

Text Classification - Naive Bayes

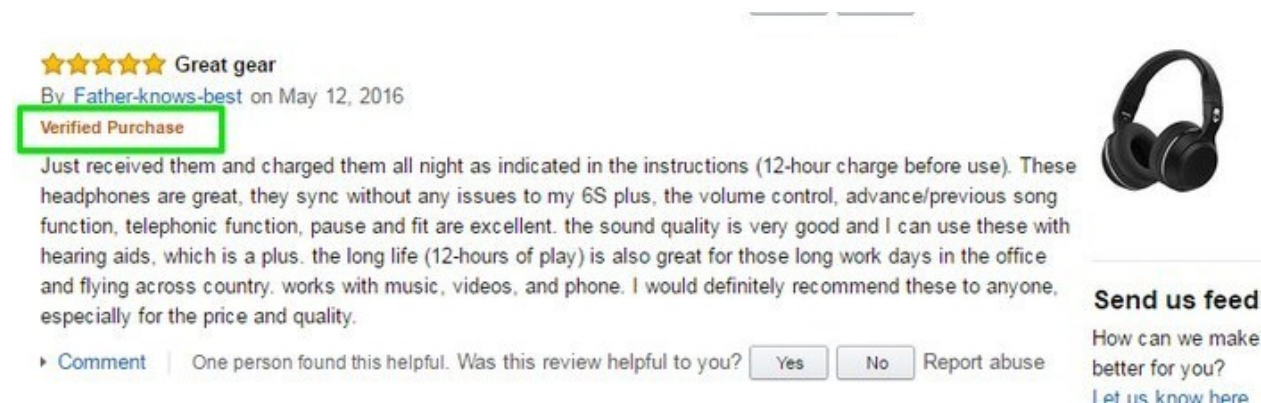
Promothesh Chatterjee*

*Copyright© by Promothesh Chatterjee. All rights reserved. No part of this note may be reproduced, distributed, transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the author.

Text Classification

Text classification is the task of assigning any kind of topic category to any piece of text. Text classification has many important applications in modern life. For instance, text classification algorithms help classify the incoming emails into different labels such as Inbox, Updates, Spam etc. As another example, consider a job posting website as Indeed.com. The job posting website classifies listings into different categories of Marketing, Finance, IT etc. using classification algorithms. For e-retailers like Amazon, given a product description, the algorithms can classify them into different product categories such as Books, Shoes, Clothing etc. Other applications of text classification include authorship attribution (out of Federalist Papers some anonymous ones were shown to be written by Madison and others by Hamilton using Bayesian methods-Mosteller and Wallace 1963) and the important commercial application of sentiment analysis.

So how can we classify text into different categories? Take these review posted on Amazon:



The screenshot shows an Amazon product review for a pair of headphones. At the top, there are five yellow stars and the text "Great gear". Below this, it says "By Father-knows-best on May 12, 2016". A green box highlights the text "Verified Purchase". The review text reads: "Just received them and charged them all night as indicated in the instructions (12-hour charge before use). These headphones are great, they sync without any issues to my 6S plus, the volume control, advance/previous song function, telephonic function, pause and fit are excellent. the sound quality is very good and I can use these with hearing aids, which is a plus. the long life (12-hours of play) is also great for those long work days in the office and flying across country. works with music, videos, and phone. I would definitely recommend these to anyone, especially for the price and quality." To the right of the text is an image of the headphones. Below the review text, there is a "Comment" link and a section for helpfulness: "One person found this helpful. Was this review helpful to you?" with "Yes" and "No" buttons, and a "Report abuse" link. On the far right, there is a section titled "Send us feed" with the text "How can we make better for you?" and a link "Let us know here".

One can make hand-coded rules. For instance, if the review contains words such as great, excellent, good etc., we can classify it as a positive review, and vice versa. The problem with these hand-coded rules is that building and maintaining these rules are expensive. So generally, we combine this with some machine learning algorithm (we first look at Naive Bayes).

