hw3

Norman Hong

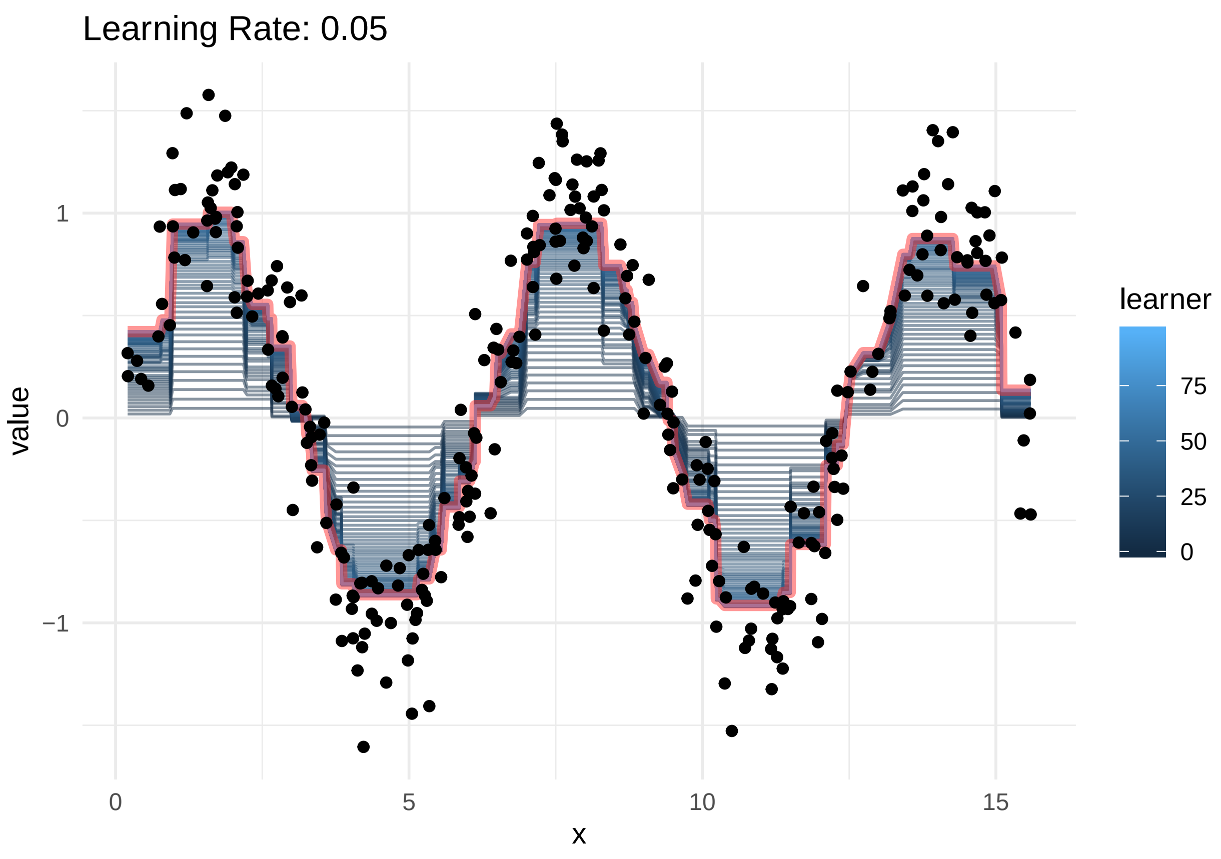
April 9, 2020

Q0.) Using rpart to build a gradient boosted tree with v=0.05 and no changes in the default rpart parameters.

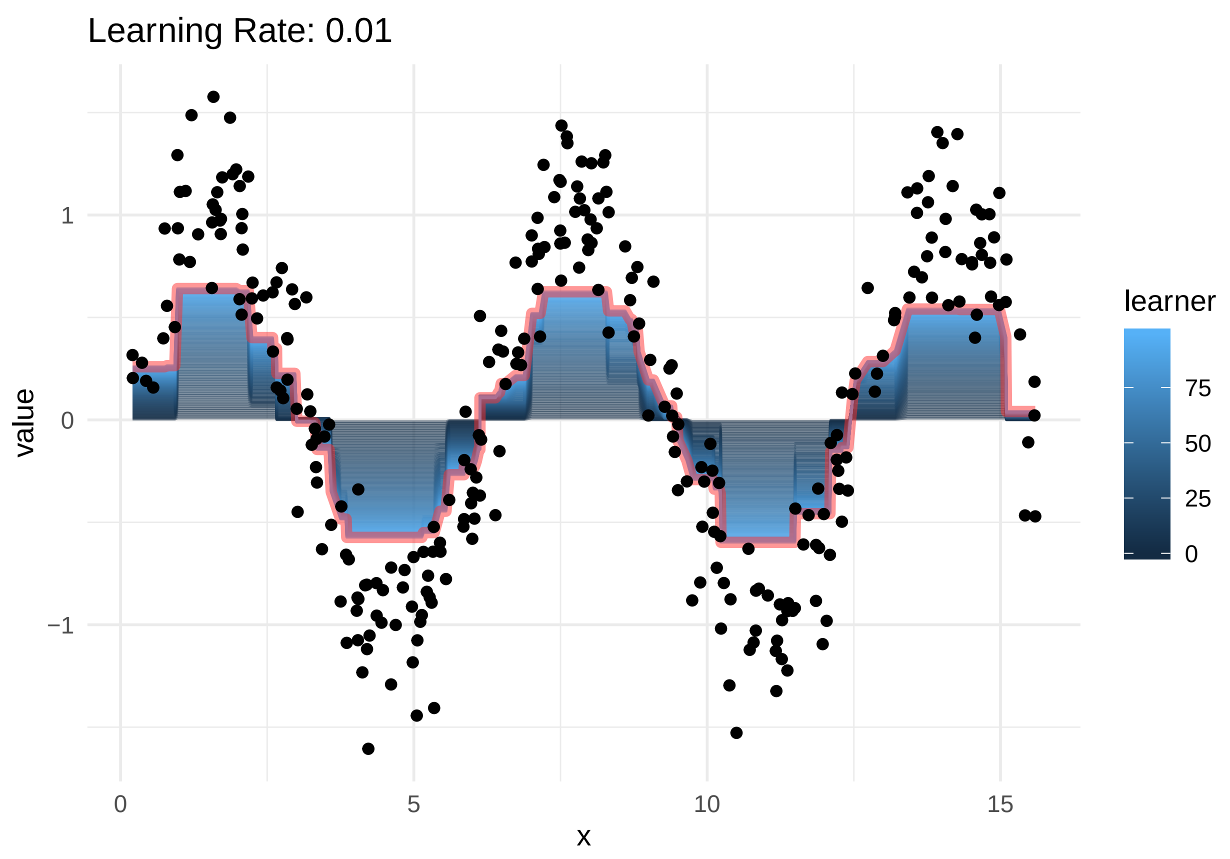
```r
# Generating sample data
n=300
set.seed(1)
u=sort(runif(n)*5*pi)
y = sin(u)+rnorm(n)/4
df = data.frame(x=u,y=y)

temp1 <- function(v, number_of_weak_learners=100){
  # v is the learning rate.
  # number_of_weak_learners is the number of trees in boosted model.

  # Fit first iteration
  fit = rpart(y ~ x, data=df)  # Regression tree.
  yp = predict(fit,newdata=df)
  df$yr = df$y - v*yp  # Update to residuals in boosting.
  YP = v*yp # This the spline prediction multiplied by a weight, known as the learning rate.
  list_of_weak_learners = list(fit)

  # Fitting rest of weak learners.
  for(t in 2:number_of_weak_learners){
    # Fit regression tree.
    fit = rpart(yr ~ x, data=df)

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

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

    # Bind to new data point; Adding v*yp to YP
    # Each column is a weak learner. Rows are the data points.
    YP = cbind(YP,v*yp)

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

boosted_model <- temp1(.05)
```

Q1.) What happens when you change v, the learning parameter? Show the fitted plots v=.01, .05, .125.

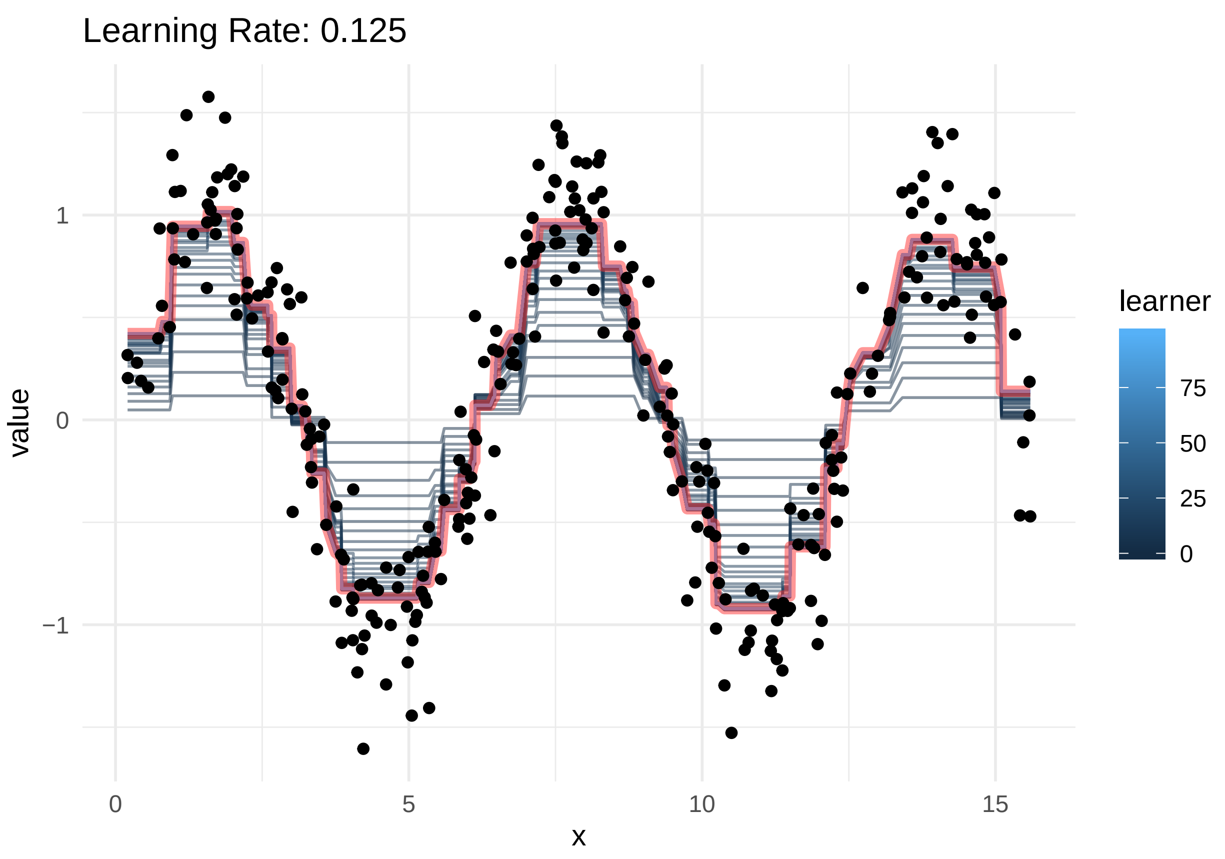
```r
temp2 <- function(v, number_of_weak_learners=100){
  # v is the learning rate.

  # Fit first iteration
  fit = rpart(y ~ x, data=df)  # Regression tree.
  yp = predict(fit,newdata=df)
  df$yr = df$y - v*yp  # Update to residuals in boosting.
  YP = v*yp # This the spline prediction multiplied by a weight, known as the learning rate.
  list_of_weak_learners = list(fit)

  # Fitting rest of weak learners.
  for(t in 2:number_of_weak_learners){
    # Fit regression tree
    fit = rpart(yr ~ x, data=df)

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

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

    # Bind to new data point; Adding v*yp to YP
    # Each column is a weak learner. Rows are the data points.
    YP = cbind(YP,v*yp)

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

# Getting predictions for each boost
# Iterate through all weak learners.
for (i in 1:number_of_weak_learners){
  # Calculating performance of first i weak_learners

  # Summing weak learner residuals
  # Summing up the predictions across all learners for
  # each data point. Final output is last i.
  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)  # TODO: not sure what the point of this is. It isn't used.
  # yp, yp1, yp2, ... column names are auto generated by r.
  df = df %>% bind_cols(yp=yp_i)  # Adding all of yp to df on columns axis.
}

# Re-arrange sequences to get pseudo residuals
plot_wl = df %>% select(-y,-yr) %>%  # remove y and yr column.
  pivot_longer(cols = starts_with("yp")) %>%  # Transpose matrix containing all "yp" in column names.
  mutate(learner = str_match(name,"[0-9]+")) %>%  # Create learner column with values based on str_matc
  mutate(learner = as.integer(ifelse(is.na(learner),0,learner))) # Turn the Na value into 0.

# final learner
final_learner = plot_wl %>% filter(learner == (number_of_weak_learners-1))  # Selecting the final learn

# 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 (data points)
  theme_minimal() +
  ggtitle(paste0("Learning Rate: ", v))

}
temp2(.05)
```



```
temp2(.01)
```



```r
temp2(.125)
```



Q2.) Using a validation and test set, develop a heuristic approach for determining when to stop training your gradient boosted tree using v=.05 and the default setting, how many trees did you include, and what is your performance on the test set for the set of selected parameters (use RMSE).

55 trees seem to be optimal with a performance of .0934 RMSE.

```
set.seed(1)
boosted_model <- temp1(.05)

temp3 <- function(new_data, list_of_weak_learners, v=.05, number_of_weak_learners = 100){
  # Doing predictions on new data.
  # number_of_weak_learners must be the same number used in temp1() and temp2().
  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)
    }
  }
  return(new_data)
}
rmse <- function(y_true, y_pred){
  # Calculates root mean squared error.
  sq <- (y_true - y_pred)^2
  out <- sqrt(mean(sq))
  return(out)
}
```

```
# Generate new data
set.seed(1)
x = sample(seq(0,4*3,0.001),size = 100,replace = T)
y = sin(x)
validation_data <- data.frame(x=x,y=y)

# validation rmse; trees = 125
boosted_model <- temp1(v=.05, number_of_weak_learners=125)
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=125)
cat('\ntrees: 120 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 120 --> RMSE:  0.0936626
```

```
# validation rmse; trees = 110
boosted_model <- temp1(v=.05, number_of_weak_learners=110)
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=110)
cat('\ntrees: 110 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 110 --> RMSE:  0.09364658
```

```
# validation rmse; trees = 100
boosted_model <- temp1(v=.05, number_of_weak_learners=100)
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=100)
cat('\ntrees: 100 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 100 --> RMSE:  0.09362737
```

```
# validation rmse; trees = 70
boosted_model <- temp1(v=.05, number_of_weak_learners=70)
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=70)
cat('\ntrees: 70 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 70 --> RMSE:  0.093493
```

```
# validation rmse; trees = 60
boosted_model <- temp1(v=.05, number_of_weak_learners=60)
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=60)
cat('\ntrees: 60 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 60 --> RMSE:  0.09343663
```

```
# validation rmse; trees = 55
boosted_model <- temp1(v=.05, number_of_weak_learners=55)
```

```r
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=55)
cat('\ntrees: 55 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 55 --> RMSE:  0.09342942
```

```r
# validation rmse; trees = 52
boosted_model <- temp1(v=.05, number_of_weak_learners=52)
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=52)
cat('\ntrees: 52 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 52 --> RMSE:  0.09515657
```

```r
# trees = 50
boosted_model <- temp1(v=.05, number_of_weak_learners=50)
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=50)
cat('\ntrees: 50 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 50 --> RMSE:  0.09623499
```

```r
# trees = 40
boosted_model <- temp1(v=.05, number_of_weak_learners=40)
vd <- temp3(validation_data, boosted_model, number_of_weak_learners=40)
cat('\ntrees: 40 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 40 --> RMSE:  0.1137517
```

```r
# trees = 25
boosted_model <- temp1(v=.05, number_of_weak_learners=25)
vd <- temp3(validation_data, boosted_model, v=.05, number_of_weak_learners=25)
cat('\ntrees: 25 --> RMSE: ', rmse(vd$y, vd$yp))
```

```
##
## trees: 25 --> RMSE:  0.202964
```

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

minsplit of 60, cp of 0.01, maxdepth of 5 give the lowest rmse.

```r
temp1 <- function(v=.05, number_of_weak_learners=55, minsplit=20, cp=.01, maxdepth=30){
  # v is the learning rate.
  # number_of_weak_learners is the number of trees in boosted model.
  # minsplit is the minimum number of data points in a node where if smaller, theen can't split anymore
  # cp is the value that a split must decrease the loss function by.
  # maxdepth is the maximum depth of a tree.

  # Fit first iteration
  controls = rpart.control(minsplit = minsplit,
                minbucket = round(minsplit/3),
                cp = cp,
                maxcompete = 4, maxsurrogate = 5,
                usesurrogate = 2, xval = 10,
                surrogatestyle = 0, maxdepth = maxdepth)
  fit = rpart(y ~ x, data=df, control=controls)  # Regression tree.
  yp = predict(fit, newdata=df)
  df$yr = df$y - v*yp  # Update to residuals in boosting.
  YP = v*yp # This the spline prediction multiplied by a weight, known as the learning rate.
  list_of_weak_learners = list(fit)

  # Fitting rest of weak learners.
  for(t in 2:number_of_weak_learners){
    # Fit regression tree.
    fit = rpart(yr ~ x, data=df)

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

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

    # Bind to new data point; Adding v*yp to YP
    # Each column is a weak learner. Rows are the data points.
    YP = cbind(YP,v*yp)

    # Store fitted model in list
  list_of_weak_learners[[t]] = fit
  }
  return(list_of_weak_learners)
}
temp3 <- function(new_data, list_of_weak_learners, v=.05, number_of_weak_learners = 55){
  # Doing predictions on new data.
  # number_of_weak_learners must be the same number used in temp1() and temp2().
  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)
    }
  }
  return(new_data)
}
rmse <- function(y_true, y_pred){
  # Calculates root mean squared error.
  sq <- (y_true - y_pred)^2
  out <- sqrt(mean(sq))
  return(out)
}
gridsearch <- function(){
  minsplit = seq(10, 100, 10)
  cp = seq(.01, 10, 1)
  maxdepth = seq(2, 8, 1)
  results =  rep(NA, length(minsplit) * length(cp) * length(maxdepth))
  params = rep(NA, length(minsplit) * length(cp) * length(maxdepth))
  index = 1 # tracks index for results
  for(i in minsplit){
    for(j in cp){
      for(k in maxdepth){
        boosted_model <- temp1(v=.05, number_of_weak_learners=55, minsplit=i, cp=j, maxdepth=k)
        vd <- temp3(validation_data, boosted_model, v=.05, number_of_weak_learners=55)
        out = rmse(vd$y, vd$yp)
        # cat('\n','minsplit: ', i, 'cp: ', j, 'maxdepth: ', k, 'RMSE: ', out)
        results[index] = out
        params[index] = paste0('minsplit: ', i, ', cp: ', j, ', maxdepth: ', k) # c(minsplit, cp, maxde
        index = index + 1
      }
    }
  }
  return(c(results, params))
}
```

```
out <- gridsearch()
results <- out[1:700]
params <- out[701:1400]
results[which.min(results)]
```

```
## [1] "0.0922613267910517"
```

```
params[which.min(results)]
```

```
## [1] "minsplit: 60, cp: 0.01, maxdepth: 5"
```

