Data Science Capstone - Tweet Classsification System - Catastrophe or Not?

Carlos Yáñez Santibáñez

April 07, 2020

Contents

1 Introduction

2 Analysis

As mentioned in the Introduction, this report presents the attempt to build a machine learning model to classify Tweets in a binary model. In this particular case, the question to answer is whether those tweets correspond to a catastrophic event or not. The idea for this was taken of a Kaggle Competition for Starters (Real or Not? NLP with Disaster Tweets). Instead of using the data provided in Kaggle, in this report the original version in Data For Everyone is used.

To start, let's have look at sample data:

Table 1: Sample Data

id	target	text	keyword	location
1	1	Just happened a terrible car crash	NA	NA
6	1	All residents asked to 'shelter in place' are being notified by officers. No other evacuation or shelter in place orders are expected	NA	NA
33	0	London is cool ;)	NA	NA
49	1	@bbcmtd Wholesale Markets ablaze http://t.co/lHYXEOHY6C	ablaze	Birmingham
160	0	'The harder the conflict the more glorious the triumph.' Thomas Paine	aftershock	304
914	1	Uganda Seen as a Front Line in the Bioterrorism Fight	bioterrorism	Alaska
3 999	0	I forgot to bring chocolate with me. Major disaster.	disaster	Los Angeles, London, Kent
7 336	1	Finnish ministers: Fennovoima nuclear reactor will go ahead http://t.co/vB3VFm76ke #worldnews #news #breakingnews	nuclear%20reactor	World

As seen above, the dataset cointains the following observations (to match Kaggle's format):

- **id** : uniqe sequential identifier.
- **target** : indicator of whether the tweet corresponds to a catastrophic event (**1**) or not (**0**).
- **text** : tweet, as extracted from Twitter.
- **keyword** : search term used in Twitter's search bar to retrieve the tweets.
- **location** : Name of the place wherre the tweet in question originated.

For this report, **keyword** and **location** will be ignored, choosing to focus on the text (tweet) only. After a brief observation of its content, in this analysis the text will be divided into four component:

- The text proper. i.e. the *natural language* sentence in the text. In the code, this will be called **words**. All the tweets in this file are written in English language, but they may contain non-English characters, emojis and emoticons.

- The **hashtags**, which are in user created categorisation. This is a meaningful component of the tweet and may help clarify whether the message is a real disaster. For example, in below tweet, the hashtag helps us to ellucidate this is cricket commentary rather than a fire disaster:

Table 2: Tweet about cricket

id	text
1 674	Australia's Ashes disaster - how the collapse unfolded at Trent Bridge... http://t.co/Dq3ddGvgBF #cricket

- The **links** contained in the tweet - perhaps the respective Internet domain could help distinguish "quality" disaster sources (e.g. official announcement from relevant authority, The Red Cross/Crescent) from non-disistarr ones (e.g. gossip magazine using hyperbolic language). To obtain meaningful content though, the links need to be "unshortened" from the t.co addresses provided by Twitter.
- Who is mentioned in the tweet (by their Twitter **handles**). Similarly to the previous points, it may be useful to distinguish between people reaching out to government/emergency service from *shock jocks* using florid language.

2.1 Processing and cleaning up the data

In order to obtain the data in the desired format for analysis, the function *prepare_data* has been created. Its code performs the below activities:

1. Extract hasthags, links and Twitter handles from the tweet.
2. Clean up the remaining text, removing links and handles, and performing transformation operations on emojis,emoticons and other special characters.
3. (Optional) Replace swear words and profane language with unique strings, in case seeing those wants to be avoided, yet retaining the words for analysis. Please note this report has been generated with the uncensored version, thus results may vary.
4. "Unshorten" link URLs to obtain domain names of linked sources.

The first step is similar for the three components - below is the code use to remove the hashtags:

```
source_data$hashtag <- str_extract_all(source_data$text, "#\\S+")
 source_data <- source_data %>%
   mutate(hashtag = gsub(x=hashtag,
          pattern = "character\\(0)", replacement = "")) %>%
   mutate(hashtag = gsub(x=hashtag, pattern = "c\\(", replacement = "")) %>%
   mutate(hashtag = gsub(x=hashtag, pattern = "\"", replacement = "")) %>%
   mutate(hashtag = gsub(x=hashtag, pattern = ")", replacement = "")) %>%
   mutate(hashtag = gsub(x=hashtag, pattern = ",", replacement = ""))
```

In the following step, both links and mentions are removed, keeping the hashtags since they may also be part of the sentence in question. Then, leveraging the functions in the **textclean** package, the remaining data is further normalised, by replacing :

- contractions with full words,
- emojis and emoticos with equivalent word (e.g. :) with 'smile'),
- all non-ASCII characters wit equivalent,
- internet slang with full words,
- html code,
- money symbols,
- numbers with full words,
- ordinals with full words replace_ordinal,
- timestamps.

The code that achieves this is shown below:

```
 #remove mentions and links from text, save original text in different observation

   source_data$original_text <- source_data$text
   source_data <- source_data %>%
    mutate( text = gsub(x = text, pattern = "#", replacement = ""))  %>%
    mutate( text = gsub(x = text, pattern = "@\\S+", replacement = "")) %>%
    mutate(text = gsub(x=text,
                       pattern = "(s?)(f|ht)tp(s?)://\\S+\\b",
                       replacement = ""))

#further clean text with textclean package

  source_data$text<-replace_contraction(source_data$text)
  source_data$text<- replace_emoji(source_data$text)
  source_data$text<- replace_emoticon(source_data$text)
  source_data$text<-replace_non_ascii(source_data$text,impart.meaning=TRUE)
  source_data$text<-replace_internet_slang(source_data$text)
  source_data$text<-replace_html(source_data$text)
  source_data$text<-replace_money(source_data$text)
  source_data$text<-replace_number(source_data$text)
  source_data$text<-replace_ordinal(source_data$text)
  source_data$text<-replace_time(source_data$text)
```

