Classification of news headlines into clickbait or non-clickbait

Ruben Rodriguez de la Fuente

February 20, 2022

Introduction

Clickbait is a sensationalized headline that encourages you to click on a link. Instead of presenting objective facts, clickbait headlines often appeal to your emotions and curiosity. The website hosting the link earns revenue from advertisers thanks to clicks, but the actual content is usually of questionable quality and accuracy, so people's time and attention will be wasterd.

Moreover, clickbait can be harmful when it's used along with the creation of fake news. The outrageous fake headlines appeal to the emotions of readers, who then spread the content on social media, often without reading the actual article.

The data is collected from various news sites.The clickbait headlines are collected from sites such as 'BuzzFeed', 'Upworthy', 'ViralNova', 'Thatscoop', 'Scoopwhoop' and 'ViralStories'. The relevant or non-clickbait headlines are collected from many trustworthy news sites such as 'WikiNews', 'New York Times', 'The Guardian', and 'The Hindu'.

Methods and analysis

We have chosen logistic regression as it is known to work well with text problems. We will try it first on basic linguistic features (such as length of headlines or the presence of question, exclamation marks or numbers, which are typical of clickbait). We will then explore differences in vocabulary between both categories and use string vectorization techniques to exploit these differences, including bag-of-words and tf-idf.

Libraries for analysis

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
```

```
##
##     lift
```

```
library(stringr)
library(text2vec)
```

```
## Warning: package 'text2vec' was built under R version 4.1.2
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.1.2
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-3
```

```
library(ggplot2)
library(tidytext)
```

```
## Warning: package 'tidytext' was built under R version 4.1.2
```

Data loading and exploration

We load the data and check there is no prevalence of one class over the other.

```
df<-read.csv(file='clickbait_data.csv')#load data into a dataframe
df %>% group_by(clickbait) %>% summarise(n=n())#check if data is balanced between both categories
```

```
## # A tibble: 2 x 2
##   clickbait     n
##       <int> <int>
## 1         0 16001
## 2         1 15999
```

```
df<-df %>% mutate(clickbait = factor(clickbait))#convert dependent variable to factor
```

Feature extraction

Here we extract basic features like length and presence of numbers and question or exclamation marks, using stringr package.

```
df$length=str_length(df$headline)#headline length in characters
df$questionmark=str_detect(df$headline,'\\?')#checking for question mark; ? is escaped with \\ as str_d
df$exclamationmark=str_detect(df$headline,'!')#checking for exclamation mark
df$numbers=str_detect(df$headline,'[0-9]')#checking for numbers
```

Train-test split

We split into train and test set. As we will use cross-validation during training, we haven't considered necessary to create also a validation set.

```
#splitting dataframe into train and test
set.seed(42, sample.kind="Rounding")
```

```
## Warning in set.seed(42, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
y<-df$clickbait
test_index <- createDataPartition(y, times = 1, p = 0.2, list = FALSE)
test_set<-df[test_index,]
train_set<-df[-test_index,]
```

Random guessing

We set up a random guessing baseline. Non-surprinsingly, as there is no class imbalance, it predicts with ~50% accuracy.

```
#random guessing baseline
set.seed(3, sample.kind="Rounding")
```

```
## Warning in set.seed(3, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
outcome<-factor(c(0,1))
y_random<-sample(outcome, size = 6401, replace=TRUE)
mean(test_set$clickbait==y_random)
```

```
## [1] 0.5178878
```

Logistic regression on basic features

We try logistic regression with our simple features, using caret package. Accuracy increases to 64%, which is a significant increase from the 51% random guessing baseline. We also check for variable importance with varImp function.The presence of numbers and the length are the strongest predictors. Question and exclamation marks seem to be negligible.

```
#predicting with glm; using 3-fold cross-validation and training percentage 90%
set.seed(8, sample.kind="Rounding")
```

```
## Warning in set.seed(8, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
fit_glm<-train(clickbait ~ length + questionmark + exclamationmark + numbers, data=train_set, method =
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
y_predict<-predict(fit_glm, test_set, type = "raw")
confusionMatrix(y_predict, test_set$clickbait)$overall['Accuracy']
```

