Assignment3-Part2

Mengtong Zhang

4/3/2020

Question 3

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

```
if (!file.exists("cookbooks.zip")) {
  download.file("http://archive.lib.msu.edu/dinfo/feedingamerica/cookbook_text.zip","coo
kbooks.zip")
}
unzip("cookbooks.zip",exdir="cookbooks")
if (!file.exists("cookbooks.txt")) prep_word2vec(origin="cookbooks",destination="cookboo
ks.txt",lowercase=T,bundle_ngrams=1)

# 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")
    }
```

```
## Starting training using file /Users/zmt/Desktop/cookbooks.txt
## 100K
200K
300K
400K
500K
600K
700K
800K
900K
1000K
1100K
1200K
1300K
1400K
1500K
1600K
1700K
1800K
1900K
2000K
2100K
2200K
2300K
2400K
2500K
2600K
2700K
2800K
2900K
3000K
3100K
3200K
3300K
3400K
3500K
3600K
3700K
3800K
3900K
4000K
4100K
4200K
4300K
4400K
4500K
4600K
4700K
4800K
4900K
5000K
5100K
5200K
```

```
5300K
5400K
5500K
5600K
5700K
5800K
5900K
6000K
6100K
6200K
6300K
6400K
6500K
6600K
6700K
6800K
6900K
7000K
7100K
7200K
7300K
7400K
7500K
7600K
7700K
7800K
7900K
8000K
8100K
8200K
8300K
8400K
8500K
8600K
8700K
8800K
8900K
9000K
9100K
9200K
9300K
9400K
9500K
9600K
9700K
9800K
9900K
10000K
10100K
10200K
10300K
10400K
Vocab size: 18952
## Words in train file: 10304635
```

```
##
  |
  |                                                                      |   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%
```

1.
To improve the quality of embedding, I would do stop words removal, stemming or lemmatization and text normalization.

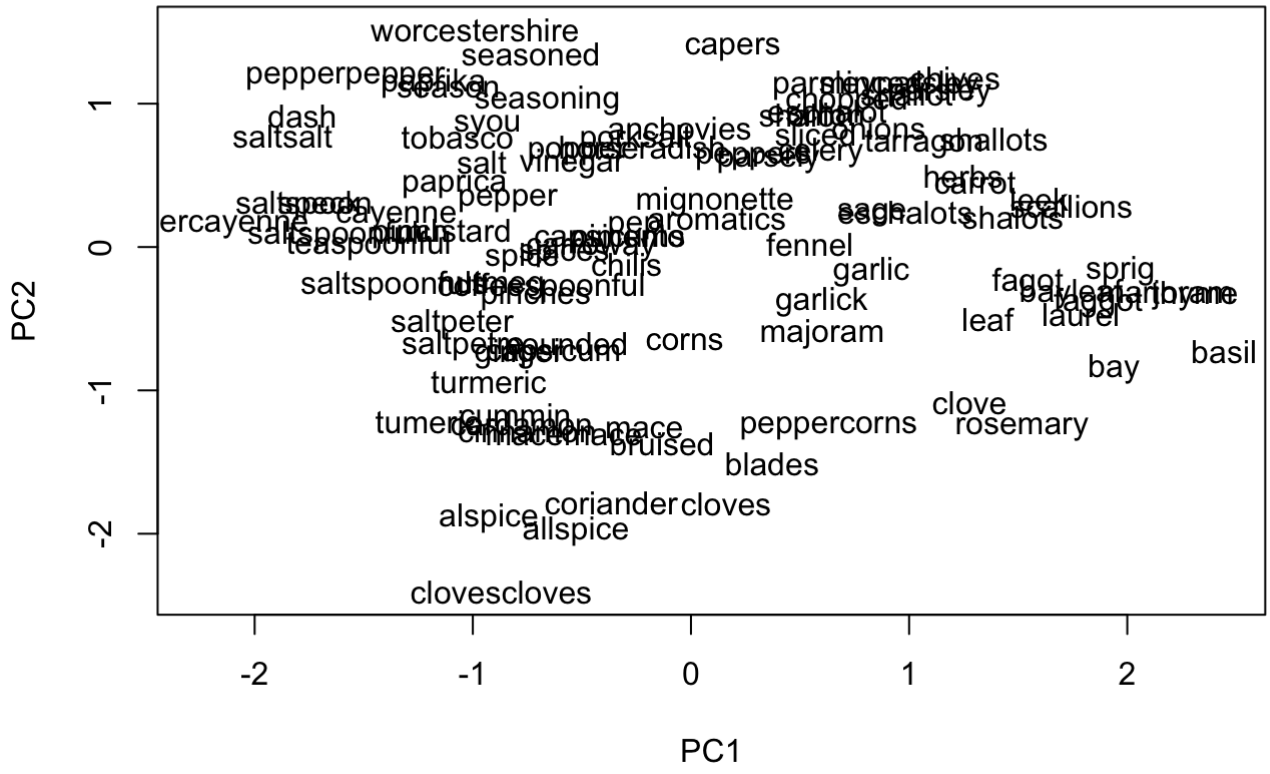
2.
I choose ‘salt’,‘pepper’ and ‘garlic’ as three ingresients. And pepper, salt, cayenne, garlic and maca are top 5 similar ingredients.

```
# -- Select ingredient and cuisine --
ingredient = 'salt'
ingredient_2 = 'pepper'
ingredient_3 = 'garlic'
list_of_ingredients = c(ingredient, ingredient_2, ingredient_3)
# Set of closest words to "sage", "thyme","basil"
model %>% closest_to(model[[list_of_ingredients]],5)
```

```
##      word similarity to model[[list_of_ingredients]]
## 1  pepper                                    0.9007534
## 2    salt                                    0.8565778
## 3 cayenne                                    0.8440435
## 4  garlic                                    0.8355527
## 5    mace                                    0.7898229
```

