Final Report : Toxic Comment Classification Project

Report Prepared By:

Sugand. A

April 29, 2018

Disclaimer: the report contains text that may be considered profane, vulgar, or offensive.

Table of Content

Executive Summary

This report summarizes the modeling and analysis results associated with the Toxic Comment Classification.The purpose of this report is to document both the implemented design and all corresponding data modeling and inference techniques used during the subsequent statistical analyses. Programming language used was R through a kernel in Jupter notebook.The evaluation metric used is Area Under the Curve.

1.0 Problem Statement

Discussing things you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation.One area of focus is the study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion).

Comments in facebook, twitter etc always means a lot to a person and sometimes can be stressfull to deal with. It sometimes makes a person uncomfortable if there are any abusive comments , so what makes these comments such is the words used and impact it has on people, obviously every word has a vibe associated with it.

Example: You were looking gorgeous today!

This inturn creates happiness, confidence, a sense of beauty within themselves.

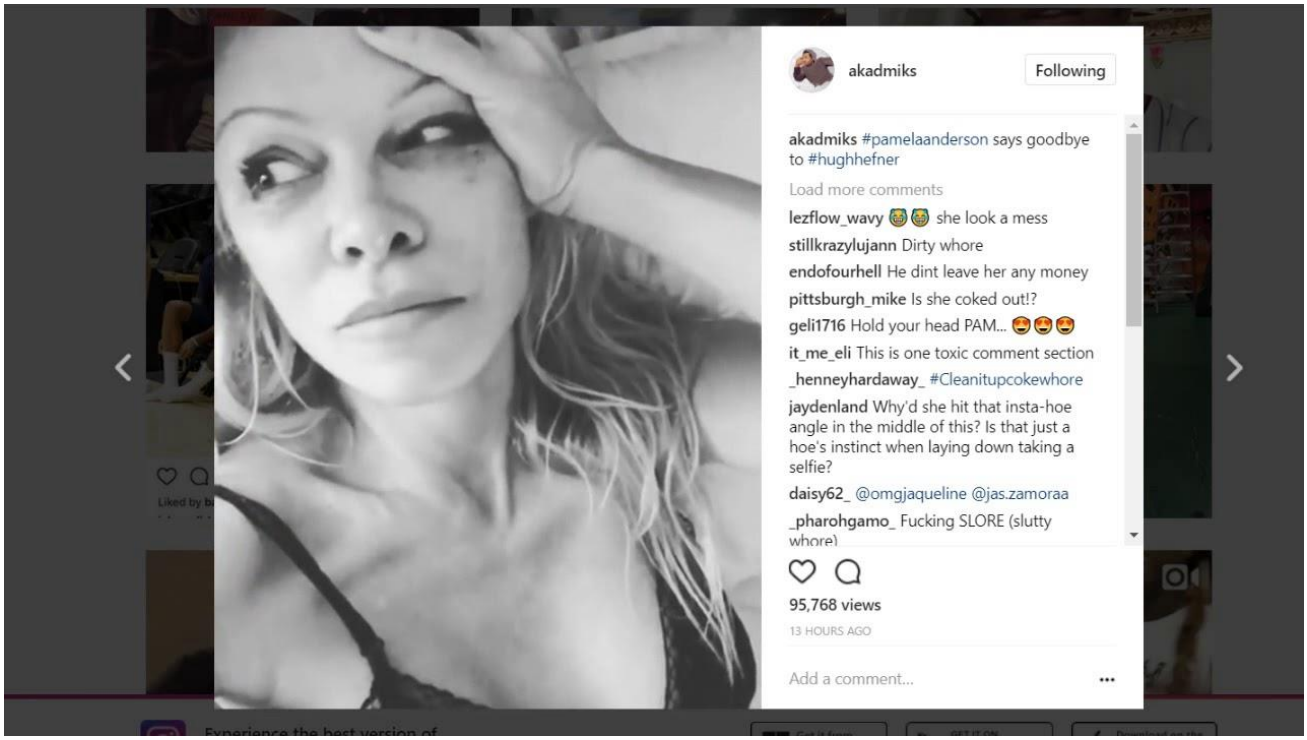
Example: Shut up shit face!

This inturn creates sadness, lack of confidence decrease in the person.

We just saw how 2 words can make a huge difference.

So it all depends on the words and their vibe associated with it and person receiving it.

We have also come across a lot of trolls on twitter going on, which in turn may push the people to a sense of depression.



Celebrity Pamela Anderson who got trolled for sharing her condolenses in Hugh Hefner Passing.

There are many such cases occuring each day.

We here build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate based on dataset of comments from Wikipedia's talk page edits.

2.0 Data used

As provided, the data is already pre split into train, test and sample submission. The train data contain's 8 variables and 159571 observations. The test data contain's 2 variables and 15 3164 observations which excludes our targets and sample submission which includes our target s containing sample values of targets.

3.0 Variables used

1. **Comment_text -** consists of wikipedia comments
2. **Toxic**
3. **severe_toxic**
4. **obscene**
5. **threat**
6. **insult**
7. **identity_hate**

4.0 Data Preprocessing:

Our train and test datasets were loaded and checked for the following,

- **Structure**
- **Summary**
- **Missing values**

After getting the data altogether it has dimension of 8 variables and 1482535 observations. Out of 8 variable's 2 were character(id, comment_text), the rest were integers.

There were found to be no missing values.

This entire analysis is divided into 2 parts:

PART 1: Initial level analysis and EDA

PART 2: Data modeling and prediction

So let's get started,

5.0 Data Conditioning and analysis

This is Interesting as the main focus will be here on the comment_text feature which holds the toxic comments beneath it.

So first let's try to explore what it has based on which we'll be making the predictions for the other 6 target's such as toxic, severe_toxic, obscene, threat, insult, identity_hate making it a multi-label classification. Analysis of target variable

Since this is text data, data conditioning plays a major role,

The steps followed are:

1. **Remove contractions**
2. **Remove special character's**
3. **Convert them to lower**
4. **Lemmatize the text/strings # we'll go over this later in part 2**
5. **Remove number's**
6. **Remove punctuation's**
7. **Remove stop words**
8. **Strip white spaces distribution of the variable.**

The above steps were performed for the data and a Corpus was created.

A corpus is nothing but a large, structured set of texts.

The first comment was inspected.

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

 [1] explanation edits made username hardcore metallica fan reverted weren
t vandalisms just closure gas voted new york dolls fac please dont remove
template talk page since im retired now
```

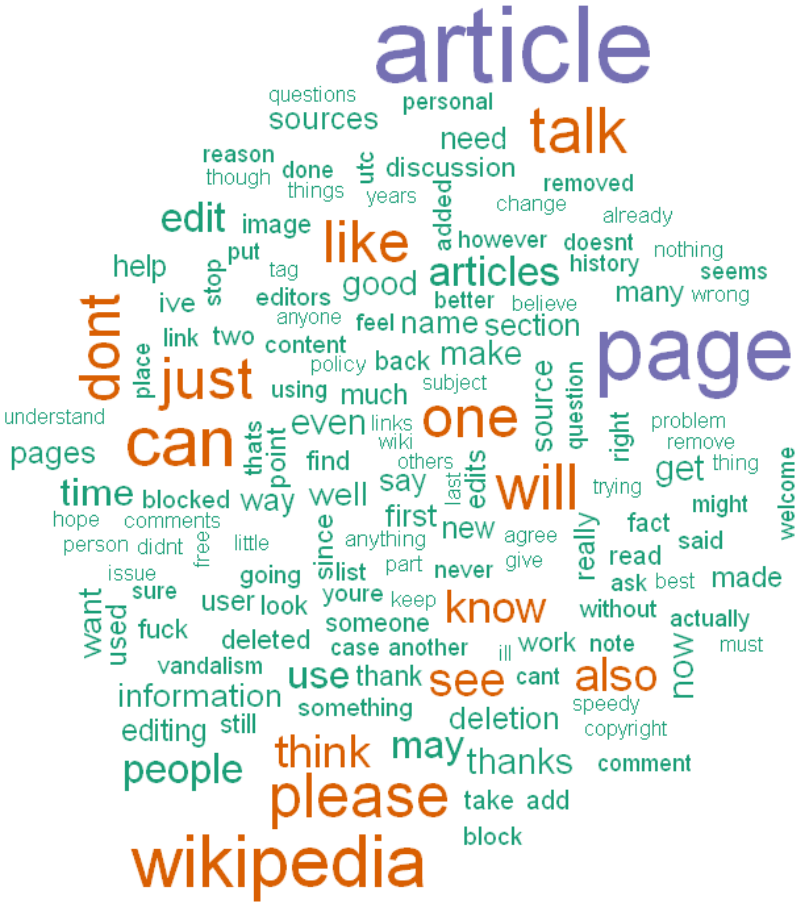
To analyze the textual data, we use a Document-Term Matrix (DTM) representation: documents as the rows, terms/words as the columns, frequency of the term in the document as the entries because the number of unique words in the corpus the dimension can be large.

To reduce the dimension of the DTM, we can remove the less frequent terms such that the sparsity is less than 0.99 and the let's inspect.

Now let's inspect the same:

```
 <<DocumentTermMatrix (documents: 1, terms: 20)>>
Non-/sparse entries: 13/7
Sparsity           : 35%
Maximal term length: 8
Weighting          : term frequency (tf)
Sample             :
    Terms
Docs dont edits just made new now page please remove reverted
   1    1     1    1    1   1   1    1      1      1        1
```

Wordcloud was plotted for the above case to explore the important words.



One may think that in the wordcloud, words such as article, wikipedia, do not carry too much meaning in the setting, since we know that the entire corpus is about comments. Therefore sometimes it is necessary to use the tf–idf(term frequency–inverse document frequency) instead of the frequencies of the term as entries, tf-idf measures the relative importance of a word to a document.

Let's get beneath into the text data using EDA.

6.0 Exploratory Data Analysis for Text

Here we'll be using the tidy_text() package to unlock the textdata. Using tidy data principles is a powerful way to make handling data easier and more effective, and this is no less true when it comes to dealing with text. Tidy data has a specific structure:

Each variable is a column

Each observation is a row

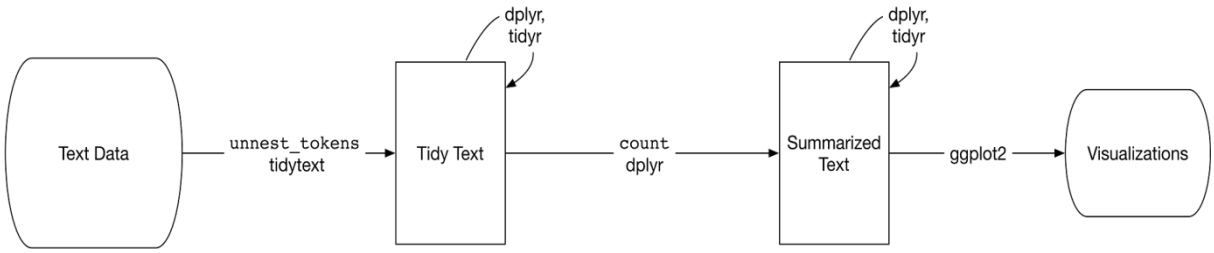
Each type of observational unit is a table

We thus define the tidy text format as being a table with one-token-per-row. A token is a meaningful unit of text, such as a word, that we are interested in using for analysis, and

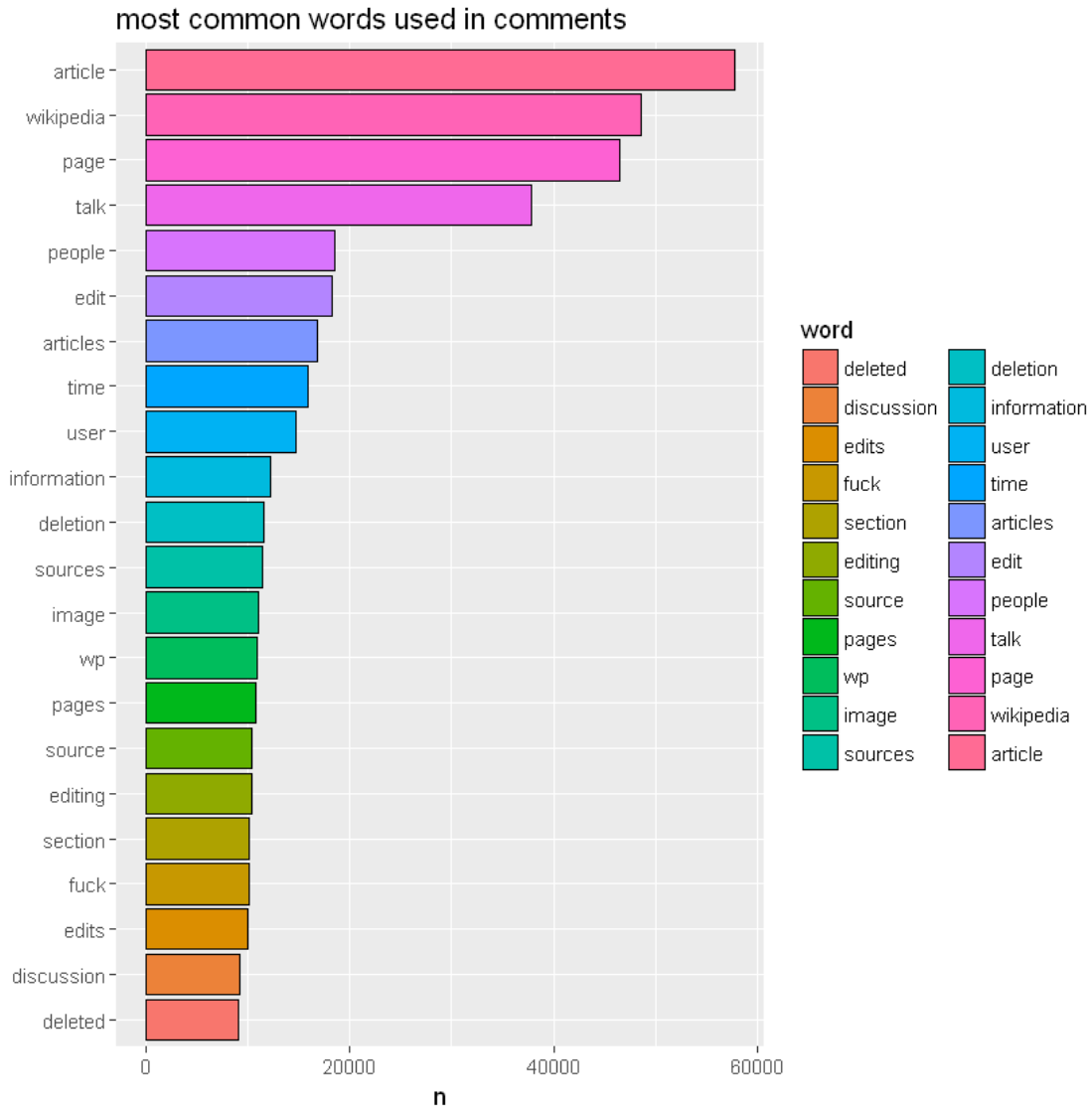
tokenization is the process of splitting text into tokens. This one-token-per-row structure is in contrast to the ways text is often stored in current analyses, perhaps as strings or in a document-term matrix. For tidy text mining, the token that is stored in each row is most often a single word, but can also be an n-gram, sentence, or paragraph. In the tidytext package, we provide functionality to tokenize by commonly used units of text like these and convert to a one-term-per-row format.

Within our tidy text framework, we need to both break the text into individual tokens (a process called tokenization) and transform it to a tidy data structure. To do this, we use tidytext's unnest_tokens() function.

