Text Mining and Sentiment Analysis

Prof. Annamaria Bianchi
A.Y. 2024/2025

Lecture 15

14 April 2025

UNIVERSITÀ DEGLI STUDI DI BERGAMO
Dipartimento di Scienze Aziendali

Outline

Preprocessing with quanteda

SA with Machine Learning: Naive Bayes

Packages: **quanteda, quanteda.textmodels**

Quanteda

Quanteda is an R package built to be used with textual data–perhaps from books, Tweets, or transcripts–to both manage that data (sort, label, condense, etc.) and analyze its contents.

Two common forms of analysis with quanteda are sentiment analysis and content analysis.

```
> install.packages('quanteda')

> library(quanteda)
```

Quanteda basics

There are three major components of a text as understood by quanteda:

- **corpus**: A corpus is an object within R that we create by loading our text data into R and using the `corpus()` command. It is only by turning our data into a corpus format that quanteda is able to work with and process the text we want to analyze. A corpus holds documents separately from each other and is typically unchanged as we conduct our analysis.
- **tokens**: tokens are typically each individual word in a text. This default can be changed to be sentences or characters instead if we want.
- **document-feature matrix (dfm)**: The dfm is the analytical unit on which we will perform analysis. A dfm puts the documents into a matrix format. The rows are the original texts and the columns are the features of that text (often tokens). The value in the matrix is typically word count. Typically, this matrix contains a lot of zeros

Dataset

Let us consider Trip Advisor Hotel reviews. The reviews have been labelled and have associated a sentiment (positive or negative). The dataset contains 3 variables:

- `text`
- `doc_id`
- `sentiment`

```
> reviews = read_rds(file = "reviews.rds")
> reviews
# A tibble: 10,475 x 3
   text                                 doc_id sentiment
   <chr>                                 <dbl> <chr>
 1 unique, great stay, wonderful ~          1 positive
 2 great stay great stay, went se~          2 positive
 3 love monaco staff husband stay~          3 positive
 4 cozy stay rainy city, husband ~          4 positive
 5 hotel stayed hotel monaco crui~          5 positive
 6 excellent stayed hotel monaco ~          6 positive
 7 horrible customer service hote~          7 negative
 8 fantastic stay monaco seattle ~          8 positive
 9 good choice hotel recommended ~          9 positive
10 service service service spent ~         10 positive
# ... with 10,465 more rows
```

Preprocessing with quanteda

For the pre-processing part, we could move from the tidy format to the quanteda dfm using the function `tidytext::cast_dfm()`.

Here we start using quanteda from the beginning to see how it works.

So, we create the corpus, we tokenize, we remove stopwords, punctuation, numbers and we also stem the words.

Stemming is very useful when implementing ML algorithms in order to reduce the dimensionality and speed up the estimation process.

Preprocessing with quanteda

The function `corpus()` creates a corpus object from available sources:

- a character vector, consisting of one document per element;
- a data frame (or a tibble tbl_df), whose default document id is a variable identified by `docid_field`; the text of the document is a variable identified by `text_field`; and other variables are imported as document-level meta-data.
- ….

```
corpus(x, ...)
```

`x`	a valid corpus source object
`text_field`	the character name or numeric index of the source data.frame indicating the variable to be read in as text, which must be a character vector. All other variables in the data.frame will be imported as docvars. This argument is only used for data.frame objects (including those created by readtext).

Output. A **corpus class object** containing the original texts, document-level variables, document-level metadata, corpus-level metadata, and default settings for subsequent processing of the corpus.

Preprocessing with quanteda

The function `tokens()` constructs a tokens object

```
tokens(x, what = "word", remove_punct = FALSE, remove_symbols = FALSE,
remove_numbers = FALSE, remove_url = FALSE,... )
```

`x`	the input object to the tokens constructor, one of: a (uniquely) named list of characters; a tokens object; or a corpus or character object that will be tokenized
`what`	which tokenizer to use.
`remove_punct`	logical; if TRUE remove all characters
`remove_numbers`	logical; if TRUE remove tokens that consist only of numbers, but not words that start with digits, e.g. 2day
`remove_url`	logical; if TRUE find and eliminate URLs beginning with http(s)

Output: **quanteda tokens class object**, by default a serialized list of integers corresponding to a vector of types.

Preprocessing with quanteda

The function `token_remove()` discards tokens from a tokens object. The most common usage for `tokens_remove` will be to eliminate stop words from a text or text-based object

```
tokens_remove(x, pattern, ...)
```

`x` tokens object whose token elements will be removed

`pattern` a character vector, list of character vectors

Usually we remove function words (grammatical words) that have little or no substantive meaning in pre-processing. `stopwords()` returns a pre-defined list of function words.

