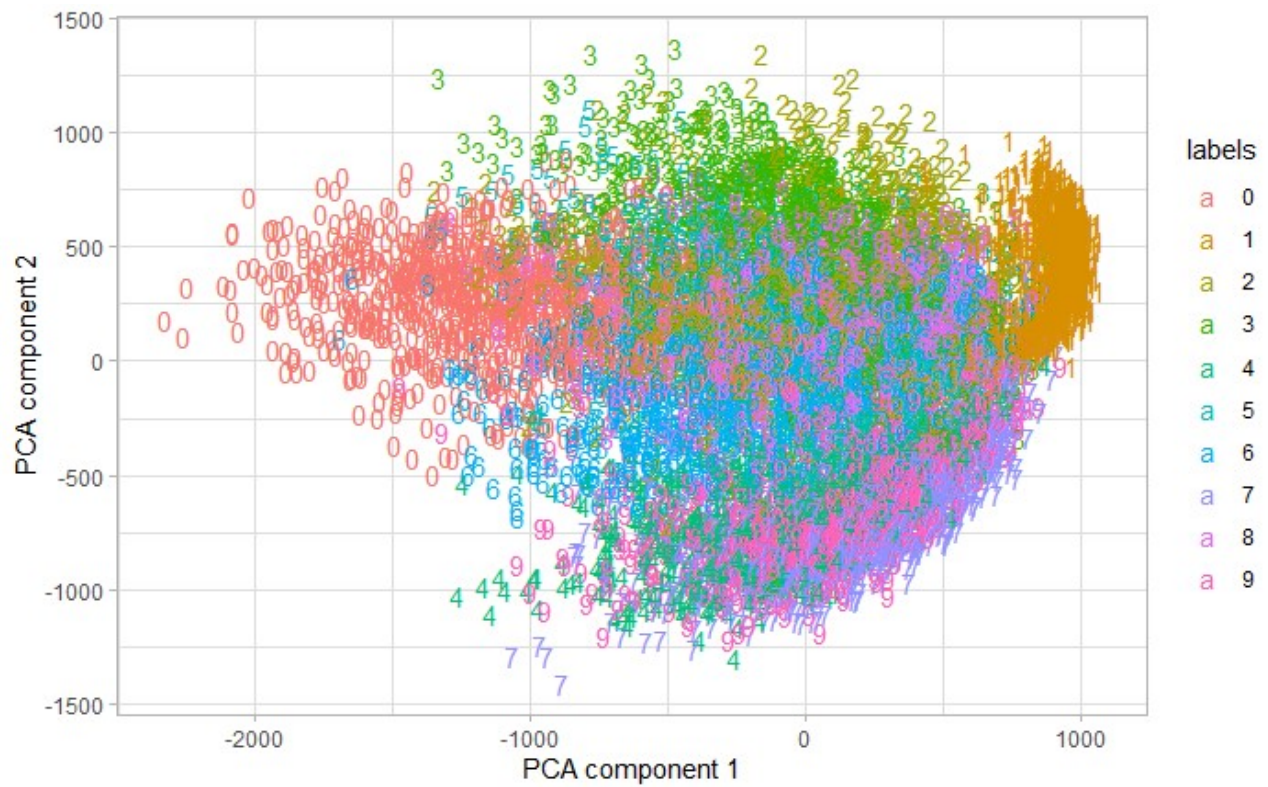
```
# Rearranging the data
pixels_gathered <- mnist_raw %>% head(10000) %>%
  rename(label = X1) %>%
  mutate(instance = row_number()) %>%
  gather(pixel, value, -label, -instance) %>%
  extract(pixel, "pixel", "(\\d+)", convert = TRUE) %>%
  mutate(pixel = pixel - 2,
         x = pixel %% 28,
         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 )
```



## Part A

Hide

```
#####
##### 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')
```