Q5: The approach above can be extendewd to multiple dimensions. Repeat Q1 to Q3 for the data set kernel_regression_2.csv from Assignment 2.

```
data <- read.csv("kernel_regression_2.csv")  # Z is the target, x and y are the inputs.

## 75% of the sample size
smp_size <- floor(0.75 * nrow(data))

## set the seed to make your partition reproducible
set.seed(1)
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

df <- data[train_ind, ]
validation_data <- data[-train_ind, ]
```

```
temp1 <- function(v=.05, number_of_weak_learners=55, minsplit=20, cp=.01, maxdepth=30){
  # v is the learning rate.
  # number_of_weak_learners is the number of trees in boosted model.
  # minsplit is the minimum number of data points in a node where if smaller, theen can't split anymore
  # cp is the value that a split must decrease the loss function by.
  # maxdepth is the maximum depth of a tree.
  # Fit first iteration
  controls = rpart.control(minsplit = minsplit,
                minbucket = round(minsplit/3),
                cp = cp,
                maxcompete = 4, maxsurrogate = 5,
                usesurrogate = 2, xval = 10,
                surrogatestyle = 0, maxdepth = maxdepth)
  fit = rpart(z ~ x + y, data=df, control=controls)  # Regression tree.
  zp = predict(fit, newdata=df)
  df$zr = df$z - v*zp  # Update to residuals in boosting.
  ZP = v*zp # This the spline prediction multiplied by a weight, known as the learning rate.
  list_of_weak_learners = list(fit)

  # Fitting rest of weak learners.
  for(t in 2:number_of_weak_learners){
    # Fit regression tree.
    fit = rpart(zr ~ x+y, data=df)

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

    # Update residuals
    df$zr=df$zr - v*zp

    # Bind to new data point; Adding v*yp to YP
    # Each column is a weak learner. Rows are the data points.
    ZP = cbind(ZP,v*zp)

    # Store fitted model in list
  list_of_weak_learners[[t]] = fit
  }
  return(list_of_weak_learners)
}
temp3 <- function(new_data, list_of_weak_learners, v=.05, number_of_weak_learners = 55){
  # Doing predictions on new data.
  # number_of_weak_learners must be the same number used in temp1() and temp2().
  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(zp=pred)
    }
  }
  return(new_data)
}
rmse <- function(y_true, y_pred){
  # Calculates root mean squared error.
  sq <- (y_true - y_pred)^2
  out <- sqrt(mean(sq))
  return(out)
```

```
}
gridsearch <- function(){
  minsplit = seq(10, 100, 10)
  cp = seq(.01, 10, 1)
  maxdepth = seq(2, 8, 1)
  number_of_weak_learners = seq(80, 100, 20)
  results =  rep(NA, length(minsplit) * length(cp) * length(maxdepth))
  params = rep(NA, length(minsplit) * length(cp) * length(maxdepth))
  index = 1 # tracks index for results
  for(i in minsplit){
    for(j in cp){
      for(k in maxdepth){
        for(l in number_of_weak_learners){
          boosted_model <- temp1(v=.05, number_of_weak_learners=l, minsplit=i, cp=j, maxdepth=k)
          vd <- temp3(validation_data, boosted_model, v=.05, number_of_weak_learners=l)
          out = rmse(vd$z, vd$zp)
          # cat('\n','minsplit: ', i, 'cp: ', j, 'maxdepth: ', k, 'number of trees: ', l, 'RMSE: ', out
          results[index] = out
          params[index] = paste0('minsplit: ', i, ', cp: ', j, ', maxdepth: ', k, ', number of trees: '
          index = index + 1
        }
      }
    }
  }
  return(c(results, params))
}
```

```
out <- gridsearch()
```

```
results <- out[1:1400]
params <- out[1401:2800]
results[which.min(results)]
```

```
## [1] "0.356440605638118"
```

```
params[which.min(results)]
```

```
## [1] "minsplit: 10, cp: 0.01, maxdepth: 3, number of trees: 100"
```

Question 2 (TSNE)

Part 1

a.) Do the distances between points in tSNE matter?

The notion of distance changes depending on regional density variation. tSNE expands dense clusters and contracts sparse ones. As a result, clusters in tSNE representations looks approximately the same in size.

b.) What does the parameter value 'perplexity' mean?

It determines how to balance attention between local and global aspects of the data. It is a guess about the number of close neighbors each point has. Varying perplexity between 5 and 50 will cause nuanced changes to the plot. Each plot won't look exactly the same, but looking for common structure across multiple plots in this range might elude to the underlying structure of the data.

c.) What effect does the number of steps have on the final outcome of embedding?

The number of iterations ensures tSNE find a stable form.

d.) Explain why you may need more than one plot to explain topological information using tSNE.

tSNE is a very flexible method that can do all sorts of transformations and bring out different structure in data when there is no underlying structure. Looking across different plots will help elude the user to the true underlying structure. For instance, changing perplexity alone will result in a wide variety of different plots.

Part 2.

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.6.3

## -- Attaching packages -----------------------------------------------------------------------------

## v tibble  2.1.3     v purrr   0.3.3
## v readr   1.3.1     v forcats 0.5.0

## Warning: package 'tibble' was built under R version 3.6.1

## Warning: package 'readr' was built under R version 3.6.3

## Warning: package 'purrr' was built under R version 3.6.3

## Warning: package 'forcats' was built under R version 3.6.3

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

```
library(Rtsne)
```

```
## Warning: package 'Rtsne' was built under R version 3.6.3
```

```
library(RColorBrewer)
```

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

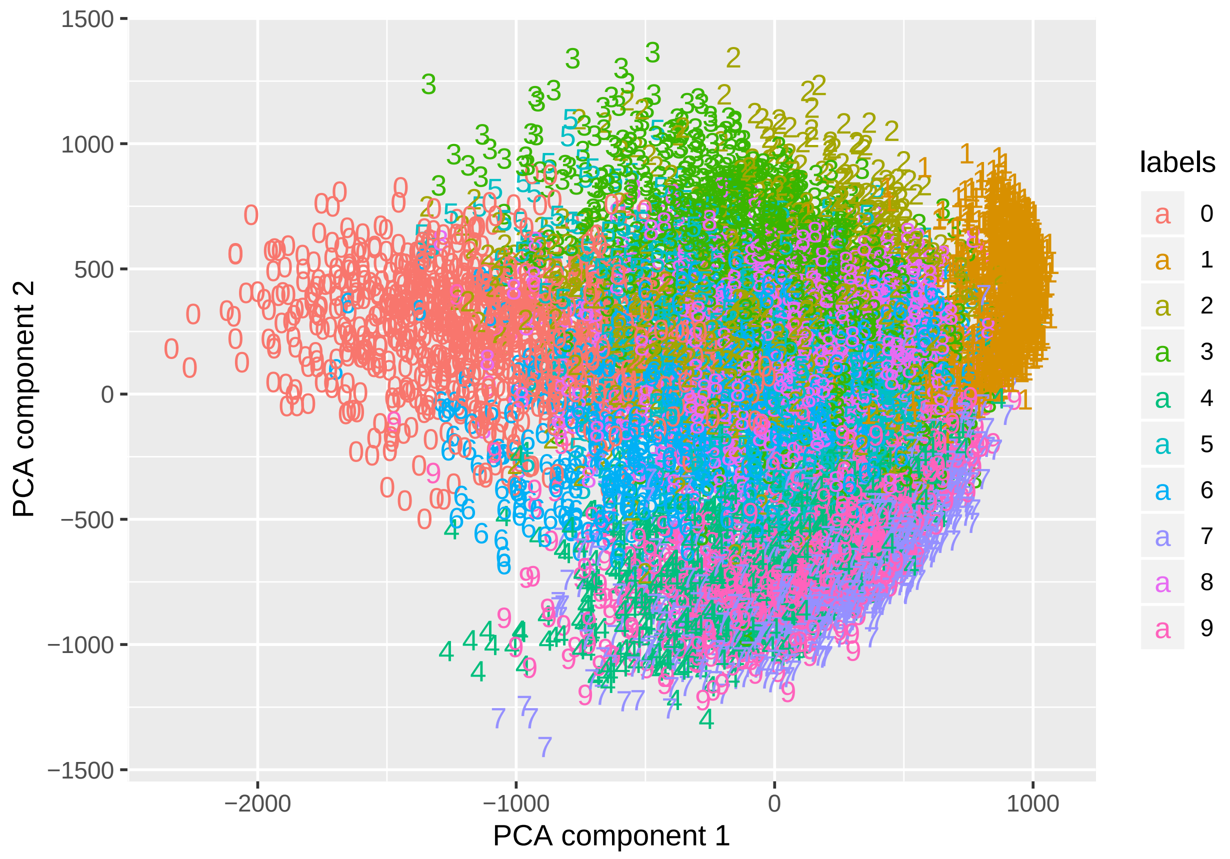
```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )

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

```
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.

```
pca = princomp(first_10k_samples)$scores[,1:2]
pca_plot = tibble(x = pca[,1], y =pca[,2], labels = as.character(first_10k_samples_labels))
```

```r
ggplot(aes(x = x, y=y,label = labels, color = labels), data = pca_plot) + geom_text() +
  xlab('PCA component 1') +ylab('PCA component 2')
```

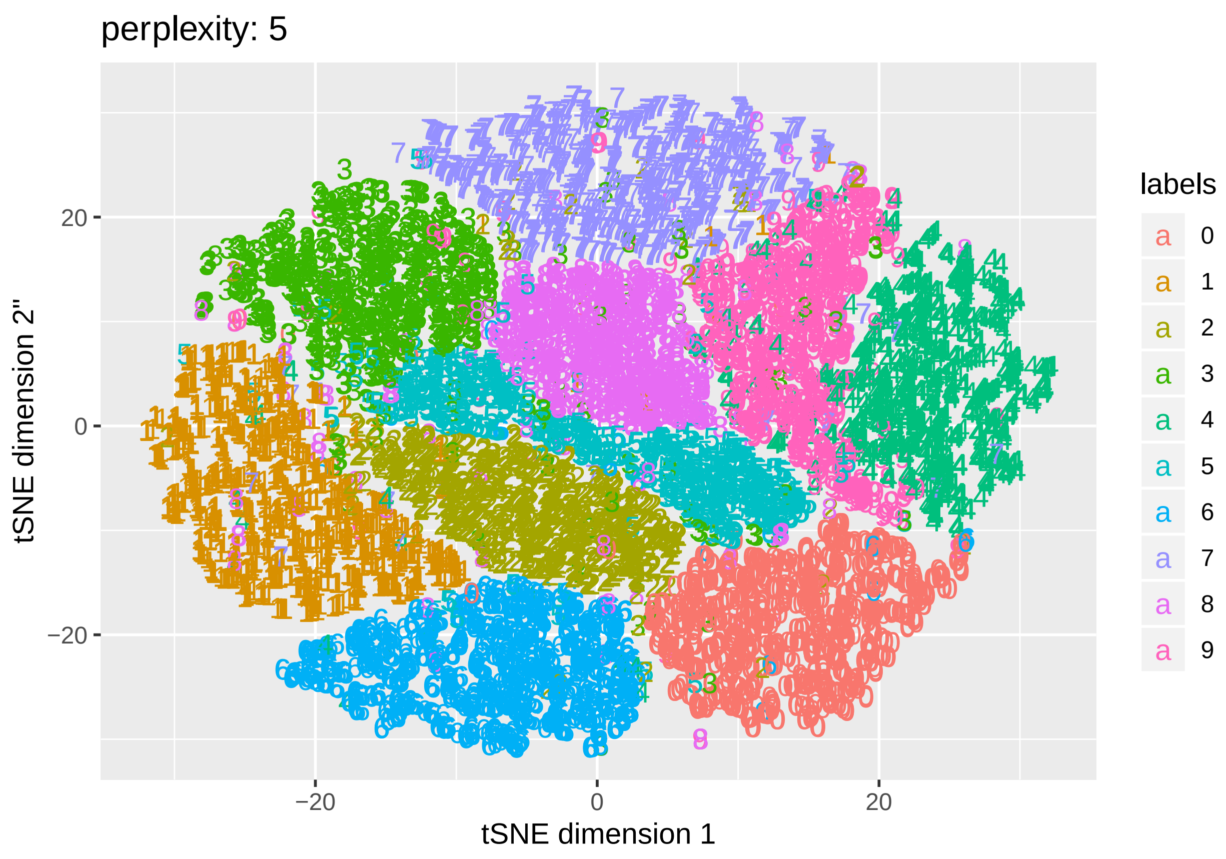


b.) Plot the TSNE embedding for perplexity = 5 and use 500 iterations.

```r
temp <- c()
perp <- c()
pep <- 5
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = pep,
                  theta = 0.5,
                  eta = 200, # learning rate.
                  pca = TRUE, verbose = TRUE,
                  max_iter = 500, # number of iterations.
                  num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 5.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 0.64 seconds (sparsity = 0.002104)!
## Learning embedding...
## Iteration 50: error is 118.296632 (50 iterations in 0.54 seconds)
## Iteration 100: error is 107.093728 (50 iterations in 0.56 seconds)
## Iteration 150: error is 99.692623 (50 iterations in 0.53 seconds)
## Iteration 200: error is 97.043226 (50 iterations in 0.54 seconds)
## Iteration 250: error is 95.394515 (50 iterations in 0.54 seconds)
## Iteration 300: error is 4.379519 (50 iterations in 0.53 seconds)
## Iteration 350: error is 3.815021 (50 iterations in 0.56 seconds)
## Iteration 400: error is 3.447625 (50 iterations in 0.58 seconds)
## Iteration 450: error is 3.179851 (50 iterations in 0.60 seconds)
## Iteration 500: error is 2.974794 (50 iterations in 0.63 seconds)
## Fitting performed in 5.61 seconds.
```

```
plot_emb <- function(embedding, pep){
  # Wrapper to plot embedding.
  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(paste0("perplexity: ",pep))
}
plot_emb(embedding, pep)
```



```
temp <- append(temp, embedding$itercosts[length(embedding$itercosts)])
perp <- append(perp, pep)
```

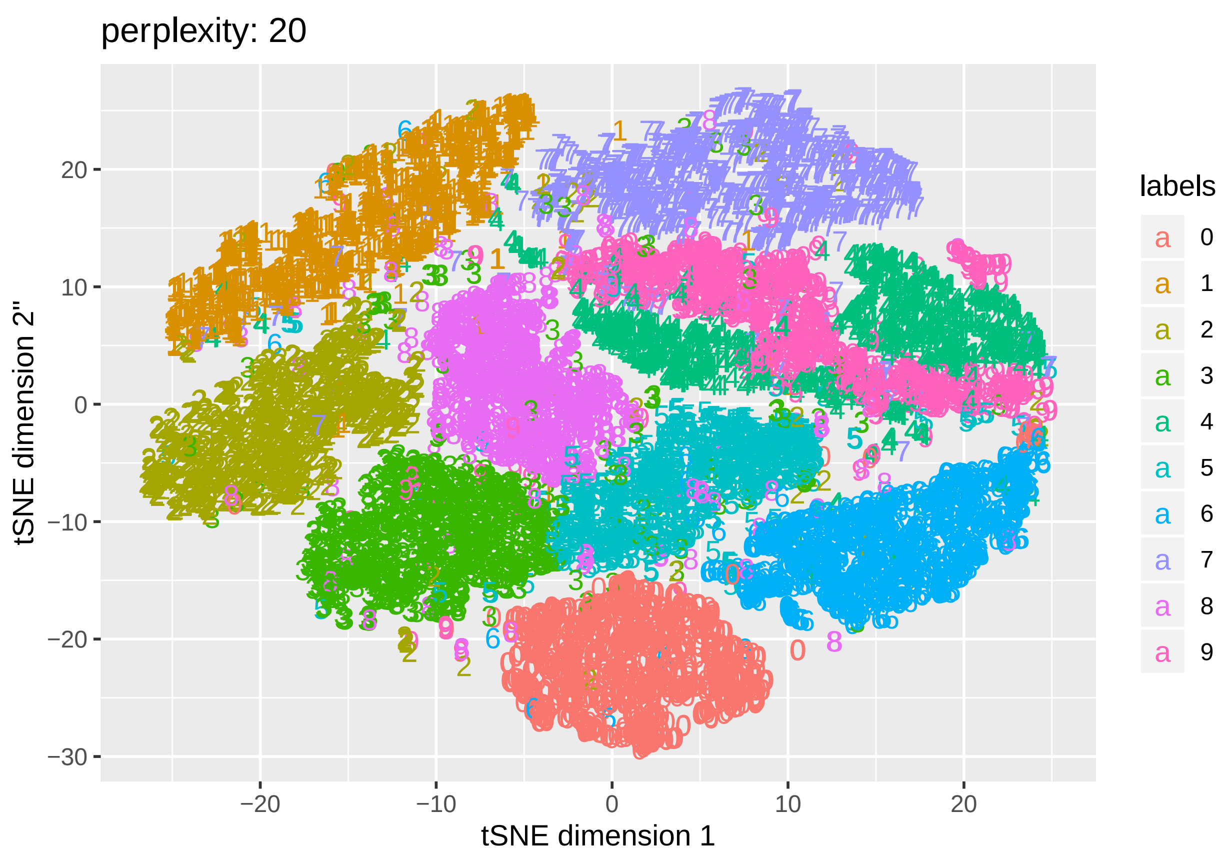
c.) Plot the TSNE embedding for perplexity =5,20,60,100,125,160, what do you notice?