```
##  Accuracy
## 0.6445868
```

```
#check variable importance
varImp(fit_glm)
```

```
## glm variable importance
##
##                     Overall
## numbersTRUE         100.000
## length               45.748
## questionmarkTRUE      2.432
## exclamationmarkTRUE   0.000
```

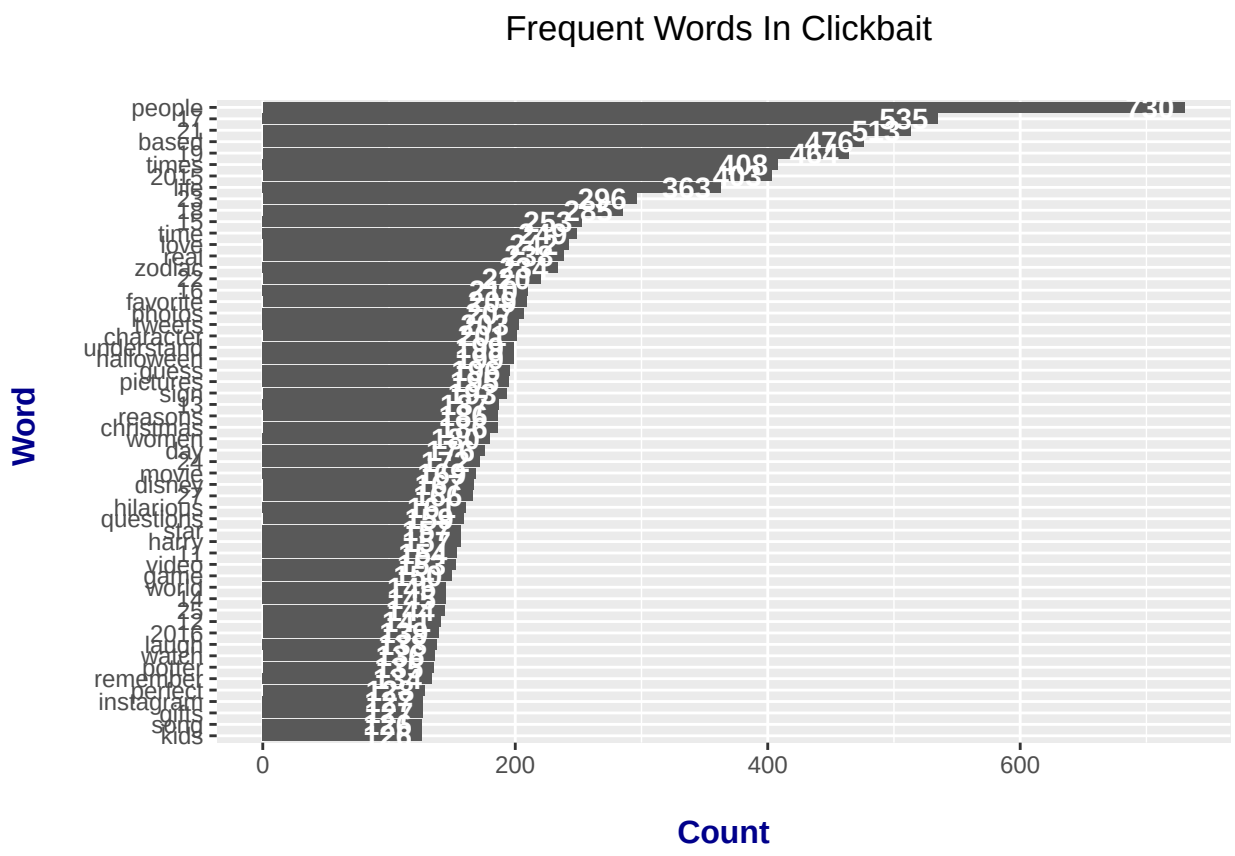
Vocabulary exploration

We check whether there are significant differences in vocabulary between both categories. First, we filter headlines for clickbait category. We use tidytext unnest_tokens function to get word lists and anti_join to remove stop words (words with little meaning, like prepositions and articles). We filter out words having less than 125 ocurrences and plot them. Then we do the same for legit headlines. We can see top words are different in clickbait and legit headlines. We could consider adding the presence of some of these keywords to our feature set, but that would be very labor intensive. Instead, we will try string vectorization.

