Kaggle: Predicting Online News Popularity - Report

Anna Corretger, Santhosh Narayanan, Guglielmo Pelino

Introduction

The goal is to develop a multi-class classifier that predicts the labels for the test data set. Observations are online news articles with 60 features and the goal is to predict the level of popularity of the article. The data comes from website mashable.com from beginning of 2015. There are five classes you are trying to predict, obscure (1) articles that are shared very few times, mediocre (2), popular (3), super popular (4) and viral (5) articles that are shared. We are provided with 30000 labeled training data, and 9644 observations in the test data, which are not labeled.

Errors in data collection

To verify the accuracy of the data collection process, we decided to manually open the articles from the url. We found that a large number of articles had errors, specifically in the following fields, Data Channel, Words in title, No. of Images, No. of Videos. See an example below where we take a snapshot of the article from the website and show that while the article clearly falls in the world category, in the dataset it's is coded incorrectly. This means that there were issues during the data collection process, but we were not able to identify any pattern in them.


```
##                                                 url
## 2 http://mashable.com/2014/05/05/etihad-first-class/
##   data_channel_is_entertainment data_channel_is_world
## 2                             1                     0
```

Exploratory Data Analysis

As a first step we want to explore the dataset: identify possible outliers, handle missing values and almost constant variables and explore possible correlations between features.

Outliers and redundant features

We removed the following observation as the columns for rate of unique words and non-stop unique words are greater than 1.

```
##                                                                    url
## 22686 http://mashable.com/2014/08/18/ukraine-civilian-convoy-attacked/
##       n_unique_tokens
## 22686             701
```

```
##                   freqRatio percentUnique zeroVar  nzv
## n_non_stop_words  33.00113    0.01000033   FALSE TRUE
## kw_min_max        44.01701    3.17677256   FALSE TRUE
```

Missing values and handling with/without imputation

We noted that in the dataset there were many missing values and they are coded as 0's: for treating them we used the function "handle.missing" in MedMast package, which either removes or imputes based on the nearest neighbours.
However since the test dataset also contains these missing values we could not remove these records: thus, we created a binary feature which was a flag for the missing values.
Also, we realized that dataset$n_non_stop_words was a constant feature, and thus we removed it from the dataset.

Correlated predictors

We observed that rate_negative_words was completely determined by the rate of positive words which was present as another feature, and thus removed it from the dataset. While there are some models that thrive on correlated predictors, other models may benefit from reducing the level of correlation between the predictors. Below shows the effect of removing descriptors with absolute correlations above 0.75.

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.858100 -0.023520  0.001262  0.013420  0.035070  0.943700
```

```
## [1] "Highly correlated predictors"
```

```
## [1] "n_non_stop_unique_tokens"  "kw_avg_avg"
## [3] "data_channel_is_world"     "LDA_00"
## [5] "kw_max_max"                "self_reference_max_shares"
## [7] "kw_max_min"                "self_reference_min_shares"
```

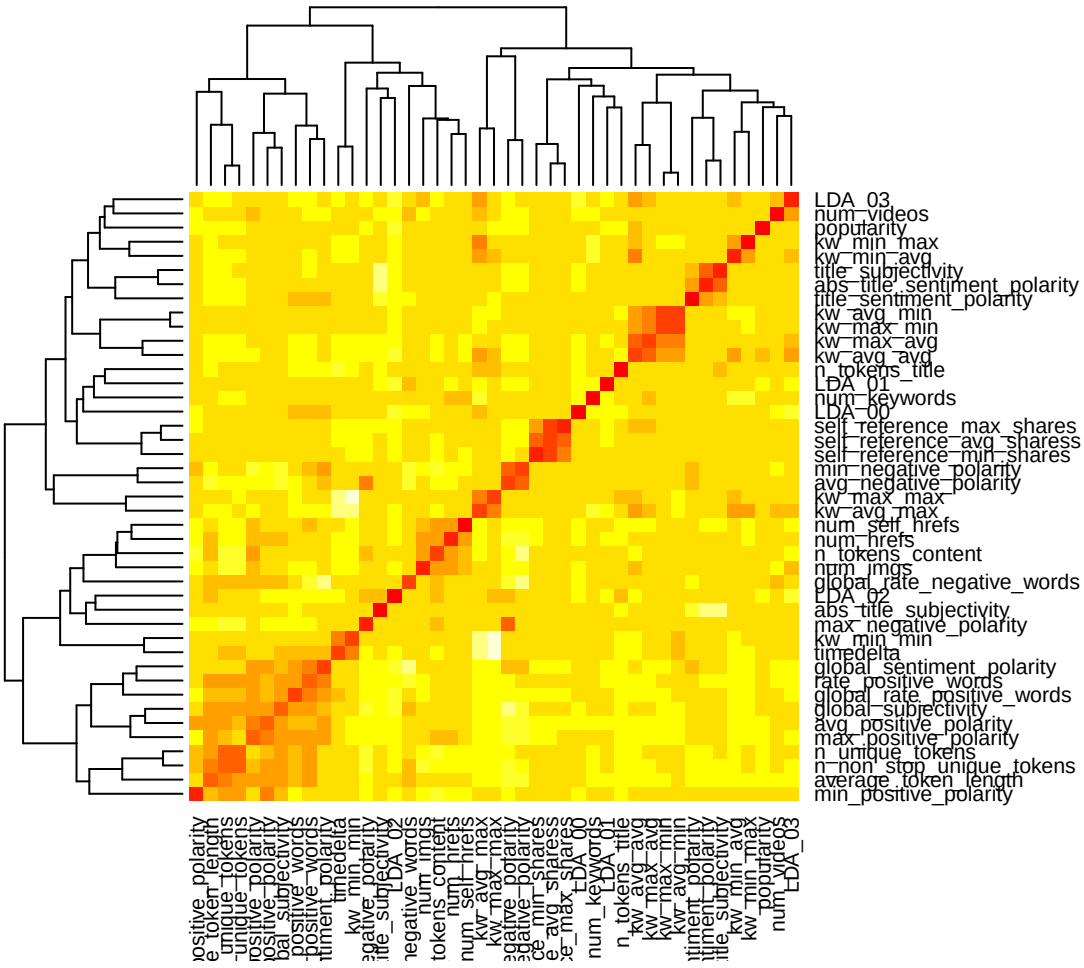
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.50170 -0.02010  0.00120  0.01282  0.03795  0.74810
```

