DS2 - Assignment 3

Ali Sial

4/17/2022

Introduction

This dataset summarizes a heterogeneous set of features about articles published by Mashable in a period of two years. The goal is to predict if the article is among the most popular ones based on sharing on social networks (coded by the variable is_popular which was created from the original shares variable in a way that is intentionally undisclosed).

The entire project including the data has been uploaded to my github and can retrieved by clicking *here*

Importing Data

To avoid complexity, data will be directly pulled from the created repository for this particular project.

```
# import data from github directly
train_data <- read_csv("https://raw.githubusercontent.com/alisial94/DS2_Assignment3_KaggleCompetition/ma
test_data <- read_csv("https://raw.githubusercontent.com/alisial94/DS2_Assignment3_KaggleCompetition/mai
```

Data Cleaning and Mungging

Upon getting the data, I begin exploring it by first reading the discription of each variable and cheking how it is recorded in the dataset to check for variables that require to be adjusted. After this I decide to explore the structure of each variable so I look in to the possible options for feature engineering and classification.

The train data intails a total of 27752 observations and the test dataset 11892 observations. The provided features/variables to classify popular and non popluar articles is 60. All the variables at this stage are stored as numaric would require to be adjusted. I also looked at the Y variable in the train dataset to check how many of the observations in the train dataset turned out to be popular. It apperas that the data is imbalanced with only around 13% articles turing out to be popular.

```
head(train_data)
```

```
## # A tibble: 6 x 61
##   timedelta n_tokens_title n_tokens_content n_unique_tokens n_non_stop_words
##       <dbl>          <dbl>            <dbl>           <dbl>            <dbl>
## 1       728             11              159           0.759             1.00
## 2        27             11             1056           0.383             1.00
## 3       119              9                0           0                 0
## 4       135             11              797           0.512             1.00
```

```
## 5      223             10              226           0.605             1.00
## 6      154             12              281           0.588             1.00
## # ... with 56 more variables: n_non_stop_unique_tokens <dbl>, num_hrefs <dbl>,
## #   num_self_hrefs <dbl>, num_imgs <dbl>, num_videos <dbl>,
## #   average_token_length <dbl>, num_keywords <dbl>,
## #   data_channel_is_lifestyle <dbl>, data_channel_is_entertainment <dbl>,
## #   data_channel_is_bus <dbl>, data_channel_is_socmed <dbl>,
## #   data_channel_is_tech <dbl>, data_channel_is_world <dbl>, kw_min_min <dbl>,
## #   kw_max_min <dbl>, kw_avg_min <dbl>, kw_min_max <dbl>, kw_max_max <dbl>, ...
```

```
# view(train_data)


str(train_data)
```

```
## spec_tbl_df [29,733 x 61] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ timedelta                    : num [1:29733] 728 27 119 135 223 154 27 66 37 530 ...
##  $ n_tokens_title               : num [1:29733] 11 11 9 11 10 12 17 14 10 11 ...
##  $ n_tokens_content             : num [1:29733] 159 1056 0 797 226 ...
##  $ n_unique_tokens              : num [1:29733] 0.759 0.383 0 0.512 0.605 ...
##  $ n_non_stop_words             : num [1:29733] 1 1 0 1 1 ...
##  $ n_non_stop_unique_tokens     : num [1:29733] 0.896 0.515 0 0.724 0.743 ...
##  $ num_hrefs                    : num [1:29733] 9 3 0 4 6 2 1 17 19 0 ...
##  $ num_self_hrefs               : num [1:29733] 0 2 0 2 5 1 1 1 6 0 ...
##  $ num_imgs                     : num [1:29733] 1 19 1 3 1 1 0 1 5 0 ...
##  $ num_videos                   : num [1:29733] 0 1 1 1 0 0 1 0 3 1 ...
##  $ average_token_length         : num [1:29733] 4.89 4.68 0 4.66 5.08 ...
##  $ num_keywords                 : num [1:29733] 7 9 7 4 6 6 7 7 7 10 ...
##  $ data_channel_is_lifestyle    : num [1:29733] 0 0 0 0 0 0 0 0 1 0 ...
##  $ data_channel_is_entertainment: num [1:29733] 0 0 1 0 0 1 0 1 0 1 ...
##  $ data_channel_is_bus          : num [1:29733] 0 0 0 0 1 0 0 0 0 0 ...
##  $ data_channel_is_socmed       : num [1:29733] 0 0 0 0 0 0 0 0 0 0 ...
##  $ data_channel_is_tech         : num [1:29733] 0 0 0 0 0 0 0 0 0 0 ...
##  $ data_channel_is_world        : num [1:29733] 0 1 0 1 0 0 1 0 0 0 ...
##  $ kw_min_min                   : num [1:29733] 217 -1 -1 -1 -1 -1 -1 -1 -1 4 ...
##  $ kw_max_min                   : num [1:29733] 417 596 662 2000 1900 455 2100 492 0 943 ...
##  $ kw_avg_min                   : num [1:29733] 337 114 103 499 425 ...
##  $ kw_min_max                   : num [1:29733] 0 1900 21900 17400 0 11800 4600 2100 0 0 ...
##  $ kw_max_max                   : num [1:29733] 28000 843300 843300 843300 843300 ...
##  $ kw_avg_max                   : num [1:29733] 6971 218089 512229 405000 486067 ...
##  $ kw_min_avg                   : num [1:29733] 0 1006 3142 2178 0 ...
##  $ kw_max_avg                   : num [1:29733] 2899 3399 7076 7325 5513 ...
##  $ kw_avg_avg                   : num [1:29733] 1063 1905 4704 3870 3149 ...
##  $ self_reference_min_shares    : num [1:29733] 0 0 0 1400 1300 556 1700 1400 922 0 ...
##  $ self_reference_max_shares    : num [1:29733] 0 0 0 27800 1300 556 1700 1400 12200 0 ...
##  $ self_reference_avg_sharess   : num [1:29733] 0 0 0 14600 1300 ...
##  $ weekday_is_monday            : num [1:29733] 0 0 0 0 0 0 0 1 0 0 ...
##  $ weekday_is_tuesday           : num [1:29733] 0 0 0 1 0 0 0 0 1 0 ...
##  $ weekday_is_wednesday         : num [1:29733] 0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday_is_thursday          : num [1:29733] 1 0 1 0 0 1 0 0 0 0 ...
##  $ weekday_is_friday            : num [1:29733] 0 1 0 0 1 0 1 0 0 0 ...
##  $ weekday_is_saturday          : num [1:29733] 0 0 0 0 0 0 0 0 0 1 ...
##  $ weekday_is_sunday            : num [1:29733] 0 0 0 0 0 0 0 0 0 0 ...
##  $ is_weekend                   : num [1:29733] 0 0 0 0 0 0 0 0 0 1 ...
```

```