3.
Here we use t-SNE to see the relationships between set of words related with the three ingredients in question2.

```
n_words = 100
closest_ingredients = closest_to(model,model[[list_of_ingredients]], n_words)$word
surrounding_ingredients = model[[closest_ingredients,average=F]]
plot(surrounding_ingredients,method="pca")
```




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

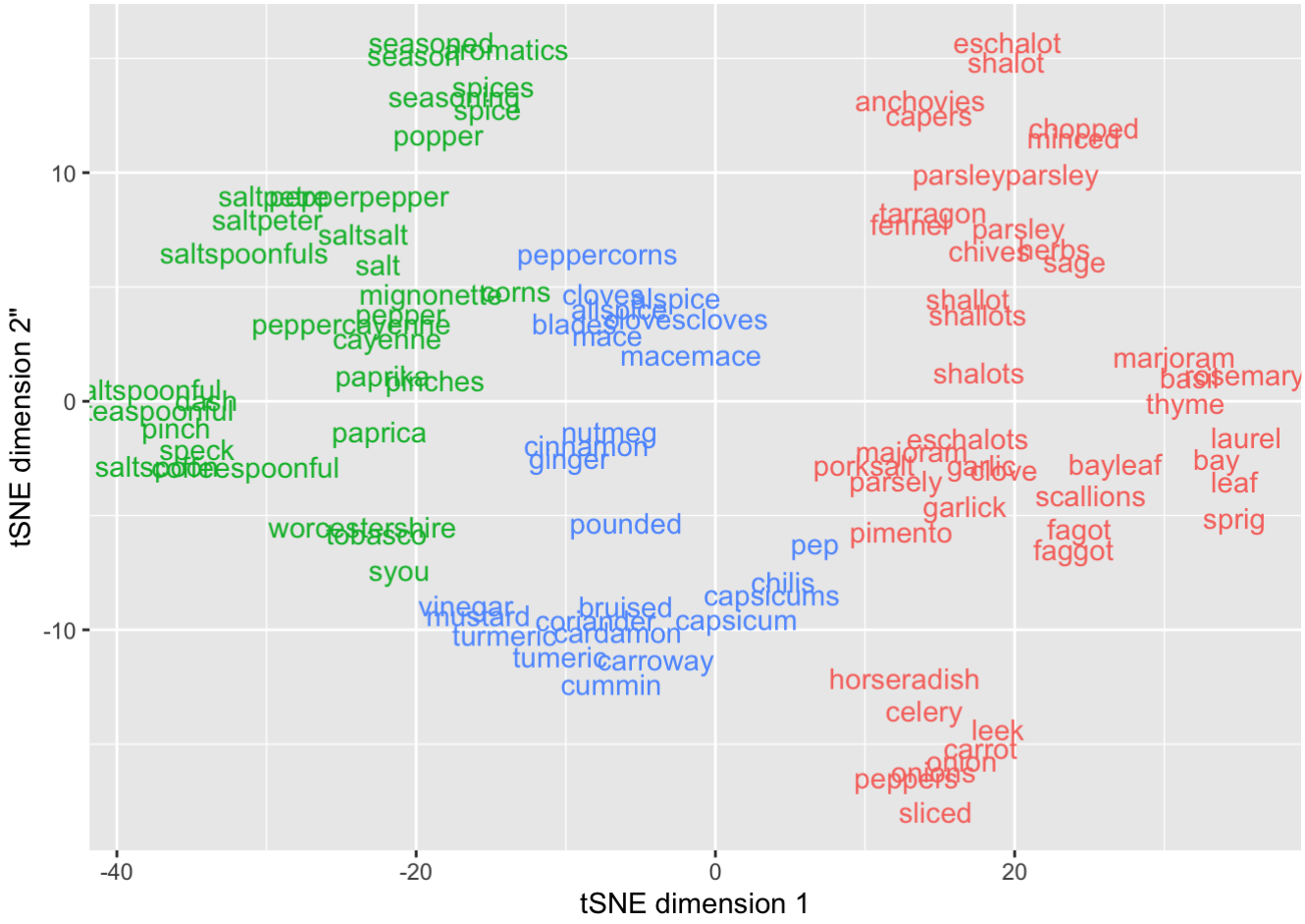
```
## Performing PCA
## Read the 100 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 4.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## Done in 0.00 seconds (sparsity = 0.172200)!
## Learning embedding...
## Iteration 50: error is 65.158023 (50 iterations in 0.01 seconds)
## Iteration 100: error is 62.558651 (50 iterations in 0.01 seconds)
## Iteration 150: error is 62.194715 (50 iterations in 0.01 seconds)
## Iteration 200: error is 62.192894 (50 iterations in 0.01 seconds)
## Iteration 250: error is 62.192339 (50 iterations in 0.01 seconds)
## Iteration 300: error is 1.079984 (50 iterations in 0.01 seconds)
## Iteration 350: error is 0.877541 (50 iterations in 0.01 seconds)
## Iteration 400: error is 0.814524 (50 iterations in 0.01 seconds)
## Iteration 450: error is 0.789361 (50 iterations in 0.01 seconds)
## Iteration 500: error is 0.776020 (50 iterations in 0.01 seconds)
## Iteration 550: error is 0.764670 (50 iterations in 0.01 seconds)
## Iteration 600: error is 0.756756 (50 iterations in 0.01 seconds)
## Iteration 650: error is 0.751964 (50 iterations in 0.01 seconds)
## Iteration 700: error is 0.748700 (50 iterations in 0.01 seconds)
## Iteration 750: error is 0.744961 (50 iterations in 0.01 seconds)
## Iteration 800: error is 0.740787 (50 iterations in 0.01 seconds)
## Iteration 850: error is 0.738726 (50 iterations in 0.01 seconds)
## Iteration 900: error is 0.737807 (50 iterations in 0.01 seconds)
## Iteration 950: error is 0.733280 (50 iterations in 0.01 seconds)
## Iteration 1000: error is 0.730007 (50 iterations in 0.01 seconds)
## Iteration 1050: error is 0.730017 (50 iterations in 0.01 seconds)
## Iteration 1100: error is 0.729868 (50 iterations in 0.01 seconds)
## Iteration 1150: error is 0.728265 (50 iterations in 0.01 seconds)
## Iteration 1200: error is 0.725698 (50 iterations in 0.01 seconds)
## Iteration 1250: error is 0.725375 (50 iterations in 0.01 seconds)
## Iteration 1300: error is 0.725193 (50 iterations in 0.01 seconds)
## Iteration 1350: error is 0.724029 (50 iterations in 0.01 seconds)
## Iteration 1400: error is 0.722758 (50 iterations in 0.01 seconds)
## Iteration 1450: error is 0.723657 (50 iterations in 0.01 seconds)
## Iteration 1500: error is 0.723741 (50 iterations in 0.01 seconds)
## Iteration 1550: error is 0.723508 (50 iterations in 0.01 seconds)
## Iteration 1600: error is 0.722337 (50 iterations in 0.01 seconds)
## Iteration 1650: error is 0.721975 (50 iterations in 0.01 seconds)
## Iteration 1700: error is 0.722479 (50 iterations in 0.01 seconds)
## Iteration 1750: error is 0.723277 (50 iterations in 0.01 seconds)
## Iteration 1800: error is 0.720843 (50 iterations in 0.01 seconds)
## Iteration 1850: error is 0.722535 (50 iterations in 0.01 seconds)
## Iteration 1900: error is 0.721761 (50 iterations in 0.01 seconds)
## Iteration 1950: error is 0.722125 (50 iterations in 0.01 seconds)
## Iteration 2000: error is 0.720295 (50 iterations in 0.01 seconds)
## Fitting performed in 0.33 seconds.
```

