Introducing Text Mining

September 1, 2017

Now we move on to learn how to use R for classification and clustering, related to *text mining* as well — matter less how we define the concepts, but the two former ones is usually taken to mean something like the grouping of similar things, while separating unsimilar things from each other.

In many cases we need to handle increasingly higher numbers of variables, which in some cases are termed *dimensions*. For certain analytical tasks we need to consider n variables (e.g. word forms occurring in a text collection), so we then say that we have an *n-dimensional space* of variables.

A typical example is where we have a large number of documents in which the contents can be considered to treat different topics. In libraries this task has been performed by human analyzers applying a classfication scheme or a controlled vocabulary of some other kind, such as the *Library of Congress Subject Headings* scheme. Exactly what variables are considered by humans for each document to be classified are largely unknown. Having machines to perform this task is another option, but in that case we need to elaborate on what variables to consider in classification, and we often term these variables *features*.

A multivariate analysis of a document collection may have a predictive purpose to classify each document in one of n number of predefined classes (classification), or simply a purpose to assign each document to one of n number of classes (or clusters) according to some similarity measure.

One of the most crucial issues is to decide upon the features (the variables) to be considered in such tasks. The most crude approach to group n text

documents in *x* number of groups is to treat each occurring string (defined by preceding and succeeding whitespace characters as delimiters) in the whole collection as a feature, resulting in a very high dimensionality (number of variables) for each text document. The preceeding part of this workshop gave you the experience of splitting texts into term-document matrices, a starting point for many text mining tasks.

This workshop aims to prepare you for the following learning objectives, that relates to the learning outcomes of the courses.

- understand the concepts of tokenization, term frequency and feature weighting and what roles they play in the treatment of texts for different analytical purposes,
- understand the underlying principles for categorization of individual instances in a large data set in a very basic way,
- be able to preprocess textual data with the help of R STUDIO, and
- being prepared to find and use some model for categorization.

As we have done in previous parts of this workshop, we lean more or less on tutorials freely available on the web, with adaptations and recommendations for our courses. Partly we will use the data produced in the preceding part of this workshop.

Classification

Classification, or *supervised machine learning*, leans on the idea that we have a set of instances for which we know the correct class and we let a machine learning algorithm generalize or abstract from this data to be able to classify new unseen instances as accurate as possible.

There are many models for classification, and common for most of them is the fact that they need to be trained and improved by choosing or compile good training data, have the set of features of the data adjusted and some parameters set.

As a starter, follow the instructions given by Karlijn Willems in *Machine Learning in R for beginners*, available on
`https://www.datacamp.com/community/tutorials/machine-learning-in-r`
until you reach the section *Machine Learning in R with caret* where you can stop. These instructions apply to the *k-Nearest Neighbor* algorithm, an algorithm that is quite transparent compared to e.g. *Neural Networks*. The underlying principles of the *k-Nearest Neighbor* algorithm therefore becomes quite possible to understand. Consult the literature for this algorithm.

Note that in Willems' tutorial we have more variations on how to visualize tabular data. There is no harm in that. In fact, it is good to know different ways of doing the same thing, because looking for help on the web may require you to know the different ways of doing things. Choose what methods suit you best in the end.

As an exercise, to check if you understand the underlying variables for producing plots, translate the plot produced by `ggvis` (as given by Willems) to a plot produced by `ggplot2` as presented in Chapter 3 by Wickham and Grolemund (2017) – which you worked on in a previous part of this workshop.

Note that `%>%` is just a different way of writing things.

```
iris %>% ggvis(~Sepal.Length, ~Sepal.Width, fill = ~Species)
 %>% layer_points()
```

is equivalent to

```
layer_points(ggvis(iris, ~Sepal.Length, ~Sepal.Width, fill = ~Species))
```

The *Iris* data set is one of the most common data sets in the literature of machine learning, probably because it results in quite accurate outcomes. Real world data, especially text collections, provide more complex tasks.

Now, try to apply what you learned from Willems' introduction to your data from the previous part of the workshop (i.e. the preprocessed Project Gutenberg texts). Here are some advices.

Preparing for classification

Your exercise data consists of 20 texts downloaded from the psychology collection of Project Gutenberg and 20 texts from the history collection. We can use this fact as an indication of the correct topical classification for each text. If we use 0 for history and 1 for psychology, here is a vector of these classes.

```
classes <-
c(0,1,1,0,0,1,1,1,0,0,0,0,0,1,1,1,1,0,1,1,0,1,0,1,0,0,1,1,1,0,1,0,0,1,0,1,0,1,0,0)
```

For convenience you may (as you see) store this vector temporarily in a variable `classes` and then add it as a last column of the data. In doing this, there are two facts to observe. The document term matrix (which we refer to as DTM in short in the following) is a special object that needs to be transformed into a data frame and we need to ensure that the class indicators are not treated as numbers, but as categorical values. Let us do this in two steps for transparency. First conversion into a data frame

```
DTM <- as.data.frame(as.matrix(dtms))
```

and then the addition of column `classes` as factors.

```
DTM$class <- as.factor(classes)
```

Here you need to keep track of how many features you have, in order to follow Willems' instructions. (I have 285 features, so the classes will then be in column 286, but you may have a different number of features.)

SVM classification

Now, try the SVM model on the same data (**DTM** and/or **iris**). You access this by loading the library `e1071` (after installing it, if it is not present).

You'll need two functions, `svm` and `predict`. The first one trains the model on the basis of training data and the class labels pertaining to the training data, and the second one take the model as input and some data to test it

on. Explore this by means of help functions and searching on the web for particular problems you may encounter.