Preprocessing with quanteda

The function `token_wordstem()` applies a stemmer to words. This is a wrapper to `wordStem` designed to allow this function to be called without loading the entire **SnowballC** package.

```
tokens_wordstem(x, language = quanteda_options("language_stemmer"))
```

`x`	a tokens object whose word stems are to be removed. If tokenized texts, the tokenization must be word-based.
`language`	the name of a recognized language

Preprocessing with quanteda

```
> corpus <- tokens(corpus,
+ remove_punct = TRUE,
+ remove_number = TRUE) |>
+ tokens_remove(pattern = stopwords("en")) |>
+ tokens_wordstem() > corpus
```

```
Tokens consisting of 10,475 documents and 1 docvar.
1 :
 [1] "uniqu"  "great"  "stay"   "wonder" "time"   "hotel"  "monaco"
 [8] "locat"  "excel"  "short"  "stroll" "main"
[ ... and 74 more ]

2 :
 [1] "great"   "stay"    "great"   "stay"    "went"    "seahawk"
 [7] "game"    "awesom"  "downfal" "view"    "build"   "n't"
[ ... and 168 more ]
```

Preprocessing with quanteda

The function `dfm()` construct a sparse document-feature matrix, from a character, corpus, tokens, or even other dfm object.

```
dfm(x,…)
```

`x`	a tokens or dfm object

Preprocessing with quanteda

Next, we create the dfm, i.e., the document-feature matrix based on the bag of words representation.

```
> dfm <- dfm(corpus)
> dfm
Document-feature matrix of: 10,475 documents, 39,788 features (99.82% sparse) and 1 docvar.
    features
docs uniqu great stay wonder time hotel monaco locat excel short
   1     1     3    2      1    1     4      2     1     2     1
   2     0     4    2      1    1     3      0     1     0     0
   3     0     0    3      1    0     2      1     0     0     0
   4     0     1    2      0    0     1      1     2     0     0
   5     1     2    1      0    0     3      1     1     0     0
   6     0     1    1      0    0     1      1     0     1     0
[ reached max_ndoc ... 10,469 more documents, reached max_nfeat ... 39,778 more features ]
```

NB with quanteda

Steps to follow:

- We need to split our data in training and test data
- Next we train the model using the training data set - function `quanteda.textmodels::textmodel_nb()`
- We match the features (words) in the training model with those in the test set – function `dfm_match()`
- We test the performance on the test set – function `predict()`
- Evaluate performance through proper metrics – package caret

NB with quanteda

We split our data into a training set (to train the model) and a test set (to test the performance of the model on new labelled data). If the results are satisfying we can eventually use the model on new unlabelled data.

To select the training set, we use two functions: `base::sample() + quanteda::dfm_subset()`

`base::sample()` takes a sample of the specified size from a set of elements

`sample(x, size, replace = FALSE, …)`

`x`	either a vector of one or more elements from which to choose
`size`	a non-negative integer giving the number of items to choose.
`replace`	should sampling be with replacement?

NB with quanteda

`quanteda::dfm_subset()` returns document subsets of a dfm that meet certain conditions, including direct logical operations on docvars (document-level variables)

```
dfm_subset(x, subset, ...)
```

`x` dfm object to be subsetted

`subset` logical expression indicating the documents to keep: missing values are taken as false

To define the `subset` argument, we use `quanteda::docid()` function, which gets (or sets) the document ids of a corpus, tokens, or dfm object. It returns an internal variable denoting the original "docname" from which a document came.

```
docid(x)
```

`x` the object with docnames

NB with quanteda

We create two dfms: `dfm_training` and `dfm_test`

```
> set.seed(6272)
> id_train <- sample(1:nrow(reviews), 0.7*nrow(reviews), replace =
FALSE)
> head(id_train, 10)
[1] 5897 9107 8787 1325 3929 7993 2359 7584 3633 10330
> dfm_training <- dfm_subset(dfm, docid(dfm) %in% id_train)
> dfm_test <- dfm_subset(dfm, !docid(dfm) %in% id_train)
```

NB with quanteda

After splitting the data, we use the function `quanteda.textmodels::textmodel_nb()`, which fits a multinomial or Bernoulli Naive Bayes model, given a dfm and some training labels. It requires to specify the training set, the column with the label and other arguments including the prior type that we do not discuss.

```
textmodel_nb(x, y, ….)
```

`x`	the dfm on which the model will be fit. Does not need to contain only the training documents.
`y`	vector of training labels associated with each document identified in train.

NB with quanteda

```
> tmod_nb <- textmodel_nb(dfm_training, dfm_training$sentiment)
> summary(tmod_nb)
Call:
textmodel_nb.dfm(x = dfm_training, y = dfm_training$sentiment)

Class Priors:
(showing first 2 elements)
negative positive
     0.5      0.5

Estimated Feature Scores:
             uniqu    great     stay    wonder     time   hotel    monaco    locat
negative 1.354e-05 0.001551 0.009683 0.0005011 0.004449 0.01855 2.031e-05 0.001598
positive 1.797e-04 0.012222 0.014570 0.0033047 0.005729 0.02689 2.287e-05 0.006810
             excel     short    stroll      main  downtown      shop     area
negative 0.0003995 0.0003995 2.708e-05 0.0005281 0.0001287 0.0004401 0.001652
positive 0.0046034 0.0008952 1.552e-04 0.0010651 0.0003823 0.0019276 0.003102
               pet   friend    room      show      sign      anim      hair     smell
negative 6.771e-05 0.001138 0.02071 0.0004740 0.0004537 2.708e-05 0.0002573 0.0011985
positive 3.267e-05 0.004844 0.01913 0.0005881 0.0001960 1.699e-04 0.0001193 0.0002254
             suit     sleep       big    stripe   curtain      pull    close
negative 0.000799 0.0010631 0.0006094 6.771e-06 0.0002912 0.0002641 0.001070
positive 0.001678 0.0004737 0.0013297 8.168e-06 0.0001144 0.0001045 0.001817
```

