Data Science 2 - A3 - Online News Popularity

Son N. Nguyen

17 April 2022

Introduction

This notebook was used to take part in a Kaggle competition hosted by CEU for the Data Science 2. - ML Tools class.

Initialize standard plot theme

I have created my custom theme setting to apply it for most of the charts used here and avoid repeated codes.

```
standard_theme <- function() {
    theme_bw() + theme(plot.title = element_text(size = 6,
        face = "bold", hjust = 0.5, color = "#2ca25f"),
        plot.subtitle = element_text(size = 4,
            hjust = 0.5), axis.title = element_text(size = 6),
        axis.text = element_text(size = 6),
        plot.background = element_blank())
}
```

About the dataset

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 in social networks (coded by the variable "is_popular").

The articles were published by Mashable and their content as the rights to reproduce it belongs to them. Hence, this dataset does not share the original content but some statistics associated with it. Acquisition date: January 8, 2015.

Requirements

> I expect you to experiment with all the methods we covered: linear models, random forest, gradient boosting, neural nets + parameter tuning, feature engineering, stacking.
>
> You should also submit to Moodle the documentation (Rmd and pdf) of the work you did, including exploratory data analysis, data cleaning, parameter tuning, and evaluation - with concise explanations.

Setup

I have used a handful of packages here:

- For EDA: tidyverse, modelsummary, reshape2, ggpubr
- For Markdown: kableExtra
- For ML predictions: glmnet, ranger, xgboost, keras
- For AutoML: h2o

```
# Loading packages with pacman
if (!require("pacman")) {
    install.packages("pacman")
}

pacman::p_load(tidyverse, modelsummary, glmnet,
    ranger, xgboost, kableExtra, keras, reshape2,
    ggpubr, h2o, BBmisc)
```

First, lt's get the data in. The work and holdout set was downloaded to my local folder from the Kaggle competition page.

I have set aside the holdout set which should be only used when making predictions.

```
# got the data from kaggle
work <- read_csv("../data/online_news/train.csv")
holdout <- read_csv("../data/online_news/test.csv")
head(work)
```

```
## # 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>, ...
```

Exploratory Data Analysis

Descriptive Statistics

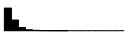
There is an overwhelming number of variables from which I aim to run the EDA on a certain sample.

No discrepancies can be recognized from the chosen metrics. It can be seen that along these verticals there are is not missing data, and the distributions, ranges seem reasonable with notable extreme values. Extreme values here are welcomed here and could not be considered as errors to be removed.

One thing to consider later is that we might want to drop record which do not have a content in their body as well as stopwords. These are not meaningfull in our analysis and only act as noises.

```
# for numerics
cols <- c("n_tokens_title", "n_tokens_content",
    "n_non_stop_words", "num_hrefs", "num_imgs",
    "num_videos", "average_token_length",
    "num_keywords", "global_sentiment_polarity",
    "global_rate_positive_words", "global_rate_negative_words")

num_descr <- work[cols]

datasummary_skim(num_descr, title = "Summary Metrics of Selected Numerics") %>%
    kableExtra::kable_styling(latex_options = c("hold_position",
        "striped"), full_width = F)
```

