Naive Bayes Classifier with R

B.M Njuguna

2023-02-14

Contents

Naive Bayes

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying the Bayes theorem. Naive Bayes classifiers are a collection of classification algorithm and not one algorithm. Based on this theorem, the Naive Bayes Classifier give the conditional probability of event A given event B. Generally, the following steps are followed;

1. Convert the given data set into frequency tables
2. Generate Likelihood table by finding the probabilities of given features
3. Now use the Bayes theorem ti calculate the posterior probability.

Suppose we have a problem where we want to decide whether to play or not on a particular day, given some weather conditions, the naive bayes can be applied as follows. First lets create the sample data set;

```
data <- data.frame(
  Outlook = c(
    "Rainy",
    "Sunny",
    "Overcast",
    "Overcast",
    "Sunny",
    "Rainy",
    "Sunny",
    "Overcast",
    "Rainy",
    "Sunny",
    "Sunny",
    "Rainy",
    "Overcast",
    "Overcast"
  ),
  Play = c(
    "Yes",
    "Yes",
    "Yes",
    "Yes",
    "No",
    "Yes",
    "Yes",
```

```
    "Yes",
    "No",
    "No",
    "Yes",
    "No",
    "Yes",
    "Yes"
  )
)
```

Then we create the frequency table;

```
table(data)
```

```
          Play
Outlook    No Yes
  Overcast  0   5
  Rainy     2   2
  Sunny     2   3
```

Then we calculate the likelihood of each occurrence
For the overcast = $5/14 = 0.35$
For the Rainy = $4/14 = 0.29$
and for the sunny = $(5/14 = 0.35)$
For all "NOs" = $4/14 = 0.29$ and for all "Yess" = $10/14 = 0.71$.
Thus the likelihood table will be as follows;

Table 1: Likelihood Table

Weather	Yes	No	
Overcast	5	0	$5/14 = 0.35$
Rainy	2	2	$4/14 = 0.29$
Sunny	3	2	$5/14 = 0.35$
All	$10/14 = 0.71$	$4/14 = 0.29$	

Column 2:
Note that 14 is the total observations. Now the Bayes Therem is applied using the following formula; (Supposing that we want the conditional probability that play was yes given that it was suny)

$$\mathbb{P}(Yes/Sunny) = \mathbb{P}(Sunny/Yes) \times \frac{\mathbb{P}(Yes)}{\mathbb{P}(Sunny)}$$

From here, you will just plug in values and then do the same for "No". For the results, choose the one with the greatest posterior probability. Note that, if the independent variables were more than one, you create the likelihood table for each independent variable.

Recall that;

$$\mathbb{P}(\mathcal{A}/\mathcal{B} = \frac{\mathbb{P}(A \cap \mathcal{B})}{\mathbb{P}(\mathcal{B}/\mathcal{A})}$$

The Bayes theorem used is;

$$\mathbb{P}(\mathcal{A}/\mathcal{B}) = \frac{\mathbb{P}(\mathcal{B}/\mathcal{A})\mathbb{P}(\mathcal{A})}{\mathbb{P}(\mathcal{B})}$$

Where;

- $\mathbb{P}(\mathcal{A}/\mathcal{B})$ is the posterior probability
- $\mathbb{P}(\mathcal{B}/\mathcal{A})\mathbb{P}(\mathcal{A})$ is the likelihood or the probability of event B occurring given that event A has already occurred
- $\mathbb{P}(\mathcal{B})$ is the predictor prior or the probability of event B occurring.

Naive Baye is applied in many fields such as;

1. Credit Scoring
2. Medical data classification
3. It can be used in real time classification
4. Used in text classification such as **Spam filtering** and **Sentimental Analysis**.

Assumptions of Naive Bayes

First, all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

Secondly, each feature is given the same weight(or importance). That is one independent variable cannot be able to accurately predict the outcome alone.

Types of Naive Bayes Models

1. **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.

2. **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.

3. **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Applying Mulitnomial Naive Bayes in Natural Language Processing

As mentioned ealier, Bayes theorem calculates probability P(c|x) where c is the class of the possible outcomes and x is the given instance which has to be classified, representing some certain features.

