HelpfulReviewPrediction

Simon Lee

2025-10/20

```
# 1. Load Libraries
packages<-c("jsonlite", "dplyr", "text2vec", "xgboost", "pROC", "Matrix", "LiblineaR", "ggplot2")
for(p in packages){
  if(!require(p, character.only=TRUE)) install.packages(p)
  library(p, character.only=TRUE)
}
```

```
## Loading required package: jsonlite
```

```
## Warning: package 'jsonlite' was built under R version 4.5.2
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## Loading required package: text2vec
```

```
## Warning: package 'text2vec' was built under R version 4.5.2
```

```
## Loading required package: xgboost
```

```
## Warning: package 'xgboost' was built under R version 4.5.2
```

```
## Loading required package: pROC
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

## Loading required package: Matrix

## Loading required package: LiblineaR

## Warning: package 'LiblineaR' was built under R version 4.5.2

## Loading required package: ggplot2
```

```r
# 2. Load JSONL file
temporary<-file("All_Beauty.jsonl", open="r")
reviews<-jsonlite::stream_in(temporary, verbose=FALSE)
close(temporary)

# 3. Prepare data
df<-reviews%>%
  select(text, helpful_vote)%>%
  filter(!is.na(text))%>%
  mutate(helpful_flag=helpful_vote>0,
         review_length=nchar(text))

set.seed(123)
  df_sample<-df%>%sample_n(80000)

# Train/test split
set.seed(123)
train_idx<-sample(seq_len(nrow(df_sample)), size=0.8*nrow(df_sample))
df_train<-df_sample[train_idx, ]
df_test<-df_sample[-train_idx, ]

# 4. Text pre-processing
library(text2vec)
prep_fun<-tolower
tok_fun<-word_tokenizer

it_train<-itoken(df_train$text, preprocessor=prep_fun, tokenizer=tok_fun, progressbar=TRUE)
it_test<-itoken(df_test$text,  preprocessor=prep_fun, tokenizer=tok_fun,  progressbar=TRUE)
```

1. Introduction

This report records the methods used to determine whether a product review is helpful or not helpful with a 2023 dataset of amazon reviews for beauty products provided by McAuley Lab. https://cseweb.ucsd.edu/~jmcauley/datasets.html#amazon_reviews

2. Exploratory Data

The dataset contains:

- the written review
- number of helpful votes
- additional metadata (product ID)

We label binary variable 1 as helpful and 0 as unhelpful.

2.1 Distribution Plot

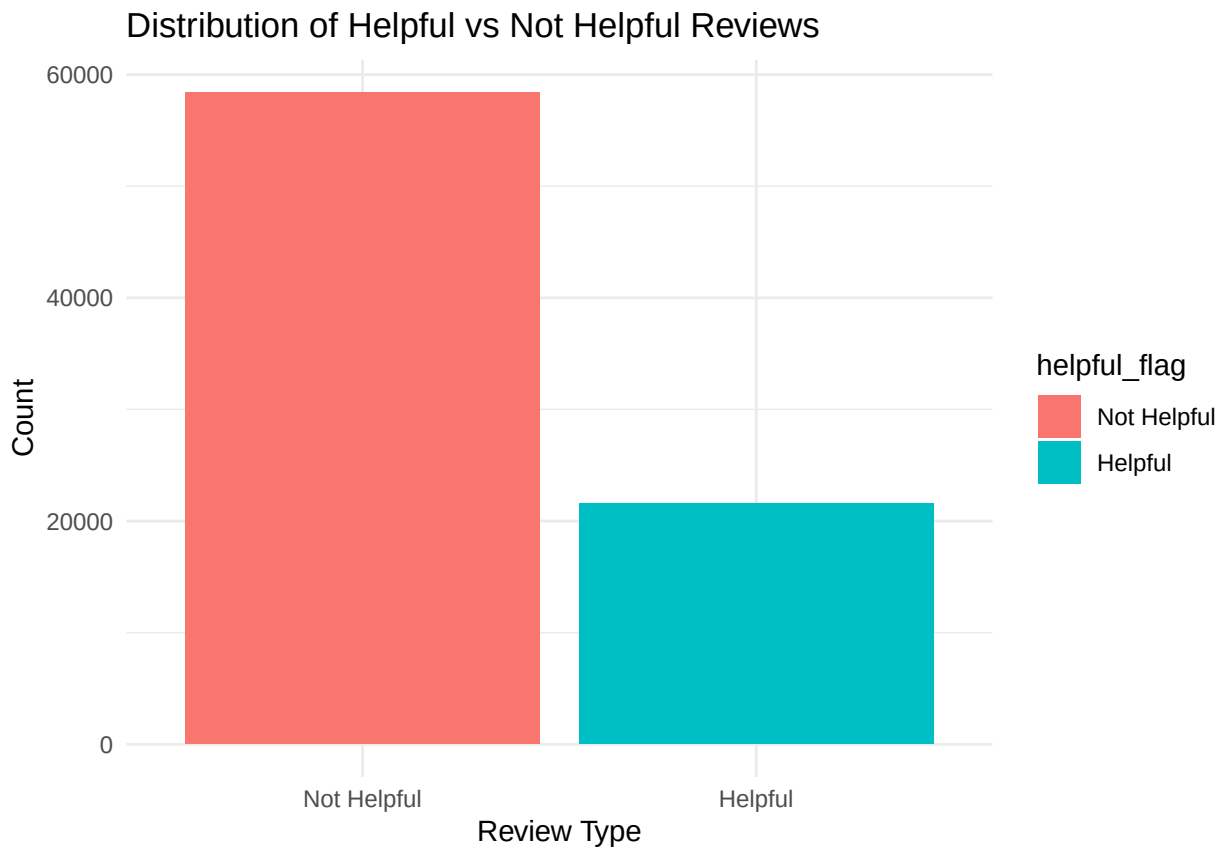
```
#5. Graphs
library(dplyr)
library(ggplot2)

# Classify "helpful"
df_sample<-df_sample%>%
  mutate(helpful_flag=ifelse(helpful_vote>0, 1, 0))

# Class distribution
table(df_sample$helpful_flag)
```

```
##
##     0     1
## 58410 21590
```

```
prop.table(table(df_sample$helpful_flag))
```

```
##
##        0        1
## 0.730125 0.269875
```