Table 1: Summary Metrics of Selected Numerics

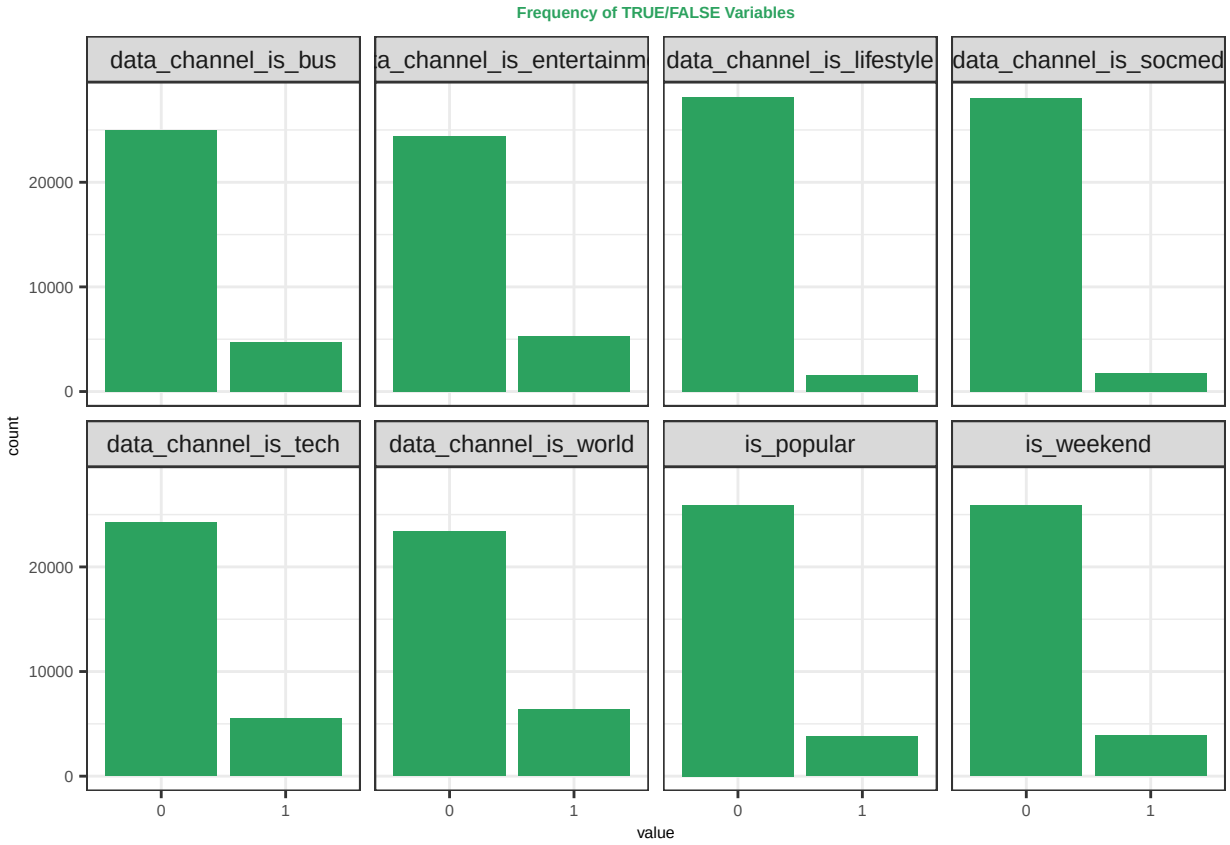
	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
n_tokens_title	19	0	10.4	2.1	2.0	10.0	20.0	
n_tokens_content	2227	0	545.7	473.5	0.0	410.0	8474.0	
n_non_stop_words	1402	0	1.0	6.0	0.0	1.0	1042.0	
num_hrefs	120	0	10.8	11.2	0.0	8.0	187.0	
num_imgs	86	0	4.5	8.3	0.0	1.0	128.0	
num_videos	50	0	1.2	4.1	0.0	0.0	75.0	
average_token_length	23602	0	4.5	0.8	0.0	4.7	8.0	
num_keywords	10	0	7.2	1.9	1.0	7.0	10.0	
global_sentiment_polarity	26399	0	0.1	0.1	−0.4	0.1	0.7	
global_rate_positive_words	11082	0	0.0	0.0	0.0	0.0	0.2	
global_rate_negative_words	8753	0	0.0	0.0	0.0	0.0	0.2	

Histograms for booleans

As per the dozens of booleans in the dataset, let's check their frequencies in the work set. We can see that the number of popular items is about one-sixth of the non-popular articles. Also, the topics are very diverse ranging from lifestyle to business and tech. Most of the articles are published on weekdays.

```
bools <- c("data_channel_is_lifestyle", "data_channel_is_entertainment",
    "data_channel_is_bus", "data_channel_is_socmed",
    "data_channel_is_tech", "data_channel_is_world",
    "is_weekend", "is_popular")

bdata <- work[bools]

bdata[] <- lapply(bdata, factor)

ggplot(gather(bdata, cols, value), aes(x = value)) +
```

```
    geom_histogram(binwidth = 20, stat = "count",
        size = 0.1, fill = "#2ca25f") + labs(title = "Frequency of TRUE/FALSE Variables") +
    standard_theme() + facet_wrap(. ~ cols,
    ncol = 4)
```