Next we explore if there are any linear dependencies within the predictors. We obviously see that the weekday indicators saturday, sunday and weekend are a linear combination as well the 5 LDA predictors. Hence we decide to remove is_weekend and LDA04 columns.

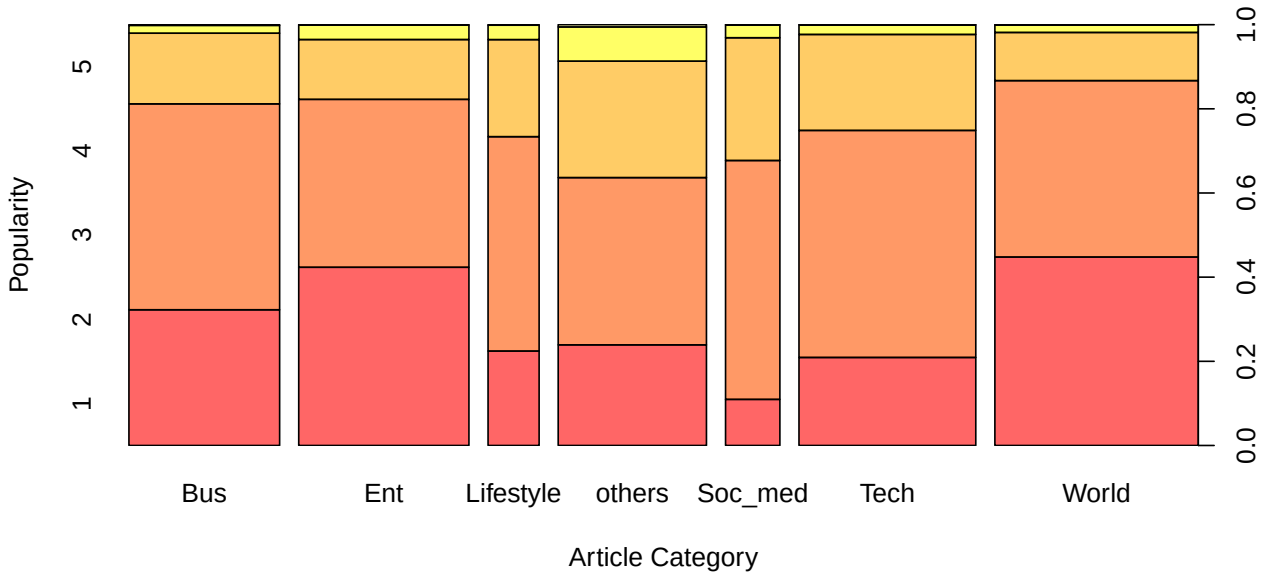
```
## $linearCombos
## $linearCombos[[1]]
## [1] "is_weekend"          "weekday_is_saturday" "weekday_is_sunday"
##
## $linearCombos[[2]]
##  [1] "LDA_04"               "weekday_is_monday"    "weekday_is_tuesday"
##  [4] "weekday_is_wednesday" "weekday_is_thursday"  "weekday_is_friday"
##  [7] "weekday_is_saturday"  "weekday_is_sunday"    "LDA_00"
## [10] "LDA_01"               "LDA_02"               "LDA_03"
##
##
## $remove
## $remove[[1]]
## [1] "is_weekend"
##
## $remove[[2]]
## [1] "LDA_04"
```