An overview of the process:



Since tokenization is done now let's plot the most common words used.



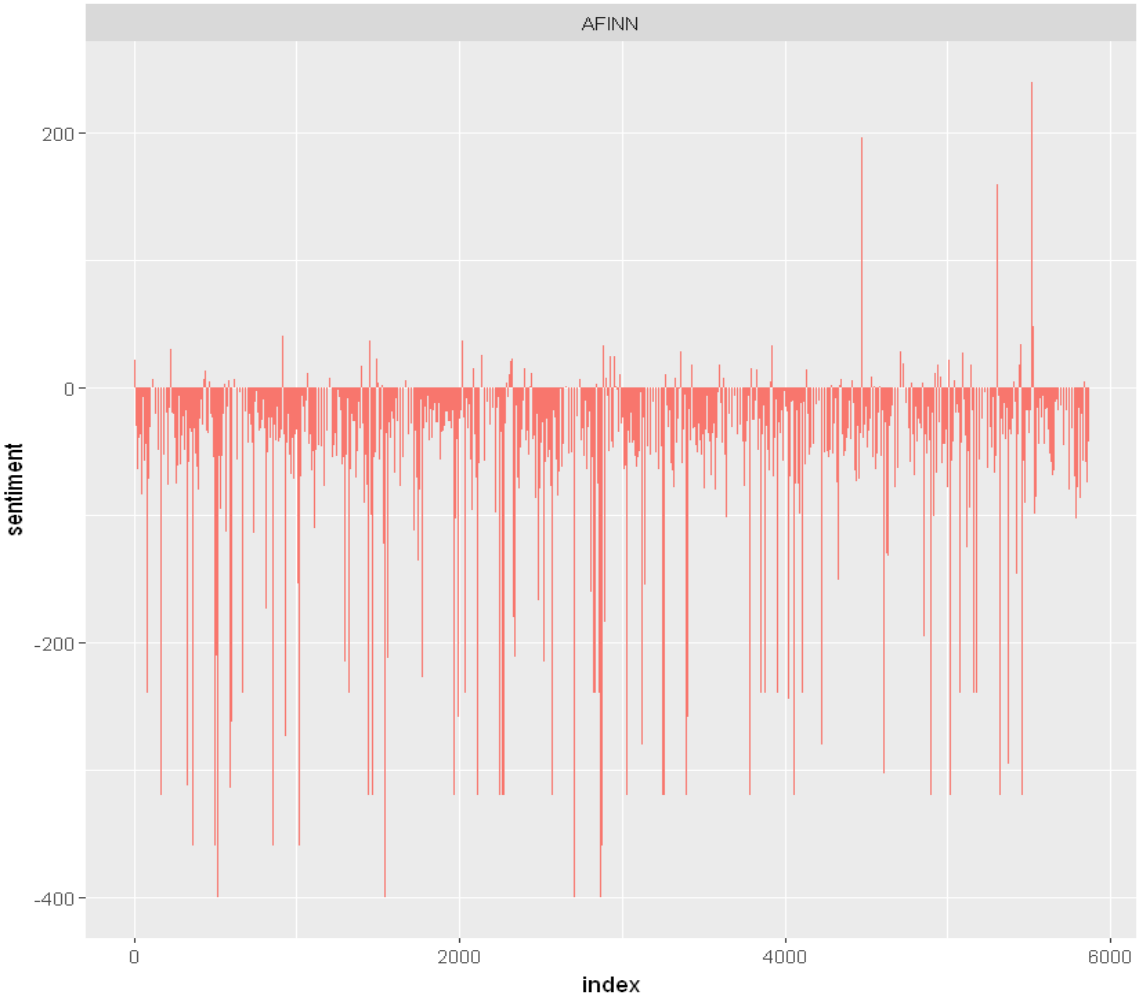
6.1 Sentiment Analysis

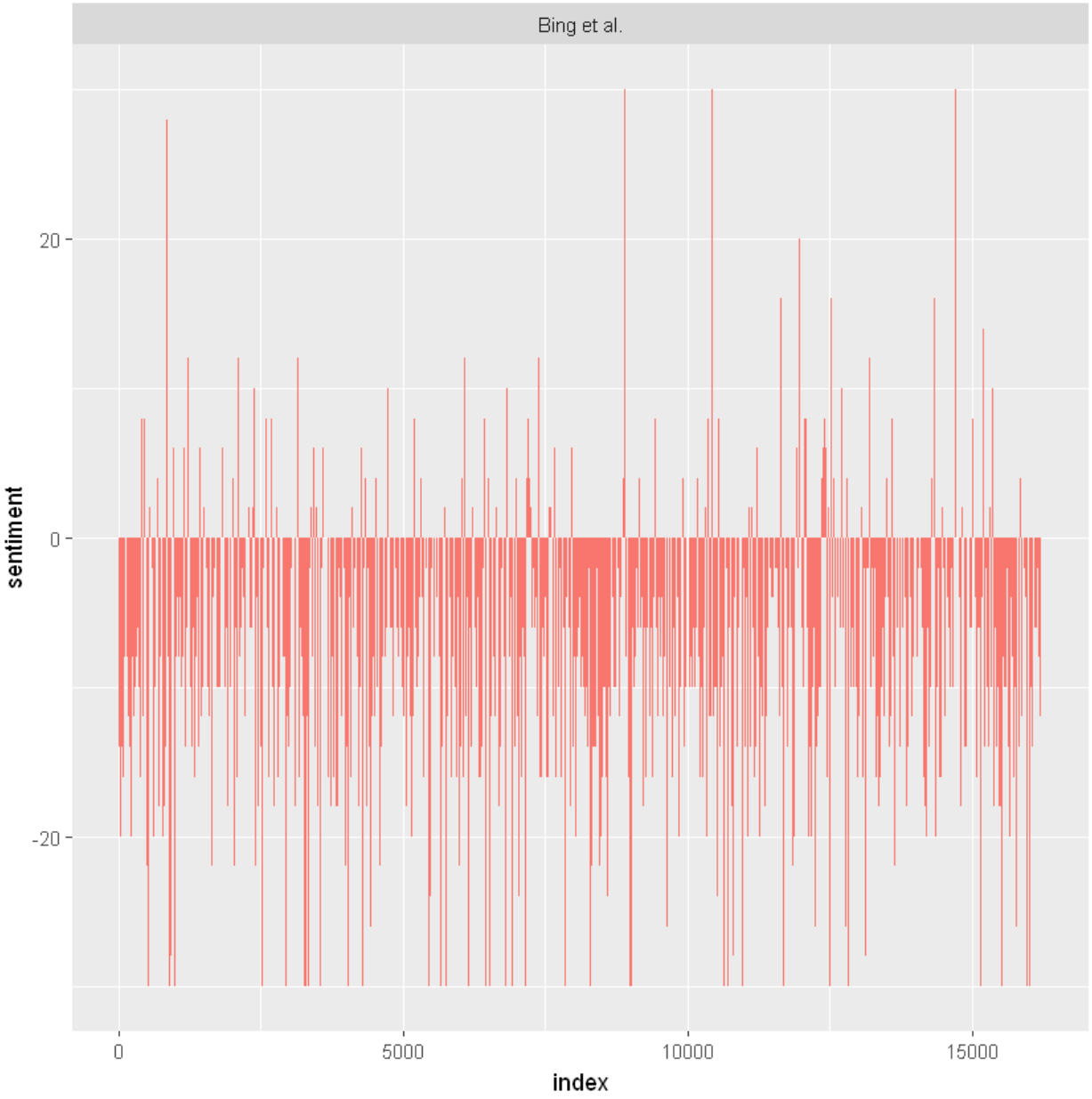
Understanding of the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterized by some other more nuanced emotion like surprise or disgust plays a key in ananlysis of text.

Let's analyze the sentiments using 2 popular lexicons: bing and afinn.

Bing lexicon categorizes words in a binary fashion into positive and negative categories. The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment. All of this

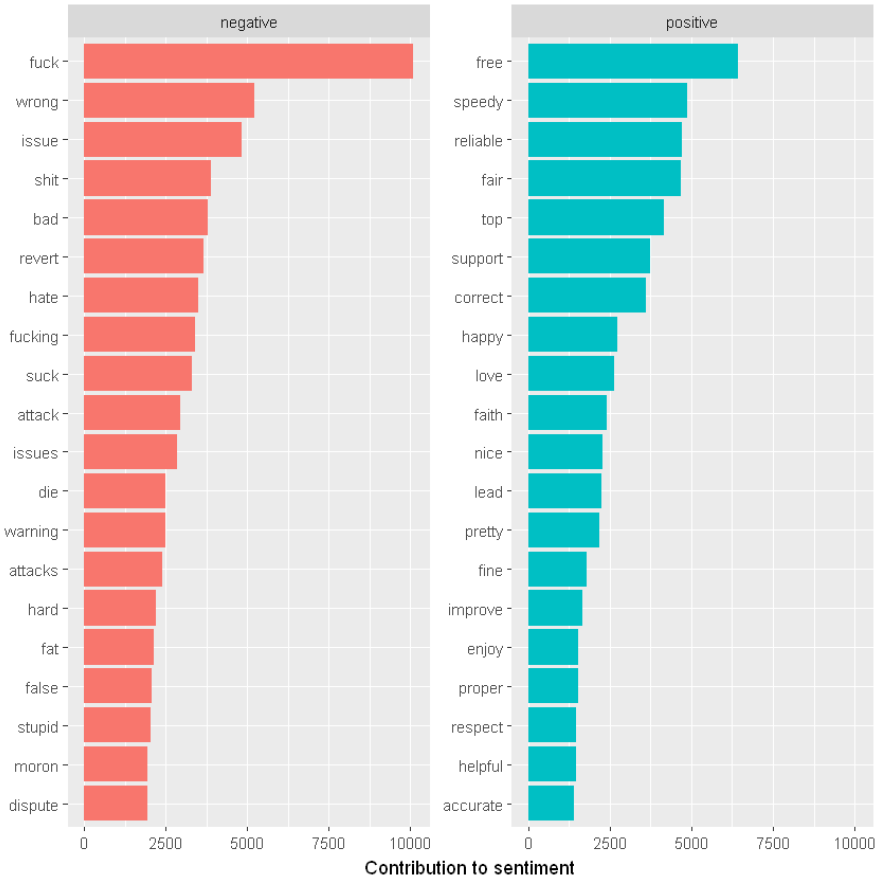
information is tabulated in the sentiments dataset, and tidytext provides a function get_sentiments() to get specific sentiment lexicons without the columns that are not used in that lexicon.





Interesting to see the ratio of negative is more than positive, this will be crucial for our ananlysis.

Visualization of the most common positive and negative words and their contribution towards sentiment.



6.2 TF- IDF

The statistic tf-idf is intended to measure how important a word is to a document in a collection (or corpus) of documents, for example, to one novel in a collection of novels or to one website in a collection of websites.

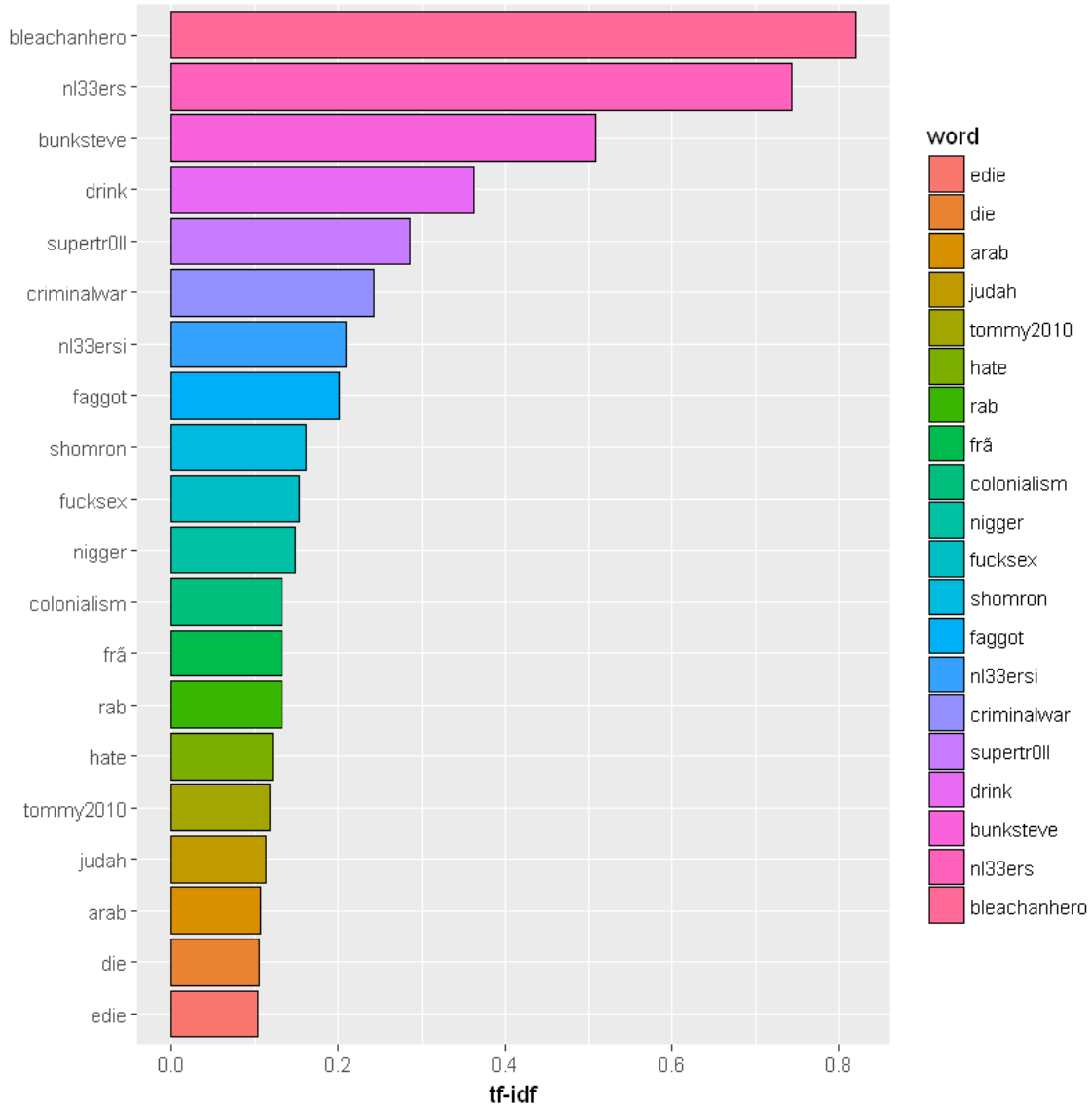
One measure of how important a word may be is its term frequency (tf), how frequently a word occurs in a document.

Inverse document frequency (idf), decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. This can be combined with term frequency to calculate a term's tf-idf (the two quantities multiplied together), the frequency of a term adjusted for how rarely it is used.

The inverse document frequency for any given term is defined as

$$idf(\text{term}) = \ln\left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}}\right)$$

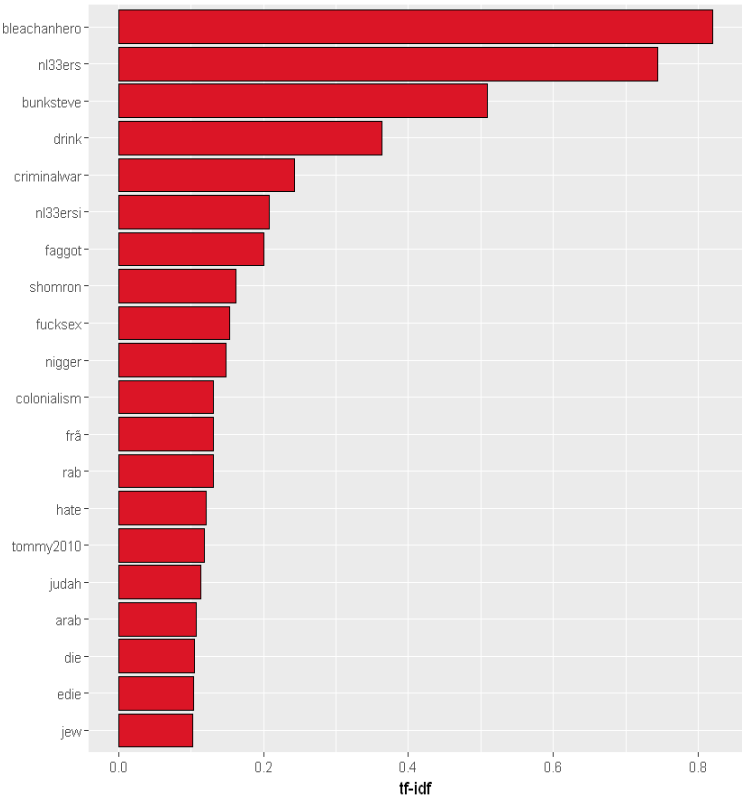
list of 20 most important words:



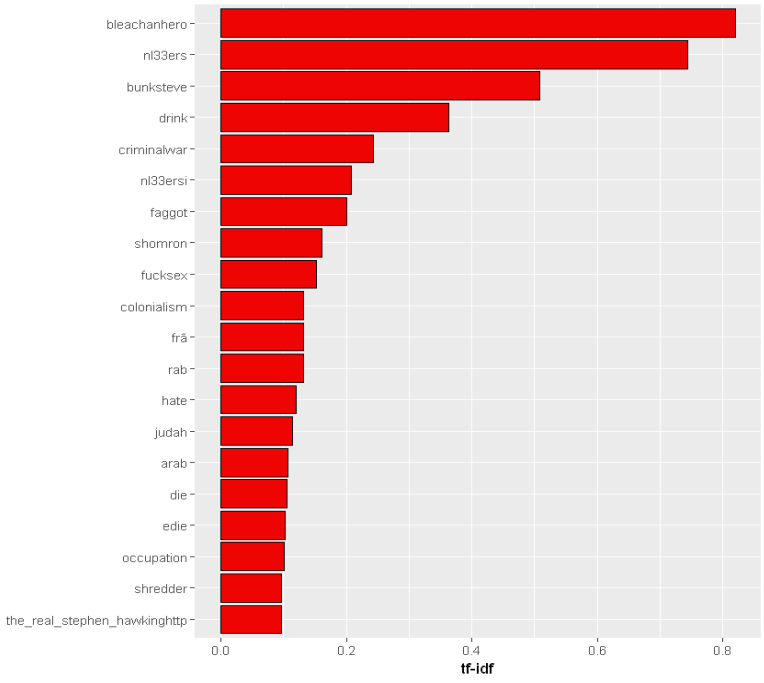
We now are able to draw the most important words and clearly it makes sense to have them(toxic words) in our list.

Let's look into them in detail by each target variable.

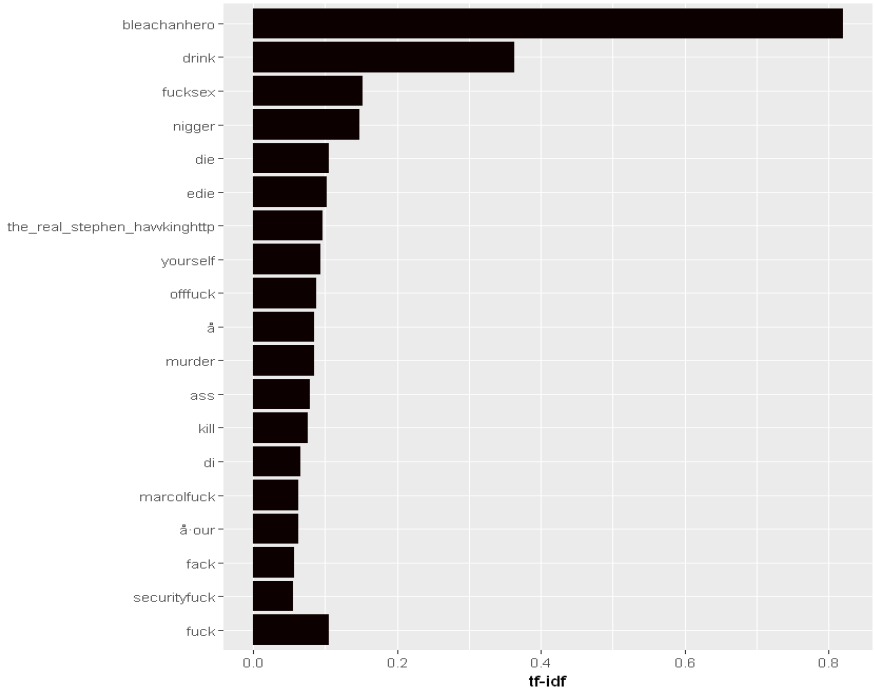