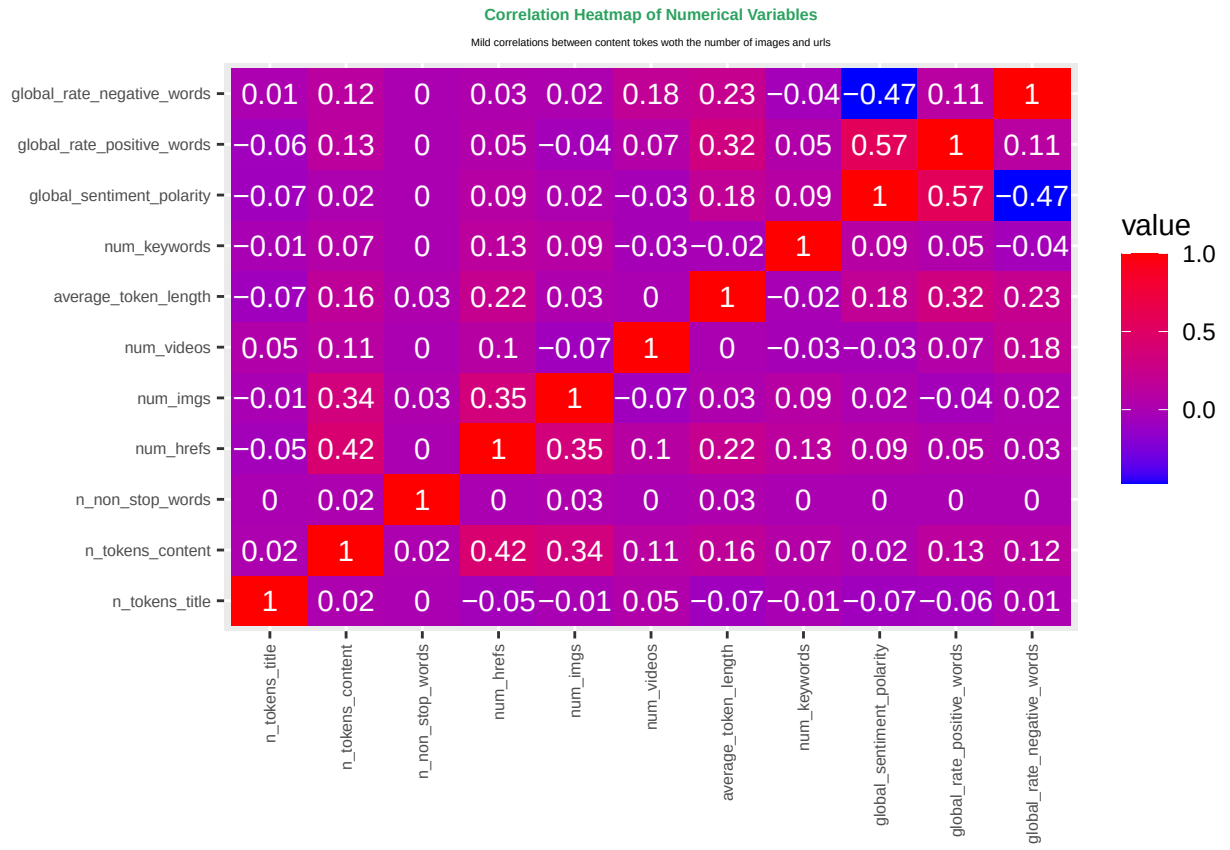


Correlation Plot

Porality is driven by the number of positive and negative words, so the correlation is not surprise. Also it seems that the number of images have a positive relationship with the count of url references in the article. The number of tokens in the article goes sligthly hand in hand with the images and references.

```
descr.cor = round(cor(num_descr, method = c("pearson")),
    2)

# Get some colors
melted_corr_mat <- melt(descr.cor)

# plotting the correlation heatmap
ggplot(data = melted_corr_mat, aes(x = Var1,
    y = Var2, fill = value)) + geom_tile() +
    geom_text(aes(Var2, Var1, label = value),
        color = "white", size = 4) + theme(plot.title = element_text(size = 6,
    face = "bold", hjust = 0.5, color = "#2ca25f"),
    plot.subtitle = element_text(size = 4,
```

```
        hjust = 0.5), axis.text.x = element_text(angle = 90,
        vjust = 0.5, hjust = 1), axis.title.x = element_blank(),
    axis.title.y = element_blank(), axis.text = element_text(size = 6),
    plot.background = element_blank()) +
    scale_fill_gradient(low = "blue", high = "red") +
    labs(title = "Correlation Heatmap of Numerical Variables",
        subtitle = "Mild correlations between content tokes woth the number of images and urls")
```



Bivariate Analysis

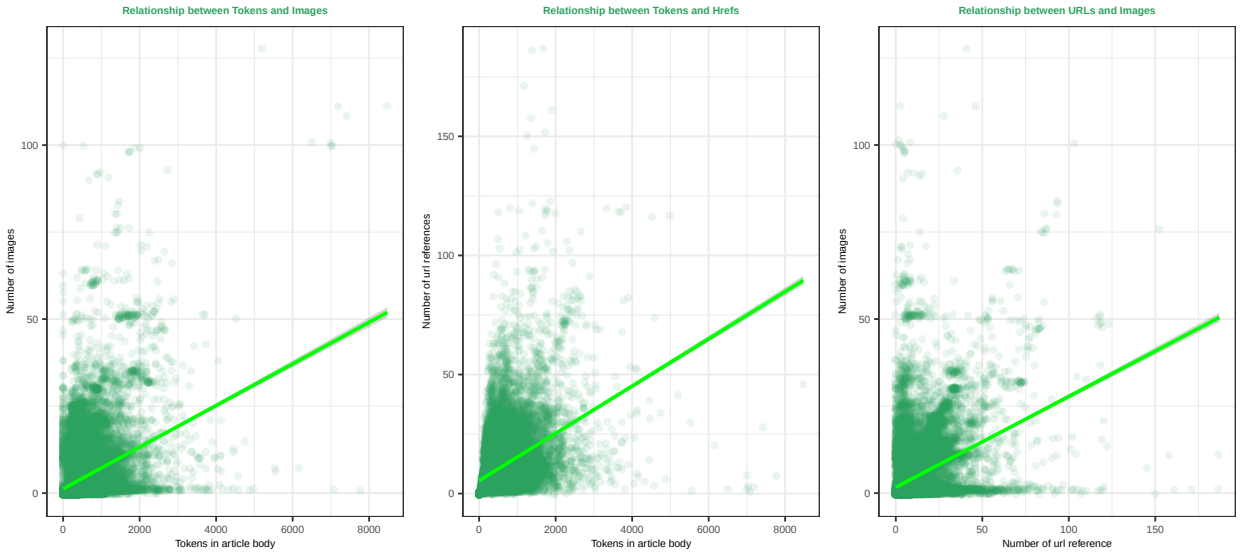
Let's look at the correlations mentioned above more closely