Naive Bayes are mostly used in natural language processing (NLP) problems. Naive Bayes predict the tag of a text. They calculate the probability of each tag for a given text and then output the tag with the highest one.

Consider the training data below;

Table 2: comments and the tag

Text	Reviews
"I liked the movie"	Positive
"It's a good movie. Nice story"	Positive
"Nice songs. But sadly boring ending."	negative
"Sad, boring movie"	negative
"Hero's acting is bad but heroine looks good. Overall nice movie"	positive

We wish to classify whether the text "Overall liked the movie" has a positive or a negative review. We have to calculate the probability that the review is positive given that the sentence is "overall liked the movie" and also the probability that the review is negative given that the text is "overall liked the movie".

Before doing this, we need to do two things;

1. Removing Stopwords

Stop word removal is one of the most commonly used preprocessing steps across different NLP applications. The idea is simply removing

the words that occur commonly across all the documents in the corpus. Typically, articles and pronouns are generally classified as stop words. The idea is simply removing the words that occur commonly across all the documents in the corpus. Typically, articles and pronouns are generally classified as stop words.

2. Stemming

Stemming is basically removing the suffix [1] from a word and reduce it to its root word [2] . For example: "Flying" is a word and its suffix is "ing", if we remove "ing" from "Flying" then we will get base word or root word which is "Fly".

[1] Some common examples of suffixes include -able, -al, er, est, ful and ible.

[2] A basic word to which affixes (prefixes and suffixes) are added is called a root word because it forms the basis of a new word

Lemmatization

Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma. For instance, stemming the word 'Caring' would return 'Car'. For instance, lemmatizing the word 'Caring' would return 'Care'. Stemming is used in case of large dataset where performance is an issue. After applying these two techniques, the data set becomes;

Text	Reviews
"Ilikedthemovie"	Positive
"Itsgood movieNicestory"	Positive
"Nicesongsboringend."	negative
"Sadboringmovie"	negative
"herosactingisbadbutheroinelooksgoodoverallnicemovi"	positive

Feature Engineering

The important part is to find the features from the data to make machine learning algorithms works. In this case, we have text. We need to convert this text into numbers that we can do calculations on. We use word frequencies. That is treating every document as a set of the words it contains. Our features will be the counts of each of these words. In our case, we have P(positive | overall liked the movie), by using this theorem:

P(positive | overall liked the movie) = P(overall liked the movie | positive) * P(positive) / P(overall liked the movie)

There's a problem though: "overall liked the movie" doesn't appear in our training dataset, so the probability is zero. Here, we assume the 'naive' condition that every word in a sentence is independent of the other ones. This means that now we look at individual words. We can write this as:

P(overall liked the movie) = P(overall) * P(liked) * P(the) * P(movie)

The next step is just applying the Bayes theorem;

P(overall liked the movie| positive) = P(overall | positive) * P(liked | positive) * P(the | positive) * P(movie | positive)

And now, these individual words actually show up several times in our training data, and we can calculate them!

First we calculate the prior probability of each tag. For a give sentence in our training data, the prior probability that the tag is positive is 3/5 and that for negative is 2/5.

Then, calculating P(overall | positive) means counting how many times the word "overall" appears in positive texts (1) divided by the total number of words in positive (17). Therefore, P(overall | positive) = 1/17, P(liked/positive) = 1/17, P(the/positive) = 2/17, P(movie/positive) = 3/17.

If probability comes out to be zero then By using Laplace smoothing: we add 1 to every count so it's never zero. To balance this, we add the number of possible words to the divisor, so the division will never be greater than 1. In our case, the total possible words count are 21. Applying smoothing, The results are:

word	P(word/positive)	P(word/negative)
overall	$1 + 1/17 + 21$	$0 + 1/7 + 21$
liked	$1 + 1/17 + 21$	$0 + 1/7 + 21$
the	$2 + 1/17 + 21$	$0 + 1/7 + 21$
movie	$3 + 1/17 + 21$	$1 + 1/7 + 21$

Now we just multiply all the probabilities, and see who is bigger:

P(overall | positive) * P(liked | positive) * P(the | positive) * P(movie | positive) * P(positive)

