Text Classification with R

Wouter van Atteveldt

June 1, 2016

Machine Learning or automatic text classification is a set of techniques to train a statistical model on a set of annotated (coded) training texts, that can then be used to predict the category or class of new texts.

R has a number of different packages for various machine learning algorithm such as maximum entropy modeling, neural networks, and support vector machines. `RTextTools` provides an easy way to access a large number of these algorithms.

In principle, like 'classical' statistical models, machine learning uses a number of (independent) variables called *features* to predict a target (dependent) category or class. In text mining, the independent variables are generally the term frequencies, and as such the input for the machine learning is the document-term matrix.

RTextTools can be installed directly from CRAN:

```
install.packages("RTextTools")
```

Obtaining data

For this example, we will use Amazon reviews from http://jmcauley.ucsd.edu/data/amazon/ and classify whether they are positive or negative. Note that text classification can be used generally for nominal values, so you can also use it for classifying topic, frames, etc.

These reviews are stored in a gzipped while which contains one json record per line, so we use scan to split the file into lines, and then read each line with a custom functions that converts the line from json and into a data frame. We then use `ldply` to apply this function to each line:

```
url = "http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Automotive_5.json.gz"
download.file(url, destfile="reviews.json.gz")
lines = scan(gzfile("reviews.json.gz"), sep = "\n", what="")
readline = function(line) {
  x = rjson::fromJSON(line)
  x$helpful = NULL
  as.data.frame(x)
}
reviews = plyr::ldply(lines, readline)
saveRDS(reviews, "data/reviews.rds")
```

Since this takes a while to process, we save the reviews as an RDS file, which you can also download directly from here:

```
reviews = readRDS("data/reviews.rds")
```

Creating the Document Term Matrix

So, the first step is to create a document-term matrix. To make it run faster for testing, we take a limited data set here. Since reviews are mostly positive (taking positive to be 4 or 5 stars), we sample 500 positive and 500 negative reviews to use:

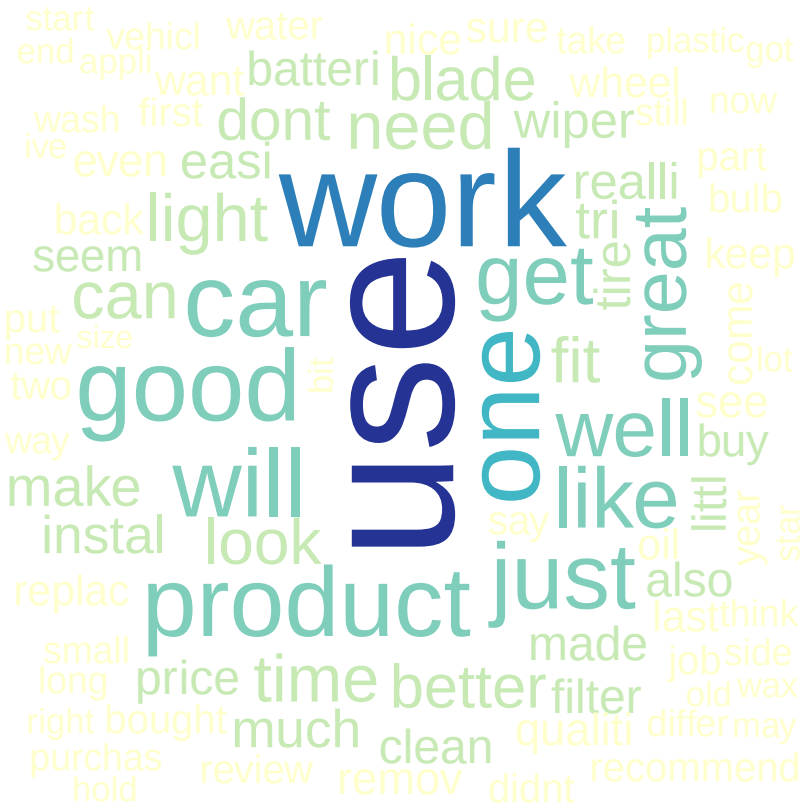
```
reviews$id = 1:nrow(reviews)
reviews$positive = as.numeric(reviews$overall >= 4)
pos = sample(reviews$id[reviews$positive == 1], 500)
neg = sample(reviews$id[reviews$positive == 0], 500)
reviews = reviews[reviews$id %in% c(pos, neg), ]
```

Now, we can create a dtm:

```
library(RTextTools)
dtm = create_matrix(reviews[c("summary", "reviewText")], language="english", stemWords=T)
```

Of course, now that we have a DTM we can plot a word cloud to get some feeling of the most frequent words:

```
library(corpustools)
dtm.wordcloud(dtm)
```



(we could also e.g. compare the words in positive and negative reviews, or run a topic model on only the positive or negative terms; see the handouts `comparing.pdf` and `lda.pdf`, respectively)

Preparing the training and testing data

The next step is to create the RTextTools *container*. This contains both the dt matrix and the manually coded classes, and you specify which parts to use for training and which for testing.

