DS - Assignment 3

Oszkar Egervari

2022-04-19

Introduction

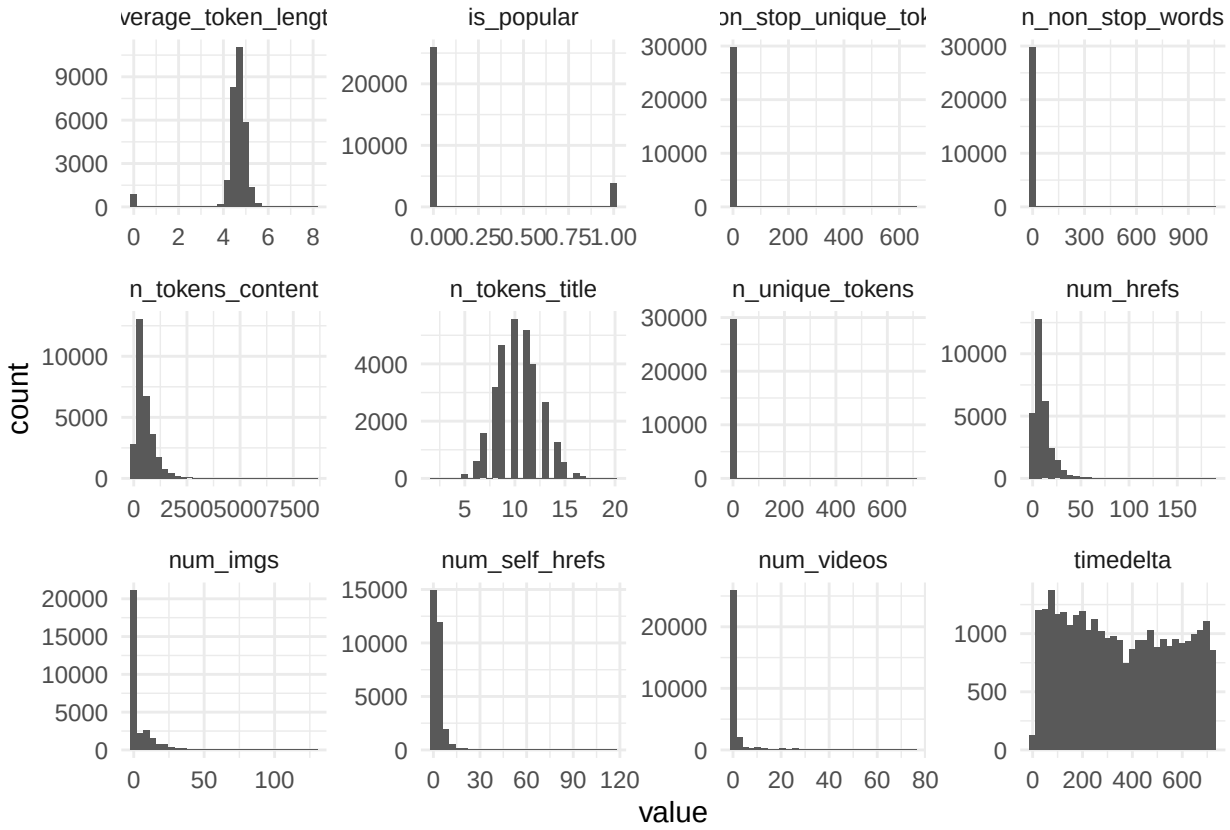
In this assignment I'm going to showcase the models I for our in class Kaggle competition. The data contains articles published by Mashable. The task was to predict whether they were popular based on social media shares. More can be read about the data on the Kaggle page.

Exploring the data

I explored the variables with the plot below. For lack of space I only display twelve plots, however I explored all variables.

```
# skimr::skim(train)
# cor(train[1:60], train$is_popular)

ggplot(gather(train[c(1:11, 60)], cols, value), aes(value)) +
  geom_histogram() +
  facet_wrap(.~cols, scales = "free") +
  theme_minimal()
```



Feature Engineering

Feature engineering is an important step of the process in order to maximize the prediction accuracy. First I removed the outlier values, then I transformed the binary variables to factors. (A side note here, first I did the modeling without this step, then after transforming these variables to factors, and it didn't seem to effect the accuracy of the models.)

```
# removing outlier values
train <- train %>% filter(n_unique_tokens <= 1)

# variables to factor
factors <- train %>% select(contains("data_channel")|contains("week")) %>% names()
train[factors] <- lapply(train[factors], factor)

# check the class of the variables
# sapply(train, class)
```

LASSO

The first model I run was a LASSO model. I decided to skip a linear regression model, because of the large number of variables. This was surprisingly one of the best models in terms of accuracy on the public test set.