An optional step is to remove swear and profane vocabulary from the text, in case that is desired. This has been done through the below code:

```
  #profanity removal

  if(profanity_clean==1){

    #download profane words and prep for matching
    data(profanity_zac_anger)
    data(profanity_alvarez)
    data(profanity_arr_bad)
    data(profanity_banned)
```

```
data(profanity_racist)
Sys.sleep(100)

special_chars <- as_tibble(c("\\!", "\\@",  "\\#","\\$", "\\&","\\(","\\)",
                             "\\-","\\‘","\\.","\\/","\\+",'\\"','\\"'))
special_chars$replacement <- paste0("st",
                                    stri_rand_strings(nrow(special_chars),
                                    3, '[a-zA-Z0-9]'))

profanity<-as_tibble(c(profanity_zac_anger,profanity_alvarez,
                       profanity_arr_bad,profanity_banned,
                       profanity_racist))
profanity<-unique(profanity)
profanity$replacement <- paste0("pr",
                                stri_rand_strings(nrow(profanity),
                                7, '[a-zA-Z0-9]'))

for(i in 1:nrow(special_chars)){
  profanity <- profanity %>%
    mutate(value=gsub(special_chars[i,]$value, special_chars[i,]$replacement,
                      value))
}
profanity <- profanity%>% mutate(value=paste0('\\b', value, '\\b'))

rm(profanity_zac_anger,profanity_alvarez,profanity_arr_bad,
   profanity_banned,profanity_racist)

#clean up profane words, replace by random string

for(i in 1:nrow(special_chars)){
  twitter_df <- twitter_df %>%
    mutate(text=gsub(special_chars[i,]$value,
                     special_chars[i,]$replacement,
                     text))
}

for(i in 1:nrow(profanity)){
  twitter_df <- twitter_df %>%
    mutate(text=gsub(profanity[i,]$value,
                     profanity[i,]$replacement,
                     text,ignore.case = TRUE))
}


twitter_df <- twitter_df %>%
  mutate(text=gsub("st[a-zA-Z0-9][a-zA-Z0-9][a-zA-Z0-9]",
                   "",text,ignore.case = TRUE))
```

```
    rm(profanity,special_chars)
  }
```

As a last step, the functions XXX from the YYY package has been leveraged to unshorten the t.co URLS, to obtain the domains of the links. This is done trough the below code:

```
split<-200
value <- round(nrow(processed_source)/split,0)
segments <- tibble(start=integer(),end=integer())
segments <- add_row(segments, start=1,end=value)

for(i in 2:(split-1)){
   segments <- add_row(segments, start=value*(i-1)+1,end=i*value)
}

   segments <- add_row(segments, start=value*(split-1)+1,
                       end=nrow(processed_source))

# first segment
   segment <- processed_source[segments[1,]$start:segments[1,]$end,]
   domains<-get_domains(segment,anonimised = 0)
   iteration <- 1

for(i in 179:split){
  segment <- processed_source[segments[i,]$start:segments[i,]$end,]
   domains_i<-get_domains(segment,anonimised = 0)
   domains <- rbind(domains,domains_i)
   iteration <-segments[i,]$start
}

domains <-  domains %>%
    mutate(domain = gsub(x=expanded_url,
                         pattern = "(http|ftp|https)://",replacement = "")) %>%
    mutate(domain = gsub(x=domain,
                         pattern = "www.", replacement = "")) %>%
    mutate(domain = gsub(x=domain,
                         pattern = "ww[0-9].", replacement = "")) %>%
    mutate(domain = gsub(x = domain,
                         pattern = "/S+", replacement = ""))

domains$domain <- domain(domains$expanded_url)
domains <-  domains %>%
    mutate(domain = gsub(x=domain, pattern = "www.", replacement = "")) %>%
    mutate(domain = gsub(x=domain, pattern = "ww[0-9].", replacement = "")) %>%
    mutate(domain = gsub(x = domain, pattern = "m.", replacement = ""))

domains_list <- domains %>% select(domain) %>% unique(.)
```

```
domains_list$domain_key <- paste0("domain_",seq.int(nrow(domains_list)))
domains <- domains %>% left_join(domains_list,by="domain")
```

As result of this processing, we have two data frames to use for further analysis: one with the processed tweets and another with a list of t.co URLs and domains. A sample of boths is shown below:

Table 3: Sample Data

id	target	text	hashtag	link	mention
1	1	Just happened a terrible car crash			
6	1	All residents asked to 'shelter in place' are being notified by officers. No other evacuation or shelter in place orders are excuse me tongue sticking out ected			
33	0	London is cool wink			
49	1	Wholesale Markets ablaze		http://t.co/lHYXEOHY6C	@bbcmtd
160	0	'The harder the conflict the more glorious the triumph.' Thomas Paine			
914	1	Uganda Seen as a Front Line in the Bioterrorism Fight			
3 999	0	I forgot to bring chocolate with me. Major disaster.			
7 336	1	Finnish ministers: Fennovoima nuclear reactor will go ahead worldnews news breakingnews	#worldnews #news #breakingnews	http://t.co/vB3VFm76ke	

Table 4: Sample Data

link	domain	domain_key
https://t.co/irWqCEZWEU	bbc.co.uk	domain_1
https://t.co/lHYXEOHY6C	twitter.com	domain_2
https://t.co/pmlOhZuRWR	sigalert.com	domain_17
https://t.co/GKYe6gjTk5	lawsociety.org.uk	domain_18
https://t.co/FaXDzI90dY	change.org	domain_61
https://t.co/SB5R7ShcCJ	change.org	domain_61
https://t.co/ZNTg2wndmJ	twitter.com	domain_2
https://t.co/JLRw0Oi9ee	twitter.com	domain_2

Both datasets in their "uncensored" and "sanitised" versions have been made available on GitHub.

Before proceeding, we will split this data into three datasets:

- a **training** dataset, to be used for further analysis, modelling and tuning.
- a **testing** dataset, to be used in tuning.
- a **validation** dataset, which will put aside and only use at the end to generate the last results.

Unless said otherwise all the below analysis has been done with the **training** dataset only!

2.2 Scoring each tweet.

In order to use a machine learning method that allows us to determine whether any relationship between the content of each tweet and its target status, we need to be able to generate a function that rates each piece of content. A way of do this is to generate a numeric value for each tweet, that then can be used for estimate whether it is "catastrophic" or not.

