Text Classification - Customer Churn Use Case

Promothesh Chatterjee*

Why Customer Churn Analysis?

Customer churn analysis is crucial for many businesses for several reasons:

1. **Customer Retention is Cheaper than Acquisition**: It generally costs significantly more to acquire a new customer than it does to retain an existing one. By identifying and addressing the reasons for customer churn, a business can increase its customer retention rate and decrease its costs.
2. **Identifying Problems and Opportunities for Improvement**: Churn analysis can help a business identify areas where it's underperforming. For instance, if customers are churning because of poor customer service, then improving in this area could reduce churn.
3. **Increase Customer Lifetime Value (CLV)**: By reducing churn, a business can increase the average lifetime value of its customers. This means more revenue over the long term.
4. **Forecasting and Planning**: Churn analysis can also help with forecasting and planning. By understanding its churn rate, a business can make more accurate predictions about its future revenue.
5. **Improving Product or Service**: If customers are leaving due to problems with the product or service, this is a clear indication that improvements are needed. Identifying these issues can lead to better products or services and ultimately to more satisfied customers.
6. **Segmentation**: Understanding which types of customers are most likely to churn can help a business tailor its customer retention strategies more effectively. For example, if a certain demographic is more likely to churn, a business might target them with specific marketing campaigns or offers.

Let's understand how can we perform customer churn analysis with text data.

Example: "Predicting Customer Churn at XYZ Telecom: A Naive Bayes Approach"

Scenario: Over the past few years, XYZ Telecom has been experiencing high customer churn rates. With competition in the industry intensifying, retaining existing customers has become more critical than ever. The company has collected a wealth of data from customer interactions, including customer reviews. They believe these reviews may hold clues about why customers are leaving.

Let's first create the dataset for analysis.

```
### Creating the dataset
set.seed(123)
library(tidyverse)

# Define a set of possible sentences for the reviews
sentences <- c("I love the service.",
               "The customer service was terrible.",
               "I am unhappy with the pricing.",
               "The quality of the service was great.",
               "I had an awful experience.",
               "The customer support was excellent.",
               "The service was satisfactory.",
               "I am not happy with the service.",
```

```
                "The pricing was very reasonable.",
                "I had a bad experience with the customer support.")

# Generate 500 random reviews
reviews <- map_chr(1:500, ~ paste(sample(sentences, 5, replace = TRUE), collapse = " "))

# Generate a random tenure between 1 and 60 months
tenure <- sample(1:60, 500, replace = TRUE)

# Generate a binary churn variable, with a 20% churn rate
churn <- case_when(
    runif(500) < 0.2 ~ 1,
    TRUE ~ 0
)

# Generate demographic information
age <- sample(18:75, 500, replace = TRUE) # Age
gender <- sample(c("Male", "Female"), 500, replace = TRUE) # Gender
location <- sample(c("Urban", "Suburban", "Rural"), 500, replace = TRUE) # Location

# Generate usage data
usage <- runif(500, min = 1, max = 100) # Usage could represent hours of use, data consumed, etc.

# Generate financial data
payment_history <- sample(c("On Time", "Late"), 500, replace = TRUE, prob = c(0.7, 0.3)) # Payment hist

# Generate customer service interactions
num_interactions <- rpois(500, lambda = 2) # Number of customer service interactions

# Combine into a tibble (tidyverse version of a data frame)
data <- tibble(reviews, tenure, churn, age, gender, location, usage, payment_history, num_interactions)
```

This R code is creating a simulated dataset for 500 customers. Here is a breakdown of what each part does:

1. `set.seed(123)`: This sets the seed for random number generation, which allows for the random operations performed in this script to be reproducible.
2. `sentences`: This is a character vector that contains possible sentences that could be in a customer's review.
3. `reviews`: This uses the `map_chr` function from the `purrr` package to generate 500 random reviews. Each review is a combination of 5 sentences chosen randomly from the `sentences` vector.
4. `tenure`: This generates a random number between 1 and 60 (inclusive) for each of the 500 customers, representing the length of their relationship with the company in months.