Intuitive Explanation of Naive Bayes

Let's understand this with a simple explanation. Consumers and policy-makers are very concerned that some unethical companies post fake reviews. Now let's say that these fake reviews have a characteristic that they use high degree of superlatives in their reviews. For instance, let's say that a review containing words such as 'fabulous' and 'astonishing' has a high probability of being a fake review. For convenience, assume that we have a dataset of 100 reviews where we know which are fake and which are genuine reviews. Obviously, then we can find out how many of the fake reviews contain the words 'fabulous' and 'astonishing' and similarly we can count the frequency of these words for the genuine reviews.

	Fake		Not Fake	
Total	20		80	
Fabulous	16	80%	4	5%
Astonishing	10	50%	12	15%
Fabulous & Astonishing	20 x .4	40%	80 x 0.0075	0.75%

$$20 \times 0.4 = 8$$

$$80 \times 0.0075 = 0.6$$

$$\frac{8}{8 + 0.6} = 93.02\%$$

Let's make some numbers up to understand this better. Let's assume that out of the 100 reviews, we know that 20 are fake and 80 are genuine (first row of the table above). Next, we count how many of the fake reviews (row 2 of the table above) have the word 'fabulous', say that's 16. So, 80% (i.e. 16/20) of the fake reviews have the word 'fabulous'. We compute the same for the genuine reviews and find that 4/80 i.e., 5% genuine reviews have the word 'fabulous'. We do similar calculations for the word 'astonishing' (row 3 of the table above). 10/20 that is, 50% of fake reviews have the word 'astonishing' whereas 12/80 (15%) have the word present in genuine reviews. Now, we want to know what is probability that a review is spam if both the words 'fabulous' and 'astonishing' are present? If we continue, with our present strategy of counting, there might occur a few issues. 1) If in our small sample of 100, none of the reviews in the genuine category show both the words together, that puts the probability of both words occurring in genuine reviews as zero. And more importantly, it implies that any time both the words occur in a review, we are 100% sure that the review is a spam (which obviously is a big claim to make!). 2) Additionally, when the number of reviews is large, it would be cumbersome to count them. So what do we do?

Here comes the basic idea of Naive Bayes. We make the 'naive' assumption that the occurrence of both the words are independent events. This allows us to generate the joint probability of both words occurring together, simply by multiplying their individual probabilities (more on these later). For instance, the probability of both 'fabulous' and 'astonishing' occurring together in the fake reviews is simply the product of their individual probabilities, $80\% \times 50\% = 40\%$. Similarly, the probability of both words co-occurring in genuine reviews is $5\% \times 15\% = 0.75\%$. Once we have the probabilities, we can generate the predicted count of instances for both the words in fake as well as genuine reviews. Thus, the predicted number of reviews in the fake category that contain both the words are simple, total number of reviews multiplied by the probability, $20 \times 0.4 = 8$. Similarly, in the genuine category, we expect a total of $80 \times 0.0075 = 0.6$ (this is expected value so it may not be an integer) reviews to contain both the words. So out of total of $8 + 0.6$ instances where both the words co-occurred, 8 were in the fake category. Thus, if a review contains both the words, 'fabulous' and 'astonishing', $8/8.6$ (i.e., 93.02 %) chances that the review is fake. This is the basic idea underlying Naive Bayes. Now let's formalize the relationship, so that we can extend the number of cases to beyond 2.

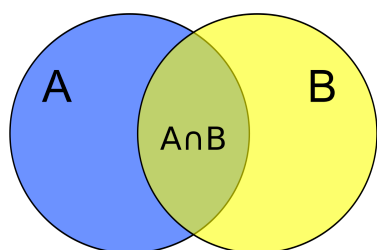
Math underlying Naive Bayes

For independent events, we know that $p(AB) = p(A).p(B)$. For instance, probability of two consecutive Heads showing up in two coin tosses is simply $0.5 \times 0.5 = 0.25$ as both coin tosses are independent events.

For correlated events, though, the same relationship does not hold true. For example, if, the probability of getting a heart attack for a normal person is 0.05 and Jack's brother had a heart attack, what is the probability of both of them having a heart attack? If we apply the above rule, the answer will be 0.05×0.05 , but that would not be correct. If one brother has a heart attack, because of genetic factors, the other brother's chances will be higher. It is in situations like these, that conditional probability comes into picture. Thus, for dependent events, we have:

$p(AB) = p(B).p(A|B)$. How did we get to this equation?

