Homework 3

Jack Hart

Question 1: Boosting

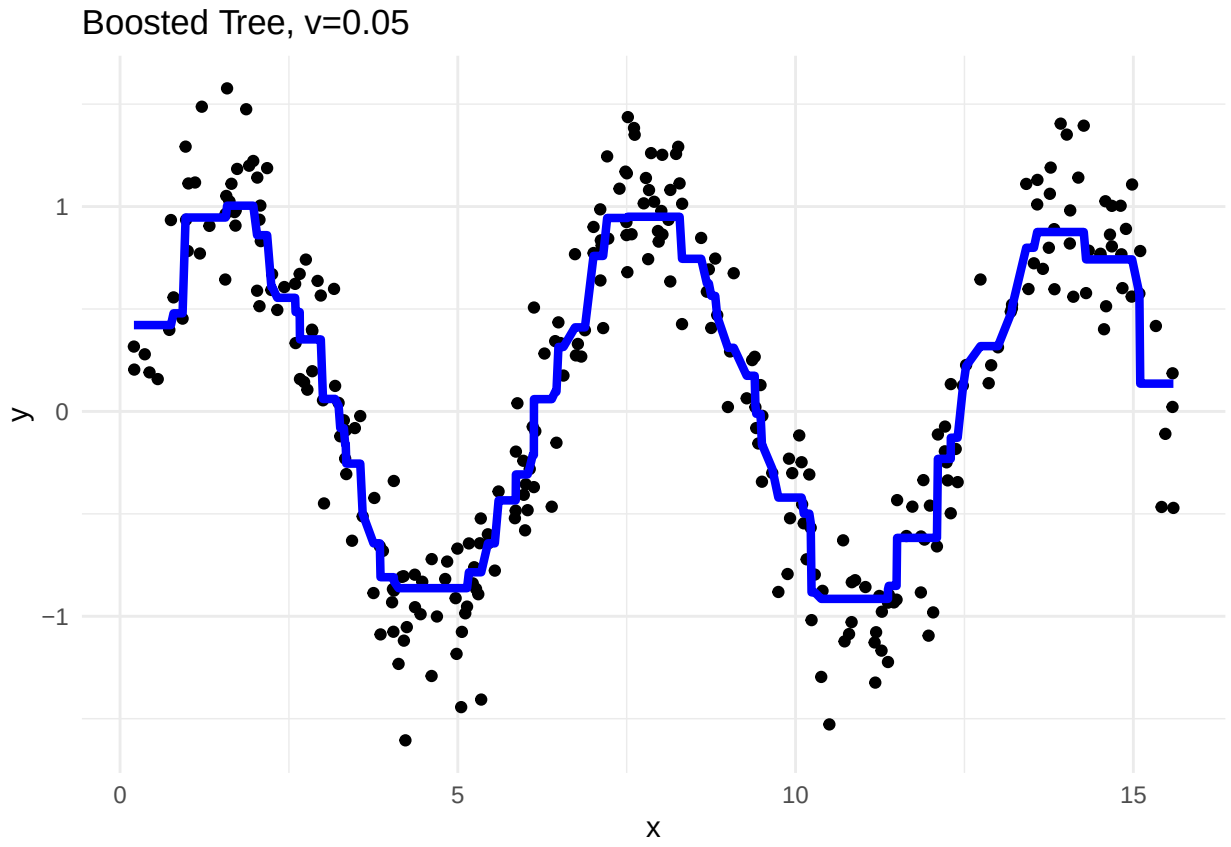
Part 1: Building A Gradient Boosted Tree

```
fit_boosted_tree <- function(df, v = 0.05, number_of_weak_learners = 100){

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

  for(t in 2:number_of_weak_learners){
    # Fit linear spline
    fit = fit=rpart(yr~.-y,data=df)

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

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

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

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

  return(list("weak_learners" = list_of_weak_learners, "YP" =  YP,
              "v" = 0.05, "number_of_weak_learners" = number_of_weak_learners) )
}

predict_boosted_tree <- function(trained_learners, new_data){
  for (i in 1:trained_learners$number_of_weak_learners){
  weak_learner_i = trained_learners$weak_learners[[i]]

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

  if(i==trained_learners$number_of_weak_learners){
    new_data = new_data %>% bind_cols(yp=pred)
    }
  }
  return(new_data %>% dplyr::select(yp))
}
```

Q0

The following code uses the functions above to build a gradient boosted tree with v=0.05. The original data and the resulting prediction are plotted.

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

# fit and plot predictions.
fit_results <- fit_boosted_tree(df, v = 0.05, number_of_weak_learners = 100)
df$ypred <- predict_boosted_tree(fit_results, df)$yp
```

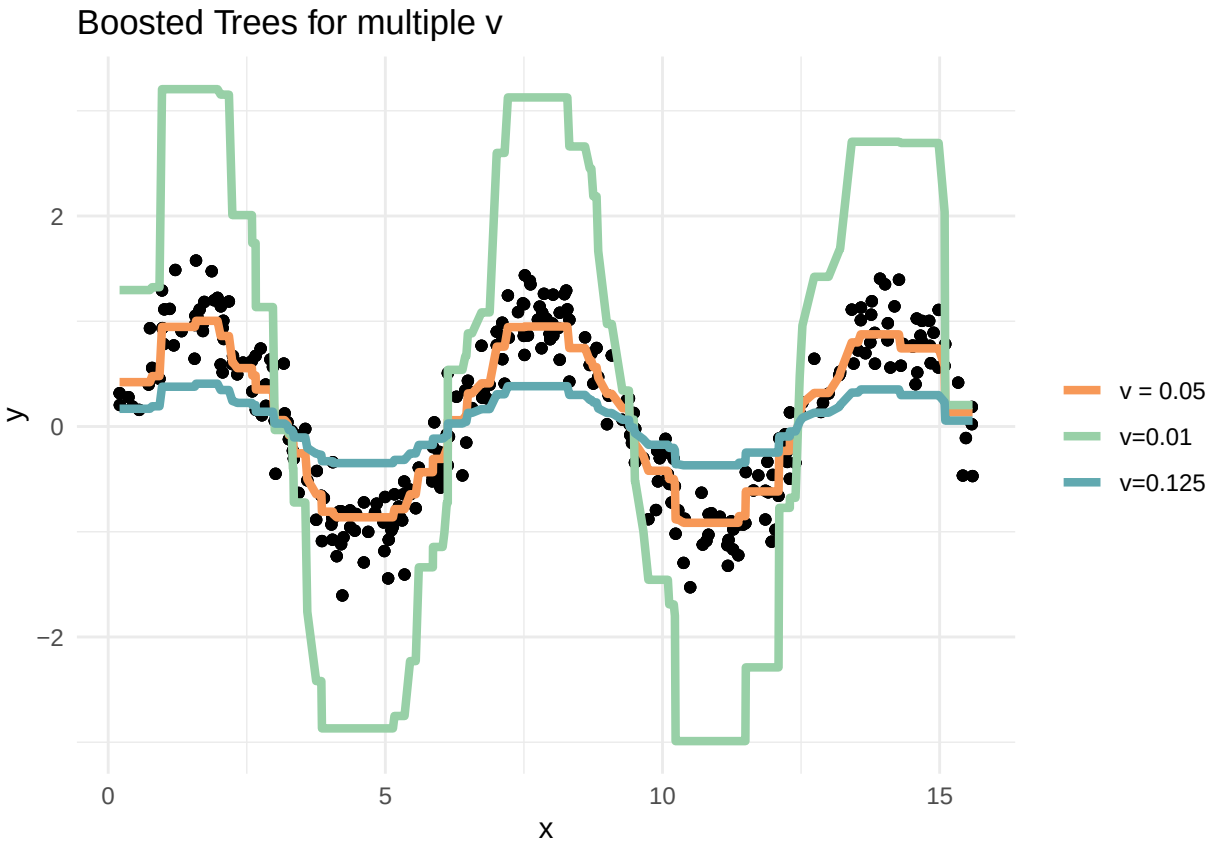


Q1

The following code and plot show what happens when we vary the learning parameter v. **A smaller v parameter results in greater variance (e.g. 0.01), while a larger v value increases bias (e.g. 0.125).** This is because v tunes how much of the previous prediction we will take off the current y value.

```
df_orig <- df %>% dplyr::select(x, y)
v_vals=c(0.01,0.125)

for(v in v_vals){
```

```
  fit_results <- fit_boosted_tree(df_orig, v = v, number_of_weak_learners = 100)
  df[,paste0("ypred", v)] <- predict_boosted_tree(fit_results, df_orig)$yp
}
```



Q2

A

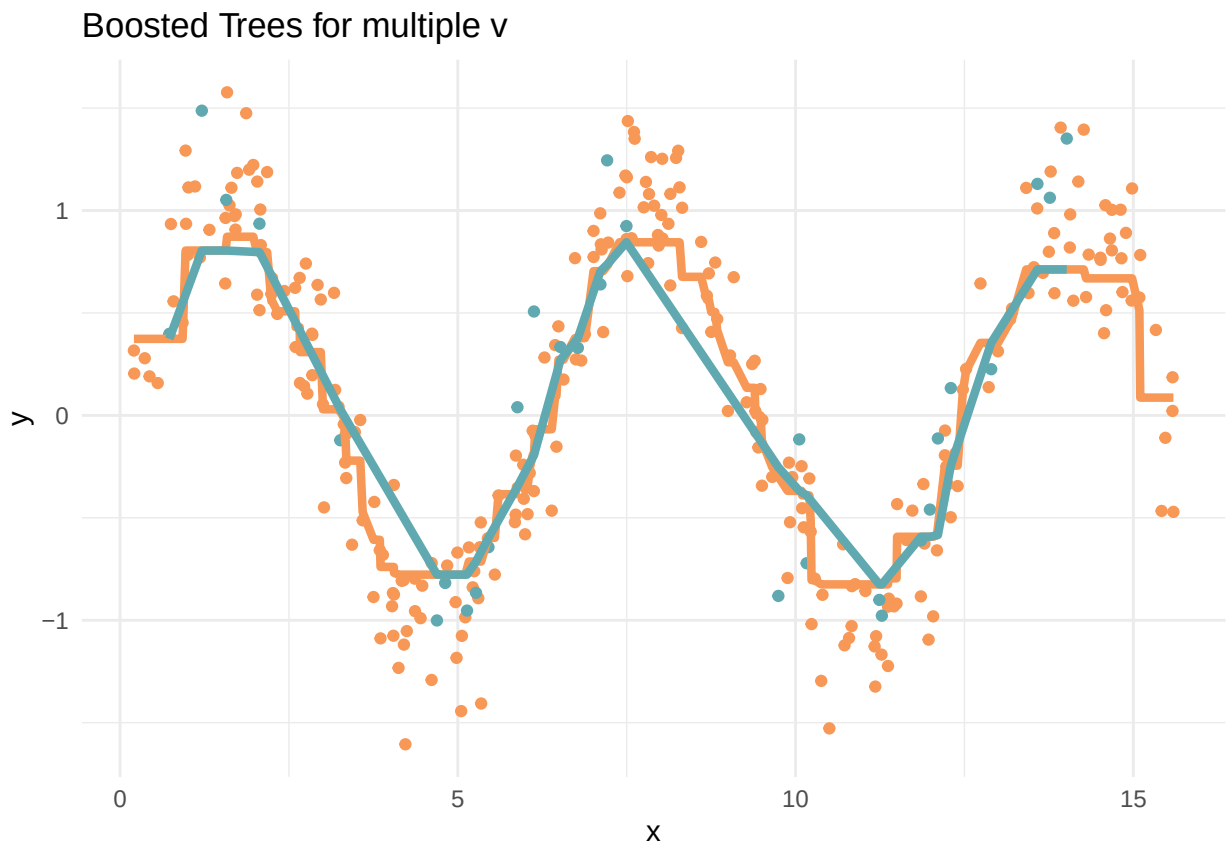
The following code adjusts the previous functions to now stop training once the mean of the "loss"/residuals is smaller than some number. This is comparative to early stopping. This will assure that if the residuals have reached bellow a certain level, we arn't adding any more trees that training on useless data. I implement this heuristic in the fitting method by adding a `min_residual` parameter.

```
fit_boosted_tree_heuristic <- function(df, v = 0.05, max_number_of_weak_learners = 100, min_residual = 0

  # Fit round 1
  fit=rpart(y~.,data=df, control = control_val)

  yp = predict(fit, newdata=df)
  df$yr = df$y - v*yp
  YP = v*yp
  list_of_weak_learners = list(fit)

  for(t in 2:max_number_of_weak_learners){
```

```
    fit = fit=rpart(yr~.-y,data=df, control = control_val)

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

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

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

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

    if(mean(abs(df$yr)) <= min_residual){
      return(list("weak_learners" = list_of_weak_learners, "YP" =  YP,
          "v" = 0.05, "number_of_weak_learners" = length(list_of_weak_learners)) )
      }
    }

  return(list("weak_learners" = list_of_weak_learners, "YP" =  YP,
              "v" = 0.05, "number_of_weak_learners" = max_number_of_weak_learners) )
}
```

Next, the following code splits the data into training and testing sets and applies the heuristic training (*stopping learning after residuals are 0.2*). The models are plotted. The training data was plotted in orange, and the test data in blue. As you can see, this model with a heuristic approach trained similarly to the original model, and we can see from the test data that it generalizes pretty well.

```
# split into validation and test sets 90-10
set.seed(52)
idx_val <- sample(1:nrow(df_orig),270)
validation_df <- df_orig[idx_val,]
test_df <- df_orig[-idx_val,]

# fit tree with heuristic approach
fit_results <- fit_boosted_tree_heuristic(validation_df, v = 0.05, max_number_of_weak_learners = 100, m
validation_df$pred <- predict_boosted_tree(fit_results, validation_df)$yp
test_df$pred <- predict_boosted_tree(fit_results, test_df)$yp
```



B

The function I created returns the number of learners trained. In this case, **it trained 36 trees**, and got similar results as the original model trained with 100.

```
fit_results$number_of_weak_learners
```

```
## [1] 36
```

C

The RMSE on the test set was **0.3314776**, which is still larger than the training RMSE. This is probably due to the much smaller sample (30 vs 270), but also may be indicative that we could increase the maximum residual and decreaes the fexibility of the model (number of trees) even more.