## Part B

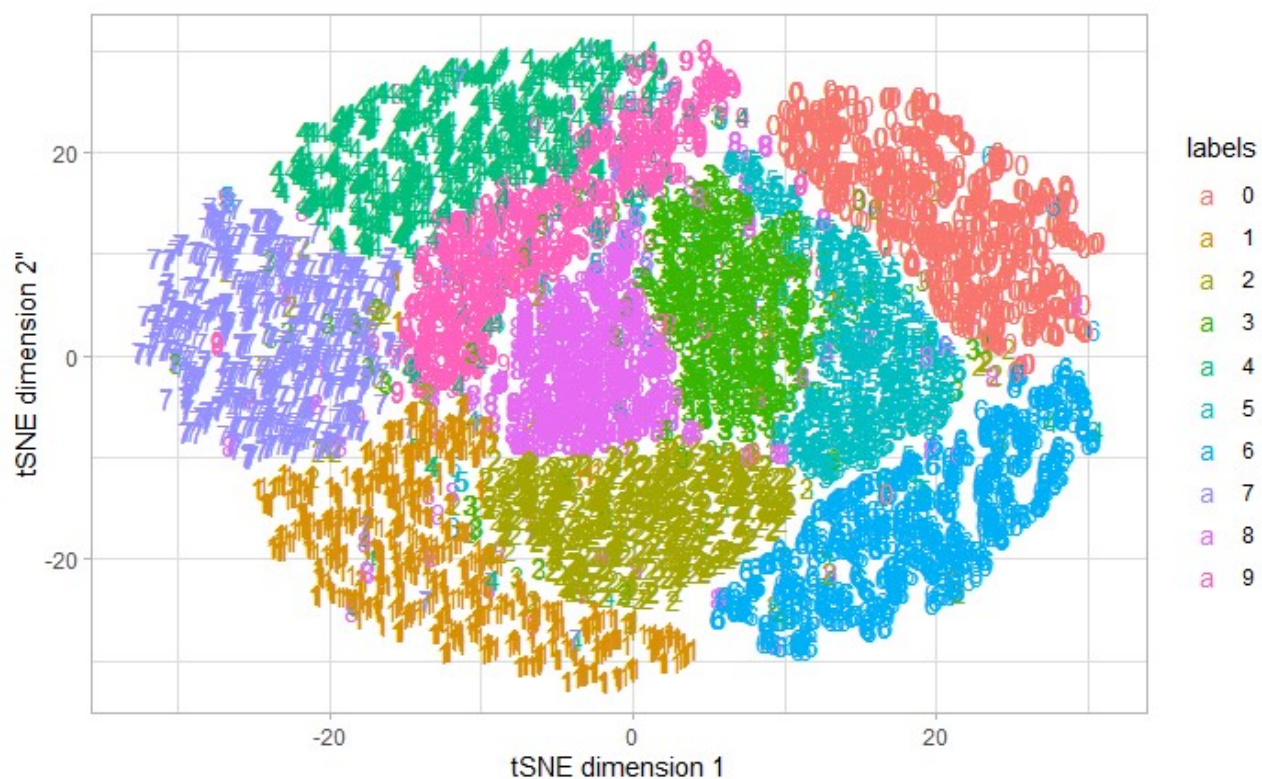
Hide

```
#####  
#####    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 11.23 seconds (sparsity = 0.002104)!
Learning embedding...
Iteration 50: error is 118.296638 (50 iterations in 2.17 seconds)
Iteration 100: error is 106.538721 (50 iterations in 2.14 seconds)
Iteration 150: error is 99.015842 (50 iterations in 1.74 seconds)
Iteration 200: error is 96.230348 (50 iterations in 1.80 seconds)
Iteration 250: error is 94.735976 (50 iterations in 1.72 seconds)
Iteration 300: error is 4.370408 (50 iterations in 1.76 seconds)
Iteration 350: error is 3.810689 (50 iterations in 1.76 seconds)
Iteration 400: error is 3.443766 (50 iterations in 1.70 seconds)
Iteration 450: error is 3.177969 (50 iterations in 1.80 seconds)
Iteration 500: error is 2.973647 (50 iterations in 1.87 seconds)
Fitting performed in 18.45 seconds.
```

Hide

```
# 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'')
```



## Part C

Hide

```
perps <- c(5, 20, 60, 100, 125, 160)
for (i in perps){
  embedding = Rtsne(X = first_10k_samples, dims = 2,
                    perplexity = i,
                    theta = 0.5,
                    eta = 200,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
  # Visualizing TSNE output
  embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                          labels = as.character(first_10k_samples_labels))
  plotvar <- ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_p
lot) +
    geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2')
  print(plotvar)
}
```



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 10.99 seconds (sparsity = 0.002104)!

Learning embedding...

Iteration 50: error is 118.296620 (50 iterations in 2.20 seconds)

Iteration 100: error is 106.435698 (50 iterations in 2.65 seconds)

Iteration 150: error is 98.874747 (50 iterations in 2.01 seconds)

Iteration 200: error is 96.131810 (50 iterations in 1.96 seconds)

Iteration 250: error is 94.603758 (50 iterations in 1.99 seconds)

Iteration 300: error is 4.357661 (50 iterations in 1.71 seconds)

Iteration 350: error is 3.801872 (50 iterations in 1.77 seconds)

Iteration 400: error is 3.437645 (50 iterations in 1.74 seconds)

Iteration 450: error is 3.172457 (50 iterations in 1.72 seconds)

Iteration 500: error is 2.967835 (50 iterations in 1.78 seconds)

Fitting performed in 19.53 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 13.71 seconds (sparsity = 0.008217)!

Learning embedding...

