Bayesian Methods: Assignment 1

Carbonera Francesco
Cesaro Giada

23/10/2019

Contents

1 Description of the data

The **SMS Spam Collection**[1] is a set of 5,572 SMS in English. The file contains one message per line and each line is composed by two columns: `v1` contains the label (**ham** or **spam**) and `v2` contains the raw text. The aim is to use this dataset to build a prediction model that will accurately classify which texts are spam. For this purpose we will use the **Naive Bayes** approach, which will be implemented in two different ways (the *frequentist* and the *Bayesian* way). We will describe everything in **Section 3.**

2 Exploratory Data Analysis and preprocessing

Let's read in the data

```
mess<- read.csv("spam.csv")
```

and change the name of the two columns into `type`, which will contain the label **spam** or **ham**, and `text`, which will contain the message

```
mess<- mess%>% rename("type"= v1, "text"= v2)
```

The command `Corpus()` is used to work with textual data more easily. Internally, the command transforms the dataframe into a list that contains as many elements as the number of messages. In turn, each element of the `corpus` (our list) is a list of two elements. The command `inspect()` allows us to view the contents of the messages.

```
corpus <- Corpus(VectorSource(mess$text))
inspect(corpus[c(2,4,5)])
```

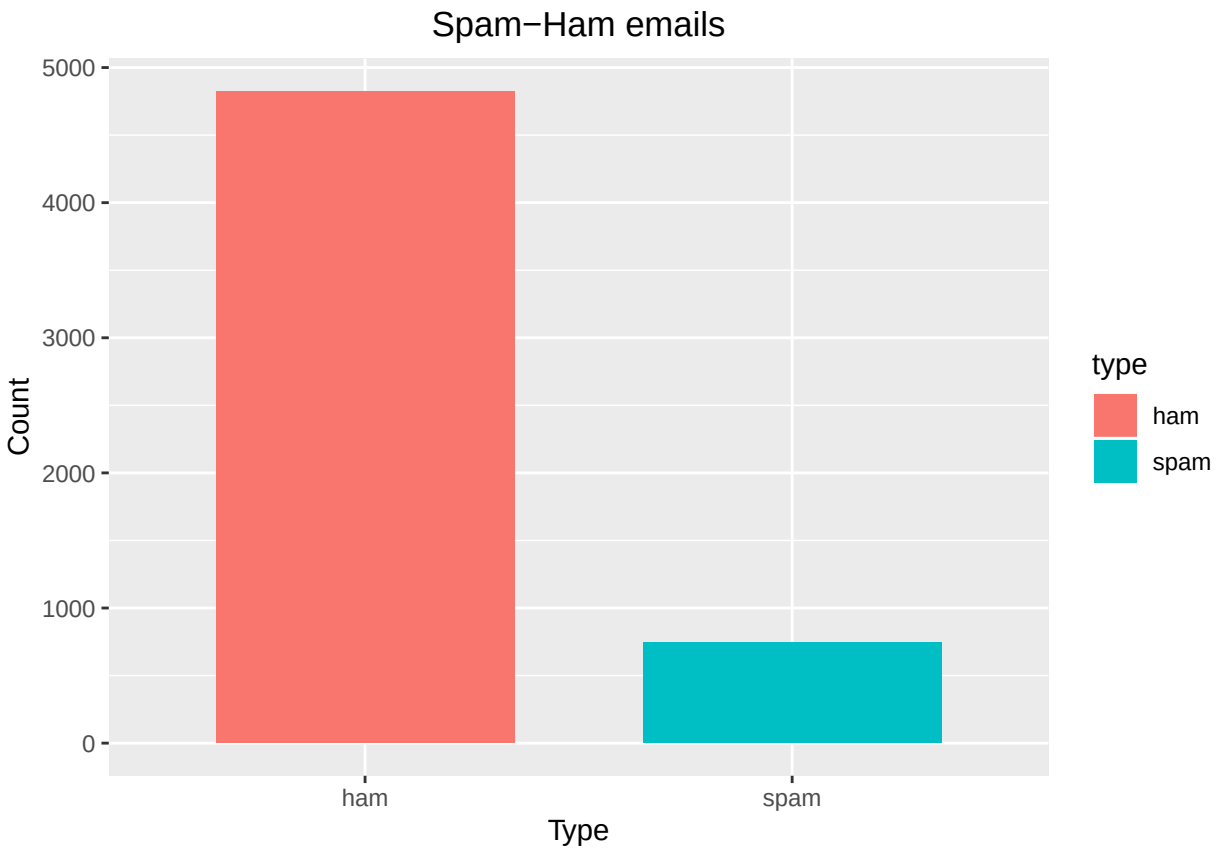
```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3

[1] Ok lar... Joking wif u oni...
[2] U dun say so early hor... U c already then say...
[3] Nah I don't think he goes to usf, he lives around here though
```