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

```
embedding_vals = embedding$Y
rownames(embedding_vals) = rownames(surrounding_ingredients)
set.seed(10)
n_centers = 3
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')
```



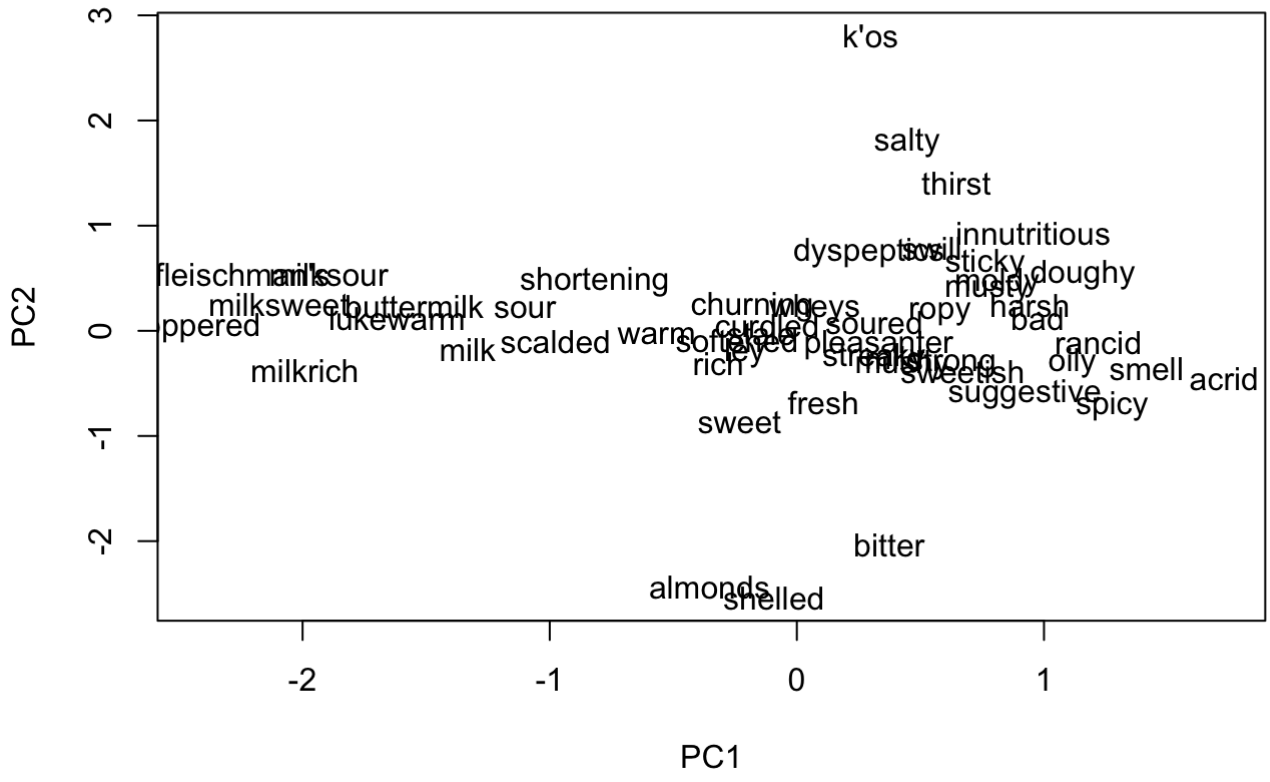

```
# Topics produced by the top 3 words
sapply(sample(1:n_centers,n_centers),function(n) {
  names(clustering$cluster[clustering$cluster==n][1:10])
})
```

```
##       [,1]       [,2]          [,3]
##  [1,] "vinegar"  "salt"        "chopped"
##  [2,] "nutmeg"   "pepper"      "parsley"
##  [3,] "cinnamon" "teaspoonful" "onion"
##  [4,] "cloves"   "season"      "onions"
##  [5,] "mustard"  "cayenne"     "celery"
##  [6,] "mace"     "pinch"       "sliced"
##  [7,] "ginger"   "seasoning"   "herbs"
##  [8,] "pounded"  "seasoned"    "minced"
##  [9,] "allspice" "spices"      "bay"
## [10,] "bruised"  "spice"       "leaf"
```