Iteration 50: error is 102.343944 (50 iterations in 2.25 seconds)

Iteration 100: error is 93.048610 (50 iterations in 2.27 seconds)

Iteration 150: error is 88.722638 (50 iterations in 1.97 seconds)

Iteration 200: error is 87.827237 (50 iterations in 1.86 seconds)

Iteration 250: error is 87.587934 (50 iterations in 1.89 seconds)

Iteration 300: error is 3.399457 (50 iterations in 1.86 seconds)

Iteration 350: error is 2.949292 (50 iterations in 1.78 seconds)

Iteration 400: error is 2.698757 (50 iterations in 1.79 seconds)

Iteration 450: error is 2.530392 (50 iterations in 1.84 seconds)

Iteration 500: error is 2.407511 (50 iterations in 1.86 seconds)

Fitting performed in 19.38 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 16.59 seconds (sparsity = 0.024328)!

```
Learning embedding...
Iteration 50: error is 89.428527 (50 iterations in 2.27 seconds)
Iteration 100: error is 84.939968 (50 iterations in 2.39 seconds)
Iteration 150: error is 81.459923 (50 iterations in 2.27 seconds)
Iteration 200: error is 81.451576 (50 iterations in 2.20 seconds)
Iteration 250: error is 81.451097 (50 iterations in 2.18 seconds)
Iteration 300: error is 2.603989 (50 iterations in 2.16 seconds)
Iteration 350: error is 2.266014 (50 iterations in 2.12 seconds)
Iteration 400: error is 2.093328 (50 iterations in 2.11 seconds)
Iteration 450: error is 1.985335 (50 iterations in 2.06 seconds)
Iteration 500: error is 1.909953 (50 iterations in 2.09 seconds)
Fitting performed in 21.85 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 20.81 seconds (sparsity = 0.040571)!
Learning embedding...
Iteration 50: error is 83.371128 (50 iterations in 2.69 seconds)
Iteration 100: error is 82.085545 (50 iterations in 2.79 seconds)
Iteration 150: error is 78.379137 (50 iterations in 2.59 seconds)
Iteration 200: error is 78.297408 (50 iterations in 2.65 seconds)
Iteration 250: error is 78.269857 (50 iterations in 2.68 seconds)
Iteration 300: error is 2.267728 (50 iterations in 2.59 seconds)
Iteration 350: error is 1.969324 (50 iterations in 2.55 seconds)
Iteration 400: error is 1.823938 (50 iterations in 2.50 seconds)
Iteration 450: error is 1.737029 (50 iterations in 2.54 seconds)
Iteration 500: error is 1.678444 (50 iterations in 2.51 seconds)
Fitting performed in 26.09 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 23.23 seconds (sparsity = 0.050806)!
Learning embedding...
```

It looks like a perplexity of 5 was perhaps the best at separating the various numbers effectively. Higher perplexity iterations found more high-level differences within the number sets - for example at perplexity 100 a group of twos split off from the main cluster of twos and moves between the ones and fours, which might be because those particular twos look somewhat different to the rest of the cluster. Also, with increased perplexity there is more mingling between the four and seven clusters, with the sevens appearing to split the four cluster in half. In any case, at all perplexities most of the numbers

appear to be grouped together logically. Since perplexity is somewhat similar to nearest neighbors it makes intuitive sense to me that 5 was the most effective value of those tested above because it is closest to the number of actual values present in the data - 10.

## Part D

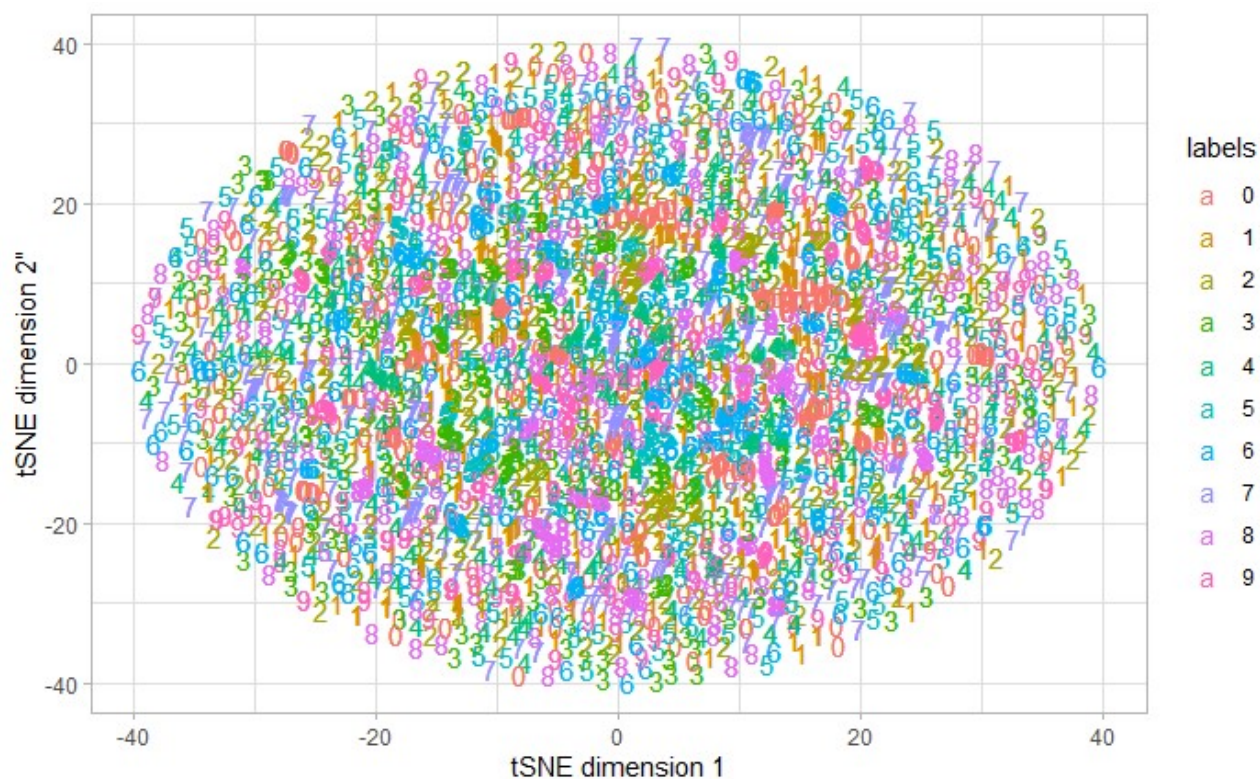
Hide

```
#####
####      Running the TSNE emebdding      ####
#####
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 12.00 seconds (sparsity = 0.000442)!
Learning embedding...
Iteration 50: error is 135.156715 (50 iterations in 2.70 seconds)
Iteration 100: error is 116.635701 (50 iterations in 3.18 seconds)
Iteration 150: error is 108.512685 (50 iterations in 3.04 seconds)
Iteration 200: error is 104.024420 (50 iterations in 3.31 seconds)
Iteration 250: error is 100.908398 (50 iterations in 3.14 seconds)
Iteration 300: error is 5.266115 (50 iterations in 2.58 seconds)
Iteration 350: error is 4.625565 (50 iterations in 3.12 seconds)
Iteration 400: error is 4.128942 (50 iterations in 3.62 seconds)
Iteration 450: error is 3.738648 (50 iterations in 3.49 seconds)
Iteration 500: error is 3.425794 (50 iterations in 3.63 seconds)
Fitting performed in 31.80 seconds.
```