Let's visualize this with a Venn diagram:



Remember that probability of any event is = Number of ways it can happen | Total number of events

Here we are trying to compute $p(A|B)$ that is, probability of A given that B has occurred. In other words, our sample space has shrunk to B, i.e., the yellow area in the venn diagram denotes instances of B having occurred. The number of instances where A and B coincide (since we want to find out instances of A given B) is the area of A intersection B (also called as the area common to A and B). Thus, probability of $A \cap B$ or $p(A.B)$ becomes the numerator and $p(B)$ becomes denominator.

$$p(A|B) = \frac{p(A.B)}{p(B)} \text{ which leads to } p(AB) = p(B).p(A|B)$$

Similarly, we can infer that $p(AB) = p(A).p(B|A)$. We can equate both these equations to derive the Bayes' theorem.

$$p(B).p(A|B) = p(A).p(B|A)$$

$$p(A|B) = \frac{p(A).p(B|A)}{p(B)}$$

So how can we use the Bayes' theorem formulation for text classification?

The key assumption in Naive Bayes is that each feature makes an independent and equal contribution to the outcome. Continuing with the previous example: Occurrence of words such as 'fabulous' and 'astonishing' are taken as independent events. We know that may not be the case in real life (and hence the term 'naive') but though the independence assumption is practically never true but often works well in practice.

For text classification, we have as input:

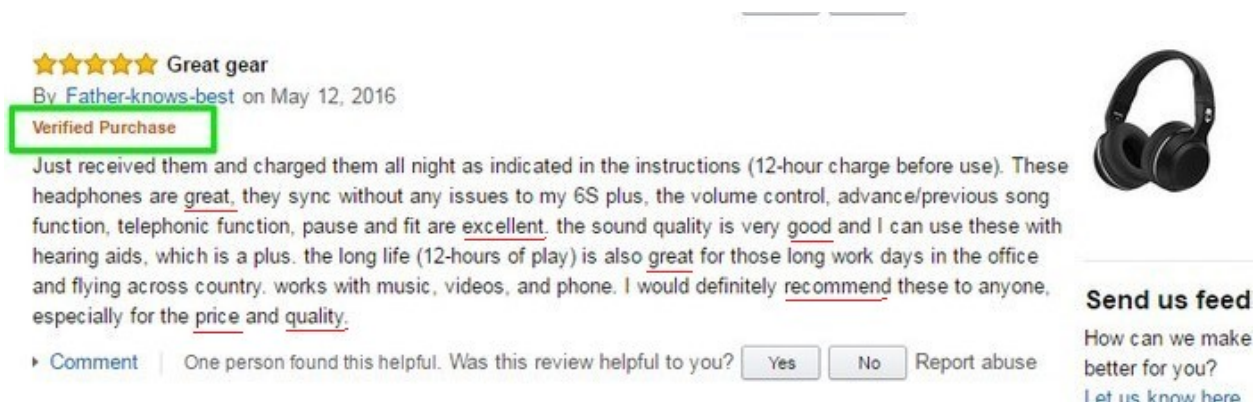
- a document d
- a fixed set of j classes $C = \{c_1, c_2, \dots, c_j\}$
- a training set of m hand-labeled documents $(d_1, c_1), (d_2, c_2), \dots, (d_m, c_m)$

Output is a learned classifier $\gamma : d \rightarrow c \in C$, i.e, given a new document d , the function γ will give a class c .

The Naive Bayes algorithm is one of the most important algorithms for text classification and it relies on the bag of words representation. For instance, let's take the amazon review that we had discussed previously (represented by TEXT in the equation below). Our job here is to build this function gamma which takes the document and returns a class. The class here could be fake or genuine.

$$\gamma(\text{TEXT}) = c$$

One way the classifier might work is to look at the individual words in the document such as 'great', 'excellent', 'price' etc. Or it may look at the all the words. In either case, in the bag of words representation, we lose the information contained in the order of the words and get a vector of words with their counts instead.



The classifier function γ will take the bag of words representation and assign a class 'fake' or 'genuine' (or whatever the classification task is).