Can see a gap appears between clusters and clusters are being stretched or rotated on the x-axis, tSNE dimension 1.

```
pep <- 20
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = pep,
                  theta = 0.5,
                  eta = 200, # learning rate.
                  pca = TRUE, verbose = TRUE,
                  max_iter = 500, # number of iterations.
                  num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 20.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 0.84 seconds (sparsity = 0.008217)!
## Learning embedding...
## Iteration 50: error is 102.343943 (50 iterations in 0.55 seconds)
## Iteration 100: error is 92.707271 (50 iterations in 0.56 seconds)
## Iteration 150: error is 88.495508 (50 iterations in 0.55 seconds)
## Iteration 200: error is 87.605685 (50 iterations in 0.56 seconds)
## Iteration 250: error is 87.350292 (50 iterations in 0.56 seconds)
## Iteration 300: error is 3.392819 (50 iterations in 0.52 seconds)
## Iteration 350: error is 2.940838 (50 iterations in 0.53 seconds)
## Iteration 400: error is 2.687773 (50 iterations in 0.54 seconds)
## Iteration 450: error is 2.519106 (50 iterations in 0.54 seconds)
## Iteration 500: error is 2.396772 (50 iterations in 0.55 seconds)
## Fitting performed in 5.47 seconds.
```

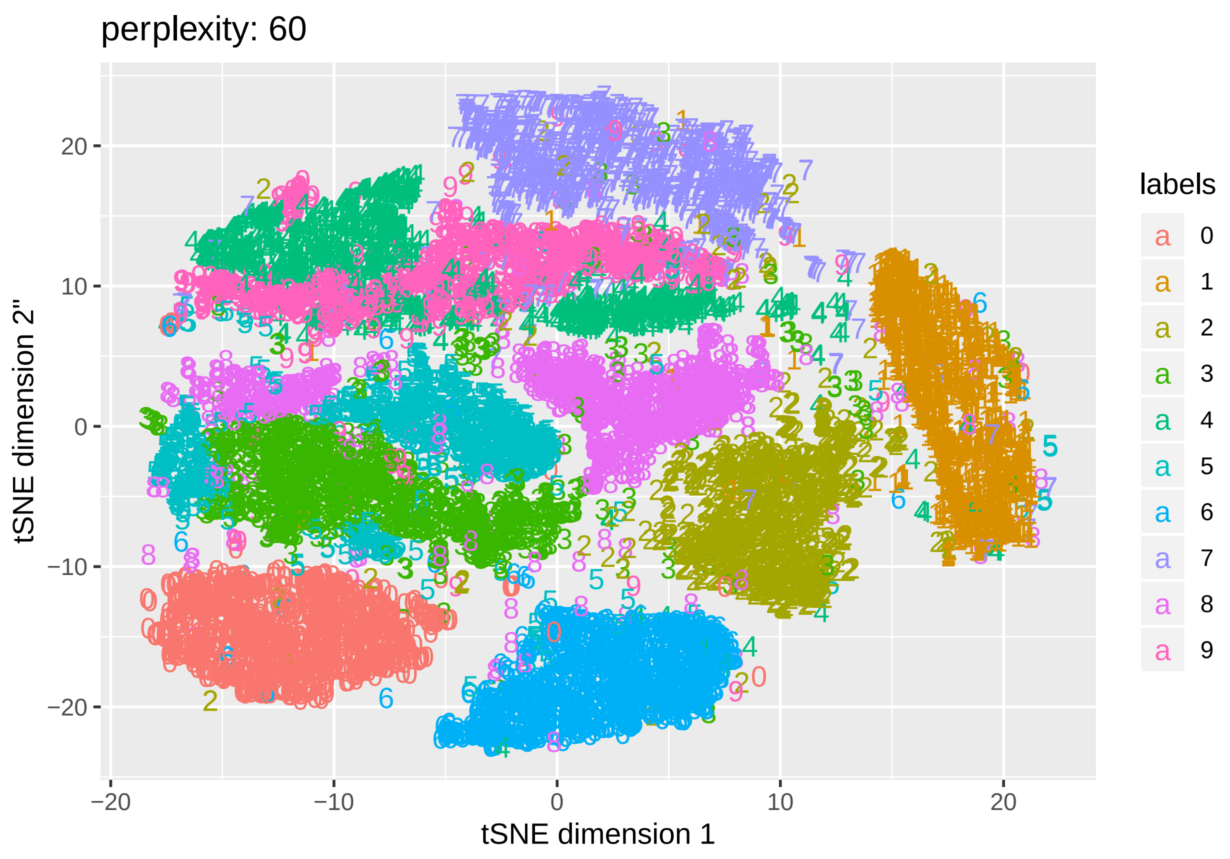
```
temp <- append(temp, embedding$itercosts[length(embedding$itercosts)])
perp <- append(perp, pep)

plot_emb <- function(embedding, pep){
  # Wrapper to plot embedding.
  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(paste0("perplexity: ",pep))
}
plot_emb(embedding, pep)
```



```
pep <- 60
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = pep,
                  theta = 0.5,
                  eta = 200, # learning rate.
                  pca = TRUE, verbose = TRUE,
                  max_iter = 500, # number of iterations.
                  num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 60.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 1.56 seconds (sparsity = 0.024328)!
## Learning embedding...
## Iteration 50: error is 89.428527 (50 iterations in 0.60 seconds)
## Iteration 100: error is 84.825082 (50 iterations in 0.60 seconds)
## Iteration 150: error is 81.618001 (50 iterations in 0.58 seconds)
## Iteration 200: error is 81.557264 (50 iterations in 0.58 seconds)
## Iteration 250: error is 81.553220 (50 iterations in 0.58 seconds)
## Iteration 300: error is 2.624199 (50 iterations in 0.57 seconds)
## Iteration 350: error is 2.284436 (50 iterations in 0.57 seconds)
## Iteration 400: error is 2.109506 (50 iterations in 0.58 seconds)
## Iteration 450: error is 1.998635 (50 iterations in 0.57 seconds)
## Iteration 500: error is 1.921933 (50 iterations in 0.58 seconds)
## Fitting performed in 5.81 seconds.
```

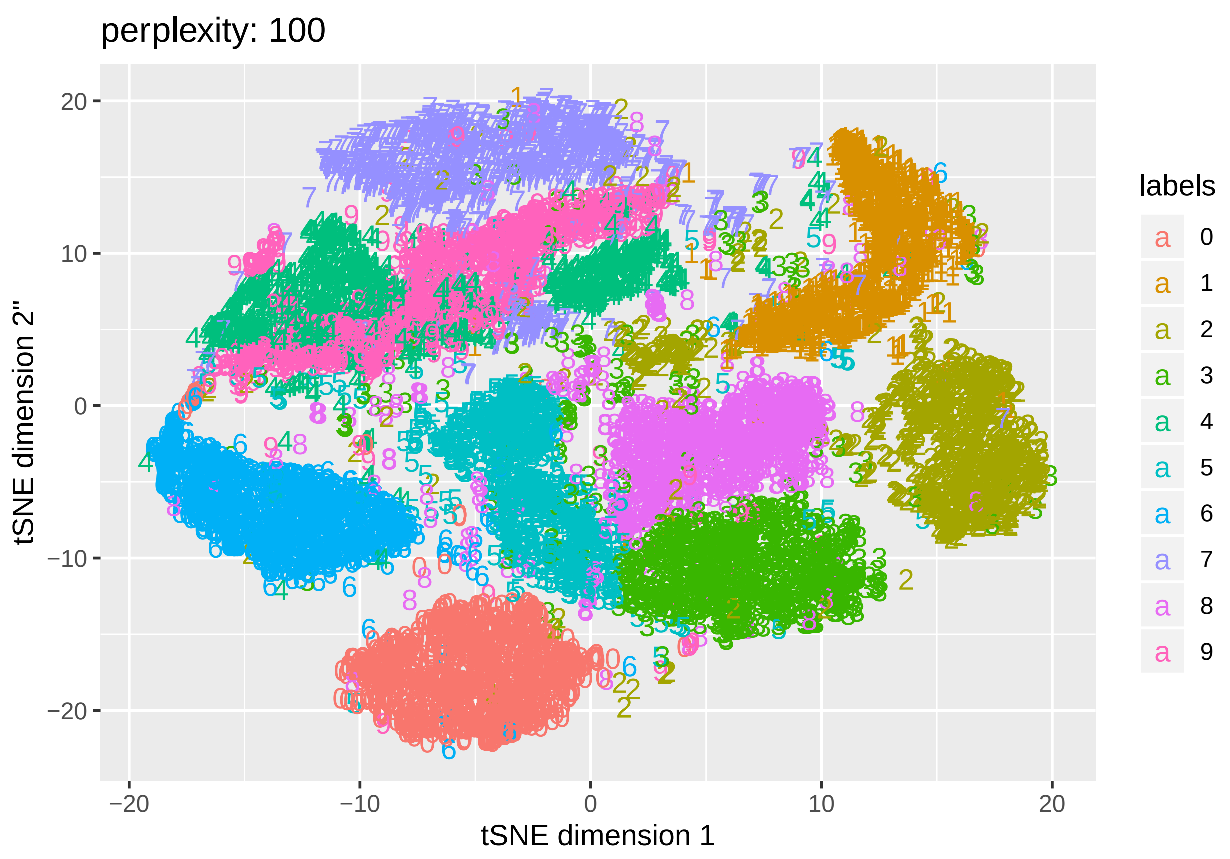
```
temp <- append(temp, embedding$itercosts[length(embedding$itercosts)])
perp <- append(perp, pep)

plot_emb <- function(embedding, pep){
  # Wrapper to plot embedding.
  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(paste0("perplexity: ",pep))
}
plot_emb(embedding, pep)
```



```
pep <- 100
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = pep,
                  theta = 0.5,
                  eta = 200, # learning rate.
                  pca = TRUE, verbose = TRUE,
                  max_iter = 500, # number of iterations.
                  num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 100.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 2.70 seconds (sparsity = 0.040571)!
## Learning embedding...
## Iteration 50: error is 83.371128 (50 iterations in 0.63 seconds)
## Iteration 100: error is 82.395070 (50 iterations in 0.67 seconds)
## Iteration 150: error is 78.335127 (50 iterations in 0.66 seconds)
## Iteration 200: error is 78.271214 (50 iterations in 0.65 seconds)
## Iteration 250: error is 78.247942 (50 iterations in 0.64 seconds)
## Iteration 300: error is 2.267060 (50 iterations in 0.63 seconds)
## Iteration 350: error is 1.976042 (50 iterations in 0.62 seconds)
## Iteration 400: error is 1.831931 (50 iterations in 0.62 seconds)
## Iteration 450: error is 1.744623 (50 iterations in 0.62 seconds)
## Iteration 500: error is 1.685757 (50 iterations in 0.65 seconds)
## Fitting performed in 6.40 seconds.
```

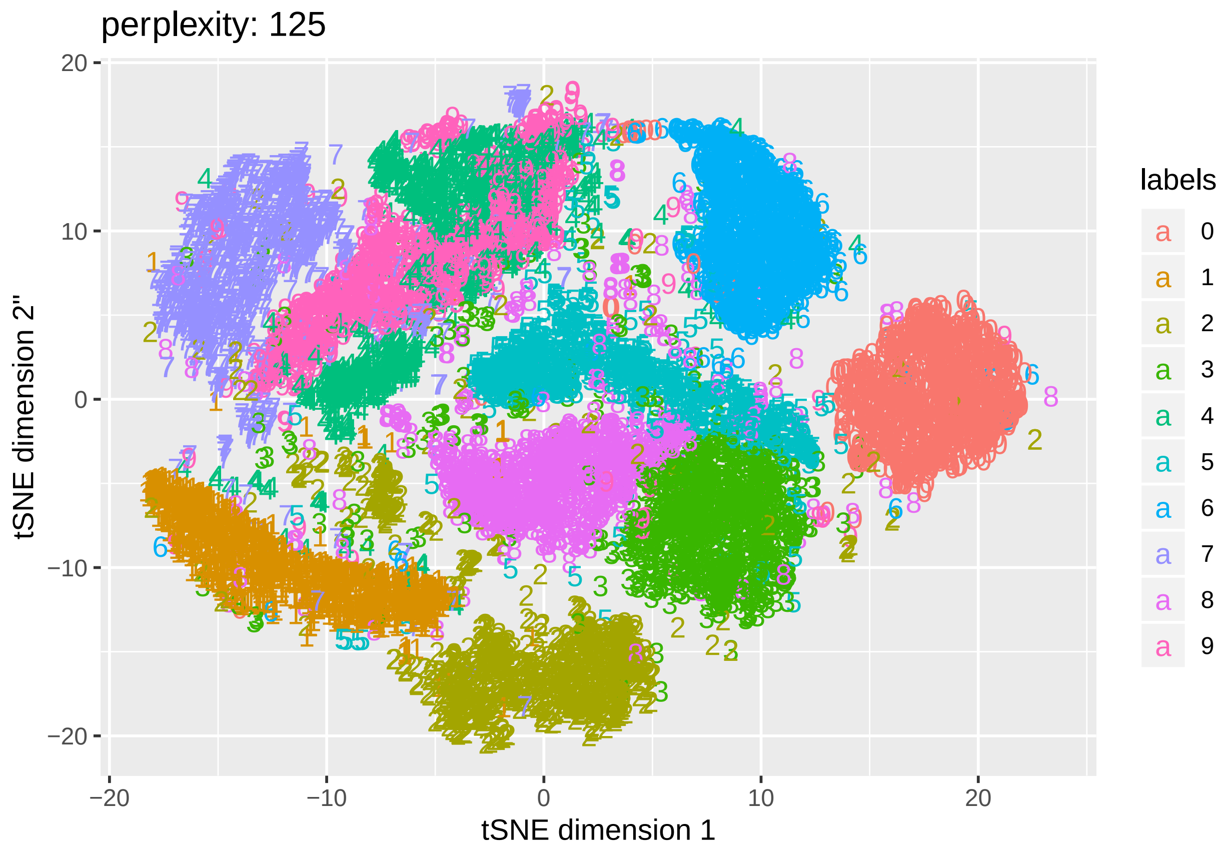
```
temp <- append(temp, embedding$itercosts[length(embedding$itercosts)])
perp <- append(perp, pep)

plot_emb <- function(embedding, pep){
  # Wrapper to plot embedding.
  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(paste0("perplexity: ",pep))
}
plot_emb(embedding, pep)
```



```
pep <- 125
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = pep,
                  theta = 0.5,
                  eta = 200, # learning rate.
                  pca = TRUE, verbose = TRUE,
                  max_iter = 500, # number of iterations.
                  num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 125.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 3.59 seconds (sparsity = 0.050806)!
## Learning embedding...
## Iteration 50: error is 80.714217 (50 iterations in 0.67 seconds)
## Iteration 100: error is 80.700196 (50 iterations in 0.69 seconds)
## Iteration 150: error is 76.764721 (50 iterations in 0.70 seconds)
## Iteration 200: error is 76.740705 (50 iterations in 0.68 seconds)
## Iteration 250: error is 76.732503 (50 iterations in 0.68 seconds)
## Iteration 300: error is 2.131652 (50 iterations in 0.66 seconds)
## Iteration 350: error is 1.855905 (50 iterations in 0.65 seconds)
## Iteration 400: error is 1.722410 (50 iterations in 0.64 seconds)
## Iteration 450: error is 1.644363 (50 iterations in 0.65 seconds)
## Iteration 500: error is 1.592143 (50 iterations in 0.65 seconds)
## Fitting performed in 6.69 seconds.
```

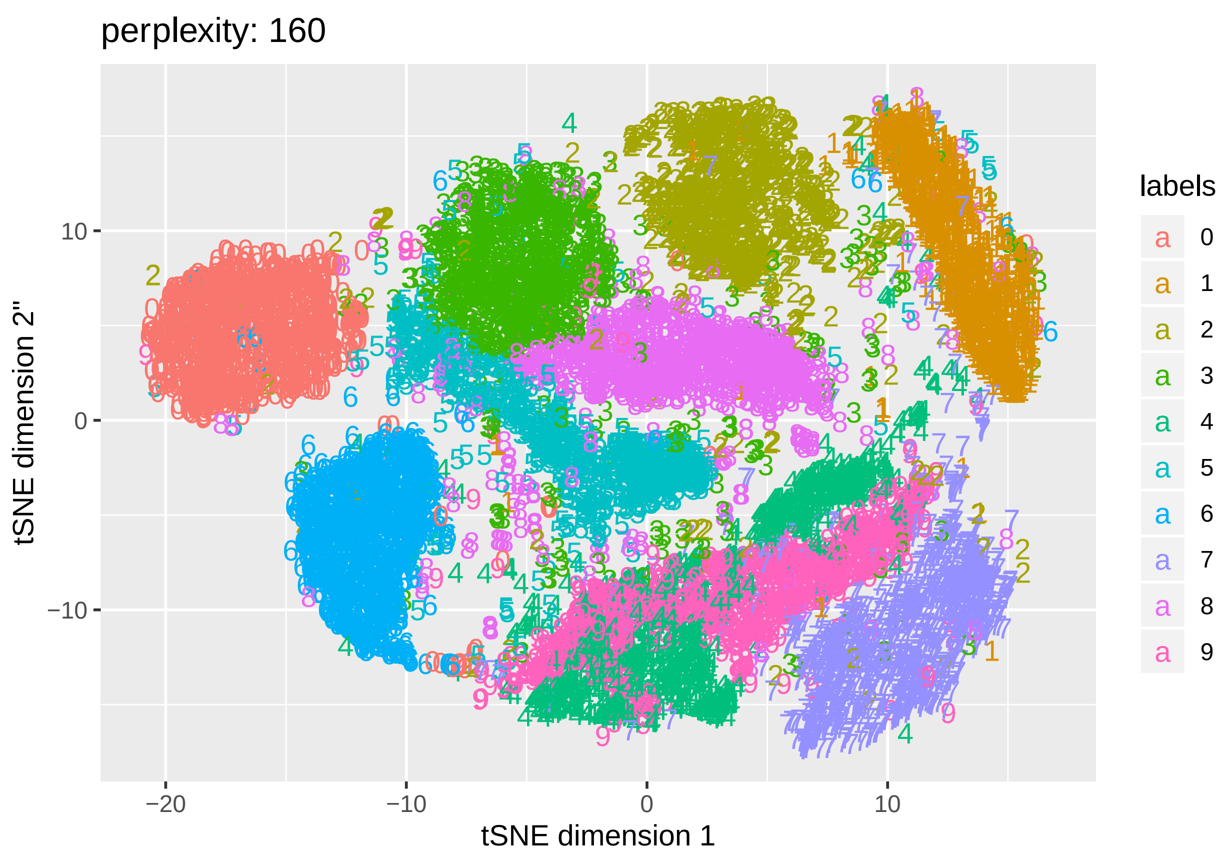
```
temp <- append(temp, embedding$itercosts[length(embedding$itercosts)])
perp <- append(perp, pep)

plot_emb <- function(embedding, pep){
  # Wrappper to plot embedding.
  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(paste0("perplexity: ",pep))
}
plot_emb(embedding, pep)
```



```
pep <- 160
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = pep,
                  theta = 0.5,
                  eta = 200, # learning rate.
                  pca = TRUE, verbose = TRUE,
                  max_iter = 500, # number of iterations.
                  num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 160.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 5.07 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 0.72 seconds)
## Iteration 100: error is 77.739049 (50 iterations in 0.72 seconds)
## Iteration 150: error is 75.195195 (50 iterations in 0.72 seconds)
## Iteration 200: error is 74.932254 (50 iterations in 0.70 seconds)
## Iteration 250: error is 74.918588 (50 iterations in 0.70 seconds)
## Iteration 300: error is 2.025450 (50 iterations in 0.69 seconds)
## Iteration 350: error is 1.744454 (50 iterations in 0.68 seconds)
## Iteration 400: error is 1.614932 (50 iterations in 0.68 seconds)
## Iteration 450: error is 1.539436 (50 iterations in 0.69 seconds)
## Iteration 500: error is 1.489349 (50 iterations in 0.69 seconds)
## Fitting performed in 6.98 seconds.
```

```r
temp <- append(temp, embedding$itercosts[length(embedding$itercosts)])
perp <- append(perp, pep)

plot_emb <- function(embedding, pep){
  # Wrapper to plot embedding.
  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(paste0("perplexity: ",pep))
}
plot_emb(embedding, pep)
```