To make sure that we get a random sample of documents for training and testing, we sample 80% of the set for training and the remainder for testing. (Note that it is important to sort the indices as otherwise GLMNET will fail)

```
n = nrow(dtm)
train = sort(sample(1:n, n*.8))
test = sort(setdiff(1:n, train))
```

Now, we are ready to create the container:

```
c = create_container(dtm, reviews$positive, trainSize=train, testSize=test, virgin=F)
```

Using this container, we can train different models:

```
SVM <- train_model(c,"SVM")
MAXENT <- train_model(c,"MAXENT")
GLMNET <- train_model(c,"GLMNET")
```

Testing model performance

Using the same container, we can classify the 'test' dataset

```
SVM_CLASSIFY <- classify_model(c, SVM)
MAXENT_CLASSIFY <- classify_model(c, MAXENT)
GLMNET_CLASSIFY <- classify_model(c, GLMNET)
```

Let's have a look at what these classifications yield:

```
head(SVM_CLASSIFY)
```

SVM_LABEL	SVM_PROB
1	0.7803867
1	0.5692446
1	0.5265868
0	0.5813466
1	0.8148137
0	0.5490586

For each document in the test set, the predicted label and probability are given. We can compare these predictions to the correct classes manually:

```
t = table(SVM_CLASSIFY$SVM_LABEL, as.character(reviews$positive[test]))
t
```

```
##
##      0  1
##   0 73 16
##   1 37 74
```

(Note that the as.character cast is necessary to align the rows and columns) And compute the accuracy:

```
sum(diag(t)) / sum(t)
```

```
## [1] 0.735
```

Analytics

To make it easier to compute the relevant metrics, RTextTools has a built-in analytics function:

```
analytics <- create_analytics(c, cbind(SVM_CLASSIFY, GLMNET_CLASSIFY, MAXENT_CLASSIFY))
names(attributes(analytics))
```

```
## [1] "label_summary"     "document_summary"  "algorithm_summary"
## [4] "ensemble_summary"  "class"
```

The `algorithm_summary` gives the performance of the various algorithms, with precision, recall, and f-score given per algorithm:

```
head(analytics@algorithm_summary)
```

	SVM_PRECISION	SVM_RECALL	SVM_FSCORE	GLMNET_PRECISION	GLMNET_RECALL	GLMNE
0	0.82	0.66	0.73	0.73	0.61	
1	0.67	0.82	0.74	0.60	0.72	

The `label_summary` gives the performance per label (class):

```
head(analytics@label_summary)
```

	NUM_MANUALLY_CODED	NUM_CONSENSUS_CODED	NUM_PROBABILITY_CODED	PCT_CONSEN
0	110	92	108	
1	90	108	92	

Finally, the `ensemble_summary` gives an indication of how performance changes based on the amount of classifiers that agree on the classification:

```
head(analytics@ensemble_summary)
```

	n-ENSEMBLE COVERAGE	n-ENSEMBLE RECALL
n >= 1	1.00	0.73
n >= 2	1.00	0.73
n >= 3	0.66	0.80

The last attribute, `document_summary`, contains the classifications of the various algorithms per document, and also lists how many agree and whether the consensus and the highest probability classifier where correct:

```
head(analytics@document_summary)
```

SVM_LABEL	SVM_PROB	GLMNET_LABEL	GLMNET_PROB	MAXENTROPY_LABEL	MAXENTROPY_
1	0.7803867	1	0.6059908	1	1.
1	0.5692446	1	0.5936875	0	1.
1	0.5265868	0	0.7224154	0	0.
0	0.5813466	0	0.5351748	0	1.
1	0.8148137	1	0.9826154	1	1.
0	0.5490586	1	0.7689459	1	0.

Classifying new material

New material (called 'virgin data' in RTextTools) can be coded by placing the old and new material in a single container. Let's assume that we don't know the sentiment of 20% of our material:

```
reviews$positive[1:200] = NA
```

We now set all documents with a sentiment score as training material, and specify `virgin=T` to indicate that we don't have coded classes on the test material:

```
coded = which(!is.na(reviews$positive))
c = create_container(dtm, reviews$positive, trainSize=coded, virgin=T)
```

We can now build and test the model as before:

```
SVM <- train_model(c,"SVM")
MAXENT <- train_model(c,"MAXENT")
GLMNET <- train_model(c,"GLMNET")
SVM_CLASSIFY <- classify_model(c, SVM)
MAXENT_CLASSIFY <- classify_model(c, MAXENT)
GLMNET_CLASSIFY <- classify_model(c, GLMNET)
analytics <- create_analytics(c, cbind(SVM_CLASSIFY, GLMNET_CLASSIFY, MAXENT_CLASSIFY))
names(attributes(analytics))
```

```
## [1] "label_summary"    "document_summary" "class"
```

As you can see, the analytics now only has the `label_summary` and `document_summary`:

```
analytics@label_summary
```

```
##   NUM_CONSENSUS_CODED NUM_PROBABILITY_CODED
## 0                 410                   410
## 1                 390                   390
```

```
head(analytics@document_summary)
```

SVM_LABEL	SVM_PROB	GLMNET_LABEL	GLMNET_PROB	MAXENTROPY_LABEL	MAXENTROPY_
0	0.7404320	0	0.7576594	0	0.
1	0.6352287	1	0.7937420	1	0.
1	0.7229216	1	0.8304490	1	0.
0	0.6022700	0	0.7174788	0	0.
1	0.7229603	1	0.6833278	1	0.
1	0.5918435	1	0.7173803	1	0.

The label summary now only contains an overview of how many where coded using consensus and probability.
The document_summary lists the output of all algorithms, and the consensus and probability code.