##  $ LDA_00                       : num [1:29733] 0.0286 0.3552 0.0286 0.05 0.8667 ...
##  $ LDA_01                       : num [1:29733] 0.0286 0.0222 0.0292 0.05 0.0333 ...
##  $ LDA_02                       : num [1:29733] 0.4606 0.5781 0.0286 0.551 0.0333 ...
##  $ LDA_03                       : num [1:29733] 0.1722 0.0222 0.8851 0.299 0.0333 ...
##  $ LDA_04                       : num [1:29733] 0.3099 0.0222 0.0286 0.05 0.0333 ...
##  $ global_subjectivity          : num [1:29733] 0.556 0.358 0 0.397 0.315 ...
##  $ global_sentiment_polarity    : num [1:29733] 0.2406 0.0452 0 0.0495 0.346 ...
##  $ global_rate_positive_words   : num [1:29733] 0.044 0.0152 0 0.0389 0.0752 ...
##  $ global_rate_negative_words   : num [1:29733] 0.0189 0.0104 0 0.0376 0 ...
##  $ rate_positive_words          : num [1:29733] 0.7 0.593 0 0.508 1 ...
##  $ rate_negative_words          : num [1:29733] 0.3 0.407 0 0.492 0 ...
##  $ avg_positive_polarity        : num [1:29733] 0.339 0.299 0 0.315 0.585 ...
##  $ min_positive_polarity        : num [1:29733] 0.1 0.05 0 0.05 0.136 ...
##  $ max_positive_polarity        : num [1:29733] 0.6 0.7 0 0.8 1 0.6 0.8 1 0.8 0 ...
##  $ avg_negative_polarity        : num [1:29733] -0.169 -0.242 0 -0.225 0 ...
##  $ min_negative_polarity        : num [1:29733] -0.25 -0.6 0 -0.7 0 -1 0 -1 -0.8 0 ...
##  $ max_negative_polarity        : num [1:29733] -0.1 -0.1 0 -0.05 0 -0.1 0 -0.1 -0.125 0 ...
##  $ title_subjectivity           : num [1:29733] 0 0 0.35 0 0.6 ...
##  $ title_sentiment_polarity     : num [1:29733] 0 0 0.05 0 0.4 ...
##  $ abs_title_subjectivity       : num [1:29733] 0.5 0.5 0.15 0.5 0.1 ...
##  $ abs_title_sentiment_polarity : num [1:29733] 0 0 0.05 0 0.4 ...
##  $ is_popular                   : num [1:29733] 0 0 1 0 0 0 0 0 0 0 ...
##  $ article_id                   : num [1:29733] 1 2 4 6 11 12 14 15 17 18 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   timedelta = col_double(),
##   ..   n_tokens_title = col_double(),
##   ..   n_tokens_content = col_double(),
##   ..   n_unique_tokens = col_double(),
##   ..   n_non_stop_words = col_double(),
##   ..   n_non_stop_unique_tokens = col_double(),
##   ..   num_hrefs = col_double(),
##   ..   num_self_hrefs = col_double(),
##   ..   num_imgs = col_double(),
##   ..   num_videos = col_double(),
##   ..   average_token_length = col_double(),
##   ..   num_keywords = col_double(),
##   ..   data_channel_is_lifestyle = col_double(),
##   ..   data_channel_is_entertainment = col_double(),
##   ..   data_channel_is_bus = col_double(),
##   ..   data_channel_is_socmed = col_double(),
##   ..   data_channel_is_tech = col_double(),
##   ..   data_channel_is_world = col_double(),
##   ..   kw_min_min = col_double(),
##   ..   kw_max_min = col_double(),
##   ..   kw_avg_min = col_double(),
##   ..   kw_min_max = col_double(),
##   ..   kw_max_max = col_double(),
##   ..   kw_avg_max = col_double(),
##   ..   kw_min_avg = col_double(),
##   ..   kw_max_avg = col_double(),
##   ..   kw_avg_avg = col_double(),
##   ..   self_reference_min_shares = col_double(),
##   ..   self_reference_max_shares = col_double(),
```

```
##   ..   self_reference_avg_sharess = col_double(),
##   ..   weekday_is_monday = col_double(),
##   ..   weekday_is_tuesday = col_double(),
##   ..   weekday_is_wednesday = col_double(),
##   ..   weekday_is_thursday = col_double(),
##   ..   weekday_is_friday = col_double(),
##   ..   weekday_is_saturday = col_double(),
##   ..   weekday_is_sunday = col_double(),
##   ..   is_weekend = col_double(),
##   ..   LDA_00 = col_double(),
##   ..   LDA_01 = col_double(),
##   ..   LDA_02 = col_double(),
##   ..   LDA_03 = col_double(),
##   ..   LDA_04 = col_double(),
##   ..   global_subjectivity = col_double(),
##   ..   global_sentiment_polarity = col_double(),
##   ..   global_rate_positive_words = col_double(),
##   ..   global_rate_negative_words = col_double(),
##   ..   rate_positive_words = col_double(),
##   ..   rate_negative_words = col_double(),
##   ..   avg_positive_polarity = col_double(),
##   ..   min_positive_polarity = col_double(),
##   ..   max_positive_polarity = col_double(),
##   ..   avg_negative_polarity = col_double(),
##   ..   min_negative_polarity = col_double(),
##   ..   max_negative_polarity = col_double(),
##   ..   title_subjectivity = col_double(),
##   ..   title_sentiment_polarity = col_double(),
##   ..   abs_title_subjectivity = col_double(),
##   ..   abs_title_sentiment_polarity = col_double(),
##   ..   is_popular = col_double(),
##   ..   article_id = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
# variables are all stored as numarics will need to adjust them, most of them will be factorised

# display the class and type of each columns
sapply(train_data, class)
```

```
##                    timedelta                n_tokens_title
##                    "numeric"                     "numeric"
##             n_tokens_content               n_unique_tokens
##                    "numeric"                     "numeric"
##             n_non_stop_words      n_non_stop_unique_tokens
##                    "numeric"                     "numeric"
##                    num_hrefs                num_self_hrefs
##                    "numeric"                     "numeric"
##                     num_imgs                    num_videos
##                    "numeric"                     "numeric"
##         average_token_length                  num_keywords
##                    "numeric"                     "numeric"
##     data_channel_is_lifestyle data_channel_is_entertainment
##                    "numeric"                     "numeric"
```

```