Now that we have cleaned the data, we look at the remaining features in more detail.

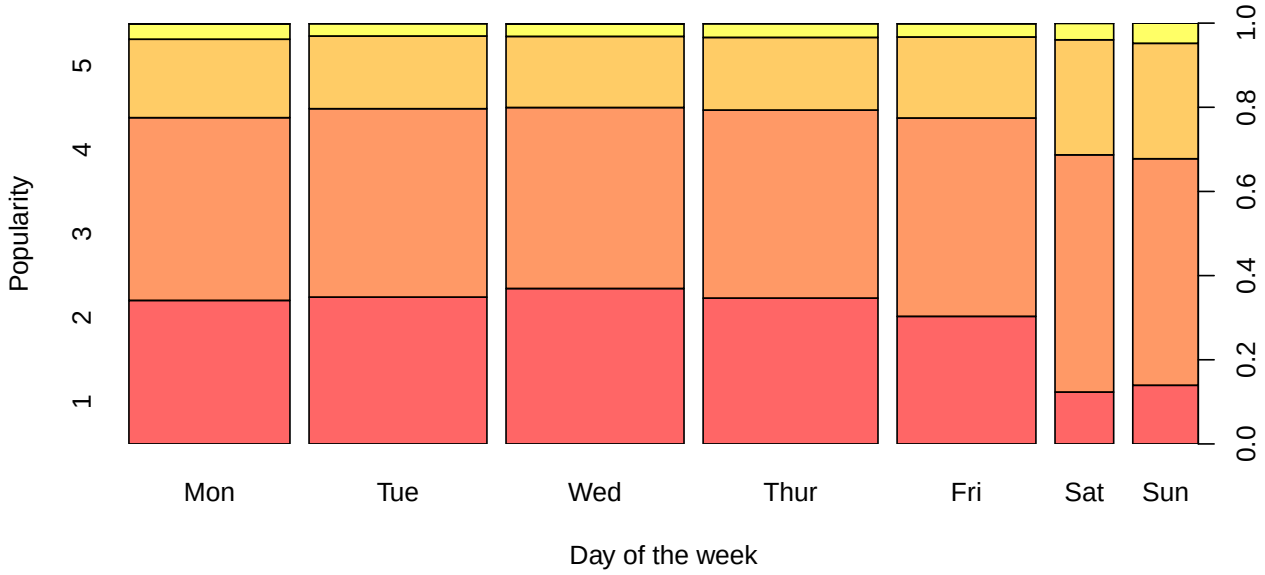
We first plot a correlation heatmap to study the interaction between the variables. The figure also shows the clustering of the predictors



Does the article category have an impact on the popularity?



Does the day of the week have an impact on the popularity?



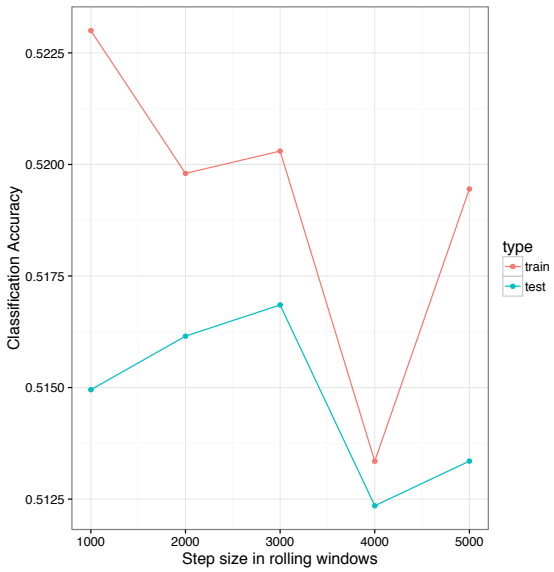
Here we note that the weekends are showing a significant difference from the weekdays.