In the case with text, a simple way to achieve this is to assign each constituent word a score and then use those to calculate a sentence value (with a simple way of calculating such value being adding to individual scores). The following lines explain the way this was done in this particular exercise - please note that as a starting point, we are making them assumption this is a good method - how to determine the right formula is a problem by itself.

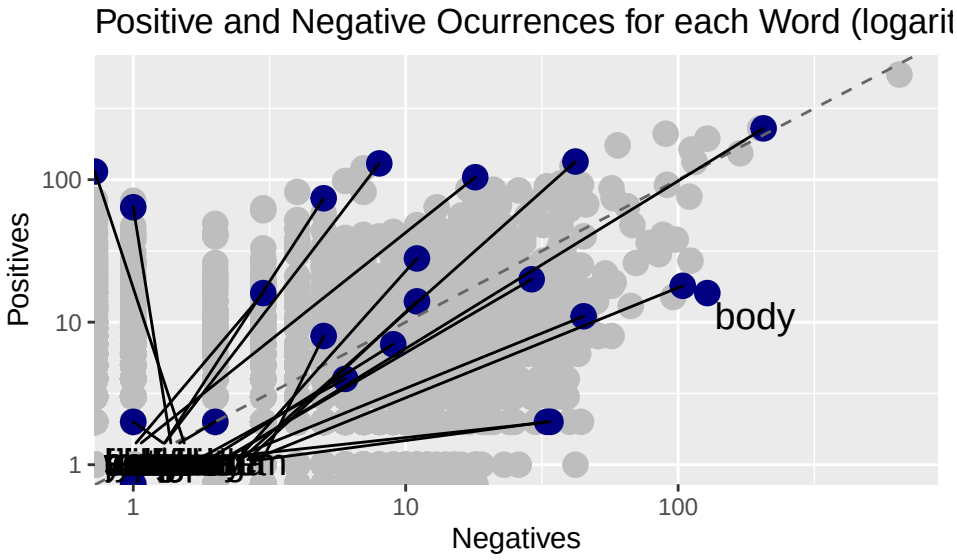
Taking the each tweets processed text, the first step is to "tokenise" this dataset, i.e. split it in its component words - for this the function *unnest_tokens* from the *tidytext* package has been used. Since we would like to know how this words are split between catasthropic ("positive") and non-catastrophic ("negative") entries and perhaps use this a scoring base, we will also remove "stop words" (common words such as "and" "or", basic verbs,etc), we will filter them using the *stop_words* dataset available on *tidytext*.

The code to achieve this goes as follows:

```
data("stop_words")
extra_stop_words$lexicon <- "EXTRA"
stop_words<-rbind(stop_words,extra_stop_words)
rm(extra_stop_words)


tokenised_words <- training_dataset %>% unnest_tokens(word,text)
tokenised_words <- output$tokenised_words %>%
                           anti_join(stop_words, by="word")
```

With the data, we can calculate how frequent each word is, in general and in both positive and negative tweets. The chart below shows this, highlighting some selected words for analysis:

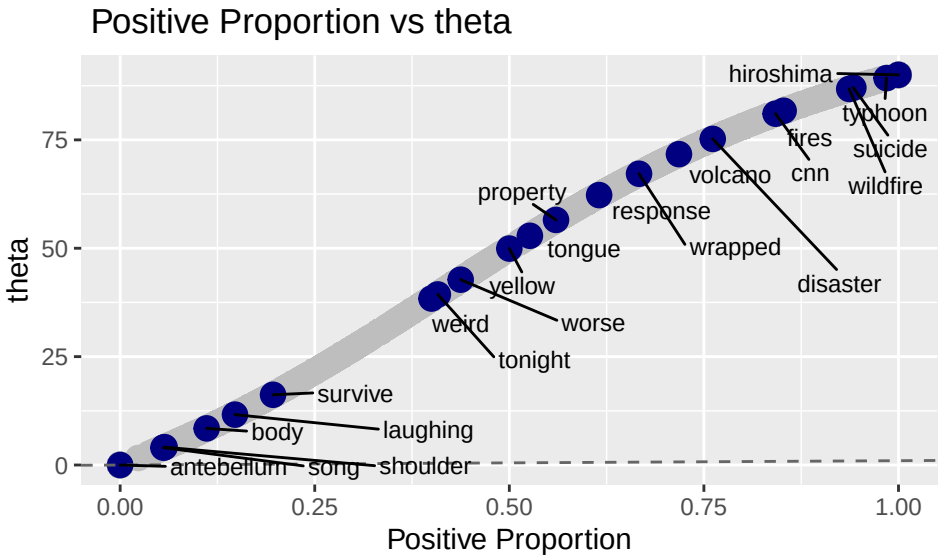


Since we are looking at the impact of the word for a tweet being positive, the above chart can be reduced to one indicator, the proportion each words appears positive tweets against its total number of appearances.

From here, the question is how to take this into a score for each word that:

- Properly weighs "disaster" related words.
- Penalises "non disaster" words but without reducing the impact of more ambivalent terms.
- Allows to establish a clear division, so when compoundend it clearly helps to distinguish positives from negatives.

