PPOL 6081 - Problem Set 2

Amber Ni

2025-03-19

In this problem set, I am working with a dataset of skincare product reviews from the Sephora online store, which I obtained from Kaggle. https://www.kaggle.com/datasets/nadyinky/sephora-products-and-skincare-reviews/data (https://www.kaggle.com/datasets/nadyinky/sephora-products-and-skincare-reviews/data).

My dataset

```
reviews <- read.csv("reviews_0-250.csv")
```

Select categories & Split data into train/test sets

```
# Get a subset of 10000 observations
set.seed(123)
reviews_subset <- reviews %>% slice_sample(n = 10000)

# Classify reviews into three categories
reviews_subset <- reviews_subset %>%
    mutate(review_type = case_when(
    rating %in% c(4, 5) ~ "Positive",
    rating == 3 ~ "Neutral",
    rating %in% c(1, 2) ~ "Negative"
  ))

# Check the class distribution
prop.table(table(reviews_subset$review_type))
```

```
##
## Negative  Neutral Positive
##   0.0999   0.0779   0.8222
```

```
# Only keep texts and categories for simplicity
reviews_samp <- reviews_subset %>%
  select(review_text, review_title, review_type)

# Split the data into training and test sets
set.seed(1310)

prop_train <- 0.8
ids <- 1:nrow(reviews_samp)  # Create an index for all rows
ids_train <- sample(ids, ceiling(prop_train * length(ids)), replace = FALSE) # Randomly
sample indices for training data
ids_test <- ids[-ids_train] # Remaining indices for test data

train_set <- reviews_samp[ids_train, ]
test_set <- reviews_samp[ids_test, ]
```

Pre-processing

```
# Check a few reviews to see the nature of the texts
head(train_set$review_text, n =3)
```

```
## [1] "I saw immediate immediate brightening and smoothness of my skin after just one u
se! I’ve been using this consistently for a few weeks, Doesn’t break me out.  I have sen
sitive acne prone skin. After a long week, this helps my skin feel refreshed and clean.
I received this as a 100point sephora reward, then received it again in the Influenster
box. So glad I did! Will buy the full size once my trial sizes run out."
## [2] "This is step 4 in my skincare routine. I apply it right after The Essence. I thi
nk it’s very light and smells great and sets up the skin for the day."
## [3] "I’m about to turn 48, and when I turned 46 my skin started getting dry patches o
n my cheeks that I never had before. I found that most day-use moisturizers weren’t cutt
ing it, and it was time to whip out the big guns. I actually use this stuff on cold, dr
y, winter days on my cheeks, and it’s great. It works under my makeup and absorbs really
well. A little goes a long way, so don’t go nuts. It’s lovely as a night-cream, too, and
in my slightly younger days this would be all I needed at night... but nowadays I admit
I need stronger stuff for night-only winter use. (Yay for Bobbi Brown Extra Repair Moist
ure Cream. That’s some heavy-duty night-only stuff!)"
```

```
# Remove common English contractions and avoid empty spaces being recognized by a token
reviews_samp$review_text <- gsub("'s|'m|'re|'d|'ve|'ll|n't|’s|’m|’re|’d|’ve|’ll|n’t",
"", reviews_samp$review_text)
train_set$review_text <- gsub("'s|'m|'re|'d|'ve|'ll|n't|’s|’m|’re|’d|’ve|’ll|n’t", "", t
rain_set$review_text)
test_set$review_text <- gsub("'s|'m|'re|'d|'ve|'ll|n't|’s|’m|’re|’d|’ve|’ll|n’t", "", te
st_set$review_text)

# Clean texts using textclean
reviews_samp$review_text <- reviews_samp$review_text %>%
  replace_contraction() %>%
  replace_word_elongation() %>%
  replace_symbol() %>%
  replace_number() %>%
  replace_non_ascii()

train_set$review_text <- train_set$review_text %>%
  replace_contraction() %>%
  replace_word_elongation() %>%
  replace_symbol() %>%
  replace_number() %>%
  replace_non_ascii()

test_set$review_text <- test_set$review_text %>%
  replace_contraction() %>%
  replace_word_elongation() %>%
  replace_symbol() %>%
  replace_number() %>%
  replace_non_ascii()

# Convert to a document-feature matrix (DFM)
reviews_dfm <- tokens(reviews_samp$review_text, remove_punct = TRUE, remove_numbers = TR
UE) %>%
  tokens_tolower() %>%
  dfm() %>%
  dfm_wordstem() %>%
  dfm_remove(stopwords("en"))

# Repeat preprocessing steps for train set
train_dfm <- tokens(train_set$review_text, remove_punct = TRUE, remove_numbers = TRUE) %
>%
  tokens_tolower() %>%
  dfm() %>%
  dfm_wordstem() %>%
  dfm_remove(stopwords("en"))

# Repeat preprocessing steps for test set
test_dfm <- tokens(test_set$review_text, remove_punct = TRUE, remove_numbers = TRUE) %>%
  tokens_tolower() %>%
  dfm() %>%
  dfm_wordstem() %>%
  dfm_remove(stopwords("en"))
```

```
# Check top features of each DFM
topfeatures(train_dfm)
```

```
##    skin     use product    love    feel    like moistur    face     dri  realli
##    9206    6309    5533    3423    3236    2644    2557    2523    2164    2042
```

```
topfeatures(test_dfm)
```

```
##    skin     use product    love    feel    face    like moistur     dri  realli
##    2255    1496    1417     857     779     650     617     600     519     481
```

```
# Inspect the DFM
as.matrix(train_dfm)[1:5, 1:5]
```

```
##        features
## docs    saw immedi brighten smooth skin
##   text1   1      2        1      1    3
##   text2   0      0        0      0    1
##   text3   0      0        0      0    1
##   text4   0      0        0      0    0
##   text5   0      0        0      0    0
```

```
as.matrix(test_dfm)[1:5, 1:5]
```

```
##        features
## docs    lip balm great good men
##   text1   1    1     2    1   1
##   text2   0    0     0    0   0
##   text3   2    1     1    0   0
##   text4   0    0     0    0   0
##   text5   2    1     0    0   0
```

```
# Ensure that test set DFM has the same features as training set DFM
test_dfm <- dfm_match(test_dfm, features = featnames(train_dfm))
```

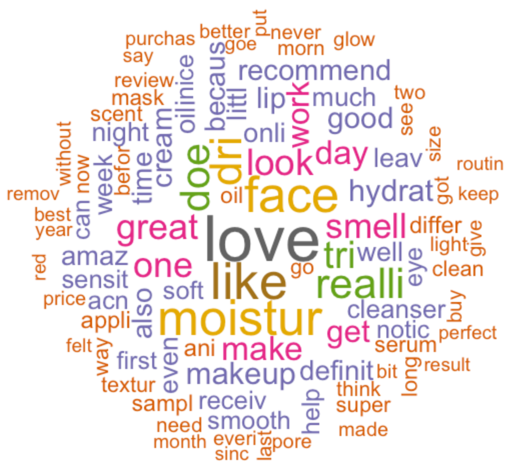
Review Word Clouds

```
# Load package necessary to create word clouds
pacman::p_load(quanteda.textplots)

# Create a word cloud for reviews

reviews_wordcloud <- dfm_remove(reviews_dfm, c("skin", "product", "feel", "use", "just",
"veri"))

reviewsWordcloud <- textplot_wordcloud(reviews_wordcloud, random_order = FALSE,
                   rotation = .25, max_words = 100,
                   min_size = 0.5, max_size = 2.8,
                   colors = RColorBrewer::brewer.pal(8, "Dark2"))
```