6.2.1 Important words in Toxic



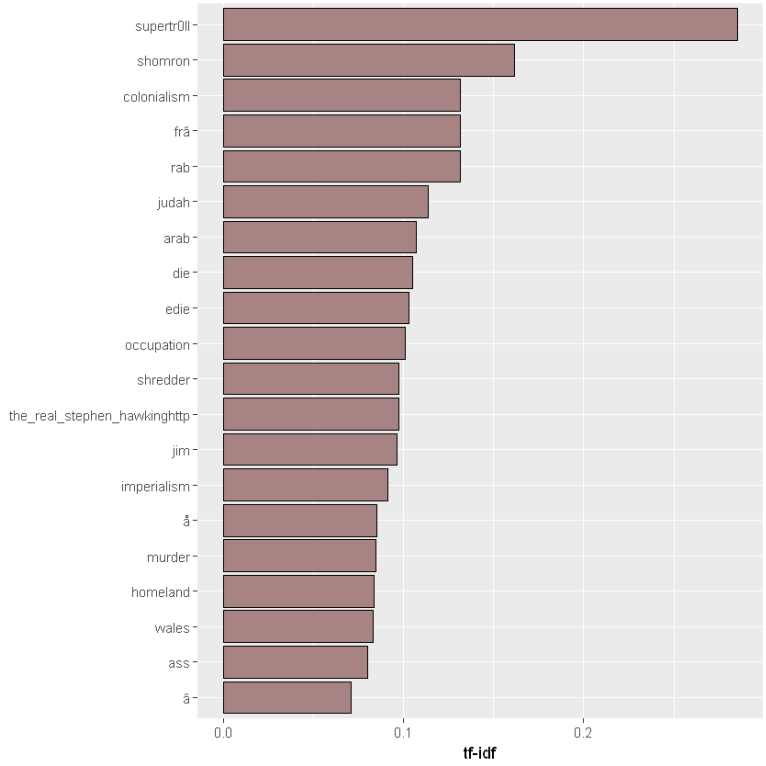
6.2.2 Important words in Severe_toxic



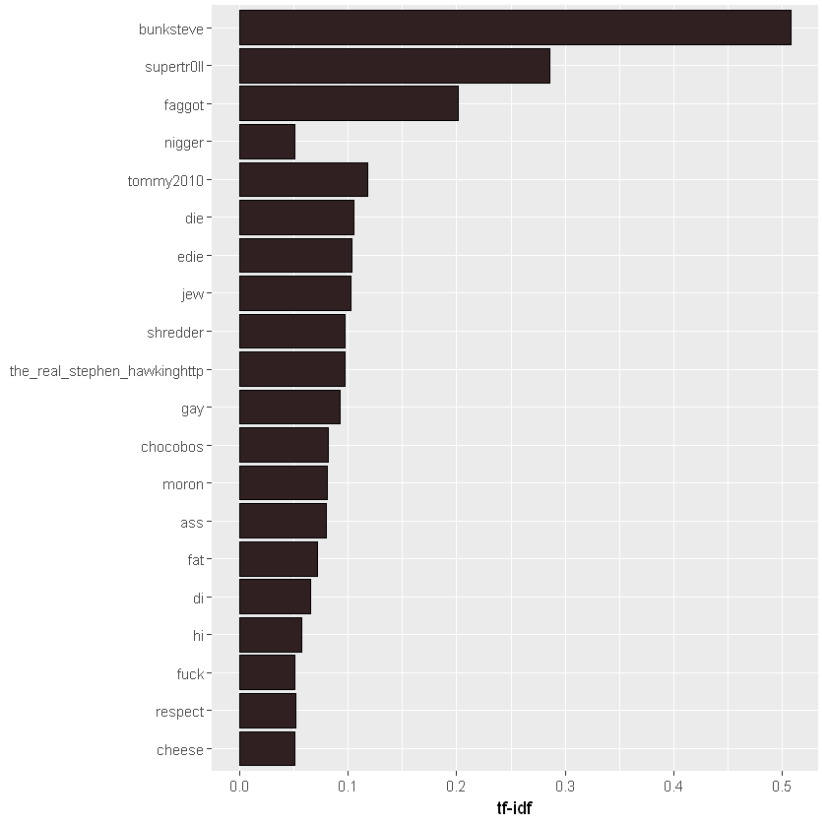
6.2.3 Important words in Obscene



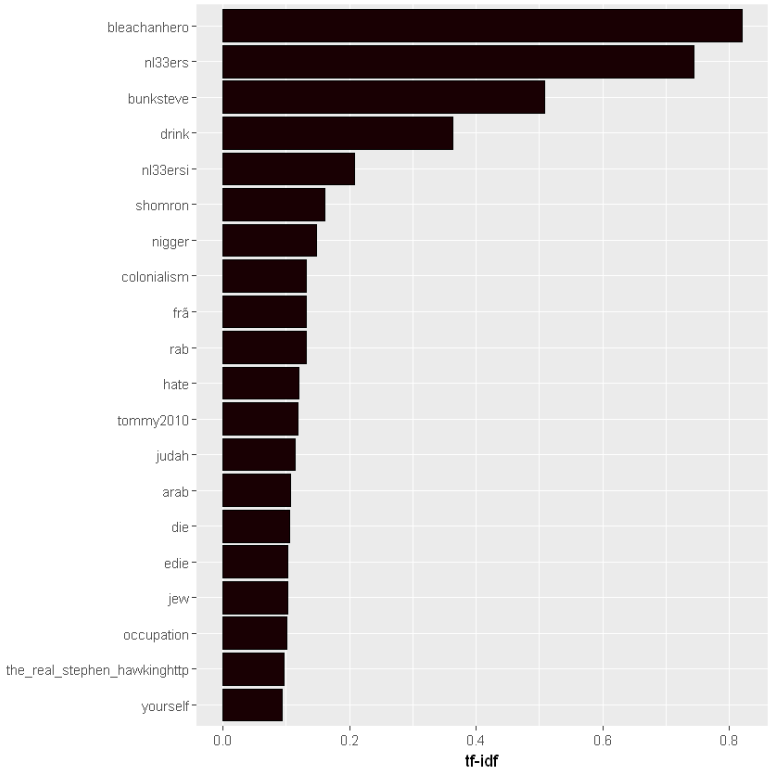
6.2.4 Important words in Threat



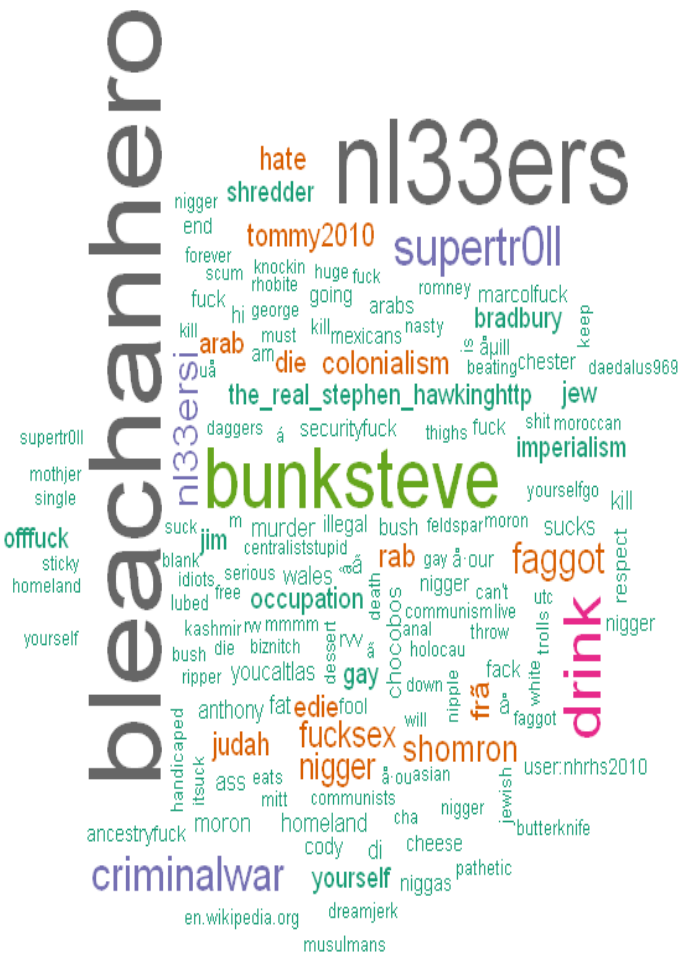
6.2.5 Important words in Insult



6.2.6 Imporant words in Identity hate



Wordcloud for the above:



Now since we have done our EDA, we now have a clear idea on the top features contributing to the targets, in our next section i,e Part – 2 we'll see the extraction of these through feature engineering.

TOXIC COMMENT CLASSIFICATION PART 2:

Further analysis will be done by binding the complete dataset for text cleaning and create a document term matrix and later split into train and test. Train further will be split into train and validation.

We follow the standard process for text cleaning along with lemmatization.

As described in part1, now to proceed further we lemmatize the strings.

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

7.0 Feature Engineering

From part 1 we extracted wordclouds, most used words, sentiment analysis and TD-IDF where we we were able to find the most crucial words. Now we make use of them and select 59 new key features and extract them.

We then bind these features with the cleaned text from DTM as matrix and then into a dataframe.

8.0 Model training for data

In order to build, validate and check for accuracy of our model, we split the train data set further into train and validation in the ratio of 60:40 where 60% is to training and 40% is to validate.

8.1 Decision tree model

Model was built on training data and tested on validation data. Confusion matrix, ROC, AUC, Accuracy has been checked.

The model is 93 % accurate.

8.2 Logistic regression model

Using the generalized linear model, glm() function, we make a logistic regression analysis using 'toxic' feature as outcome, with the rest of features in the training dataset as independent predictors. Specified binomial(link = 'logit') in the family argument will analyze the data using logistic regression.

The model was built on training data and the summary looks like:

```
Call:
glm(formula = toxic ~ ., family = binomial(link = "logit"), data = subset
(tr,
    select = -c(1, 3, 4, 5, 6, 7)), maxit = 100)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-5.0047  -0.3638  -0.2857  -0.1581   8.4904

Coefficients:
             Estimate Std. Error  z value Pr(>|z|)
(Intercept) -2.682132   0.019244 -139.376  < 2e-16 ***
fuck         5.485864   0.110677   49.566  < 2e-16 ***
self        -0.017338   0.069459   -0.250 0.802887
your         0.872398   0.037089   23.522  < 2e-16 ***
shit         3.642637   0.094845   38.406  < 2e-16 ***
hate         1.257445   0.080505   15.620  < 2e-16 ***
hell         0.388729   0.067363    5.771 7.89e-09 ***
fat          0.764887   0.126261    6.058 1.38e-09 ***
shut         2.409232   0.147106   16.377  < 2e-16 ***
attack       0.533755   0.092665    5.760 8.41e-09 ***
stupid       3.196758   0.092605   34.521  < 2e-16 ***
moron        2.844653   0.155298   18.317  < 2e-16 ***
dispute     -1.371020   0.269228   -5.092 3.54e-07 ***
suck         3.879706   0.132596   29.260  < 2e-16 ***
cock         1.978958   0.205021    9.652  < 2e-16 ***
dick         3.274563   0.137685   23.783  < 2e-16 ***
bitch        4.027549   0.184260   21.858  < 2e-16 ***
free        -0.388078   0.121507   -3.194 0.001404 **
reliable    -0.150703   0.179284   -0.841 0.400581
fair        -0.840980   0.174777   -4.812 1.50e-06 ***
support     -0.480672   0.126175   -3.810 0.000139 ***
correct     -0.706358   0.126327   -5.591 2.25e-08 ***
happy       -0.473235   0.174568   -2.711 0.006710 **
love         0.315190   0.107386    2.935 0.003334 **
```