After thinking how to achieve this objective, a score was defined taken angle of each point in the avoid chart, being this an effective measure of how "positive" or "negative" a word is. If we call this angle θ, we can generate an initial score that looks like the below chart:



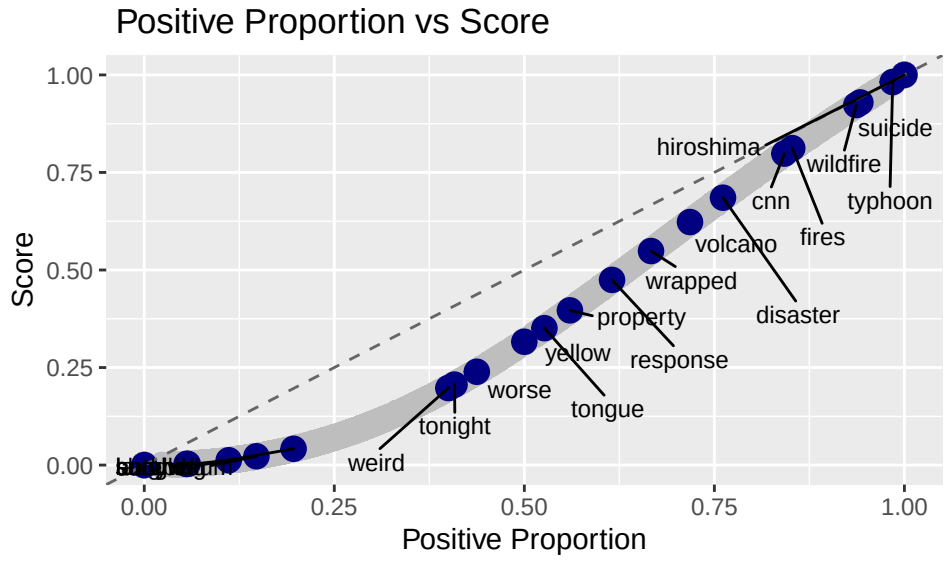
Although this is a good approach, after several iterations the following formula has been chosen instead:

$$pre\_score = sin(pos\_proportion)^{\lambda_1} * \frac{\theta}{90°}$$

$$score = \frac{pre\_score}{max(pre\_score)} - \lambda_2$$

where * $pos\_proportion$: Proportion of times where word was found in a "positive" tweet. This sinus of this parameter is used to help penalise less relevant words. * λ_1: Parametrisation parameter, used to modulate the effect of less relevant words. * θ: angle of vector $(positive\_proportion, negative\_proportion)$, as discussed previously. * λ_2: Offset to potentially give a negative value to less relevant words, in order to provide compensation - e.g. in case a "disaster" word is used in figurative context amongst many non relevant words. This will be treated as a tuning parameter.

Setting λ_1 to 1 and λ_2, we have a sample distribution, as follows:

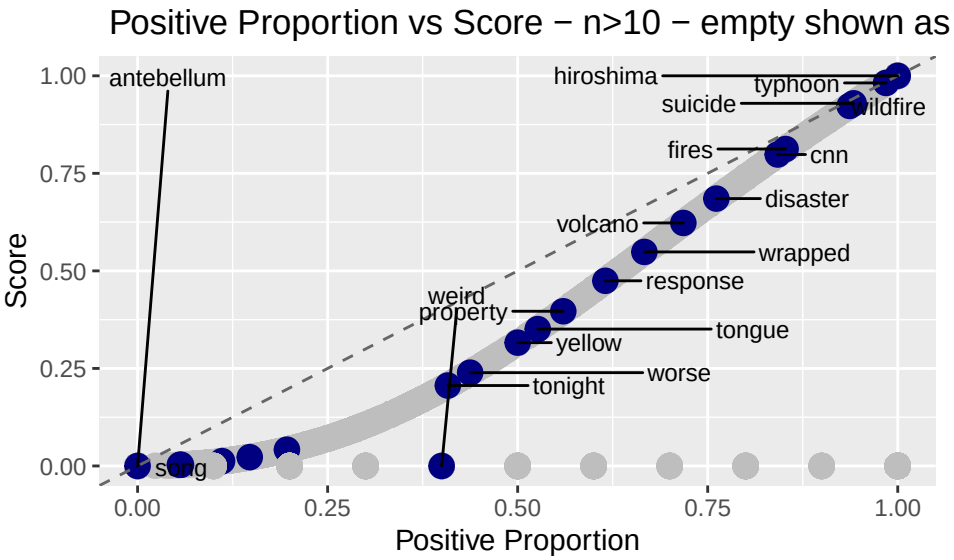


Looking at the chart above, we have achieve the objectives of giving distinct scores depending on the apparent "catastrophe" value of each word. This however, does not reflect the amount of times a word appears - it may be the case that a particular term appears just one and gets an extreme score which is not really a good indicator of its real "catastrophe value. Thus, a last modification of the formula es required:

$$score = \begin{cases} \emptyset & n \leq \lambda_3 \\ \frac{pre\_score}{max(pre\_score)} - \lambda_2 & n > \lambda_3 \end{cases}$$

where * n : number of times the word is appears in the training set, * λ_3: arbitrary theshold to filter word with a low number of ocurrences (and possibly biased). This is a tuning parameter.

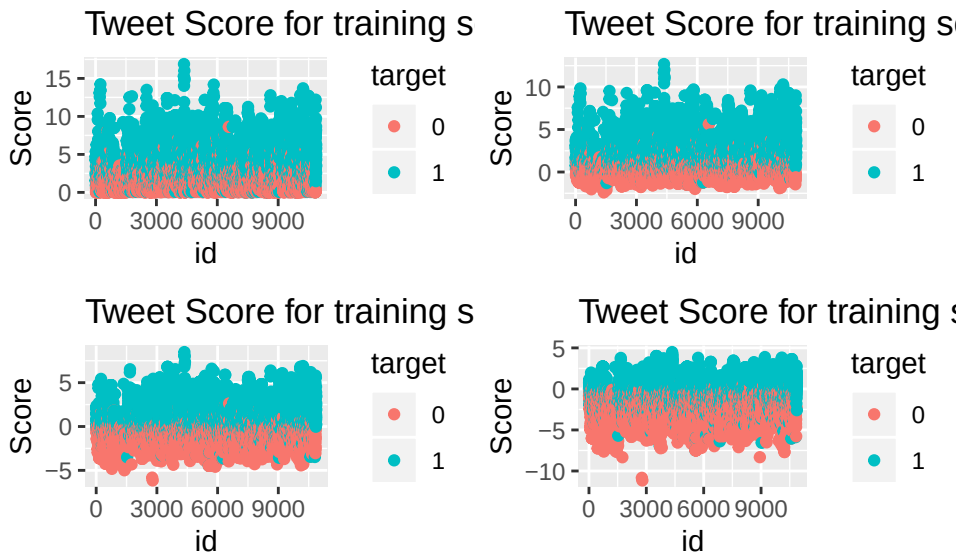
Using $\lambda_3 = 10$ for illustration purposes, we have the below chart:



With a set of scores, we would like to know if this produce a set of tweet scores that are a potential good input for a regression function. Due its simplicity, if we using a sum to calculate a score per tweet we can obtain those values with the below code:

```
tweet_score <- tokenised_words %>%
               left_join(word_scores,by="word") %>% filter(!is.na(score)) %>%
               group_by(id) %>% summarise(word_score=sum(score)) %>%
               ungroup()
```

wich, will produce results like the below graphs (for different offset values):



From the above, we can see that there is a diffuse division between positive and negative tweets. It is worth noticing this could potentially be improved with better modelling - however this could a problem but itself and the above - with tuning - is deemed acceptable for a first approximation.