##           data_channel_is_bus        data_channel_is_socmed
##                     "numeric"                     "numeric"
##          data_channel_is_tech         data_channel_is_world
##                     "numeric"                     "numeric"
##                    kw_min_min                    kw_max_min
##                     "numeric"                     "numeric"
##                    kw_avg_min                    kw_min_max
##                     "numeric"                     "numeric"
##                    kw_max_max                    kw_avg_max
##                     "numeric"                     "numeric"
##                    kw_min_avg                    kw_max_avg
##                     "numeric"                     "numeric"
##                    kw_avg_avg     self_reference_min_shares
##                     "numeric"                     "numeric"
##     self_reference_max_shares    self_reference_avg_sharess
##                     "numeric"                     "numeric"
##             weekday_is_monday            weekday_is_tuesday
##                     "numeric"                     "numeric"
##          weekday_is_wednesday           weekday_is_thursday
##                     "numeric"                     "numeric"
##             weekday_is_friday           weekday_is_saturday
##                     "numeric"                     "numeric"
##             weekday_is_sunday                    is_weekend
##                     "numeric"                     "numeric"
##                        LDA_00                        LDA_01
##                     "numeric"                     "numeric"
##                        LDA_02                        LDA_03
##                     "numeric"                     "numeric"
##                        LDA_04           global_subjectivity
##                     "numeric"                     "numeric"
##     global_sentiment_polarity    global_rate_positive_words
##                     "numeric"                     "numeric"
##    global_rate_negative_words           rate_positive_words
##                     "numeric"                     "numeric"
##           rate_negative_words         avg_positive_polarity
##                     "numeric"                     "numeric"
##         min_positive_polarity         max_positive_polarity
##                     "numeric"                     "numeric"
##         avg_negative_polarity         min_negative_polarity
##                     "numeric"                     "numeric"
##         max_negative_polarity            title_subjectivity
##                     "numeric"                     "numeric"
##      title_sentiment_polarity        abs_title_subjectivity
##                     "numeric"                     "numeric"
##  abs_title_sentiment_polarity                    is_popular
##                     "numeric"                     "numeric"
##                    article_id
##                     "numeric"
```

```
sapply(train_data, typeof)
```

```
##                     timedelta                n_tokens_title
##                      "double"                      "double"
##              n_tokens_content               n_unique_tokens
```

```
##                     "double"                       "double"
##             n_non_stop_words      n_non_stop_unique_tokens
##                     "double"                       "double"
##                    num_hrefs                num_self_hrefs
##                     "double"                       "double"
##                     num_imgs                    num_videos
##                     "double"                       "double"
##         average_token_length                  num_keywords
##                     "double"                       "double"
##    data_channel_is_lifestyle data_channel_is_entertainment
##                     "double"                       "double"
##          data_channel_is_bus        data_channel_is_socmed
##                     "double"                       "double"
##         data_channel_is_tech         data_channel_is_world
##                     "double"                       "double"
##                   kw_min_min                    kw_max_min
##                     "double"                       "double"
##                   kw_avg_min                    kw_min_max
##                     "double"                       "double"
##                   kw_max_max                    kw_avg_max
##                     "double"                       "double"
##                   kw_min_avg                    kw_max_avg
##                     "double"                       "double"
##                   kw_avg_avg     self_reference_min_shares
##                     "double"                       "double"
##    self_reference_max_shares    self_reference_avg_sharess
##                     "double"                       "double"
##            weekday_is_monday            weekday_is_tuesday
##                     "double"                       "double"
##         weekday_is_wednesday           weekday_is_thursday
##                     "double"                       "double"
##            weekday_is_friday           weekday_is_saturday
##                     "double"                       "double"
##            weekday_is_sunday                    is_weekend
##                     "double"                       "double"
##                       LDA_00                        LDA_01
##                     "double"                       "double"
##                       LDA_02                        LDA_03
##                     "double"                       "double"
##                       LDA_04           global_subjectivity
##                     "double"                       "double"
##    global_sentiment_polarity    global_rate_positive_words
##                     "double"                       "double"
##   global_rate_negative_words           rate_positive_words
##                     "double"                       "double"
##          rate_negative_words         avg_positive_polarity
##                     "double"                       "double"
##        min_positive_polarity         max_positive_polarity
##                     "double"                       "double"
##        avg_negative_polarity         min_negative_polarity
##                     "double"                       "double"
##        max_negative_polarity            title_subjectivity
##                     "double"                       "double"
##     title_sentiment_polarity        abs_title_subjectivity
```