```
# Training RMSE
sqrt(mean((validation_df$y - validation_df$pred)^2))
```

```
## [1] 0.2601418
```

```
# Test RMSE
sqrt(mean((test_df$y - test_df$pred)^2))
```

```
## [1] 0.3314776
```

Q3

According to this grid search, it appears **max_depth doesn't impact the results very much**. However, a cp (complexity param) of 0.001 with a minsplit of two resulted in the best training RMSE rates. When using the test set, however, **A cp of 0.1** was found to be best. Lastly, it appeared having a small number of minimum split values, **2 in each node**, was good for the training and test sets.

```
# values to search on
minsplit_vals = c(2,5,10,20,30)
cp_vals = c(0.001, 0.005, 0.01, 0.05, 0.1)
maxdepth_vals = c(2,10,20,30)

#gridsearch
grid_predictions <- data.frame()
for(minsplit in minsplit_vals){
  for(cp in cp_vals){
      for(maxdepth in maxdepth_vals){
        control <- rpart.control(minsplit = minsplit,  cp = cp, maxdepth = maxdepth)
        fit_results <- fit_boosted_tree_heuristic(validation_df, v = 0.05, max_number_of_weak_learners =
                                                  min_residual = 0.2, control_val=control) # set tree p

        validation_df$pred <- predict_boosted_tree(fit_results, validation_df)$yp
        test_df$pred <- predict_boosted_tree(fit_results, test_df)$yp

        tr_rmse <- sqrt(mean((validation_df$y - validation_df$pred)^2))
        te_rmse <- sqrt(mean((test_df$y - test_df$pred)^2))

        new_preds <- data.frame("maxdepth"=maxdepth,"cp" = cp, "minsplit" = minsplit,
                                "tr_rmse"=tr_rmse, "te_rmse"=te_rmse)
        grid_predictions <- rbind(grid_predictions, new_preds)
    }
  }
}
```

```
# minimum Training RMSE
grid_predictions[which(grid_predictions$tr_rmse == min(grid_predictions$tr_rmse)),]
```

```
##   maxdepth    cp minsplit   tr_rmse   te_rmse
## 4       30 0.001        2 0.2266439 0.3993241
```

```
# minimum Testing RMSE
grid_predictions[which(grid_predictions$te_rmse == min(grid_predictions$te_rmse)),]
```

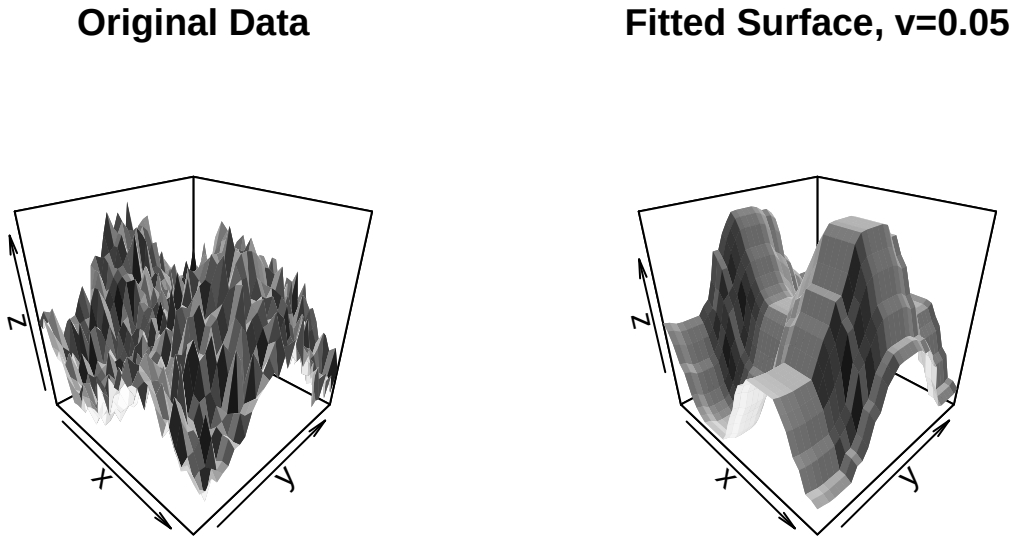
```
##   maxdepth    cp minsplit  tr_rmse   te_rmse
## 1        2 0.001        2 0.256369 0.3212162
```

Part 2: Multi-variable Data

Q0

The following code uses the functions above to build a gradient boosted tree with v=0.05. The data and the resulting prediction are plotted. The boosted tree does a pretty good job at creating a surface that fits to the data.

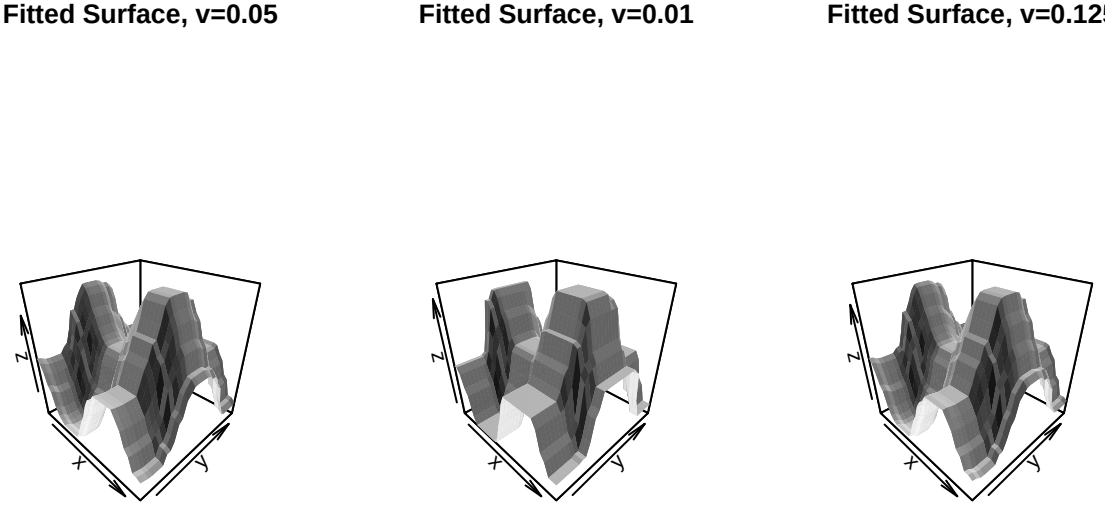
```r
#import data and format for plotting
kernel_regression_2 <- read.csv(paste0(DATA_DIR, "kernel_regression_2.csv")) %>% dplyr::rename(x1=x,x2=y
x1 <- unique(kernel_regression_2$x1)
x2 <- unique(kernel_regression_2$x2)
z_matrix <- pivot_wider(kernel_regression_2, names_from = x2, values_from = y) %>% dplyr::select(-x1) %

# fit predictions.
df_plot <- kernel_regression_2
fit_results <- fit_boosted_tree(kernel_regression_2, v = 0.05, number_of_weak_learners = 100)
df_plot$ypred <- predict_boosted_tree(fit_results, kernel_regression_2)$yp
```



Q1

The following code and plot show what happens when we vary the learning parameter v. A similar relaitonship with v is seen in these plots as in part 1. **A smaller v parameter results in greater variance (e.g. 0.01), while a larger v value increases bias (e.g. 0.125).** Although the plots with a larger v value may look like the created surface is more variable, this actually indicates that the model is making less generalizations about the data, and is therefore less flexible.

```r
v_vals=c(0.01,0.125)
for(v in v_vals){
  fit_results <- fit_boosted_tree(kernel_regression_2, v = v, number_of_weak_learners = 100)
  df_plot[,paste0("ypred", v)] <- predict_boosted_tree(fit_results, kernel_regression_2)$yp
}
```



Q2

A

Next, the following code splits the data into training and testing sets and applies the heuristic training (*stopping learning after residuals are 0.25*).

```
# split into validation and test sets 90-10
set.seed(52)
idx_val <- sample(1:nrow(kernel_regression_2),1040)
validation_df <- kernel_regression_2[idx_val,]
rownames(validation_df) <- NULL
test_df <- kernel_regression_2[-idx_val,]
rownames(test_df) <- NULL

# fit tree with heuristic approach
fit_results <- fit_boosted_tree_heuristic(validation_df, v = 0.05, max_number_of_weak_learners = 100, m
validation_df$pred <- predict_boosted_tree(fit_results, validation_df)$yp
test_df$pred <- predict_boosted_tree(fit_results, test_df)$yp
```

B

The function I created returns the number of learners trained. In this case, **it trained 56 trees**, and got similar results as the original model trained with 100.

```
fit_results$number_of_weak_learners
```

```
## [1] 56
```

C

The RMSE on the test set was **0.37014**, which is larger than the training RMSE. This indicates we probably could decrease the flexibility (number of trees) even more.

```
# Training RMSE
sqrt(mean((validation_df$y - validation_df$pred)^2))
```

```
## [1] 0.3157351
```

```
# Test RMSE
sqrt(mean((test_df$y - test_df$pred)^2))
```

```
## [1] 0.37014
```

Q3

According to this grid search, it appears **max_depth of 2 was best for both the training and testing data**. Additionally, a **cp (complexity param) of 0.001 was the best for training and testing data**. Lastly, it appeared having a small number of minimum split values was best, 2 on the test data and **a minsplit of 2 on the training data found the smallest RMSE**.

```
# values to search on
minsplit_vals = c(2,5,10,20,30)
cp_vals = c(0.001, 0.005, 0.01, 0.05, 0.1)
maxdepth_vals = c(2,10,20,30)

#gridsearch
grid_predictions <- data.frame()
for(minsplit in minsplit_vals){
  for(cp in cp_vals){
      for(maxdepth in maxdepth_vals){
        control <- rpart.control(minsplit = minsplit,  cp = cp, maxdepth = maxdepth)
        fit_results <- fit_boosted_tree_heuristic(validation_df, v = 0.05, max_number_of_weak_learners =
                                                  min_residual = 0.25, control_val=control) # set tree

        validation_df$pred <- predict_boosted_tree(fit_results, validation_df)$yp
        test_df$pred <- predict_boosted_tree(fit_results, test_df)$yp

        tr_rmse <- sqrt(mean((validation_df$y - validation_df$pred)^2))
        te_rmse <- sqrt(mean((test_df$y - test_df$pred)^2))

        new_preds <- data.frame("maxdepth"=maxdepth,"cp" = cp, "minsplit" = minsplit,
                                "tr_rmse"=tr_rmse, "te_rmse"=te_rmse)
        grid_predictions <- rbind(grid_predictions, new_preds)
    }
  }
}
```

```r
# minimum Training RMSE
grid_predictions[which(grid_predictions$tr_rmse == min(grid_predictions$tr_rmse)),]
```

```
##    maxdepth    cp minsplit   tr_rmse   te_rmse
## 24       30 0.001        5 0.3008067 0.4580776
```

```r
# minimum Testing RMSE
grid_predictions[which(grid_predictions$te_rmse == min(grid_predictions$te_rmse)),]
```

