We work on two aspects of Text Mining:

1. Sentiment Mining: Here, we are more concerned about deciphering the sentiment of the author.
2. Subject Extraction: Here, we wish to pull out the main subject of the chosen speech. This is done prior to sentiment mining.

You may find numerous ways on internet to do sentiment analysis. However, subject extraction is very specific to the context. In this article, I have shared the top 4 hacks applied in the industry to do subject extraction in R. For ease, I’ve also highlighted the strength and weakness associated with each trick.

**1. Keyword Match Algorithm**

This is the most powerful tool to do text mining. Let’s first look at the code in R to execute this step

ss <- read.csv("keywords.csv")

#Import the list of Keywords with first column as the keyword you wish to match and the tag you need to populate

Keywords <- as.character(ss$Keywords)

tags <- as.character(ss$Merchant\_Name)

for (i in 1:length(Keywords)) {

for (j in 1:nrow(Data1)) {

#Data1 is the complete data from which you are trying to extract the text. We will look at the text line by line

if(grepl(Keywords[i],Data1[j,1]) == 1){Data1[j,2] <- tags[i]

#Here is where you do an actual search

Data1[j,4] <- 1

#Flag 1 to those observations where you find a match

}

}

}

Now let’s try to see the strengths and weaknesses of this algorithm.

**Strengths**

1. It is highly effective in extracting keywords from not so well separated words. For instance, this algorithm can pull out “Tavish” from “#DataScientistTavishSrivatava”.
2. This algorithm has the option of assigning priority order in the keyword match algorithm. For instance, if I need to give “Tavish” higher priority than “Srivastava” in the above hash-tag, it can easily be done.

**Weaknesses**

1. It needs a pre-defined list of keywords from where you need to search.
2. It can capture many mis-classified cases. For instance, if want to search “APE” from the context, you will also erroneously tag “CAPE” as “APE”.

**2. Word Match Algorithm**

This is the fix for the second weakness (mis-classified cases) in the previous algorithm. In this algorithm, we try to match words instead of keywords. Here is the R-code :

words <- read.csv("word\_match.csv")

word <- as.character(words$Keywords)

tags <- as.character(words$Tag)

for (i in 1:length(word)) {

for (j in 1:nrow(Data1)) {

if(word(unlist(Data1[j,1]),1) == word[i]){Data1[j,2] <- tags[i]

Data1[j,4] <- 1

}

}

}

**Strengths**

1. It operates perfectly on finding well separated words. For instance, this algorithm can effortlessly pull out “Tavish” from “Tavish Srivatava”.
2. This algorithm also allows priority order in the word match algorithm. For instance if I need to give “Tavish” higher priority than “Srivastava” in the above hash-tag, it can easily be executed.

**Weaknesses**

1. It needs a pre-defined list of keywords from where you need to search.
2. It only captures the first well separated word. The algorithm can be modified to search among all words though.
3. It misses out not on so well separated words.

**3. General Expressions**

This methods needs extensive research on the sentence structures. For ease of understanding, I’ve taken an uncomplicated example of “www.dummyvalue.com”. Here is the code :

for (i in 1:nrow(Data1)) {

if(grepl("WWW",Data1[i,1]) == 1 & grepl("COM",Data1[i,1]) == 1){

start <- str\_locate(unlist(Data1[i,1]),"WWW")[2]

end <- str\_locate(unlist(Data1[i,1]),"CO")[1]

Data1[i,2] <- paste("www",tolower(substr(unlist(Data1[i,1]),start + 1,end-1)),"com", sep = ".")

Data1[i,4] <- 1}

}

**Strengths**

1. It does not need any kind of list to start with.
2. Usually, it turns out to be highly accurate if you are able to find out a strong regular expression.

**Weaknesses**

1. It needs deep research to create a regular expression.
2. In case of a not so well structured data, this method is able to tag a very small number of observation

**4. Word Association:**

I bet, this method is good enough to challenge you intellectually. So, that you could work on it, instead of giving away the entire code, I’ve provided the step by step methods to do the same. If you still find it difficult, mention your request for code in the comment section below.

Step 1: Find most frequent words which can possibly be something what you are looking for.

Step 2: Find the most associated word with these frequently occurring words.

Step 3: For each of the pairs find the best frequency-association pair (this will need some number of iterations)

**Strengths**

1. No dictionary is required.
2. If parameters are optimized well, it can be highly predictive.
3. It can act as a feedback to other algorithms.
4. You can use this algorithm even if you don’t know the language of the text.

**Weaknesses**

1. It sometimes is not very precise on the subject name. It tends to capture even those trends which does not mean anything significant.