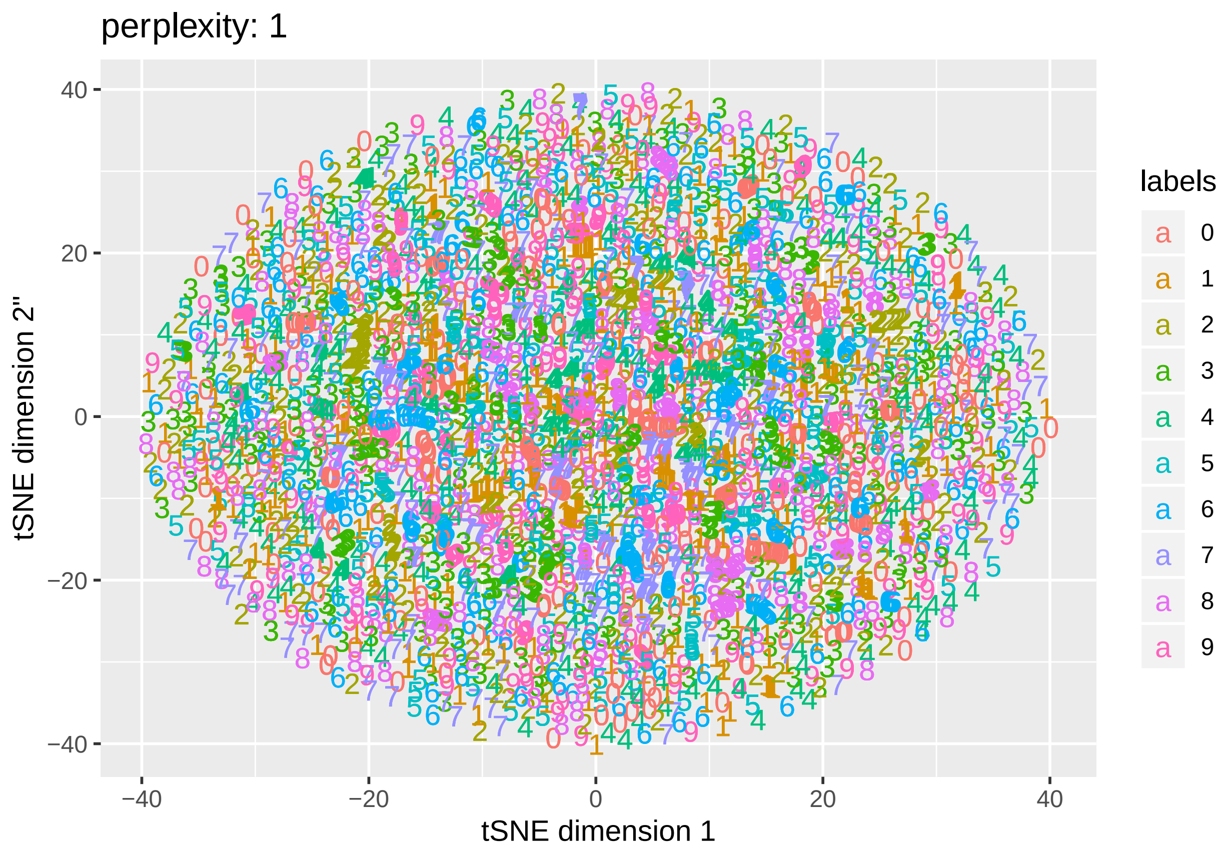
d.) If the perplexity is set to 1, what would the distribution of values look like in 2d, provide an explanation as to why.

The distribution of value will look random and uniform. Perplexity represents the number of neighbors used in manifold learning. This means that a perplexity of 1 causes the algorithm to use only 1 neighbor. Therefore, it would be hard, in large datasets, to find patterns or underlying structure.

```
pep <- 1
embedding = Rtsne(X = first_10k_samples, dims = 2,
                  perplexity = pep,
                  theta = 0.5,
                  eta = 200, # learning rate.
                  pca = TRUE, verbose = TRUE,
                  max_iter = 500, # number of iterations.
                  num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 1.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 0.54 seconds (sparsity = 0.000442)!
## Learning embedding...
## Iteration 50: error is 135.176605 (50 iterations in 0.62 seconds)
## Iteration 100: error is 116.584875 (50 iterations in 0.67 seconds)
## Iteration 150: error is 108.518713 (50 iterations in 0.87 seconds)
## Iteration 200: error is 104.020155 (50 iterations in 1.01 seconds)
## Iteration 250: error is 100.905375 (50 iterations in 1.10 seconds)
## Iteration 300: error is 5.265184 (50 iterations in 1.04 seconds)
## Iteration 350: error is 4.626883 (50 iterations in 1.25 seconds)
## Iteration 400: error is 4.129754 (50 iterations in 1.35 seconds)
## Iteration 450: error is 3.739940 (50 iterations in 1.38 seconds)
## Iteration 500: error is 3.425963 (50 iterations in 1.46 seconds)
## Fitting performed in 10.73 seconds.
```

```
plot_emb <- function(embedding, pep){
  # Wrapper to plot embedding.
  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(paste0("perplexity: ",pep))
}
plot_emb(embedding, pep)
```



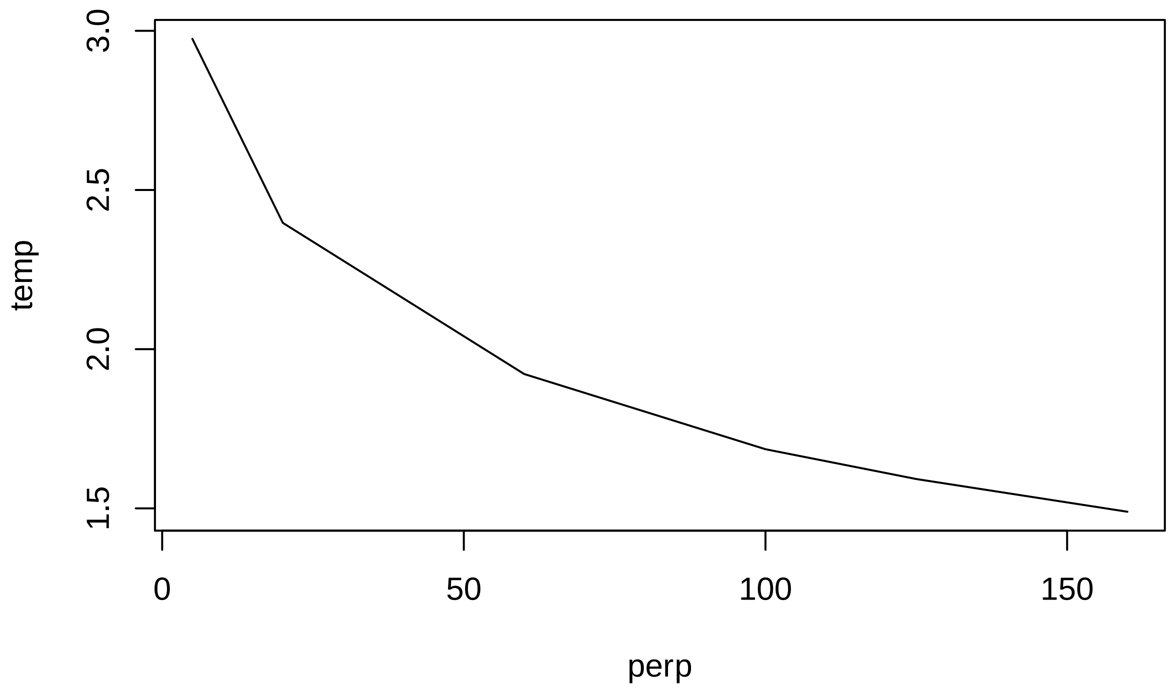
e.) How about if the perplexity is set to 5000? What would the distribut ion of values look like in 2d? Provide an explanation as to why. Note, don't try this on computer -Rtsne is not optimized for distance calculations and will take a long time to run.

I think the plot would look similar to when perplexity is 1. It might not be as evenly distributed in a spherical shape, but we won't get clean clusters like in part c.

f.) Plot iter_cost (KL divergence) against perplexity. What is the optimal perplexity value from the set of perplexities above? Why?

Optimal value of perplexity is at 160 because that had the lowest KL divergence.

```r
plot(perp, temp, 'l')
```



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?

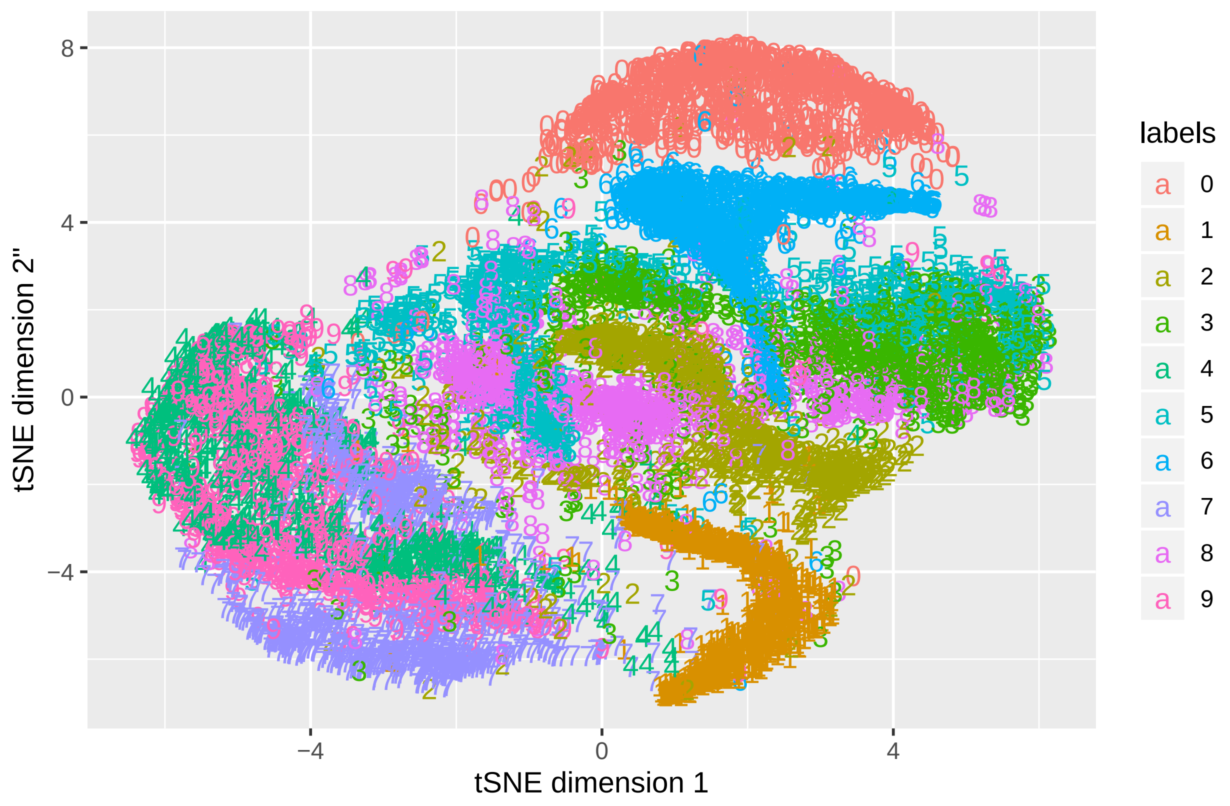
The learning rate affects tSNE is two ways. First, the clusters seem to be better separated and formed. Secondly, it also rotates and shifts the clusters around.

```r
pep <- 160
plot_emb <- function(embedding, i){
  # Wrapper to plot embedding.
  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(paste0("Learning rate: ",i))
}

i = 10
embedding = Rtsne(X = first_10k_samples, dims = 2,
                perplexity = pep,
                theta = 0.5,
                eta = i, # learning rate.
                pca = TRUE, verbose = TRUE,
                max_iter = 500, # number of iterations.
                num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 160.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 5.08 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 0.71 seconds)
## Iteration 100: error is 77.767491 (50 iterations in 0.72 seconds)
## Iteration 150: error is 77.767491 (50 iterations in 0.72 seconds)
## Iteration 200: error is 77.767491 (50 iterations in 0.77 seconds)
## Iteration 250: error is 77.767490 (50 iterations in 0.80 seconds)
## Iteration 300: error is 3.911241 (50 iterations in 0.77 seconds)
## Iteration 350: error is 2.813733 (50 iterations in 0.73 seconds)
## Iteration 400: error is 2.446492 (50 iterations in 0.72 seconds)
## Iteration 450: error is 2.256965 (50 iterations in 0.72 seconds)
## Iteration 500: error is 2.130269 (50 iterations in 0.71 seconds)
## Fitting performed in 7.36 seconds.
```

```
plot_emb(embedding, i)
```

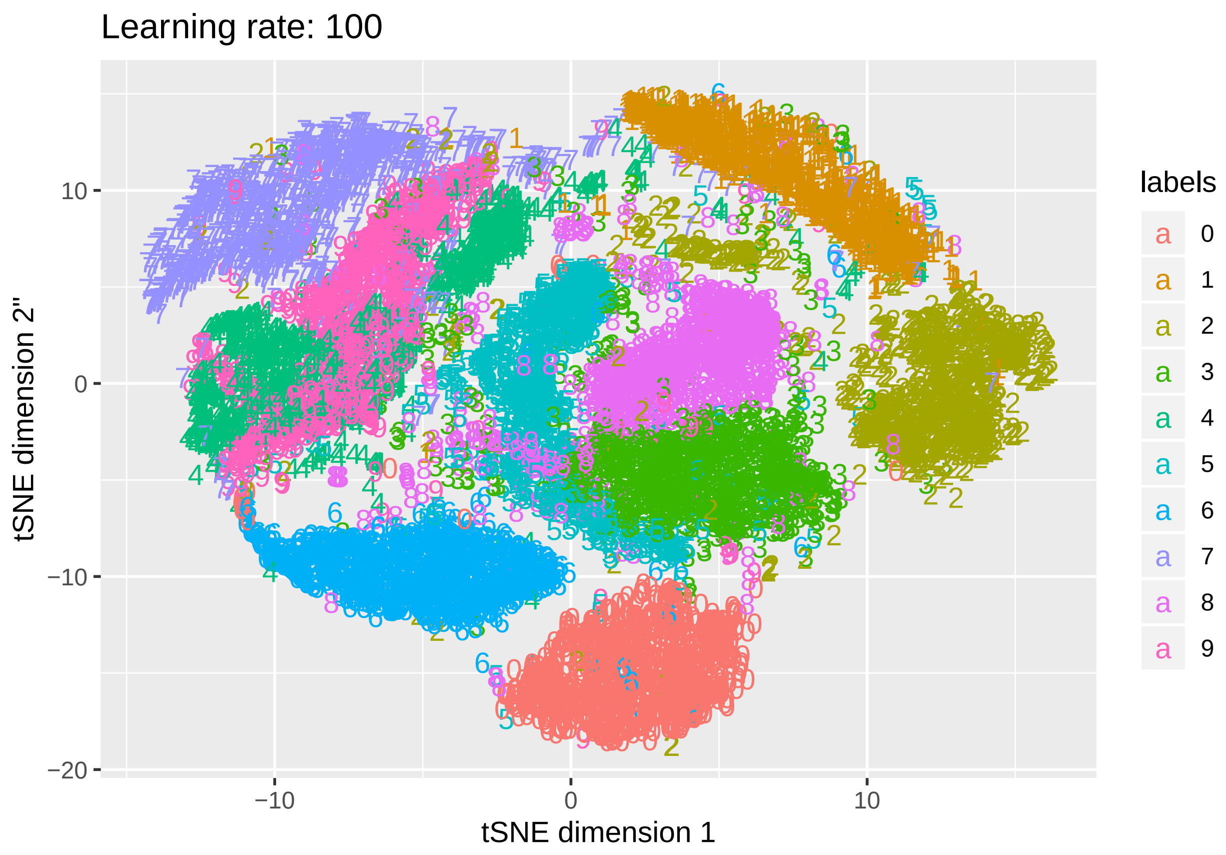
Learning rate: 10



```
i = 100
embedding = Rtsne(X = first_10k_samples, dims = 2,
                perplexity = pep,
                theta = 0.5,
                eta = i, # learning rate.
                pca = TRUE, verbose = TRUE,
                max_iter = 500, # number of iterations.
                num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 160.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 5.09 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 0.70 seconds)
## Iteration 100: error is 77.767490 (50 iterations in 0.73 seconds)
## Iteration 150: error is 76.742087 (50 iterations in 0.75 seconds)
## Iteration 200: error is 74.961881 (50 iterations in 0.72 seconds)
## Iteration 250: error is 74.930334 (50 iterations in 0.71 seconds)
## Iteration 300: error is 2.082697 (50 iterations in 0.70 seconds)
## Iteration 350: error is 1.826704 (50 iterations in 0.68 seconds)
## Iteration 400: error is 1.694990 (50 iterations in 0.68 seconds)
## Iteration 450: error is 1.611896 (50 iterations in 0.68 seconds)
## Iteration 500: error is 1.554673 (50 iterations in 0.69 seconds)
## Fitting performed in 7.04 seconds.
```