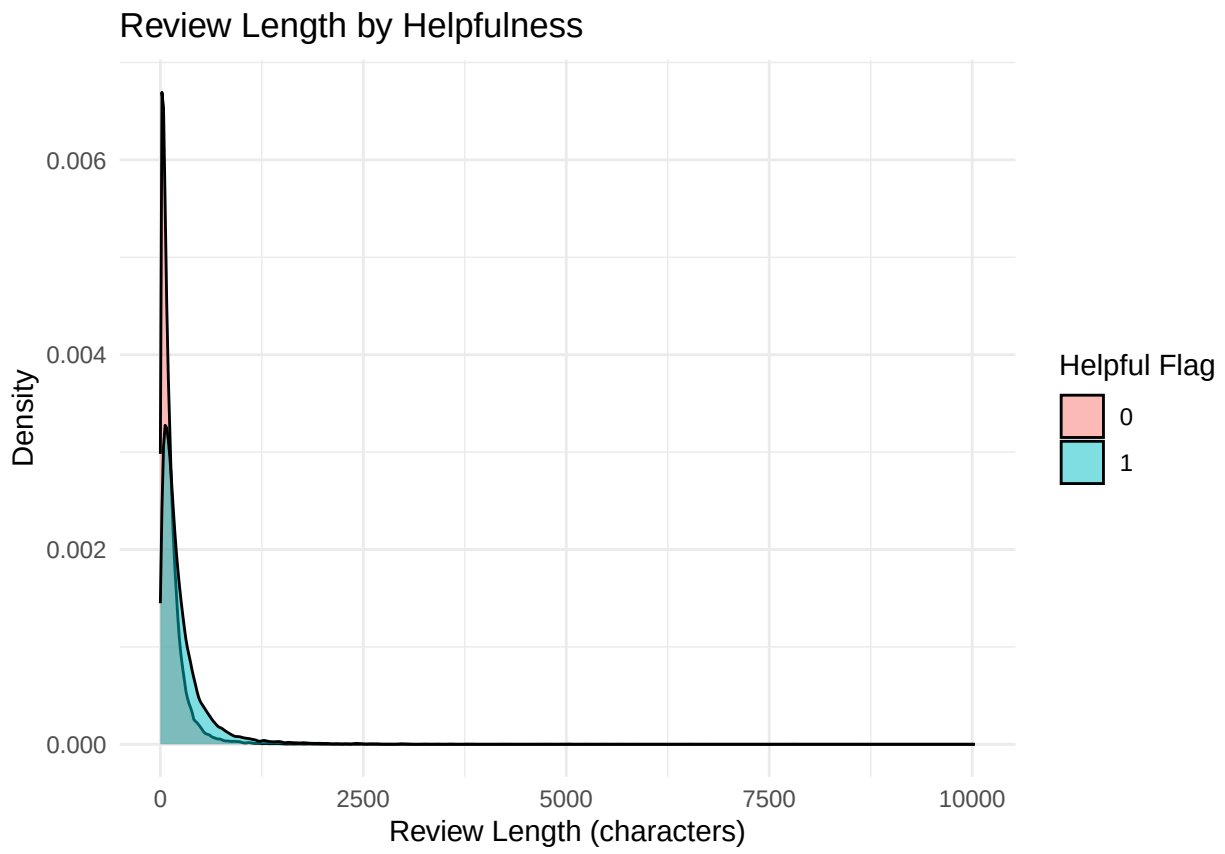
```
# Distribution plot
df_sample %>%
  mutate(helpful_flag=factor(helpful_flag, labels=c("Not Helpful", "Helpful")))%>%
  ggplot(aes(x=helpful_flag, fill=helpful_flag))+
  geom_bar()+
  theme_minimal()+
  labs(title="Distribution of Helpful vs Not Helpful Reviews",
       x="Review Type", y="Count")
```