Different machine learning models

NAIVE BAYES WITH SMOOTHING

```
# Train a Naive Bayes model with Laplace smoothing (smooth = 1)
nb_model_sm <- textmodel_nb(train_dfm, train_set$review_type, smooth = 1, prior = "unifo
rm")
summary(nb_model_sm)
```

```
##
## Call:
## textmodel_nb.dfm(x = train_dfm, y = train_set$review_type, smooth = 1,
##     prior = "uniform")
##
## Class Priors:
## (showing first 3 elements)
## Negative  Neutral Positive
##   0.3333   0.3333   0.3333
##
## Estimated Feature Scores:
##                saw    immedi  brighten    smooth    skin     just      one
## Negative 0.0007080 0.0005731 0.0002697 0.0004046 0.02549 0.007248 0.004821
## Neutral  0.0004080 0.0004080 0.0004080 0.0018176 0.02437 0.007233 0.004154
## Positive 0.0005608 0.0007133 0.0005854 0.0042700 0.03835 0.005219 0.006710
##              use  consist     week      doe    break   sensit      acn
## Negative 0.01868 0.000708 0.003405 0.006844 0.003540 0.003304 0.003371
## Neutral  0.01640 0.001002 0.002671 0.007567 0.001595 0.002559 0.001892
## Positive 0.02615 0.001122 0.003945 0.007620 0.001525 0.003630 0.003768
##              prone      long      help     feel   refresh    clean   receiv
## Negative 0.0009103 0.0008765 0.0009777 0.006473 0.0002360 0.001079 0.001113
## Neutral  0.0004822 0.0019289 0.0017063 0.010164 0.0004822 0.001632 0.003227
## Positive 0.0011118 0.0030352 0.0038961 0.013641 0.0014364 0.002991 0.003832
##           100point   sephora    reward influenst       box      glad      buy
## Negative 3.371e-05 0.0011125 6.743e-05  0.000472 0.0003371 0.0002023 0.001854
## Neutral  3.709e-05 0.0005564 7.419e-05  0.001187 0.0004822 0.0005193 0.001966
## Positive 1.476e-05 0.0009642 5.411e-05  0.001579 0.0003788 0.0005706 0.002469
##               full     size
## Negative 0.0006743 0.001045
## Neutral  0.0009273 0.001818
## Positive 0.0015004 0.002558
```

```
# Predict on the test set with smoothed model
predicted_class_sm <- predict(nb_model_sm, newdata = test_dfm)

# Confusion matrix for the smoothed model
tabclass_sm <- table(test_set$review_type, predicted_class_sm)
confusionMatrix(tabclass_sm, mode = "everything") # from caret package in R
```

```
## Confusion Matrix and Statistics
##
##           predicted_class_sm
##            Negative Neutral Positive
##   Negative       97      21       66
##   Neutral        23      33       97
##   Positive       67      22     1574
##
## Overall Statistics
##
##                Accuracy : 0.852
##                  95% CI : (0.8357, 0.8673)
##     No Information Rate : 0.8685
##     P-Value [Acc > NIR] : 0.9856
##
##                   Kappa : 0.4443
##
##  Mcnemar's Test P-Value : 2.903e-10
##
## Statistics by Class:
##
##                      Class: Negative Class: Neutral Class: Positive
## Sensitivity                   0.5187         0.4342          0.9062
## Specificity                   0.9520         0.9376          0.6616
## Pos Pred Value                0.5272         0.2157          0.9465
## Neg Pred Value                0.9504         0.9767          0.5163
## Precision                     0.5272         0.2157          0.9465
## Recall                        0.5187         0.4342          0.9062
## F1                            0.5229         0.2882          0.9259
## Prevalence                    0.0935         0.0380          0.8685
## Detection Rate                0.0485         0.0165          0.7870
## Detection Prevalence          0.0920         0.0765          0.8315
## Balanced Accuracy             0.7354         0.6859          0.7839
```

The Naive Bayes model achieved **an overall accuracy of 85.2%**, indicating strong general performance in predicting class labels. The **Kappa score of 0.4443** suggests moderate agreement between predictions and actual labels beyond random chance.

The model performs exceptionally well for the **Positive Class**, with **a high sensitivity of 0.9062, precision of 0.9465, and an F1 score of 0.9259**. This demonstrates its effectiveness in correctly identifying positive samples. However, performance for the **Neutral Class** is poor, with **sensitivity at 0.4342, precision at 0.2157, and an F1 score of 0.2882**, which indicates difficulty distinguishing neutral instances from others. The **Negative Class** shows moderate performance with balanced performance statistics.

```
# Investigate the most discriminative features
posterior <- tibble(
  feature = colnames(nb_model_sm$param),
  Negative = t(nb_model_sm$param)[, 1],
  Neutral = t(nb_model_sm$param)[, 2],
  Positive = t(nb_model_sm$param)[, 3]
)
head(t(nb_model_sm$param))
```

```
##              Negative     Neutral     Positive
## saw      0.0007079765 0.000408027 0.0005608001
## immedi   0.0005731239 0.000408027 0.0007132983
## brighten 0.0002697053 0.000408027 0.0005853966
## smooth   0.0004045580 0.001817575 0.0042699514
## skin     0.0254871553 0.024370340 0.0383508542
## just     0.0072483312 0.007233206 0.0052193761
```

Based on the most discriminative features we examined, words like **"immediately," "brighten," and "smooth"** are strongly associated with the **Positive Class**. This aligns with our expectation that these positive comments related to skincare product effectiveness are clear indicators of positive sentiment. Conversely, words like **"just"** are more closely associated with the **Neutral and Negative Classes**, probably reflecting complaint-like expressions and moderate emotions, which also matches our assumptions. There are no obvious words that are incorrectly contributing to the wrong class.

```
# Top 10 most predictive features
# posterior %>% arrange(-Negative) %>% head(10)
# posterior %>% arrange(-Neutral) %>% head(10)
# posterior %>% arrange(-Positive) %>% head(10)

posterior <- posterior %>%
  mutate(
    positive_vs_negative = log(Positive / Negative),
    positive_vs_neutral  = log(Positive / Neutral),
    negative_vs_neutral  = log(Negative / Neutral)
  )
posterior %>%
  arrange(-positive_vs_negative) %>%
  head(10)
```

```
## # A tibble: 10 × 7
##    feature   Negative  Neutral Positive positive_vs_negative positive_vs_neutral
##    <chr>        <dbl>    <dbl>    <dbl>                <dbl>               <dbl>
##  1 awesom   0.0000337  1.85e-4 0.000418                 2.52               0.813
##  2 silki    0.0000337  1.11e-4 0.000418                 2.52               1.32
##  3 stapl    0.0000337  7.42e-5 0.000418                 2.52               1.73
##  4 smooth   0.000405   1.82e-3 0.00427                  2.36               0.854
##  5 soft     0.000438   1.48e-3 0.00416                  2.25               1.03
##  6 sooth    0.0000674  7.42e-5 0.000605                 2.19               2.10
##  7 perfect  0.000270   5.19e-4 0.00241                  2.19               1.53
##  8 thicker  0.0000337  3.34e-4 0.000300                 2.19              -0.107
##  9 penni    0.0000337  1.11e-4 0.000271                 2.08               0.888
## 10 smoother 0.000101   3.34e-4 0.000787                 2.05               0.858
## # ℹ 1 more variable: negative_vs_neutral <dbl>
```

I then examined the 10 most predictive features in each categories. However same words appear across multiple classes, because they're common words related to the general topic, which makes it harder to tell which category a word really signals. I changed to use the ratios, instead of just looking at how frequent a word is in one class, to examine how much more likely it is to appear in one class compared to another.