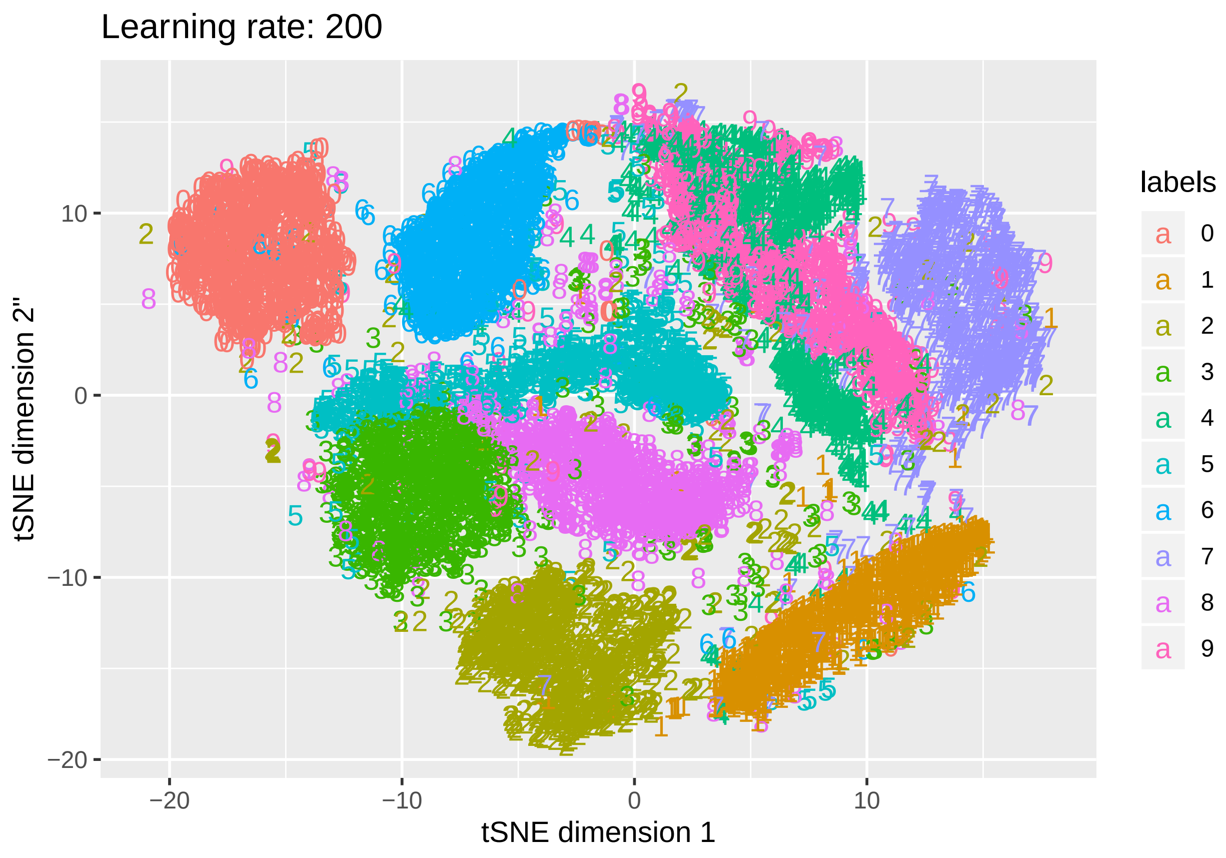
```
plot_emb(embedding, i)
```



```
i = 200
embedding = Rtsne(X = first_10k_samples, dims = 2,
                perplexity = pep,
                theta = 0.5,
                eta = i, # learning rate.
                pca = TRUE, verbose = TRUE,
                max_iter = 500, # number of iterations.
                num_threads=0)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 16 threads.
## Using no_dims = 2, perplexity = 160.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 5.04 seconds (sparsity = 0.065215)!
## Learning embedding...
## Iteration 50: error is 77.767491 (50 iterations in 0.70 seconds)
## Iteration 100: error is 77.695131 (50 iterations in 0.88 seconds)
## Iteration 150: error is 75.305843 (50 iterations in 0.85 seconds)
## Iteration 200: error is 74.945930 (50 iterations in 0.76 seconds)
## Iteration 250: error is 74.924692 (50 iterations in 0.71 seconds)
## Iteration 300: error is 2.000484 (50 iterations in 0.69 seconds)
## Iteration 350: error is 1.744165 (50 iterations in 0.69 seconds)
## Iteration 400: error is 1.613192 (50 iterations in 0.68 seconds)
## Iteration 450: error is 1.536506 (50 iterations in 0.69 seconds)
## Iteration 500: error is 1.487848 (50 iterations in 0.69 seconds)
## Fitting performed in 7.33 seconds.
```

```
plot_emb(embedding, i)
```



Part 2 word-2-vec and Gaussian processes.

```
library(wordVectors)
library(Rtsne)
library(tidytext)
```

```
## Warning: package 'tidytext' was built under R version 3.6.3
```

```
library(tidyverse)
```

2.) Run through the starter code set and try different ingredients, list your ingredients and the top 5 closest to them. They cannot be the same as presented in the starter code. Mark anything that is interesting.

```r
if (!file.exists("cookbooks.zip")) {
  download.file("http://archive.lib.msu.edu/dinfo/feedingamerica/cookbook_text.zip","cookbooks.zip")
}
unzip("cookbooks.zip",exdir="cookbooks")
if (!file.exists("cookbooks.txt")) prep_word2vec(origin="cookbooks",destination="cookbooks.txt",lowerca

# Training a Word2Vec model
if (!file.exists("cookbook_vectors.bin")) {
  model = train_word2vec("cookbooks.txt","cookbook_vectors.bin",
                         vectors=100,threads=4,window=6,
                         min_count = 10,
                         iter=5,negative_samples=15)
} else{
    model = read.vectors("cookbook_vectors.bin")
}
```

```
## Filename ends with .bin, so reading in binary format

## Reading a word2vec binary file of 18952 rows and 100 columns

##
  |
  |                                                                   |   0%
  |
  |                                                                   |   1%
  |
  |=                                                                  |   1%
  |
  |=                                                                  |   2%
  |
  |==                                                                 |   2%
  |
  |==                                                                 |   3%
  |
  |==                                                                 |   4%
  |
  |===                                                                |   4%
  |
  |===                                                                |   5%
  |
  |====                                                               |   5%
  |
  |====                                                               |   6%
  |
  |====                                                               |   7%
  |
  |=====                                                              |   7%
  |
  |=====                                                              |   8%
  |
  |======                                                             |   8%
```

```
|
|======                                                           |   9%
|
|======                                                           |  10%
|
|=======                                                          |  10%
|
|=======                                                          |  11%
|
|=======                                                          |  12%
|
|========                                                         |  12%
|
|========                                                         |  13%
|
|=========                                                        |  13%
|
|=========                                                        |  14%
|
|=========                                                        |  15%
|
|==========                                                       |  15%
|
|==========                                                       |  16%
|
|===========                                                      |  16%
|
|===========                                                      |  17%
|
|===========                                                      |  18%
|
|============                                                     |  18%
|
|============                                                     |  19%
|
|=============                                                    |  19%
|
|=============                                                    |  20%
|
|=============                                                    |  21%
|
|==============                                                   |  21%
|
|==============                                                   |  22%
|
|===============                                                  |  22%
|
|===============                                                  |  23%
|
|===============                                                  |  24%
|
|================                                                 |  24%
|
|================                                                 |  25%
```

```
|
|=================                                                |  25%
|
|=================                                                |  26%
|
|=================                                                |  27%
|
|==================                                               |  27%
|
|==================                                               |  28%
|
|===================                                              |  28%
|
|===================                                              |  29%
|
|===================                                              |  30%
|
|====================                                             |  30%
|
|====================                                             |  31%
|
|====================                                             |  32%
|
|=====================                                            |  32%
|
|=====================                                            |  33%
|
|======================                                           |  33%
|
|======================                                           |  34%
|
|======================                                           |  35%
|
|=======================                                          |  35%
|
|=======================                                          |  36%
|
|========================                                         |  36%
|
|========================                                         |  37%
|
|========================                                         |  38%
|
|=========================                                        |  38%
|
|=========================                                        |  39%
|
|==========================                                       |  39%
|
|==========================                                       |  40%
|
|==========================                                       |  41%
|
|===========================                                      |  41%
```

```
|
|===========================                                      |  42%
|
|============================                                     |  42%
|
|============================                                     |  43%
|
|============================                                     |  44%
|
|=============================                                    |  44%
|
|=============================                                    |  45%
|
|==============================                                   |  45%
|
|==============================                                   |  46%
|
|==============================                                   |  47%
|
|===============================                                  |  47%
|
|===============================                                  |  48%
|
|================================                                 |  48%
|
|================================                                 |  49%
|
|================================                                 |  50%
|
|=================================                                |  50%
|
|=================================                                |  51%
|
|=================================                                |  52%
|
|==================================                               |  52%
|
|==================================                               |  53%
|
|===================================                              |  53%
|
|===================================                              |  54%
|
|===================================                              |  55%
|
|====================================                             |  55%
|
|====================================                             |  56%
|
|=====================================                            |  56%
|
|=====================================                            |  57%
|
|=====================================                            |  58%
```

```
|
|======================================                            |  58%
|
|======================================                            |  59%
|
|=======================================                           |  59%
|
|=======================================                           |  60%
|
|=======================================                           |  61%
|
|========================================                          |  61%
|
|========================================                          |  62%
|
|=========================================                         |  62%
|
|=========================================                         |  63%
|
|=========================================                         |  64%
|
|==========================================                        |  64%
|
|==========================================                        |  65%
|
|===========================================                       |  65%
|
|===========================================                       |  66%
|
|===========================================                       |  67%
|
|============================================                      |  67%
|
|============================================                      |  68%
|
|=============================================                     |  68%
|
|=============================================                     |  69%
|
|=============================================                     |  70%
|
|==============================================                    |  70%
|
|==============================================                    |  71%
|
|==============================================                    |  72%
|
|===============================================                   |  72%
|
|===============================================                   |  73%
|
|================================================                  |  73%
|
|================================================                  |  74%
```

```
|
|================================================                 |  75%
|
|=================================================                |  75%
|
|=================================================                |  76%
|
|==================================================               |  76%
|
|==================================================               |  77%
|
|==================================================               |  78%
|
|===================================================              |  78%
|
|===================================================              |  79%
|
|====================================================             |  79%
|
|====================================================             |  80%
|
|====================================================             |  81%
|
|=====================================================            |  81%
|
|=====================================================            |  82%
|
|======================================================           |  82%
|
|======================================================           |  83%
|
|======================================================           |  84%
|
|=======================================================          |  84%
|
|=======================================================          |  85%
|
|========================================================         |  85%
|
|========================================================         |  86%
|
|========================================================         |  87%
|
|=========================================================        |  87%
|
|=========================================================        |  88%
|
|==========================================================       |  88%
|
|==========================================================       |  89%
|
|==========================================================       |  90%
|
|===========================================================      |  90%
```

```
  |
  |===========================================================      |  91%
  |
  |===========================================================      |  92%
  |
  |============================================================     |  92%
  |
  |============================================================     |  93%
  |
  |=============================================================    |  93%
  |
  |=============================================================    |  94%
  |
  |=============================================================    |  95%
  |
  |==============================================================   |  95%
  |
  |==============================================================   |  96%
  |
  |===============================================================  |  96%
  |
  |===============================================================  |  97%
  |
  |===============================================================  |  98%
  |
  |================================================================ |  98%
  |
  |================================================================ |  99%
  |
  |=================================================================|  99%
  |
  |=================================================================| 100%
```

```r
ingredient = 'tamarind'
ingredient_2 = 'chives'
ingredient_3 = 'sugar'
ingredient_4 = 'garlic'
list_of_ingredients = c(ingredient, ingredient_2, ingredient_3, ingredient_4)

model %>% closest_to(model[[list_of_ingredients]], 20)
```

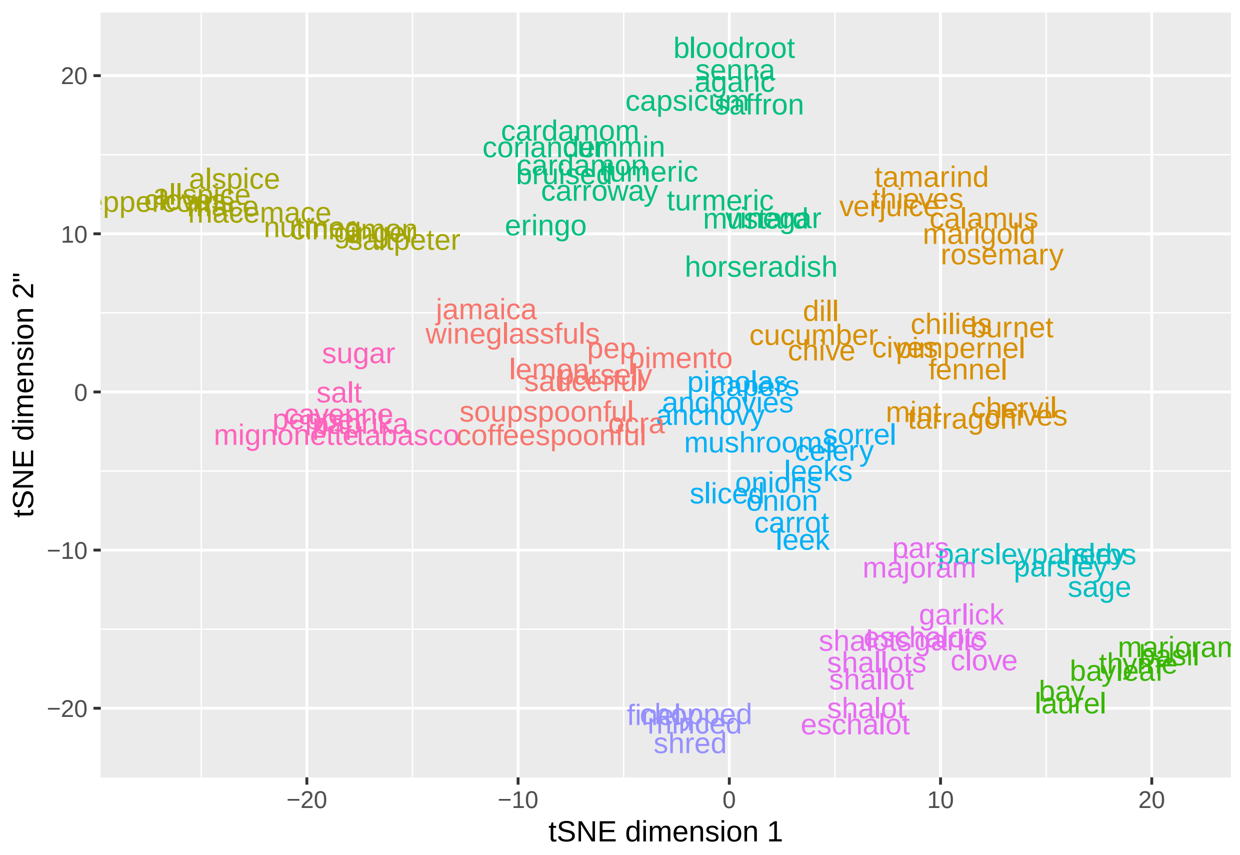
```
##              word similarity to model[[list_of_ingredients]]
## 1          garlic                                  0.7739218
## 2       eschalots                                  0.7571770
## 3        tarragon                                  0.7560458
## 4          chives                                  0.7386657
## 5        shallots                                  0.7129154
## 6         shallot                                  0.7016770
## 7          shalot                                  0.6959030
## 8         bayleaf                                  0.6835791
## 9          ginger                                  0.6833702
## 10      pimpernel                                  0.6782730
## 11         capers                                  0.6780106
## 12        parsley                                  0.6754827
## 13          clove                                  0.6743132
## 14        majoram                                  0.6714663
## 15           mint                                  0.6713989
## 16         celery                                  0.6681578
## 17       cinnamon                                  0.6673642
## 18        chervil                                  0.6657163
## 19 coffeespoonful                                  0.6655557
## 20        shalots                                  0.6654542
```

3.) Use TSNE to find interesting relationships amongst the different set of ingredients that you have selected.

```
# We have a list of potential herb-related words from old cookbooks.
n_words = 100
# selecting 100 closest ingredients to sage, thyme, basil.
# model[[list_of_ingredients]] returns a vector that is the average of the ingredient vectors.
closest_ingredients = closest_to(model,model[[list_of_ingredients]], n_words)$word
surrounding_ingredients = model[[closest_ingredients,average=F]] # don't return average

embedding = Rtsne(X = surrounding_ingredients, dims = 2,
                  perplexity = 4,
                  theta = 0.5,
                  eta = 10,
                  pca = TRUE, verbose = TRUE,
                  max_iter = 2000)
```

```
## Performing PCA
## Read the 100 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 4.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## Done in 0.00 seconds (sparsity = 0.172400)!
## Learning embedding...
## Iteration 50: error is 64.999954 (50 iterations in 0.01 seconds)
## Iteration 100: error is 64.999867 (50 iterations in 0.01 seconds)
## Iteration 150: error is 64.862888 (50 iterations in 0.01 seconds)
## Iteration 200: error is 63.824607 (50 iterations in 0.01 seconds)
## Iteration 250: error is 63.802806 (50 iterations in 0.01 seconds)
## Iteration 300: error is 1.090106 (50 iterations in 0.01 seconds)
## Iteration 350: error is 0.944397 (50 iterations in 0.01 seconds)
## Iteration 400: error is 0.902271 (50 iterations in 0.01 seconds)
## Iteration 450: error is 0.883490 (50 iterations in 0.01 seconds)
## Iteration 500: error is 0.875544 (50 iterations in 0.01 seconds)
## Iteration 550: error is 0.860538 (50 iterations in 0.01 seconds)
## Iteration 600: error is 0.857514 (50 iterations in 0.01 seconds)
## Iteration 650: error is 0.853609 (50 iterations in 0.01 seconds)
## Iteration 700: error is 0.851947 (50 iterations in 0.01 seconds)
## Iteration 750: error is 0.849282 (50 iterations in 0.01 seconds)
## Iteration 800: error is 0.846524 (50 iterations in 0.01 seconds)
## Iteration 850: error is 0.846660 (50 iterations in 0.01 seconds)
## Iteration 900: error is 0.846967 (50 iterations in 0.01 seconds)
## Iteration 950: error is 0.846794 (50 iterations in 0.01 seconds)
## Iteration 1000: error is 0.845070 (50 iterations in 0.01 seconds)
## Iteration 1050: error is 0.845554 (50 iterations in 0.01 seconds)
## Iteration 1100: error is 0.845091 (50 iterations in 0.01 seconds)
## Iteration 1150: error is 0.845828 (50 iterations in 0.01 seconds)
## Iteration 1200: error is 0.842268 (50 iterations in 0.01 seconds)
## Iteration 1250: error is 0.835845 (50 iterations in 0.01 seconds)
## Iteration 1300: error is 0.838225 (50 iterations in 0.01 seconds)
## Iteration 1350: error is 0.840702 (50 iterations in 0.01 seconds)
## Iteration 1400: error is 0.837600 (50 iterations in 0.01 seconds)
## Iteration 1450: error is 0.843170 (50 iterations in 0.01 seconds)
## Iteration 1500: error is 0.840481 (50 iterations in 0.01 seconds)
## Iteration 1550: error is 0.840732 (50 iterations in 0.01 seconds)
## Iteration 1600: error is 0.842598 (50 iterations in 0.01 seconds)
## Iteration 1650: error is 0.843596 (50 iterations in 0.01 seconds)
## Iteration 1700: error is 0.840756 (50 iterations in 0.01 seconds)
## Iteration 1750: error is 0.841898 (50 iterations in 0.01 seconds)
## Iteration 1800: error is 0.840279 (50 iterations in 0.01 seconds)
## Iteration 1850: error is 0.837712 (50 iterations in 0.01 seconds)
## Iteration 1900: error is 0.838400 (50 iterations in 0.01 seconds)
## Iteration 1950: error is 0.837605 (50 iterations in 0.01 seconds)
## Iteration 2000: error is 0.836635 (50 iterations in 0.01 seconds)
## Fitting performed in 0.31 seconds.
```