4.

Here I replace three ingredients by three tastes: ‘sweet’, ‘sour’ and ‘bitter’.

```
list_of_ingredients = c('salty','sour','bitter')
closest_ingredients = closest_to(model,model[[list_of_ingredients]], 50)$word
surrounding_ingredients = model[[closest_ingredients,average=F]]
plot(surrounding_ingredients,method="pca")
```



Not quite make sense. Because there are supposed to be 3 obvious different clusters. Then we perform t-SNE.

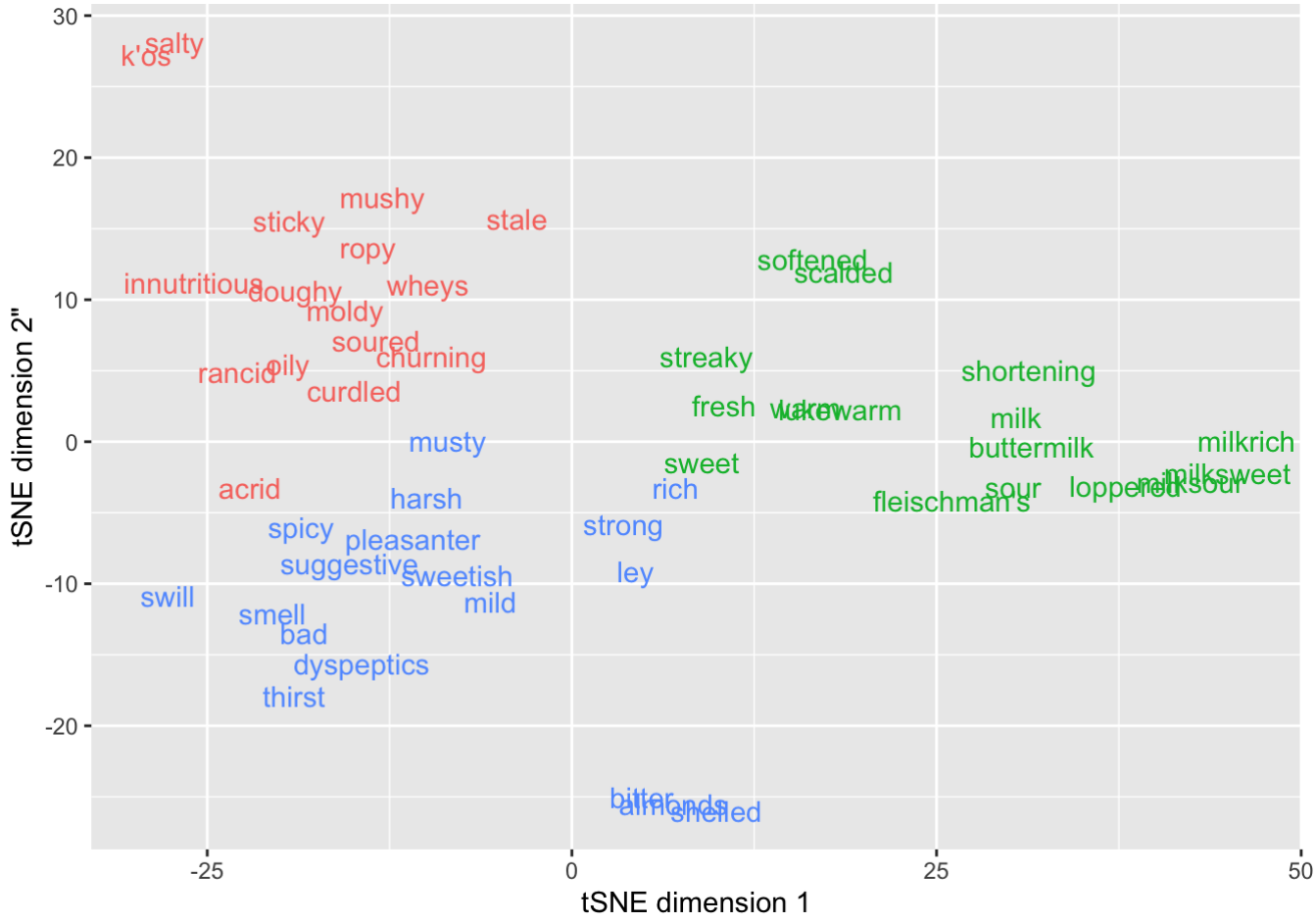
```
embedding = Rtsne(X = surrounding_ingredients, dims = 2,
                  perplexity = 4,
                  theta = 0.5,
                  eta = 10,
                  pca = TRUE, verbose = TRUE,
                  max_iter = 2000)
```

```
## Performing PCA
## Read the 50 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 = 0.358400)!
## Learning embedding...
## Iteration 50: error is 56.443514 (50 iterations in 0.01 seconds)
## Iteration 100: error is 56.183694 (50 iterations in 0.01 seconds)
## Iteration 150: error is 56.183581 (50 iterations in 0.00 seconds)
## Iteration 200: error is 56.183590 (50 iterations in 0.00 seconds)
## Iteration 250: error is 56.183597 (50 iterations in 0.00 seconds)
## Iteration 300: error is 0.710225 (50 iterations in 0.00 seconds)
## Iteration 350: error is 0.659776 (50 iterations in 0.00 seconds)
## Iteration 400: error is 0.644492 (50 iterations in 0.00 seconds)
## Iteration 450: error is 0.642430 (50 iterations in 0.00 seconds)
## Iteration 500: error is 0.636172 (50 iterations in 0.00 seconds)
## Iteration 550: error is 0.635768 (50 iterations in 0.00 seconds)
## Iteration 600: error is 0.628761 (50 iterations in 0.00 seconds)
## Iteration 650: error is 0.629706 (50 iterations in 0.00 seconds)
## Iteration 700: error is 0.627427 (50 iterations in 0.00 seconds)
## Iteration 750: error is 0.626906 (50 iterations in 0.00 seconds)
## Iteration 800: error is 0.627323 (50 iterations in 0.00 seconds)
## Iteration 850: error is 0.623582 (50 iterations