Bayes Theorem Applied to Classification Task For a document d and a class c , our goal is to compute for each class of it's conditional probability given a document.

$$p(c|d) = \frac{p(d|c).p(c)}{p(d)}$$

where $p(c|d)$ is termed Posterior Probability, $p(d|c)$ is likelihood, $p(c)$ is class prior probability, $p(d)$ is document prior probability

Then, we are going to use these probabilities to pick the best class. More formally,

$$c_{MAP} = \operatorname{argmax} p(c|d) \text{ where,}$$

c_{MAP} is the best maximum a-posteriori class i.e, the class that we are looking to assign the document to.

What is argmax? argmax is a mathematical function that returns the argument (arg) for the target function that returns the maximum (max) value from the target function.

For instance, let's consider a function $f(x)$ that takes a variable x (say $x= 1,2,3,4,5$) and computes the square of the x .

$$f(1) = 1^2 = 1,$$

$$f(2) = 2^2 = 4,$$

$$f(3) = 3^2 = 9,$$

$$f(4) = 4^2 = 16,$$

$$f(5) = 5^2 = 25$$

The argmax for the $f(x)$ is 5 (and not 25 which is the largest value of the function) because that results in the largest value from the target function.

So argmax $p(c|d)$ is out of all classes, the class with highest probability given the document.

Substituting, $\frac{p(d|c).p(c)}{p(d)}$ in place of $p(c|d)$, we get

$$c_{MAP} = \operatorname{argmax}_c \frac{p(d|c).p(c)}{p(d)} \text{ for } c \in C.$$

We can drop the denominator in the equation which is the probability of document. The reason why it is ok to drop the denominator is because if you are computing the right side of the equation for say 5 classes; for each class the probability of d will remain the same. So, if were to compare the output of these 5 things, each of which is divided by probability of the document, that number is same and we can cross it out.

$c_{MAP} = \operatorname{argmax}_c p(d|c).p(c)$ for $c \in C$. Thus, c_{MAP} is the class that maximizes the product of the two probabilities— probability of document given class (called the likelihood) and the probability of class (called the prior probability).

The document in our example is the Amazon review and the class might be ‘fake’ and ‘genuine’. So how do we compute the the two probabilities? Probability of class is easy to compute, following the frequentist approach, we can simply count the number of fake and genuine reviews in the corpus (collection of reviews) to generate the probability. What exactly do we mean by probability of document given class and how do we calculate it?

It’s easier to understand if we say that we are representing the document by a set of features (words present in the document) x_1 through x_n . Therefore, the probability of a document given a class is the probability of a vector of features given the class. We can represent that as follows:

$$c_{MAP} = \operatorname{argmax}_c p(x_1, x_2, \dots, x_n | c).p(c)$$

Calculating $p(x_1, x_2, \dots, x_n | c)$ for each class will be computationally very resource intensive so we make the simplifying assumptions (the Naive assumptions) that the feature probabilities are $p(x_i | c_j)$ are independent (which is obviously not true) and the order/position has no impact on outcome. Making these simplifying assumptions can make our problem much simpler to tackle yet results in a high degree of accuracy. As a result we get:

$$p(x_1, x_2, \dots, x_n | c) = p(x_1 | c).p(x_2 | c).p(x_3 | c) \dots p(x_n | c)$$

So the result of the simplifying assumptions is we’re going to represent the joint probability of a whole set of features x_1 through x_n conditioned on a class as the product of all of independent probabilities of x_1 given the class, probability of x_2 given the class and so on, and then multiply them all together. Thus, the Naive Bayes classifier is:

$$c_{NB} = \operatorname{argmax}_c p(c_j) \prod p(x_i | c_j)$$

For Naive Bayes calculations, we take the log to simplify calculations and increase speed.