Hide

```
# 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')
```



The distribution looks completely random with an even spread of various values across both dimensions. This is because since the perplexity is set at 1, the data has been loosely organized into one cluster with each point having about one close neighbor. This likely led to there being extreme emphasis on local variation to the degree that there are only a few groups of a few numbers slightly clustered together in the center of the circle.

#### Part E

I think it is likely that when the perplexity is set to 5000 there would be a plot somewhat similar to the above because the perplexity would be at half the number of points in the dataset. With extremely high emphasis on global variations between the data, there would be much more diffusion across the labels since there would effectively be about 5000 “close neighbors” to each point.

#### Part F

Hide

```
perps <- c(5, 20, 60, 100, 125, 160)
itercosts <- numeric(6)
n <- 1
for (i in perps){
  embedding = Rtsne(X = first_10k_samples, dims = 2,
                    perplexity = i,
                    theta = 0.5,
                    eta = 200,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
  itercosts[n] <- embedding$itercosts[length(embedding$itercosts)]

  n <- n+1
}
```

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 12.98 seconds (sparsity = 0.002104)!

Learning embedding...

Iteration 50: error is 118.296639 (50 iterations in 2.14 seconds)

Iteration 100: error is 106.425980 (50 iterations in 2.05 seconds)

Iteration 150: error is 99.052204 (50 iterations in 1.64 seconds)

Iteration 200: error is 96.248845 (50 iterations in 1.61 seconds)

Iteration 250: error is 94.683380 (50 iterations in 1.67 seconds)

Iteration 300: error is 4.364248 (50 iterations in 1.62 seconds)

Iteration 350: error is 3.804556 (50 iterations in 1.56 seconds)

Iteration 400: error is 3.438433 (50 iterations in 1.66 seconds)

Iteration 450: error is 3.171820 (50 iterations in 1.69 seconds)

Iteration 500: error is 2.967393 (50 iterations in 1.74 seconds)

Fitting performed in 17.38 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 12.87 seconds (sparsity = 0.008217)!

Learning embedding...

Iteration 50: error is 102.343946 (50 iterations in 2.13 seconds)

Iteration 100: error is 93.714221 (50 iterations in 2.94 seconds)

Iteration 150: error is 88.534339 (50 iterations in 2.25 seconds)

Iteration 200: error is 87.742914 (50 iterations in 2.25 seconds)

Iteration 250: error is 87.547134 (50 iterations in 2.26 seconds)

Iteration 300: error is 3.432634 (50 iterations in 1.99 seconds)

Iteration 350: error is 2.970269 (50 iterations in 1.72 seconds)

Iteration 400: error is 2.712982 (50 iterations in 1.74 seconds)

Iteration 450: error is 2.539589 (50 iterations in 1.81 seconds)

Iteration 500: error is 2.415545 (50 iterations in 1.86 seconds)

Fitting performed in 20.93 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 17.06 seconds (sparsity = 0.024328)!