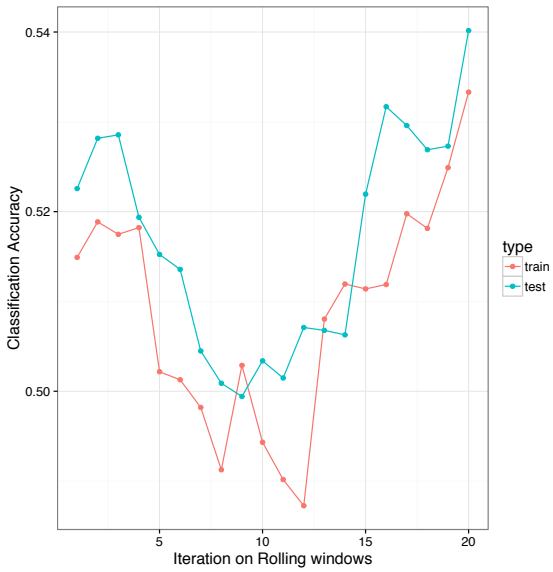
New features

As previously seen in the missing values section, we first added a flag feature called "missing.flag" which takes the value 1 if the row had missing or non-sensical values.
We then added date variables (year, month and quarter) in order to exploit time and seasonality factors in the underlying dynamics of popularity of the published news.
For analyzing the actual text in the url of the article, we created different bins for different popular topics categories in the url: tech brands, gossip, politics, sports, socialMedia, cars, tvshows, which were stored in 7 distinct binary additional features.

Our MetaModel: Rolling Windows

For the prediction experiments, we adopted the rolling windows scheme with a training window size of W = 10, 000 and a step size = 1000. We ordered the articles by time and at each iteration made predictions on the test set. Under this setup, each classification model is trained 30 times (iterations), producing 30 prediction sets. We then take the majority vote as the final predictions for the test data. We definied a function rolling.windows() to execute this scheme for any model of interest.