is_popular	cnt
0	25912
1	3821

```
##                      "double"                             "double"
##  abs_title_sentiment_polarity                           is_popular
##                      "double"                             "double"
##                    article_id
##                      "double"
```

```
# looking at the possible distribution of popular and unpopular articles in the train dataset
train_data %>%
  group_by(is_popular) %>%
  summarise(cnt = n()) %>%
  kbl() %>%
  kable_minimal()
```

The first towards data cleaning, started to change the dummy variables in the data to factors in order to make it easy for R Studio to read the variable.

```
# creating a function to covert the variables in both train and test datasets
con_var_fun <- function(x) {
      x %>% mutate(
    data_channel_is_lifestyle = factor(data_channel_is_lifestyle),
    data_channel_is_entertainment = factor(data_channel_is_entertainment),
    data_channel_is_bus = factor(data_channel_is_bus),
    data_channel_is_socmed = factor(data_channel_is_socmed),
    data_channel_is_tech = factor(data_channel_is_tech),
    data_channel_is_world = factor(data_channel_is_world),
    weekday_is_monday = factor(weekday_is_monday),
    weekday_is_tuesday = factor(weekday_is_tuesday),
    weekday_is_wednesday = factor(weekday_is_wednesday),
    weekday_is_thursday = factor(weekday_is_thursday),
    weekday_is_friday = factor(weekday_is_friday),
    weekday_is_saturday = factor(weekday_is_saturday),
    weekday_is_sunday = factor(weekday_is_sunday),
    is_weekend = factor(is_weekend),
    article_id = factor(article_id),
)
}


conversion <- list( train_data, test_data ) %>%
    lapply( con_var_fun )

train_data <- conversion[[1]]
test_data <- conversion[[2]]

# I also decide to convert the outcome variable "is_popular" in the train dataset to factor
train_data <- train_data %>% mutate(
    is_popular = factor(is_popular))
```

Next step was to explore the data to identifying columns with missing values and based on the result there are no empty columns.

```
to_filter <- sapply(setdiff(names(train_data),'is_popular'), function(x) sum(is.na(x)))
to_filter[to_filter > 0]
```

```
## named integer(0)
```

After this I decided to look at the distribution and other attributes of numeric variable to identify individual variables that might require some imputation or adjustment.

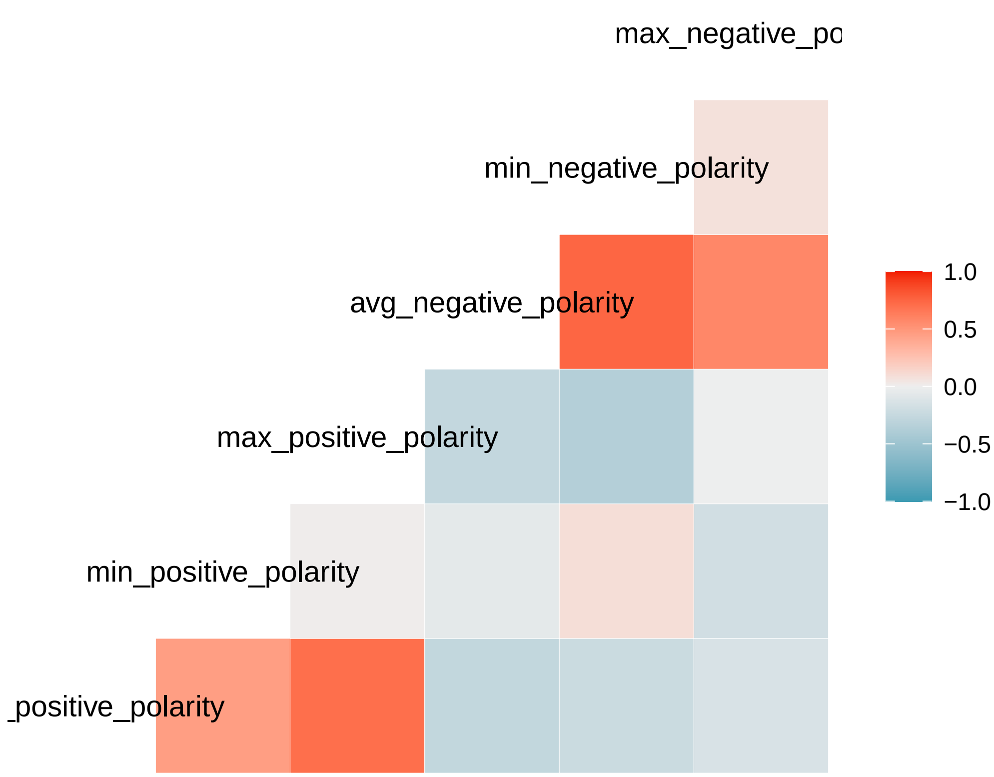
```
# taking a look at all the variables (identify skews for feature engineering)
#skim(train_data)
```

It appears that a lot of numaric variables tend are skewed therefore it will be wise to take log for these variables in order to incorporate that in our complex models. While looking for distribution of the varibales using 'skim', you can also observe few variables tend to have negative values thus, before I went on to add log terms for the features, I decided to carry out some feature engineering.