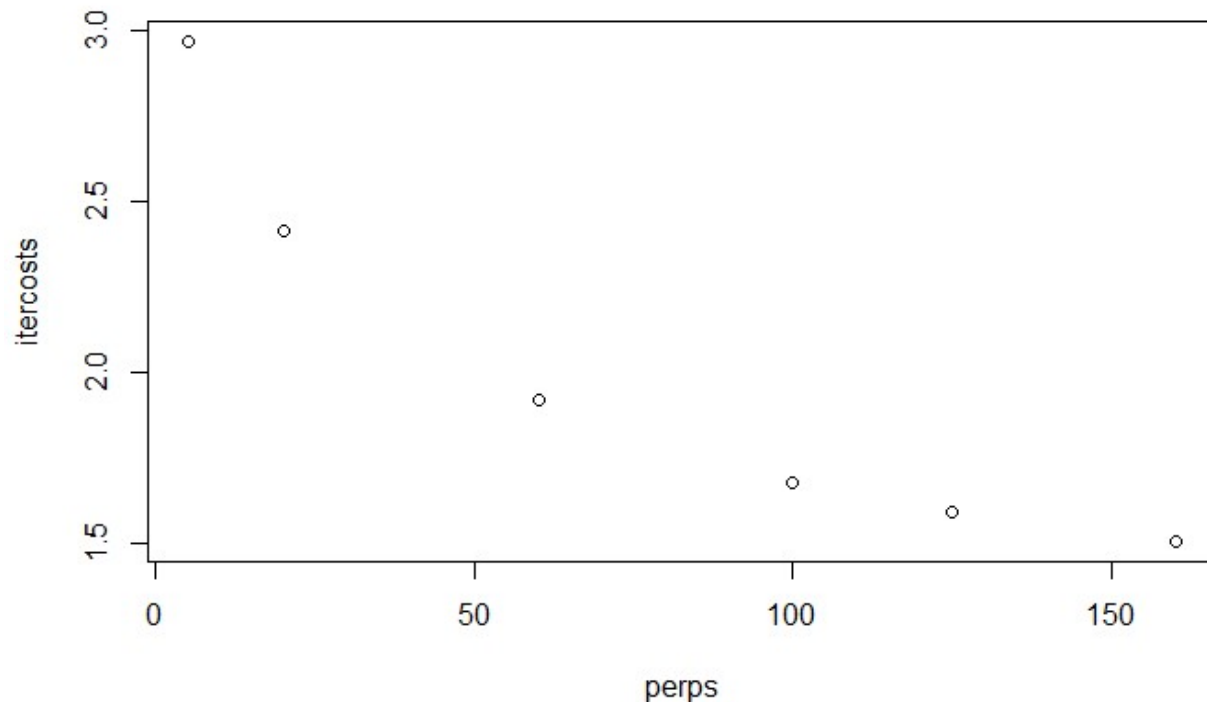
```
Learning embedding...
Iteration 50: error is 89.428527 (50 iterations in 2.47 seconds)
Iteration 100: error is 83.883239 (50 iterations in 2.39 seconds)
Iteration 150: error is 81.705015 (50 iterations in 2.46 seconds)
Iteration 200: error is 81.658544 (50 iterations in 2.45 seconds)
Iteration 250: error is 81.654334 (50 iterations in 2.47 seconds)
Iteration 300: error is 2.641383 (50 iterations in 2.25 seconds)
Iteration 350: error is 2.287528 (50 iterations in 2.21 seconds)
Iteration 400: error is 2.109440 (50 iterations in 2.22 seconds)
Iteration 450: error is 1.997344 (50 iterations in 2.25 seconds)
Iteration 500: error is 1.919300 (50 iterations in 2.24 seconds)
Fitting performed in 23.41 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 20.85 seconds (sparsity = 0.040571)!
Learning embedding...
Iteration 50: error is 83.371128 (50 iterations in 3.14 seconds)
Iteration 100: error is 80.645713 (50 iterations in 4.78 seconds)
Iteration 150: error is 78.378166 (50 iterations in 3.73 seconds)
Iteration 200: error is 78.310978 (50 iterations in 3.21 seconds)
Iteration 250: error is 78.286956 (50 iterations in 3.14 seconds)
Iteration 300: error is 2.265710 (50 iterations in 2.79 seconds)
Iteration 350: error is 1.969091 (50 iterations in 2.62 seconds)
Iteration 400: error is 1.824555 (50 iterations in 2.59 seconds)
Iteration 450: error is 1.737675 (50 iterations in 2.63 seconds)
Iteration 500: error is 1.678882 (50 iterations in 2.67 seconds)
Fitting performed in 31.31 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 23.01 seconds (sparsity = 0.050806)!
Learning embedding...
Iteration 50: error is 80.714217 (50 iterations in 3.77 seconds)
Iteration 100: error is 79.709691 (50 iterations in 5.43 seconds)
Iteration 150: error is 77.287527 (50 iterations in 5.01 seconds)
Iteration 200: error is 76.765205 (50 iterations in 3.58 seconds)
Iteration 250: error is 76.724266 (50 iterations in 3.20 seconds)
Iteration 300: error is 2.142039 (50 iterations in 2.89 seconds)
Iteration 350: error is 1.856637 (50 iterations in 2.85 seconds)
Iteration 400: error is 1.724251 (50 iterations in 2.82 seconds)
```

```
Iteration 450: error is 1.645046 (50 iterations in 2.89 seconds)
Iteration 500: error is 1.592167 (50 iterations in 2.87 seconds)
Fitting performed in 35.30 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 25.52 seconds (sparsity = 0.065215)!
Learning embedding...
Iteration 50: error is 77.767491 (50 iterations in 3.61 seconds)
Iteration 100: error is 77.638361 (50 iterations in 5.96 seconds)
Iteration 150: error is 75.273651 (50 iterations in 6.23 seconds)
Iteration 200: error is 74.936092 (50 iterations in 3.99 seconds)
Iteration 250: error is 74.923410 (50 iterations in 3.68 seconds)
Iteration 300: error is 2.010979 (50 iterations in 3.29 seconds)
Iteration 350: error is 1.751679 (50 iterations in 3.28 seconds)
Iteration 400: error is 1.628257 (50 iterations in 3.24 seconds)
Iteration 450: error is 1.555052 (50 iterations in 3.25 seconds)
Iteration 500: error is 1.507294 (50 iterations in 3.23 seconds)
Fitting performed in 39.76 seconds.
```