in 0.00 seconds)
## Iteration 900: error is 0.624616 (50 iterations in 0.00 seconds)
## Iteration 950: error is 0.621263 (50 iterations in 0.00 seconds)
## Iteration 1000: error is 0.622416 (50 iterations in 0.00 seconds)
## Iteration 1050: error is 0.619942 (50 iterations in 0.00 seconds)
## Iteration 1100: error is 0.618914 (50 iterations in 0.00 seconds)
## Iteration 1150: error is 0.617018 (50 iterations in 0.00 seconds)
## Iteration 1200: error is 0.618047 (50 iterations in 0.00 seconds)
## Iteration 1250: error is 0.615848 (50 iterations in 0.00 seconds)
## Iteration 1300: error is 0.615204 (50 iterations in 0.00 seconds)
## Iteration 1350: error is 0.613697 (50 iterations in 0.00 seconds)
## Iteration 1400: error is 0.618941 (50 iterations in 0.00 seconds)
## Iteration 1450: error is 0.619128 (50 iterations in 0.00 seconds)
## Iteration 1500: error is 0.615199 (50 iterations in 0.00 seconds)
## Iteration 1550: error is 0.612338 (50 iterations in 0.01 seconds)
## Iteration 1600: error is 0.612665 (50 iterations in 0.00 seconds)
## Iteration 1650: error is 0.610929 (50 iterations in 0.00 seconds)
## Iteration 1700: error is 0.611216 (50 iterations in 0.00 seconds)
## Iteration 1750: error is 0.609664 (50 iterations in 0.00 seconds)
## Iteration 1800: error is 0.607684 (50 iterations in 0.00 seconds)
## Iteration 1850: error is 0.609559 (50 iterations in 0.00 seconds)
## Iteration 1900: error is 0.613312 (50 iterations in 0.00 seconds)
## Iteration 1950: error is 0.615322 (50 iterations in 0.00 seconds)
## Iteration 2000: error is 0.613586 (50 iterations in 0.00 seconds)
## Fitting performed in 0.16 seconds.
```

```
embedding_vals = embedding$Y
rownames(embedding_vals) = rownames(surrounding_ingredients)
set.seed(10)
n_centers = 3
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')
```




```
# Topics produced by the top 3 words
sapply(sample(1:n_centers,n_centers),function(n) {
  names(clustering$cluster[clustering$cluster==3][1:10])
})
```

