Predicting Recipe Type Based on Directions

Set Up and Introduction

In this assignment we are going to be looking at a data set of pie, cookie, and cake recipes and trying to determine if we can predict what type of recipe it is based on the words used in the directions of each recipe. If a person were to read the directions without knowing the type of recipe, there is a good chance they would be able to determine what type it is based on context, but the analysis of individual words will allow us to ask, are there certain "buzz-words" or methods used in different baking recipes that determine what kind of good is produced? Can we create an algorithm that can predict what type it is based on the main words used? We will first analyze the data, then create a general prediction algorithm. Finally, we will tune this algorithm in different ways to figure out how we can create the best supervised learning algorithm while also making sure we are making *helpful* predictions (not just using words that make obvious predictions and do not tell us about our data set).

To start, we will set up our data set.

```
library(caret)
```

```
Loading required package: ggplot2
```

```
Loading required package: lattice
```

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v lubridate 1.9.3     v tibble    3.2.1
v purrr     1.0.2     v tidyr     1.3.1
```

```
-- Conflicts --------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
x purrr::lift()   masks caret::lift()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
library(tidytext)
library(stringr)

recipes <- read_csv("recipes.csv")
```

```
New names:
Rows: 1090 Columns: 15
-- Column specification
-------------------------------------------------------- Delimiter: "," chr
(12): recipe_name, prep_time, cook_time, total_time, yield, ingredients,... dbl
(3): ...1, servings, rating
i Use `spec()` to retrieve the full column specification for this data. i
Specify the column types or set `show_col_types = FALSE` to quiet this message.
* `` -> `...1`
```

```
recipes <- recipes |>
  select(recipe_name, ingredients, directions, rating)

recipes_cut <- recipes |>
    mutate(type = case_when(grepl("cake", recipe_name, ignore.case = TRUE) ~ "cake",
                            grepl("pie", recipe_name, ignore.case = TRUE) ~ "pie",
                            grepl("cookie", recipe_name, ignore.case = TRUE) ~ "cookie",
                            TRUE ~ "other"))

recipes_cut <- recipes_cut |>
  filter(type != "other")
```

The imported data set is a data frame of all types of recipes with their ingredients, directions, rating, and more. To make this data useful for our context, we will create a new column called `type` and use `stringr` to determine whether the recipe is for a pie, cookie, cake, or other. Finally, we filter out all of the `other` recipes to get a data set with just the recipes we want to look at.

Unsupervised Learning

The next step is to analyze and try to understand our data using unsupervised learning models. This will also help us tune our supervised learning algorithm for later.

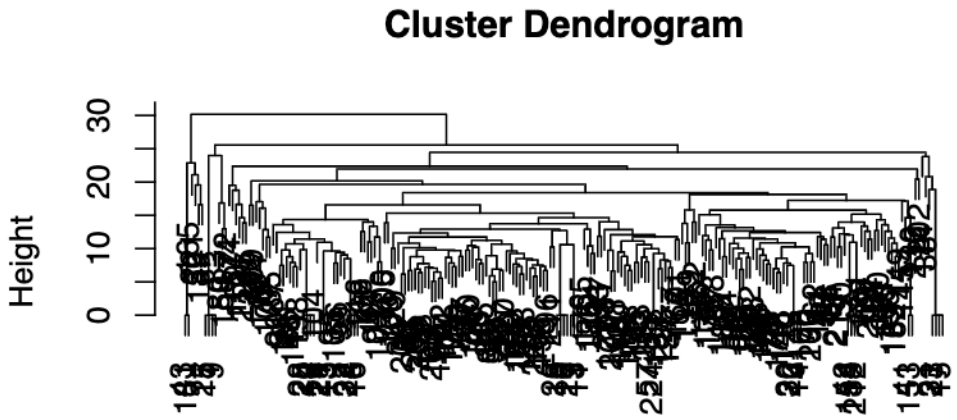
First, we will make a data set that has a row for each word, then set up a document feature matrix for our data.