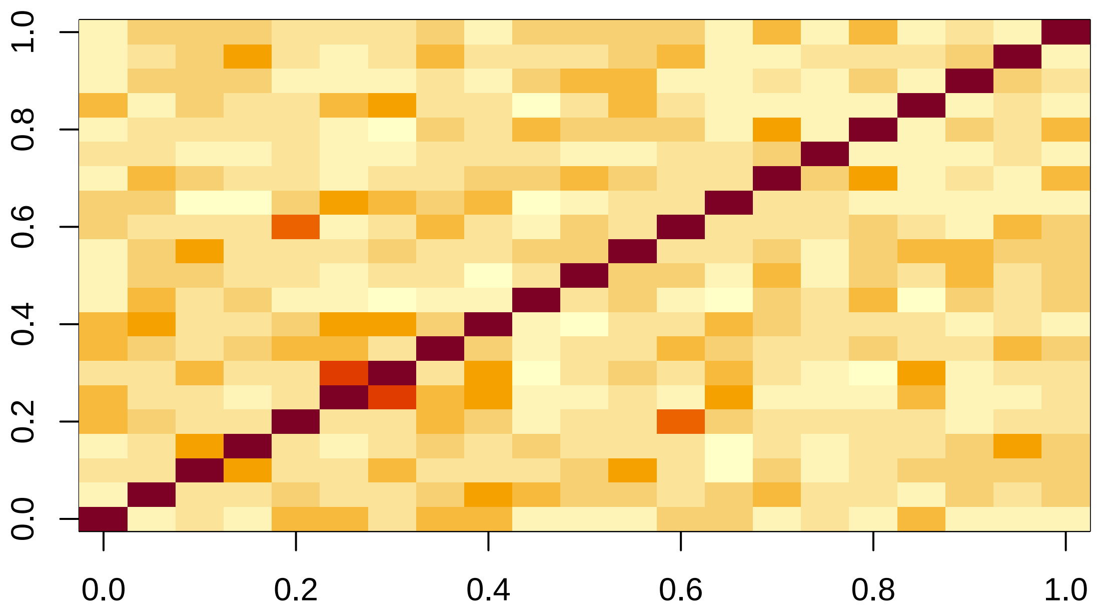
```r
embedding_vals = embedding$Y
rownames(embedding_vals) = rownames(surrounding_ingredients)

# Looking for clusters for embedding
set.seed(10)
n_centers = 10
clustering = kmeans(embedding_vals,centers=n_centers,
                    iter.max = 5)

# Setting up data for plotting
embedding_plot = tibble(x = embedding$Y[,1],
                        y = embedding$Y[,2],
                        labels = rownames(surrounding_ingredients)) %>%
  bind_cols(cluster = as.character(clustering$cluster))

# Visualizing TSNE output
ggplot(aes(x = x, y=y,label = labels, color = cluster), data = embedding_plot) +
  geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2"')+theme(legend.position = 'none')
```



4.) Replace the set of 'tastes' by 3 other orthogonal dimensions. Before selecting your three words, explore the set of words in the data set. Generate the plot for the set of words. Do they make sense?

Looking at the image and the poorly printed output of values, "egg" is roughly orthogonal to "juice" and "cake". Also, "cake" is roughly orthogonal to "juice".

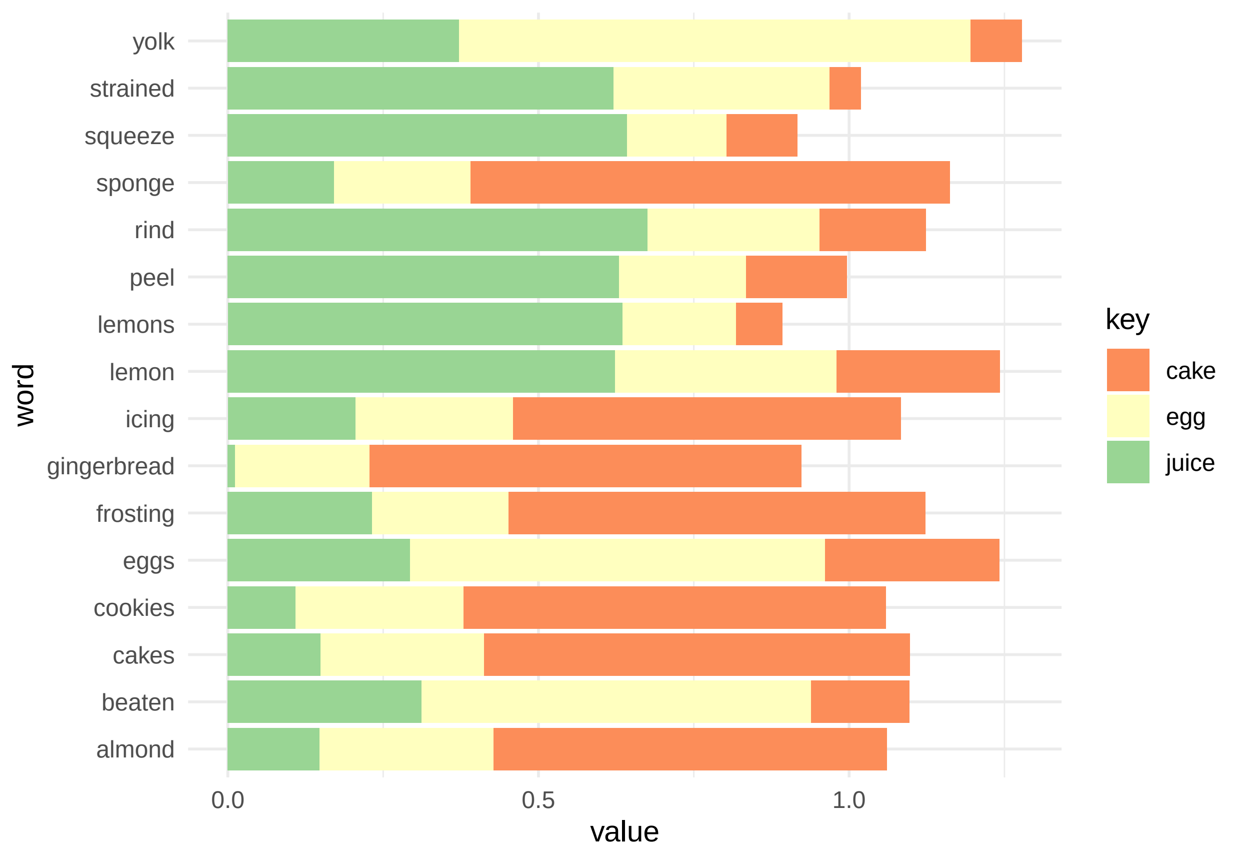
```r
temp <- cosineSimilarity(model[[100:120,]], model[[100:120,]])
image(temp)
```



```
temp
```

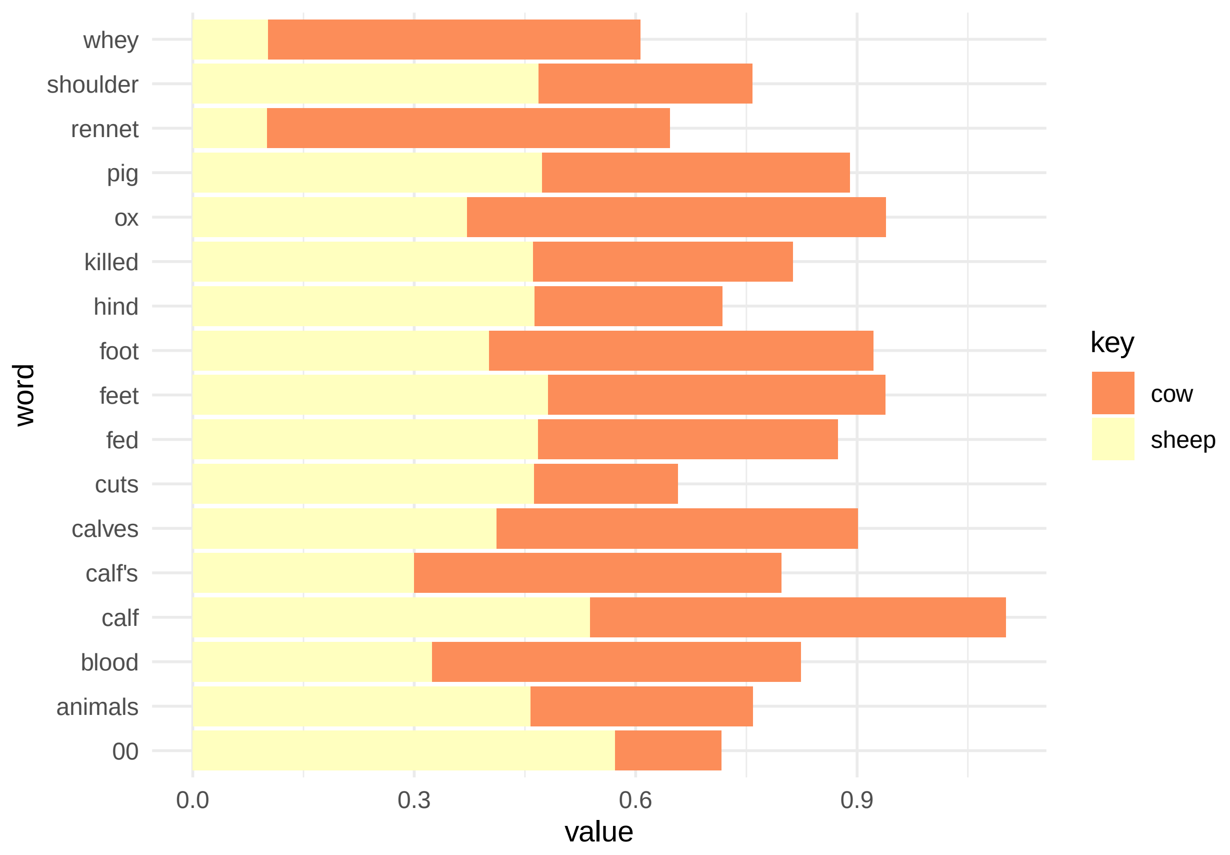
```
##               egg       good     before        off      lemon       bake
## egg     1.00000000 0.08091263 0.17551665 0.09430084 0.35655590 0.30661792
## good    0.08091263 1.00000000 0.15838475 0.13529849 0.23563665 0.15901998
## before  0.17551665 0.15838475 1.00000000 0.39214770 0.15442225 0.18753065
## off     0.09430084 0.13529849 0.39214770 1.00000000 0.11616436 0.09141339
## lemon   0.35655590 0.23563665 0.15442225 0.11616436 1.00000000 0.18398073
## bake    0.30661792 0.15901998 0.18753065 0.09141339 0.18398073 1.00000000
## oven    0.16415893 0.19426996 0.33546497 0.12552719 0.12369629 0.70167385
## mix     0.37588378 0.25199036 0.20300736 0.21113721 0.34122597 0.36637567
## brown   0.30077226 0.38612656 0.17554407 0.17285084 0.22882205 0.43233380
## their   0.04080624 0.31298377 0.14019303 0.22471201 0.02667400 0.03793992
## can     0.06526605 0.25483475 0.29060323 0.16284584 0.15231080 0.06381467
## time    0.08911979 0.26012807 0.42812524 0.18407354 0.13633457 0.12419372
## juice   0.25865124 0.18048231 0.19432254 0.14586409 0.62382810 0.07079738
## cake    0.22094376 0.20718213 0.02541748 0.01786550 0.26241495 0.40941146
## used    0.08937736 0.31156457 0.26029594 0.18876227 0.16085914 0.08815347
## 233     0.16636781 0.11822501 0.06257403 0.09598722 0.14319125 0.09064221
## other   0.05363520 0.16714889 0.17727345 0.18065394 0.11884382 0.04189038
## hour    0.35848182 0.09066566 0.24249563 0.19823430 0.14117701 0.36710133
## has     0.06076254 0.20704554 0.27615491 0.26233422 0.05867454 0.06369573
## through 0.08282664 0.12984730 0.26906998 0.38452609 0.17174951 0.05486750
## same    0.10679378 0.26258020 0.25666937 0.25642447 0.14731857 0.13936282
##                oven         mix       brown         their          can
## egg      0.164158930 0.37588378 0.300772262  0.040806239 0.065266050
## good     0.194269955 0.25199036 0.386126556  0.312983773 0.254834748
## before   0.335464971 0.20300736 0.175544073  0.140193032 0.290603230
## off      0.125527190 0.21113721 0.172850844  0.224712014 0.162845840
## lemon    0.123696290 0.34122597 0.228822050  0.026674005 0.152310802
## bake     0.701673851 0.36637567 0.432333795  0.037939925 0.063814672
## oven     1.000000000 0.17308070 0.388508765 -0.008274226 0.145934990
## mix      0.173080698 1.00000000 0.222805013  0.061426341 0.191472615
## brown    0.388508765 0.22280501 1.000000000  0.106262319 0.001608516
## their   -0.008274226 0.06142634 0.106262319  1.000000000 0.182876316
## can      0.145934990 0.19147261 0.001608516  0.182876316 1.000000000
## time     0.262063717 0.16848605 0.152906955  0.246172215 0.276436484
## juice    0.134574468 0.32449021 0.184868579  0.092008507 0.246118659
## cake     0.331253628 0.21393348 0.298978534 -0.062048538 0.099332652
## used     0.127980418 0.13610176 0.228257722  0.265177158 0.296069654
## 233      0.059055967 0.16378123 0.193726421  0.129862545 0.046400731
## other   -0.001354690 0.27134167 0.124502963  0.358174914 0.287673397
## hour     0.421395479 0.16464572 0.157028647  0.022930556 0.148854227
## has      0.085415324 0.12007649 0.083754402  0.232146325 0.325363691
## through  0.136651653 0.30794624 0.162959005  0.121044075 0.162345321
## same     0.148865088 0.25268178 0.084164344  0.224331832 0.279794600
##                time      juice        cake       used        233
## egg     0.08911979 0.25865124  0.22094376 0.08937736 0.16636781
## good    0.26012807 0.18048231  0.20718213 0.31156457 0.11822501
## before  0.42812524 0.19432254  0.02541748 0.26029594 0.06257403
## off     0.18407354 0.14586409  0.01786550 0.18876227 0.09598722
## lemon   0.13633457 0.62382810  0.26241495 0.16085914 0.14319125
## bake    0.12419372 0.07079738  0.40941146 0.08815347 0.09064221
## oven    0.26206372 0.13457447  0.33125363 0.12798042 0.05905597
## mix     0.16848605 0.32449021  0.21393348 0.13610176 0.16378123
## brown   0.15290695 0.18486858  0.29897853 0.22825772 0.19372642
## their   0.24617222 0.09200851 -0.06204854 0.26517716 0.12986254
## can     0.27643648 0.24611866  0.09933265 0.29606965 0.04640073
## time    1.00000000 0.17098724  0.13876615 0.21784042 0.05788285
## juice   0.17098724 1.00000000  0.19064189 0.16301698 0.13910752
## cake    0.13876615 0.19064189  1.00000000 0.12485060 0.19339974
## used    0.21784042 0.16301698  0.12485060 1.00000000 0.20374661
## 233     0.05788285 0.13910752  0.19339974 0.20374661 1.00000000
## other   0.21276090 0.21563209  0.06635242 0.38248120 0.11392501
## hour    0.33728790 0.17650572  0.11193935 0.09480775 0.07688582
## has     0.31206723 0.11311894  0.07623799 0.13586221 0.06378687
## through 0.27235496 0.32459318  0.03934868 0.09149176 0.13523815
## same    0.23213979 0.24919320  0.08435674 0.33380377 0.10799010
##                other       hour         has    through       same
## egg      0.05363520 0.35848182 0.06076254 0.08282664 0.10679378
## good     0.16714889 0.09066566 0.20704554 0.12984730 0.26258020
## before   0.17727345 0.24249563 0.27615491 0.26906998 0.25666937
## off      0.18065394 0.19823430 0.26233422 0.38452609 0.25642447
## lemon    0.11884382 0.14117701 0.05867454 0.17174951 0.14731857
## bake     0.04189038 0.36710133 0.06369573 0.05486750 0.13936282
## oven    -0.00135469 0.42139548 0.08541532 0.13665165 0.14886509
## mix      0.27134167 0.16464572 0.12007649 0.30794624 0.25268178
## brown    0.12450296 0.15702865 0.08375440 0.16295900 0.08416434
## their    0.35817491 0.02293056 0.23214633 0.12104408 0.22433183
## can      0.28767340 0.14885423 0.32536369 0.16234532 0.27979460
## time     0.21276090 0.33728790 0.31206723 0.27235496 0.23213979
## juice    0.21563209 0.17650572 0.11311894 0.32459318 0.24919320
## cake     0.06635242 0.11193935 0.07623799 0.03934868 0.08435674
## used     0.38248120 0.09480775 0.13586221 0.09149176 0.33380377
## 233      0.11392501 0.07688582 0.06378687 0.13523815 0.10799010
## other    1.00000000 0.02911262 0.20742853 0.15722241 0.37214188
## hour     0.02911262 1.00000000 0.08404000 0.13991566 0.05983771
## has      0.20742853 0.08404000 1.00000000 0.28031750 0.12407894
## through  0.15722241 0.13991566 0.28031750 1.00000000 0.06201306
## same     0.37214188 0.05983771 0.12407894 0.06201306 1.00000000
```

```r
# We can plot along mltiple dimensions:
tastes = c("egg","juice","cake")
common_similarities_tastes = model[1:3000,]%>% cosineSimilarity( model[[tastes,average=F]])
high_similarities_to_tastes = common_similarities_tastes[rank(-apply(common_similarities_tastes,1,max))

# - Plotting
high_similarities_to_tastes %>%
  as_tibble(rownames='word') %>%
  filter( ! (is.element(word,tastes))) %>%
  #mutate(total = salty+sweet+savory+bitter+sour) %>%
  #mutate( sweet=sweet/total,salty=salty/total,savory=savory/total,bitter=bitter/total, sour = sour/tot
  #select(-total) %>%
  gather(key = 'key', value = 'value',-word) %>%
  ggplot(aes(x = word,
             y = value,
             fill = key)) + geom_bar(stat='identity') +
  coord_flip() + theme_minimal() + scale_fill_brewer(palette='Spectral')
```