```
#explore clickbait and regular news vocabulary
sentences<-train_set %>% filter(clickbait==1) %>% select(headline)#select clickbait sentences
words<-sentences %>% unnest_tokens(output=word, input=headline)
words<-words %>% anti_join(stop_words)
```

```
## Joining, by = "word"
```

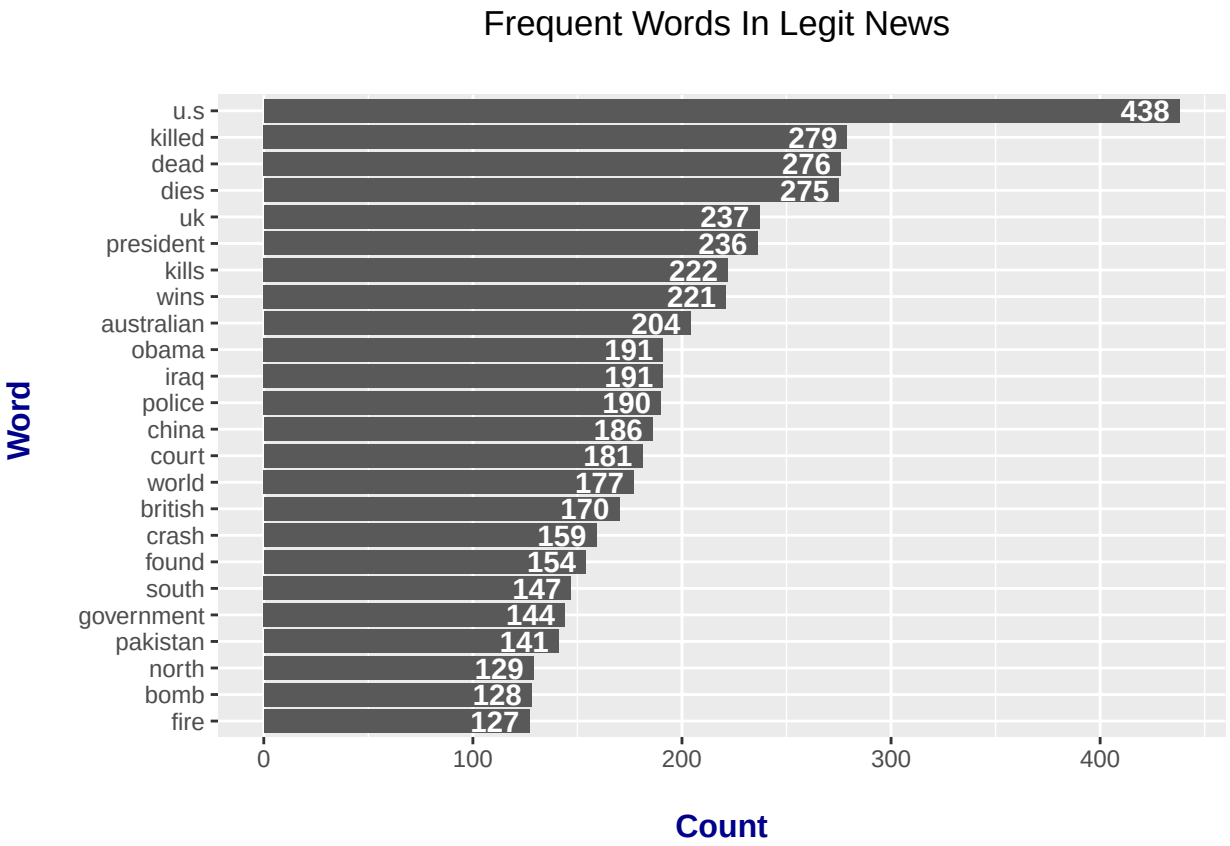
```
word_counts<-words %>% count(word, sort = TRUE)
word_counts %>% filter(n > 125) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  coord_flip() +
  labs(x = "Word \n", y = "\n Count ", title = "Frequent Words In Clickbait \n") +
  geom_text(aes(label = n), hjust = 1.2, colour = "white", fontface = "bold") +
  theme(plot.title = element_text(hjust = 0.5),
        axis.title.x = element_text(face="bold", colour="darkblue", size = 12),
        axis.title.y = element_text(face="bold", colour="darkblue", size = 12))
```



```
sentences<-train_set %>% filter(clickbait==0) %>% select(headline)#select legit news
words<-sentences %>% unnest_tokens(output=word, input=headline)
words<-words %>% anti_join(stop_words)
```

```
## Joining, by = "word"
```

```
word_counts<-words %>% count(word, sort = TRUE)
word_counts %>% filter(n > 125) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  coord_flip() +
  labs(x = "Word \n", y = "\n Count ", title = "Frequent Words In Legit News \n") +
  geom_text(aes(label = n), hjust = 1.2, colour = "white", fontface = "bold") +
  theme(plot.title = element_text(hjust = 0.5),
        axis.title.x = element_text(face="bold", colour="darkblue", size = 12),
        axis.title.y = element_text(face="bold", colour="darkblue", size = 12))
```



Bag of words model String vectorization implies representing text as a numerical vector. In the bag of words model, we just count how many times words appear in sentences. We use text2vec for string vectorization and glmnet for logistic regression, as caret seemeed to struggle with string vectors as input.Accuracy increases to 93.4%.