We are interested to know the number of spam/ham data in our dataset:

```
ggplot(mess, aes(x=factor(type)))+ geom_bar(stat="count", width=0.7, aes(fill=type))+
  labs(x= "Type", y= "Count")+ ggtitle("Spam-Ham emails") +
  theme(plot.title = element_text(hjust = 0.5))
```

[1] for more details about the dataset check https://www.kaggle.com/uciml/sms-spam-collection-dataset



The proportion of spam data is 0.155.

2.1 Preprocessing

Preprocessing is an important task and critical step in Text mining problems such as spam detection, which is used for extracting interesting and non-trivial information about the corpus of messages. The idea is to remove those text features which are assumed not to be relevant to be able to discriminate between a spam and non-spam message. This step depends very much on the type of text being analyzed and on the objectives of the analysis, for example removing punctuation marks may be useful in our case, but may not be adequate in sentiment analysis problems, where the use of many exclamation points, for instance, could help associate that message with "anger". It is important, therefore, to understand what is implicitly assumed in every preprocessing step that is commonly done. In what follows we:

- remove uppercase;
- remove numbers;
- remove punctuation marks;
- remove white spaces in excess
- perform stemming.

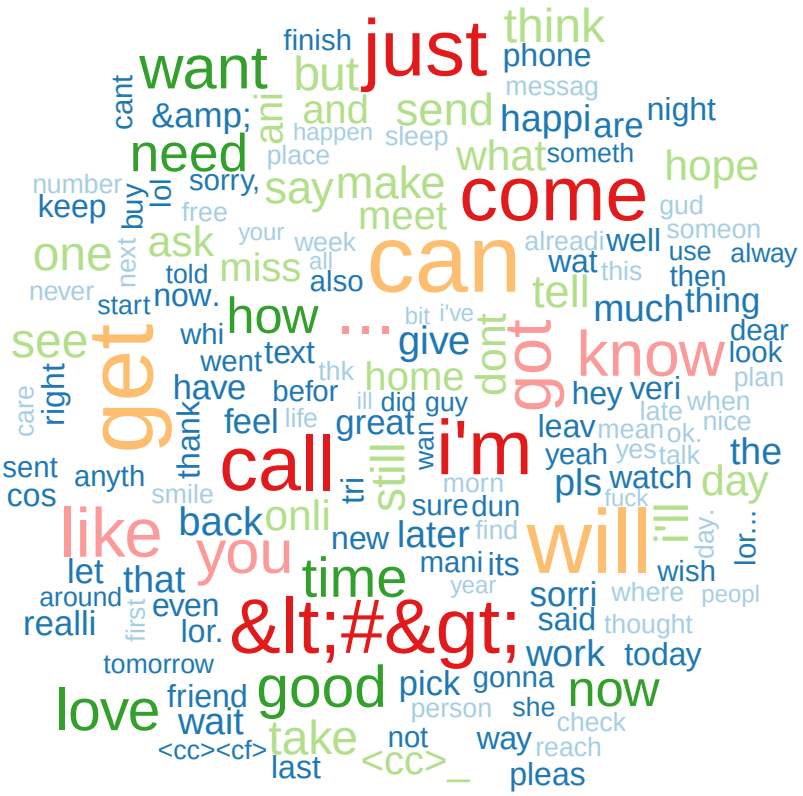
We are assuming that there is no reason to believe that spam messages may have more capital letters, more numbers (etc...) than non-spam messages. *Stemming* means that every word is reduced to its root, since it is assumed that the meaning is contained in the root of the word and not in the different derivative forms in which this word can appear. For example, there is no reason to believe that the word "drink" carries a different meaning than "drinks" or "drinking". Therefore the *stemming* allows to reduce the number of words in the corpus (it will be useful for the resulting data matrix, which we will discuss in **Section 3**) and to preserve only what really matters from a semantic point of view. Finally we remove stopwords, which are generally the most common words in a language (words linked to the grammar, articles, conjunction which do not carry any particular meaning .

```
clean_corpus <- tm_map(corpus, tolower)
clean_corpus <- tm_map(clean_corpus, removeNumbers)
clean_corpus <- tm_map(clean_corpus, removePunctuation)
clean_corpus <- tm_map(clean_corpus, stripWhitespace)
clean_corpus <-tm_map(corpus, stemDocument, language = "english")
clean_corpus <- tm_map(clean_corpus, removeWords, stopwords("en"))
```