```
recipes_words <- recipes_cut |>
  mutate(recipe_id = row_number()) |>
  unnest_tokens(input = "directions",
                output = "Word",
                token = "words") |>
  filter(!(Word %in% stop_words$word))

dfm <- recipes_words |>
  pivot_wider(id_cols = c("recipe_id", "recipe_name"),
              names_from = "Word",
              values_from = "Word",
              values_fn = length,
              values_fill = 0)
```

Hierarchical Clustering

```
dfm |>
  dplyr::select(-recipe_id, -recipe_name) |>
  dist() |>
  hclust() |>
  plot()
```




```
recipes_cut |>
  slice(42,154)
```

```
# A tibble: 2 x 5
  recipe_name                                    ingredients directions rating type
  <chr>                                          <chr>       <chr>       <dbl> <chr>
1 Grandma's Skillet Pineapple Upside-Down C~ 1 stick bu~ "Preheat ~    4.7 cake
2 Grandma's Skillet Pineapple Upside-Down C~ 1 stick bu~ "Preheat ~    4.7 cake
```

Initially looking at this dendrogram, we can see that there are a few different clusters of recipes. There are a few recipes that are at height zero, and after looking closer at these specific recipes, it turns out they are actually duplicates. We will want to remove these from our data set. Let's also form clusters by cutting at different heights to see what we can find.

```
recipes_cut2 <- recipes_cut |>
  distinct()

recipes_words2 <- recipes_cut2 |>
  mutate(recipe_id = row_number()) |>
  unnest_tokens(input = "directions",
                output = "Word",
                token = "words") |>
```

```
  filter(!(Word %in% stop_words$word))

dfm2 <- recipes_words2 |>
  pivot_wider(id_cols = c("recipe_id", "recipe_name"),
              names_from = "Word",
              values_from = "Word",
              values_fn = length,
              values_fill = 0)

height <- 20

hclustering <- dfm2 |>
  dplyr::select(-recipe_id, -recipe_name) |>
  dist() |>
  hclust()

clusters <- cutree(hclustering, h = height)

dfm2 |>
  dplyr::select(-recipe_id, -recipe_name) |>
  mutate(cluster = clusters) |>
  group_by(cluster) |>
  summarize_all(mean)
```

```
# A tibble: 13 x 1,108
   cluster   peel    core apples  thinly  slice   set preheat  oven  `425`
     <int>  <dbl>   <dbl>  <dbl>   <dbl>  <dbl> <dbl>   <dbl> <dbl>  <dbl>
 1       1 1      1       5      1       1      1       1      2     1
 2       2 0.0347 0.00578 0.0983 0.00578 0.0347 0.329   0.815  1.80  0.0636
 3       3 0      0       4      0       0      0       1      3.5   0
 4       4 0      0       0.333  0.167   0.667  0.833   0.5    1.33  0.167
 5       5 0      0       0      0       2      0       1      2     0
 6       6 0      0       0      0       0.333  1       1      1.67  0
 7       7 0      0       0      0       0      0.667   1      2     0.333
 8       8 0      0       0      0       0      2       1      2     0
 9       9 0      0       0      0       0      2       1      3     0
10      10 0      0       0      0       0      2       1      4.5   0
11      11 0      0       0      0       0      0       1      1     0
12      12 0      0       0      0       0      2       1      2     0
13      13 0      0       0      0       0      0       1      2     0
# i 1,098 more variables: degrees <dbl>, `220` <dbl>, melt <dbl>, butter <dbl>,
#   saucepan <dbl>, medium <dbl>, heat <dbl>, add <dbl>, flour <dbl>,
```

```
#   stir <dbl>, form <dbl>, paste <dbl>, cook <dbl>, fragrant <dbl>, `1` <dbl>,
#   `2` <dbl>, minutes <dbl>, sugars <dbl>, water <dbl>, bring <dbl>,
#   boil <dbl>, reduce <dbl>, low <dbl>, simmer <dbl>, `3` <dbl>, `5` <dbl>,
#   remove <dbl>, press <dbl>, pastry <dbl>, bottom <dbl>, `9` <dbl>,
#   inch <dbl>, pie <dbl>, pan <dbl>, roll <dbl>, remaining <dbl>, ...
```

We do not learn too much from this since there are too many individual words to look at how each one affects the creating of a cluster. But, we can use Hierarchical clustering to see what outliers are in our data set.