```
##   maxdepth    cp minsplit   tr_rmse   te_rmse
## 1        2 0.001        2 0.3141224 0.3740037
```

Question 2: t-SNE

Part 1

a. Do the distances between points in tSNE matter?

This depends on what distances you're looking at, but generally, no. *For instance, the size of clusters (i.e. distances between points within clusters) is meaningless.* This is because t-SNE creates "distance" through regional density variations, and thus expands dense clusteres ad contracts sparse ones. Therefore, t-SNE cluster size will not say anything important about the original data.

The relaltionships between distances between clusters is not so clear-cut. Sometimes, with a correctly tuned complexity, you're able to find good representation of *global* distances between clusters. But this is not always the case, and often t-SNE does not depict global cluster distances well.

b. What does the parameter value "perplexity" mean?

Perplexity is intuitivly a guess about the number of "close" neghbors every point in the data has. *It is therfore highly effected by the size of your data.* It is therefore a rule-of-thumb that perplexity should be smaller than your number of data points, and it's noted that it's generally betweed 5 and 50. It is usually a hyper-param you need to plot multiple values of to determine.

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

t-SNE is an itterative algorithm that minimizes a loss function, therefore there are a certain number of steps needed to converge. *The point is to have the correct number of steps to each a stable distribution.* Therefore, if you stop too early, the algorithm may not converge, even if other hyperparameters are correct for the dataset. Additionally, if you stop after too many itterations and your eta was too large, you may find that you've still not reached a stable distribution.

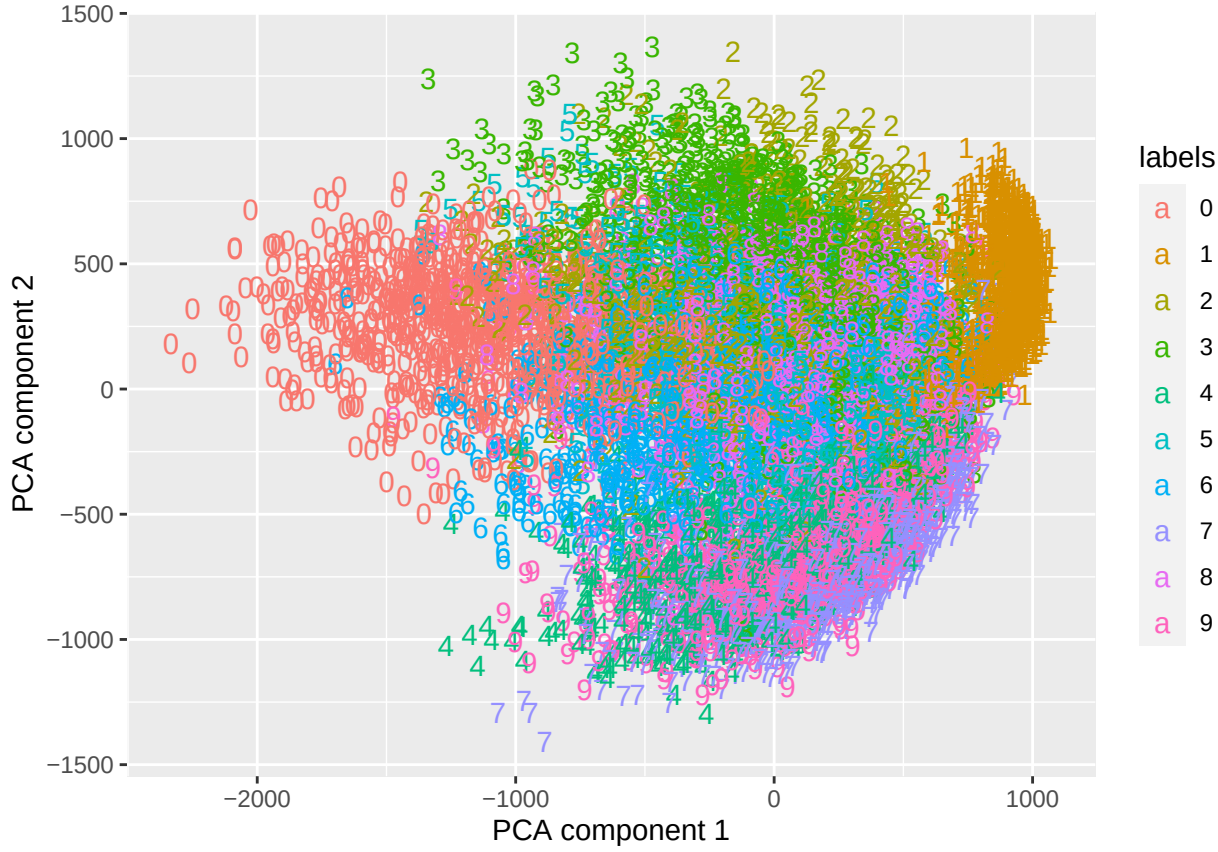
d. Explain why you may need more than one plot to explain topological information using tSNE

With all dimention reduction, the goal is to get some kind of topological information of the data. For t-SNE, this is possible but it usually required multiple plots of different perplexities. This is because, as noted before t-SNE embeddings are highly influences by this hyper-parameter, and thus you may see different properties in various plots. It is important to try multiple plots with varying hyper-parameters to see where there is a consistent trend in the shape and distribution of embedded points. Additionally, different runs of t-SNE can get different results, so it is important to create multiple plots for that reason as well.

Part 2

a. Plot of PCA

The following is a plot the the first two Principle Components for the first 10,000 samples in the MNIST training set. As you can see, there are some unique clusters that form, but still a lot of overlap.



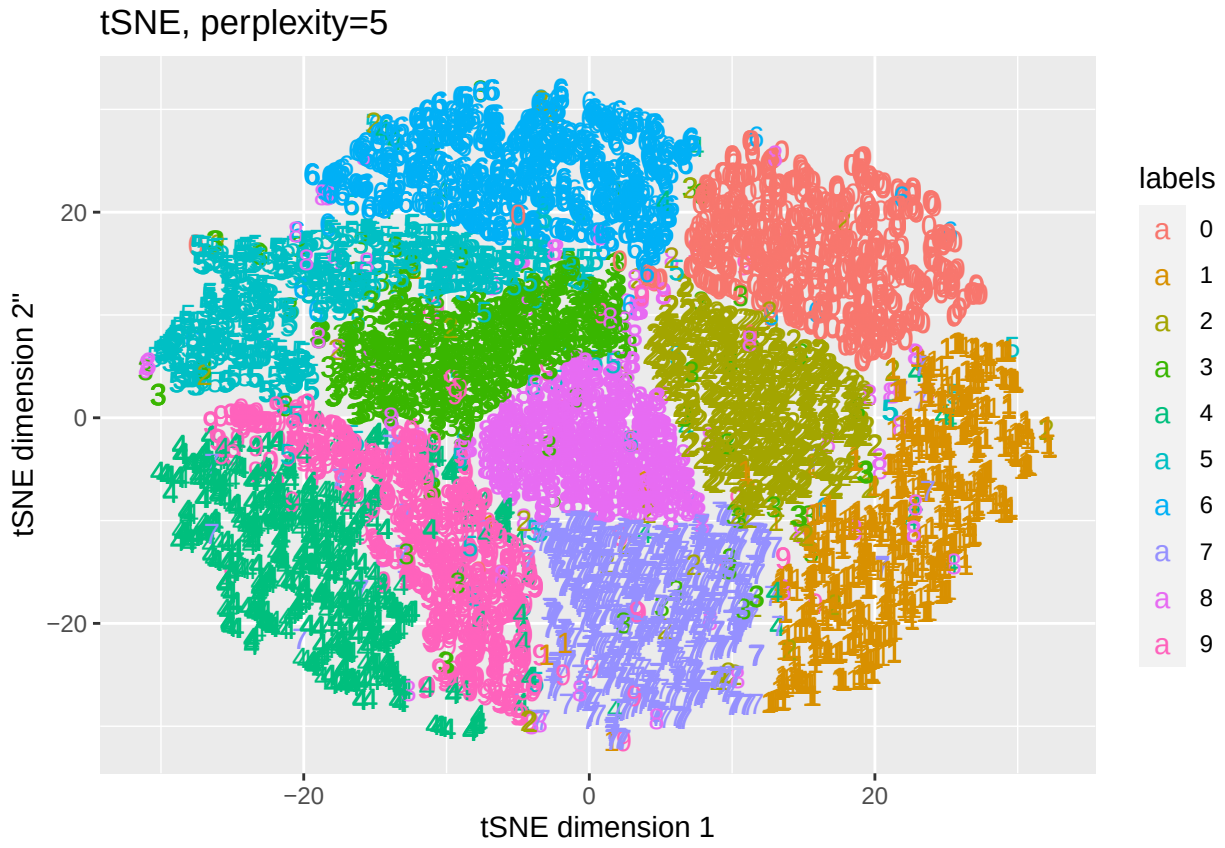
b. Plot the TSNE embedding for perplexity = 5 use 500 iterations.

Conversly, the following code creates an embedding with perplexity = 5. In the plot we can see that there are very distinct non-overlapping clusters.

```
embedding_5 = Rtsne(X = first_10k_samples, dims = 2,
                    perplexity = 5,
                    theta = 0.5,
                    eta = 200,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
```

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 5.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
```

```
## Done in 9.13 seconds (sparsity = 0.002104)!
## Learning embedding...
## Iteration 50: error is 118.296620 (50 iterations in 1.75 seconds)
## Iteration 100: error is 105.685762 (50 iterations in 1.69 seconds)
## Iteration 150: error is 98.718553 (50 iterations in 1.61 seconds)
## Iteration 200: error is 95.994401 (50 iterations in 1.58 seconds)
## Iteration 250: error is 94.538673 (50 iterations in 1.67 seconds)
## Iteration 300: error is 4.360393 (50 iterations in 1.61 seconds)
## Iteration 350: error is 3.803723 (50 iterations in 1.50 seconds)
## Iteration 400: error is 3.438262 (50 iterations in 1.48 seconds)
## Iteration 450: error is 3.172776 (50 iterations in 1.30 seconds)
## Iteration 500: error is 2.968350 (50 iterations in 1.36 seconds)
## Fitting performed in 15.55 seconds.
```



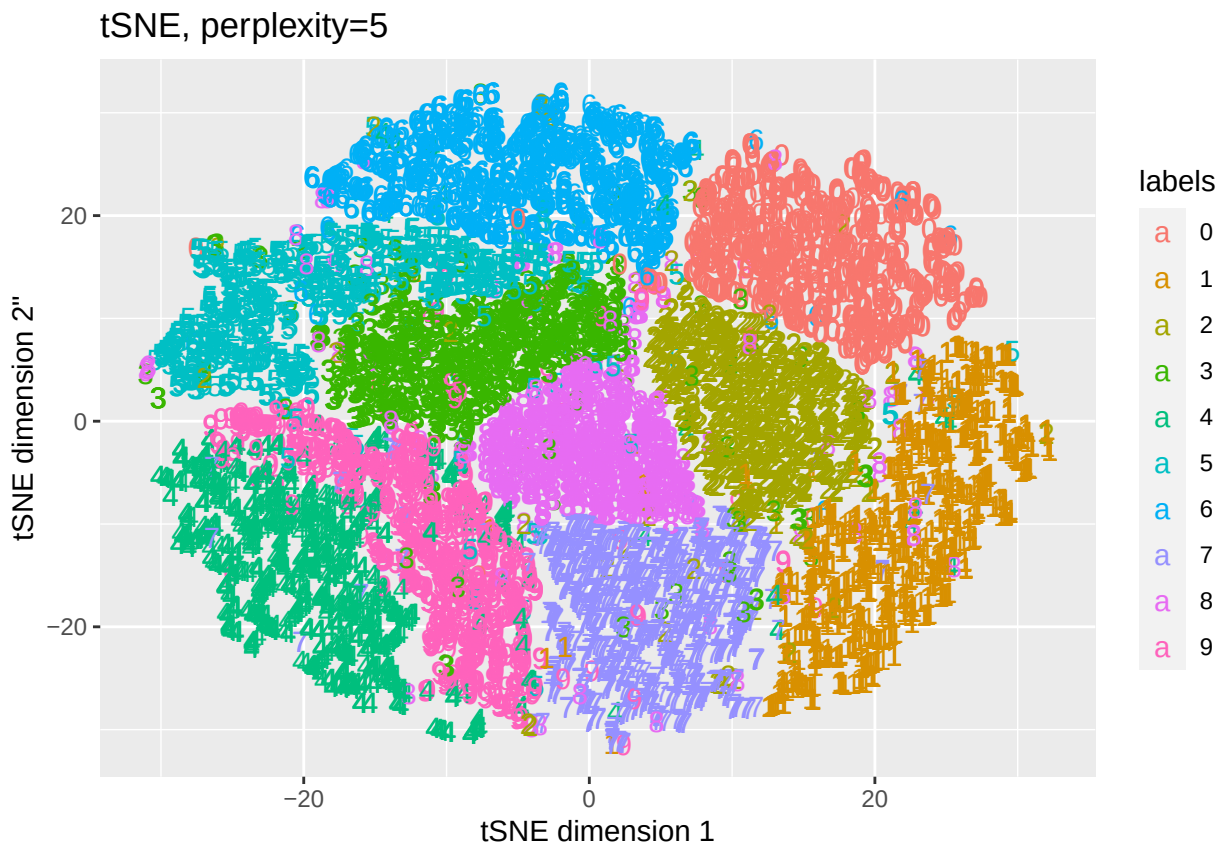
c. Plot the TSNE embedding for perplexity = 5,20,60,100,125,160, what do you notice?

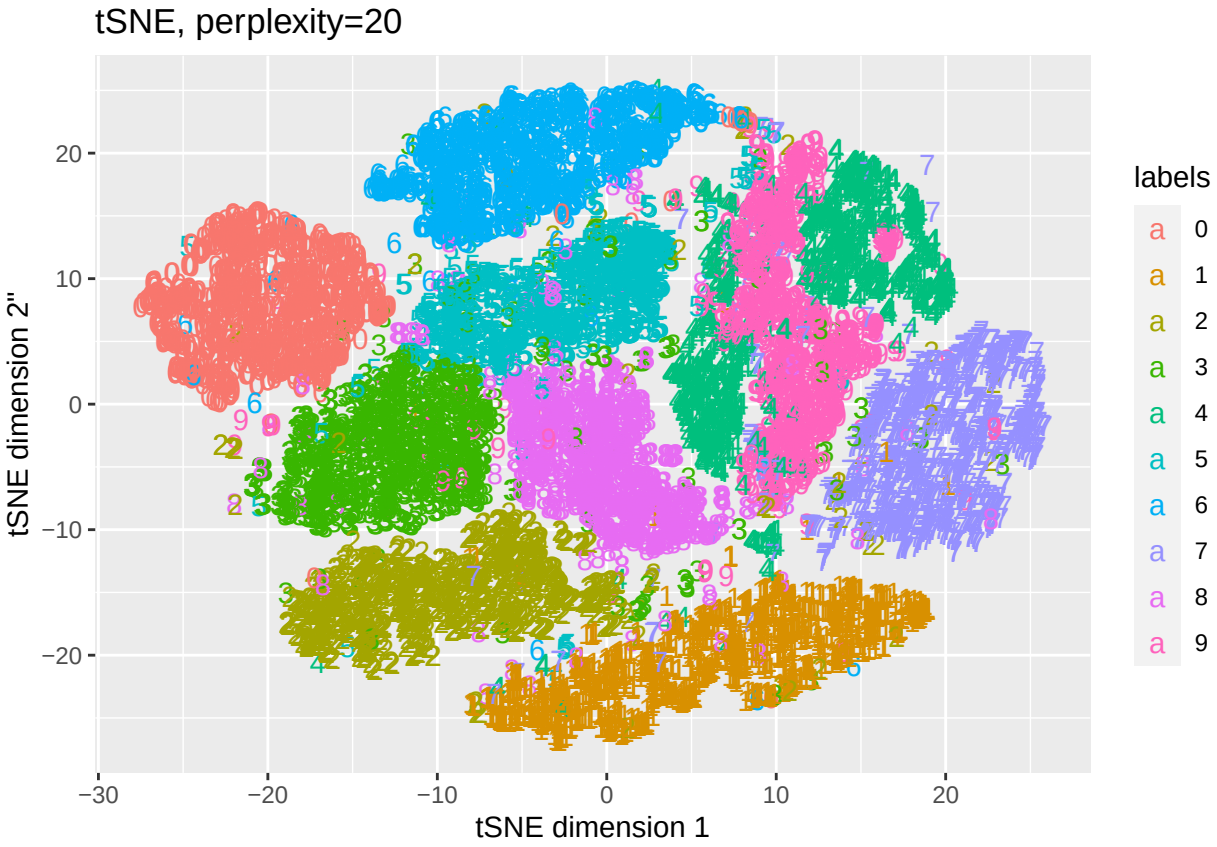
Next, the following creates TSNE embeddings for different complexity values and then plots them. As the perplexity increases, **the clusters get denser, but the space between clusters is more likely to overlap**. For instance, as perplexity increases, the clusters for 9 and 4 get closer and closter together, to the point where they appear to be in mostly the same cluster by p=160. Also, the number of points on in any identifable cluster appear to increase as perplexity increases.