= 1.38 * 10^{-5} = 0.0000138

P(overall | negative) * P(liked | negative) * P(the | negative) * P(movie | negative) * P(negative)

= 0.13 * 10^{-5} = 0.0000013

In R, it is done as follows; We first import the data;

```
nvData <- read.csv(file.choose(new = TRUE), as.is = TRUE)
```

You can see that the text has so many words which might not be necessary or useful for our model, and hence we need to remove them through either Stemming or lemmatization and also remove the stop words, which we can call cleaning the corpus^ [A corpus is a large collection of texts selected in a systematic way and stored as an electronic database.]. Thus we first of create a corpus as follows;

```
## creting sms corpus(collection of docs)
library(tm)
sms_corpus<-Corpus(VectorSource(nvData$text))

library(tm) # text mining
# This one converts all to lower case
corpus_clean <- tm_map(sms_corpus, tolower)
 # Removes all the digits
corpus_clean <- tm_map(corpus_clean, removeNumbers)
# removes punctuation
corpus_clean<-tm_map(corpus_clean, removePunctuation)
# removes stopwords e.g: and, is...
corpus_clean<-tm_map(corpus_clean, removeWords, stopwords())
 # remove extra whitespaces
corpus_clean<-tm_map(corpus_clean, stripWhitespace)
```

Note that in NLP, text corpus is a collection of documents. Docs or Document in NLP may refer to a sentence, a group of sentences or even a phrase, depending upon the use case. For example here is a simple example of corpus with only 3 documents; D1, D2 and D3,

- D1 : NLP is super cool to learn and apply.
- D2 : I am currently learning NLP, you can too!
- D3 : CS224n at Stanford is the best NLP class you can ever take!

The plots on the side margin shows most frequent words in the spam and ham class. The words reduce size as the frequency reduces. It has been done using the package *wordcloud*

Then for spam is as shown below;

The next step will be to tokenize the corpus. **Tokenization** is used in Natural Language Processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning. The whole sentence is now split into meaningful words. But before that we first create the **Document-Term Matrix (DTM)** which is a matrix representation of the text corpus.

```
sms_dtm <- DocumentTermMatrix(corpus_clean)
kable(inspect(sms_dtm), format = "latex")
```

```
<<DocumentTermMatrix (documents: 5574, terms: 8302)>>
Non-/sparse entries: 44265/46231083
Sparsity           : 100%
Maximal term length: 40
Weighting          : term frequency (tf)
Sample             :
```

Figure 1: Words in the ham type

Figure 2: Words in the spam type

```
      Terms
Docs   call can dont free get just know ltgt now will
  1086    0   0    1    0   1    0    0    0   0    9
  1580    0   0    0    0   0    0    0   18   0    0
  1864    0   0    3    0   0    0    1    0   0    0
  2159    0   0    0    0   0    0    0    0   0    0
  2371    0   0    0    1   0    0    0    0   0    0
  2381    0   1    0    0   0    0    0    1   0    0
  2435    0   3    0    1   1    0    0    6   0    0
  2850    0   0    0    0   0    0    0    0   0    0
  3018    0   0    0    0   0    0    0    2   0    0
  5107    0   0    0    1   0    0    0    0   0    0
```

	call	can	dont	free	get	just	know	ltgt	now	will
1086	0	0	1	0	1	0	0	0	0	9
1580	0	0	0	0	0	0	0	18	0	0
1864	0	0	3	0	0	0	1	0	0	0
2159	0	0	0	0	0	0	0	0	0	0
2371	0	0	0	1	0	0	0	0	0	0
2381	0	1	0	0	0	0	0	1	0	0
2435	0	3	0	1	1	0	0	6	0	0
2850	0	0	0	0	0	0	0	0	0	0
3018	0	0	0	0	0	0	0	2	0	0
5107	0	0	0	1	0	0	0	0	0	0

So, in this data set, we have 5574 documents and 8302 terms. In other words. Each row represents a document/message while each column represents term/word.

The sparse matrix needs to be transformed into a data structure that can be used to train a naive Bayes classifier. Not all the terms/ words in the sparse matrix are useful for classification. In order to reduce the number of features we can proceed to consider the words that appears at least a certain number of times (frequent words) and identify the features (terms dictionary) [3].