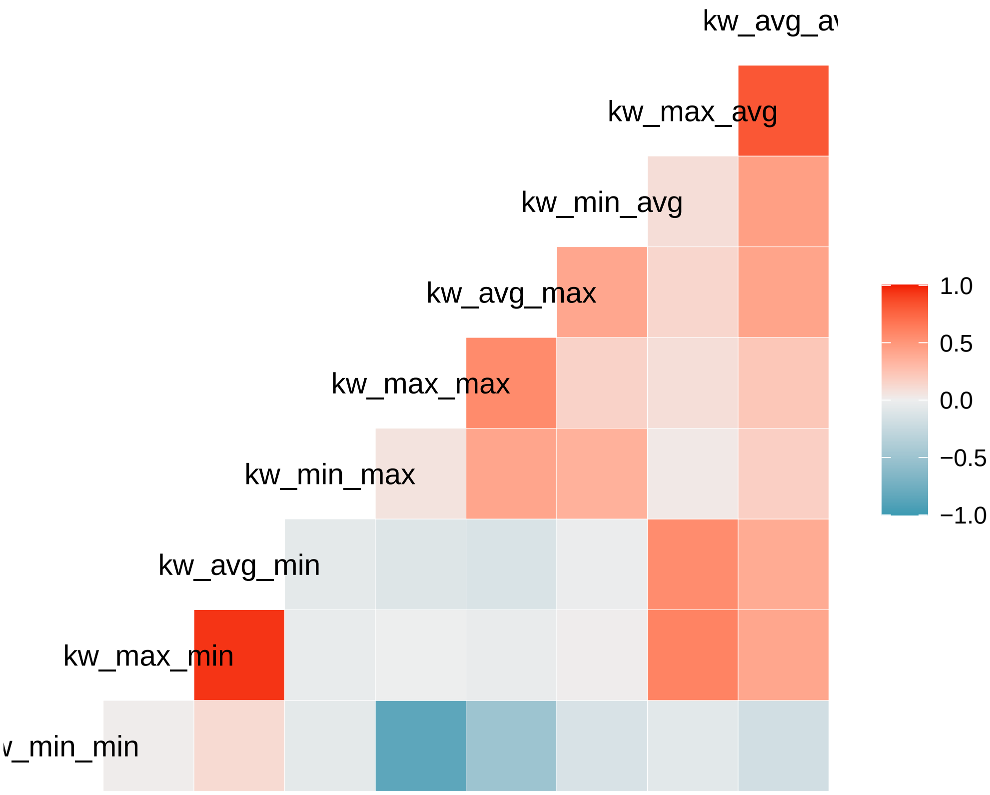
Feature Engineering

Before I fixed the columns with the negative values, I decided to filter out variables that either had a high correlation or were redundant due not having much variation in values imputed in the column. To this I explored the correaltion between similiar type of variables.

```
# looing at correlations between polarity features
ggcorr(subset(train_data,select = c(avg_positive_polarity,min_positive_polarity,max_positive_polarity,
                                    avg_negative_polarity,min_negative_polarity,max_negative_polarity)))
```

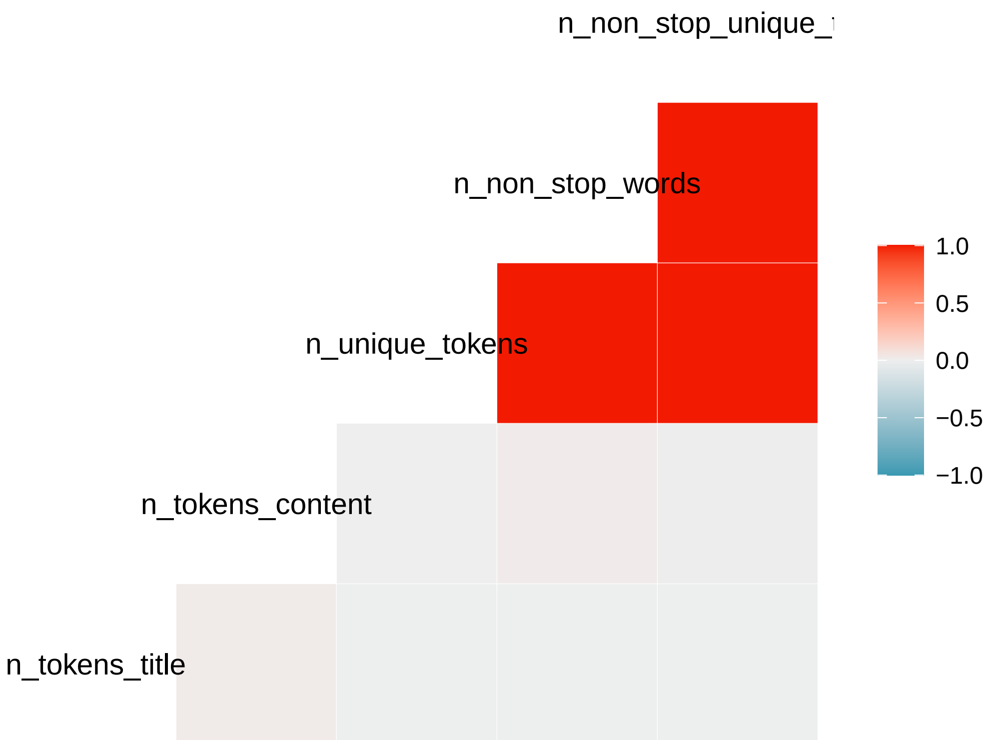


```r
# looking at correlations between keyword measures
ggcorr(subset(train_data,select = c(kw_min_min, kw_max_min, kw_avg_min,
                                     kw_min_max, kw_max_max, kw_avg_max,
                                     kw_min_avg, kw_max_avg, kw_avg_avg)))
```




```r
# drop max and min columns and keep the avg columns


# check correlations between word measures
ggcorr(subset(train_data,select = c(n_tokens_title, n_tokens_content,
                                     n_unique_tokens, n_non_stop_words,
                                     n_non_stop_unique_tokens)))
```




```
# correlation between Rate of non-stop words, Rate of unique non-stop words and Rate of unique words
# is extremely high (as expected), therefore drop two variables from three.
```