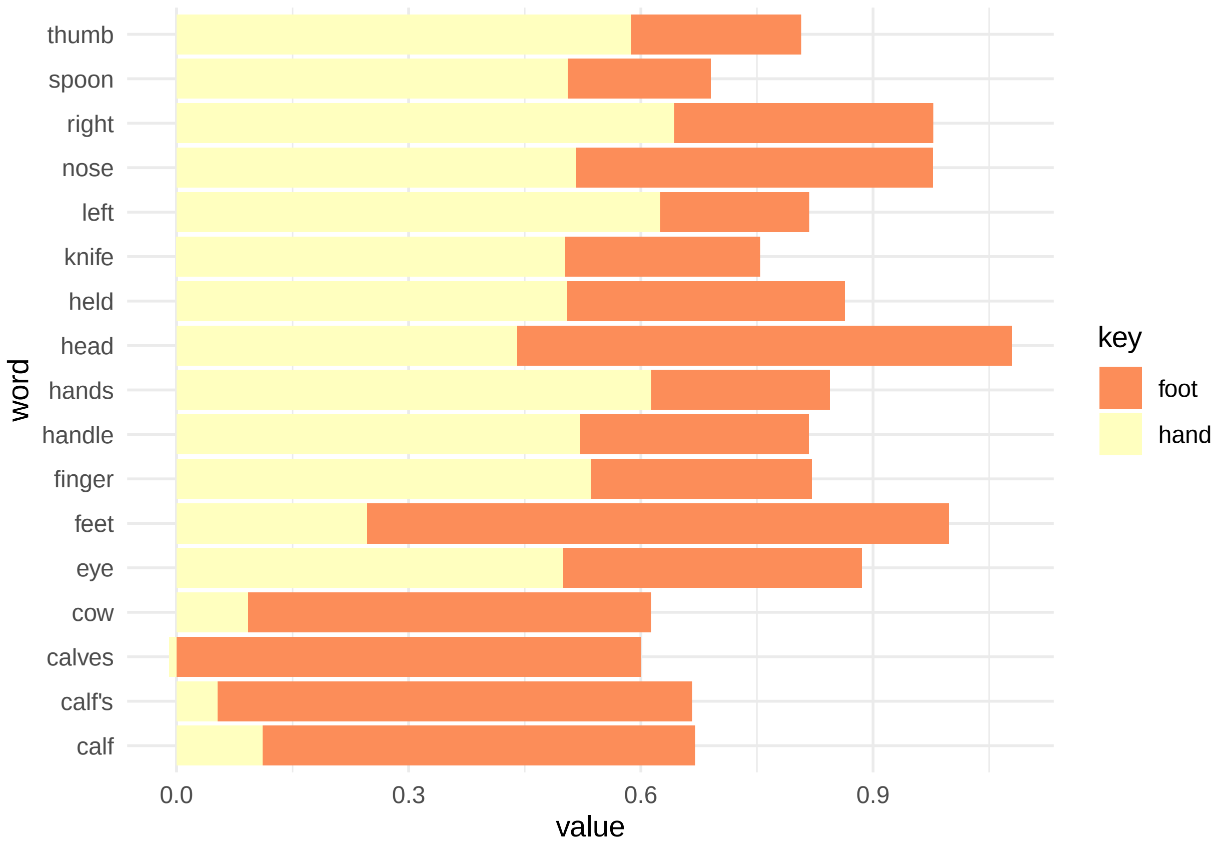


5.) Try different combinations for analogic reasoning. (Given the data is not munged, I expect that some relations will be odd)

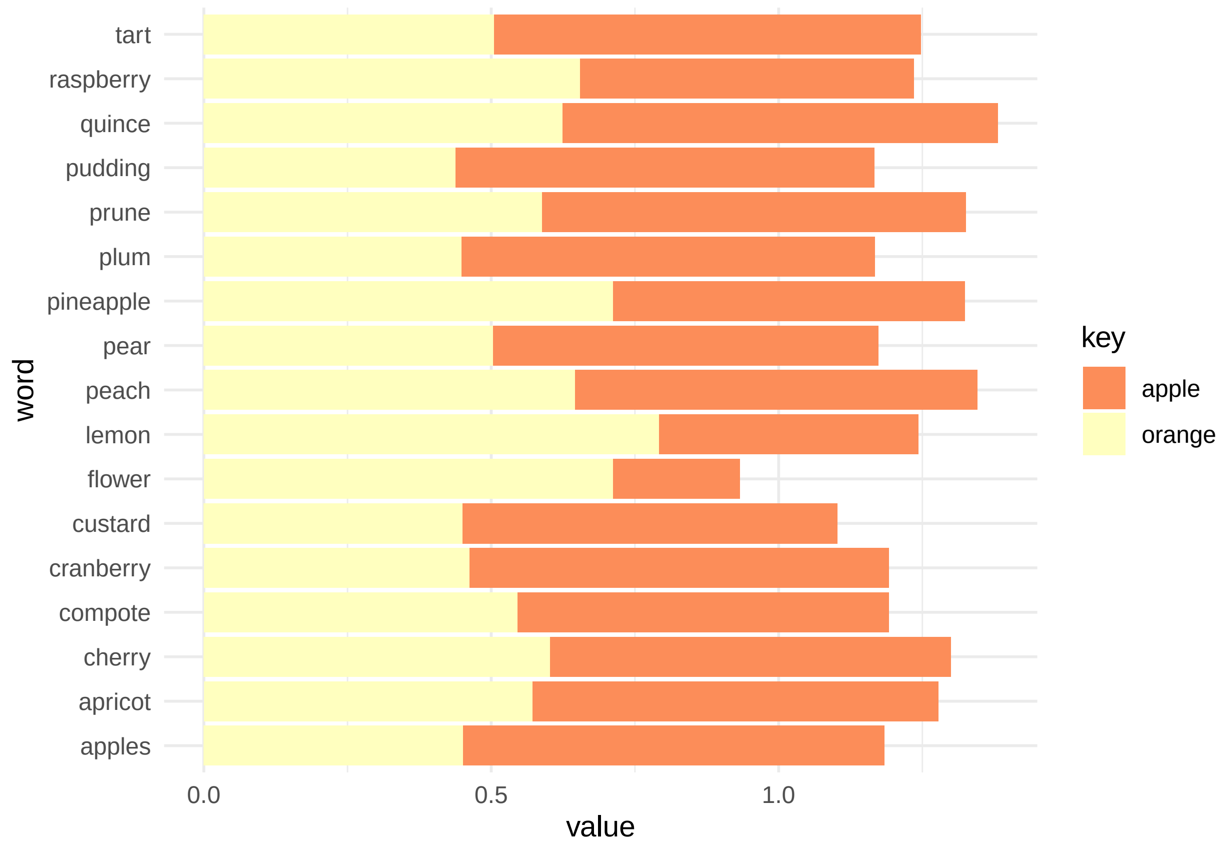
```
# We can plot along mltiple dimensions:
tastes = c("cow","sheep")
common_similarities_tastes = model[1:3000,]%>% cosineSimilarity( model[[tastes,average=F]])
high_similarities_to_tastes = common_similarities_tastes[rank(-apply(common_similarities_tastes,1,max))

# - Plotting
high_similarities_to_tastes %>%
  as_tibble(rownames='word') %>%
  filter( ! (is.element(word,tastes))) %>%
  #mutate(total = salty+sweet+savory+bitter+sour) %>%
  #mutate( sweet=sweet/total,salty=salty/total,savory=savory/total,bitter=bitter/total, sour = sour/tot
  #select(-total) %>%
  gather(key = 'key', value = 'value',-word) %>%
  ggplot(aes(x = word,
             y = value,
             fill = key)) + geom_bar(stat='identity') +
  coord_flip() + theme_minimal() + scale_fill_brewer(palette='Spectral')
```



```
# We can plot along mltiple dimensions:
tastes = c("hand","foot")
common_similarities_tastes = model[1:3000,]%>% cosineSimilarity( model[[tastes,average=F]])
high_similarities_to_tastes = common_similarities_tastes[rank(-apply(common_similarities_tastes,1,max))

# - Plotting
high_similarities_to_tastes %>%
  as_tibble(rownames='word') %>%
  filter( ! (is.element(word,tastes))) %>%
  #mutate(total = salty+sweet+savory+bitter+sour) %>%
  #mutate( sweet=sweet/total,salty=salty/total,savory=savory/total,bitter=bitter/total, sour = sour/tot
  #select(-total) %>%
  gather(key = 'key', value = 'value',-word) %>%
  ggplot(aes(x = word,
             y = value,
             fill = key)) + geom_bar(stat='identity') +
  coord_flip() + theme_minimal() + scale_fill_brewer(palette='Spectral')
```



```
# We can plot along mltiple dimensions:
tastes = c("apple","orange")
common_similarities_tastes = model[1:3000,]%>% cosineSimilarity( model[[tastes,average=F]])
high_similarities_to_tastes = common_similarities_tastes[rank(-apply(common_similarities_tastes,1,max))

# - Plotting
high_similarities_to_tastes %>%
  as_tibble(rownames='word') %>%
  filter( ! (is.element(word,tastes))) %>%
  #mutate(total = salty+sweet+savory+bitter+sour) %>%
  #mutate( sweet=sweet/total,salty=salty/total,savory=savory/total,bitter=bitter/total, sour = sour/tot
  #select(-total) %>%
  gather(key = 'key', value = 'value',-word) %>%
  ggplot(aes(x = word,
             y = value,
             fill = key)) + geom_bar(stat='identity') +
  coord_flip() + theme_minimal() + scale_fill_brewer(palette='Spectral')
```



6.) Given the set of analogic words and the values that are close to them, what are some interesting relations you find? List 2-3 of them.

Whey is more closely associated with cow than sheep, which makes sense bcause of whey protein and whey milk. Pudding is closer to apple than orange is weird because there is no apple pudding or orange pudding. Lastly, flower is more associated to orange than apple, which makes no sense and should be closely orthogonal.

7.) The starter code sets bundle_ngram=1, which means we are interested in single word embeddings and not common pairs, if we set bundle_ngrams=2 what common pairs of words do you get - list 10.

```
# -- Check to see  if file exists --
if (!file.exists("cookbooks.zip")) {
  download.file("http://archive.lib.msu.edu/dinfo/feedingamerica/cookbook_text.zip","cookbooks.zip")
}
unzip("cookbooks.zip",exdir="cookbooks")
if (!file.exists("cookbooks2.txt")) prep_word2vec(origin="cookbooks",destination="cookbooks2.txt",lower
```

```
# Training a Word2Vec model
if (!file.exists("cookbook_vectors2.bin")) {
  model = train_word2vec("cookbooks2.txt","cookbook_vectors2.bin",
                         vectors=100,threads=4,window=6,
                         min_count = 10,
                         iter=5,negative_samples=15)
} else{
    model = read.vectors("cookbook_vectors2.bin")
    }
```

```
## Filename ends with .bin, so reading in binary format

## Reading a word2vec binary file of 25375 rows and 100 columns

##
  |
  |                                                                      |   0%
  |
  |                                                                      |   1%
  |
  |=                                                                     |   1%
  |
  |=                                                                     |   2%
  |
  |==                                                                    |   2%
  |
  |==                                                                    |   3%
  |
  |==                                                                    |   4%
  |
  |===                                                                   |   4%
  |
  |===                                                                   |   5%
  |
  |====                                                                  |   5%
  |
  |====                                                                  |   6%
  |
  |====                                                                  |   7%
  |
  |=====                                                                 |   7%
  |
  |=====                                                                 |   8%
  |
  |======                                                                |   8%
  |
  |======                                                                |   9%
  |
  |======                                                                |  10%
  |
  |=======                                                               |  10%
  |
  |=======                                                               |  11%
  |
  |=======                                                               |  12%
  |
  |========                                                              |  12%
  |
  |========                                                              |  13%
  |
  |=========                                                             |  13%
  |
|=========                                                         |  14%
|
|=========                                                         |  15%
|
|==========                                                        |  15%
|
|==========                                                        |  16%
|
|===========                                                       |  16%
|
|===========                                                       |  17%
|
|===========                                                       |  18%
|
|============                                                      |  18%
|
|============                                                      |  19%
|
|=============                                                     |  19%
|
|=============                                                     |  20%
|
|=============                                                     |  21%
|
|==============                                                    |  21%
|
|==============                                                    |  22%
|
|===============                                                   |  22%
|
|===============                                                   |  23%
|
|===============                                                   |  24%
|
|================                                                  |  24%
|
|================                                                  |  25%
|
|=================                                                 |  25%
|
|=================                                                 |  26%
|
|=================                                                 |  27%
|
|==================                                                |  27%
|
|==================                                                |  28%
|
|===================                                               |  28%
|
|===================                                               |  29%
|
|===================                                               |  30%
|
|====================                                             |  30%
|
|====================                                             |  31%
|
|====================                                             |  32%
|
|=====================                                            |  32%
|
|=====================                                            |  33%
|
|======================                                           |  33%
|
|======================                                           |  34%
|
|======================                                           |  35%
|
|=======================                                          |  35%
|
|=======================                                          |  36%
|
|========================                                         |  36%
|
|========================                                         |  37%
|
|========================                                         |  38%
|
|=========================                                        |  38%
|
|=========================                                        |  39%
|
|==========================                                       |  39%
|
|==========================                                       |  40%
|
|==========================                                       |  41%
|
|===========================                                      |  41%
|
|===========================                                      |  42%
|
|============================                                     |  42%
|
|============================                                     |  43%
|
|============================                                     |  44%
|
|=============================                                    |  44%
|
|=============================                                    |  45%
|
|==============================                                   |  45%
|
|==============================                                   |  46%
|
|==============================                                   |  47%
|
|===============================                                  |  47%
|
|===============================                                  |  48%
|
|================================                                 |  48%
|
|================================                                 |  49%
|
|================================                                 |  50%
|
|=================================                                |  50%
|
|=================================                                |  51%
|
|=================================                                |  52%
|
|==================================                               |  52%
|
|==================================                               |  53%
|
|===================================                              |  53%
|
|===================================                              |  54%
|
|===================================                              |  55%
|
|====================================                             |  55%
|
|====================================                             |  56%
|
|=====================================                            |  56%
|
|=====================================                            |  57%
|
|=====================================                            |  58%
|
|======================================                           |  58%
|
|======================================                           |  59%
|
|=======================================                          |  59%
|
|=======================================                          |  60%
|
|=======================================                          |  61%
|
|========================================                         |  61%
|
|========================================                         |  62%
|
|=========================================                        |  62%
|
|=========================================                        |  63%
|
|=========================================                        |  64%
|
|==========================================                       |  64%
|
|==========================================                       |  65%
|
|===========================================                      |  65%
|
|===========================================                      |  66%
|
|===========================================                      |  67%
|
|============================================                     |  67%
|
|============================================                     |  68%
|
|=============================================                    |  68%
|
|=============================================                    |  69%
|
|=============================================                    |  70%
|
|==============================================                   |  70%
|
|==============================================                   |  71%
|
|==============================================                   |  72%
|
|===============================================                  |  72%
|
|===============================================                  |  73%
|
|================================================                 |  73%
|
|================================================                 |  74%
|
|================================================                 |  75%
|
|=================================================                |  75%
|
|=================================================                |  76%
|
|==================================================               |  76%
|
|==================================================               |  77%
|
|==================================================               |  78%
|
|===================================================              |  78%
|
|===================================================              |  79%
|
|===================================================              |  79%
|
|===================================================              |  80%
|
|===================================================              |  81%
|
|====================================================             |  81%
|
|====================================================             |  82%
|
|=====================================================            |  82%
|
|=====================================================            |  83%
|
|=====================================================            |  84%
|
|======================================================           |  84%
|
|======================================================           |  85%
|
|=======================================================          |  85%
|
|=======================================================          |  86%
|
|=======================================================          |  87%
|
|========================================================         |  87%
|
|========================================================         |  88%
|
|=========================================================        |  88%
|
|=========================================================        |  89%
|
|=========================================================        |  90%
|
|==========================================================       |  90%
|
|==========================================================       |  91%
|
|==========================================================       |  92%
|
|===========================================================      |  92%
|
|===========================================================      |  93%
|
|============================================================     |  93%
|
|============================================================     |  94%
|
|============================================================     |  95%
|
|=============================================================    |  95%
|
|==============================================================   |  96%
|
|===============================================================  |  96%
|
|===============================================================  |  97%
|
|===============================================================  |  98%
|
|================================================================ |  98%
|
|================================================================ |  99%
|
|=================================================================|  99%
|
|=================================================================| 100%
```

Question 4 Gaussian Processes

Part 1

1.) Fit a model using the radial basis function to the data kernel_regression_1.csv

```r
library(MASS)
```

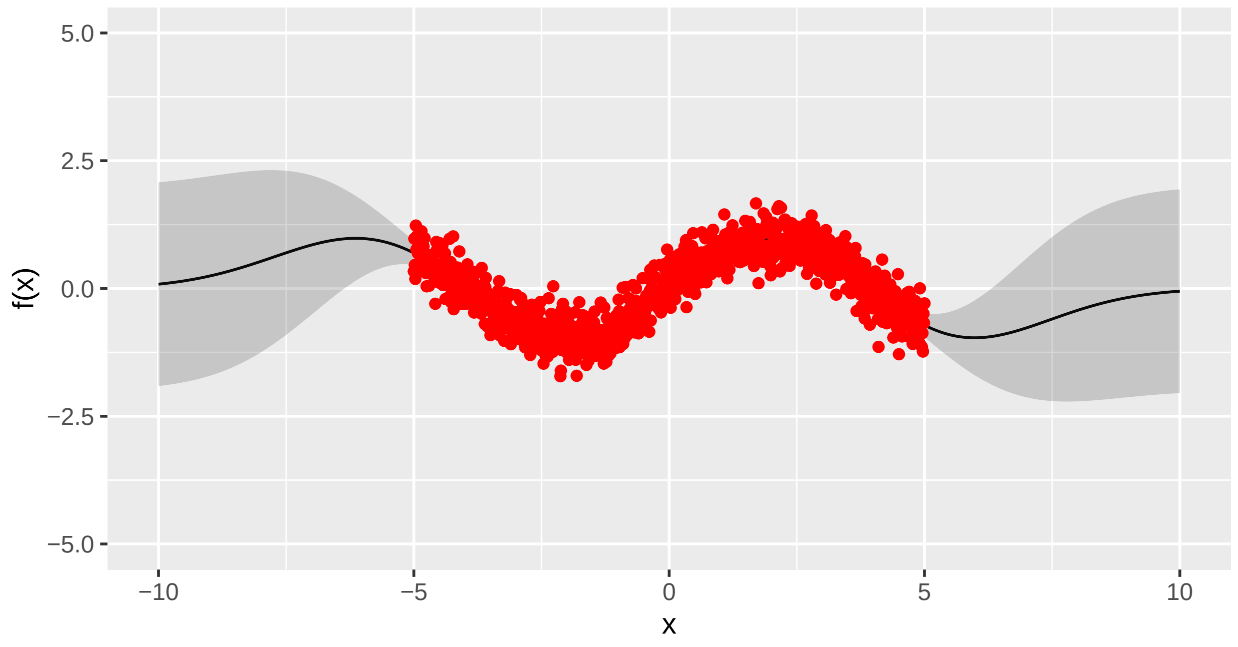
```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(ggplot2)
library(dplyr)
library(tidyr)
```

```r
data <- read.csv("kernel_regression_1.csv")
```

```r
# Fitting a 0 mean gaussian process.
# Kernel matrix
# Radial basis function.
K = function(x,x_prime,l){
  d = sapply(x, FUN = function(x_in)(x_in - x_prime)^2)
  return(t(exp(-1/(2*l^2) *d))) # t() is transpose function.
}

sampling_from_a_gp <- function(theta=1){
  # Generating Data
  set.seed(12345)
  n = 500
  x_observed = data$x  # Training data.
  x_prime = seq(-10, 10,length.out = n)  # vector of new points.
  f = data$y

  # Setting up GP
  mu = mean(f) # note that mean(f) approximately 0.
  mu_star = 0

  # Covariance of f
  K_f = K(x_observed,x_observed,theta) + diag(var(f), length(x_observed))

  # Marginal and conditional covariance of f_star|f (posterior predictive)
  K_star = K(x_observed,x_prime,theta)
  K_starstar = K(x_prime,x_prime,theta)

  # Conditional distribution  f_star|f (Posterior predictive)
  # Page 77 in lecture notes, x = f-mu. It isn't stated, but this is what it represents.
  # $ E[f_{star}|f] = K_{star}^T * K^{-1} * (f-mu)
  mu_star = mu_star + t(K_star) %*% solve(K_f) %*% (f - mu)
  Sigma_star = K_starstar - t(K_star)%*% t(solve(K_f)) %*% K_star

  # Re-arranging values for plotting
  plot_gp = tibble(x = x_prime,
                   y = mu_star %>% as.vector(),  # This is the mean surface function.
                   sd_prime = sqrt(diag(Sigma_star)))

  # Calculate negative log likelihood
  likelihood = (-1/2) * f %*% solve(K_f) %*% f - (1/2) * log(det(K_f)) - (length(x_observed)/2) * log(2

  # Plotting values
  # The plot in lecture notes does not match this plot. There is a mistake in the lecture notes.
  # The shaded region is 1 standard deviation of f(x), above and below.
  # Ploting the mean surface function as a function of x.
  p=ggplot(aes(x = x, y = y), data = plot_gp) +
    geom_line()+
    geom_ribbon(aes(ymin = y - (2 * sd_prime),ymax = y + (2 * sd_prime)), alpha = 0.2)+
    geom_point(aes(x =x , y= y), data = tibble(x = x_observed, y = f),
               color = 'red') + # red is the actual data points.
    xlim(c(-10,10))+ylim(c(-5,5))+
    coord_fixed(ratio = 1) +ylab('f(x)')
  return(list('likelihood' = likelihood, 'plot' = p))
}
```

```r
sampling_from_a_gp(theta=2)$p
```