```
recipes_cut |>
  slice(188)
```

```
# A tibble: 1 x 5
  recipe_name                                ingredients directions rating type
  <chr>                                      <chr>       <chr>       <dbl> <chr>
1 Chewy Coconut Bibingka (Filipino Rice Cak~ 1 (13.5 ou~ "Preheat ~    4.4 cake
```

This recipe is maybe not what you would find for a typical cake, so it makes sense that it would be farther from the other cakes in our data set.

PCA

We can use PCA to reduce the number of variables we are using which will help us better understand our data.

```
pca <- dfm2 |>
  dplyr::select(-recipe_id, -recipe_name) |>
  prcomp()

pca$rotation |>
  data.frame() |>
  dplyr::select(PC1) |>
  arrange(PC1) |>
  slice(1:10)
```

```
              PC1
1       -0.6315128
2       -0.2791254
cup     -0.2674398
sugar   -0.1871813
cake    -0.1537328
```

```
mixture -0.1461352
add     -0.1438912
beat    -0.1430664
4       -0.1379208
flour   -0.1329994
```

Let's take out numbers from our document feature matrix (helpful for our supervised algorithm later) and run this again:

```
numbers <- data.frame(num = 1:500)

recipes_words2 <- recipes_cut2 |>
  mutate(recipe_id = row_number()) |>
  unnest_tokens(input = "directions",
                output = "Word",
                token = "words") |>
  filter(!(Word %in% stop_words$word)) |>
  filter(!(Word %in% numbers$num))

dfm2 <- recipes_words2 |>
  pivot_wider(id_cols = c("recipe_id", "recipe_name"),
              names_from = "Word",
              values_from = "Word",
              values_fn = length,
              values_fill = 0)

pca <- dfm2 |>
  dplyr::select(-recipe_id, -recipe_name) |>
  prcomp()

pca$rotation |>
  data.frame() |>
  dplyr::select(PC1) |>
  arrange(PC1) |>
  slice(1:10)
```

```
                PC1
pie      -0.35540911
crust    -0.29903605
top      -0.20173762
sprinkle -0.08021943
strips   -0.07579761
```

```
apples   -0.07035290
filling  -0.07008114
pastry   -0.06644102
cut      -0.06641750
dough    -0.05833988
```

```
pca$rotation |>
  data.frame() |>
  dplyr::select(PC1) |>
  arrange(-PC1) |>
  slice(1:10)
```

```
             PC1
cake   0.3102338
beat   0.2683853
pan    0.2201878
batter 0.2062617
flour  0.2007141
bowl   0.1819725
add    0.1418341
cup    0.1368301
stir   0.1337301
baking 0.1278076
```

It looks like our PC1 is determining whether or not a recipe is a cake (large positive scores) or a pie (large negative scores).

```
pca$x[,1]
```

```
  [1] -4.90388117 -1.33956099 -1.91608810 -3.57548397 -2.16320137  3.30004192
  [7] -4.44516411 -2.80703979  1.39927656 -0.13664292 -3.87997008 -4.12175788
 [13] -1.59354150 -6.16748155 -2.02271278  2.20595470  2.42396870  1.36950106
 [19]  5.84288445 -2.70537958 -4.68841205 -2.41000636  1.33894574 -4.30714558
 [25]  1.01620630  1.44226013 -1.50439566 -2.02225040 -2.08299357  0.65926097
 [31]  3.05140637 -2.43847084 -0.46638530  2.09226741  8.89119422 -2.48599553
 [37] -1.59151407  2.57257021 -2.56266997  1.47098915 -1.23871921  2.91411681
 [43]  4.48432663  0.25327587  3.78480899  4.16260020  4.26980630 -1.07247192
 [49] -2.38376490  0.20932138  1.67532444  1.68132946  1.26203101 -1.07273305
 [55]  1.66688547 -6.52817065 -2.03184267 -1.99300161 -8.76008861  0.41715835
 [61] -0.83573257  2.68232941 -3.91520983 -0.50077000  1.24163022 -4.88646942
 [67] -2.60620763 -0.95803522 -2.51863483 -2.58533190 -0.86489534 -2.35778134
```