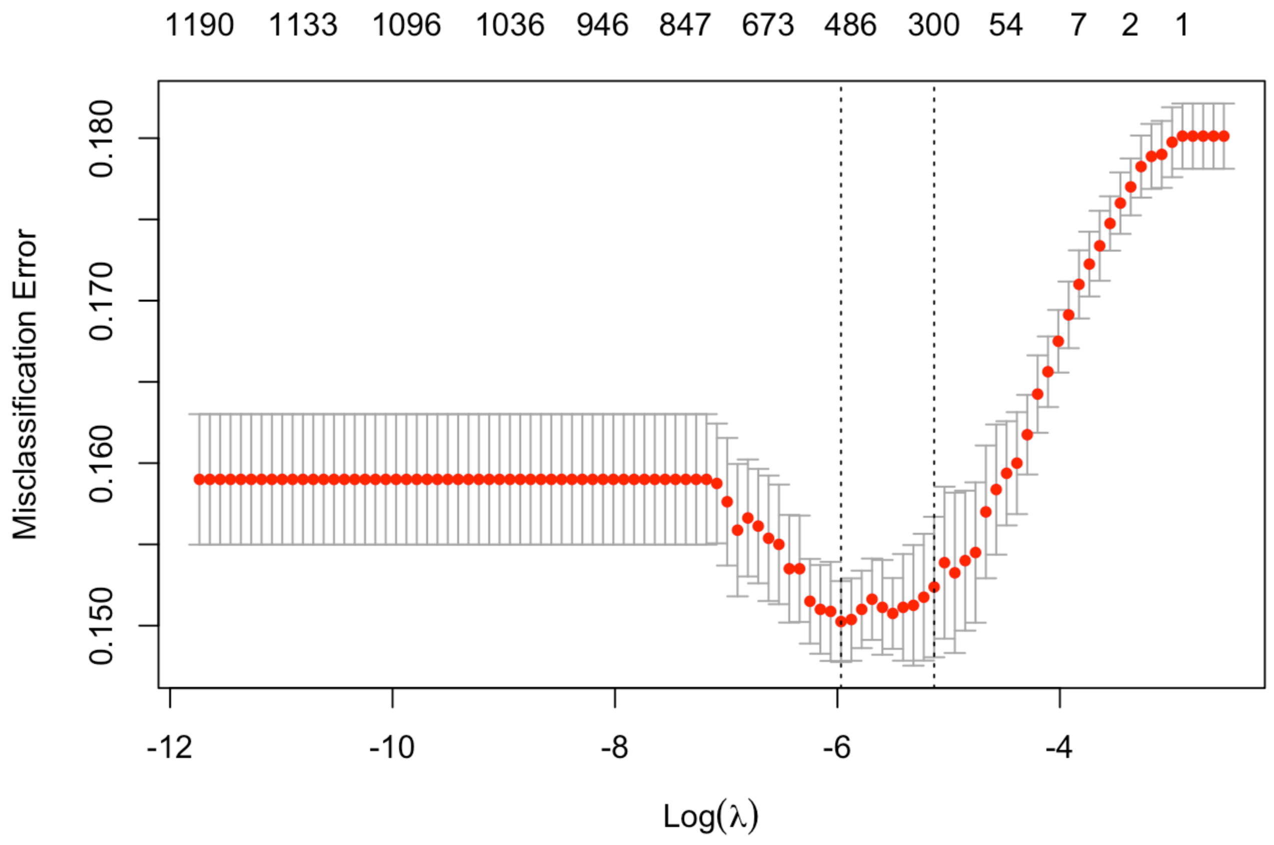
The results show that most of the words commonly associated with skincare effectiveness, such as **"smooth," "soft," "perfect," "sooth,"** and **"smoother,"** tend to have higher probabilities of appearing in the **Positive** class. This is evident from the high `positive_vs_negative` and `positive_vs_neutral` scores, suggesting that users who are satisfied with skincare products frequently use these descriptors to describe their experiences. Consumers, when buying with skincare products, seemingly care more about smoothing effect and product texture.

REGULARIZATION (LASSO)

```
# Convert class labels to factors; set reference class
train_set$positive <- fct_relevel(as_factor(train_set$review_type), "Positive")
test_set$positive <- fct_relevel(as_factor(test_set$review_type), "Positive")

# Build the LASSO model
lasso <- cv.glmnet(
  x = train_dfm, y = train_set$positive,
  family = "multinomial", alpha = 1, nfolds = 5, #  Logistic regression (binomial); Lass
o (alpha = 1); 5-fold Cross-Validation
  intercept = TRUE, type.measure = "class"
)

# Plot the lambda values
plot(lasso) # check out optimal lambda
```



```
# See Lasso model result
lasso
```

```
##
## Call:  cv.glmnet(x = train_dfm, y = train_set$positive, type.measure = "class",
nfolds = 5, family = "multinomial", alpha = 1, intercept = TRUE)
##
## Measure: Misclassification Error
##
##       Lambda Index Measure       SE Nonzero
## min 0.002559    38  0.1502 0.002489     500
## 1se 0.005912    29  0.1524 0.004323     300
```

```
# Extract the lambda values
best_lambda_min <- lasso$lambda.min  # Optimal lambda (minimum error)

# Print the values
print(best_lambda_min)
```

```
## [1] 0.002559357
```

```
# Predict on the test set
predicted_lasso <- predict(lasso, newx = test_dfm, s = best_lambda_min, type = "class")

# Confusion matrix for Lasso to evaluate preformance
tabclass_lasso <- table(fct_rev(test_set$positive), predicted_lasso)
print(tabclass_lasso)
```

```
##           predicted_lasso
##            Negative Neutral Positive
##   Negative       83       9       92
##   Neutral        15      37      101
##   Positive       20      26     1617
```

```
lasso_cmat <- confusionMatrix(tabclass_lasso, mode = "everything")
print(lasso_cmat)
```

```
## Confusion Matrix and Statistics
##
##           predicted_lasso
##            Negative Neutral Positive
##   Negative       83       9       92
##   Neutral        15      37      101
##   Positive       20      26     1617
##
## Overall Statistics
##
##                Accuracy : 0.8685
##                  95% CI : (0.8529, 0.883)
##     No Information Rate : 0.905
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.4505
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: Negative Class: Neutral Class: Positive
## Sensitivity                   0.7034         0.5139          0.8934
## Specificity                   0.9463         0.9398          0.7579
## Pos Pred Value                0.4511         0.2418          0.9723
## Neg Pred Value                0.9807         0.9811          0.4273
## Precision                     0.4511         0.2418          0.9723
## Recall                        0.7034         0.5139          0.8934
## F1                            0.5497         0.3289          0.9312
## Prevalence                    0.0590         0.0360          0.9050
## Detection Rate                0.0415         0.0185          0.8085
## Detection Prevalence          0.0920         0.0765          0.8315
## Balanced Accuracy             0.8249         0.7269          0.8256
```

The LASSO regularized logistic regression model achieved an overall accuracy of **86.85%**, which shows a slight improvement compared to the Naive Bayes model's performance (an overall accuracy of **85.2**). The model's Kappa score of **0.4505** indicates moderate agreement between predictions and actual labels, slightly better than the Bayes model.

The model demonstrates strong predictive power for the **Positive Class**, with high sensitivity (**0.8934**), precision (**0.9723**), and an impressive F1 score of **0.9312**. However, similar to the Naive Bayes model, performance for the **Neutral Class** remains poor, with low precision (**0.2418**) and an F1 score of **0.3289**, indicating difficulty in accurately identifying neutral samples. The **Negative Class** shows moderate performance, with a precision of **0.4511** and an F1 score of **0.5497**. Generally speaking, most performance statistics of the LASSO model are better than the Naive Bayes model.

REGULARIZATION (Ridge)

```
# Train a Ridge regression model (alpha = 0 for Ridge)
ridge <- cv.glmnet(
  x = train_dfm, y = train_set$positive,
  family = "multinomial", alpha = 0, nfolds = 5,
  intercept = TRUE, type.measure = "class"
)

# Predict on the test set
predicted_ridge <- predict(ridge, newx = test_dfm, type = "class")

# Confusion matrix for Ridge
tabclass_ridge <- table(fct_rev(test_set$positive), predicted_ridge)
ridge_cmat <- confusionMatrix(tabclass_ridge, mode = "everything")

# Print Ridge results
print(ridge_cmat)
```

```
## Confusion Matrix and Statistics
##
##           predicted_ridge
##            Negative Neutral Positive
##   Negative        7       0      177
##   Neutral         0       3      150
##   Positive        1       1     1661
##
## Overall Statistics
##
##                Accuracy : 0.8355
##                  95% CI : (0.8185, 0.8515)
##     No Information Rate : 0.994
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.049
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Negative Class: Neutral Class: Positive
## Sensitivity                  0.87500        0.75000         0.83551
## Specificity                  0.91114        0.92485         0.83333
## Pos Pred Value               0.03804        0.01961         0.99880
## Neg Pred Value               0.99945        0.99946         0.02967
## Precision                    0.03804        0.01961         0.99880
## Recall                       0.87500        0.75000         0.83551
## F1                           0.07292        0.03822         0.90989
## Prevalence                   0.00400        0.00200         0.99400
## Detection Rate               0.00350        0.00150         0.83050
## Detection Prevalence         0.09200        0.07650         0.83150
## Balanced Accuracy            0.89307        0.83742         0.83442
```

The **Ridge regularized logistic regression model** achieved an overall accuracy of **83.55%**, which is slightly lower compared to both the **LASSO model (86.85%)** and the **Naive Bayes model (85.2%)**. The model's Kappa score of **0.049** indicates very low agreement between predictions and actual labels.