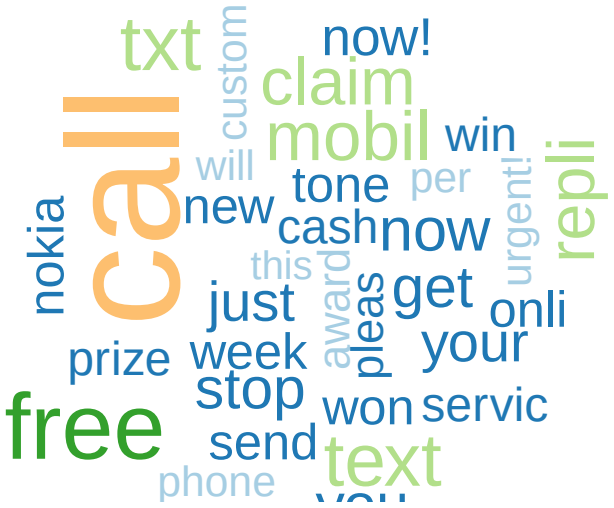
A **wordcloud** is used to summarize the messages in the corpus belonging to the two categories. After the preprocessing steps, the wordcloud allows you to view the most frequent words (in this example we make a word appear only if its absolute frequency in the group is at least 40). Size and color vary based on relative frequency, so you can get an immediate idea of what the most relevant words are

```
spam_indices <- which(mess$type == "spam")
ham_indices <- which(mess$type == "ham")

wordcloud(clean_corpus[ham_indices], min.freq=40, scale=c(3,.5), colors = brewer.pal(7, "Paired"))
```




```
# colors = brewer.pal(7, "Paired") per cambiare il colore al wordcloud: forse serve RcolorBrewer
wordcloud(clean_corpus[spam_indices], min.freq=40, scale=c(5,.8), colors = brewer.pal(7, "Paired"))
```



3 Building a Naive Bayes classifier

3.1 Turn our corpus into a data matrix

In order to use a machine learning algorithm in text mining problems it is necessary to use some technique to have a numerical representation of the whole corpus. These techniques are functions which map words onto vectors of real numbers. The **bag of words** approach is one of the simplest methods of embedding words into numerical vectors. It is not often used in practice due to its oversimplification of language

Using the `DocumentTermMatrix()` command the bag of words approach is implemented: it makes a sparse matrix structure where the rows of the matrix refer to the document and the columns refer to each single word which appear in the corpus. The integer number in position (i, j) of the matrix indicates how many times the i-th word appeared in message j. **Section 4** will illustrate the importance of the matrix representation of the corpus in order to go beyond some basic assumptions of the Naive Bayes method.

3.2 The *frequentist* approach

In what follows, we will build a binary classifier called **Naive Bayes**, and we will initially see its implementation in a *frequentist* perspective. The Naive Bayes classifier assumes that the words in each message are *independent "successes"* conditioned on the type of message. The idea is therefore to computee the probability that a message is spam /not spam in the following way:

- first, the probabilities (relative frequencies) of each word occurring in a spam/not spam message are estimated;
- second, the product of those estimated probabilities for each word in a given message is computed. In this way we obtain the probability to observe that particular message, given that it is spam/ham respectively;
- third, the relative frequencies of spam/ham messages in the whole corpus is computed. Those relative frequencies will be the estimation of the probability with which spam/ham messages appear;
- finally the Bayes theorem is applied to obtain the desired probability, that is, given a message, the probability of it being spam/ham.

If we did not use the above mentiones simplification (words in a message are *independent "successes"*) a message whose exact content had not been seen yet (neither among spam nor non-spam messages) would occur with probability 0, even if all the words that compose it had been individually observed.