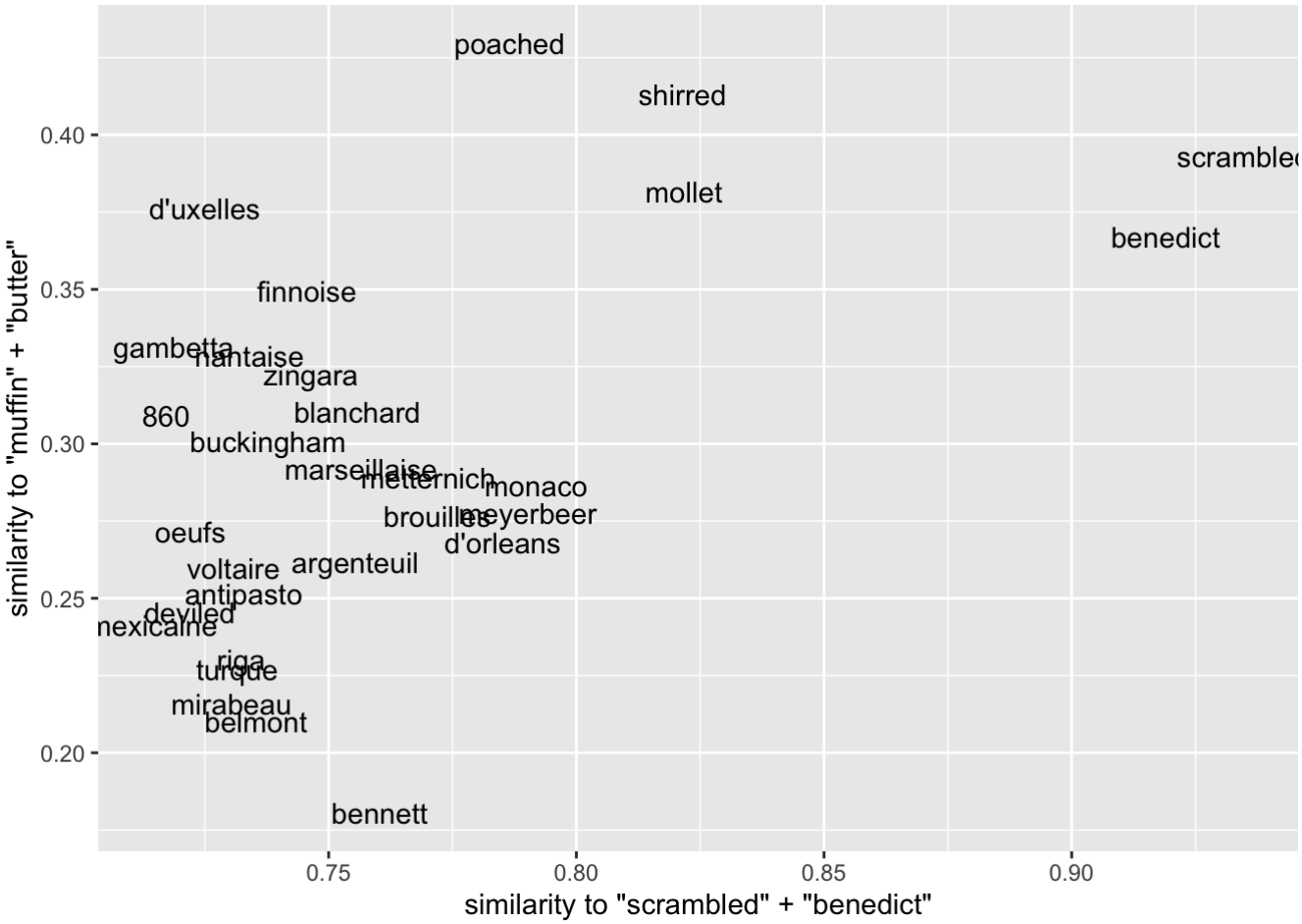
```
##       [,1]      [,2]      [,3]
##  [1,] "rich"    "rich"    "rich"
##  [2,] "almonds" "almonds" "almonds"
##  [3,] "strong"  "strong"  "strong"
##  [4,] "bitter"  "bitter"  "bitter"
##  [5,] "bad"     "bad"     "bad"
##  [6,] "shelled" "shelled" "shelled"
##  [7,] "smell"   "smell"   "smell"
##  [8,] "mild"    "mild"    "mild"
##  [9,] "thirst"  "thirst"  "thirst"
## [10,] "musty"   "musty"   "musty"
```

5.

```
top_evaluative_words = model %>%
  closest_to(~ "scrambled"+"benedict",n=30)
goodness = model %>%
  closest_to(~ "scrambled"-"benedict",n=Inf)
taste = model %>%
  closest_to(~ "muffin" + "butter", n=Inf)

top_evaluative_words %>%
  inner_join(goodness) %>%
  inner_join(taste) %>%
  ggplot() +
  geom_text(aes(x=`similarity to "scrambled" + "benedict"`,
                y=`similarity to "muffin" + "butter"`,
                label=word))
```

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



6.

I found that:

– belmont has no obvious relation with scrambled+benedict and muffin+butter

– pouched has obvious relation with scrambled+benedict, which is interesting and quite make sense for me because usually egg benedict goes with pouched egg

– duxelles has relatively obvious relation with muffin+butter, which probably because it usually goes with butter

7.