```
#Creating models based on string vectors

# define preprocessing function and tokenization function
prep_fun = tolower#convert all characters to lower case
tok_fun = word_tokenizer#split sentences into words
```

```
it_train = itoken(train_set$headline, #preprocess
                  preprocessor = prep_fun,
                  tokenizer = tok_fun,
                  progressbar = FALSE)
vocab = create_vocabulary(it_train)#create vocabulary

vectorizer = vocab_vectorizer(vocab)#create vectorizer
dtm_train = create_dtm(it_train, vectorizer)#create document-term matrix for train set



it_test = tok_fun(prep_fun(test_set$headline))#preprocess test set set
it_test = itoken(it_test, progressbar = FALSE)
dtm_test = create_dtm(it_test, vectorizer)

#bag of words
glmnet_classifier = cv.glmnet(x = dtm_train, y = train_set[['clickbait']],
                              family = 'binomial',
                              alpha = 1,
                              type.measure = "auc",
                              nfolds = 3)
```

```
## Warning: from glmnet Fortran code (error code -39); Convergence for 39th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
## Warning: from glmnet Fortran code (error code -72); Convergence for 72th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
## Warning: from glmnet Fortran code (error code -72); Convergence for 72th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
## Warning: from glmnet Fortran code (error code -72); Convergence for 72th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
preds = predict(glmnet_classifier, dtm_test, type = 'class')
print ('bag of words')
```

```
## [1] "bag of words"
```

```
confusionMatrix(factor(preds), test_set$clickbait)$overall['Accuracy']
```

```
##  Accuracy
## 0.9346977
```

TF-IDF

Bag of words only counts frequencies, it assigns the same weight to all words. TF-IDF tries to correct this by penalizing words that are common to all sentences. Text2vec allows to apply a TF-IDF transformation on top of the bag of words. Surprisingly, TF-IDF performs slightly worse than bag og words, 93.2% accuracy.

```
# define tfidf model
tfidf = TfIdf$new()
```

```
# fit model to train data and transform train data with fitted model
dtm_train_tfidf = fit_transform(dtm_train, tfidf)
# tfidf modified by fit_transform() call!
# apply pre-trained tf-idf transformation to test data

dtm_test_tfidf = transform(dtm_test, tfidf)

glmnet_classifier = cv.glmnet(x = dtm_train_tfidf, y = train_set[['clickbait']],
                              family = 'binomial',
                              alpha = 1,
                              type.measure = "auc",
                              nfolds = 3)
```

```
## Warning: from glmnet Fortran code (error code -38); Convergence for 38th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
## Warning: from glmnet Fortran code (error code -71); Convergence for 71th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
## Warning: from glmnet Fortran code (error code -71); Convergence for 71th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
preds = predict(glmnet_classifier, dtm_test_tfidf, type = 'class')
print ('tfidf')
```

```
## [1] "tfidf"
```

```
confusionMatrix(factor(preds), test_set$clickbait)$overall['Accuracy']
```

```
##  Accuracy
## 0.9325105
```

Results

The table below shows the comparisons between different models.

```
models<-c('random', 'linguistic features', 'bag of words', 'tf-idf')
scores<-c(0.517, 0.644, 0.934, 0.932)
df<-data.frame(models, scores)
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 4.1.2
```

```
kable(df, caption='model comparison')
```

Table 1: model comparison

models	scores
random	0.517
linguistic features	0.644
bag of words	0.934
tf-idf	0.932

Conclusions and Future work

Simple linguistic features, like using numbers of headline length seem to be good indicators, but they are clearly outperformed by string vectorization techniques. Bag of words seems to perform slightly better than TF-IDF, but I would pick TF-IDF as a more robust model in general.

There are several avenues for further improvement that I haven't tried due to time constraints. We could try other algorithms frequently used with text (Naive Bayes, neuronal networks) or more complex forms of string vectorization, like words embeddings (which consider not only the words, but also the context where they happen).