[Hide](#)

```
plot(perps, itercosts)
```





Based on this graph the optimal value appears to be a perplexity of 160, because it minimizes the iter\_costs (KL divergence) more than any other perplexity value. Intuitively based on the graphs above, I don't think this makes sense, but that could be why the article recommends plotting at multiple perplexities rather than selecting a perplexity based on a numeric output like the minimization of the KL-divergence, which I think in this case might be overfitting and focusing on more global variations in the data.

## Part G

Hide

```
etas <- c(10, 100, 200)
for (i in etas){
  embedding = Rtsne(X = first_10k_samples, dims = 2,
                    perplexity = 160,
                    theta = 0.5,
                    eta = i,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
  # Visualizing TSNE output
  embedding_plot = tibble(x = embedding$Y[,1], y = embedding$Y[,2],
                          labels = as.character(first_10k_samples_labels))
  plotvar <- ggplot(aes(x = x, y=y,label = labels, color = labels), data = embedding_plot) +
    geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2')
  print(plotvar)
}
```

```
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 26.09 seconds (sparsity = 0.065215)!
Learning embedding...
Iteration 50: error is 77.767491 (50 iterations in 3.48 seconds)
Iteration 100: error is 77.767491 (50 iterations in 3.70 seconds)
Iteration 150: error is 77.767491 (50 iterations in 4.80 seconds)
Iteration 200: error is 77.767491 (50 iterations in 6.26 seconds)
Iteration 250: error is 77.767487 (50 iterations in 8.00 seconds)
Iteration 300: error is 3.811714 (50 iterations in 9.12 seconds)
Iteration 350: error is 2.893403 (50 iterations in 6.50 seconds)
Iteration 400: error is 2.459480 (50 iterations in 4.01 seconds)
Iteration 450: error is 2.238439 (50 iterations in 3.84 seconds)
Iteration 500: error is 2.095257 (50 iterations in 3.71 seconds)
Fitting performed in 53.41 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 26.44 seconds (sparsity = 0.065215)!
Learning embedding...
Iteration 50: error is 77.767491 (50 iterations in 3.31 seconds)
Iteration 100: error is 77.767482 (50 iterations in 5.53 seconds)
Iteration 150: error is 75.842008 (50 iterations in 6.45 seconds)
Iteration 200: error is 75.277272 (50 iterations in 6.92 seconds)
Iteration 250: error is 74.942551 (50 iterations in 4.16 seconds)
Iteration 300: error is 2.080620 (50 iterations in 3.09 seconds)
Iteration 350: error is 1.827789 (50 iterations in 3.01 seconds)
Iteration 400: error is 1.692479 (50 iterations in 3.01 seconds)
Iteration 450: error is 1.613633 (50 iterations in 3.01 seconds)
Iteration 500: error is 1.558856 (50 iterations in 3.06 seconds)
Fitting performed in 41.54 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 25.88 seconds (sparsity = 0.065215)!
```

Learning embedding...