The variation between datapoints are all similar across combinations. The more we want to say the more images and external refereneces we have to include to the article to make it less dry and enjoyable to the reader.

```
# generate scatterplot

h <- ggplot(num_descr, aes(x = n_tokens_content,
    y = num_hrefs)) + geom_jitter(alpha = 0.1,
    col = "#2ca25f") + geom_smooth(method = "lm",
    col = "green") + labs(title = "Relationship between Tokens and Hrefs",
    y = "Number of url references", x = "Tokens in article body") +
    standard_theme()
```

```
i <- ggplot(num_descr, aes(x = n_tokens_content,
    y = num_imgs)) + geom_jitter(alpha = 0.1,
    col = "#2ca25f") + geom_smooth(method = "lm",
    col = "green") + labs(title = "Relationship between Tokens and Images",
    y = "Number of images", x = "Tokens in article body") +
    standard_theme()

k <- ggplot(num_descr, aes(x = num_hrefs,
    y = num_imgs)) + geom_jitter(alpha = 0.1,
    col = "#2ca25f") + geom_smooth(method = "lm",
    col = "green") + labs(title = "Relationship between URLs and Images",
    y = "Number of images", x = "Number of url reference") +
    standard_theme()

ggarrange(i, h, k, ncol = 3)
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



Prepare

Duplicates and incomplete records

I've decided to check whether there are duplicates or incomplete rows in our dataset. Turns out that there are no duplicates nor missing values.

```
# remove duplicated rows if exist
work <- work |>
    distinct()

# inspect missing values
work[!complete.cases(work), ]
```

```
## # A tibble: 0 x 61
## # ... with 61 variables: timedelta <dbl>, n_tokens_title <dbl>,
## #   n_tokens_content <dbl>, n_unique_tokens <dbl>, n_non_stop_words <dbl>,
## #   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>, ...
```

Articles without body text

Like mentioned before, some of the articles have zero tokens, images, hrefs, in their content body which can generate noise. We can drop these records to reform our definition that "online news" are the ones which have textual content. With that criteria, I found 883 records without a token in the content.

```
# find number of rows that contain 0
# for n_tokens_content
nrow(work[work["n_tokens_content"] == 0,
    ])
```

```
## [1] 883
```

```
# 883 rows with no body just title,
# drop them
work <- work[work["n_tokens_content"] !=
    0, ]
```

Noisy features

Lastly in the cleaning process, I have identified columns which were either highly dependent on each other or does not have a meaningfull sense to include them in our predictions and it would just only generate noise.

For instance, the number of unique, total tokens (which are not stop words) as well as the minimum average keyword shares are highly correlated with the total tokens (including stop words), therefore it would greatly reduce the model complecity if we would just drop them.

Also, the article_id and days between publication and data collection (timedelta) does not have a predictive sense whatsoever, so we should not include them either.

```
to_drop <- c("n_non_stop_unique_tokens",
    "n_non_stop_words", "kw_avg_min", "article_id",
    "timedelta")

# drop useless columns which highly
# correlate or is not informative
work <- select(work, -all_of(to_drop))
```

Split work data to train and validation

I have split the work data into train and validation set using a 80-20 ratio.

```
# Data partition to work and test set
# 80-20 split
set.seed(1134)
ind <- sample(nrow(work), size = floor(0.8 *
    nrow(work)))
train <- work[ind, ]
valid <- work[-ind, ]
```

Define classification loss function

I have choose accuracy as my loss function to locally test the fitted models. My strategy to thrive in this competition was to first submit predictions from individual model designs ranging from a logistic regression to neural net. Thereafter, I also wanted to submit stacked ensemble solutions including h2o's AutoML.

```
# define accuracy function
calculateAccuracy <- function(prediction,
    y_obs) {
    sum(ifelse(prediction == y_obs, 1, 0),
        na.rm = T)/length(y_obs)
}
```

Logit

The very first model I choose was a logistic regression without any particular tuning and feature engineering. The validation accuracy is decent at a 87% but when submitting but when submitting it on Kaggle, I didn't expect a good AUC coming out as a result.

```
# fit model
logit <- glm(is_popular ~ ., family = "binomial",
    data = train)

# transform fitted values and add to
# table
accuracy_results <- tibble(model = "Logit",
    train = calculateAccuracy(ifelse(logit$fitted.values <
        0.5, 0, 1), train$is_popular), validation = calculateAccuracy(ifelse(predict(logit,
        valid, type = "response") < 0.5,
        0, 1), valid$is_popular))

# create reusable table display
show_table <- function() {
    kable(x = accuracy_results, digits = 3,
        caption = "Evaluation Metrics (Accuracy)") %>%
        kable_styling(latex_options = c("hold_position",
            "striped"), font_size = 8) %>%
        add_footnote(c("Source data: Online News Popularity"))
}

show_table()
```

Table 2: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874

[a] Source data: Online News Popularity

LASSO

The next model was a LASSO regression with all of the existing features, and using 200 lambda values. The precision have not improved much but it seems to perform better than logit with a 87.5% validation accuracy.