```
pretty       -0.733248   0.196695   -3.728 0.000193 ***
enjoy         0.276801   0.180045    1.537 0.124195
speed        -0.896858   0.291465   -3.077 0.002090 **
respect      -0.589072   0.188477   -3.125 0.001775 **
accurate     -0.334236   0.207187   -1.613 0.106698
help         -0.649398   0.106403   -6.103 1.04e-09 ***
fine         -0.859842   0.168907   -5.091 3.57e-07 ***
faith        -0.634444   0.209975   -3.022 0.002515 **
lead         -1.318814   0.197817   -6.667 2.61e-11 ***
improve      -0.458587   0.208629   -2.198 0.027942 *
bleach       -8.908755  91.754918   -0.097 0.922653
hero          0.231507   0.283941    0.815 0.414880
sex           1.290943   0.113419   11.382  < 2e-16 ***
nigger        3.348088   0.269145   12.440  < 2e-16 ***
judah        -8.717276 205.020317   -0.043 0.966085
criminal      0.636328   0.290616    2.190 0.028554 *
war          -0.251441   0.064422   -3.903 9.50e-05 ***
faggot        4.382692   0.267809   16.365  < 2e-16 ***
drink         0.528592   0.270634    1.953 0.050800 .
die           0.753142   0.106038    7.103 1.22e-12 ***
bunk         -0.510041   0.788245   -0.647 0.517594
troll         1.105901   0.131948    8.381  < 2e-16 ***
super        -0.362816   0.224354   -1.617 0.105844
rab          -0.248680   0.200791   -1.239 0.215529
ass           0.745406   0.048967   15.223  < 2e-16 ***
gay           2.460873   0.114535   21.486  < 2e-16 ***
murder        0.597831   0.205238    2.913 0.003581 **
nl           -0.479466   0.062163   -7.713 1.23e-14 ***
colonial      0.044706   0.647321    0.069 0.944939
occupation   -0.645865   0.621916   -1.039 0.299033
arab          0.238654   0.298136    0.800 0.423429
jew           0.569247   0.135840    4.191 2.78e-05 ***
ers          -0.411936   0.054267   -7.591 3.18e-14 ***
kill          0.980140   0.109256    8.971  < 2e-16 ***
moth          0.654122   0.173210    3.776 0.000159 ***
jerk          3.109311   0.194198   16.011  < 2e-16 ***
edit         -0.013106   0.023942   -0.547 0.584103
make         -0.248398   0.040477   -6.137 8.42e-10 ***
page         -0.148808   0.031933   -4.660 3.16e-06 ***
please       -0.644684   0.057933  -11.128  < 2e-16 ***
remove        0.052759   0.046217    1.142 0.253640
talk         -0.173819   0.040075   -4.337 1.44e-05 ***
thank        -0.847279   0.066633  -12.716  < 2e-16 ***
article      -0.556870   0.034214  -16.276  < 2e-16 ***
can          -0.009860   0.009085   -1.085 0.277775
good         -0.321516   0.039385   -8.163 3.26e-16 ***
one          -0.397121   0.045134   -8.799  < 2e-16 ***
think        -0.271041   0.045400   -5.970 2.37e-09 ***
wikipedia     0.002406   0.004969    0.484 0.628224
work         -0.107660   0.046310   -2.325 0.020085 *
like          0.333788   0.032535   10.259  < 2e-16 ***
conjurer     -0.627922   0.067882   -9.250  < 2e-16 ***
source       -0.508125   0.067327   -7.547 4.45e-14 ***
time         -0.143746   0.044858   -3.204 0.001353 **
people        0.166431   0.035907    4.635 3.57e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 60771  on 95741  degrees of freedom
Residual deviance: 36973  on 95663  degrees of freedom
AIC: 37131