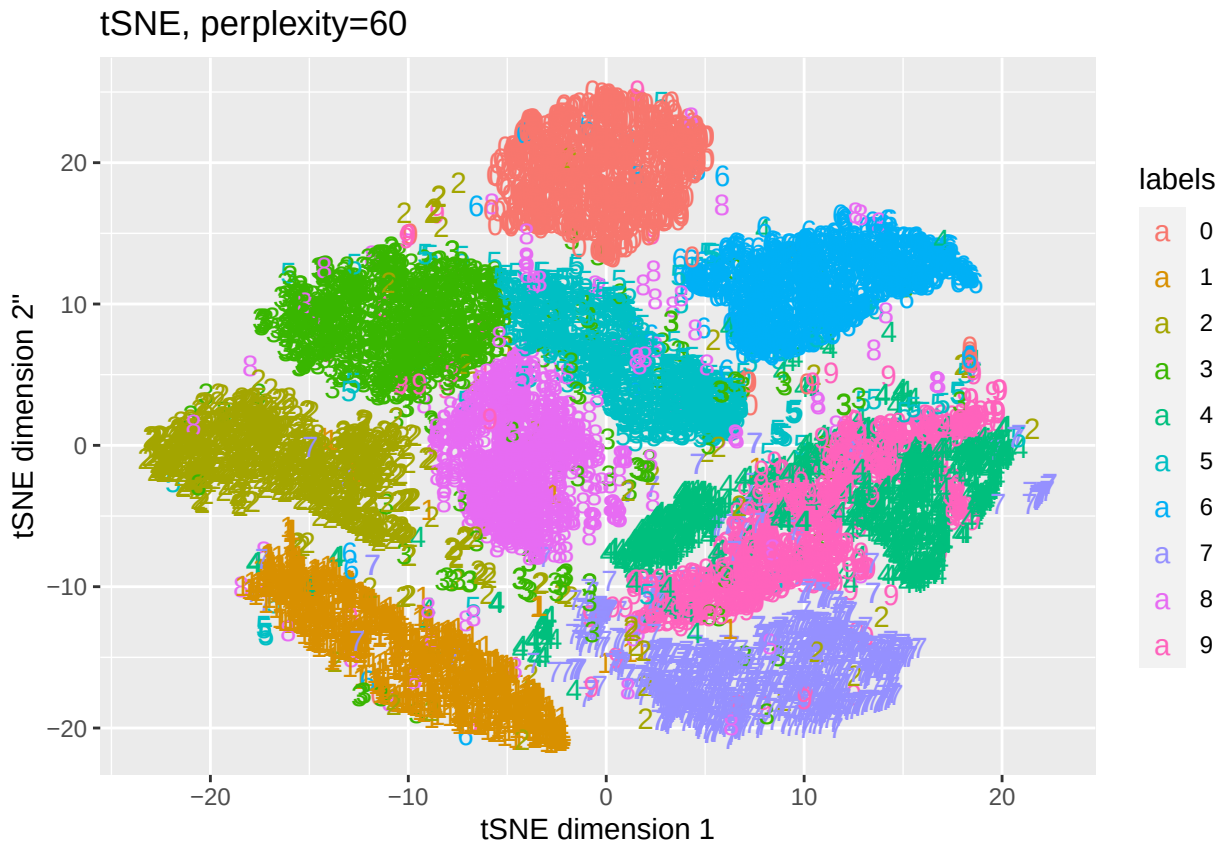
```
embeddings_all <- list("embeding_5" = embedding_5)
perplexities <- c(20,60,100,125,160)
```

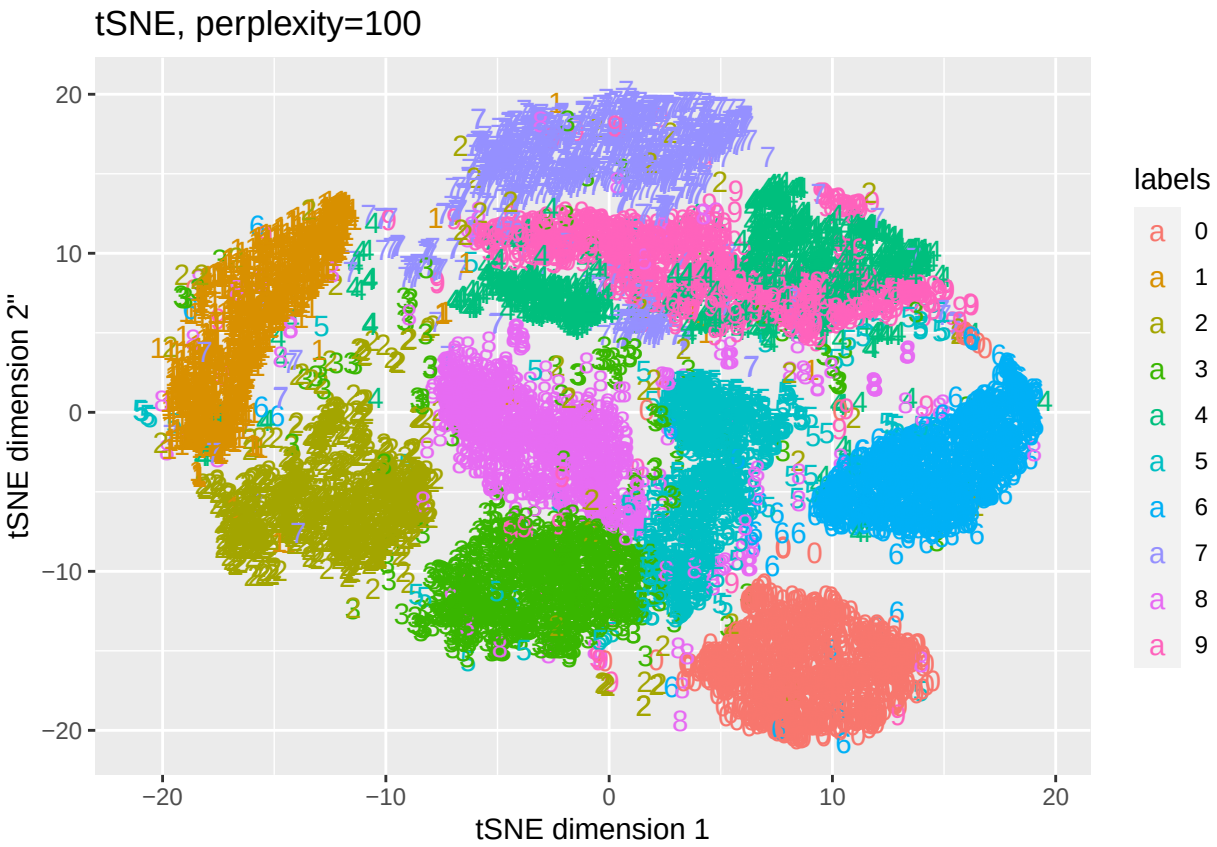
```
for(p in perplexities){
  embeddings_all[[paste0("embeding_",p)]] <- Rtsne(X = first_10k_samples, dims = 2,
                    perplexity =p,
                    theta = 0.5,
                    eta = 200,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
}
```

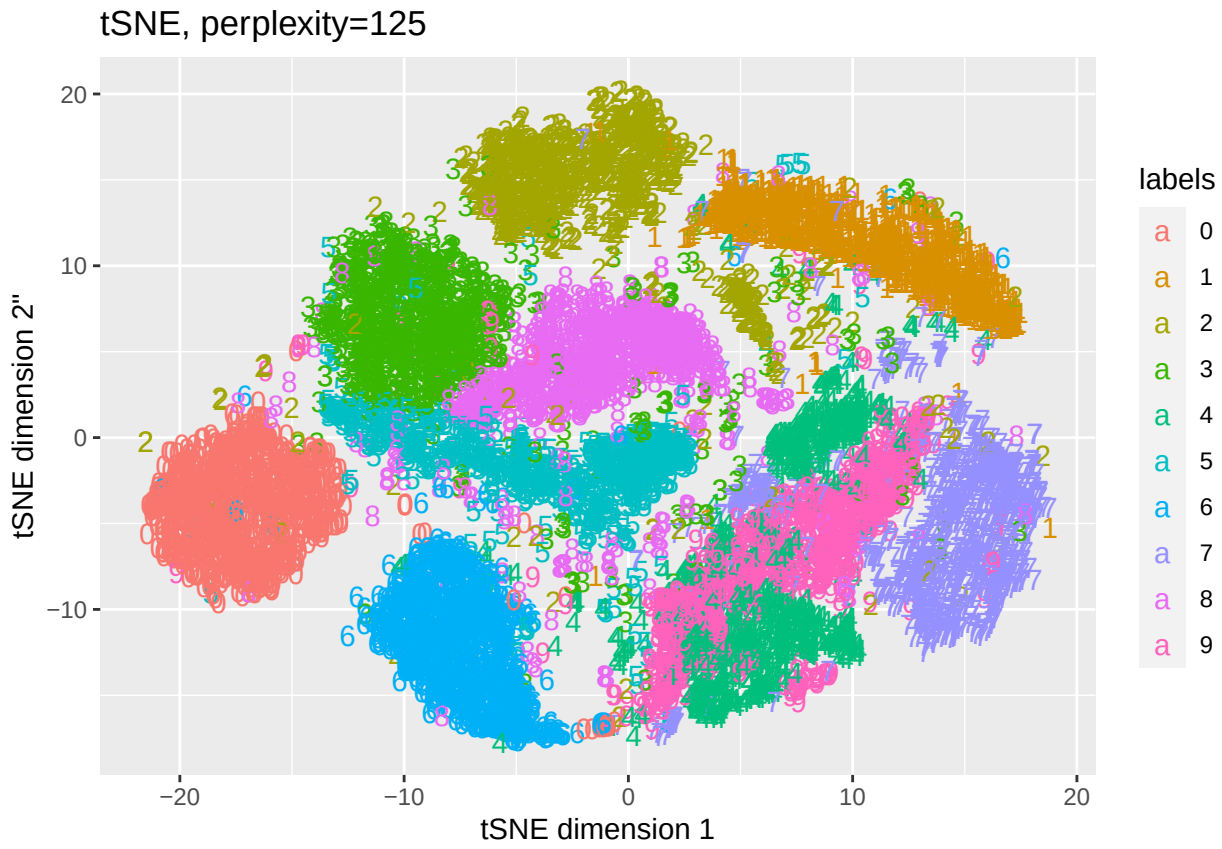
```
perplexities <- c(5,perplexities)
for(p in perplexities){
  new_emb <- embeddings_all[[paste0("embeding_",p)]]

  print(
  tibble(x = new_emb$Y[,1], y = new_emb$Y[,2],
                           labels = as.character(first_10k_samples_labels)) %>%
  ggplot(aes(x = x, y=y,label = labels, color = labels)) +
    geom_text() +xlab('tSNE dimension 1') +ylab('tSNE dimension 2"') +
    labs(title = paste0("tSNE, perplexity=",p)))
}
```

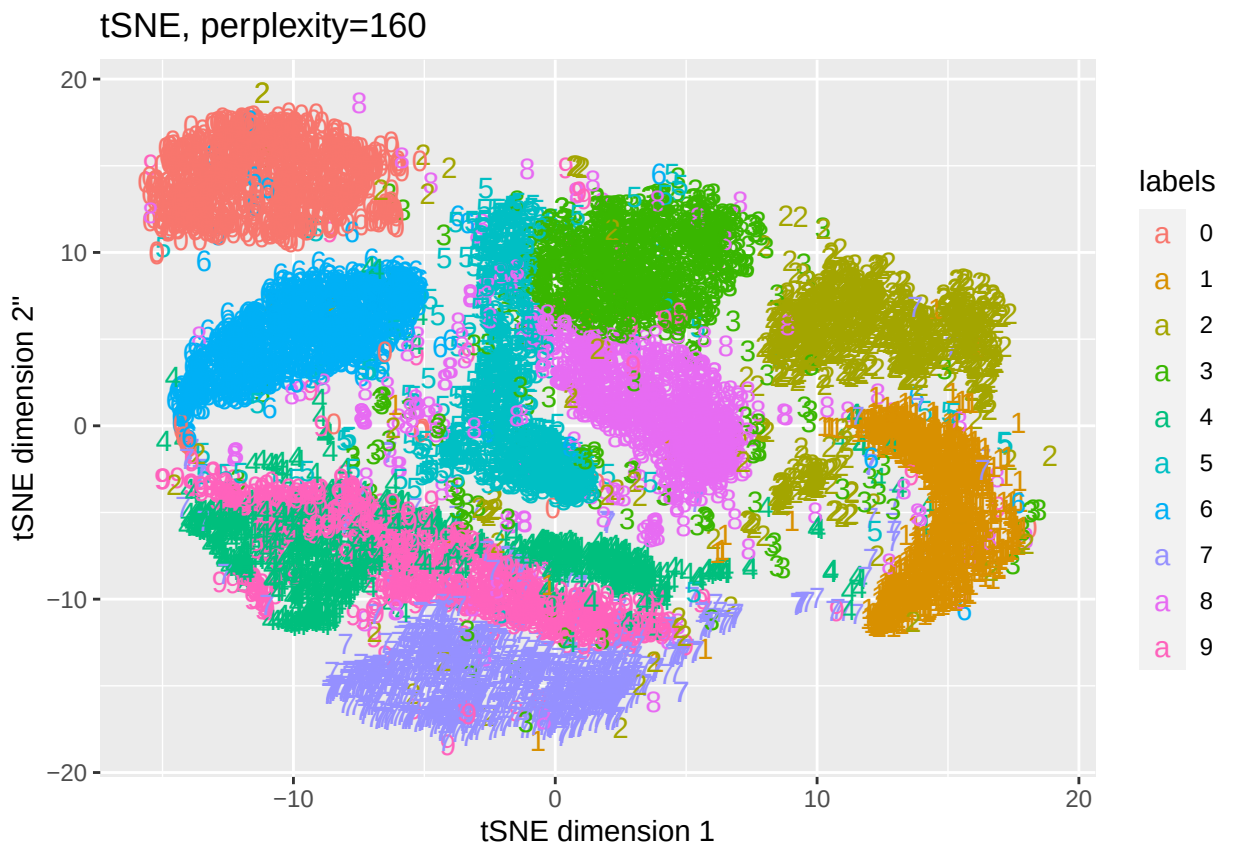



tSNE, perplexity=20
tSNE dimension 2"
tSNE dimension 1
20
10
0
−10
−20
−30
−20
−10
0
10
20
labels
a 0
a 1
a 2
a 3
a 4
a 5
a 6
a 7
a 8
a 9


tSNE, perplexity=60
tSNE dimension 2"
tSNE dimension 1
20
10
0
−10
−20
−20
−10
0
10
20
labels
a 0
a 1
a 2
a 3
a 4
a 5
a 6
a 7
a 8
a 9


tSNE, perplexity=100
tSNE dimension 2"
tSNE dimension 1
20
10
0
−10
−20
−20
−10
0
10
20
labels
a 0
a 1
a 2
a 3
a 4
a 5
a 6
a 7
a 8
a 9


tSNE, perplexity=125
tSNE dimension 2"
tSNE dimension 1
20
10
0
−10
−20
−10
0
10
20
labels
a 0
a 1
a 2
a 3
a 4
a 5
a 6
a 7
a 8
a 9