```
 [73] -2.13470515 -1.90436543 -2.88214971 -1.55125192 -2.80886860 -1.91926855
 [79] -1.37706595 -3.76655668 -1.20504075  0.62302532  3.40718889  1.34457790
 [85] -0.16888764  2.36381344  3.65644401  3.75020773 -0.62329367  3.26584761
 [91]  2.47947572 -3.59567808 -1.21529504  2.10489018  0.09439173  0.06875982
 [97]  1.37936950  0.72956714  0.86401897  5.52083943  1.06074038 -2.69842187
[103]  0.04600882  1.37313192 -2.41156414  4.70910589 -0.15415513  1.23321577
[109] -0.61533023  5.74405945  1.97120749 -0.96263720  1.14986111  1.41387300
[115] -1.26161750  2.99473101 -0.64710000  0.33833759  0.95850670 -3.95011123
[121]  3.18999173  1.73824341  3.90652276  2.32733323  0.40492327 -2.00648334
[127] -0.44071283  3.34685471 -2.83465763  0.57339282  1.95612482 -6.09371296
[133] -2.70250432 -1.14971703  4.28459759  2.57883646  1.26587767  3.19695076
[139]  3.68103390 -2.03755679  4.71710436 -3.86018322 -1.22166121  2.07024229
[145] -1.22875091 -1.84769726 -0.46263213 -1.18572061  5.05519009 -1.33218183
[151]  1.16288646  3.93569246  1.77191989 -1.90476847  0.18715340  2.14067706
[157] -2.08249058 -3.12179957 -2.10231227  3.66953564  1.30375676  6.94669566
[163]  2.42938374 -1.66634842 -1.03796416  0.72995474 -4.44781117 -0.29930889
[169] -1.36552790 -4.34308314 -4.24365280  4.40532797  0.09168599  0.92979532
[175]  1.59163747 -2.12792053 -1.77400299 -1.44139458  2.00274603  1.39875699
[181]  0.90819816  1.46813114 -0.84412777 -1.20701462  2.01364391  2.91343018
[187]  3.87024788 -0.40882647 -0.23037041 -0.17301632  0.92482600 -1.64467095
[193]  1.92507471  1.57311348 -0.08639569 -0.75600978
```

Looking at the scores for each recipe and checking them against our data set, we can see that this makes sense in our context.

```
pca$rotation |>
  data.frame() |>
  dplyr::select(PC2) |>
  arrange(-PC2) |>
  slice(1:10)
```

```
                PC2
coconut      0.03686102
strawberries 0.02807731
pineapple    0.02429251
pecans       0.02263611
whipped      0.02259917
drop         0.01838599
watermelon   0.01736526
sheets       0.01514278
oats         0.01431492
slice        0.01354580
```

PC2 seems to be determine whether or not a recipe includes a fruit.

We can continue to look at each PC and find what type of variance it accounts for and what words it uses (we will use more PCA and PCA features below).

Supervised Learning

Now we will create a supervised learning algorithm to predict the type of recipe based on the words used in the directions.

To start, lets use what we found doing PCA.

Using PC1 and PC2, we can create a random forest algorithm that uses these two variables to predict what type of recipe it is.

```
recipes_PC <- dfm2 |>
  mutate(type = factor(recipes_cut2$type)) |>
  mutate(PC1 = pca$x[,1],
         PC2 = pca$x[,2]) |>
  dplyr::select(recipe_name, type, PC1, PC2)

recipes_rf_PCA <- train(type ~ PC1 + PC2,
                    data = recipes_PC,
                    method = "ranger",
                    importance = "impurity")
```

```
note: only 1 unique complexity parameters in default grid. Truncating the grid to 1 .
```

```
varImp(recipes_rf_PCA)
```

```
ranger variable importance

    Overall
PC1     100
PC2       0
```

```
round(pca$sdev^2/sum(pca$sdev^2),2)
```

```
 [1] 0.09 0.08 0.05 0.04 0.03 0.03 0.03 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[16] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
[31] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
```

```
 [46] 0.01 0.01 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [61] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [76] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [91] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
[106] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
[121] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
[136] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
[151] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
[166] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
[181] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
[196] 0.00
```

This algorithm has a pretty high accuracy for only using two PCs. When looking at the how much variance is accounted for by each PC we see that it may be helpful to extend the amount of PCs we use as variables.