[3] This is generally known as **Feature Engineering**

```
# First lets split our data set
sms_raw_train <- nvData[1:4180, ]
sms_raw_test <- nvData[4181:5574, ]

sms_dtm_train <- sms_dtm[1:4180, ]
sms_dtm_test <- sms_dtm[4181:5574, ]

sms_corpus_train <- corpus_clean[1:4180]
sms_corpus_test <- corpus_clean[4181:5574]
```

```
# Then we identify the most frequent words;
sms_features <- findFreqTerms(sms_dtm, 5)
summary(sms_features)
```

```
   Length     Class      Mode
     1554 character character
```

```
head(sms_features)
```

```
[1] "available" "bugis"     "cine"      "crazy"     "got"       "great"
```

There are 1557 terms that have been identified as frequent terms. Then the next step is that we want to limit our training and testing matrix to only the words in the dictionary of frequent terms.

```
sms_dtm_train <-
  DocumentTermMatrix(sms_corpus_train, list(dictionary  = sms_features))
sms_dtm_test <-
  DocumentTermMatrix(sms_corpus_test, list(dictionary = sms_features))
sms_dtm_train
```

```
<<DocumentTermMatrix (documents: 4180, terms: 1554)>>
Non-/sparse entries: 25728/6469992
Sparsity           : 100%
Maximal term length: 19
Weighting          : term frequency (tf)
```

You can see that the terms have reduced.

The naive Bayes classifier is typically trained on data with categorical features. This poses a problem since the cells in the sparse matrix indicate a count of the times a word appears in a message. We should change this to a factor variable that simply indicates yes or no depending on whether the word appears at all in a document.

```
convert_counts <- function(x) {
  x <- ifelse(x > 0, 1, 0)
  x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
  return (x)
}
# Don't run this twice, it will affect ypur results
sms_dtm_train <- apply(sms_dtm_train, MARGIN = 2, convert_counts)
sms_dtm_test <- apply(sms_dtm_test, MARGIN = 2, convert_counts)
kable(head(sms_dtm_train[, 1:10], format = "latex"))
```

available	bugis	cine	crazy	got	great	point	wat	world	joking
Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
No	No	No	No	No	No	No	No	No	Yes
No	No	No	No	No	No	No	No	No	No
No	No	No	No	No	No	No	No	No	No
No	No	No	No	No	No	No	No	No	No
No	No	No	No	No	No	No	No	No	No

The only remaining thing is now to train our model as follows;

```
library(e1071)
```

```
model <- naiveBayes(sms_dtm_train, sms_raw_train$type)
kableExtra::kable(model$tables["available"], caption =
                    "available", position = "left")
```

Table 6: available

	No	Yes
ham	0.9969571	0.0030429
spam	0.9964602	0.0035398

```
kable(model$tables["crazy"], caption =
                    "crazy", format = "latex")
```

Table 7: crazy

	No	Yes
ham	0.9977870	0.0022130
spam	0.9964602	0.0035398

For each term/word, available in the `sms_feature`, the probabilities are given as shown above. Such words are used to calculate the Bayesian probability of a word being ham or spam.

Then we can predict as follows;

```
library(generics)
predict <- predict(model, sms_dtm_test)
table(predict, sms_raw_test$type)
```

```
predict  ham spam
   ham  1204   22
   spam    8  160
```

Then we assess the model accuracy as follows;

```
library(gmodels)
CrossTable(predict, sms_raw_test$type,
           prop.chisq = F, prop.t = F,
           dnn=c("Predicted", "Actual"))
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  1394


             | Actual
   Predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1204 |        22 |      1226 |
             |     0.982 |     0.018 |     0.879 |
             |     0.993 |     0.121 |           |
-------------|-----------|-----------|-----------|
        spam |         8 |       160 |       168 |
             |     0.048 |     0.952 |     0.121 |
             |     0.007 |     0.879 |           |
-------------|-----------|-----------|-----------|
Column Total |      1212 |       182 |      1394 |
             |     0.869 |     0.131 |           |
-------------|-----------|-----------|-----------|
```