A similar process has been done two create tweet scores for hasthags, links(or rather the domains of the unshorten URLs) and mentions. After try and error, a slightly different formula has been chosen for these three observations, represented below for hasthags:

$$pre\_score = \left(\frac{pos\_proportion}{\lambda_1}\right)^{\lambda_2} \cdot \frac{\theta}{90}$$

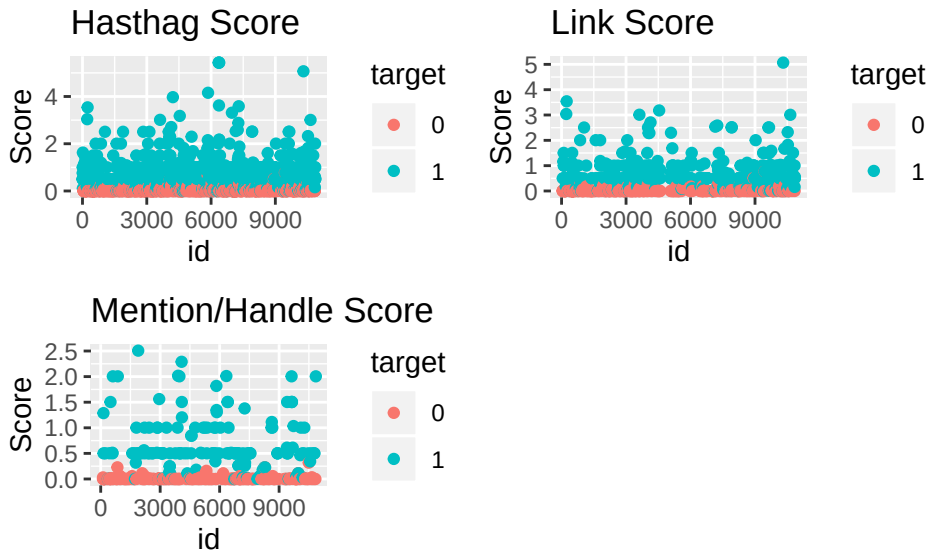
$$score = \begin{cases} \emptyset & n \leq \lambda_4 \\ \frac{pre\_score}{max(pre\_score)} - \lambda_3 & n > \lambda_4 \end{cases}$$

where

- *pos_proportion* is the proportion of the term in the entire corpus of positive hasthags/domains/mentions,
- λ_1 tuning parameter that re-scales - everything above the threshold is relevant, the rest not so much,
- λ_2 tuning parameter to weigh down irrelavent terms and weigh up relevent ones,

- θ angle of vector $(negatives, positives)$,
- λ_3, Offset to potentially give a negative value to less relevant term, in order to provide compensation - e.g. in case a "disaster" word is used in figurative context amongst many non relevant term This will be treated as a tuning parameter.
- λ_4, threshold of number of ocurrences necessary to consider the score as valid.

The below charts show an example of score generated using this method:



In order to facilitate the next steps, the above process has been written into three functions, namely:

- **tokenise_data**, which takes the data in the pre-processed format and generates a dataframes with the token words, hasthags, domains and Twitter handles.
- **calculate_scores**,which takes the output of the previous function (*tokenised* data) and calculate the scores for each word, hashtag, link and handle. All the previously discussed tuning parameter have been defined as inputs for this function.
- **score_tweets**, which will generate a score for earch tweeet for each of categories previously mentioned.

All these functions are written in the accompanying .R file, with inline commentary explaining all the steps involved in the process.

2.3 Finalising the model and data quality.

From here, we can move to the next step and to generate the last part of the model - the one that will render a prediction.

Considering the work done so far, there are potentially many ways to use to the scores to create a model for prediction. After weighing the options, the following two-step approach will used:

1. First, a regression algorithm will be used to train a model for each attribute - i.e. 4 different predictions will be generated using each of the 4 scores previously created independently. The result of this process will be vector for each tweet, with the results of each prediction (1,0 or potentially N/A if there is no such attribute for a given tweet.)
2. Then this vector will be used again as a input of a second machine learning method , which will attempt to predict the target result based on the individual responses.

For this process, two functions have been written in this report .R file:

- **fit_model**, which is the training function. Its inputs are the scores plus relevant tuning parameters. It will output 5 machine learning models (for each attribute plus the aggregate one).
- **predict_values**, which will take the training score, the test data and the models previously generated. it will output the predictions and measure accuracy if the target value is provided.

These functions leverage the caret package and they have been written to provide some flexibility - for example **fit_model** allows to be run using different machine learning models. For further details, please refer to the detailed commentary in the .R file.