```
recipes_PC_large <- dfm2 |>
  mutate(type = factor(recipes_cut2$type)) |>
  mutate(PC1 = pca$x[,1],
         PC2 = pca$x[,2],
         PC3 = pca$x[,3],
         PC4 = pca$x[,4],
         PC5 = pca$x[,5],
         PC6 = pca$x[,6]) |>
  dplyr::select(recipe_name, type, PC1, PC2, PC3, PC4, PC5, PC6)

recipes_rf_PCA_large <- train(type ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6,
                     data = recipes_PC_large,
                     method = "ranger",
                     importance = "impurity")

varImp(recipes_rf_PCA_large)
```

```
ranger variable importance

    Overall
PC1 100.000
PC3  58.345
PC2  13.394
PC6   7.496
PC5   5.807
PC4   0.000
```

This has a much higher accuracy and did not take much longer to run. We can also see that PC1 and PC3 are the most important variables for this algorithm (this will be important later for our visual)

Now let's see how some other algorithms work and how we can tune them to best fit our context:

Random Forest (with all words as variables)

```
recipe_rf_data <- dfm2 |>
  mutate(type = factor(recipes_cut2$type)) |>
  dplyr::select(-recipe_id) |>
  na.omit()

recipe_rf <- train(type ~.,
                   data = recipe_rf_data,
                   method = "ranger",
                   importance = "impurity")

recipe_rf
```

```
Random Forest

 196 samples
1050 predictors
   3 classes: 'cake', 'cookie', 'pie'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 196, 196, 196, 196, 196, 196, ...
Resampling results across tuning parameters:

  mtry  splitrule   Accuracy   Kappa
     2  gini        0.5019253  0.02033543
     2  extratrees  0.4989402  0.01103527
    49  gini        0.9343002  0.89081352
    49  extratrees  0.9014730  0.83710597
  1244  gini        0.9066770  0.84458899
  1244  extratrees  0.9192242  0.86585065

Tuning parameter 'min.node.size' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were mtry = 49, splitrule = gini
```

```
 and min.node.size = 1.
```

The accuracy of this random forest is higher than it is when we use PCs, which makes sense because our random forest is using all words while PCA is using a reduced number of variables. However, this algorithm took much longer to run and the PCA accuracy was still very high so which algorithm we would want to use would depend on how important accuracy vs. run time is (and how big our data set is in a different context). We could also add a few more PC variables to our first algorithm to try to get the accuracy a bit higher.

```
varImp(recipe_rf)
```

```
ranger variable importance

  only 20 most important variables shown (out of 1244)

          Overall
pie        100.00
cake        81.34
drop        46.83
crust       42.78
batter      38.50
sheets      38.00
clean       24.68
pan         23.74
baking      20.34
cookie      20.18
soda        18.75
inserted    18.53
toothpick   16.21
dough       14.96
inches      12.65
eggs        12.45
grease      11.77
shell       11.61
beat        11.53
`9x13`      10.58
```

Looking at what variables our random forest found as most important when making splits, we can see that "pie," "cake," and "cookie" were all in the top ten. This makes sense because the directions would have to refer to the type of dessert being made. Although this helps our algorithm make good predictions, let's see if we can still make accurate predictions without

these key words so that we are looking at actual method/buzz-word similarities between baking types.

```
recipe_rf_data2 <- dfm2 |>
  mutate(type = factor(recipes_cut2$type)) |>
  dplyr::select(-recipe_id, -recipe_name, -cookie, -cake, -pie) |>
  na.omit()

recipe_rf2 <- train(type ~.,
                    data = recipe_rf_data2,
                    method = "ranger",
                    importance = "impurity")

recipe_rf2
```

```
Random Forest

 196 samples
1046 predictors
   3 classes: 'cake', 'cookie', 'pie'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 196, 196, 196, 196, 196, 196, ...
Resampling results across tuning parameters:

  mtry  splitrule   Accuracy   Kappa
     2  gini        0.5178476  0.03137625
     2  extratrees  0.5111084  0.01725132
    45  gini        0.8907283  0.81951359
    45  extratrees  0.8688101  0.78470464
  1046  gini        0.8676641  0.77986539
  1046  extratrees  0.8672909  0.78086174

Tuning parameter 'min.node.size' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were mtry = 45, splitrule = gini
 and min.node.size = 1.
```

```
varImp(recipe_rf2)
```

```
ranger variable importance
```

```
  only 20 most important variables shown (out of 1046)