```
formula <- as.formula(is_popular ~ .)
X = model.matrix(formula, train)[, -1]
Y = train$is_popular

# fit model
lasso <- cv.glmnet(X, Y, alpha = 1, nlambda = 200)

# transform fitted values and add to
# table
accuracy_results <- add_row(accuracy_results,
    model = "LASSO", train = calculateAccuracy(ifelse(predict(lasso,
        newx = X, s = lasso$lambda.min) <
        0.5, 0, 1), train$is_popular), validation = calculateAccuracy(ifelse(predict(lasso,
        newx = model.matrix(formula, valid)[,
            -1], s = lasso$lambda.min) <
        0.5, 0, 1), valid$is_popular))

show_table()
```

Table 3: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874
LASSO	0.873	0.875

[a] Source data: Online News Popularity

LASSO w/ feature engineering

Let's see how lasso changes if we apply some minor feature engineering. I have took the logarithmic form of variables with larger unit of dispersion. As per polynomials I wanted to emphasize the effect of multimedia in the article with a polynomial functions, thus I have created columns with the square terms of image, video, references. Also included square terms of average token length and number of keywords.

```
# copy df
lasso_fe <- train

base <- colnames(train[, -55])

# we can only paly around with absolute
```

```
# numerics, ratios and booleans are
# avoided

# logs
logs <- c("log(n_tokens_content)", "log(n_tokens_title)")

# polynomials
polynomials <- c("I(num_imgs^2)", "I(num_videos^2)",
    "I(num_hrefs^2)", "I(average_token_length^2)",
    "I(num_keywords)")

# interactions
interactions <- c("num_imgs*num_hrefs", "num_imgs*num_videos",
    "data_channel_is_lifestyle*num_imgs",
    "data_channel_is_bus*n_tokens_content",
    "data_channel_is_socmed*num_imgs")
```

Turns out that the performance haven't changed a single bit because lasso has penalized most of the additional features, and with that, I sticked to the simpler model.

```
features <- c(base, logs, polynomials, interactions)

formula <- as.formula(paste0("is_popular ~",
    paste0(features, collapse = " + ")))
X_fe = model.matrix(formula, train)[, -1]

# fit model
lasso_fe <- cv.glmnet(X_fe, Y, alpha = 1,
    nlambda = 200)

# transform fitted values and add to
# table
accuracy_results <- add_row(accuracy_results,
    model = "LASSO_FE", train = calculateAccuracy(ifelse(predict(lasso_fe,
        newx = X_fe, s = lasso_fe$lambda.min) <
        0.5, 0, 1), train$is_popular), validation = calculateAccuracy(ifelse(predict(lasso_fe,
        newx = model.matrix(formula, valid)[,
            -1], s = lasso_fe$lambda.min) <
        0.5, 0, 1), valid$is_popular))

show_table()
```

Table 4: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874
LASSO	0.873	0.875
LASSO_FE	0.873	0.875

[a] Source data: Online News Popularity

Ridge

What if we use tighter regularization? I have only changed the regularization parameter to use the ridge penalties. You can see below that it didn't change anything.

```
formula <- as.formula(is_popular ~ .)

# fit model
ridge <- cv.glmnet(X, Y, alpha = 0, nlambda = 200)

# transform fitted values and add to
# table
accuracy_results <- add_row(accuracy_results,
    model = "Ridge", train = calculateAccuracy(ifelse(predict(ridge,
        newx = X, s = ridge$lambda.min) <
        0.5, 0, 1), train$is_popular), validation = calculateAccuracy(ifelse(predict(ridge,
        newx = model.matrix(formula, valid)[,
            -1], s = ridge$lambda.min) <
        0.5, 0, 1), valid$is_popular))

show_table()
```

Table 5: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874
LASSO	0.873	0.875
LASSO_FE	0.873	0.875
Ridge	0.873	0.876

[a] Source data: Online News Popularity

Random Forest

Turning our heads to the ensemble models without a closed form solution, my first idea was to use a random forest ensemble. I have slightly tuned it to fit 300 classification trees with 8 variables to split by at each iteration. I have chose 8 because it is a common best practice to split by the square root of the total column in the data. Setting the `mtry` parameter low would mean that we have a larger chance to miss out on a lot of variation from a particular dimension ignored by the split.

This has produced the best validation accuracy by a negligible margin, while there is a pattern of overfitting on the training set. Fortunately, we don;t have to do feature engineering in tree-based ensemble algorithms because the machine manually figures out all the polynomials and interactions by itself.

The very first individual model I submitted was the Random Forest model which produced a test AUC around 0.68, putting me at the bottom of the leaderboard.

```
rf <- ranger(is_popular ~ ., num.trees = 300,
    mtry = 8, data = train)

accuracy_results <- add_row(accuracy_results,
    model = "Random Forest", train = calculateAccuracy(ifelse(predict(rf,
        train)$predictions < 0.5, 0, 1),
        train$is_popular), validation = calculateAccuracy(ifelse(predict(rf,
```

```
        valid)$predictions < 0.5, 0, 1),
        valid$is_popular))

show_table()
```

Table 6: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874
LASSO	0.873	0.875
LASSO_FE	0.873	0.875
Ridge	0.873	0.876
Random Forest	0.999	0.876

[a] Source data: Online News Popularity

```
# predict holdout
article_ids <- holdout$article_id
to_drop <- c("n_non_stop_unique_tokens",
    "n_non_stop_words", "kw_avg_min", "article_id",
    "timedelta")

# drop useless columns which highly
# correlate or is not informative

sub <- select(holdout, -all_of(to_drop))

rf_score <- predict(rf, data = sub)$predictions

rf_submission <- data.frame(article_id = article_ids,
    score = rf_score)

write_csv(rf_submission, "./submissions/rf_submission.csv")
```

XGBoost

Moving on, I have expected XGBoost to perform better than Random Forest as it is a very effective algorithm widely used in Kaggle competitions. In terms of computation it is very optimized and the construction of 300 trees was relatively faster compared to GBM models.

In terms of validation accuracy it got worse, but I managed to push down the effect of overfitting.

Submitting this solution yielded an AUC around 0.706 which still kept me at the bottom of the leaderboard at that time.