Number of Fisher Scoring iterations: 11
```

The output of the logistic regression object, toxic.glm , shows that the features such as 'self','reliable','enjoy','bleach','hero','judah','bunk','colonial','occupation','arab','jew','edi t','remove' and 'can' are insignificant features for predicitng toxic outcome. The rest of the model coefficients suggests significant contribution in toxic prediction. For example, increase one unit in 'conjurer' will decrease the log odd of toxic by 0.62;'source' will decrease the log odd of toxic by 0.5; and being in 'dispute' will decrease the log odd of toxic by 1.3 and so on.

Furthermore, using the pR2() function in the pscl package allows to see a linear regression R-square value equivalent, which is the McFadden R-square index. This is equivalently saying that the logsitic regresion model has well explained 39% of variation in the toxic comment prediction.

McFadden
0.391601504763594

8.3 Prediction and model performance evaluation for logistic regression model

Model was tested on validation data. Confusion matrix, ROC, AUC, Accuracy has been checked.

The model is 94 % accurate.

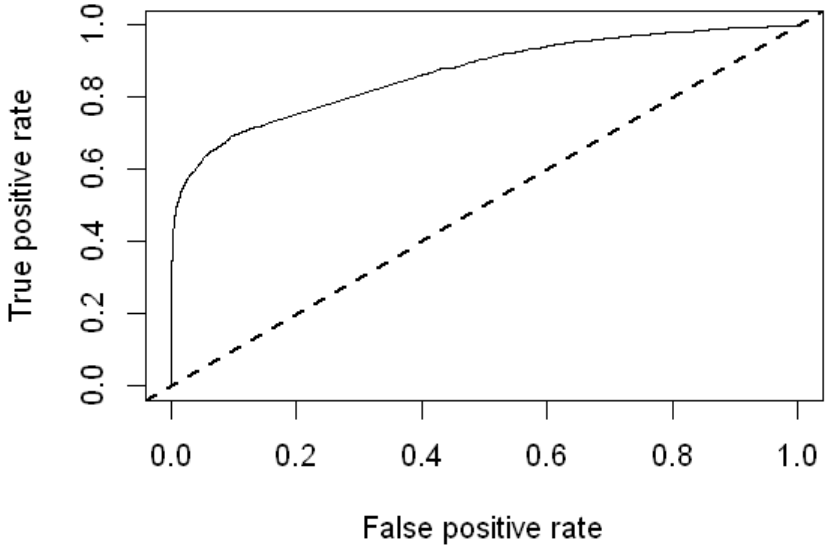
Threshold was set at 0.5.

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 57353  3181
         1   423  2872

         Accuracy : 0.9435
         95% CI : (0.9417, 0.9453)
```

```
    No Information Rate : 0.9052
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5868
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9927
            Specificity : 0.4745
         Pos Pred Value : 0.9475
         Neg Pred Value : 0.8716
             Prevalence : 0.9052
         Detection Rate : 0.8985
   Detection Prevalence : 0.9484
      Balanced Accuracy : 0.7336

       'Positive' Class : 0
```