NB with quanteda

Next we test the performance on the test data using `dfm_match()` and `predict()` functions.

`dfm_match()` matches the feature set of a dfm to a specified vector of feature names. For existing features in x for which there is an exact match for an element of features, these will be included. Any features in `x` not `features` will be discarded, and any feature names specified in `features` but not found in `x` will be added with all zero counts.

```
dfm_match(x, features)
```

`x`	a dfm
`features`	character; the feature names to be matched in the output dfm

Selecting on another dfm's `featnames()` is useful when you have trained a model on one dfm, and need to project this onto a test set whose features must be identical.
`quanteda::featnames()` gets the features from a dfm, which are stored as the column names of the dfm object. It returns a character vector of the feature labels

NB with quanteda

`quanteda::predict()` implements class predictions using trained Naive Bayes examples

```
predict(object, newdata = NULL, ...)
```

`object`	a fitted Naive Bayes textmodel
`newdata`	dfm on which prediction should be made

NB with quanteda

```
> dfm_matched <- dfm_match(dfm_test, features = featnames(dfm_training))
> predicted_class <- predict(tmod_nb, newdata = dfm_matched)
> predicted_class
       3        8       16       19       21       25       34       38       40       44       45 
positive positive positive positive positive positive negative positive positive negative positive 
      48       49       50       51       52       59       61       62       66       69       71 
positive positive positive positive positive positive positive positive positive negative negative 
      76       77       80       82       90       91       95       96       98      100      104 
```

NB with quanteda

Finally, we compare the predicted classes with the actual ones. In addition, we can use the confusionMatrix function from the caret package that returns some performance metrics.

```
> actual_class <- dfm_matched$sentiment
> tab_class <- table(predicted_class, actual_class)
> tab_class
               actual_class
predicted_class negative positive
       negative      364       15
       positive       57     2707
> 
```

PRED. / ACT.	NEG	POS
NEG	TN	FN
POS	FP	TP
Tot.	N	P

- **TP**: True positive;
- **TN**: True Negative;
- **FP**: False Positive;
- **FN**: False Negative

NB with quanteda

`caret::confusionMatrix()` calculates a cross-tabulation of observed and predicted classes with associated statistics

```
confusionMatrix(data, positive = NULL, prevalence = NULL, mode =
"sens_spec", ... )
```

`data`	a factor of predicted classes (for the default method) or an object of class table
`positive`	an optional character string for the factor level that corresponds to a "positive" result (if that makes sense for your data). If there are only two factor levels, the first level will be used as the "positive" result.
`mode`	a single character string either "sens_spec", "prec_recall", or "everything"

NB with quanteda

```
> confusionMatrix(tab_class, positive = "positive", mode = "everything")
```

We look at:

- **Accuracy**: rate of correctly classified documents (TN+TP)/(N+P)
- **Sensitivity**: true positive rate TP/P
- **Specificity**: true negative rate TN/N
- **Balanced accuracy**: is useful when the dataset is imbalanced (Sensitivity+Specificity)/2

Moreover, in real applications you should think about how to deal with unbalanced data and other techniques such as cross validation.

```
Confusion Matrix and Statistics

               actual_class
predicted_class negative positive
       negative      364       15
       positive       57     2707

               Accuracy : 0.9771
                 95% CI : (0.9712, 0.982)
    No Information Rate : 0.8661
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8969

 Mcnemar's Test P-Value : 1.352e-06

            Sensitivity : 0.9945
            Specificity : 0.8646
         Pos Pred Value : 0.9794
         Neg Pred Value : 0.9604
              Precision : 0.9794
                 Recall : 0.9945
                     F1 : 0.9869
             Prevalence : 0.8661
         Detection Rate : 0.8613
   Detection Prevalence : 0.8794
      Balanced Accuracy : 0.9295
```

Exercise for you

We want now to apply the obtained NB classifier to new (unlabeled) data. Consider the dataset `sentiment_lexicon.rds` containing the polarity of Boston Airbnb apartments according to bing, afinn, nrc, and udpipe.

1) Prepare the data using quanteda preprocessing.

2) Apply the NB classifier obtained from the Trip Advisor Hotel reviews (developed in class).

3) Check how many times all the methods have provided the same results.

4) Using cross tabulations, compare the results obtained with the NB classifier with those obtained from the other methods. With which of the previous methods the NB agrees the most?