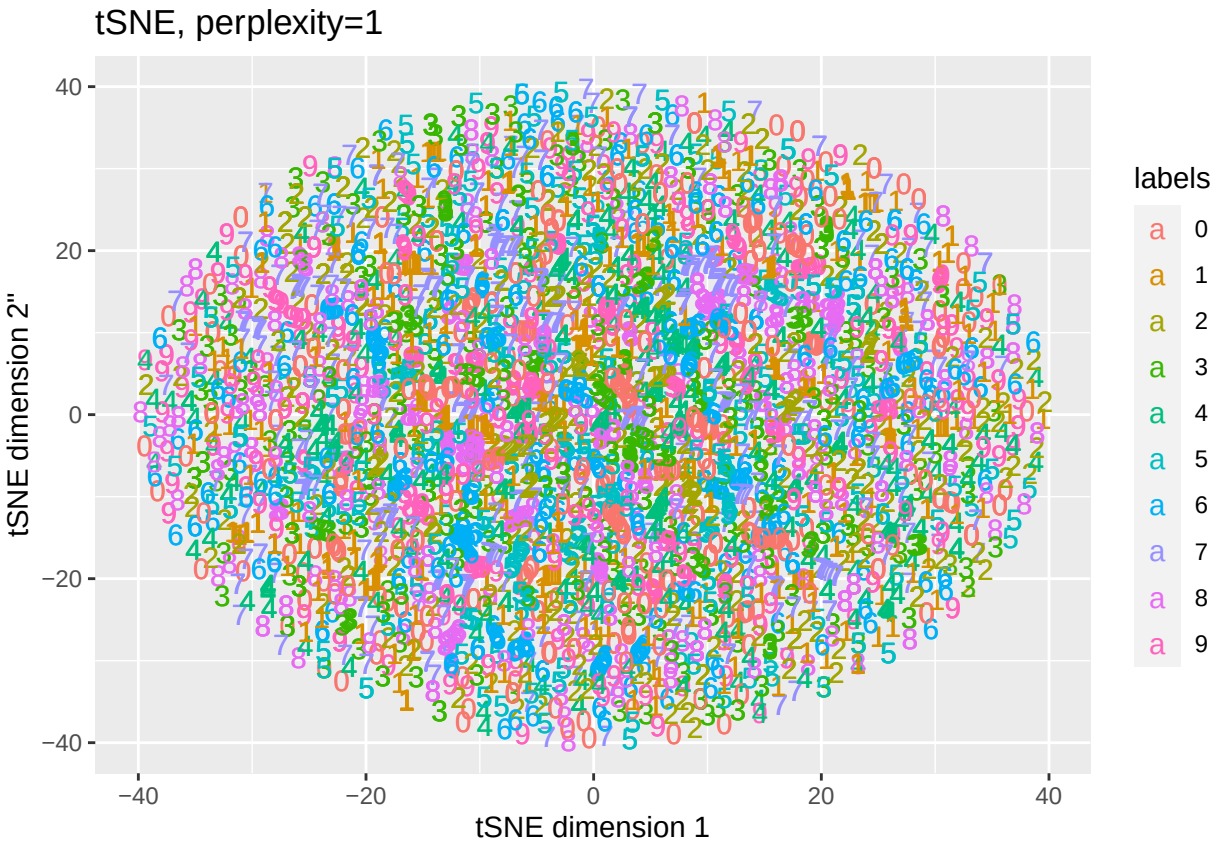


d. If the perplexity is set to 1 what would the distribution of values look like in 2d, provide an explanation as to why.

Here's what it looks like. The distribution of values is equal for all points, and looks like an oval of evenly distributed datapoints. This is because, when t-SNE is small, local variations dominate. Thus not global relationships are found. Instead, the algorithm just makes sure each point has on average one close neighbor, which will evenly distriubte all the points.

```
## Performing PCA
## Read the 10000 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 1.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
##  - point 10000 of 10000
## Done in 11.76 seconds (sparsity = 0.000442)!
## Learning embedding...
## Iteration 50: error is 135.169145 (50 iterations in 2.04 seconds)
## Iteration 100: error is 116.636799 (50 iterations in 1.87 seconds)
## Iteration 150: error is 108.509254 (50 iterations in 1.95 seconds)
## Iteration 200: error is 104.007323 (50 iterations in 2.06 seconds)
## Iteration 250: error is 100.896104 (50 iterations in 2.18 seconds)
## Iteration 300: error is 5.264544 (50 iterations in 2.03 seconds)
## Iteration 350: error is 4.625414 (50 iterations in 2.28 seconds)
## Iteration 400: error is 4.129135 (50 iterations in 2.31 seconds)
## Iteration 450: error is 3.739518 (50 iterations in 2.36 seconds)
```

```
## Iteration 500: error is 3.426792 (50 iterations in 2.40 seconds)
## Fitting performed in 21.48 seconds.
```



e. How about if the perplexity is set to 5000 what would the distribution of values look like in 2d, provide an explanation as to why.

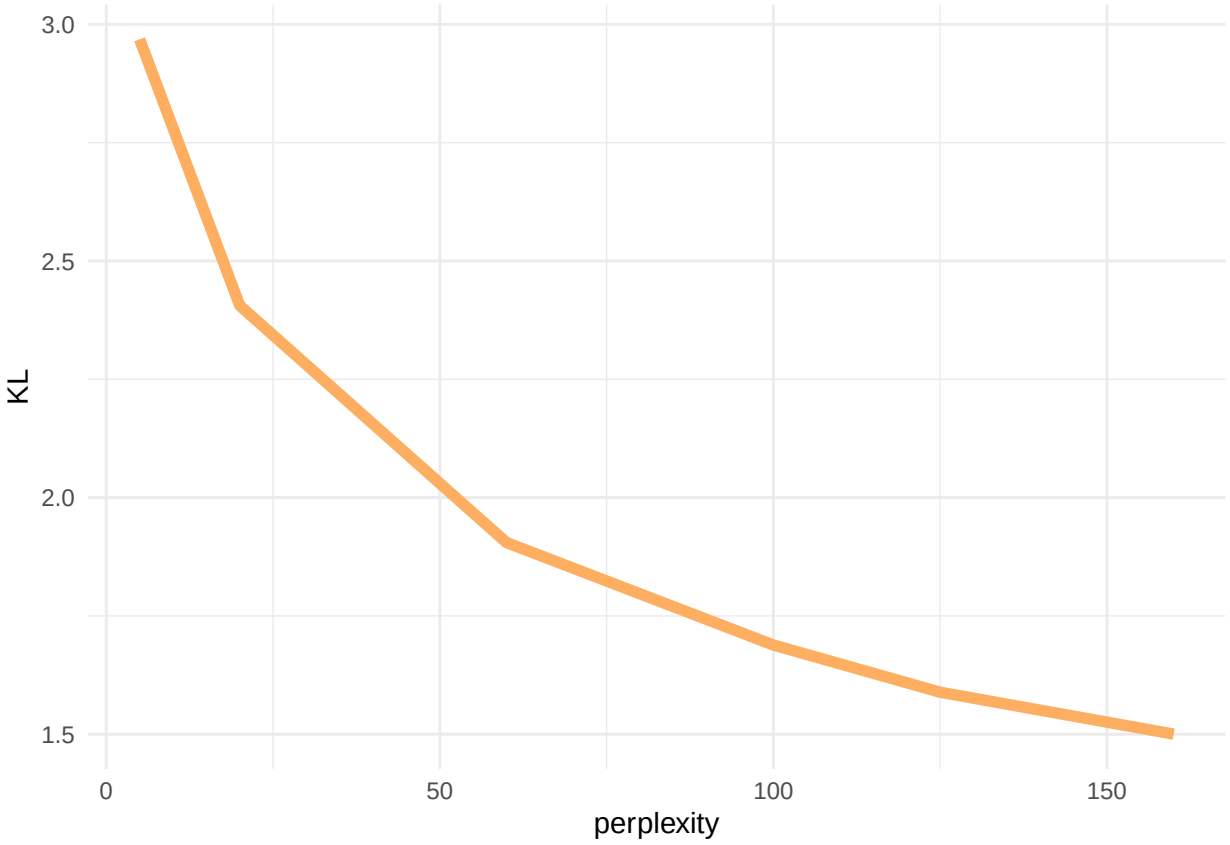
If perplexity is set to 5,000, then since we're working with 10,000 examples this is half the dataset, and will thus highly bias the t-SNE embeddings. More specifically, the algorithm will try to make sure each point has about 5,000 close neighrbors, which will mean that no global trends will be picked up on. Instead, the embedding will just look like a random collection of points.

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

The goal is to **maximize** KL diverence by the end of the iterations. Here we can see that KL-Divergence is maximized at the perplexity of 5. This makes sense given the plots we saw above.

```
kl_byp <- data.frame()
for(p in perplexities){
  new_emb <- embeddings_all[[paste0("embeding_",p)]]
  new <- data.frame("perplexity" = p,  "KL" = new_emb$itercosts[length(new_emb$itercosts)])
  kl_byp <- rbind(kl_byp, new)
}

kl_byp %>% ggplot() +
```

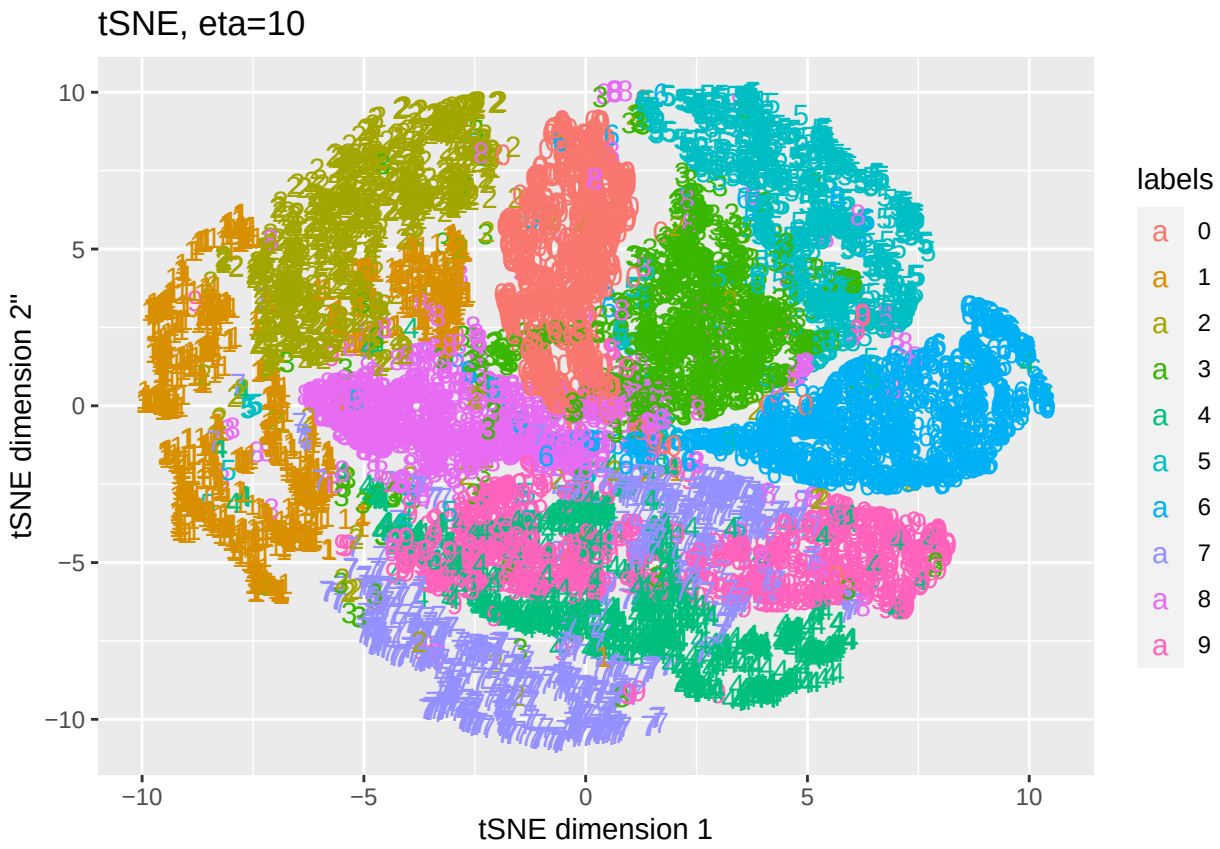
```
  geom_line(aes(x=perplexity, y=KL), color=colors[4], size=2) +
  theme_minimal()
```