We now implement a classifier that works as we have described. We will use a training set consisting of 75% of the messages (chosen at random) and test the classifier on the 25% left aside.

```
nobs=dim(mess)[1]
indices<- sample(1:nobs, size=round(nobs*0.75), replace=FALSE)


mess_train <- mess[indices,]
mess_test <- mess[-indices,]

corpus_train <- clean_corpus[indices]
corpus_test <- clean_corpus[-indices]

mess_dtm <- DocumentTermMatrix(clean_corpus)

mess_dtm_train <- mess_dtm[indices,]
mess_dtm_test <- mess_dtm[-indices,]
```

For the way it works internally, the classifier only needs to know whether the word is present or not in the document, and not counts. Therefore we transform the numbers that indicate the presence or absence of the word in "yes"" or "no".

```
convert_count <- function(x){
y <- ifelse(x > 0, 1,0)
y <- factor(y, levels=c(0,1), labels=c("No", "Yes"))
y
}
```

We use the function we have just defined to apply it to the columns of the train corpus and to those of the test.

```
mess_dtm_train <- apply(mess_dtm_train, 2, convert_count)
mess_dtm_test <- apply(mess_dtm_test, 2, convert_count)
```

Finally we build the classifier we described using the library `e1071` and the command `naiveBayes()`. With the instruction `predict` we can use our model, `classifier`, to classify the messages in the test set.

```
classifier <- naiveBayes(mess_dtm_train, mess_train$type)
class(classifier)
```

```
[1] "naiveBayes"
```

```
predicciones <- predict(classifier, newdata=mess_dtm_train)
table(predicciones, mess_train$type)
```

```
predicciones  ham spam
        ham  3575   43
        spam   49  512
```

A problem inherent in the frequentist approach is, however, the fact that if a word only appears in one of the two groups, any message containing that word would automatically be assigned to the group in which the word appears at least once. This happens because, trivially, in the numerator we would have a product of probabilities where one is 0:

$$Pr(spam|message_i) = \frac{Pr(message_i|spam) \cdot Pr(spam)}{Pr(message_i)}$$

and

$$Pr(message_i|spam) = Pr(word1_i|spam) \cdot Pr(word2_i|spam) \cdot ... \cdot Pr(wordJ_i|spam) \cdot ... \cdot Pr(wordN_i|spam)$$

where N is the number of words in the message. Assuming that $wordJ_i$ never appears among the spam messages, the probability above will then be

$$Pr(message_i|spam) = Pr(word1_i|spam) \cdot Pr(word2_i|spam) \cdot ... \cdot 0 \cdot ... \cdot Pr(wordJ_i|spam) \cdot ... \cdot Pr(wordN_i|spam) = 0$$

The message will be classified as ham, despite the fact that the other words ($wordH_i,\ H \in \{1, ..., J-1, J+1, ...N\}$) may all have higher probabilities to be spam rather than ham messages.
The following approach to the Naive Bayes method will help solve this issue.