Iteration 50: error is 77.767491 (50 iterations in 3.24 seconds)

Iteration 100: error is 77.760560 (50 iterations in 4.57 seconds)

Iteration 150: error is 75.284980 (50 iterations in 4.53 seconds)

Iteration 200: error is 74.947997 (50 iterations in 3.91 seconds)

Iteration 250: error is 74.927275 (50 iterations in 3.82 seconds)

Iteration 300: error is 1.997175 (50 iterations in 3.17 seconds)

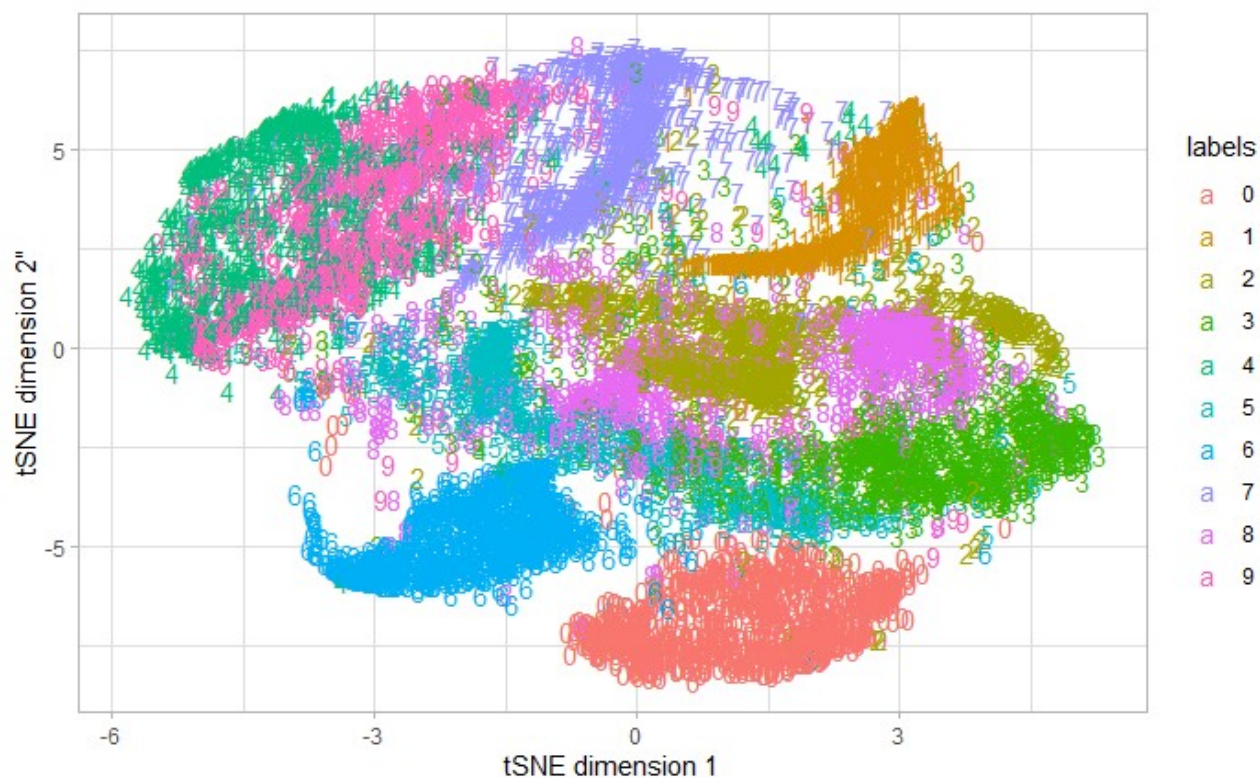
Iteration 350: error is 1.741911 (50 iterations in 3.11 seconds)