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

features <- model.matrix(~., train[predictors])
outcome <- train$is_popular

lasso <- cv.glmnet(features,  outcome, alpha = 1)
#plot(lasso)
lasso_pred <- predict(lasso, newx = model.matrix(~., test[predictors]), type = "class", s = lasso$lambda

# kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = lasso_pred[,1]) %>%
#   write.csv("../AS3/models/lasso.csv", row.names = F)
```

PCA

The principal component analysis model performed around the same accuracy as the LASSO model.

```
load("/Users/oszkar/Documents/CEU/Winter/DS/AS3/models/pca.RData")
# pcr_model <- pcr(outcome ~ features[,-1], scale = TRUE)
#summary(pcr_model)

pca_pred <- predict(pcr_model, newdata = as.matrix(test[, predictors]), ncomp = 59)

# kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = pca_pred) %>%
#   write.csv("../AS3/models/pca.csv", row.names = F)
```

Tree model

The tree model performs the worst so far. I used rpart with the default number trees, which is 500.

```
load("/Users/oszkar/Documents/CEU/Winter/DS/AS3/models/tree.RData")

# train[factors] <- lapply(train[factors], as.numeric) # I had to transform factor variables back to nu
# in order for the tree model to run
#
# tree_model <- rpart(
#   formula(paste0("is_popular ~ ", paste(predictors, collapse = "+"))),
#   train
#   )


tree_pred <- predict(tree_model, newdata = test)

# save(tree_model, file = "/Users/oszkar/Documents/CEU/Winter/DS/AS3/models/tree.RData")
```

```
# data.frame(tree_pred)
# as.array(tree_pred)
#
#
# # kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = tree_pred) %>%
#   write.csv("../AS3/models/tree.csv", row.names = F)
#
# train[factors] <- lapply(train[factors], factor)
```

Random Forest

The RF model achieved the highest accuracy so far on the test set. I used ranger with the default settings for the number of trees, which is 500.

```
load("/Users/oszkar/Documents/CEU/Winter/DS/AS3/models/simple_rf.RData")
# set.seed(my_seed)
# simple_rf <- ranger(
#   formula(paste0("is_popular ~ ", paste(predictors, collapse = "+"))),
#   train
# )

#save(simple_rf, file = "../AS3/models/simple_rf.RData")

rf_pred <- predict(simple_rf, test)
#rf_pred$predictions
# kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = rf_pred$predictions) %>%
#   write.csv("../AS3/models/rf.csv", row.names = F)
```

GBM

I have achieved the highest accuracy on the public test set with GBM using 5000 trees and interaction depth of 6. I tried it with the default setting of 100 trees and interaction depth of 4, the results were slightly less accurate with the latter settings.

```
load("../AS3/models/gbm.RData")
# gbm <- gbm(
#   formula(paste0("is_popular ~ ", paste(predictors, collapse = "+"))),
#   data = train,
#   n.trees = 5000,
#   distribution = "gaussian",
#   cv.folds = 5,
#   shrinkage = 0.1,
#   interaction.depth = 6
# )
#
gbm_pred <- predict(gbm, test)
```

```
# kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = gbm_pred) %>%
#   write.csv("../AS3/kaggle_submission_gbm.csv", row.names = F)
```

Models with H2O

For the rest of the models I chose to use H2O. The reason for that is its simplicity (also I used keras for my previous assignment and I wanted to try H2O as well). The other reason is that I wanted to run H2O's automl model. I did that, achieved mediocre results compared to the other models. However I decided to skip it here, because when I tried to run it most recently, I received an error.

```
h2o <- as.h2o(train)
```

```
##   |                                                                              |
```

```
validation_h2o <- as.h2o(test)
```

```
##   |                                                                              |
```

```
h2o_data_splits <- h2o.splitFrame(data =  h2o, ratios = 0.8, seed = my_seed)
train_h2o <- h2o_data_splits[[1]]
test_h2o <- h2o_data_splits[[2]]
```

I split the train set so the models would have a validation set. Previously - not included in this markdown - I run the models without the "validation_frame", the results were about the same.

XGBoost

XGBoost was my second most accurate model, achieving almost the same accuracy as GBM on the public test set. I used it with different settings, 200, the default 500 and a 1000 trees with pretty much the same results.