3.3 The *Bayesian* approach

In the Bayesian approach unknown parameters are treated as random variables. In our case the parameters we are interested in are the probabilities of occurrence of each word in the spam or non spam group, and the probabilities that a message belongs to one or the other category. We will indicate these last two parameters of interest as $\theta_s = Pr(spam)$ and $\theta_h = Pr(ham)(= 1 - \theta_s)$, while the probability of occurrence of a word `m` in a spam (ham) message will be written as θ_{ms} (θ_{mh}). Each parameter has its own prior distribution: we will consider a uniform distribution for all of them.
How does this framework solve the problem of the frequentist case, when a word did not appear in one of the two groups? If we consider words as independent successes, the likelihood would then be proportional to $\theta_{ms}^S \cdot (1 - \theta_{ms}^{n-S})$ where S is the number of spam messages where the word `m` has appeared[2] and `n` the number of messages observed. Since the uniform distribution in [0,1] is a $Beta(1, 1))$, we know that if the a-priori is a $Beta(\alpha, \beta)$ and the experiment is binomial, the a-posteriori is proportional to $\theta_{ms}^{\alpha+S} \cdot (1 - \theta_{ms}^{\beta+(n-S)})$, where $n - S$ is the number of $failures$[3]. This means that α and β in the a-priori play the role of successes and failures ($\alpha \approx S$ and $\beta \approx failures$), therefore choosing a $Beta(1, 1)$ is like saying that we expect to see the word `m` once out of two spam messages. This is the key to solving the problem: apparently it seems that we are adding irrelevant (neutral) information by using these a-priori, yet it is as if we were saying that we actually observed every single word once in each group and once not (if we adopt a $Beta(1, 1)$ as an a-priori for each $\theta_{mk},\ k \in \{s, h\}$). Then we can calculate Bayesian point estimates using the a-posteriori means, which proves to be a weighted average of the a-priori mean and of the maximum likelihood estimator. The information given by the a-priori tends to decrease in importance as the size of the sample increases. For example, if I am interested in θ_{ms} and I use a uniform a-priori, (which is equivalent to saying that in two spam messages one had the word `m` and one did not), then if I see only messages where the word `m` does not appears then the probability of "success" (in this case θ_{ms}) will become smaller and smaller but never 0: that is because, even if the maximum likelihood estimate for the parameter would be 0, there is always a weight ($w = \frac{\alpha+\beta}{\alpha+\beta+n}$, decreasing with the number of messages `n`) that is given to the a-priori mean, which is 0.5 ($\frac{\alpha}{\alpha+\beta}$).

The classifier with uniform a priori distributions is equivalent to the so-called "Laplace smoothing". In R it can be included adding the parameter `laplace=1` when building the Naive Bayes classifier:

```
B.clas <- naiveBayes(mess_dtm_train, mess_train$type,laplace = 1)
class(B.clas)
```

```
[1] "naiveBayes"
```

```
B.preds <- predict(B.clas, newdata=mess_dtm_test)
a<- table(B.preds, mess_test$type)
knitr::kable(a)
```

	ham	spam
ham	1199	62
spam	2	130

The overall accuracy of the Bayesian classifier is 0.954056.

4 Alternative solutions and possible improvements

The simplification made by the Naive Bayes approaches, however, is based on the assumption that the probability of two words occuring together is simply the product of the probabilities of the two words occurring separately. It does not take into account the fact that there could be groups of words which are more likely to appear together. For

[2]Here "success" is seeing the word `m` in a spam message
[3]A failure is when the word `m` does not appear in a spam message

example, two words taken individually could belong to the spam or non spam group with the same probability, but the two together could appear much more likely in a spam message. A more sophisticated way of approaching the problem would be to consider groups of words, the so-called *ngrams*. There are in fact more ways of vectorizing text, each one corresponding to different approaches to the problem. Mapping every word to a vector is the solution which we have used so far, but using groups of 2/3/4 words and estimating probabilities such as $\theta_{w1,w2,w3;\ s}$ (if we choose groups of 3 words) could help improving the classification rate.

Another possible way of tackling the classification problem is by using any other machine learning algorithm that can work with a matrix such as the one which results from the way we represent our corpus (which has, if we use the Naive Bayes approach, as many columns[4] as the number of different words in the corpus, and as many rows as messages). These algorithms are not based on the assumption of the Naive Bayes: even if the data matrix is constructed by taking each word individually and counting the number of occurrances in every message, they allow, by their nature, to find relationships among the attributes[5] without the need to use ngrams (as we should do instead in the case of Naive Bayes to make it possible to consider relations between the variables). For example, SVMs, Random Forest etc...

Moreover, another possible improvement could be to tune the thresold t such that the probability to classify a message as spam if it is not is the lowest possible. It would be interesting to tune t in such a way that the message is labelled as spam only if $\hat{\theta}_{message,\ spam}|ham > t$ with t large enough to ensure that the lower possible number of ham emails get wrongly classified as being spam, without affecting too much the classification error rate $\frac{\#\ of\ wrongly\ classified\ messages}{\#\ of\ messages\ in\ the\ corpus}$. This error measure gives equal weight to the two types of error, while we could think instead that moving a message to the spam folder when it is not is more harmful than making a spam message appear in the ham folder. An idea would be therefore to find a suitable value of t (higher than the usual 0.5) to keep the error that we are interested in minimizing low, while keeping the total error within a given threshold

Finally, a possible improvement in the approach described so far would be to assign a more "informative" a-priori distributions for each word.

[4] also: features
[5] here: words