$$c_{NB} = \operatorname{argmax}_c \log p(c) + \sum \log p(x_i | c)$$

The practical implementation of Naive Bayes classifier has a few subtleties that we are not going in here, for details see Jurafsky and Martin Chapter 4 (posted in Canvas). Process can be summarized in the figure 4.2 of the chapter (see below)

```

function TRAIN NAIVE BAYES(D,C) returns  $\log P(c)$  and  $\log P(w|c)$ 

for each class  $c \in C$            # Calculate  $P(c)$  terms
     $N_{doc}$  = number of documents in D
     $N_c$  = number of documents from D in class c
     $\logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$ 
     $V \leftarrow$  vocabulary of D
     $bigdoc[c] \leftarrow$  append(d) for  $d \in D$  with class c
    for each word  $w$  in  $V$            # Calculate  $P(w|c)$  terms
         $count(w,c) \leftarrow$  # of occurrences of  $w$  in  $bigdoc[c]$ 
         $\loglikelihood[w,c] \leftarrow \log \frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$ 
return  $\logprior, \loglikelihood, V$ 

function TEST NAIVE BAYES( $testdoc, \logprior, \loglikelihood, C, V$ ) returns best  $c$ 

for each class  $c \in C$ 
     $sum[c] \leftarrow \logprior[c]$ 
    for each position  $i$  in  $testdoc$ 
         $word \leftarrow testdoc[i]$ 
        if  $word \in V$ 
             $sum[c] \leftarrow sum[c] + \loglikelihood[word,c]$ 
return  $\operatorname{argmax}_c sum[c]$ 

```

Figure 4.2 The naive Bayes algorithm, using add-1 smoothing. To use add- α smoothing instead, change the +1 to + α for loglikelihood counts in training.

Evaluating Model Fit Classification analysis is different from LSA and LDA, because in topic models and LSA we infer natural groupings without knowing the real state of affairs, however, in supervised learning models like Naive Bayes, we know the true classification of the document. So how do we assess the performance of the models?

First, we split the documents into two groups (at least): the training set and the testing set (often a third validation set is also included). In a true project, we would want to use a three-way split of training, validation, and test. We train our model on the training set, and assess the testing set where our samples are ‘held out’. Generally, 70% of the data is put in training set and 30% in the testing set.

Confusion Matrices/ Contingency Tables

Based on the actual and predicted classes, we can create a contingency table or confusion matrix. Let's create a hypothetical example.

		Actual		
Predicted		Yes	No	Total Predicted
	Yes	11	1	12
	No	4	4	8
	Total Actual	15	5	20

We can use the contingency table to compute measures of goodness of fit. Some of the measures are (Jurafsky's book chapter on Naive Bayes):

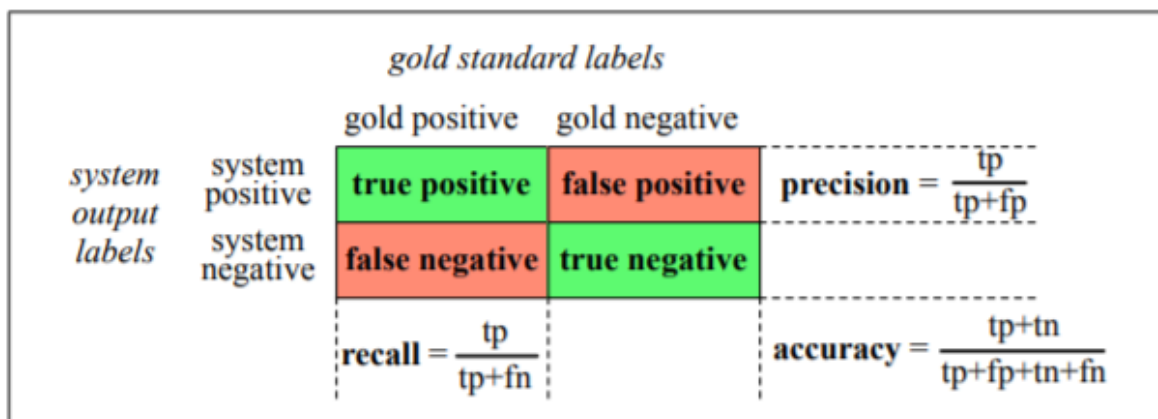


Figure 4.4 A confusion matrix for visualizing how well a binary classification system performs against gold standard labels.