For the initial run - that will help us establish and initial benchmark - the general linear model (glm) in caret is used for the indidivual predictions. For the aggregate preidction, we will use xgbBoost. For this the below code is used:

The accuracy for table and confusion matrix for this example are:

Table 5: Initial results

eval	result_method	result_agg
word	0.7556701	0.7567010
hashtag	0.7076923	0.8307692
link	0.7543860	0.7719298
handle	0.8095238	1.0000000
aggregate	0.7571429	0.7571429

Table 6: Initial results - Confusion Matrix

	0	1
0	470	72
1	166	272

As observed, there are large number of false positives and false negatives. A sample of those is provided below:

Table 7: Initial results - False Positives

id	target	original_text
368	0	U.S National Park Services Tonto National Forest: Stop the Annihilation of the Salt River Wild Horse... https://t.co/W5Rhtuey90 via @Change
744	0	Metro Nashville Police - simply the best. Thank you for protecting us. 911 call: http://t.co/ZWIG51QECF via @AOL
1 699	0	Listening to Blowers and Tuffers on the Aussie batting collapse at Trent Bridge reminds me why I love @bbctms! Wonderful stuff! #ENGvAUS
1 716	0	Leicester_Merc : ICYMI - #Ashes 2015: Australia collapse at Trent Bridge - how Twitter reaâ€_ http://t.co/HqeWMREysO) http://t.co/y4y8fclJED
4 385	0	#USGS M 1.4 - 4km E of Interlaken California: Time2015-08-06 00:52:25 UTC2015-08-05 17:52:25 -07:00 at ep... http://t.co/zqrcptLrUM #SM
9 435	0	People with netflix there's a really good documentary about Hiroshima narrated by John Hurt. 2 Parter that interviews Pilots + Survivors.
9 641	0	Falling asleep to the sounds to thousands of River Plate fans in the stadium and a thunderstorm. #VivaArgentina

Table 8: Initial results - False Negatives

id	target	original_text
351	1	Please sign & RT to save #SaltRiverWildHorses http://t.co/GB8ispiaRP http://t.co/Bx0l87iNc8
2 160	1	Learning from the Legacy of a Catastrophic Eruption http://t.co/RbmuCURS2F
2 817	1	First time for everything! @ Coney Island Cyclone https://t.co/SdNT3Dhs3W
4 173	1	I can't drown my demons they know how to swim
4 614	1	Call for Tasmania's emergency services to be trained in horse rescues http://t.co/zVQwLpScSC
4 721	1	Firefigthers Evacuate from Northampton Township House Fire http://t.co/hPplD1jHtZ
7 214	1	its only getting colder and colder and faster and faster and when i first realized it it was like a natural disaster

A more detailed review of both sets, seems to indicated that the source file curation does seem to contain some errors : tweets that are seemingly not a catastrophe are tagged as such and vicerversa. One particular example of this are tweets allusive to the Hiroshima bombing in World War 2. Although those tweets a tragedy remembrance and do not carry the connotation of a current catastrophe, there are marked as "disaster" in the dataset.

Table 9: Initial results - Hiroshima Tweets

id	target	original_text
1 536	1	The Guardian view on the Hiroshima legacy: still in the shadow of the bomb | Editorial: The world longs to cas... http://t.co/RhxMGhsPd7
1 557	1	The crew on #EnolaGay had nuclear bomb on board disarmed. 15 mins to #Hiroshima they got ready to arm Little Boy http://t.co/JB25fHKe6q
1 599	1	6th July 1945 ...Hiroshima was bombed with nuclear weapon _bomb was named little boy&carried in a plane called Enola Gay ...1/2
1 633	1	It's been 70 years (and one hour) since the bombing of Hiroshima. Let's take this time to remember.
1 637	1	Today marks the 70th anniversary of the bombing of Hiroshima a city I visited and was humbled by in November 2013 http://t.co/AcC1z5Q9Zw
1 640	1	The cryptic words that guided pilots on the Hiroshima bombing mission http://t.co/nSS5L64cvR #canada
1 661	1	#Japan marks 70th anniversary of #Hiroshima atomic bombing (from @AP) http://t.co/qREInWg0GS
3 736	1	70 years after #ABomb destroyd #Hiroshimaâ€"#BBC looks at wht #survived http://t.co/dLgNUuuUYn #CNV Watch Peace Vigils: http://t.co/jvkYzNDtja
5 190	1	Estimated damage and fatalities of an Hiroshima-sized atomic bomb dropped on your hometown - http://t.co/BSrERJbY0I #Hiroshima70
9 367	1	The 390-Year-Old Tree That Survived the Bombing of Hiroshima http://t.co/kEirA8MA3K
9 376	1	'Planted 390 years ago' it was moved to U-S. This Bonsai Survived Hiroshima But Its Story Was Nearly Lost http://t.co/jID4RO34gb via @NatGeo
9 435	0	People with netflix there's a really good documentary about Hiroshima narrated by John Hurt. 2 Parter that interviews Pilots + Survivors.

The above reflects two important problems with machine learning and natural language:

1. This re-enforces that quality training data is essential. Bad data will probably result in bad results - even with a very sophisticated algorithm (It's called *Data* Science after all!).
2. Human language is complicated. Meaning of each word is highly contextual and changes depending on the register, dialect and over time. This can be difficult for a machine learning model to **grasp**.

To illustrate the second point, the above false positives and false negatives have been manually review and their new positive/negative result has been modified accordingly and added into the original data set. After feeding them back into the trainig dataset, selecting a new testing sample and running the machine learning model again, the below results are obtained:

This illustrates that better training data can lead to better results. It is also worth noticing this may well reflect a real-life scenario, where a social media monitoring centre is constantly reviewing false negatives and false negatives with the purpose of keep re-training a model to adapt to changes language, emerging events, etc.

Of course, the other components in the model can also benefit of this *curated* approach. In this report, and for illustraition process - the list of hasthags in thre training set has been reviewed and

Table 10: Initial results - corrected dataset

eval	result_method	result_agg
word	0.8174078	0.8183538
hashtag	0.8024691	0.8148148
link	0.7837838	0.8108108
handle	0.7619048	0.8095238
aggregate	0.8188679	0.8188679

then detected a couple of hashtags that most probably mean catasthrophe (e.g. like #earthquake most like means one) and other most probably won't (like #ashes2017, which is unmistakebly cricket, not matter how colourful and hiperbolic is language).

Using a sample set of curated hashtag, it is possible to observe a small increment in accuracy. Handles and links will be left as a action to improve for the time being.