5. `churn`: This creates a binary variable that indicates whether each customer has churned (left the company). The `case_when` function is used to assign a 20% probability of churn (represented by 1) and an 80% probability of not churning (represented by 0).
6. `age`: This generates a random age between 18 and 75 (inclusive) for each customer.
7. `gender`: This randomly assigns a gender ("Male" or "Female") to each customer.
8. `location`: This randomly assigns a location ("Urban", "Suburban", or "Rural") to each customer.
9. `usage`: This generates a random usage score between 1 and 100 (inclusive) for each customer. This could represent something like hours of use or data consumed.
10. `payment_history`: This randomly assigns a payment history ("On Time" or "Late") to each customer, with a 70% probability of being "On Time" and a 30% probability of being "Late".
11. `num_interactions`: This generates a random number of customer service interactions for each customer using the Poisson distribution, with a lambda (average rate of occurrence) of 2. The Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space. In this context, it's used to simulate the number of customer service interactions.

    It's chosen because it's particularly appropriate for count data—data where the values represent the number of occurrences of an event. The properties of the Poisson distribution make it a good choice for modeling events that happen independently (i.e., one event doesn't affect the probability of the next event) and at a relatively constant average rate.

    In this code, it's assumed that customer service interactions occur independently (one interaction doesn't influence the next) and at an average rate of 2 interactions per customer (that's the 'lambda = 2' part).

    Also, the Poisson distribution is skewed, with a long tail on the right, which is a common pattern in real-world situations. For instance, you might have many customers with a few interactions, but a few customers with a large number of interactions, which the Poisson distribution can capture.
12. `data`: This combines all of the above vectors into a tibble (a type of data frame used in the `tidyverse` set of packages), with each vector becoming a column in the tibble.

Next we perform, preprocessing and analysis.

```
# Create a corpus from the 'reviews' variable
library(quanteda)
library(quanteda.textmodels)
library(caret)

corp <- corpus(data$reviews)

# Preprocess the text data
toks <- tokens(corp, remove_punct = TRUE, remove_numbers = TRUE, remove_symbols = TRUE)
toks <- tokens_tolower(toks)
```

```
toks <- tokens_remove(toks, stopwords("en"))
toks <- tokens_wordstem(toks, language = "english")

# Create a Document-Feature Matrix (DFM)
dfmat <- dfm(toks)

# Split the DFM and the 'churn' variable into training (80%) and test (20%) sets
set.seed(123)
trainIndex <- createDataPartition(data$churn, p = .8, list = FALSE, times = 1)

# Use dfm_subset() instead of normal subsetting
dfmat_train <- dfm_subset(dfmat, docnames(dfmat) %in% docnames(dfmat)[trainIndex])
dfmat_test <- dfm_subset(dfmat, !(docnames(dfmat) %in% docnames(dfmat)[trainIndex]))

churn_train <- data$churn[trainIndex]
churn_test <- data$churn[-trainIndex]
```

This R code uses a Naive Bayes classifier based on the text of the reviews, along with whether or not the customer churned, to build a predictive model. Here is a breakdown:

1. `corp <- corpus(data$reviews)`: This creates a corpus from the reviews. A corpus is a large and structured set of texts.
2. The next few lines preprocess the text data. This involves tokenization, converting to lowercase, removing stopwords, and stemming:
    - `tokens(corp, remove_punct = TRUE, remove_numbers = TRUE, remove_symbols = TRUE)`: This breaks the text into individual words or "tokens", and removes punctuation, numbers, and symbols.
    - `tokens_tolower(toks)`: This converts all the tokens to lowercase.
    - `tokens_remove(toks, stopwords("en"))`: This removes common words (like "and", "the", etc.) that are often filtered out in text analysis.
    - `tokens_wordstem(toks, language = "english")`: This reduces words to their root or stem (e.g., "running" becomes "run").
3. `dfmat <- dfm(toks)`: This creates a Document-Feature Matrix (DFM). In this matrix, each row represents a document (review in this case) and each column represents a feature (unique word).
4. The next few lines split the data into training and test sets. The training set is used to train the model, and the test set is used to evaluate its performance.
    - `trainIndex <- createDataPartition(data$churn, p = .8, list = FALSE, times = 1)`: This creates an index to split the data, with 80% going to the training set.
    - The DFM is being subsetted into a training set and a test set. Here's how it works:

`dfmat_train <- dfm_subset(dfmat, docnames(dfmat) %in% docnames(dfmat)[trainIndex])`: This line is creating the training set from the DFM. The `dfm_subset` function is used to subset a DFM. The first

argument is the DFM to subset, and the second argument is a logical vector that specifies which documents (rows) to include in the subset.