Now let us use try some *unsupervised learning*.

Grouping Texts — Clustering

Here, we proceed with the data we have processed in the *Text Processing* section. We will use what is stored in the (program) variable `DTM` and start by looking at the *k-means* algorithm for clustering. If you have lost the data in your R Studio session, just repeat what you did.

The algorithm is theoretically quite simple, though its details are most likely more tricky to understand for you. The Wikipedia article on the topic (which otherwise is quite technical) states "k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster".

In our case we have 11 observations (texts) with a large set of features (i.e. variables, the number depending on the preprocessing result) which we might want to analyse with respect to similarity. However, though we will be content with the data for these exercises, there are some knowledge missing, e.g.

- we need to define our notion of similarity which is dependent on what question(s) we would like to pose to the data and/or what we want to do,
- the amount of observations is most likely too low for doing any reliable quantitative analysis (i.e. clustering).

For instance, we could aim for a classification on the basis of *text type*, though we already know that three texts are non-fictional abstracts, two texts are longer non-fictional works and six are fictional works, and the purpose thereby would be to investigate if we could find features that are particularly

apt for that kind of classification. We could also aim for topical classification, having the algorithm group the texts according to their aboutness, with similar purposes as for text type. However, with a simple single-string (bag of words) approach, such tasks are not that promising. Just think about how the length of the text probably would be a better discriminator for identifying the abstracts (in the text type identification task) than word frequency.

Since we already have our data loaded, invoking k-means is quite simple. Remember not to include the column for class indicators. In my case only the features 1 to 245.

```
kmeansResult = kmeans(as.matrix(DTM[,1:285]),2)
```

and look at the result

```
kmeansResult
```

which will give something like

```
K-means clustering with 2 clusters of sizes 39, 1

Cluster means:
        access   accessible   accordance      account       actual
1 6.997802e-05 1.196191e-05 1.322028e-05 2.033834e-05 1.017062e-05
2 5.968222e-05 3.580933e-05 7.853354e-06 1.473005e-05 2.325798e-06

/ ... /

Clustering vector:
   arizona.txt    bergson.txt      binet.txt    BritMus.txt      bryce.txt
             1              1              1              1              1
    butler.txt    cameron.txt    carroll.txt    cushing.txt      cyrus.txt
             1              1              1              1              1
    dorsey.txt      egypt.txt      fowke.txt      freud.txt   freudSex.txt
             1              1              2              1              1
     gross.txt       hall.txt   hallhist.txt  havelock1.txt  havelock2.txt
             1              1              1              1              1
haverfield.txt      healy.txt    henshaw.txt     hilton.txt     holmes.txt
             1              1              1              1              1
   holmes2.txt    jackson.txt      james.txt       kent.txt   lanciani.txt
             1              1              1              1              1
     lebon.txt      mason.txt  mindeleff.txt      moisa.txt    nichols.txt
             1              1              1              1              1
```

```
    porter.txt     pueblo.txt       pyle.txt    ragozin.txt     yarrow.txt
             1              1              1              1              1

Within cluster sum of squares by cluster:
[1] 1.737619e-05 0.000000e+00
 (between_SS / total_SS =  31.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

We can see that `fowke.txt` gets another cluster than the others. All the other texts fall into the other cluster. Disappointing.

The last number in `kmeans(as.matrix(DTM[,1:285]),2)` indicates the number of clusters wanted. Try to increase this value and rerun the clustering. Remember that you only have 40 instances, so there would be no point in aiming for too many clusters.

When you have explored the Gutenberg data enough, you will try the same on the iris data and compare the results (see below). Most likely, the results will be better then, which depends on the task complexity and available data.

From this we may learn that the class decisions made by humans tend to increase the resulting accuracy (i.e. with supervised learning), but if such indications are unavailable (e.g. because of cost), we may lean on clustering.

Visualizing the data

Visualization might also be a way of analysing the outcome of the result. We first create an array of the text-cluster pairs from `kmeansResult`. This will carry information on the cluster assignments.

```
cluster = as.factor(kmeansResult$cluster)
```

Then we plot the result as coordinates for two chosen features, say `committed` and `compliance`.

```
library(ggplot2)

ggplot(as.data.frame(as.matrix(DTM)),
  aes(committed, compliance, color = cluster, label =
  rownames(as.data.frame(as.matrix(DTM))))) +
  geom_point() + geom_text(size=3,nudge_x = 0.001,
       nudge_y = 0.00005)
```

Check out that the "words" are in the document-term matrix, otherwise you will get an error. It might be best to transpose the document-term-matrix into a term-document-matrix by means of

```
View(t(DTM))
```

in order to browse the words easier in the View panel.

Note that you may also choose to use the class labels as color markers in stead of the cluster labels. Change `cluster` to `classes` in the command above.

This is taken from the ggplot2 library for creating different kinds of plot patterns. Ask your teacher what the different parts really do, if you are curious and don't want to investigate it on your own.

Now for some work on your own. Return to the iris dataset that consists of 150 iris instances that belong to one of three different species and have as features measures sepal length, sepal width, petal length, and petal width. Thus, not text. Have a reminding look at the data by

```
View(iris)
```

and by plotting the data on any two features:

```
ggplot(iris, aes(Sepal.Length, Sepal.Width,
      color = Species, label = rownames(iris)))
      + geom_point()
```

Try now, on your own, to cluster these in a similar manner as above. Observe that you need to cluster only on the four first features, since we do not want to feed the known class beforehand. In this way you will refer to only the wanted data, namely column 1 — 4.

```
iris[,1:4]
```