Table 11: Initial results

eval	result_method	result_agg
word	0.8174078	0.8183538
hashtag	0.8024691	0.8148148
link	0.7837838	0.8108108
handle	0.7619048	0.8095238
aggregate	0.8188679	0.8188679

Looking at the work done so far, we have the below table showing the improvements obtained by improving and supplementing the data:

2.4 Tuning

With this new, "augmented" dataset the next step of the process is to tune the model. Looking at how the model it's constructed, there are *many* aspects to tune, namely:

- The parameters used to calculate the tweet, hasthag, link and mention scores.
- The selection of classification algorithms used.
- The tunning parameters of each selection algorithm.

Obviously, tuning every single parameter will result in quite a lengthy excercise. Instead, this report will focus on some of the key components in this model and discus what works and what else could be done.

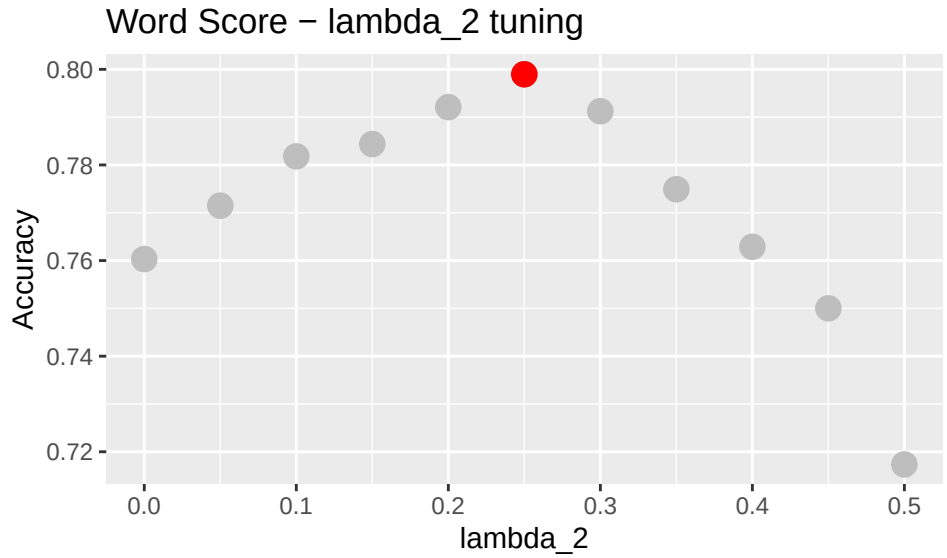
The first aspect to tune is the each word in the training corpus gets a indvidual score. This is perhaps the key component of the model. From the previous section, we have that this score is calculated through the below formulas:

$$pre\_score = sin(pos\_proportion)^{\lambda_1} * \frac{\theta}{90°}$$

$$score = \begin{cases} \emptyset & n \leq \lambda_3 \\ \frac{pre\_score}{max(pre\_score)} - \lambda_2 & n > \lambda_3 \end{cases}$$

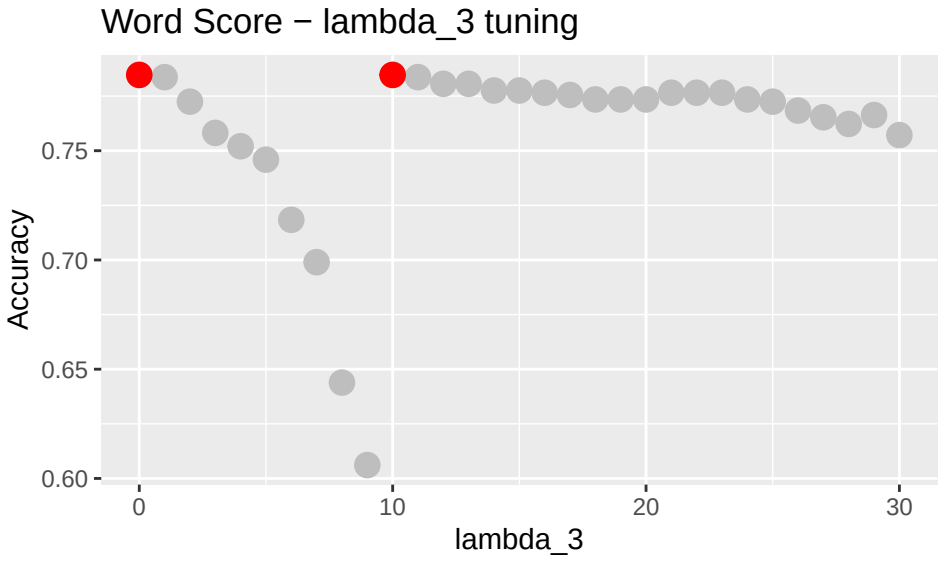
where * *pos_proportion*: Proportion of times where word was found in a "positive" tweet. This sinus of this parameter is used to help penalise less relevant words. * λ_1: Parametrisation parameter, used to modulate the effect of less relevant words. * θ: angle of vector $(positive\_proportion, negative\_proportion)$, as discussed previously. * λ_2: Offset to potentially give a negative value to less relevant words, in order to provide compensation - e.g. in case a "disaster" word is used in figurative context amongst many non relevant words. This will be treated as a tuning parameter. * n : number of times the word is appears in the training set, * λ_3: arbitrary theshold to filter word with a low number of ocurrences (and possibly biased). This is a tuning parameter.

In this report, λ_2 and λ_3 will be tuned, to see how the penalisation of irrelevant words and minimum number of ocurrences affect the model. The results of tuning lambda_2 are shown below.



As shown above, the best result is obtained with $lambda_2$=0.25. It is also worth noticing that increasing this value won't necessarily result in better results, since it will then penalise "neutral" terms too.