The logical vector is created with the expression 'docnames(dfmat) %in% docnames(dfmat)[trainIndex]'. The 'docnames' function is used to get the names of the documents (rows) in the DFM. The '%in%' operator is used to check if each element in the first vector (the names of all the documents in the DFM) is present in the second vector (the names of the documents in the training set). This creates a logical vector where each element is 'TRUE' if the corresponding document is in the training set, and 'FALSE' otherwise. This logical vector is used to subset the DFM, so that only the documents in the training set are included.

`dfmat_test <- dfm_subset(dfmat, !(docnames(dfmat) %in% docnames(dfmat)[trainIndex]))`: This line is creating the test set from the DFM. It works the same way as the previous line, but the logical vector is negated with the `!` operator. This means that it's `TRUE` for the documents that are not in the training set, and `FALSE` for the documents that are. So this line subsets the DFM to include only the documents that are not in the training set, which forms the test set.

So, in summary, these lines of code are using the `dfm_subset` function to subset the DFM into a training set and a test set, based on a logical vector that indicates which documents should be included in the training set.

- 'churn_train' and 'churn_test' are the churn indicators for the training and test sets, respectively.

-

Next, we will train our Naive Bayes classifier and predict which customers are likely to churn.

```
# Train a Naive Bayes classifier
model <- textmodel_nb(dfmat_train, churn_train)



dfmat_test_matched <- dfm_match(dfmat_test, featnames(dfmat_train))

predictions <- predict(model, newdata = dfmat_test_matched)

predictions <- factor(predictions, levels = unique(churn))
churn_test <- factor(churn_test, levels = unique(churn))
confusionMatrix(predictions, churn_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1  9 34
##          0 11 46
##
##                Accuracy : 0.55
##                  95% CI : (0.4473, 0.6497)
##     No Information Rate : 0.8
```

```
##     P-Value [Acc > NIR] : 1.00000
##
##                   Kappa : 0.0175
##
##  Mcnemar's Test P-Value : 0.00104
##
##             Sensitivity : 0.4500
##             Specificity : 0.5750
##          Pos Pred Value : 0.2093
##          Neg Pred Value : 0.8070
##              Prevalence : 0.2000
##          Detection Rate : 0.0900
##    Detection Prevalence : 0.4300
##       Balanced Accuracy : 0.5125
##
##        'Positive' Class : 1
##
```

```
data_test <- data[-trainIndex, ] # get the test data
data_test$prediction <- predictions
likely_to_churn <- data_test %>% filter(prediction == 1)
head(likely_to_churn)
```

```
## # A tibble: 6 x 10
##   reviews               tenure churn   age gender location usage payment_history
##   <chr>                  <int> <dbl> <int> <chr>  <chr>    <dbl> <chr>
## 1 I am unhappy with th~     29     1    48 Female Urban    17.7  Late
## 2 I am not happy with ~     58     1    19 Male   Suburban 64.1  On Time
## 3 The service was sati~      8     0    52 Female Urban    56.8  On Time
## 4 I love the service. ~     42     0    52 Male   Urban     1.53 Late
## 5 I had a bad experien~     49     0    45 Male   Rural    74.1  On Time
## 6 I love the service. ~     51     0    70 Female Urban    21.7  On Time
## # i 2 more variables: num_interactions <int>, prediction <fct>
```

1. `model <- textmodel_nb(dfmat_train, churn_train)`: This trains a Naive Bayes classifier using the training data. The classifier will be used to predict whether a review is associated with a customer who churned.

2. `dfmat_test_matched <- dfm_match(dfmat_test, featnames(dfmat_train))`: This matches the features in the test DFM to those in the training DFM. This is done because the classifier can only make predictions based on the features it was trained on.

3. `predictions <- predict(model, newdata = dfmat_test_matched)`: This uses the trained model to make predictions on the test data.