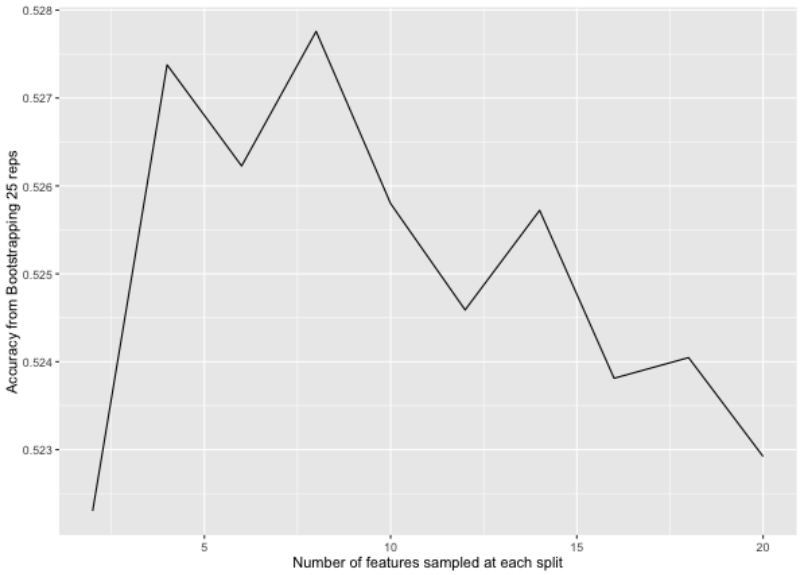


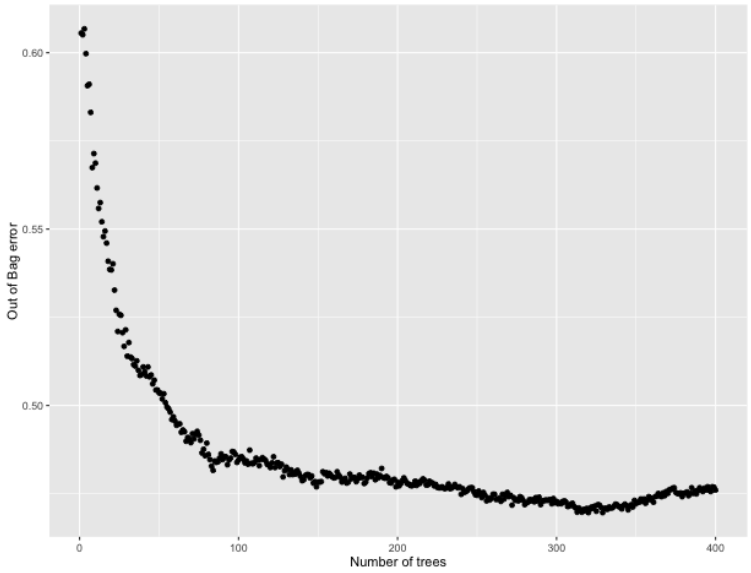


Model Selection & Tuning

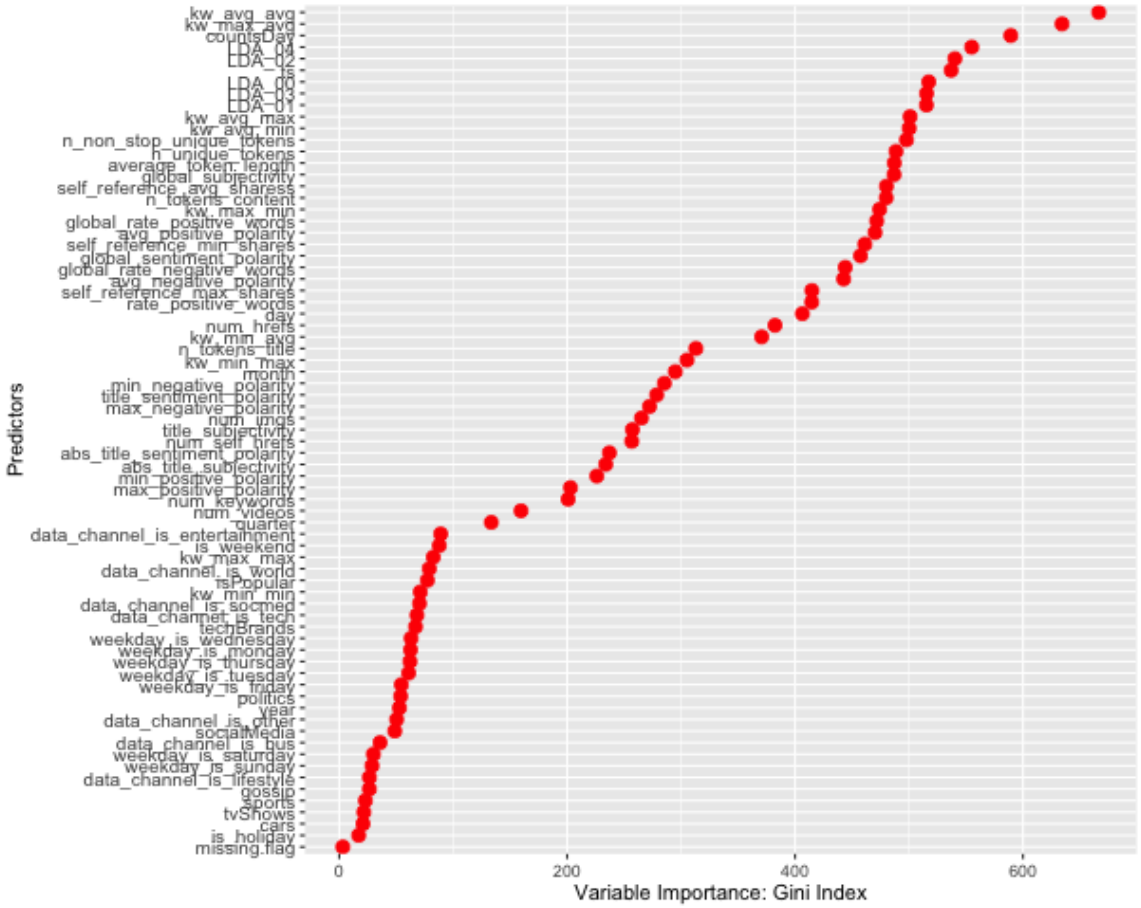
Random Forest

We tuned the parameters for the random forest model using a bootstrap approach from the caret package. In the following figures we show the results obtained.





We can also see the Variable Importance obtained from the rf model.



Text Mining

We wanted to use the keywords of the article from the url field to explore their correlations with the popularity. Our approach can be summarised in the following steps:

- First we obtain and store the list of keywords for each class in the training set.
- From the list of all keywords of the testing set, remove the top 10% most frequent words.
- Intersect the test and training lists to obtain the class-wise keywords which potentially have predictive power.
- Remove prepositions, adjectives, verbs, nouns, numbers and other frequently used words.
- Remove all common keywords across classes in order to obtain a final list of unique keywords per class.
- Obtain class predictions for the testing set if the url contains keywords belonging to one of the classes (breaking the ties in favour of higher popularity).

We tested and tuned the above approach taking random splits of the training set and choosing the one that maximized the number of common words between the out of sample and testing set.