The model shows strong predictive power for the **Positive Class**, with high sensitivity (**0.8355**), precision (**0.9988**), and a F1 score of **0.9099**. However, the performance of the model for the **Negative Class** is exceptionally poor, with a precision of only **0.0380** and an F1 score of **0.0729**. The **Neutral Class** also performs poorly, with a precision of **0.0196** and an F1 score of **0.0382**. It indicates that it rarely captures neutral and negative reviews accurately.

Overall, the **Ridge model** appears to be overly focused on correctly predicting the **Positive Class**, probably due to the fact that it is the most dominant class. Its poor performance for the **Negative and Neutral Classes** makes it seemingly unsuitable for scenarios where a balanced prediction across categories is desired. Both the **LASSO and Naive Bayes models** show better handling of class imbalances.

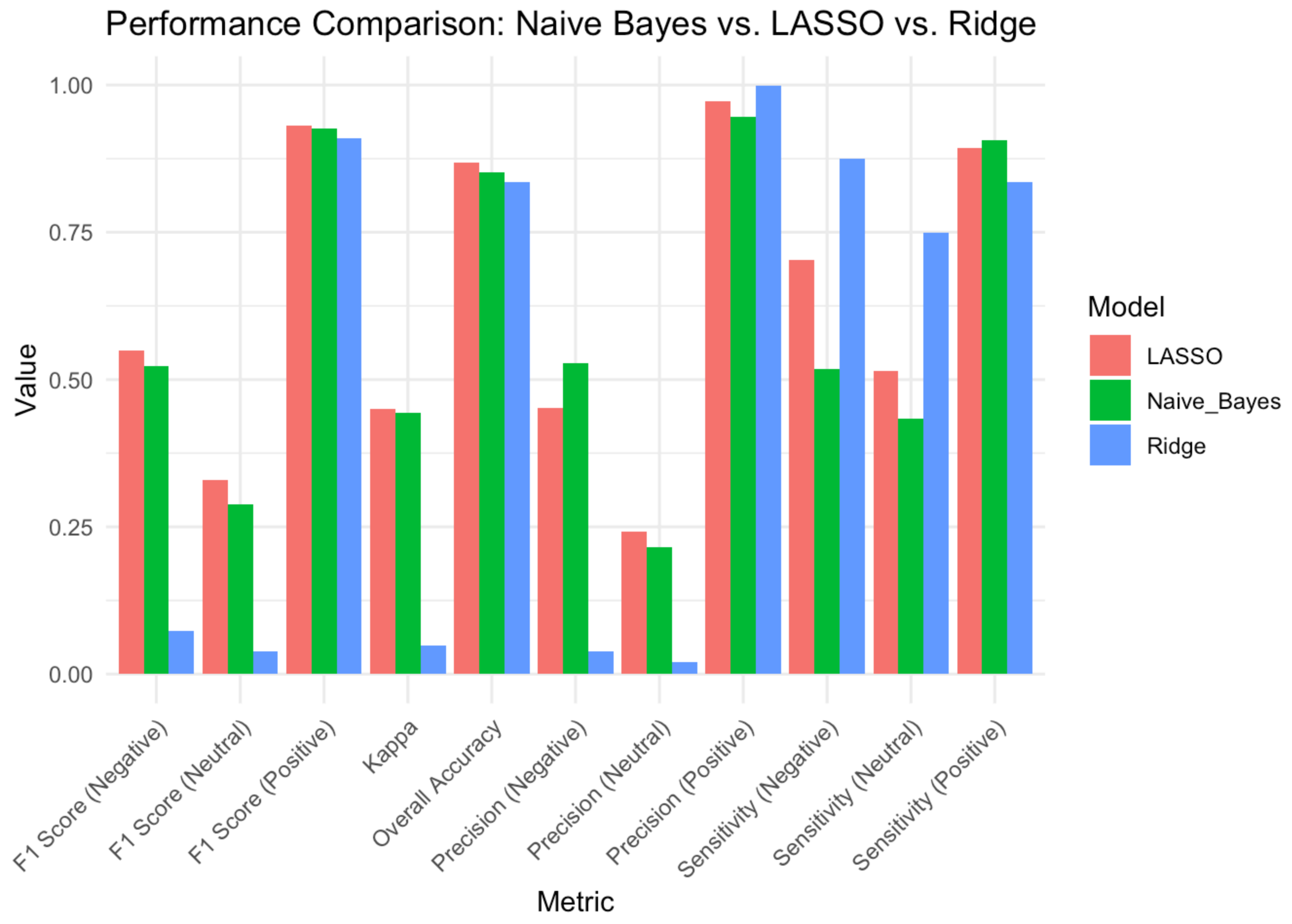
Comparison: Naive Bayes vs. LASSO vs. Ridge

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

# Create a data frame for comparison
model_comparison <- data.frame(
  Metric = c("Overall Accuracy", "Kappa",
             "Precision (Positive)", "Sensitivity (Positive)", "F1 Score (Positive)",
             "Precision (Negative)", "Sensitivity (Negative)", "F1 Score (Negative)",
             "Precision (Neutral)", "Sensitivity (Neutral)", "F1 Score (Neutral)"),
  Naive_Bayes = c(0.852, 0.4443,
                  0.9465, 0.9062, 0.9259,
                  0.5272, 0.5187, 0.5229,
                  0.2157, 0.4342, 0.2882),
  LASSO = c(0.8685, 0.4505,
            0.9723, 0.8934, 0.9312,
            0.4511, 0.7034, 0.5497,
            0.2418, 0.5139, 0.3289),
  Ridge = c(0.8355, 0.049,
            0.9988, 0.8355, 0.9098,
            0.03804, 0.8750, 0.07292,
            0.01961, 0.7500, 0.03822)
)

# Convert to long format for plotting
comparison_long <- model_comparison %>%
  pivot_longer(cols = c("Naive_Bayes", "LASSO", "Ridge"),
               names_to = "Model", values_to = "Value")

# Plotting
ggplot(comparison_long, aes(x = Metric, y = Value, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Performance Comparison: Naive Bayes vs. LASSO vs. Ridge",
       fill = "Model") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



The comparison graph clearly shows that, generally speaking, the **LASSO model** outperforms both the **Naive Bayes model** and the **Ridge model** across most metrics, particularly in terms of overall accuracy and Kappa score. While all three models are effective at predicting the **Positive Class**, which is likely due to the strong presence of positive sentiment and easily identifiable keywords, the **Ridge model** achieves the highest precision for positive reviews- but it may be overfitting to the positive class.

Performance for the **Negative Class** is moderate across all models, with the **Ridge model** showing a particularly low F1 score, indicating poor recall despite a high precision. Both the **Naive Bayes model** and **LASSO model** handle negative predictions better. All three models struggle significantly with predicting the **Neutral Class.** The **LASSO model** provides the most balanced performance, but still, its F1 score for the **Neutral Class.** This is likely because negative and especially neutral language tends to be more subtle, ambiguous, and harder to detect and differentiate from other classes. The lack of clear indicators for neutral sentiments further complicates accurate classification.