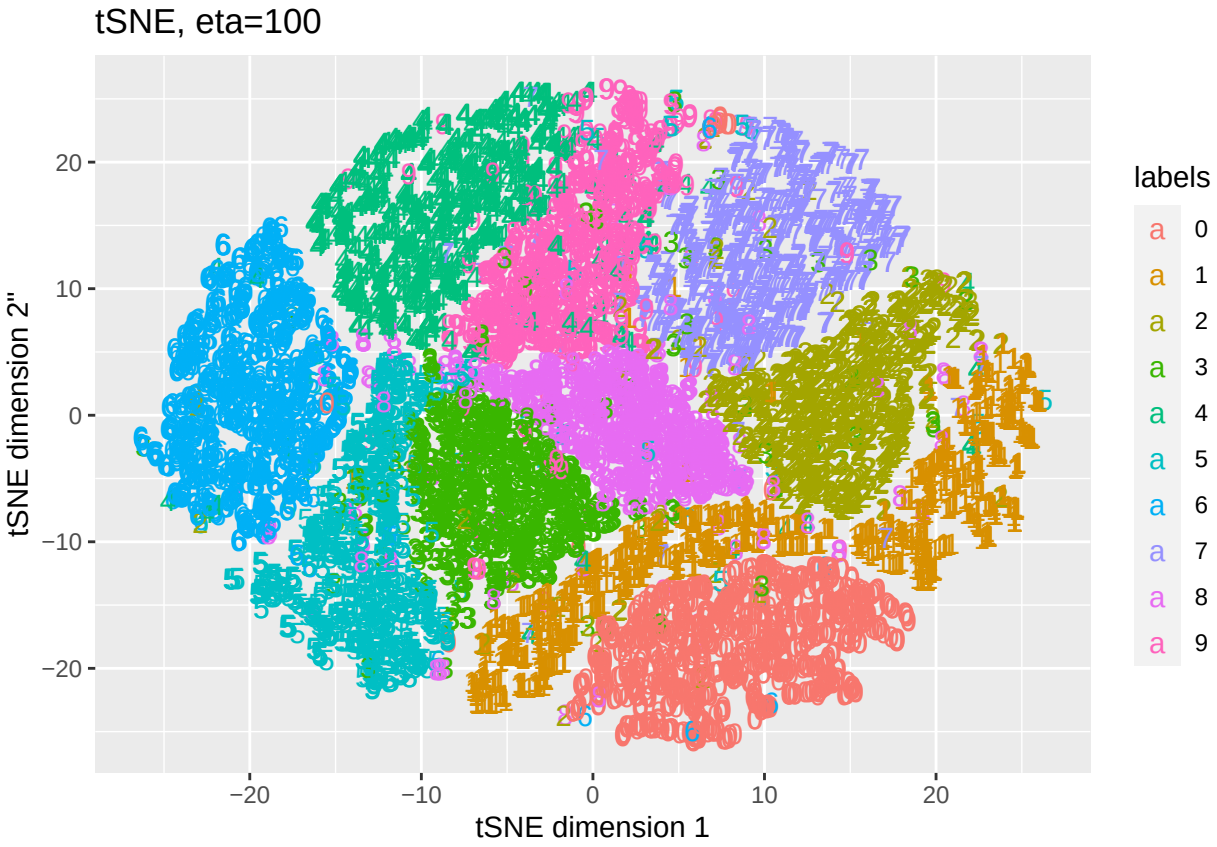


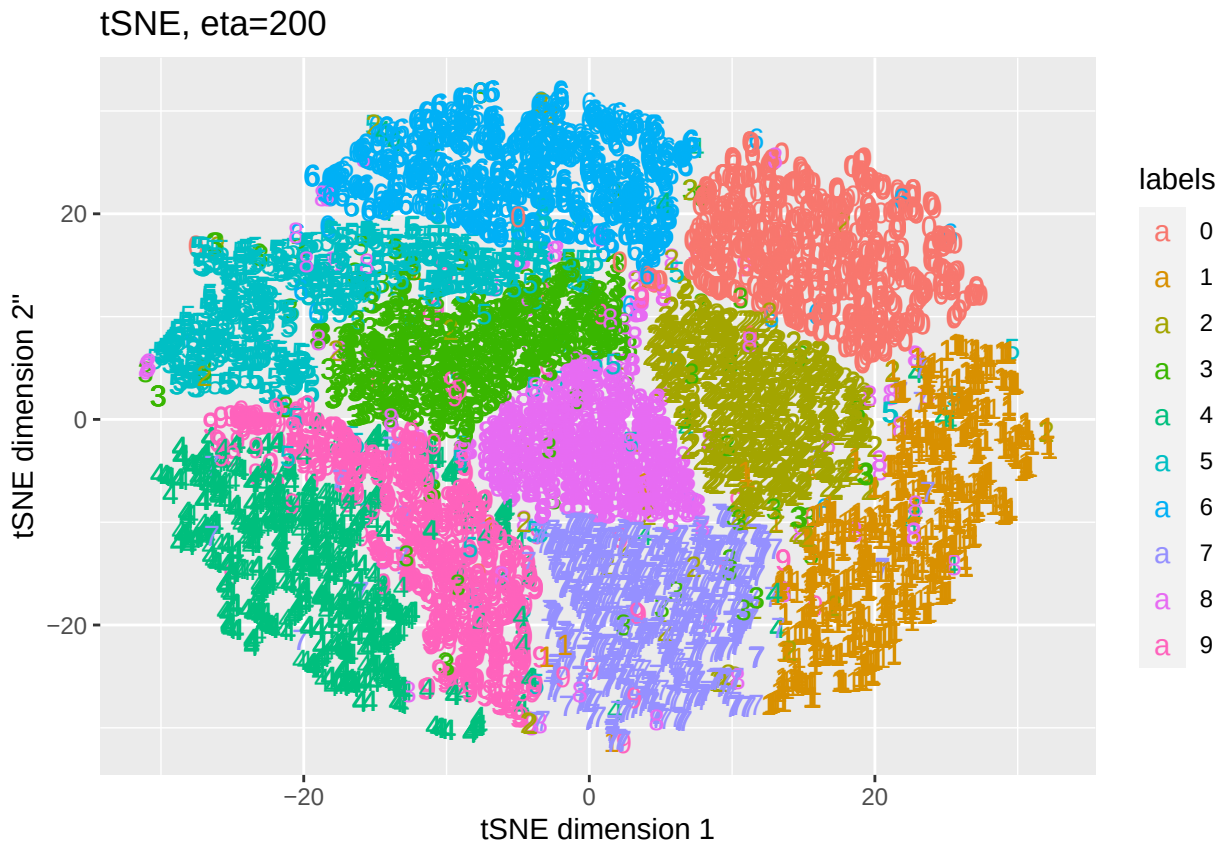
g. Plot the embeddings for eta=(10,100,200) while keeping max_iter and your optimal perplexity value selected above constant. What do you notice?

The following plots show the impact of varying eta, otherwise known as the learning rate, for the same number of itterations and a perplexity of 5. As you can see from the plots, eta impacts how quickly tSNE will converge. The smaller eta is, the less converged each plot appears to be.

```
etas=c(10,100)
etas_all <- list("embeding_200" = embedding_5)

for(e in etas){
  etas_all[[paste0("embeding_",e)]] <- Rtsne(X = first_10k_samples, dims = 2,
                    perplexity =5,
                    theta = 0.5,
                    eta = e,
                    pca = TRUE, verbose = TRUE,
                    max_iter = 500)
}
```


tSNE, eta=10
tSNE dimension 2"
tSNE dimension 1
10
5
0
−5
−10
−10
−5
0
5
10
labels
a 0
a 1
a 2
a 3
a 4
a 5
a 6
a 7
a 8
a 9


tSNE, eta=100
tSNE dimension 2"
tSNE dimension 1
20
10
0
−10
−20
−20
−10
0
10
20
labels
a 0
a 1
a 2
a 3
a 4
a 5
a 6
a 7
a 8
a 9



Question 3: Word2Vec

1.

Looking through a few of the files this model is trained on, it is clear that there are many different words used relativly close together in each document. This is because very different recipies are listed after one another within each cookbook. Additionally, recipies vary significantly in length, but generally a recpipy is at least two sentences long.

This these properties in the data known, 1. you could increase the size of window from 6 to perhaps even larger. Given the context words are in the same recipe for ranges larger than 6 words, it is possible you can take into account a larger window. 2. you can use a large number of negative samples, perhaps even larger than 15. This is because there a lot of very different words used in each document, and it makes sense to break this document up into many samples.

```
## 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%
```

2.

The sample ingreidents I tested were **beef, noodle, and coriander**. The following code prints out the five closest words to these ingredients. For beef we often get synonyms like "steak" and different types of meat like "veal", as well as various cuts of beef like "tenderloin". Similarly, for "noodle" we get various different types of pasta and then identical words like the pural "noodles". Then for coriander, we instead get different seasonings and garnishes like "cumin" and "anise".

```
ingredients <- c("beef", "noodle", "coriander")

# Searching closest words to sage
for(ingredient in ingredients){
  print(model %>% closest_to(model[[ingredient]],6))
}
```

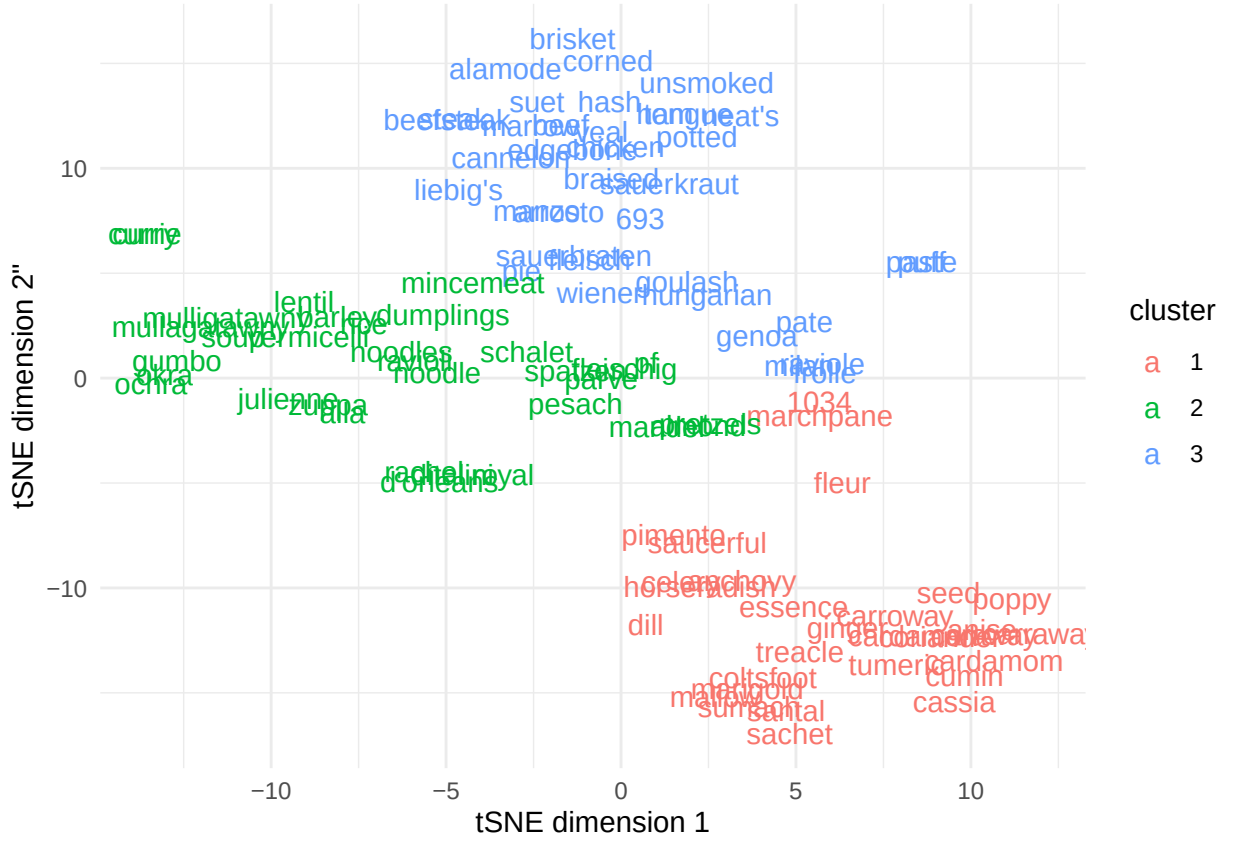
```
##         word similarity to model[[ingredient]]
## 1       beef                           1.0000000
## 2       veal                           0.7858701
## 3     corned                           0.7818829
## 4     mutton                           0.7772374
## 5      steak                           0.7430162
## 6 tenderloin                           0.7397652
##      word similarity to model[[ingredient]]
## 1  noodle                          1.0000000
## 2 ravioli                          0.7151656
## 3 noodles                          0.6806432
## 4    puff                          0.6264613
## 5 schalet                          0.6069160
## 6   milan                          0.6034482
##        word similarity to model[[ingredient]]
## 1 coriander                           1.0000000
## 2  cardamon                           0.8163085
## 3   caraway                           0.8095634
## 4     anise                           0.7795191
## 5  cardamom                           0.7481335
## 6     cumin                           0.7185984
```

3.