```
#define predictor and response variables in training set
train_x <- data.matrix(train[, 1:55])
train_y <- train$is_popular

#define predictor and response variables in testing set
valid_x <- data.matrix(valid[, 1:55])
valid_y <- valid$is_popular

#define final training and testing sets in an xgb matrix
```

```
xgb_train <- xgb.DMatrix(data = train_x, label = train_y)
xgb_valid <- xgb.DMatrix(data = valid_x, label = valid_y)

# train a model using our training data
xgb <- xgboost(data = xgb_train,
                 max.depth = 7,
                 nround = 300,
                 eta = 0.05, #learning rate
                 eval_metric = 'auc',
                 scale_pos_weight = 6,
                 early_stopping_rounds = 10,
                 objective = "binary:logistic",
                 verbose = FALSE)

accuracy_results <- add_row(accuracy_results,
    model = "XGBoost",
    train = calculateAccuracy(ifelse(predict(xgb, xgb_train) < 0.5, 0, 1), train$is_popular),
    validation = calculateAccuracy(ifelse(predict(xgb, xgb_valid) < 0.5, 0, 1), valid$is_popular)
)

show_table()
```

Table 7: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874
LASSO	0.873	0.875
LASSO_FE	0.873	0.875
Ridge	0.873	0.876
Random Forest	0.999	0.876
XGBoost	0.939	0.795

[a] Source data: Online News Popularity

```
xgb_score <- predict(xgb, xgb.DMatrix(as.matrix(sub)))

xgb_submission <-data.frame(article_id = article_ids, score = xgb_score)

write_csv(xgb_submission, "./submissions/xgb_submission.csv")
```

Neural Nets

Among the last individual models, I chose to implement a neural network model with 3 hidden layers, equipped with ReLU activation functions and a dropout layer after each. The output layer is has a sigmoid activation function with only one node for classification. I have added an early stopping callback which monitors the validation loss and stops the fitting if it has not decreased after three epochs. The checkpoint callback ensures that we roll back the model to the one which has the least validation loss. All in all, my network has more than six thousand weight and bias parameters to train.

```
# Separate x & rescale
data_train_x <- as.matrix(select(train, -is_popular))
data_valid_x <- as.matrix(select(valid, -is_popular))
```

```r
# Separate y & one-hot encoding
data_train_y <- train$is_popular
data_valid_y <- valid$is_popular

news_nn <- keras_model_sequential()
```

```
## Loaded Tensorflow version 2.8.0
```

```r
news_nn %>%
    layer_dense(units = 64, activation = "relu",
        input_shape = c(ncol(data_train_x))) %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 32, activation = "relu") %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 1, activation = "sigmoid")

# define early stopping callback
early_stop <- keras::callback_early_stopping(monitor = "val_loss",
    patience = 3)

# define checkpoint callback
checkpoint <- keras::callback_model_checkpoint(filepath = "models/online_news_nn.h5",
    monitor = "val_loss", save_best_only = TRUE)

summary(news_nn)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## =========================================================================
##  dense_3 (Dense)                    (None, 64)                       3584
##
##  dropout_2 (Dropout)                (None, 64)                       0
##
##  dense_2 (Dense)                    (None, 32)                       2080
##
##  dropout_1 (Dropout)                (None, 32)                       0
##
##  dense_1 (Dense)                    (None, 16)                       528
##
##  dropout (Dropout)                  (None, 16)                       0
##
##  dense (Dense)                      (None, 1)                        17
##
## =========================================================================
## Total params: 6,209
## Trainable params: 6,209
## Non-trainable params: 0
## _________________________________________________________________________
```

I have used binary crossentropy as a loss function and the Adam optimizer as a gradient descent strategy. I have set the learning rate to be in the lower regions due to the size of the dataset.

I have fitted the model in 15 epoch and a batch size of 30.

```
news_nn |>
    compile(loss = "binary_crossentropy",
        optimizer = optimizer_adam(learning_rate = 1e-05),
        metrics = c("accuracy"))

# history <- fit( news_nn,
# data_train_x, data_train_y, epochs =
# 15, batch_size = 30, validation_data
# = list(data_valid_x, data_valid_y),
# callbacks = c(early_stop, checkpoint)
# ) plot(history)

best_nn <- keras::load_model_hdf5("models/online_news_nn.h5")
```

Putting the NN model in comparison with the precious ones, you can see that the train and validation accuracy matches that of the LASSO and Ridge.

```
accuracy_results <- add_row(accuracy_results,
    model = "Neural Net", train = calculateAccuracy(ifelse(predict(best_nn,
        data_train_x, batch_size = 30) <
        0.5, 0, 1), data_train_y), validation = calculateAccuracy(ifelse(predict(best_nn,
        data_valid_x, batch_size = 30) <
        0.5, 0, 1), data_valid_y))

show_table()
```

Table 8: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874
LASSO	0.873	0.875
LASSO_FE	0.873	0.875
Ridge	0.873	0.876
Random Forest	0.999	0.876
XGBoost	0.939	0.795
Neural Net	0.873	0.875

[a] Source data: Online News Popularity

Submitting the Neural Net predictions yielded a staggeringly poor score slightly above 0.5. It seems that Neural net models are not really optimal on small sized structured datasets.