K-means clustering

```
dfm_reviews_prop <- dfm_weight(reviews_dfm, scheme = "prop")

# ------ k-means
n.clust <- 7 # choose the number of clusters (I identified 7 as the optimal k after usin
g the elbow method below)

# estimate clusters - group documents into 7 clusters based on word patterns in the DFM
set.seed(2025)
k_cluster <- kmeans(dfm_reviews_prop, centers = n.clust)

# view the structure of the clustering output
str(k_cluster)
```

```
## List of 9
##  $ cluster     : Named int [1:10000] 4 3 7 6 3 3 7 7 3 4 ...
##   ..- attr(*, "names")= chr [1:10000] "text1" "text2" "text3" "text4" ...
##  $ centers     : num [1:7, 1:7952] 0.000414 0.002836 0.001029 0.107256 0.001191 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:7] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:7952] "lip" "balm" "great" "good" ...
##  $ totss       : num 526
##  $ withinss    : num [1:7] 26 24.4 187.3 26.4 71.4 ...
##  $ tot.withinss: num 489
##  $ betweenss   : num 36.8
##  $ size        : int [1:7] 481 532 4308 485 1438 635 2121
##  $ iter        : int 6
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

```
## show the assigned cluster for each document
# k_cluster$cluster

# number of documents assigned to each cluster
table(k_cluster$cluster)
```

```
##
##    1    2    3    4    5    6    7
##  481  532 4308  485 1438  635 2121
```

The K-means clustering analysis grouped the dataset into **7 distinct clusters**. The **total within-cluster sum of squares (tot.withinss)** is **489**, which measures the compactness of the clusters — lower values indicate tighter, more cohesive clusters. Meanwhile, the **between-cluster sum of squares (betweenss)** is **36.8**, suggesting a relatively small separation between clusters. It suggests that some clusters may overlap, particularly if the features are not well-separated. The sizes of the clusters range from **481 to 4308 samples**, indicating an imbalanced distribution.

That's likely because most of our reviews belong to the Positive Class. Additionally, words and sentiments from the Neutral and Negative Classes tend to be more subtle and harder to detect, making them difficult to group into distinct clusters. As a result, these reviews are more likely to be misclassified or wrongly grouped, leading to overlap between clusters. The ambiguity in language and lack of strong indicator words for the Negative and Neutral Classes can contribute to the confusion and may negatively impact classification performance.

```
# ----- labeling the clusters

# identify the 10 most important words for each cluster based on cluster center values
top_words_per_cluster <- matrix(NA, nrow = n.clust, ncol = 10) # to store 10 most import
ant words for each cluster

# extract the most important ones
for(z in 1:n.clust) {
  top_words_per_cluster[z,] <- colnames(reviews_dfm)[order(k_cluster$center[z,], decreas
ing = TRUE)[1:10]]
}

cluster_keywords <- top_words_per_cluster%>%
  as_tibble() %>%
  pivot_longer(cols = contains("V"), names_to = "cluster", values_to = "words") %>%
  group_by(cluster) %>%
  summarise(words = list(words)) %>%
  mutate(words = map(words, paste, collapse = ", ")) %>%
  unnest()

print(cluster_keywords)
```

```
## # A tibble: 10 × 2
##    cluster words
##    <chr>   <chr>
##  1 V1      makeup, one, use, lip, product, love, skin
##  2 V10     feel, doe, veri, smell, face, moistur, sensit
##  3 V2      remov, skin, skin, balm, skin, skin, use
##  4 V3      skin, use, face, use, use, feel, feel
##  5 V4      use, product, like, love, love, product, dri
##  6 V5      love, love, feel, product, feel, use, moistur
##  7 V6      doe, tri, product, dri, like, face, product
##  8 V7      face, moistur, moistur, feel, realli, smell, love
##  9 V8      like, feel, love, hydrat, great, make, veri
## 10 V9      great, like, realli, moistur, veri, great, realli
```

```
# create a tibble linking each document to its assigned cluster
clusters <- tibble(text = names(k_cluster$cluster), cluster = as.character(k_cluster$clu
ster))

# combine document metadata with clustering results
df_clusters <- bind_cols(as_tibble(docvars(reviews_dfm)), clusters)

#  add the original reviews back into the dataset
df_clusters$review_text <- reviews_samp$review_text

# see a random sample of 5 reviews from each cluster
set.seed(2025)
df_clusters %>%
  group_by(cluster) %>%
  sample_n(5) %>%
  select(cluster,review_text) %>%
  print()
```

```
## # A tibble: 35 × 2
## # Groups:   cluster [7]
##    cluster review_text
##    <chr>   <chr>
##  1 1       everyone should have this! It just the best, takes your makeup off I…
##  2 1       Left my skin very clean, felt no residue and was able to to apply ma…
##  3 1       I like to spray this between layers of makeup. I spray it after putt…
##  4 1       I love this spray as a refresher any time I am feeling dry. It is no…
##  5 1       Love this cleanser. It removes light makeup and does make your skin …
##  6 2       I seen this all over my for you page on TikTok and I was scared that…
##  7 2       First week using it and I love it , I easily dry up with certain cle…
##  8 2       I grabbed two of these at the Sephora store while waiting in line, o…
##  9 2       i tried one hundred one different cleansers and i here to tell you t…
## 10 2       I have tried a lot of masks lately. This is one of my favorites! It …
## # ℹ 25 more rows
```

```
df_clusters %>%
  count(cluster) %>%
  arrange(n)  # Sort by smallest cluster
```

```
## # A tibble: 7 × 2
##   cluster     n
##   <chr>   <int>
## 1 1         481
## 2 4         485
## 3 2         532
## 4 6         635
## 5 5        1438
## 6 7        2121
## 7 3        4308
```

The identified keywords across the clusters reveal a significant amount of overlap and similarity, suggesting that the it is hard for clustering algorithm to distinctly separate meaningful topics. Common words like **“love”, “feel”, “skin”, “use”, “moistur”, “product”**, and **“like”** appear repeatedly across multiple clusters. This overlap could be due to the high frequency of general words and plain languages used in reviews. Additionally, some keywords are nearly identical in meaning or context (e.g., “moistur” and “hydration”), probably due to the nature of skincare product reviews, which further complicates the differentiation process.

The sampled reviews from each cluster provide some insights into how the K-means algorithm grouped the data. Similar to the keywords investigation, there is a high frequency of positive reviews across clusters, with many comments expressing satisfaction and enthusiasm for the products. This observation aligns well with the keyword analysis, where words like **“love”, “feel”, “skin”, “moistur”, “product”**, and **“use”** are recurrent and overlap between clusters. Additionally, some clusters contain **reviews about specific product experiences** (e.g., makeup removal, moisturizers, etc.), but the lack of clear separation between clusters suggests that the K-means algorithm is primarily detecting general positive sentiment rather than identifying nuanced topics or themes.

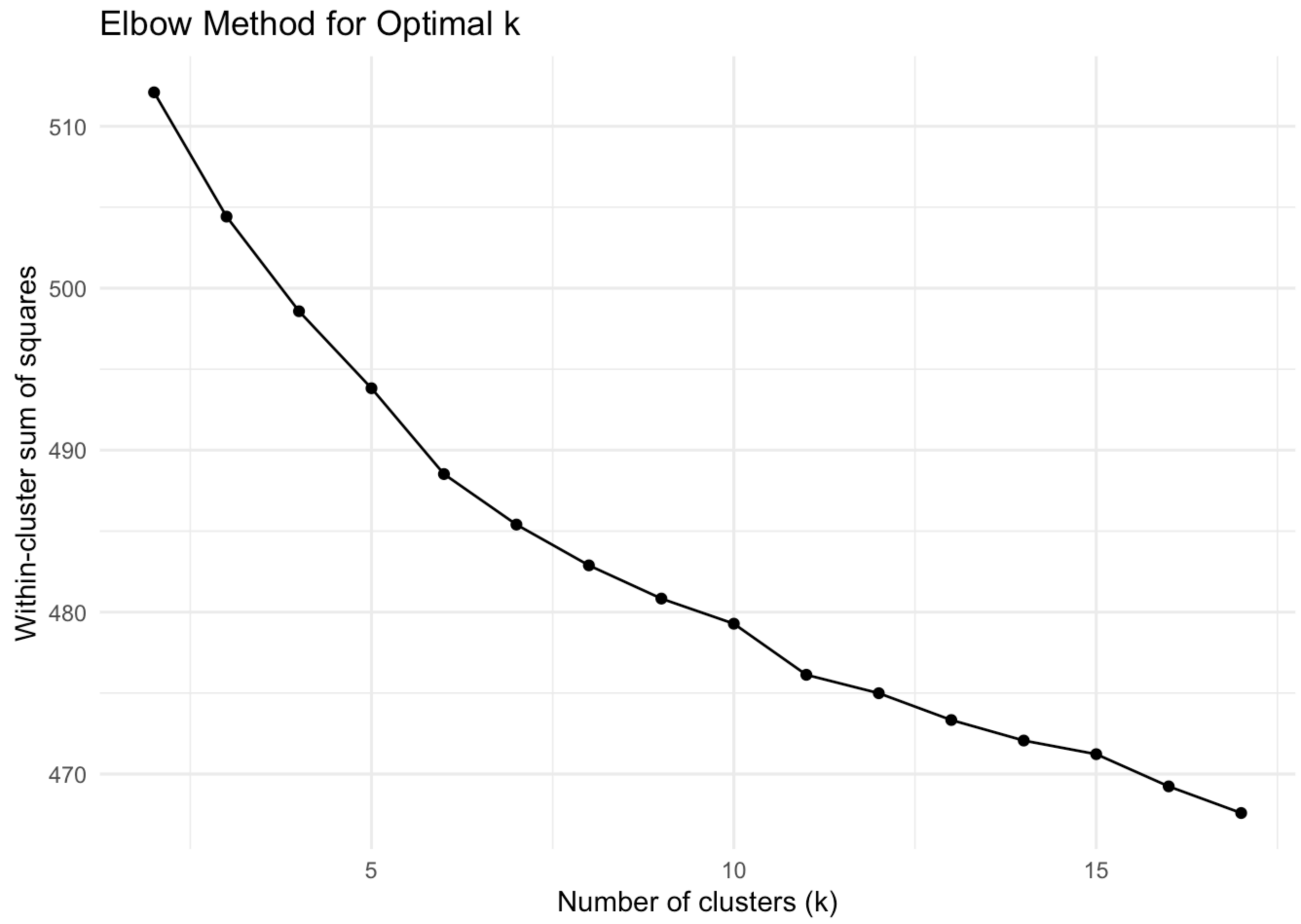
The frequent appearance of positive-sentiment words across various clusters supports the previous observation that Positive Class reviews are more easily identified, whereas Neutral and Negative Class reviews remain underrepresented and hard to distinguish. I think in this case, unsupervised learning methods like k-means clustering might not be ideal. Without labels guiding the clustering, K-means cannot differentiate sentiment effectively if the features are not well-separated and the languages are sometimes vague and not straightforward. It doesn’t have the context of class labels to guide the separation, leading to unsatisfied cluster quality. As K-means relies on distance-based similarity, it is hard for it to distinguish nuanced language patterns or low-frequency words.