```
load("/Users/oszkar/Documents/CEU/Winter/DS/AS3/models/xgboost2.RData")

# simple_xgboost <- h2o.xgboost(
#   x = predictors, y = response,
#   model_id = "simple_xgboost",
#   training_frame = train_h2o,
#   validation_frame = test_h2o,
#   nfolds = 5,
#   max_depth = 2, min_split_improvement = 0.1,
#   learn_rate = 0.05, ntrees = 1000,
#   score_each_iteration = TRUE,
#   seed = 20220412
# )

#h2o.performance(simple_xgboost, xval = T)

xgb_pred <- predict(simple_xgboost, validation_h2o, type = 'prob')
```

```
##   |                                                                      |
```

```
xgb_pred <- as.data.frame(xgb_pred)

# kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = xgb_pred[,1]) %>%
#   write.csv("../AS3/models/xgboost2.csv", row.names = F)
```

Default DL

The first deep learing model I ran was the default H2O model. This has two hidden layers with 200 nodes each. The results were a significant decrease in accuracy on the public test set, compared to the most accurate models.

```
load("../AS3/models/dl_default.RData")
# dl_default <- h2o.deeplearning(
#   x = predictors,
#   y = response,
#   training_frame = train_h2o,
#   validation_frame = test_h2o,
#   model_id = "dl_default",
#   seed = my_seed)

#
# h2o.scoreHistory(dl_default)
# plot(dl_default, metric = "deviance")

dl_default_pred <- h2o.predict(dl_default, validation_h2o)
```

```
##   |                                                                      |
```

```
dl_default_pred <- as.data.frame(dl_default_pred)

# kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = dl_default_pred[,1]) %>%
#   write.csv("../AS3/kaggle_submission_dl_default.csv", row.names = F)
```

Regularized DL

In the next DL model I ran I increased the nodes to 400 each and implemented dropout to combat overfitting. The models accuracy increased by 3% compared to the default DL model.

```
load("../AS3/models/dl_regularized.RData")
# dl_regularized <- h2o.deeplearning(
#   x = predictors,
#   y = response,
#   training_frame = train_h2o,
#   validation_frame = test_h2o,
```

```
#   model_id = "dl_regularized",
#   hidden = c(400, 400),
#   mini_batch_size = 20,
#   activation = "RectifierWithDropout",
#   hidden_dropout_ratios = c(0.2, 0.2),
#   epochs = 300,
#   score_each_iteration = TRUE,
#   seed = my_seed
# )
#save(dl_regularized, file = "../AS3/models/dl_regularized.RData")
#
# h2o.scoreHistory(dl_regularized)
# plot(dl_regularized, metric = "deviance")

dl_regularized_pred <- h2o.predict(dl_regularized, validation_h2o)
```

```
##   |                                                                      |
```

```
dl_regularized_pred <- as.data.frame(dl_regularized_pred)

# kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = dl_regularized_pred[,1]) %>%
#   write.csv("../AS3/kaggle_submission_dl_regularized.csv", row.names = F)
```

Deep DL

In the last model I increased the number of hidden layers to four, regularization measures for all layers. The accuracy remained about the same as the last model.

```
load("../AS3/models/dl_deep.RData")
# dl_deep <- h2o.deeplearning(
#   x = predictors,
#   y = response,
#   training_frame = train_h2o,
#   validation_frame = test_h2o,
#   model_id = "dl_deep",
#   hidden = c(256, 128, 64, 32),
#   mini_batch_size = 20,
#   activation = "RectifierWithDropout",
#   hidden_dropout_ratios = c(0.2, 0.2, 0.2, 0.2),
#   epochs = 300,
#   score_each_iteration = TRUE,
#   seed = my_seed
# )

#save(d, file = "../AS3/models/dl_deep.RData")

#
# h2o.scoreHistory(dl_deep)
# plot(dl_deep, metric = "deviance")
```

```
dl_deep_pred <- h2o.predict(dl_deep, validation_h2o)
```

```
##   |                                                                      |
```

```
dl_deep_pred <- as.data.frame(dl_deep_pred)

# kaggle sumbission
# select(test, article_id) %>%
#   mutate(score = dl_deep_pred[,1]) %>%
#   write.csv("../AS3/models/dl_deep.csv", row.names = F)
```

Conclusion

After running all the models above (and many other similar, but slightly different ones), we can see, that the supervised learning models, GBM and XGBoost, easily outperformed the unsupervised learning models. I think the main reason for this could be, that we were lucky to have a large training set of labeled data. The other interesting discovery was that the linear models outperformed the deep learning models. I'm not sure why this was the case, but it could be that the DL models I ran were too simple and had I ran more complex models, the results could have been different.