Most of the polarity features record pretty much the same thing, therefore, as shown above they tend to have high correaltion. In order to avoid over fitting are model, I have decided, for simiplicity, to just keep the variables recording the averages and drop the max and mins. I plan on doing the same for the keyword measures as well.

After looking at the tokens and words realted features in the dataset, I realised that correlation between rate of non-stop words, rate of unique non-stop words and rate of unique words is extremely high. I have decide to drop rate of non-stop words and rate of unique non-stop words.

```
# rate_positive_words and rate_negative_words adds  to 1 thus keeping only one
# remove redundant is_weekend as weekday dummies already exist
to_drop <- c("kw_min_min", "kw_max_min",
             "kw_avg_min", "kw_min_max",
             "kw_max_max", "kw_avg_max",
             "kw_min_avg", "kw_max_avg", "self_reference_min_shares",
             "self_reference_max_shares", "is_weekend", "rate_negative_words","min_positive_polarity",
             "max_positive_polarity", "min_negative_polarity", "max_negative_polarity")
# drop listed variables
train_data <- subset(train_data, select = setdiff(names(train_data), to_drop))
test_data <- subset(test_data, select = setdiff(names(test_data), to_drop))
```

Since we have final set of varibales that we will be using for modeling, the next step was to check the columns with negative values. It appears that only 3 columns have neagtive values and it makes sense for all of these columns to have negative values. Thus, i will be leaving them as is.

```
temp <- Filter(is.numeric, train_data)
for (col in names(temp)){
  min <- min(temp[,col])
  if (min < 0){
   print(c(col, min))
  } else {
    next
  }
}
```

```
## [1] "global_sentiment_polarity" "-0.380208333333"
## [1] "avg_negative_polarity" "-1"
## [1] "title_sentiment_polarity" "-1"
```

Now I would again look at all the remaining variables to identify the variables that I will computing log normal values.

```
# taking a look at all the variables (identify skews for feature engineering)
#skim(train_data)
```

```
# add logs of skewed features to train and test dataset
impute_log <- function(x) {
      x %>% mutate(
  log_n_tokens_content = ifelse(n_tokens_content <=0,0,log(n_tokens_content)),
  log_n_unique_tokens = ifelse(n_unique_tokens <=0,0, log(n_unique_tokens)),
  log_n_non_stop_words = ifelse(n_non_stop_words<=0,0,log(n_non_stop_words)),
         log_n_non_stop_unique_tokens = ifelse(n_non_stop_unique_tokens<=0,0,
                                               log(n_non_stop_unique_tokens)),
         log_num_hrefs = ifelse(num_hrefs<=0,0,log(num_hrefs)),
         log_num_self_hrefs = ifelse(num_self_hrefs<=0,0,log(num_self_hrefs)),
         log_num_imgs = ifelse(num_imgs<=0,0,log(num_imgs)),
         log_num_videos = ifelse(num_videos<=0,0,log(num_videos)),
  log_kw_avg_avg = ifelse(kw_avg_avg<=0,0,log(kw_avg_avg)),
  log_self_reference_avg_sharess = ifelse(self_reference_avg_sharess<=0,0,
                                                 log(self_reference_avg_sharess)),
  log_LDA_00 = ifelse(LDA_00<=0,0,log(LDA_00)),
         log_LDA_01 = ifelse(LDA_01<=0,0,log(LDA_01)),
         log_LDA_02 = ifelse(LDA_02<=0,0,log(LDA_02)),
         log_LDA_03 = ifelse(LDA_03<=0,0,log(LDA_03)),
         log_LDA_04 = ifelse(LDA_04<=0,0,log(LDA_04)),
  log_global_rate_negative_words = ifelse(global_rate_negative_words<=0,0,log(global_rate_negative_words
)
}
make_log <- list( train_data, test_data ) %>%
    lapply( impute_log )
train_data <- make_log[[1]]
test_data <- make_log[[2]]
```

Now that all the variables that required to be log transformed have done, I would be dropping all the features for log normal value have been computed. This will provide us with are final dataset to move forwards towards modeling.

```r
drop <- c("n_tokens_content","n_unique_tokens","n_non_stop_words","n_non_stop_unique_tokens",
          "num_hrefs","num_self_hrefs","num_imgs","num_videos","kw_avg_avg",
          "self_reference_avg_sharess", "LDA_00","LDA_01","LDA_02","LDA_03","LDA_04",
          "global_rate_negative_words")

train_data <- select(train_data,-drop)
test_data <- select(test_data,-drop)
```

Modelling Choices

I will start with defining the variables as you can see below.

```r
# Y variable
y <- 'is_popular'

# keep first 45 vars for level
x <- setdiff(names(train_data[, 1:45]), c("is_popular", "article_id"))
#print(x)
```

For modeling, as directed in the task, I will be creating the following models: - linear model (lasso) - random forest - gradient boosting - neural nets + parameter tuning - stacking

Before I start building the models, Lets first divide the data set into train and validate. For this task, I have decide to only assign around 15% since i did not want to reduce a lot of observation for training set.

```r
splits <- h2o.splitFrame(as.h2o(train_data), ratios = 0.85, seed = my_seed)
```

```
##   |                                                                            |
```

```r
data_train <- splits[[1]]
data_valid <- splits[[2]]

data_test <- as.h2o(test_data)
```

```
##   |                                                                            |
```

I have saved the results of the selected models in the computer and will be directly calling them from there to avoid long kintting time.