```r
# ---- Elbow method
dfm_matrix <- as.matrix(dfm_reviews_prop)

# Define a range of k values to test
k_values <- 2:17
wss_values <- numeric(length(k_values))  # within-cluster sum of squares

# Compute WSS for each k
set.seed(2025)
for (i in seq_along(k_values)) {
  kmeans_result <- kmeans(dfm_matrix, centers = k_values[i], nstart = 5)
  wss_values[i] <- kmeans_result$tot.withinss
}

# plot
elbow_df <- data.frame(k = k_values, wss = wss_values)
ggplot(elbow_df, aes(x = k, y = wss)) +
  geom_line() +
  geom_point() +
  labs(title = "Elbow Method for Optimal k",
       x = "Number of clusters (k)",
       y = "Within-cluster sum of squares") +
theme_minimal()
```



To determine the optimal number of clusters (**k**) for my K-means clustering, I applied the **Elbow Method** by plotting the within-cluster sum of squares against various values of **k**. I experimented with three different settings of the **nstart** parameter: **nstart = 1**, **nstart = 5 (twice)** on **range 2 to 15 (twice)** to **range 2 to 17**, which controls the number of random nationalizations for K-means. The plots show a gradual decrease in within-cluster sum of squares as **k** increases, but the rate of improvement slows down around **k = 7**. At this point, the curve starts to flatten out.

SVM

```
# Convert dfm to matrix
train_matrix <- as.matrix(train_dfm)
test_matrix <- as.matrix(test_dfm)

str(train_matrix)
```

```
##  num [1:8000, 1:7197] 1 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ docs    : chr [1:8000] "text1" "text2" "text3" "text4" ...
##   ..$ features: chr [1:7197] "saw" "immedi" "brighten" "smooth" ...
```

```
str(test_matrix)
```

```
##  num [1:2000, 1:7197] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ docs    : chr [1:2000] "text1" "text2" "text3" "text4" ...
##   ..$ features: chr [1:7197] "saw" "immedi" "brighten" "smooth" ...
```

```
dim(train_matrix)
```

```
## [1] 8000 7197
```

```
dim(test_matrix)
```

```
## [1] 2000 7197
```

```
levels(train_set$review_type)
```

```
## NULL
```

```
levels(test_set$review_type)
```

```
## NULL
```

```
train_set$review_type <- factor(train_set$review_type)
test_set$review_type <- factor(test_set$review_type, levels = levels(train_set$review_ty
pe))

# Fit a SVM model
svm_tuned <- tune(
  svm,
  train.x = train_matrix,
  train.y = as.factor(train_set$review_type),
  kernel = "linear",
  ranges = list(cost = c(0.1, 1, 10)), # trying multiple values of the cost parameter
  tunecontrol = tune.control(cross = 5)
)

svm_model <- svm_tuned$best.model

# predictions and confusion matrix as you had it
predicted_test_svm <- predict(svm_model, newdata = test_matrix)
confusionMatrix(predicted_test_svm, test_set$review_type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Negative Neutral Positive
##   Negative       80      23       90
##   Neutral        35      42       97
##   Positive       69      88     1476
##
## Overall Statistics
##
##                Accuracy : 0.799
##                  95% CI : (0.7808, 0.8164)
##     No Information Rate : 0.8315
##     P-Value [Acc > NIR] : 0.9999
##
##                   Kappa : 0.3422
##
##  Mcnemar's Test P-Value : 0.1275
##
## Statistics by Class:
##
##                      Class: Negative Class: Neutral Class: Positive
## Sensitivity                   0.4348         0.2745          0.8876
## Specificity                   0.9378         0.9285          0.5341
## Pos Pred Value                0.4145         0.2414          0.9039
## Neg Pred Value                0.9424         0.9392          0.4905
## Prevalence                    0.0920         0.0765          0.8315
## Detection Rate                0.0400         0.0210          0.7380
## Detection Prevalence          0.0965         0.0870          0.8165
## Balanced Accuracy             0.6863         0.6015          0.7108
```

Just out of curiosity, I also tried the SVM model, which turns out to have the lowest overall accuracy compared to my precious supervised learning model.

Topic modeling

```r
library(topicmodels)

# Clean dfm - remove some common topic-unrelated words
custom_stopwords <- c("use", "like", "product", "feel", "love", "good", "realli", "get",
"make", "tri", "work", "veri", "just", "day", "look", "even", "great", "much", "becaus",
"never", "skin", "give", "now", "littl", "can", "one", "also", "think", "although")
dfm_cleaned <- dfm_remove(reviews_dfm, pattern = custom_stopwords)