Here we need the package ngram.

```
library(ngram)
text = readLines('/Users/zmt/Desktop/cookbooks.txt')
txt = ''
for (i in text){txt = concatenate(txt,i)}
```

```
txt = preprocess (txt ,case ="lower", remove.punct = TRUE )
```

```
ng <- ngram (txt , n =2)
```

The list is as shown below:

```
cat('Top ten are','\n')
```

```
## Top ten are
```

```
get.phrasetable(ng)[1:10,]
```

```
##       ngrams  freq        prop
## 1    of the  61141 0.005835815
## 2    in the  46763 0.004463457
## 3      in a  45059 0.004300812
## 4    with a  29628 0.002827947
## 5    to the  24058 0.002296299
## 6     it is  23983 0.002289141
## 7  a little  23460 0.002239221
## 8      of a  22043 0.002103970
## 9  with the  20208 0.001928823
## 10    and a  20000 0.001908969
```

Question 4

Part1

1. Here we choose $\theta = 1$ for the first fitting process.

```
dat = read.csv('/Users/zmt/Desktop/kernel_regression_1.csv')
set.seed(129)
idx = 990:1001
x_observed = dat$x[-idx]
f = dat$y[-idx]
x_prime = dat$x
K = function(x,x_prime,l){
  d = sapply(x,FUN=function(x_in)(x_in-x_prime)^2)
  return(t(exp(-1/(2*l)*d)))
}
mu=0
mu_star=0
l=10
K_f = K(x_observed,x_observed,l)
for (i in 1:dim(K_f)[1]){K_f[i,i]=K_f[i,i]+0.000001}
K_star = K(x_observed,x_prime,l)
K_starstar = K(x_prime,x_prime,l)
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
```

2. Note that here we only take the kernel of log likelihood (So it can be positive).

$$L(y_1, \ldots, y_n) = \frac{1}{(2\pi)^n |\Sigma|} e^{-(y-\mu)^T \Sigma^{-1} (y-\mu)}$$

$$logLL \propto -log(|\Sigma|) - (y-\mu)^T \Sigma^{-1} (y-\mu)$$

```
for (ll in c(1,0.1,0.01,0.00001)){
mu=0
mu_star=0
l=ll
K_f = K(x_observed,x_observed,l)
# Add little perbutation to make K_f inversable
for (i in 1:dim(K_f)[1]){K_f[i,i]=K_f[i,i]+0.000001}
K_star = K(x_observed,x_prime,l)
K_starstar = K(x_prime,x_prime,l)
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
Sigma_star_test = Sigma_star[idx,idx]
#for (i in 1:dim(Sigma_star)[1]){Sigma_star[i,i]=Sigma_star[i,i]+1}
# Here only take the kernel of likelihood
#logLL = log(dmvnorm(dat$y[idx],mean = mu_star[idx], sigma=Sigma_star_test))
logLL = -log(det(Sigma_star_test)) - t(dat$y[idx]-mu_star[idx])%*%solve(Sigma_star_tes
t)%*%(dat$y[idx]-mu_star[idx])
cat('theta=',ll,'Negative Log Likelihood is',-logLL,'\n')}
```

```
## Warning in log(det(Sigma_star_test)): NaNs produced
```

```
## theta= 1 Negative Log Likelihood is NaN
```

```
## Warning in log(det(Sigma_star_test)): NaNs produced
```

```
## theta= 0.1 Negative Log Likelihood is NaN
```

```
## Warning in log(det(Sigma_star_test)): NaNs produced
```

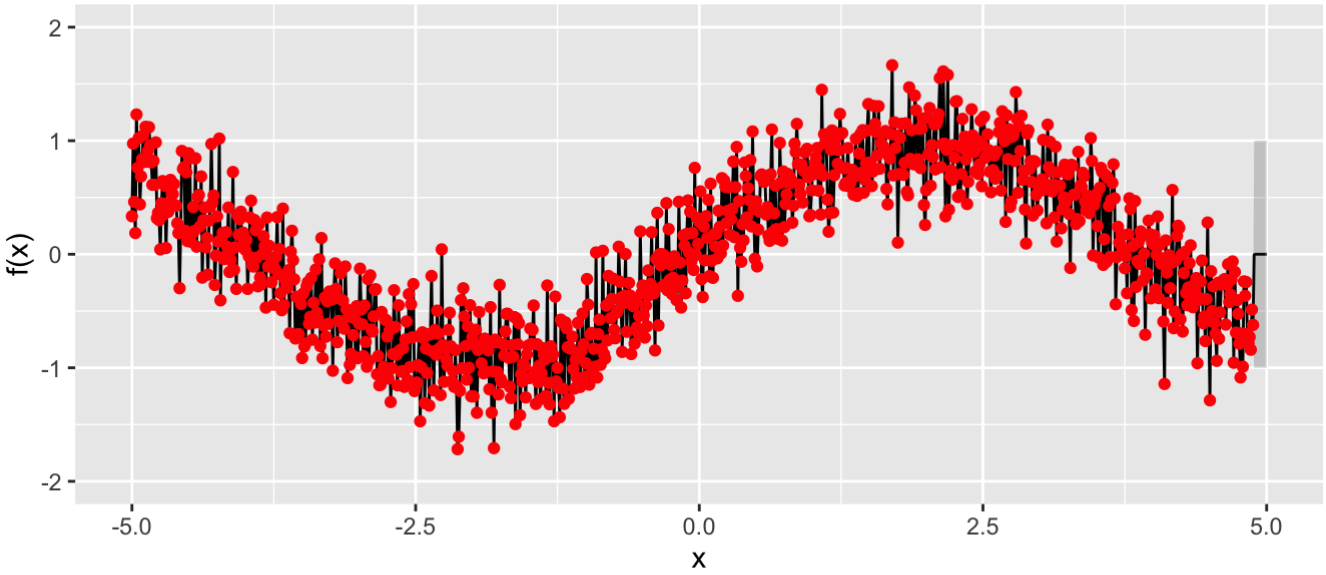
```
## theta= 0.01 Negative Log Likelihood is NaN
## theta= 1e-05 Negative Log Likelihood is 7.944166
```

$\theta = 1$ is the best choice.