The following code plots t-SNE embeddings of various words related to the three ingredients we created above. After some tuning, we can see that t-SNE is able to separate out words and cluster them by their relationship to the original word. Athough, interestingly, it appears the meat and noodle clusters are more connected than the seasoning cluster.

```
closest_ingredients = closest_to(model,model[[ingredients]], 100)$word
surrounding_ingredients = model[[closest_ingredients,average=F]]

# use t-SNE
```

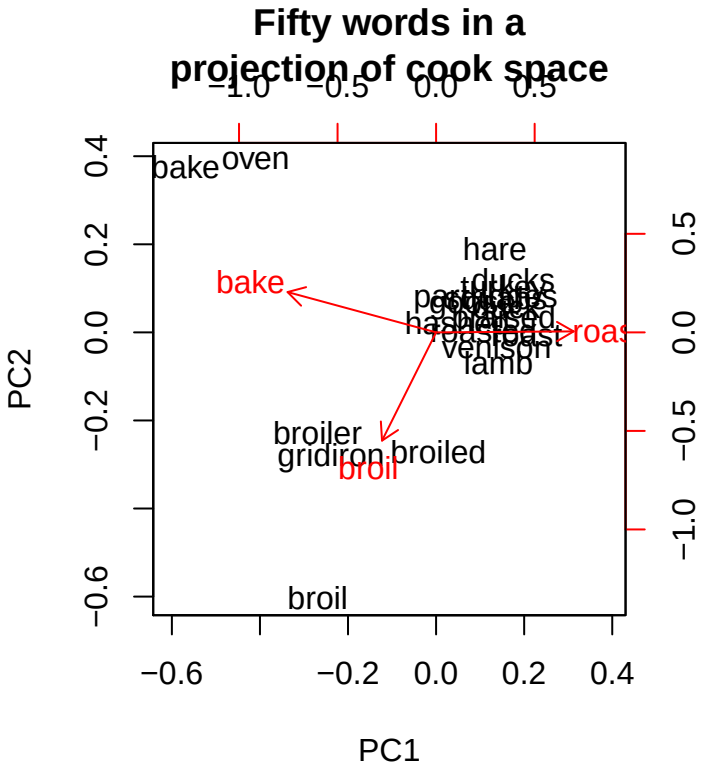
```
embedding = Rtsne(X = surrounding_ingredients, dims = 2,
                  perplexity = 8,
                  theta = 0.5,
                  eta = 10,
                  pca = TRUE, verbose = TRUE,
                  max_iter = 2000)

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


# see if we can separate them by the original 3 ingredients
set.seed(53)
n_centers = 3
clustering = kmeans(embedding_vals,centers=n_centers,
                    iter.max = 5)
```



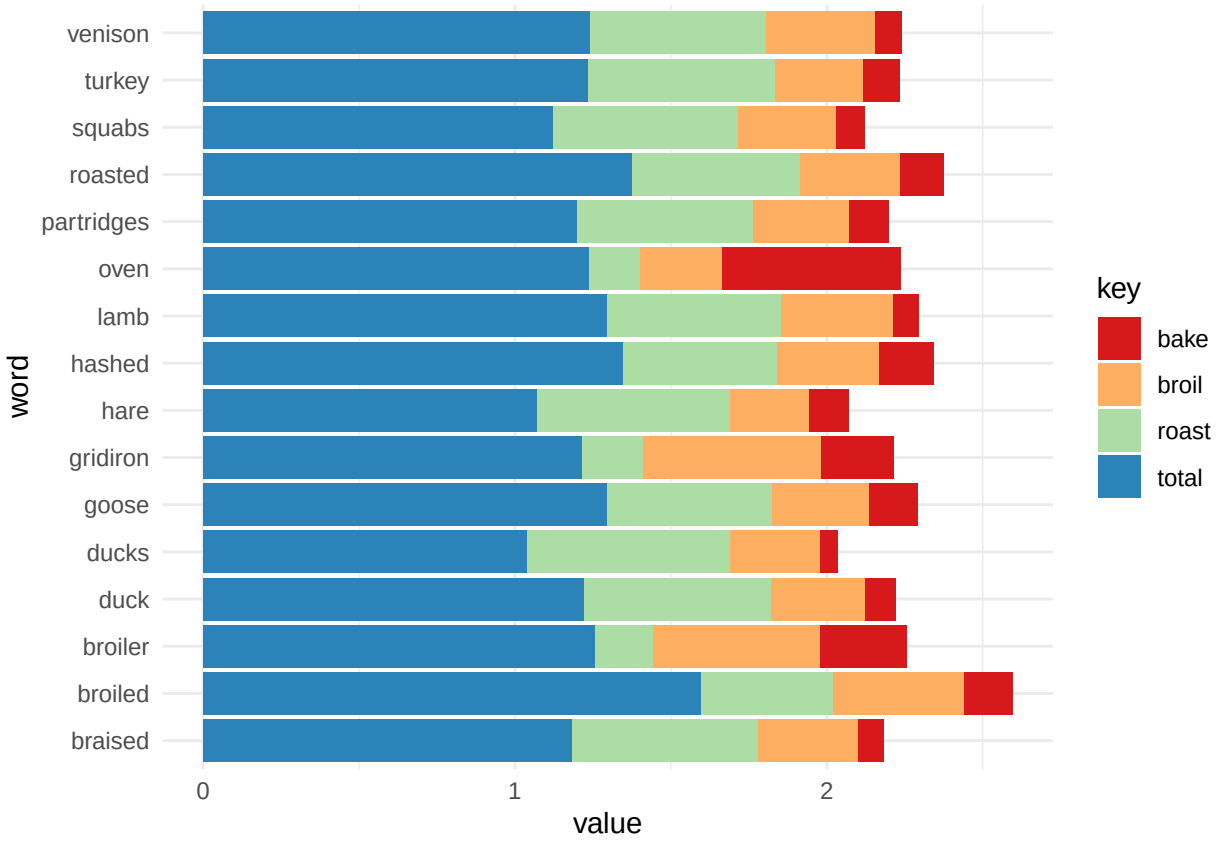
4.

After some experimenting, I found a set of three "orthoginal" words that mapped out correctly in the dataset. **bake, broil, and roast** all map out in a way that makes sense. You can see from the plot below that the words are orthoginal, as they are intuitivly as well, since they are different types of cooking. Additionally we can see that similar words for bake are "oven", for broil are "gridiron", and for roast are "ducks", "lamb", and other foods you would often roast.

```
cooking_types = c("bake", "broil", "roast")
common_similarities_tastes = model[1:3000,]%>% cosineSimilarity( model[[cooking_types,average=F]])
high_similarities_to_tastes = common_similarities_tastes[rank(-apply(common_similarities_tastes,1,max))

# plot to make sure they make sense
high_similarities_to_tastes %>%
  prcomp %>%
  biplot(main="Fifty words in a\nprojection of cook space")
```



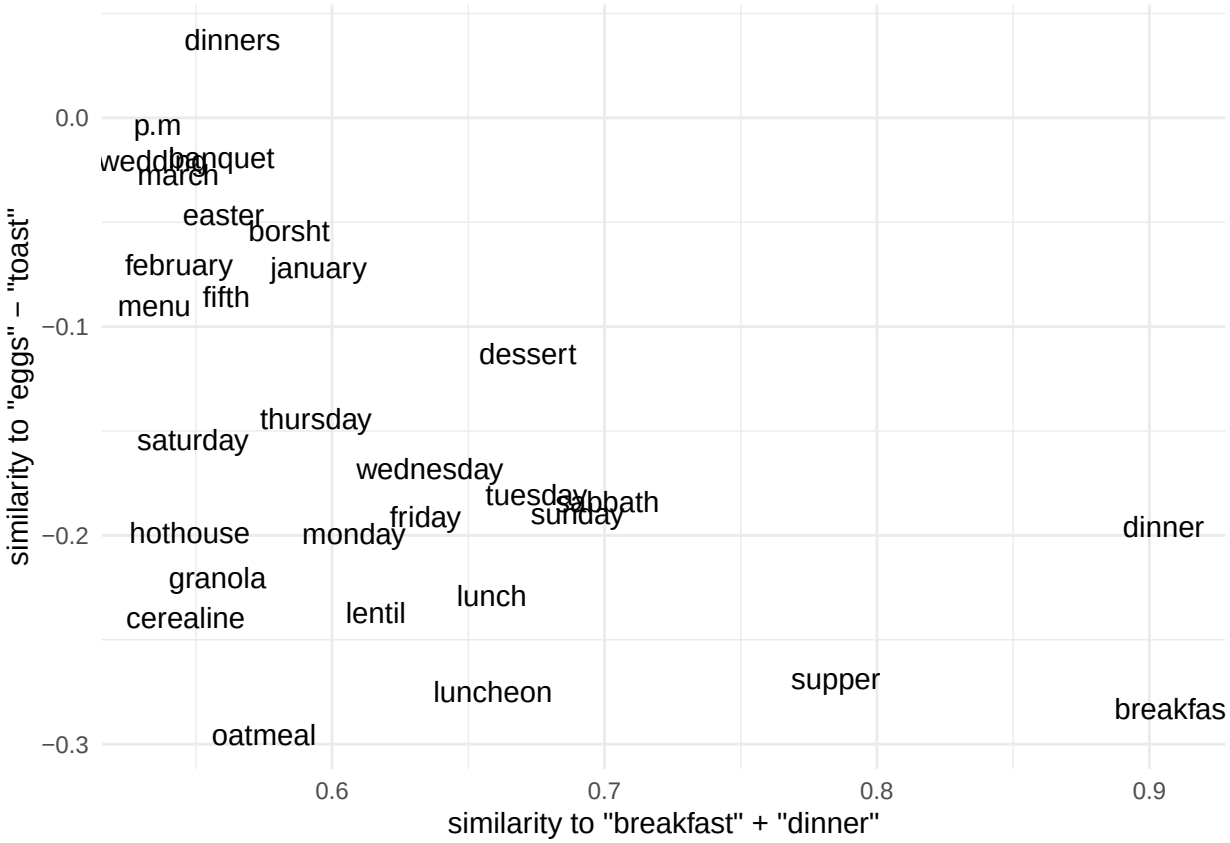
We can also plot all the words selected and see how they relate to each word. Of the words selected, the majority are most similar to roast. However, certain words like "broilder" and "oven" are more closely related to other words. Often roast can relate to both foods that are baked and roasted, which makese sense for why it's often close to all the words.



5.

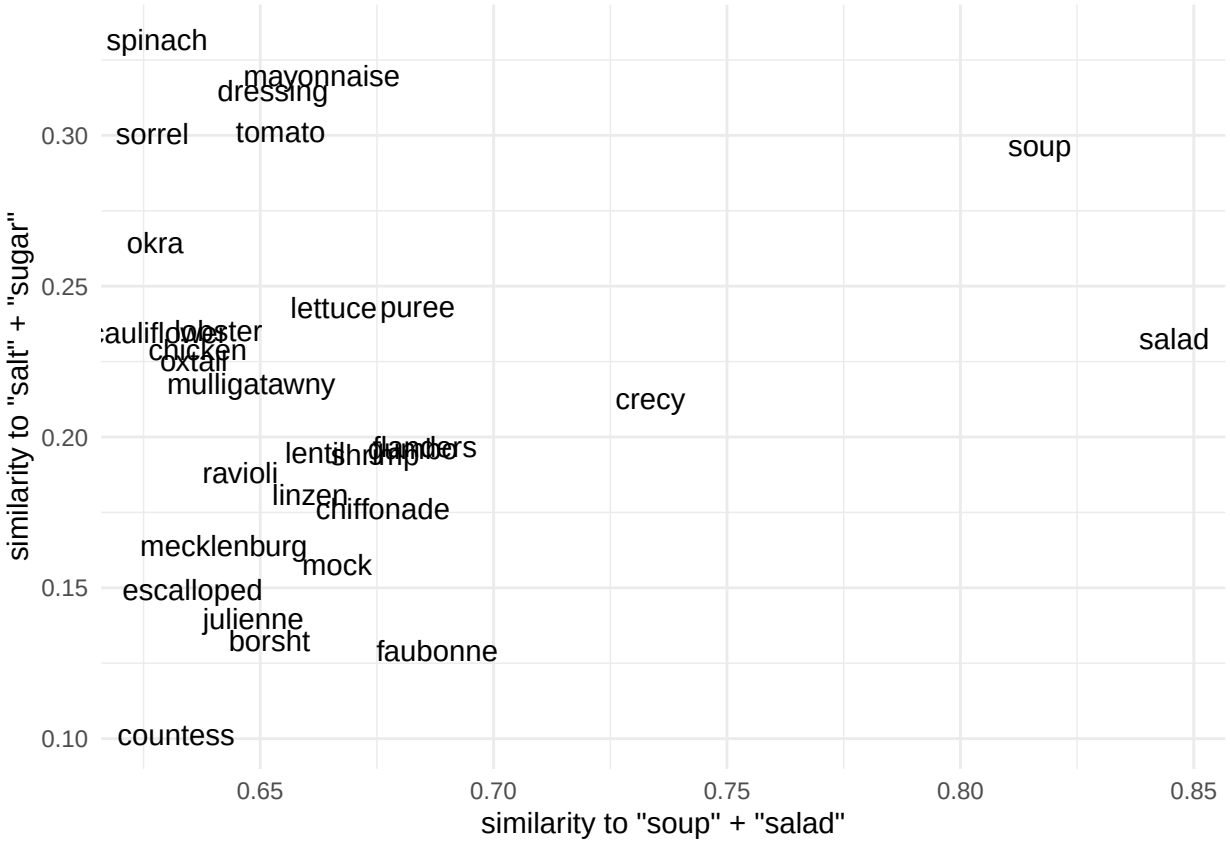
Here I try adding together **breakfast** and **dinner** as well as subtracting **eggs** and **toast**. We can see that for words closest to the sum of breakfast and dinner we see words we would expect like "super", "lunch", and "dessert". We also see words without a lot of similarity at all, however, like "feburary" and days of the week. These are perhaps close to these words because they are also related to time.

```
## Joining, by = "word"
```



I also performed with for the sums of salad and soup as they are related to the sums of salt and sugar. Similar relationships are seen in this plot.

```
## Joining, by = "word"
```



6.

1. **From the sum of breakfast and dinner we get other words related to time.** For instance we get months and days of the week. This is interesting to me because it is showing that these words have similar similaties to the sum of other words related to time.

2. **From the difference of eggs and toast we get other breakfast foods.** For instance we get foods like oatmeal and granola. This is interesting because it means the embedding is picking up that both of those foods are breakfast foods, and for some reason thinks that their difference is still breakfast food.

3. **Adding together non-similar words (like salt and sugar) get a small similarity and a lot of unrelated words.** Adding together those words we see no words that are similar to the two. Meaning that adding together two very different vectors isn't going to give us much information, which makes sense.

###7. The following code creates the data with bi-grams. The model trained but took a long time and I think there were some errors. Looking into the repo this came from for a bit, this appears to be a larger problem.

```
#if (!file.exists(paste0(DATA_DIR,"cookbooks_two.txt"))){ prep_word2vec(origin=paste0(DATA_DIR,"cookboo
```

```
#if (!file.exists(paste0(DATA_DIR,"cookbook_vectors_two.bin"))) {
#model_bigram = train_word2vec(paste0(DATA_DIR,"cookbooks_two.txt"), paste0(DATA_DIR,"cookbook_vectors_
#                        vectors=100,threads=4,window=6,
```

```
#                         min_count = 10,
#                         iter=5,negative_samples=15)
#} else{
#    model_bigram = read.vectors("cookbook_vectors_two.bin")
#}
```

How creating a model with pairs of words, here's an example with "stove top". The model couldn't be trained on my computer, but this would be the general process.

```
#model_bigram %>% closest_to(model_bigram[["noodles", "pho"]],11)
#model_bigram %>% closest_to(model_bigram[["chicken", "noodle"]],11)
```

Question 4 Gaussian Processes

The following code imports the data and creates a function for a Gaussian process. This is used the for answering the next set of questions.