There are less helpful than not helpful reviews, creating an imbalanced classification problem.

2.2 Review Length by Helpfulness

```
# Review length by helpfulness
df_sample<-df_sample%>%
  mutate(review_length=nchar(text))

ggplot(df_sample, aes(x=review_length, fill=factor(helpful_flag)))+
  geom_density(alpha=0.5)+
  theme_minimal()+
  labs(title="Review Length by Helpfulness",
       x="Review Length (characters)", y="Density", fill="Helpful Flag")
```



We observed that helpful reviews tend to be longer. Since text data is readily available, TF–IDF is used for representing sparse, high-dimensional text.

3. Train/Test Split and TF-IDF Matrix

```
# 6. Create TF-IDF matrix
vocab<-create_vocabulary(it_train)
vocab<-prune_vocabulary(vocab, term_count_min=5)
vectorizer<-vocab_vectorizer(vocab)
dtm_train<-create_dtm(it_train, vectorizer)
dtm_test<-create_dtm(it_test, vectorizer)

tfidf<-TfIdf$new()
dtm_train_tfidf<-cbind(dtm_train, df_train$review_length)
dtm_test_tfidf<-cbind(dtm_test, df_test$review_length)


# Add review length as feature
dtm_train_tfidf<-cbind(dtm_train_tfidf, df_train$review_length)
dtm_test_tfidf<-cbind(dtm_test_tfidf, df_test$review_length)

# Labels
y_train<-as.numeric(df_train$helpful_flag)
y_test<-as.numeric(df_test$helpful_flag)
```

3.1 Train/Test Split

We induce a train/test split to be able to simulate real world scenario by comparing the training set and prediction to the test set, which the model has not seen before. Earlier, we had chosen an 80/20 train test split to avoid over-fitting and under-fitting as well as giving sufficient data for training set as well as reducing variance for test set. A 50/50 split would yield in an unnecessary small training set for the model to learn, while a 90/10 split might result in a test set with too much noise.

3.2 TF-IDF

TF: Term Frequency

$$\frac{\text{number times a word appears in a review}}{\text{total number of words}}$$

The more a word appears, the more likely it is important.

IDF: Inverse Document Frequency

$$\frac{\text{total number of reviews}}{\text{number of reviews containing said word}}$$

Reduces the significance of common words or stopwords. The less a word appears across documents, the more likely it is important. The balance created by the TF-IDF matrix makes it a useful tool for natural language processing.

4. Models

4.1 SVM

Support vector machine is a fast and simple model for classification and a good baseline model, and it works well with TF-IDF.

```
# 7. Train SVM
X_svm_train<-as.matrix(dtm_train_tfidf)
```

```
## Warning in asMethod(object): sparse->dense coercion: allocating vector of size
## 4.5 GiB
```

```
X_svm_test<-as.matrix(dtm_test_tfidf)
```

```
## Warning in asMethod(object): sparse->dense coercion: allocating vector of size
## 1.1 GiB
```

```
svm_model<-LiblineaR(data=X_svm_train, target=y_train, type=1, cost=1, bias=TRUE)

# 8. Predict SVM
pred_svm<-predict(svm_model, X_svm_test)$predictions
pred_svm_class<-ifelse(pred_svm>0, 1, 0)
```