# Convert dfm to dtm
dtm_cleaned <- convert(dfm_cleaned, to = "topicmodels")

# Apply LDA
reviews_lda <- LDA(dtm_cleaned, k = 4, control = list(seed = 1234), alpha = 0.1, eta =
0.1)
reviews_lda
```

```
## A LDA_VEM topic model with 4 topics.
```

```r
terms(reviews_lda, 10)
```

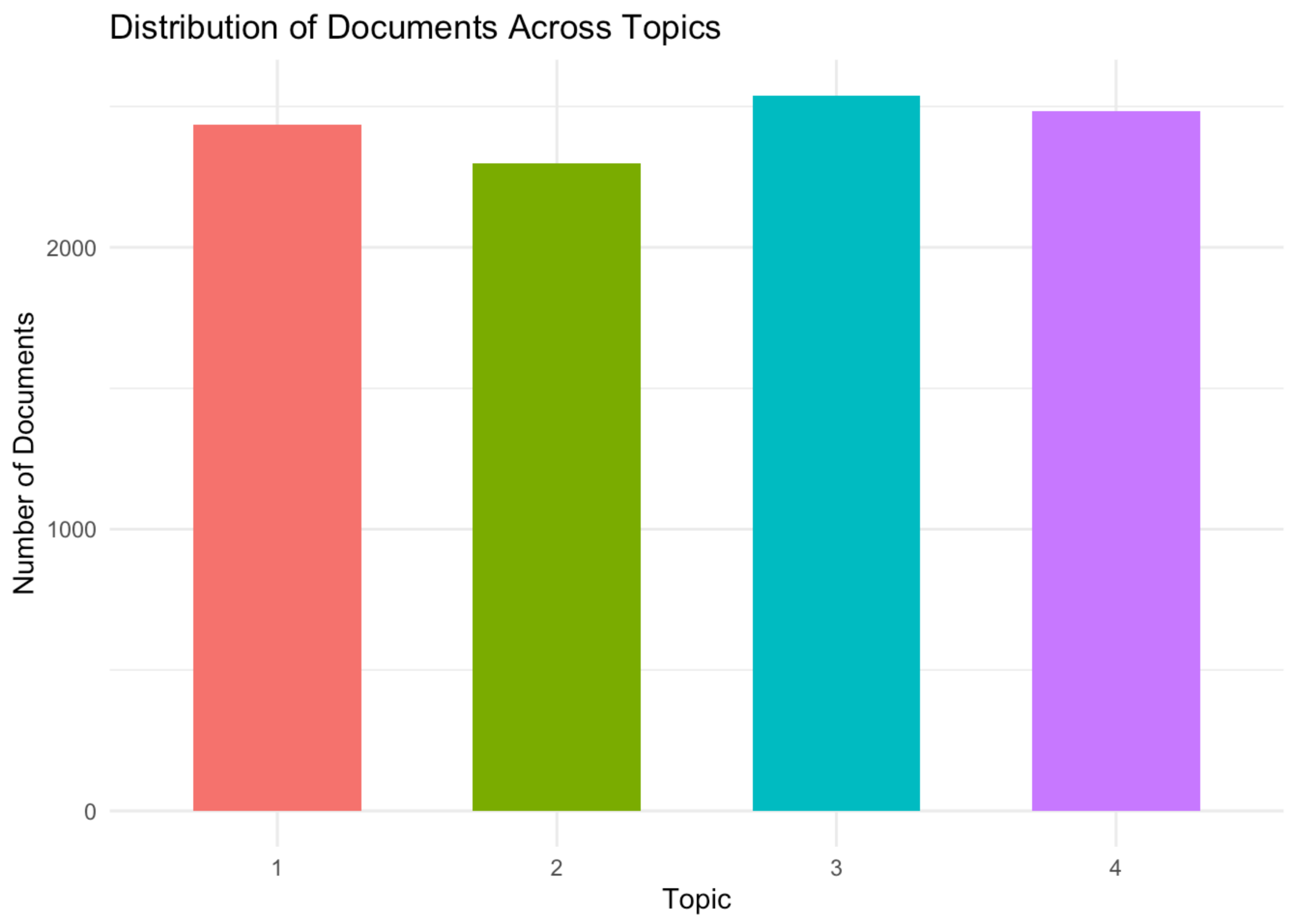
```
##       Topic 1   Topic 2   Topic 3     Topic 4
##  [1,] "moistur" "face"    "face"      "doe"
##  [2,] "dri"     "moistur" "moistur"   "hydrat"
##  [3,] "well"    "time"    "doe"       "dri"
##  [4,] "onli"    "lip"     "cream"     "definit"
##  [5,] "time"    "night"   "textur"    "smell"
##  [6,] "differ"  "acn"     "makeup"    "face"
##  [7,] "see"     "dri"     "recommend" "makeup"
##  [8,] "smell"   "well"    "dri"       "differ"
##  [9,] "lip"     "sensit"  "oili"      "light"
## [10,] "help"    "onli"    "super"     "smooth"
```

```
#  ---- STEP 0: Visualize the topic distribution

# Get the distribution of each document across all topics
topic_distributions <- as.data.frame(posterior(reviews_lda)$topics)

# Count occurrences of each topic (based on the highest probability per document)
topic_distribution_counts <- topic_distributions %>%
  mutate(Dominant_Topic = apply(., 1, which.max)) %>%  # Find the most dominant topic fo
r each document
  group_by(Dominant_Topic) %>%  # Group by dominant topic
  summarise(Count = n())  # Count the number of documents for each topic

ggplot(topic_distribution_counts, aes(x = factor(Dominant_Topic), y = Count, fill = fact
or(Dominant_Topic))) +
  geom_bar(stat = "identity", width = 0.6) +
  labs(title = "Distribution of Documents Across Topics",
       x = "Topic", y = "Number of Documents") +
  theme_minimal() +
  theme(legend.position = "none")
```



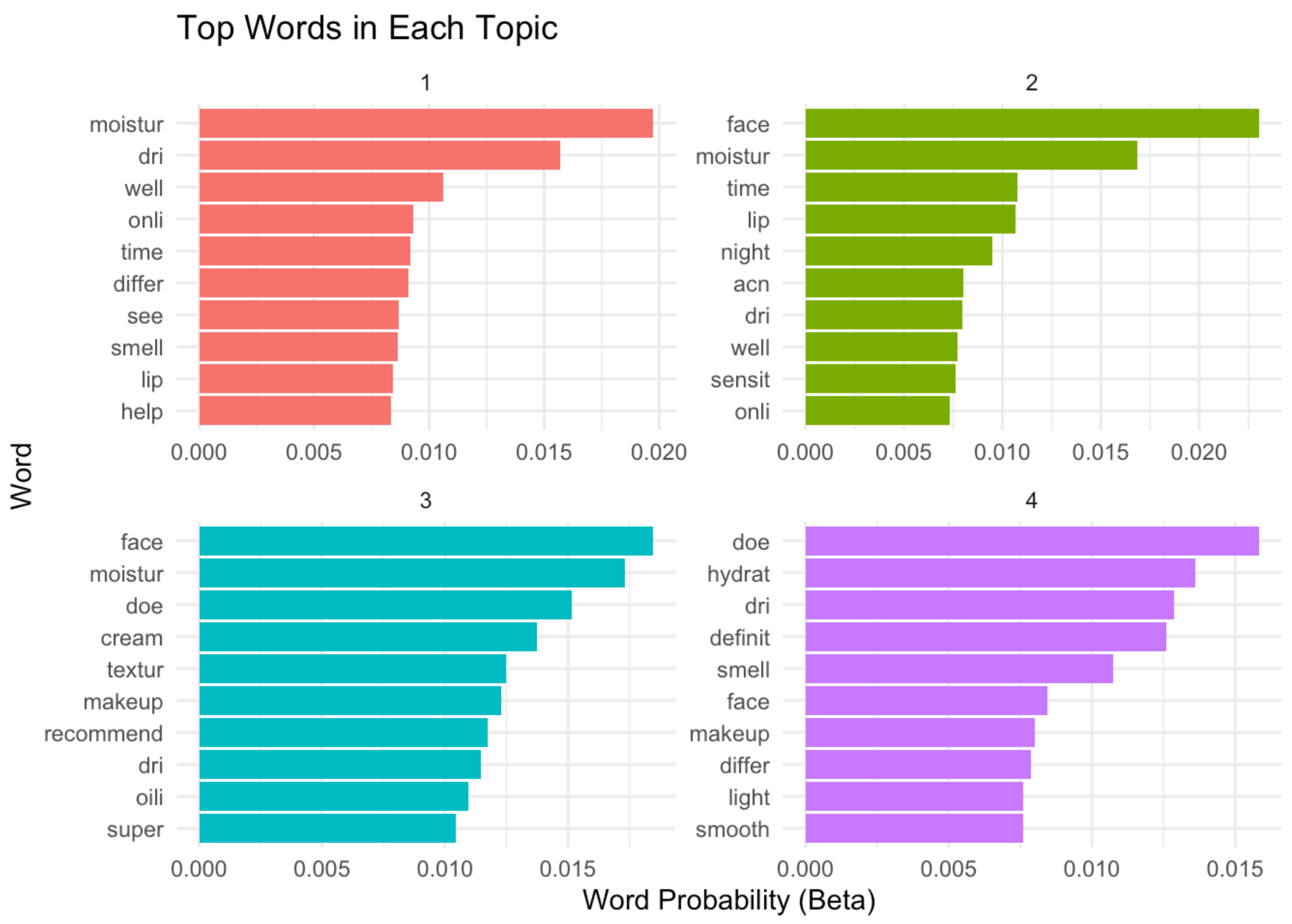
```
#  ---- STEP 1: Extract Word-Topic Probabilities ("Beta")
# (identify the most representative words for each topic)
library(tidytext)
# Extract word-topic probabilities
reviews_topics <- tidy(reviews_lda, matrix = "beta")
head(reviews_topics)
```