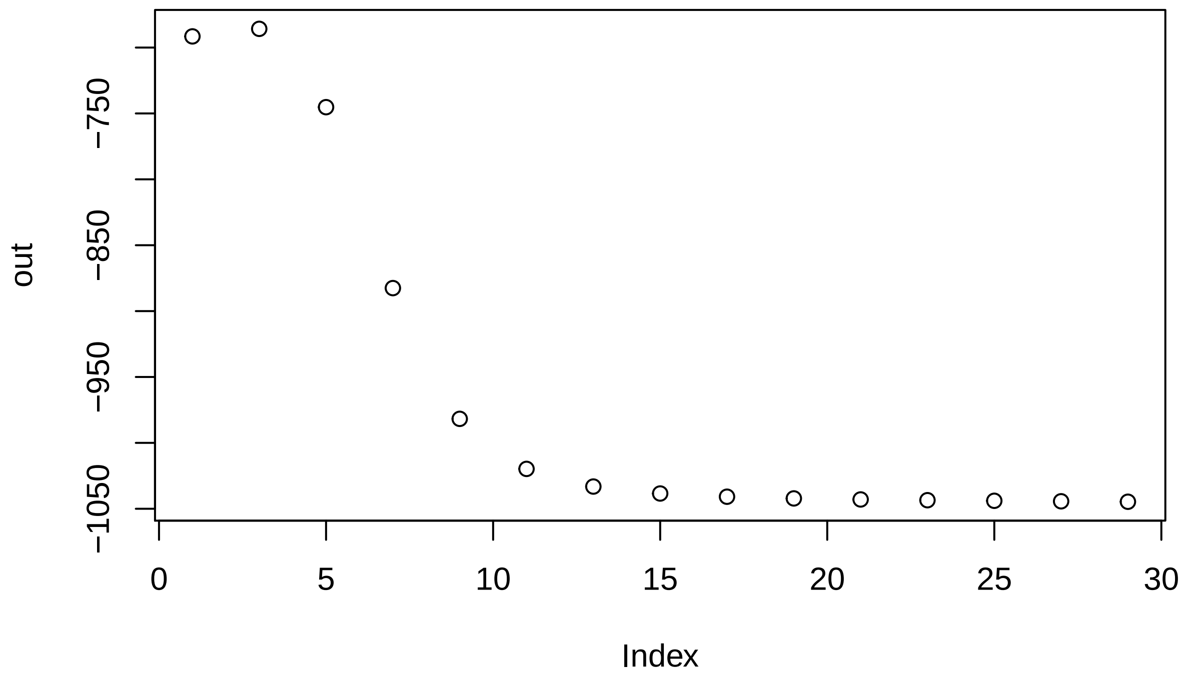


2.) Find the optimal value of the hyperparameter θ by plotting the negative log likelihood and selecting the value that minimizes the negative log likelihood.

Around 15 is a good number because the performance starts to plateau.

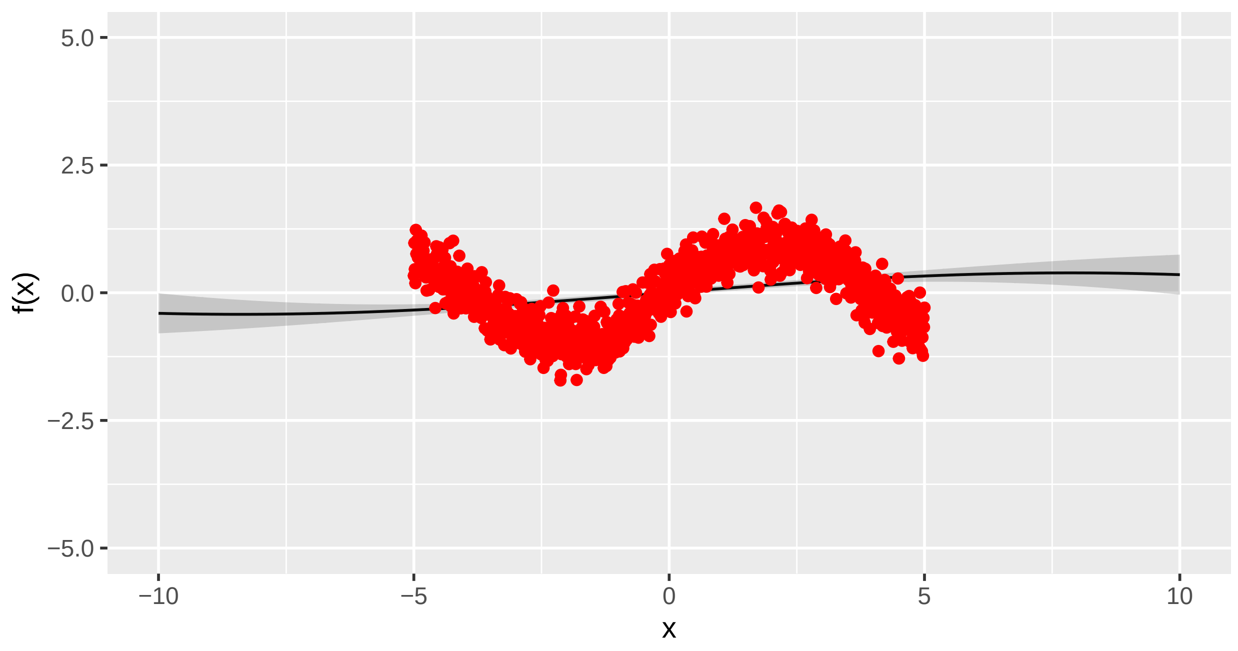
```
t <- seq(1, 30, 2)
out <- c()
for(i in t){
  a <- sampling_from_a_gp(theta=i)
  out[i] <- a$likelihood

}
```

```
plot(out)
```



3.) Plot the mean function with a 95% confidence interval.

```
sampling_from_a_gp(theta=15)$p
```



Part 2

It is possible to use Gaussian Processes in time series analysis. We do that by combining several kernels that reflect global and local periodicity along iwth macro trends. Reference the kernels in assignment pdf.

1.) Construct a GP with $x \in R$ and the covariance below:

$$Cov(x, x') = k_{Per}(x, x') + k_{LocalPer}(x, x') + k_{Lin}(x, x')$$

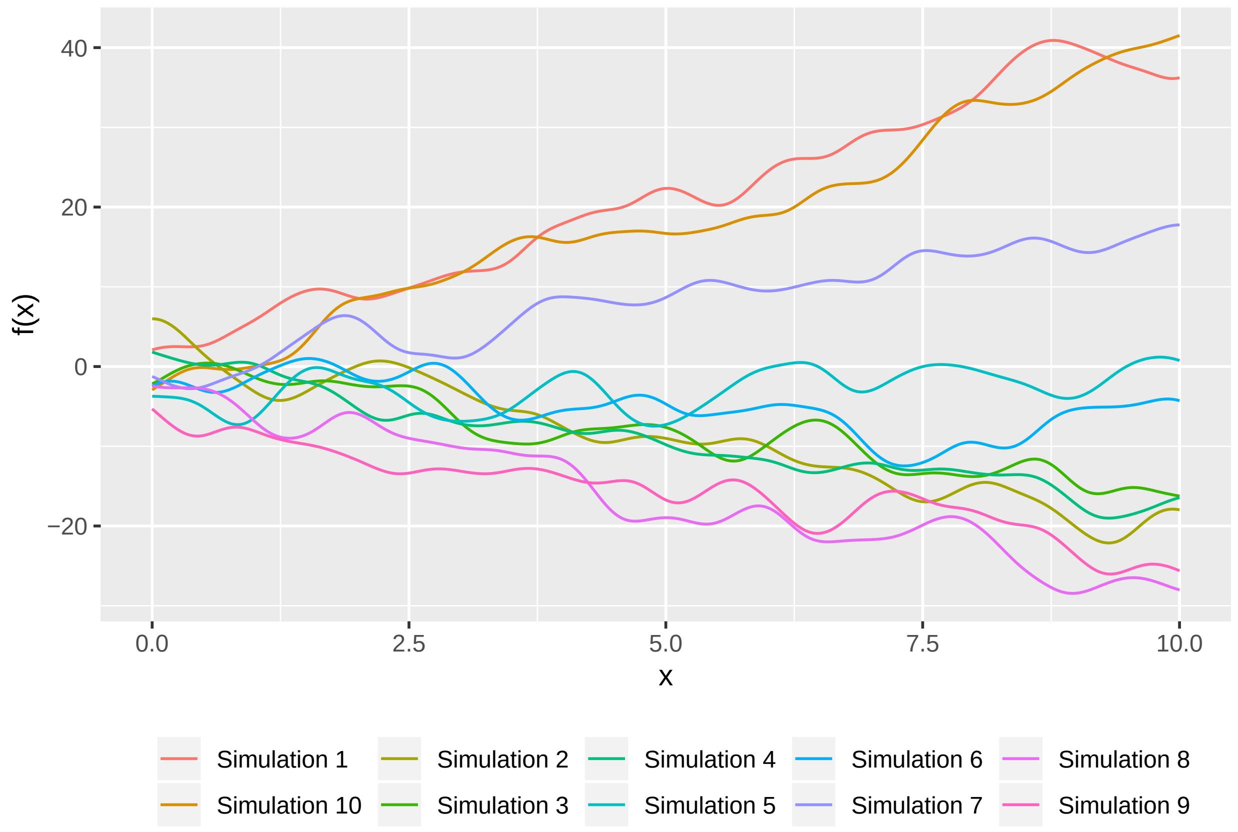
```
k_per = function(x, x_prime){
  p = 2
  l = 2
  var = 4
  return(sapply(x, FUN = function(x_in)(var * exp((-2/(l^2)) * (sin(pi * abs(x_in - x_prime)/p))^2))))
}

k_localper = function(x, x_prime){
  p = 2
  l = 2
  var = 4
  return(sapply(x, FUN = function(x_in)(var * exp((-2/(l^2)) * (sin(pi * abs(x_in - x_prime)/p))^2) * e
}
```

```r
k_lin = function(x, x_prime){
  var_b = 4
  var_c = 3
  c = .5
  return(sapply(x, FUN = function(x_in)(var_b + var_c*(x_in - .5)*(x_prime - .5))))
}

sampling_from_a_gp = function(x_min = 0,
                              x_max=1,
                              kernel_in,
                              n = 50,
                              n_gps = 10){
  # n_gps is the number of gaussian processes to simulate
  # n is the number of observed points in each gaussian process.
  # Simulation
  x = seq(x_min, x_max,length.out = n)
  K = k_localper(x, x) + k_per(x, x) + k_lin(x, x)
  L = chol(K + 1e-6*diag(n))
  # $ Y ~ \mu + L * N(0,1)
  # matrix(rnorm(n*n_gps), ncol = n_gps) is creating a multivariate N(0,I) with 0 covariance
  f_prior = t(L) %*% matrix(rnorm(n*n_gps), ncol = n_gps)

  # Reshaping
  colnames(f_prior) = paste0('Simulation ', seq(1:n_gps))
  f_prior_long_format = f_prior %>% as_tibble() %>%
    bind_cols(x = x) %>%
    pivot_longer(cols = starts_with("sim"))

  # Plot
  ggplot(aes(x = x, color = name, y = value),
              data = f_prior_long_format) +
    geom_line()+theme(legend.position = 'bottom')+
    guides(color=guide_legend(title=""))+
    ylab('f(x)')
}
```

2.) Generate and plot 10 simulations over the domain

$$x \in (0, 10)$$

```r
sampling_from_a_gp(x_min=0, x_max=10, n_gps = 10, n = 1000)
```



3.) Simulate a set of points with local and global periodicity and fit your model to the data.

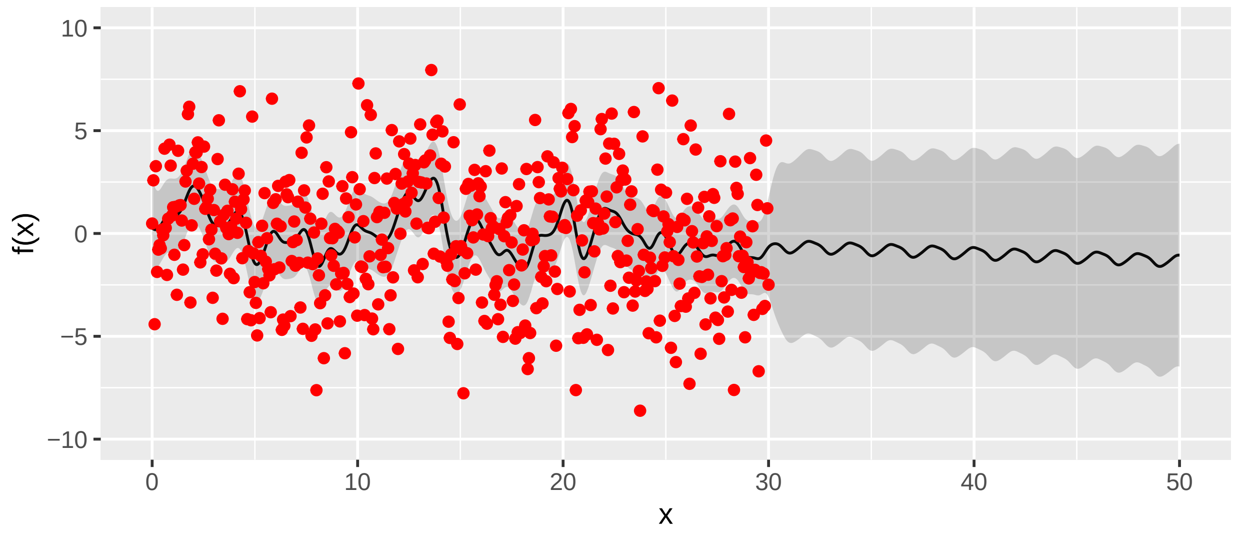
Did not optimize the hyperparameters.

```r
x <- seq(0, 30, length.out=500)
# frequency of 1/10 and frequency of 5
y <- sin(2*pi*x/10) + sin(2*pi*x*5) + rnorm(length(x), 0, 3)
```

```r
# Fitting a 0 mean gaussian process.
sampling_from_a_gp <- function(theta=1){
  # Generating Data
  set.seed(12345)
  n = 1000
  x_observed = x  # Training data.
  x_prime = seq(0, 50,length.out = n)  # vector of new points.
  f = y

  # Setting up GP
  mu = mean(f) # note that mean(f) approximately 0.
  mu_star = 0

  # Covariance of f
  K_f = k_localper(x_observed, x_observed) + k_per(x_observed, x_observed) + k_lin(x_observed, x_observe

  # Marginal and conditional covariance of f_star|f (posterior predictive)
  K_star = t(k_localper(x_observed, x_prime) + k_per(x_observed, x_prime) + k_lin(x_observed, x_prime))
  K_starstar = k_localper(x_prime, x_prime) + k_per(x_prime, x_prime) + k_lin(x_prime, x_prime)

  # Conditional distribution  f_star|f (Posterior predictive)
  # Page 77 in lecture notes, x = f-mu. It isn't stated, but this is what it represents.
  # $ E[f_{star}|f] = K_{star}^T * K^{-1} * (f-mu)
  mu_star = mu_star + t(K_star) %*% solve(K_f) %*% (f - mu)
  Sigma_star = K_starstar - t(K_star)%*% t(solve(K_f)) %*% K_star

  # Re-arranging values for plotting
  plot_gp = tibble(x = x_prime,
                   y = mu_star %>% as.vector(),  # This is the mean surface function.
                   sd_prime = sqrt(diag(Sigma_star)))

  # Calculate negative log likelihood
  likelihood = (-1/2) * f %*% solve(K_f) %*% f - (1/2) * log(det(K_f)) - (length(x_observed)/2) * log(2

  # Plotting values
  # The plot in lecture notes does not match this plot. There is a mistake in the lecture notes.
  # The shaded region is 1 standard deviation of f(x), above and below.
  # Ploting the mean surface function as a function of x.
  ggplot(aes(x = x, y = y), data = plot_gp) +
    geom_line()+
    geom_ribbon(aes(ymin = y - (2 * sd_prime),ymax = y + (2 * sd_prime)), alpha = 0.2)+
    geom_point(aes(x =x , y= y), data = tibble(x = x_observed, y = f),
               color = 'red') + # red is the actual data points.
    xlim(c(0,50))+ylim(c(-10,10))+
    coord_fixed(ratio = 1) +ylab('f(x)')
}
```

```
sampling_from_a_gp()
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



10
5
0
−5
−10
f(x)
0
10
20
30
40
50
x