          Overall
crust      100.00
drop        99.02
batter      88.90
sheets      81.35
pan         65.12
clean       63.28
baking      47.39
inserted    45.95
toothpick   42.91
soda        39.42
dough       39.07
shell       33.95
`9x13`      33.72
beat        29.51
powder      28.80
eggs        28.72
grease      28.42
inches      23.44
pour        20.39
sheet       20.18
```

Our algorithm did not lose accuracy so it looks like there are still many words unique to the specific baking recipes that our algorithm can use, such as `crust`. The most important words are the words that our algorithm is saying are good predictors for the type of recipe.

It would be interesting to pick different focuses and narrow in on how well our algorithm predicts the type given the focus. For example, if we only selected words that are methods (such as `beat`, `mix`, `fold`, etc.), or words that are ingredients.

Let's look at a few more supervised learning algorithms (using our large PCA data set) to see what we can learn from them about the data and check if they have a higher accuracy than our random forest.

kNN (with all words as variables)

```
library(janitor)
```

```
Attaching package: 'janitor'
```

```
The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

```
recipe_rd_data_clean <- recipe_rf_data2 |>
  rename_all(paste0, "_token")

recipe_knn <- train(type_token ~.,
                    data = recipe_rd_data_clean |> clean_names(),
                    method = "knn")
```

LDA (using PCA)

```
recipes_lda_PCA_large <- train(type ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6,
                     data = recipes_PC_large,
                     method = "lda")
```

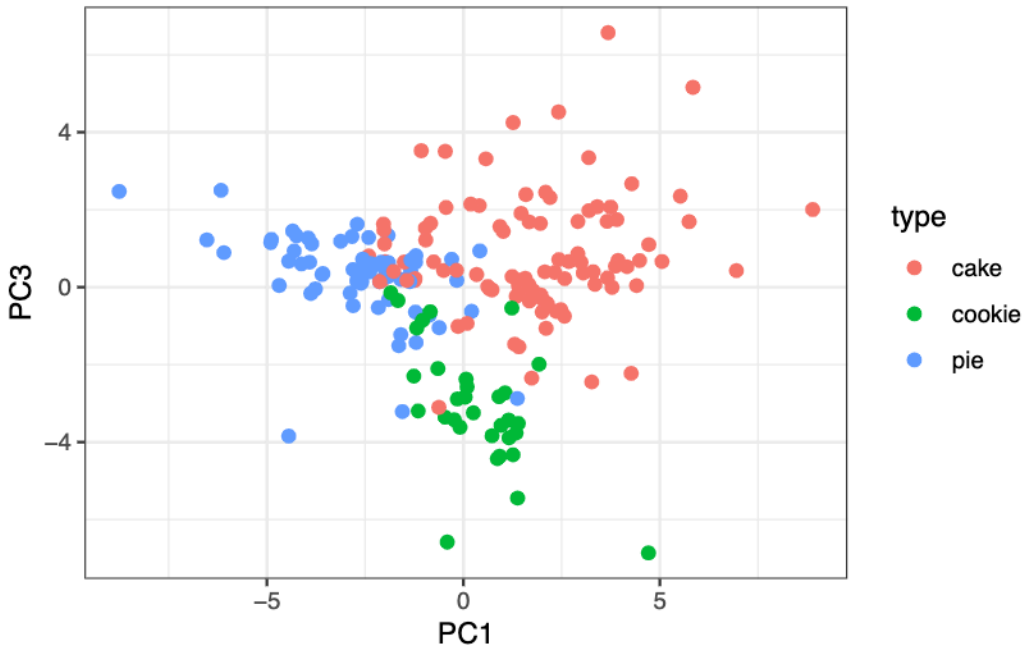
This LDA algorithm is only slightly lower than the accuracy for our random forest large PCA model.

Visualizing the Data

Since it is impossible to create a visual on how all words are related to each other, we can use PCA to make a good 2d visual. We know from above that PC1 and PC3 are the most important predictors for our random forest algorithm, so we will use those as our x and y variables.

```
recipes_PC_visual <- dfm2 |>
  mutate(type = factor(recipes_cut2$type)) |>
  mutate(PC1 = pca$x[,1],
         PC3 = pca$x[,3]) |>
  dplyr::select(recipe_name, type, PC1, PC3)

recipes_PC_visual |>
  ggplot() +
  geom_point(aes(x = PC1, y = PC3,
                 color = type), size = 2) +
  theme_bw()
```


PC3
4
0
−4
−5
0
5
PC1
type
cake
cookie
pie