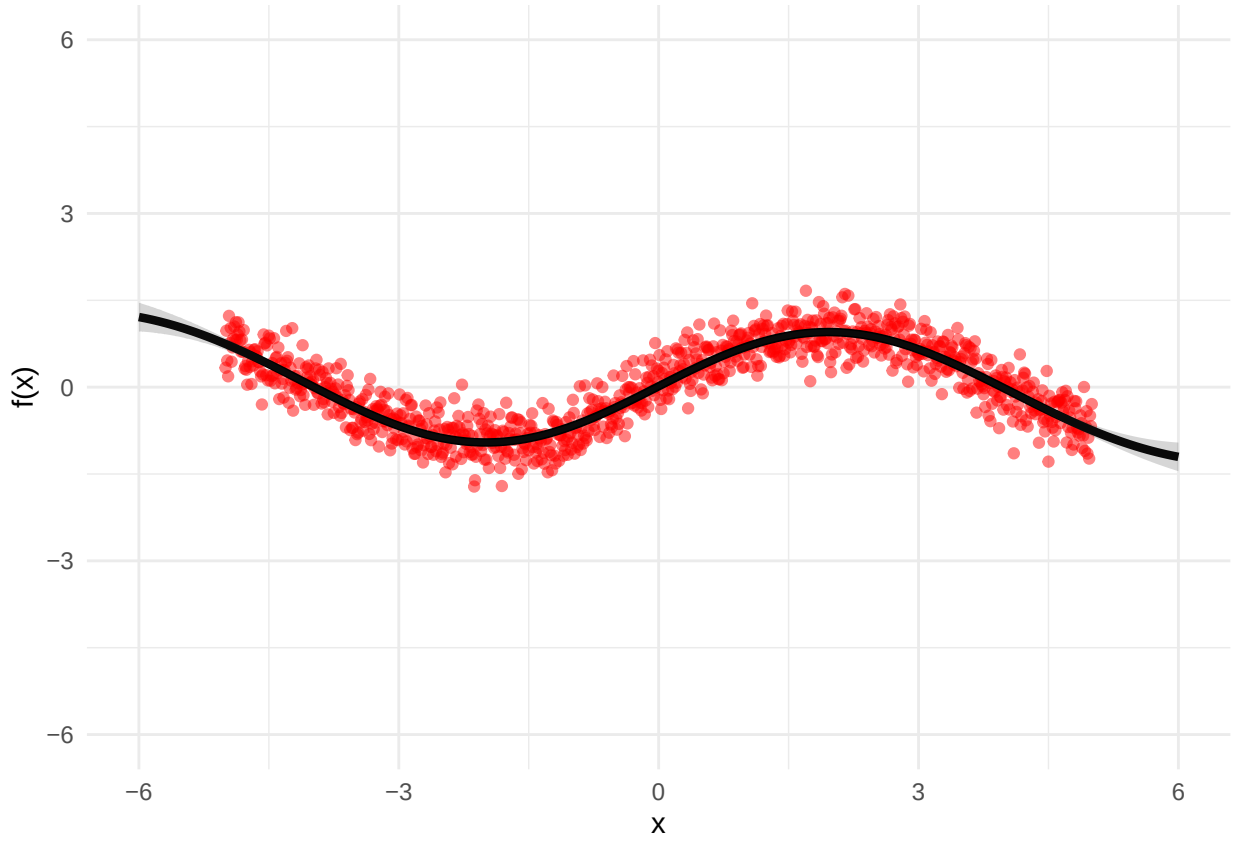
```
# import the data
kernel_reg <- read.csv(paste0(DATA_DIR, "kernel_regression_1.csv"))

# create a RBF covariance 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)))
}

# create function for questions ahead
gaussian_proc <- function(X, Y, l){

  # Setting up GP
  mu = mean(Y)
  mu_star = 0

  # also need to set up x values to do calcuation
  x_prime = seq(min(X)-1, max(X)+1, length.out = length(X))

  # Covariance of f -- add stochastic noise, which is variance of y
  K_f = K(kernel_reg$x, kernel_reg$x, l) + diag(var(Y), length(X))
  # Marginal and conditional covariance of f_star|f
  K_star = K(X,x_prime,l)
  K_starstar = K(x_prime,x_prime,l)

   # Conditional distribution of  f_star|f
  mu_star = mu_star + t(K_star) %*% solve(K_f) %*% (Y - mu)
  Sigma_star = K_starstar - t(K_star)%*% t(solve(K_f)) %*% K_star

  return(tibble(x = x_prime,
                y = mu_star %>% as.vector(),
                sd_prime = sqrt(diag(Sigma_star))) )
}
```

Part 1

1. Fit a Gaussian Model

Here is an example gaussian model and the resulting plot.

```
# example process with theta = 3
df_plot <- gaussian_proc(kernel_reg$x, kernel_reg$y, l = 3)

ggplot() +
   geom_point(data = kernel_reg, aes(x=x , y= y), color = 'red', alpha=0.5) +
   geom_line(data = df_plot, aes(x = x, y = y), size= 1.5) +
   geom_ribbon(data = df_plot, aes(x=x, y=y, ymin = y-sd_prime, ymax = y+sd_prime), alpha = 0.2) +
  xlim(c(-6,6))+ylim(c(-6,6))+ theme_minimal() +
  ylab('f(x)')
```

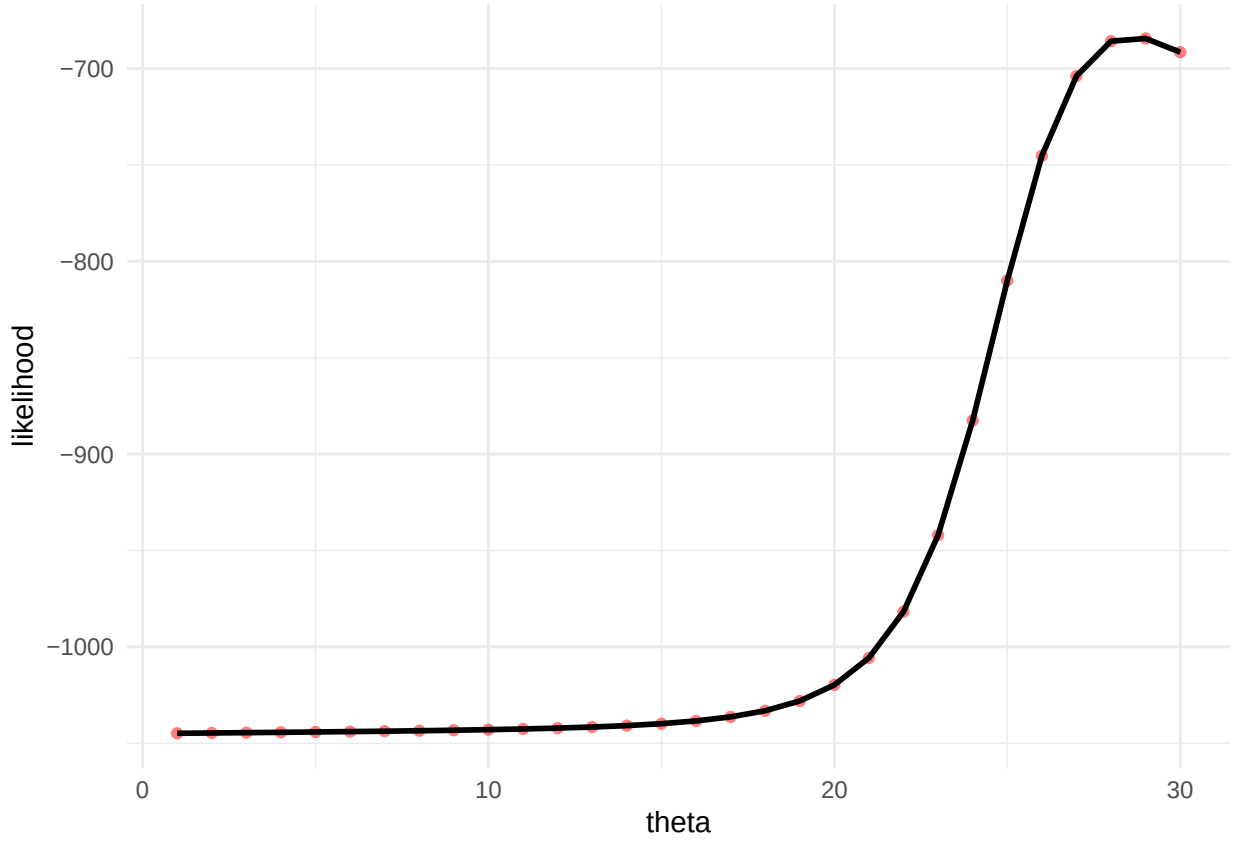


2. Find Optimal Theta

It appears the optimal Theta is around 20.

```
# function to calculate log likelihood for list of thetas
calculate_log_likelihoods <- function(X, Y, L){
  likelihoods = c()
  for (l in L){
    K_f = K(X, X, l) + diag(var(Y), length(X)) # need covariance with theta for likelihood calc
     log_likelihood = (-1/2) * Y %*% solve(K_f) %*% Y - (1/2) * log(det(K_f)) - (length(X)/2) * log(2*p
```

```
    log_likelihood
    likelihoods <- c(log_likelihood, likelihoods)
  }
  return(likelihoods)
}

likelihoods <- calculate_log_likelihoods(kernel_reg$x, kernel_reg$y, L = seq(1,30, by=1))

# plot likelihoods
data.frame("likelihood" = likelihoods, "theta" = seq(1,30, by=1)) %>%
  filter(! is.na(likelihoods)) %>%
ggplot() +
   geom_point(aes(x=theta , y= likelihood), color = 'red', alpha=0.5) +
   geom_line(aes(x=theta , y= likelihood), size= 1) + theme_minimal()
```

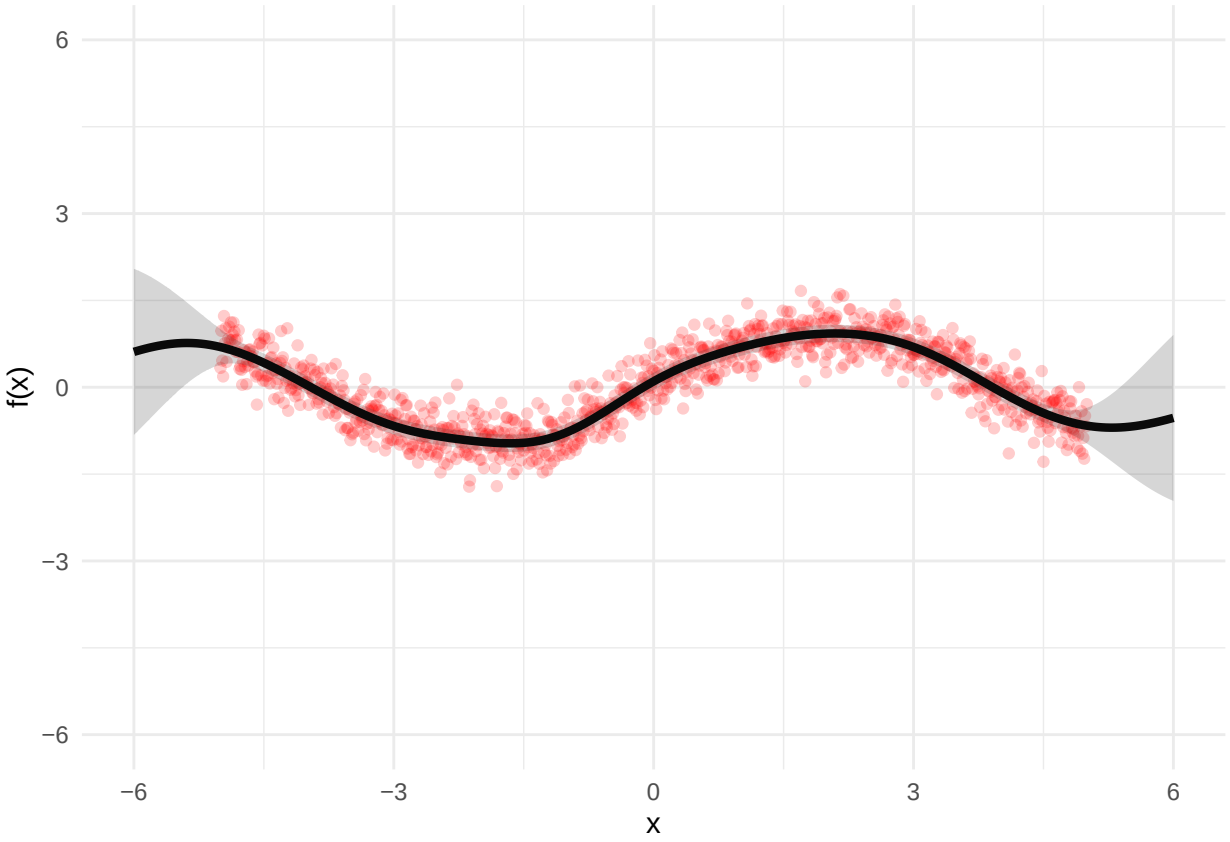


3. 95% CI

This can be done by just adjusting the range of sd_prime to be two standard deviations instead of one.

```
# example process with theta = 3
df_plot <- gaussian_proc(kernel_reg$x, kernel_reg$y, l = 1)

ggplot() +
   geom_point(data = kernel_reg, aes(x=x , y= y), color = 'red', alpha=0.2) +
   geom_line(data = df_plot, aes(x = x, y = y), size= 1.5) +
```

```
  geom_ribbon(data = df_plot, aes(x=x, y=y, ymin = y-(2*sd_prime), ymax = y+(2*sd_prime)), alpha = 0.2)
xlim(c(-6,6))+ylim(c(-6,6))+ theme_minimal() +
ylab('f(x)')
```



Part 2

Question 2 Time Series

[To Do]

Question 5 Building a Neural Network

This question is answered in `Question_5.ipynb`.

Question 6 Activation Functions

1. One Layer Neural Network

a. One Node

b. Two Nodes

1. Two Layer Neural Network

a. One Node

b. Two Nodes