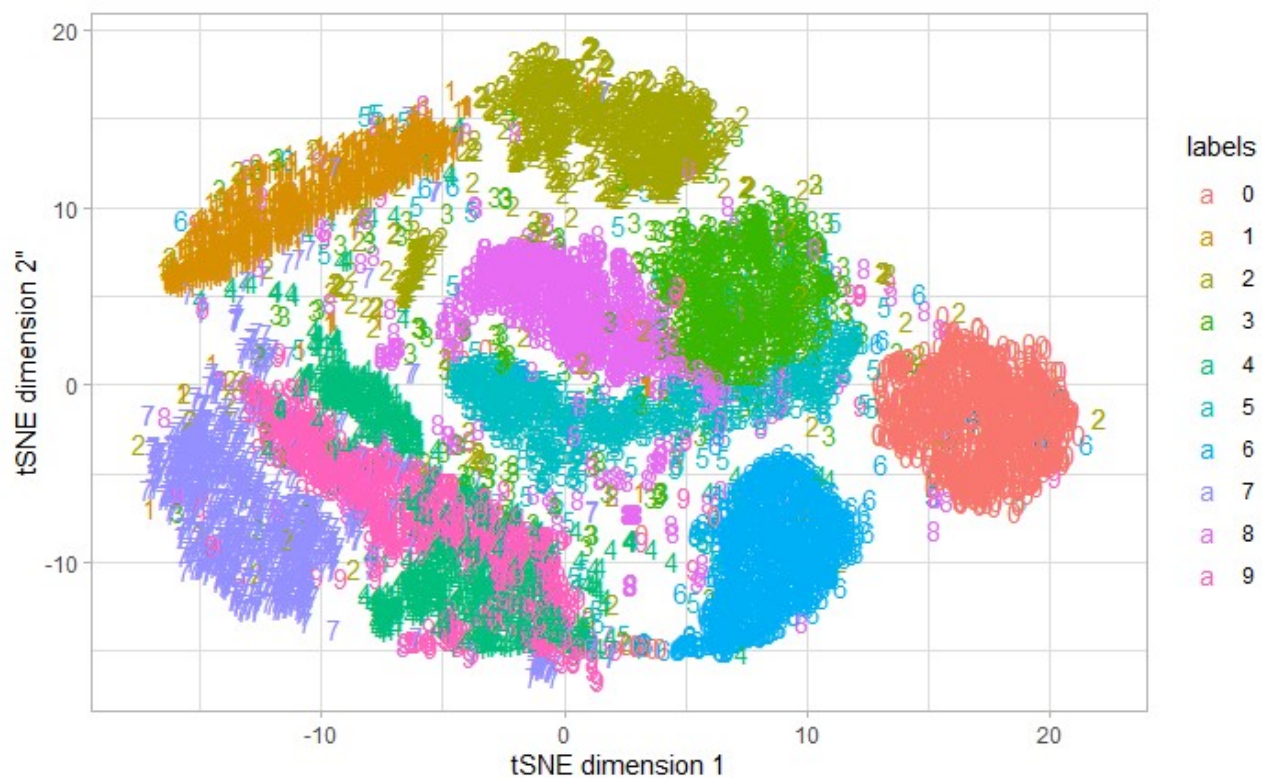
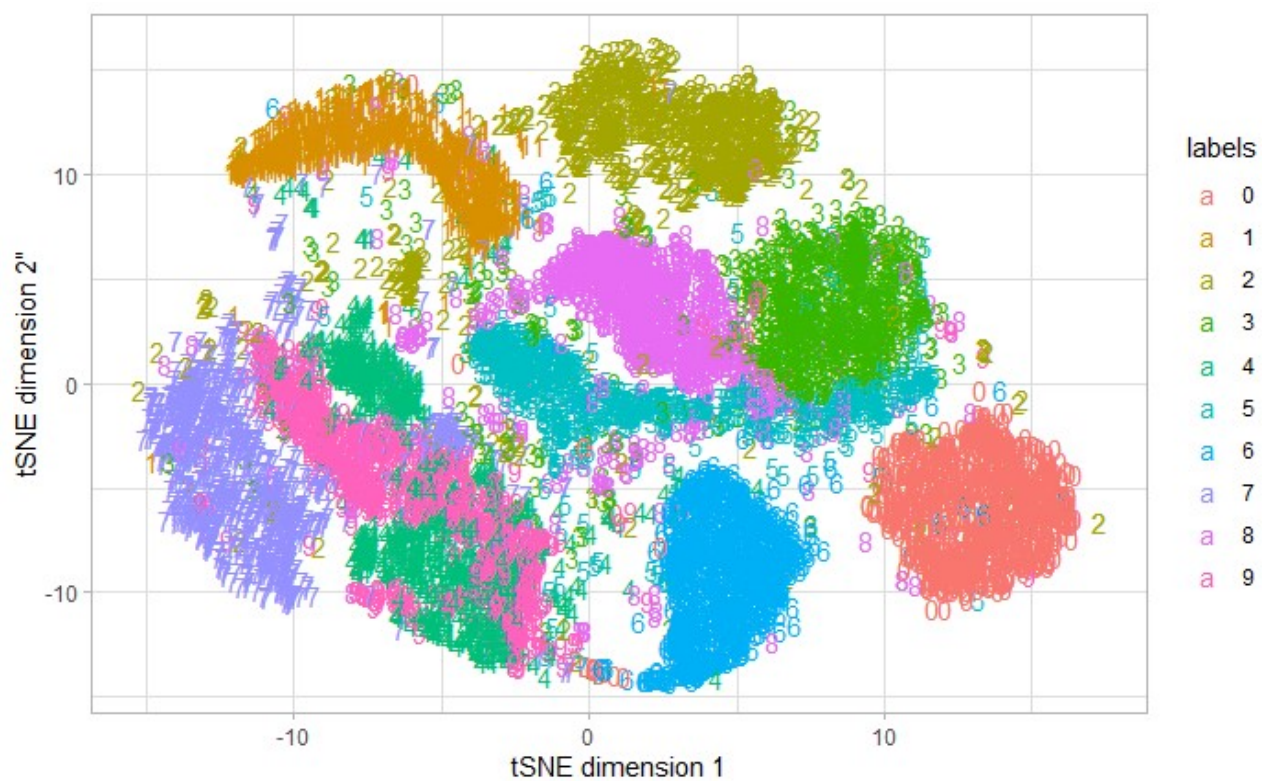
Iteration 400: error is 1.613746 (50 iterations in 3.11 seconds)

Iteration 450: error is 1.540876 (50 iterations in 3.12 seconds)

Iteration 500: error is 1.493835 (50 iterations in 3.12 seconds)

Fitting performed in 35.69 seconds.





It looks like lower learning rates result in increased bleeding over between clusters - probably because the algorithm doesn't achieve as much convergence with lower learning rates and the same number of iterations. The output also shows that the iter\_costs (KL divergence) is higher for the lower learning rates.