4. `predictions <- factor(predictions, levels = unique(churn))` and `churn_test <- factor(churn_test, levels = unique(churn))`: These lines convert the predictions and actual churn indicators to factors

with the same levels. This is necessary for the `confusionMatrix()` function.

5. `confusionMatrix(predictions, churn_test)`: This creates a confusion matrix, which is a table that is often used to describe the performance of a classification model. It compares the actual outcomes (whether a customer churned) to the predicted outcomes (whether the model predicted that they would churn).

The last few lines of code are used to identify the customers who are likely to churn according to the predictive model.

- `data_test <- data[-trainIndex, ]`: This line is creating a new dataset, `data_test`, which contains only the test data. The `-trainIndex` is used to select all the rows of the `data` that were not included in the training set.
- `data_test$prediction <- predictions`: This line is adding a new column to the `data_test` dataframe. The new column, named `prediction`, contains the predicted outcomes for each customer in the test set. The predicted outcomes were generated by the Naive Bayes classifier that was trained earlier.
- `likely_to_churn <- data_test %>% filter(prediction == 1)`: This line is using the `filter` function from the `dplyr` package to select only the rows of `data_test` where `prediction` equals 1. In the context of this analysis, a prediction of 1 means that the customer is predicted to churn. The result is a new dataframe, `likely_to_churn`, which contains only the customers who are likely to churn according to the model.
- `likely_to_churn`: This line is simply printing the `likely_to_churn` dataframe.

We have only used the text data in our model, now let's use other variables that are available to us in the dataset. We will use the previously created dataset for this analysis.

```
library(tidyverse)
library(quanteda)
library(e1071)
library(caret)

# Convert the DFM to a data frame and bind it with the original data
dfmat_df <- convert(dfmat, to = "data.frame")
data_full <- bind_cols(data, dfmat_df)

# Split the data into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(data_full$churn, p = .8, list = FALSE, times = 1)
train_data <- data_full[trainIndex,]
test_data <- data_full[-trainIndex,]

# Remove 'reviews' column from training and test datasets
train_data <- train_data %>% select(-reviews)
test_data <- test_data %>% select(-reviews)
```

```r
# Train a Naive Bayes classifier
model <- naiveBayes(churn ~ ., data = train_data)

# Make predictions on the test set
predictions <- predict(model, newdata = test_data)

# Convert to factor
predictions <- as.factor(predictions)
test_data$churn <- as.factor(test_data$churn)

# Ensure the same levels
levels(predictions) <- levels(test_data$churn)

# Print confusion matrix
confusionMatrix(predictions, test_data$churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 72 16
##          1  8  4
##
##                Accuracy : 0.76
##                  95% CI : (0.6643, 0.8398)
##     No Information Rate : 0.8
##     P-Value [Acc > NIR] : 0.8686
##
##                   Kappa : 0.1176
##
##  Mcnemar's Test P-Value : 0.1530
##
##             Sensitivity : 0.9000
##             Specificity : 0.2000
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.3333
##              Prevalence : 0.8000
##          Detection Rate : 0.7200
##    Detection Prevalence : 0.8800
##       Balanced Accuracy : 0.5500
##
##        'Positive' Class : 0
##
```

```r
data_test <- data[-trainIndex, ] # get the test data
data_test$prediction <- predictions
likely_to_churn <- data_test %>% filter(prediction == 1)
head(likely_to_churn)
```

```
## # A tibble: 6 x 10
##   reviews               tenure churn   age gender location usage payment_history
##   <chr>                  <int> <dbl> <int> <chr>  <chr>    <dbl> <chr>
## 1 I am not happy with ~     58     1    19 Male   Suburban 64.1  On Time
## 2 The service was sati~      8     0    52 Female Urban    56.8  On Time
## 3 I love the service. ~     42     0    52 Male   Urban     1.53 Late
## 4 The customer support~     12     1    27 Female Urban    72.7  On Time
## 5 I love the service. ~     14     0    23 Male   Suburban  4.06 On Time
## 6 The customer support~     40     0    63 Female Suburban  2.95 On Time
## # i 2 more variables: num_interactions <int>, prediction <fct>
```