Supervised learning models

Cornelius Erfort

9 Aug 2021

Contents

1 Summary

In this script, we test the performance of a series of classifiers across a diverse set of parameter configurations and for two different dfm representations. For most classifiers, the tfidf-transformed dfm with bigrams (min_docfreq = 5 count, max_docfreq = .06 prop) yields a better performance.

The best accuracy of 68.20% is obtained using the Superlearner ensemble of classifiers (although this is not cross-validated). Ridge (L2) is the best single classifier with 67.69%.

The fastest classifier is Naive bayes, estimated locally in 0.34 minutes (accuracy 65.98 %).

2 Setting up

This script requires the files which are not included on GitHub.

2.1 Loading packages

This script is based mainly on the functions of the quanteda package. For the cross-validation of the textmodels, quanteda.classifiers has to be loaded from GitHub.

```
start_time <- Sys.time()

packages <- c("quanteda", "quanteda.textmodels", "dplyr", "caret", "randomForest",
```

```
    "tm", "rmarkdown", "plyr", "readr", "ggplot2", "stringr", "formatR", "readstata13",
    "lubridate", "reticulate", "doMC", "glmnet", "kableExtra", "stargazer", "extrafont",
    "remotes", "devtools")

# lapply(packages[!(packages %in% rownames(installed.packages()))],
# install.packages)

if (!("quanteda.classifiers" %in% rownames(installed.packages()))) {
    remotes::install_github("quanteda/quanteda.classifiers")
    # devtools::install_github('quanteda/quanteda.classifiers')
}

invisible(lapply(c(packages, "quanteda.classifiers"), require, character.only = T))
loadfonts(device = "pdf")
theme_update(text = element_text(family = "LM Roman 10"))  # Set font family for ggplot

if (!dir.exists("supervised-files")) dir.create("supervised-files")

source("scripts/functions.R")

# Seed
seed <- 1621447882
set.seed(seed)

# Five-fold cross-validation
folds_cv <- 5
```

2.2 Loading document frequency matrix (dfm)

The sample data for Germany consists of 2,612 labeled press releases. The dataset is not uploaded on GitHub.

In order to improve the classification, similar topics were merged or subsumed under the "Other" category. In practice, press releases regarding, for instance, Environment and Energy are often not distinguishable. Furthermore, small categories with very few observations are not suitable for automated classification.

The dfm is generated in the script "Preparing the textual data".

We create a text corpus based on the header and text of each press release. We draw a random sample from the corpus to create a training and a test dataset. The test dataset consists of approx. one fifth of the documents.

Subsequently, we follow standard procedures for the preparation of the document frequency matrix. First, we remove stopwords and stem the words in order to better capture the similarities across documents. Second, we remove all punctuation, numbers, symbols and URLs. In a last step, we remove all words occurring in less than 0.5% or more than 90% of documents.

To test the robustness of the dfm representation, we run additional models with an alternative dfm. This dfm contains bigrams and is limited to features occuring in at least 5 (count) and at most 6% of the documents. The dfm is tfidf transformed.

```
load("supervised-files/data/dfmat.RData")
load("supervised-files/data/dfmat_alt.RData")
```

3 Textmodels

Following Barberá et al. (2021) we estimate the following models:

1. Naive Bayes (multinomial)
2. Ridge regression (L2)
3. Lasso regression (L1)
4. Elastic Net
5. SVM
6. Random Forest

(Barberá, P., Boydstun, A., Linn, S., McMahon, R., & Nagler, J. (2021). Automated Text Classification of News Articles: A Practical Guide. Political Analysis, 29(1), 19-42. doi:10.1017/pan.2020.8)

We are mainly using the R packages quanteda and LiblineaR for the textmodels. For the SuperLearner ensemble we are replicating the models in Python with the sklearn library and use the mlens library to build the ensemble.

For most classifications we use a five-fold cross-validation.

An overview of the results for each classifier can be found in the next section.