```
test_nn <- as.matrix(sub)

nn_score <- predict(best_nn, test_nn, batch_size = 30,
    verbose = 1)

nn_submission <- data.frame(article_id = article_ids,
    score = nn_score)

write_csv(nn_submission, "./submissions/nn_submission.csv")
```

Stacking

Among the last models, I have bundled all the models except NN together and took the mean of every prediction. This yield a slightly better validation accuracy and tops all other models.

```
# Validation accuracies

logit_vscore <- predict(logit, valid, type = "response")
lasso_vscore <- c(predict(lasso, newx = model.matrix(is_popular ~
    ., valid)[, -1], s = lasso$lambda.min))
ridge_vscore <- c(predict(ridge, newx = model.matrix(is_popular ~
    ., valid)[, -1], s = ridge$lambda.min))
rf_vscore <- predict(rf, data = valid)$predictions
xg_vscore <- predict(xgb, xgb_valid)

# average out predictions
stack_vscore <- (logit_vscore + lasso_vscore +
    ridge_vscore + rf_vscore + xg_vscore)/5

accuracy_results <- add_row(accuracy_results,
    model = "Stacking", validation = calculateAccuracy(ifelse(stack_vscore <
        0.5, 0, 1), valid$is_popular))

show_table()
```

Table 9: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874
LASSO	0.873	0.875
LASSO_FE	0.873	0.875
Ridge	0.873	0.876
Random Forest	0.999	0.876
XGBoost	0.939	0.795
Neural Net	0.873	0.875
Stacking	NA	0.876

[a] Source data: Online News Popularity

Submitting it to Kaggle, this yields a test AUC around 71.6 which moved me to the mid-top half of the leaderboard. It turned out that stacking is a good strategy and all the strengths of each algorithm improved the overall performance.

```
# Submission on holdout set

logit_score <- predict(logit, sub, type = "response")
lasso_score <- c(predict(lasso, newx = model.matrix(~.,
    sub)[, -1], s = lasso$lambda.min))
ridge_score <- c(predict(ridge, newx = model.matrix(~.,
    sub)[, -1], s = ridge$lambda.min))
rf_score <- predict(rf, data = sub)$predictions
xg_score <- predict(xgb, xgb.DMatrix(as.matrix(sub)))

# average out predictions
stack_score <- (logit_score + lasso_score +
```

```
    ridge_score + rf_score + xg_score)/5

stacked <- data.frame(article_id = article_ids,
    score = stack_score)

write_csv(stacked, "./submissions/stacked_submission.csv")
```

H2O AutoML

We have seen that stacking is a good direction but we have not aggressively tuned our models. Luckily h2o takes care of the parameter tuning and stacking both so we can sit back and watch its choosing the best model.

Initially I have fitted a maximum of 20 models (excluding stacked ensemble) with a 5-fold CV and limit of 3 minutes and balanced classes so we undersample majority or oversample minory classes. Since XGBoost is not available on ARM-based Mac, it only builds (Distrubuted and Extreme) RF, GLM, GBM, and fully-connected Deep Learning models. The algorithm chose a stacked ensemble model for prediction which had a majority of GBM models with different parametrisations.

I have also experiment with increasing, the number of models and the time to learn but it did not yield better results due to either over or underfitting.

```
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         3 hours 14 minutes
##     H2O cluster timezone:       Europe/Budapest
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.3
##     H2O cluster version age:    2 months
##     H2O cluster name:           H2O_started_from_R_nszoni_jhm757
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.31 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.1 (2021-08-10)
```

```
# import files in h2o format
workh <- h2o.importFile(path = "../data/online_news/train.csv")
```

```
##   |                                                                      |
```

```
holdouth <- h2o.importFile(path = "../data/online_news/test.csv")
```

```
##   |                                                                      |
```

```
# convert outcome to factor
workh["is_popular"] <- as.factor(workh["is_popular"])

# split dataframe
my_seed <- 1134
news_data_splits <- h2o.splitFrame(data = workh,
    ratios = 0.8, seed = my_seed)
trainh <- news_data_splits[[1]]
validh <- news_data_splits[[2]]

# set the predictor and response
# columns
response <- "is_popular"
predictors <- setdiff(names(trainh), c(response,
    "article_id"))

# fit automl automl <- h2o.automl( x =
# predictors, y = response,
# training_frame = trainh,
# validation_frame = validh,
# sort_metric = 'AUC', balance_classes
# = TRUE, preprocessing =
# c('target_encoding'), seed = my_seed,
# max_models = 20, max_runtime_secs =
# 60*3 # limit the run-time )

# best_auto_ml <-
# h2o.get_best_model(automl)

# model_path <-
# h2o.saveModel(best_auto_ml,path=getwd(),
# force=TRUE, filename =
# 'automl_no_feature_scaling')

best_auto_ml <- h2o.loadModel(path = paste0(getwd(),
    "/automl_no_feature_scaling"))

best_auto_ml@model[["model_summary"]]
```

```
## Number of Base Models: 15
##
## Base Models (count by algorithm type):
##
## deeplearning          drf          gbm          glm
##            2            2           10            1
##
## Metalearner:
##
## Metalearner algorithm: glm
## Metalearner cross-validation fold assignment:
##   Fold assignment scheme: AUTO
##   Number of folds: 5
##   Fold column: NULL
```

```
## Metalearner hyperparameters:
```

h2o's AutoML seems to outperform all the previous models in terms of validation accuracy.