3.

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

# Plotting values
ggplot(aes(x = x, y = y), data = plot_gp) +
  geom_line()+
  geom_ribbon(aes(ymin = y-sd_prime,ymax = y+sd_prime), alpha = 0.2)+
  geom_point(aes(x =x , y= y), data = tibble(x = x_observed, y = f),
             color = 'red') +
  xlim(c(-5,5))+ylim(c(-2,2))+
  coord_fixed(ratio = 1) +ylab('f(x)')
```



Part 2

1.

```
dat = read.csv('/Users/zmt/Desktop/kernel_regression_2.csv')
set.seed(129)
idx = 990:1001
x_observed = dat$x[-idx]
f = dat$z[-idx]
x_prime = dat$x
K = function(x,x_prime,l){
  d = sapply(x,FUN=function(x_in)(x_in-x_prime)^2)
  return(t(exp(-1/(2*l)*d)))
}
mu=0
mu_star=0
l=10
K_f = K(x_observed,x_observed,l)
for (i in 1:dim(K_f)[1]){K_f[i,i]=K_f[i,i]+0.000001}
K_star = K(x_observed,x_prime,l)
K_starstar = K(x_prime,x_prime,l)
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
```

2.

```
for (ll in c(0.1,0.01,0.00001)){
mu=0
mu_star=0
l=ll
K_f = K(x_observed,x_observed,l)
# Add little perbutation to make K_f inversable
for (i in 1:dim(K_f)[1]){K_f[i,i]=K_f[i,i]+0.000001}
K_star = K(x_observed,x_prime,l)
K_starstar = K(x_prime,x_prime,l)
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
Sigma_star_test = Sigma_star[idx,idx]
logLL = -log(det(Sigma_star_test)) - t(dat$y[idx]-mu_star[idx])%*%solve(Sigma_star_tes
t)%*%(dat$y[idx]-mu_star[idx])
cat('theta=',ll,'Negative Log Likelihood is',-logLL,'\n')}
```

```
## theta= 0.1 Negative Log Likelihood is 1960834402
## theta= 0.01 Negative Log Likelihood is 6756231175
## theta= 1e-05 Negative Log Likelihood is 8570229749
```

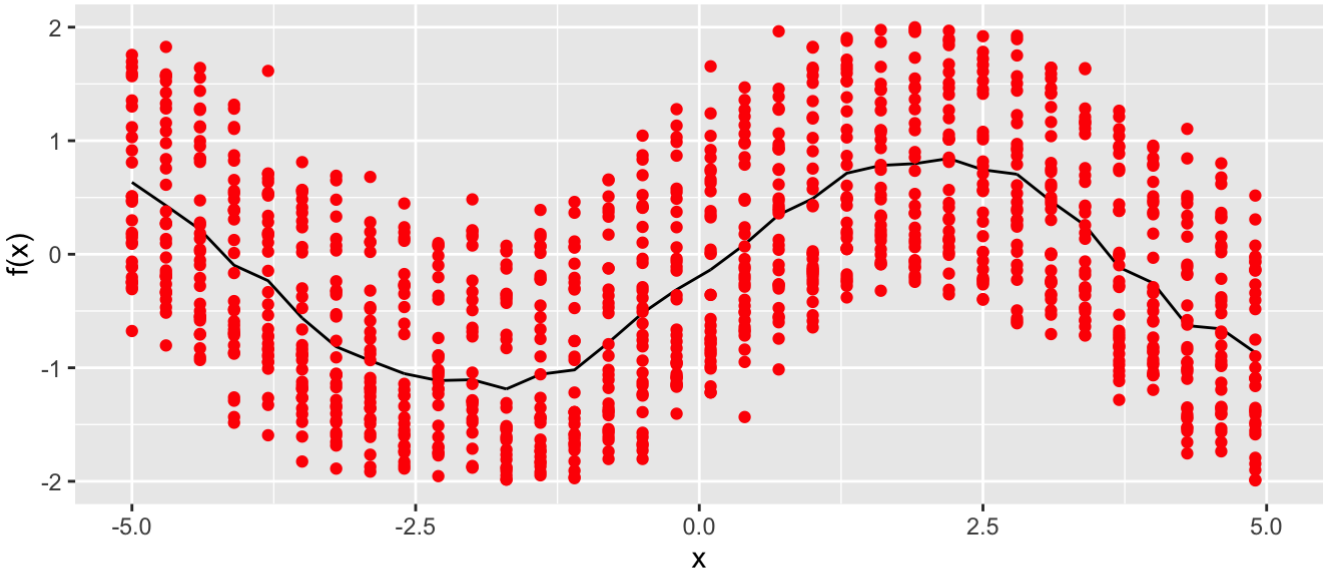
$\theta = 0.1$ is the best choice.

3.

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

# Plotting values
ggplot(aes(x = x, y = y), data = plot_gp) +
  geom_line()+
  geom_ribbon(aes(ymin = y-sd_prime,ymax = y+sd_prime), alpha = 0.2)+
  geom_point(aes(x =x , y= y), data = tibble(x = x_observed, y = f),
             color = 'red') +
  xlim(c(-5,5))+ylim(c(-2,2))+
  coord_fixed(ratio = 1) +ylab('f(x)')
```

```
## Warning: Removed 38 rows containing missing values (geom_point).
```


2
1
0
-1
-2
f(x)
-5.0
-2.5
0.0
2.5
5.0
x