Overall Measures

- 1) Accuracy = $\frac{\text{number of correct predictions}}{\text{total number of predictions}} \times 100$ Accuracy = $(11 + 4)/20 = 75\%$
- 2) Error rate = $\frac{\text{number of incorrect predictions}}{\text{total number of predictions}} \times 100 = 100 - \text{Accuracy}$ Error rate = $100 - 75 = 25\%$

Class Specific Measures

Accuracy and error rate tell us the overall of fit of the model. However, there are class-specific measures such as precision, recall and F measure. We need class-specific measures when it is necessary to identify a specific class. For instance, if we need to find a class of customers who are likely to stay with the company for long term, we don't care how good the model is in predicting short term customers.

- 3) Precision is the class-specific equivalent of accuracy. The only difference is that we measure how many of the predictions for a given class are accurate vis-a-vis the total number of documents that are predicted in the class.

$$\text{Precision}_i = \frac{\text{number of correct predicted}}{\text{total number predicted}}$$

$$\text{Precision} = 11/12 = 0.9167$$

- 4) Recall (for a given class) = $\frac{\text{number of correct predictions}}{\text{total number of actual}}$

$$\text{Recall} = 11/15 = 0.7333$$

- 5) F-Measure. In real life, we want to balance both precision and recall. The F-measure is a goodness of fit assessment for a classification that balances precision and recall (harmonic mean of Precision and Recall).

$$F = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = 2 \times \frac{0.9167 \times 0.7333}{0.9167 + 0.7333} = 0.815$$

The maximum possible value of the F-measure is 1.

Implementing Naive Bayes in R

Let's first load the libraries tidyverse, quanteda, quanteda.textmodels and caret that we will need for implementing Naive Bayes in R. We will work on the previously used hotel reviews dataset.

```
require(quanteda)
require(quanteda.textmodels)
require(caret)
require(tidyverse)

hotel_raw <- read_csv("Data/hotel-reviews.csv")
hotel_raw$Is_Response <- as.factor(hotel_raw$Is_Response) # tells whether happy or not happy
hotel_raw$Device_Used <- as.factor(hotel_raw$Device_Used)
hotel_raw$Browser_Used <- as.factor(hotel_raw$Browser_Used)
hotel_raw$user_ID <- as.factor(hotel_raw$user_ID)
summary(hotel_raw)
```

```
##      User_ID      Description      Browser_Used      Device_Used
## id10326:      1      Length:38932      Firefox      :7367      Desktop:15026
## id10327:      1      Class :character      Edge      :7134      Mobile :14976
## id10328:      1      Mode  :character      Google Chrome :4659      Tablet : 8930
## id10329:      1                                     InternetExplorer:4588
## id10330:      1                                     Mozilla Firefox :4328
## id10331:      1                                     Mozilla      :3092
## (Other):38926                                     (Other)      :7764
##      Is_Response
## happy      :26521
## not happy:12411
##
##
##
##
##
```

The output suggests we have class imbalance problem (marked difference between proportion of happy and not happy people). Since our data has non-trivial class imbalance, we'll use the caret package to create a random train/test split that ensures the correct happy/not happy class label proportions.

We set the random seed for reproducibility and also verify the proportions.

For a good article on class imbalance read: <https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/>

```
set.seed(12345)

hotel_record <- createDataPartition(hotel_raw$Is_Response, times = 1,
                                     p = 0.7, list = FALSE)

hotel_train <- hotel_raw[hotel_record,] # using hotel_record for indexing
hotel_test  <- hotel_raw[-hotel_record,]

# Verify proportions.
hotel_train %>%
count(Is_Response)%>%
  mutate(freq=n/sum(n))

## # A tibble: 2 x 3
##   Is_Response     n freq
##   <fct>         <int> <dbl>
## 1 happy         18565 0.681
## 2 not happy     8688 0.319

hotel_test %>%
count(Is_Response)%>%
  mutate(freq=n/sum(n))

## # A tibble: 2 x 3
##   Is_Response     n freq
##   <fct>         <int> <dbl>
## 1 happy         7956 0.681
## 2 not happy     3723 0.319
```

The output suggests that we have approximately the desired proportion in both training and test sets. Before implementing the Naive Bayes model, we need to preprocess the data and convert it to Document Term Matrix (or Document Feature Matrix in Quanteda parlance). We tokenize, stem and remove the stopwords from the text for both the training and test datasets and convert them to DFM.