Model 1: GLM-Lasso

Instead of running a simple linear model, I decided to go with lasso. I am using Lasso because I dont belive I have a good amount of domain knowledge for the features being used in this prediction and in order to avoid over-fitting the model, the lasso will penalise the column to zero if they do not contribute much to the predtion. I best AUC value I obtained was with alpha = 1 and lambda = 0.0034.

```r
#train lasso model with lambda search
#lasso_model <- h2o.glm(
#     x, y,
#     training_frame = data_train,
#     model_id = "lasso_model",
#     family = "binomial",
#     alpha = 1,
#     lambda_search = TRUE,
#     seed = my_seed,
#     nfolds = 5,
#     validation_frame = data_valid,
#     keep_cross_validation_predictions = TRUE, # needed for stacking
#     score_each_iteration = TRUE
#   )

# # save model to file
# model_path <- h2o.saveModel(object = lasso_model,
#                             path = "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models",
#                              force = TRUE)


# import model from file
best_lasso <- h2o.loadModel(
  "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models/lasso_model")



# Result best_lasso


AUC_results <- tibble(
    model = "best_lasso",
    train = h2o.auc(best_lasso, train = TRUE),
    valid = h2o.auc(best_lasso, valid = TRUE)
)

# prediction for test set
#prediction <- h2o.predict(best_lasso, newdata = data_test)

# bind predictions with article id-s
#solution <- cbind(test_data[, 'article_id'], as.data.frame(prediction[, 3]))

# rename columns
#colnames(solution) <- c('article_id', 'score')

# write to csv
#write_csv(solution, '~/DS2_ML/DS2_Assignment3_KaggleCompetition/submissions/best_lasso.csv')
```

Model 2: Random Forest

After lasso I decided to go head with random forest next. Since running the model was taking a lot of time due to limitation of my machine, even h2o started to fail every model after the 4th model was run. Therefore, I have only ran 4 different random forest models and selected model 4 since it produced the highest auc.

model_ids	max_depth	mtries	ntrees	sample_rate	auc
1	10	5	200	0.65	0.70854
2	10	7	200	0.65	0.70865
3	15	7	200	0.65	0.70607
4	10	10	200	0.65	0.71085

Hyper-Parameter Search Summary

```
# rf_params <- list(
#    ntrees = 200, # number of trees grown
#    mtries = 10, # number of variables to choose at each split
#    sample_rate = 0.65, # sample rate for the bootstrap samples
#    max_depth = 10 # depth of the trees
#  )

# # train model for level
# rf_grid <- h2o.grid(
#    "randomForest",
 #   x = x, y = y,
#    training_frame = data_train,
#    grid_id = "rf_model",
#    nfolds = 5,
#    seed = my_seed,
#    hyper_params = rf_params,
#    validation_frame = data_valid,
#    keep_cross_validation_predictions = TRUE
 # )

# check AUC for different parameters
#rf_results <- h2o.getGrid(rf_grid@grid_id, sort_by = "auc", decreasing = TRUE)

# save best rf model
# best_rf <- h2o.getModel(
 #   h2o.getGrid(rf_grid@grid_id, sort_by = "auc", decreasing = TRUE)@model_ids[[1]]
 # )

# save model to file
# model_path <- h2o.saveModel(object = best_rf,
#                               path = "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models
#                               force = TRUE)

# import model from file
best_rf <- h2o.loadModel("/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models/rf_model_mode


# prediction for test set
#prediction <- h2o.predict(best_rf, newdata = data_test)

# bind predictions with article id-s
#solution <- cbind(test_data[, 'article_id'], as.data.frame(prediction[, 3]))

# rename columns
#colnames(solution) <- c('article_id', 'score')
```

```
# write to csv
#write_csv(solution, '/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/submissions/best_rf.csv

AUC_results <- add_row(AUC_results,
    model = "best_rf",
    train = h2o.auc(best_rf, train = TRUE),
    valid = h2o.auc(best_rf, valid = TRUE)

)
```

Model 3: Gradient Boosting

Again when running the gbm modeli faced a lot of computational power problems resulting in model failure after the 5th model. Therefore, I have gone ahead with the best out the 5 models I was able to run.

```
# create parameter grid
# gbm_params <- list(
#    learn_rate = 0.1,
#    ntrees = 100,
#    max_depth = 5,
#    sample_rate = 0.7
#  )
#
# # train model
#  gbm_grid <- h2o.grid(
#    "gbm", x = x, y = y,
#    grid_id = "gbm_model",
#    training_frame = data_train,
#    nfolds = 5,
#    seed = my_seed,
#    hyper_params = gbm_params,
#    validation_frame = data_valid,
#    keep_cross_validation_predictions = TRUE # needed for stacking
#  )


# check AUC for different parameters
#gbm_result <- h2o.getGrid(gbm_grid@grid_id, sort_by = "auc", decreasing = TRUE)
#gbm_result

# save best gbm model
#best_gbm <- h2o.getModel(h2o.getGrid(gbm_grid@grid_id, sort_by = "auc",
#           decreasing = TRUE)@model_ids[[1]])

# save model to file
# model_path <- h2o.saveModel(object = best_gbm,
#                              path = "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models
#                              force = TRUE)

# import model from file
best_gbm <- h2o.loadModel(
  "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models/gbm_model_model_5")
```

```r
# prediction for test set
#prediction <- h2o.predict(best_gbm, newdata = data_test)

# bind predictions with article id-s
#solution <- cbind(test_data[, 'article_id'], as.data.frame(prediction[, 3]))

# rename columns
#colnames(solution) <- c('article_id', 'score')

# write to csv
#write_csv(solution, '/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/submissions/best_gbm.cs

AUC_results <- add_row(AUC_results,
    model = "best_gbm",
    train = h2o.auc(best_gbm, train = TRUE),
    valid = h2o.auc(best_gbm, valid = TRUE)

)
```