```
## # A tibble: 6 × 3
##   topic term      beta
##   <int> <chr>    <dbl>
## 1     1 lip   0.00842
## 2     2 lip   0.0106
## 3     3 lip   0.00146
## 4     4 lip   0.00580
## 5     1 balm  0.00197
## 6     2 balm  0.000335
```

```
# ----- STEP 2: Visualize the Top Words in Each Topic

reviews_top_terms <- reviews_topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

# Plot top words per topic
reviews_top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered() +
  labs(title = "Top Words in Each Topic", x = "Word Probability (Beta)", y = "Word") +
  theme_minimal()
```



The topic modeling results show four distinct categories based on the most frequent words within each topic. The bar chart shows a fairly balanced distribution of documents across the four topics, indicating that each topic is represented with a similar number of documents, suggesting no particular topic is disproportionately dominating the dataset. By looking at the the top words for four categories, we can identify:

- **Topic 1:** This topic focuses on *moisturizing and skin texture*. Key words like "moistur," "dri," "well," and "smell" suggest it relates to how well products keep the skin moisturized and help with dryness.
- **Topic 2:** This topic emphasizes *acne treatment and sensitivity*. Words like "face," "moistur," "acn," and "sensit" suggest discussions around facial skincare routines like acne treatment, and addressing sensitive skin.
- **Topic 3:** This topic is about *product texture and makeup compatibility*. Words like "cream," "makeup," "textur," and "recommend" suggest reviews about how well skincare products work with makeup or their texture and feel.
- **Topic 4:** This topic highlights *hydration and smoothness*. Terms like "doe," "hydration," "smooth," and "differ" suggest a focus on how effective the products are at providing hydration, making skin smooth, and showing noticeable differences. But this

Overall, the topic modeling has successfully identified categories related to skincare, especially focusing on product effectiveness, sensitive and problematic skin care, compatibility with makeup, and general skin texture. We still saw great overlap of words like "moistur" and "face" across multiple topics indicates some redundancy or lack of distinctiveness between topics. Also I noticed that most of the words related to product effectiveness are positive, aligning with the fact that the most dominant category in the corpus is Positive Class.

```
# ----- STEP 3: Identify Words that Differentiate the Most Between the Four Topics

beta_wide <- reviews_topics %>%
  mutate(topic = paste0("topic", topic)) %>%
  pivot_wider(names_from = topic, values_from = beta) %>%
  filter(topic1 > .001 | topic2 > .001 | topic3 > .001 | topic4 > .001) %>%
  rowwise() %>%
  mutate(
    # Identify the topic with the highest and lowest probability for each word
    max_topic = which.max(c_across(starts_with("topic"))),  # Highest probability topic
    min_topic = which.min(c_across(starts_with("topic"))),  # Lowest probability topic

    # Get the highest and lowest probabilities
    max_prob = max(c_across(starts_with("topic"))),
    min_prob = min(c_across(starts_with("topic"))),

    # Calculate log-ratio between the max and min probability
    log_ratio = log2(max_prob / min_prob)
  ) %>%
  ungroup()

# Select top 15 words with highest absolute log-ratio
top_words <- beta_wide %>%
  filter(max_prob > 0.001) %>%
  arrange(desc(log_ratio)) %>%
  slice_head(n = 15)
print(top_words)
```

```
## # A tibble: 15 × 10
##    term     topic1  topic2  topic3  topic4 max_topic min_topic max_prob min_prob
##    <chr>     <dbl>   <dbl>   <dbl>   <dbl>     <int>     <int>    <dbl>    <dbl>
##  1 thing   3.13e-4 3.39e-3 9.55e-7 4.56e-3         4         3  0.00456  9.55e-7
##  2 purchas 1.14e-3 4.28e-6 3.73e-3 7.28e-3         4         2  0.00728  4.28e-6
##  3 claim   7.13e-7 2.13e-4 1.99e-4 1.00e-3         4         1  0.00100  7.13e-7
##  4 bag     1.02e-3 2.22e-6 1.90e-4 3.93e-4         1         2  0.00102  2.22e-6
##  5 differ  9.12e-3 2.02e-5 7.12e-4 7.88e-3         1         2  0.00912  2.02e-5
##  6 mini    3.20e-6 1.80e-4 2.71e-5 1.42e-3         4         1  0.00142  3.20e-6
##  7 fragra… 3.01e-3 9.53e-6 1.20e-3 9.84e-4         1         2  0.00301  9.53e-6
##  8 year    5.33e-3 5.38e-3 7.28e-4 1.72e-5         2         4  0.00538  1.72e-5
##  9 plus    6.73e-6 2.09e-3 7.31e-5 8.05e-4         2         1  0.00209  6.73e-6
## 10 tini    1.30e-3 1.66e-4 6.21e-4 5.57e-6         1         4  0.00130  5.57e-6
## 11 bit     2.69e-3 3.24e-5 4.23e-3 5.06e-3         4         2  0.00506  3.24e-5
## 12 daili   3.72e-3 1.89e-3 1.12e-3 2.47e-5         1         4  0.00372  2.47e-5
## 13 compli… 2.33e-3 4.23e-3 7.41e-4 2.93e-5         2         4  0.00423  2.93e-5
## 14 seen    6.45e-5 1.77e-5 6.29e-4 2.22e-3         4         2  0.00222  1.77e-5
## 15 may     1.60e-5 1.58e-3 5.03e-4 1.05e-3         2         1  0.00158  1.60e-5
## # ℹ 1 more variable: log_ratio <dbl>
```

```
# Visualize Words That Differentiate Topics
# ggplot(top_words, aes(x = reorder(term, log_ratio), y = log_ratio, fill = as.factor(ma
x_topic))) +
  # geom_col() +
  # coord_flip() +  # Flip for better readability
  # labs(
    # title = "Words That Differentiate Topics",
    # x = "Word",
    # y = "Log2 Ratio (Max Topic / Min Topic)"
  # ) +
  # scale_fill_manual(values = RColorBrewer::brewer.pal(4, "Dark2"),
                   # labels = c("Topic 1", "Topic 2", "Topic 3", "Topic 4")) +
  # theme_minimal()
```

When I firstly looked at the the graph illustrating the most distinguishing words associated with each of the four topics derived from the topic modeling process. Though each topic appears to capture a different aspect of user reviews, it didn't really align with the four topics we identified earlier using top words. Also, it is hard for us to identify specific patterns from these words that can inform us some business insights. That's because some words here have high log-ratios but very low probabilities - meaning they may only appear once or twice. They're distinct, but not necessarily meaningful. So after adding a filtering on `max_prob` , we now can look at distinct words but that also appear with at least some higher frequency. Also, I deleted visualization here but only looked at the table for better interpretation.

Based on the results, we are identifying words that are distinctive and have relatively higher frequencies within certain topics, which can give us more nuances to our previously identified projects. For example, in **Topic 1**, words like "bag", "differ", "fragranc", and "daili" are prominent. Previously, Topic 1 was associated with **moisturizing and skin texture** (based on words like "moistur", "dri", "well", and "smell"). If we combine these findings, Topic 1 seems to be about **daily skincare routines and moisturizing products**—items you can put in a bag, use daily, and that make a difference in skin feel or texture. Similarly, **Topic 2** was previously identified as related to **acne treatment and sensitivity**, based on words such as "face", "moistur", "acn", and "sensit". Here, words like "year", "plus", and "complimentari" appear as distinctive terms, suggesting this topic might be about **long-term use and complimentary offers associated with acne treatment products**, which also makes sense as consumers often look for products that provide lasting results and may be attracted by complimentary samples or special deals related to acne treatment.