Having created DTM for both train and test datasets, we are ready to implement the Naive Bayes model. When implementing the Naive Bayes model, we need to specify the vector of training labels associated with each document identified in training set (in our case, it is the `Is_Response` variable which contains the judgment whether consumer is happy or not). Next we train the naive Bayes classifier using `textmodel_nb()` function from library `quanteda.textmodels`. Naive Bayes can only take features into consideration that occur both in the training set and the test set, but we can make the features identical by passing `training_dfm` to

dfm_match() as a pattern.

Running the Naive Bayes Model

```
df_training <- tokens(hotel_train$Description) %>%
  dfm() %>%
  dfm_remove(stopwords("english"))%>%
  dfm_wordstem ()

df_test <- tokens(hotel_test$Description) %>%
  dfm() %>%
  dfm_remove(stopwords("english"))%>%
  dfm_wordstem ()

train_nb <- textmodel_nb(df_training, hotel_train$Is_Response)
summary(train_nb)

##
## Call:
## textmodel_nb.dfm(x = df_training, y = hotel_train$Is_Response)
##
## Class Priors:
## (showing first 2 elements)
##      happy not happy
##      0.5      0.5
##
## Estimated Feature Scores:
##           stay      crown      plaza      april      --      -      ,
## happy      0.011790 8.541e-05 0.0001960 8.227e-05 0.004883 0.01016 0.05349
## not happy 0.008883 9.736e-05 0.0001575 6.332e-05 0.005867 0.01050 0.04726
##           ----      .      staff      friend      attent      elev      tini
## happy      0.0007698 0.08999 0.006604 0.003732 0.0004087 0.0007336 0.0001902
## not happy 0.0007187 0.08444 0.003551 0.001559 0.0002240 0.0010844 0.0004686
##           (      '      )      food      restaur      delici      price
## happy      0.006839 0.0008525 0.007118 0.0016433 0.003135 2.887e-04 0.001924
## not happy 0.005718 0.0011279 0.005914 0.0009546 0.001447 3.166e-05 0.001733
##           littl      high      side      cours      washington      dc      pool
## happy      0.001856 0.0012524 0.0006733 0.0002557 1.703e-04 0.0002997 0.001496
## not happy 0.001262 0.0006197 0.0007345 0.0002240 8.786e-05 0.0001393 0.001095
##           children      room
## happy      0.0002327 0.01731
## not happy 0.0002240 0.01909
```

*#Naive Bayes can only take features into consideration
#that occur both in the training set and the test set,*

```

#but we can make the features identical using dfm_match()
df_matched <- dfm_match(df_test, features = featnames(df_training))

# Inspecting how well the classification worked
actual_class <- hotel_test$Is_Response
predicted_class <- predict(train_nb, newdata = df_matched)
tab_class <- table(actual_class, predicted_class)
tab_class

```

```

##           predicted_class
## actual_class happy not happy
##   happy       7047       909
##   not happy   1344       2379

```

From the contingency table, we can see that the number of false positives and false negatives are somewhat dissimilar, but does not seem to over- or underestimate one class too much. This suggests that the model prediction is not outstanding. We can use the function `confusionMatrix()` from the `caret` package to formally check how the classifier performed.

```

confusionMatrix(tab_class, mode = "everything")

```

```

## Confusion Matrix and Statistics
##
##           predicted_class
## actual_class happy not happy
##   happy       7047       909
##   not happy   1344       2379
##
##               Accuracy : 0.8071
##               95% CI : (0.7998, 0.8142)
##   No Information Rate : 0.7185
##   P-Value [Acc > NIR] : < 2.2e-16
##
##               Kappa : 0.5416
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Sensitivity : 0.8398
##               Specificity : 0.7235
##   Pos Pred Value : 0.8857
##   Neg Pred Value : 0.6390
##               Precision : 0.8857
##               Recall : 0.8398
##               F1 : 0.8622
##               Prevalence : 0.7185

```

```
##          Detection Rate : 0.6034
##    Detection Prevalence : 0.6812
##          Balanced Accuracy : 0.7817
##
##          'Positive' Class : happy
##
```

As expected, the numbers suggest that the model can be improved. You can play around with the parameters 'smooth', 'prior' and 'distribution' for improving the model.