Model 4: neural nets + parameter tuning

The fourth model i used was based on nural networks. I tried differnt parameters but the improvement in the AUC was not much. Therefore, I selected the one with the highest AUC for prediction.

```r
# create parameter grid
#  nn_params <- list(
#     hidden=c(200,150),
#     hidden_dropout_ratios = c(0.20,0.30),
#     rate=c(0.15,0.25) # learning rate
#   )
#
# # train model
#  nn_grid <- h2o.grid(
#     algorithm="deeplearning",
#     x = x, y = y,
#     training_frame = data_train,
#     grid_id = "nn_model",
#     standardize = TRUE,
#     seed = my_seed,
#     nfolds = 5,
#     validation_frame = data_valid,
#     hyper_params = nn_params,
#     activation = "RectifierWithDropout", # ReLu + dropout because of dropout layers
#     epochs = 30, # standard number of epochs for computer not to catch on fire
#     stopping_rounds = 3, # 3 consecutive rounds of unimproved performance
#     stopping_metric = "AUC", # stopping metric of choice as this is classification
#     stopping_tolerance = 0.01, # stop when misclassification does not improve by >=1% for 3 scoring e
#     keep_cross_validation_predictions = TRUE # needed for stacking
#   )

# check AUC for different parameters
# h2o.getGrid(nn_grid@grid_id, sort_by = "auc", decreasing = TRUE)
```

```r
#save best gbm model
# best_nn <- h2o.getModel(
#    h2o.getGrid(nn_grid@grid_id, sort_by = "auc", decreasing = TRUE)@model_ids[[1]]
#  )

# save model to file
# model_path <- h2o.saveModel(object = best_nn,
#                             path = "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models/
#                              force = TRUE)

# import model from file
best_nn <- h2o.loadModel(
  "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models/nn_model_model_19")

# get AUC for best neural network model
#nn_auc <- h2o.auc(best_nn, train_data = TRUE, xval = TRUE, valid = TRUE)

#knitr::kable(t(nn_auc), caption = "Best Deeplearning Model - Train, CV & Validation AUC")

# prediction for test set
#prediction <- h2o.predict(best_nn, newdata = data_test)

# bind predictions with article id-s
#solution <- cbind(test_data[, 'article_id'], as.data.frame(prediction[, 3]))

# rename columns
#colnames(solution) <- c('article_id', 'score')

# write to csv
#write_csv(solution, '/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/submissions/best_nn.csv

AUC_results <- add_row(AUC_results,
    model = "best_nn",
    train = h2o.auc(best_gbm, train = TRUE),
    valid = h2o.auc(best_gbm, valid = TRUE)

)
```

Model 5: Stacking

I tried running a stacked ensemble model, but I couldn't figure out why it kept running into a error and unfortunately I couldn't solve the issue. Therefor, for my last model i decided to go ahead with a auto-ml since it pretty much looks at all models and then chooses the best.

```r
# save best models to a list
#base_learners <- list(
#  best_rf, best_gbm, best_nn, best_lasso
#)
```

```r
# stacked ensemble model with glm as the meta learner
#ensemble_model <- h2o.stackedEnsemble(
#  x = x, y = y,
```

```
#  model_id = "stacked_model",
#  training_frame = data_train,
#  base_models = base_learners,
#  validation_frame = data_valid,
#  seed = my_seed,
#  metalearner_nfolds = 5
#)
```

Model 6: Auto ML

After running auto ML (since non of the other complected models were working on my machine), it turns out the stacked ensemble model within auto ml performed the best so far and has the highest auc among other models previously computed.

```
# automl <- h2o.automl(
#     x = x, y = y,
#     training_frame = data_train,
#     validation_frame = data_valid,
#     nfolds = 5,
#     sort_metric = "AUC",
#     seed = my_seed,
#     max_runtime_secs = 600 # limit the run-time
# )
# automl
#h2o.auc(h2o.performance(automl@leader, valid = TRUE))

#save best auto-ml model
#best_automl <- automl@leader

# save model to file
#model_path <- h2o.saveModel(object = best_automl,
#                              path = "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models
#                               force = TRUE)

# import model from file
best_automl <- h2o.loadModel(
  "/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/models/StackedEnsemble_AllModels_3_AutoML_

# prediction for test set
# prediction <- h2o.predict(best_automl, newdata = data_test)

# bind predictions with article id-s
# solution <- cbind(test_data[, 'article_id'], as.data.frame(prediction[, 3]))

# rename columns
# colnames(solution) <- c('article_id', 'score')

# write to csv
# write_csv(solution, '/Users/atharsial/DS2_ML/DS2_Assignment3_KaggleCompetition/submissions/best_autom

AUC_results <- add_row(AUC_results,
    model = "best_automl-ensemble",
    train = h2o.auc(best_automl, train = TRUE),
```

```
    valid = h2o.auc(best_automl, valid = TRUE)
)
```

AUC Comparison Table

model	train	valid
best_lasso	0.6862400	0.6866800
best_rf	0.7058827	0.7108533
best_gbm	0.8319530	0.7059420
best_nn	0.8319530	0.7059420
best_automl-ensemble	0.9560590	0.7164733