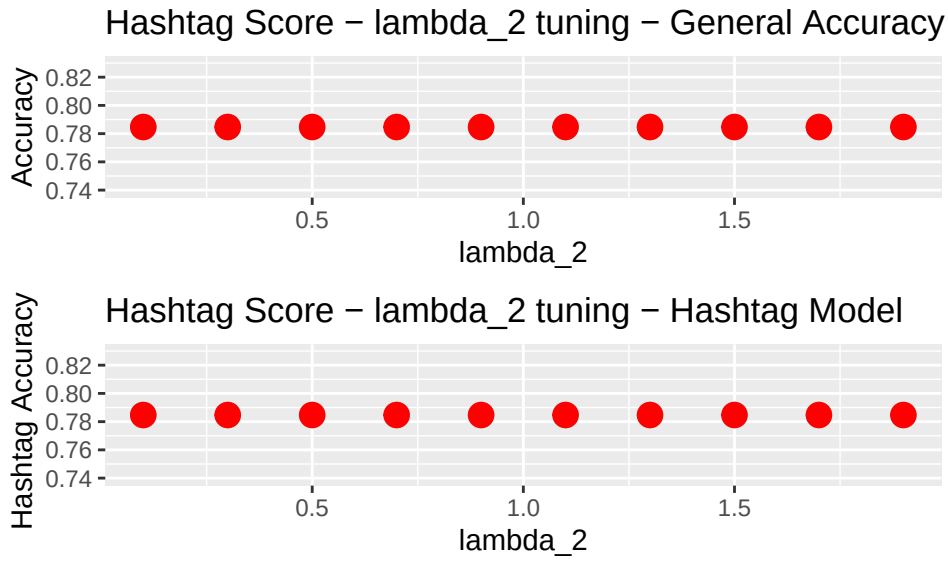
With this value, we will then also find if there is an optimal λ_3, i.e. to see if removing the words that appear seldomly have an negative impact in the model. The results are shown below:



This result is a bit more interesting. As observed above, the first optimal λ_3 is quite a low value, however there is a relative bigger decrease in accuracy around its neighbours when compared against the second best λ_3. It is important to notice that the accuracy depends on the corpus of words in the test set and how they are combined into sentences. This is of course random and it will change from set to set. Even though we can do cross validation, perhaps it is not a bad a idea (as an initial step), choose a perhaps sub-optimal value for the sake of consistency - at least until more thorough tuning is possible. Therefore, to continue with this excercise, a value of $lambda_3$=10 has been chosen.

After the word scoring, perhaps the most relevant component in the model is the scoring function for hashtags (given that hashtag intend to be **meaningful** descriptors). Similarly to the word scoring, we will tune the equivalent parameters hashtag's $lamdba_2$ and $lamdba_3$.

The results for the tuning of $lamdba_2$ are shown below:

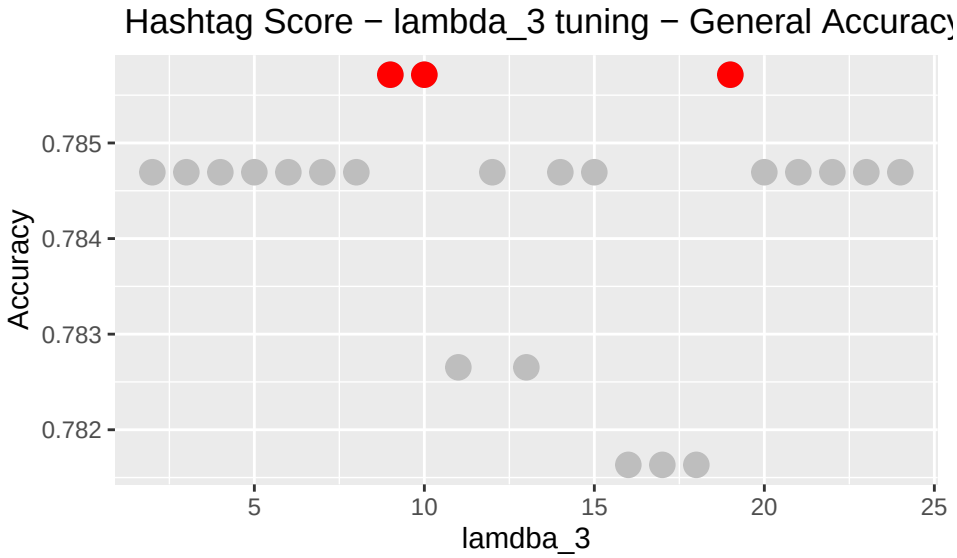


The first graph shows the overall accuracy results (comparable to the word tuning), while the

second graph shows the results where there is a hasthag. From them, the following points can be made:

- Smaller variations in the overall score can be explained to the fact that not all tweets have hasthags, thus the effect of tuning this part of the model will obvisouly have a limited effect on the general result.
- Taking the above point, it looks like its contribution to general score it is mostly when hasthags do indicate a cathastrophe (and therefore have a bigger score). It is worth noticing that many of the high-score hashtag were not ranked by the algorithm but manually added - which is perhaps and indication that manual curation is not necessarily a bad idea - especially when data is poor quality or insufficient.

With this mind, the tuning of $lamdba_3$ will help to determine the overall effect of hashtag occurence. The results are shown in the below chart.



This result seems similar in nature to the tuning for quantity cut-off for the word scoring.

Looking at the results so far (table below), this tuning process has only resulted in small gains when compared with the initial data corrections. Taking that the effect of hasthag tuning is small, in this report the tuning of the links and handle scoring functions will not be conducted. This doesn't mean they may bring an improved accuracy but perhaps are not worth the effort given the existing training dataset.

Instead, the decission is to focus on the tuning of the classification functions. At this point, the model uses the caret package to leverage to known machine learning algorithms, namely:

- *glm* (as in generalised linear model) in its logisitc regression mode to turn word, hashtag, link and mention to make a prediction on those components independently.
- *xgbTree* (extreme gradient boosting) to predict an "overall" positive or negative based on the previous predictions.

Considering that the assignment instructions state **"For this project, you will be applying machine learning techniques that go beyond standard linear regression"**, the next stage will be to asses if there is any model that produces better results than glm. (Although so far this is not a **standard** linear regression model anymore.)

After investigating a number of possible algorithms, the **LogitBoost** will use as comparison. This algorithm is available in R through the **caTools** package.

As a starting point, we will try this algorithm for the word score classification function. The below results show the best result after tuning its only parameter, which is the number of iterations.

These results are no better than the ones using the a simple logistic regression. Similar results are obtained when the same comparison is repeated for the other scoring components.

3 Results

4 Conclusion

- Good quality data matters. Bad input means bad results.
- Sometimes manual curation will improve the results.
- There is so much to tune.