```
# Validation accuracy

accuracy_results <- add_row(accuracy_results,
    model = "h2o AutoML", train = calculateAccuracy(ifelse(array(h2o.predict(best_auto_ml,
        trainh)[[3]]) < 0.5, 0, 1), train$is_popular),
    validation = calculateAccuracy(ifelse(array(h2o.predict(best_auto_ml,
        validh)[[3]]) < 0.5, 0, 1), valid$is_popular))
```

```
##   |                                                                      |
##   |                                                                      |
```

```
show_table()
```

Table 10: Evaluation Metrics (Accuracy)

model	train	validation
Logit	0.872	0.874
LASSO	0.873	0.875
LASSO_FE	0.873	0.875
Ridge	0.873	0.876
Random Forest	0.999	0.876
XGBoost	0.939	0.795
Neural Net	0.873	0.875
Stacking	NA	0.876
h2o AutoML	0.890	0.886

[a] Source data: Online News Popularity

The outcome has been the best as I have managed to improve the AUC score on the test set to 0.724 putting me at the first place on the leaderboard (2022-04-15)!

```
# Submission

aml_score <- array(h2o.predict(best_auto_ml,
    holdouth)[[3]])
```

```
##   |                                                                      |
```

```
aml_submission <- data.frame(article_id = article_ids,
    score = aml_score)

write_csv(aml_submission, "./submissions/aml_submission.csv")
```

H2O AutoML with Feature Scaling

Waking up the next day, I have sadly seen that I dropped to the second place by a fine margin. I I have tried out different hyperparametrizations to improve the precision but got no luck in any of the methods to significantly improve my results.

As a last resort, I wanted to see how the algo performs on a standardized data, so I applied feature scaling on the train, validation and holdout set separately.

```
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         3 hours 26 minutes
##     H2O cluster timezone:       Europe/Budapest
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.3
##     H2O cluster version age:    2 months
##     H2O cluster name:           H2O_started_from_R_nszoni_jhm757
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.13 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.1 (2021-08-10)
```

```
# Convert to factors
cols <- colnames(train)[grepl(".*is_.*",
    colnames(train))]
train[cols] <- lapply(train[cols], factor)
valid[cols] <- lapply(valid[cols], factor)

# Feature scaling
train_scl <- train |>
    mutate_if(is.numeric, scale) |>
    mutate_if(is.numeric, as.vector)

valid_scl <- valid |>
    mutate_if(is.numeric, scale) |>
    mutate_if(is.numeric, as.vector)

# cast to h2o frame
trainh <- as.h2o(train_scl)
```

```
##   |                                                                      |
```

```
validh <- as.h2o(train_scl)
```

```
##   |                                                                      |
```

```
# set the predictor and response
# columns
response <- "is_popular"
predictors <- setdiff(names(trainh), c(response,
    "article_id"))
```

```
# fit automl automl <- h2o.automl( x =
# predictors, y = response,
# training_frame = trainh,
# validation_frame = validh,
# sort_metric = 'AUC', balance_classes
# = TRUE, preprocessing =
# c('target_encoding'), seed = my_seed,
# max_models = 20, max_runtime_secs =
# 60*3 # limit the run-time )

# best_auto_ml <-
# h2o.get_best_model(automl)

# model_path <-
# h2o.saveModel(best_auto_ml,path=getwd(),
# force=TRUE, filename =
# 'automl_feature_scaling')

best_auto_ml <- h2o.loadModel(path = paste0(getwd(),
    "/automl_feature_scaling"))

best_auto_ml@model[["model_summary"]]
```

```
## Number of Base Models: 16
##
## Base Models (count by algorithm type):
##
## deeplearning          drf          gbm          glm
##            3            2           10            1
##
## Metalearner:
##
## Metalearner algorithm: glm
## Metalearner cross-validation fold assignment:
##   Fold assignment scheme: AUTO
##   Number of folds: 5
##   Fold column: NULL
## Metalearner hyperparameters:
```

```
# scale holdout

holdout[cols[1:14]] <- lapply(holdout[cols[1:14]],
    factor)

holdout_scl <- holdout |>
    mutate_if(is.numeric, scale) |>
    mutate_if(is.numeric, as.vector)

holdouth <- as.h2o(holdout_scl)
```

```
##   |                                                                      |
```

```
# Submission

aml_score <- array(h2o.predict(best_auto_ml,
    holdouth)[[3]])
```

```
##   |                                                                      |
```

```
aml_submission <- data.frame(article_id = article_ids,
    score = aml_score)

write_csv(aml_submission, "./submissions/aml_fe_submission.csv")
```

It turned out later that feature scaling did not help augmenting the external validity of the model. The next step would be to perform an aggressive grid search to get the best parameter tuning or gather more data in order to apply more complex models.