The Receiver Operating Characteristic (ROC) curve is plotted below for false positive rate (FPR) in the x-axis vs. the true positive rate (TPR) in the y-axis. It shows the detection of true positive while avoiding the false positive. This is the same as measuring the unspecificity (1 - specificity) in x-axis, against the sensitivity in y-axis.

This ROC curve in particular shows that its very closed to the perfect classifier meaning that its better at identifying the positive values. An index for that is the AUC (area under the curve) of this ROC, which is 0.86 for this case.




```
Formal class 'performance' [package "ROCR"] with 6 slots
  ..@ x.name      : chr "None"
```

```
..@ y.name      : chr "Area under the ROC curve"
..@ alpha.name  : chr "none"
..@ x.values    : list()
..@ y.values    :List of 1
.. ..$ : num 0.866
..@ alpha.values: list()
```

The value of AUC is : 0.866060692169774

But wait it's not over yet, this is what makes this challenge much more interesting, we have got to still classify the text into 5 more labels i,e multi-label classification. One way is to make predictions for each one individually as above and then calculate the Accuracy. That's a little tedious isn't it.

That's why R has introduced the mlr package through which we can achieve the output here for the multi-label classification.

Here two different approaches exist for multilabel classification.

1. **Problem transformation methods try to transform the multilabel classification into binary or multiclass classification problems.**

2. **Algorithm adaptation methods adapt multiclass algorithms so they can be applied directly to the problem.**

9.0 Final prediction using mlr

1. **The first thing we have to do for multilabel classification in mlr is to get our data in the right format. We need a data.frame which consists of the features and a logical vector for each label which indicates if the label is present in the observation or not.**

   One way we can simpy do this is by passing a condition as below, which directly converts into 'logical' format i,e if > 0 then 'TRUE' else 'FALSE'.

   We have to make sure before passing the condition that the variable consists only of binary(e.g 0 or 1).

2. **We create train and test labels and then the next step is to create a MultilabelTask like a normal ClassifTask. Instead of one target name we can specify a vector of targets which correspond to the names of logical variables in the data.frame.**

3. **The next step is to Construct a learner**

As mentioned above Multilabel classification in mlr can currently be done in two ways:

Algorithm adaptation methods: Treat the whole problem with a specific algorithm.

Problem transformation methods: Transform the problem, so that simple binary classification algorithms can be applied.

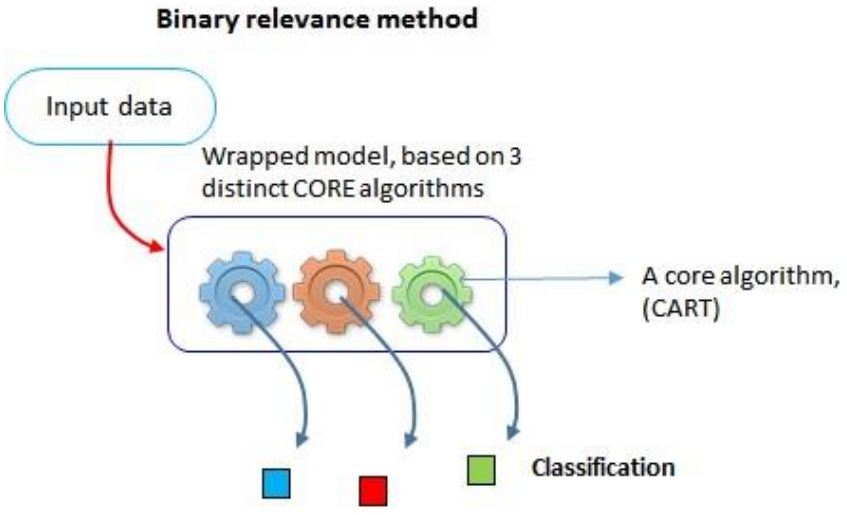
We'll be using the problem transformation method here and it in turn has the following methods:

Binary relevance
Classifier chains
Nested stacking
Dependent binary relevance
Stacking

We shall use the Binary relevance method and the classif.rpart i,e a decision tree method .

For generating a wrapped multilabel learner we first create a binary (or multiclass) classification learner with makeLearner. Afterwards apply a function like makeMultilabelBinaryRelevanceWrapper, makeMultilabelClassifierChainsWrapper, makeMultilabelNestedStackingWrapper, makeMultilabelDBRWrapper or makeMultilabelStackingWrapper on the learner to convert it to a learner that uses the respective problem transformation method.

The binary relevance problem transformation method consists of converting the multilabel problem to binary classification problems for each label and applies a simple binary classificator on these. By using this method, we assume that our labels are independent (i.e each one will be treated as a single, stand-alone target outcome, and will be classified without considering the remaining labels).



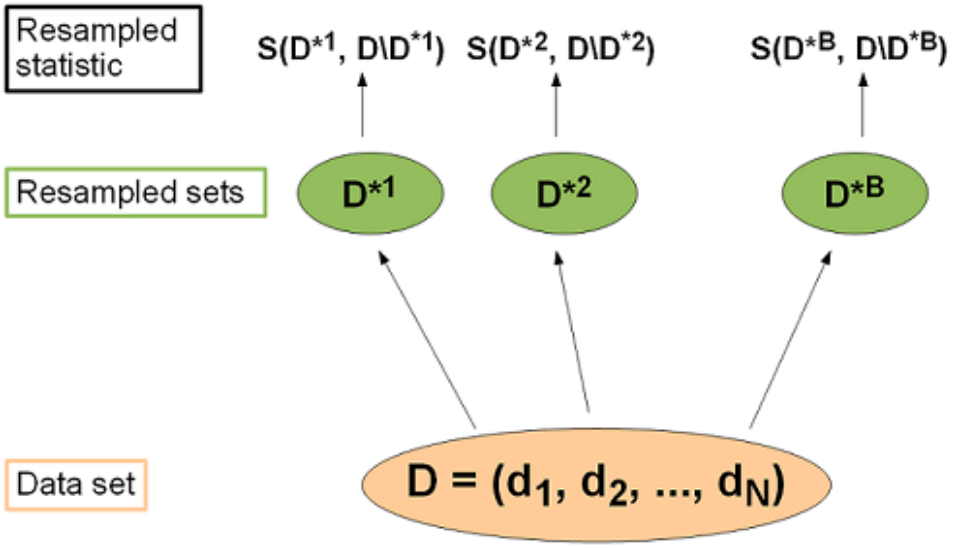
4. We train a model as usual with a multilabel learner and a multilabel task as input.

5.Prediction can be done as usual in mlr with predict and by passing a trained model and either the task to the task argument or some new data to the newdata argument.

6. For evaluating the overall performance of the learning algorithm we can do some resampling.

Resampling strategies are usually used to assess the performance of a learning algorithm: The entire data set is (repeatedly) split into training sets D∗b and test sets D\D∗b, b=1,...,B. The learner is trained on each training set, predictions are made on the corresponding test set (sometimes on the training set as well) and the performance measure S(D∗b,D\D∗b) is calculated. Then the B individual performance values are aggregated, most often by calculating the mean. There exist various different resampling strategies, for example cross-validation and bootstrap, to mention just two popular approaches.

Resampling Figure:
As usual we have to define a resampling strategy, either via makeResampleDesc or makeResampleInstance. After that we can run the resample function.



Model Evaluation was done on the validation data- Accuracy, MMCE and AUC were calculated as shown below.

	acc.test.mean	mmce.test.mean	auc.test.mean
toxic	0.9395886	0.060411412	0.7187400
severe_toxic	0.9901769	0.009823121	0.4608112
obscene	0.9745100	0.025489981	0.8257055
threat	0.9970076	0.002992370	0.4754887
insult	0.9642012	0.035798775	0.7409923
identity_hate	0.9918062	0.008193768	0.5931970

The avg model accuracy was found to be 97%.

All right so we have finally validated our validation set and it looks good, hence let's freeze our model here and proceed to pass on our actual test set and predict the values.

Conclusion: The model was predicted for the actual test data set and the set of comments toxicity classified into 6 different categories has been provided.

_____THANK YOU_____