4.2 XGBoost

```
# 9. Train XGBoost
total_positive<-sum(y_train==1)
total_negative<-sum(y_train==0)
scale_pos_weight<-total_negative/total_positive

dtrain<-xgb.DMatrix(data=dtm_train_tfidf, label=y_train)
dtest<-xgb.DMatrix(data=dtm_test_tfidf, label=y_test)

params<-list(
  objective="binary:logistic",
  eval_metric="auc",
  max_depth=6,
  eta=0.1,
  scale_pos_weight=scale_pos_weight
)

xgb_model<-xgb.train(
  params=params,
  data=dtrain,
  nrounds=150,
  verbose=1
)

# 10. F1-optimal threshold (using train set only)
pred_train_prob<-predict(xgb_model, dtrain)
thresholds<-seq(0, 1, 0.01)

f1_scores<-sapply(thresholds, function(t){
  p<-ifelse(pred_train_prob>t, 1, 0)
  precision<-sum(p&y_train)/sum(p)
  recall<-sum(p&y_train)/sum(y_train)
  if(is.na(precision)|is.na(recall)) return(0)
  2*precision*recall/(precision+recall)
})

best_threshold<-thresholds[which.max(f1_scores)]
message("F1-optimal threshold: ", best_threshold)
```

```
## F1-optimal threshold: 0.51
```

```
# 11. Predict on test set using XGBoost
pred_test_prob<-predict(xgb_model, dtest)
pred_test_class<-ifelse(pred_test_prob>best_threshold, 1, 0)
```

A more advanced model compared to SVM that handles non-linearity and class imbalance (as mentioned in section 2.1).

Evaluation

Precision

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives(TP)} + \text{False Positives}}$$

How many reviews did the model predict as "helpful" are truly helpful?

Recall

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives(TP)} + \text{False Negatives}}$$

How many "helpful" reviews did the model correctly identify?

F1

$$\text{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1 is the harmonic mean of precision and recall.

AUC The higher the AUC, the better it is at ranking helpful reviews above non-helpful reviews, regardless of a specific threshold. It is a general score evaluation of the model.

```
# Evaluate SVM
conf_matrix_svm<-table(Predicted=pred_svm_class, Actual=y_test)
print(conf_matrix_svm)
```

```
##          Actual
## Predicted    0    1
##         0 5449 1048
##         1 6209 3294
```

```
precision_svm<-sum(pred_svm_class&y_test)/sum(pred_svm_class)
recall_svm<-sum(pred_svm_class&y_test)/sum(y_test)
f1_svm<-2*precision_svm*recall_svm/(precision_svm+recall_svm)
auc_svm<-pROC::roc(y_test, as.vector(pred_svm))$auc
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
message("SVM Precision: ", round(precision_svm, 3))
```

```
## SVM Precision: 0.347
```

```
message("SVM Recall: ", round(recall_svm, 3))
```

```
## SVM Recall: 0.759
```

```r
message("SVM F1: ", round(f1_svm, 3))
```

```
## SVM F1: 0.476
```

```r
message("SVM AUC: ", round(auc_svm, 3))
```

```
## SVM AUC: 0.613
```

```r
# Evaluate XGBoost
conf_matrix_xgb<-table(Predicted=pred_test_class, Actual=y_test)
print(conf_matrix_xgb)
```

```
##          Actual
## Predicted    0    1
##         0 8310 2035
##         1 3348 2307
```

```r
precision_xgb<-sum(pred_test_class&y_test)/sum(pred_test_class)
recall_xgb<-sum(pred_test_class&y_test)/sum(y_test)
f1_xgb<-2*precision_xgb*recall_xgb/(precision_xgb+recall_xgb)
auc_xgb<-pROC::roc(y_test, pred_test_prob)$auc
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
message("XGBoost Precision: ", round(precision_xgb,3))
```

```
## XGBoost Precision: 0.408
```

```r
message("XGBoost Recall: ", round(recall_xgb,3))
```

```
## XGBoost Recall: 0.531
```

```r
message("XGBoost F1: ", round(f1_xgb,3))
```

```
## XGBoost F1: 0.462
```

```r
message("XGBoost AUC: ", round(auc_xgb,3))
```

```
## XGBoost AUC: 0.671
```

Conclusion

Both models achieve the purpose of meaningfully determining whether a review is helpful or not. SVM provides a baseline model while XGBoost is more powerful and addresses class imbalances. The above is a basic implementation of both models, and future work can aim for further optimization of models as well as incorporation of other confounding variables.