3.1 Naive Bayes (multinomial)

We calculate a Multinomial Naive Bayes (NB) text classification model. Multinomial NB models take into account the number of times a word occurs in a document, whereas Bernoulli NB models use the presence or absence of words only. We also test whether a Bernoulli NB model potentially outperforms the multinomial one.

```
# The textmodels are all saved in a directory. When the parameters are changed,
# the folder 'textmodels' should be deleted in order to re-run the models.

# Five-fold cross-validation for every possible parameter combination
if (!dir.exists("supervised-files/textmodels")) dir.create("supervised-files/textmodels")
filename <- "supervised-files/textmodels/naivebayes_eval.RData"
if (file.exists(filename)) load(filename) else {
    naivebayes_eval <- textmodel_evaluate(dfmat, dfmat$issue, k = folds_cv, seed = seed,
        model = "textmodel_nb", fun = c("accuracy", "precision", "recall", "f1_score"),
        parameters = list(prior = c("uniform", "docfreq", "termfreq"), distribution = c("multinomial",
            "Bernoulli"), smooth = c(1, 2, 3))) %>%
        dplyr::rename(weight = prior)

    # Add alternative dfm (with bigrams and max_docfreq 20%)
    naivebayes_eval <- textmodel_evaluate(dfmat_alt, dfmat_alt$issue, k = folds_cv,
        seed = seed, model = "textmodel_nb", fun = c("accuracy", "precision", "recall",
            "f1_score"), parameters = list(distribution = c("multinomial", "Bernoulli"),
            smooth = c(1, 2, 3))) %>%
        dplyr::mutate(weight = "tfidf") %>%
        rbind.fill(naivebayes_eval)

    naivebayes_eval$model <- "Naive bayes"
    save(naivebayes_eval, file = filename)
}
```

3.2 Ridge regression (L2)

```
# Five-fold cross-validation for every possible parameter combination
filename <- "supervised-files/textmodels/ridge_eval.RData"
if (file.exists(filename)) load(filename) else {
    ridge_eval <- textmodel_evaluate(dfmat, dfmat$issue, k = folds_cv, seed = seed,
```

```
        model = "textmodel_svm", fun = c("accuracy", "precision", "recall", "f1_score"),
        parameters = list(weight = c("uniform", "docfreq", "termfreq"), type = c(0,
            7)))

    # Add alternative dfm (with bigrams and max_docfreq 20%)
    ridge_eval <- textmodel_evaluate(dfmat_alt, dfmat_alt$issue, k = folds_cv, seed = seed,
        model = "textmodel_svm", fun = c("accuracy", "precision", "recall", "f1_score"),
        parameters = list(type = c(0, 7))) %>%
        dplyr::mutate(weight = "tfidf") %>%
        rbind.fill(ridge_eval)

    ridge_eval$model <- "Ridge (L2)"
    save(ridge_eval, file = filename)
}
```

3.3 Lasso regression (L1)

We estimate the L1-regularized logistic regression for our classification task.

```
# Five-fold cross-validation for every possible parameter combination
filename <- "supervised-files/textmodels/lasso_eval.RData"
if (file.exists(filename)) load(filename) else {
    lasso_eval <- textmodel_evaluate(dfmat, dfmat$issue, k = folds_cv, seed = seed,
        model = "textmodel_svm", fun = c("accuracy", "precision", "recall", "f1_score"),
        parameters = list(weight = c("uniform", "docfreq", "termfreq"), type = 6))

    # Add alternative dfm (with bigrams and max_docfreq 20%)
    lasso_eval <- textmodel_evaluate(dfmat_alt, dfmat_alt$issue, k = folds_cv, seed = seed,
        model = "textmodel_svm", fun = c("accuracy", "precision", "recall", "f1_score"),
        parameters = list(type = 6)) %>%
        dplyr::mutate(weight = "tfidf") %>%
        rbind.fill(lasso_eval)

    lasso_eval$model <- "Lasso (L1)"
    save(lasso_eval, file = filename)
}
```

3.4 Elastic Net

The elastic net method combines the L1 and L2 penalties of the lasso and ridge methods (above).

```
# Register multicore backend
registerDoMC(cores = 4)

# Estimate model
filename <- "supervised-files/textmodels/elasticnet_mod.RData"
if (file.exists(filename)) load(filename) else {
    eval_start <- Sys.time()
    elasticnet_mod <- glmnet(x = dfm_subset(dfmat, dfmat$cv_sample != 1), y = dfm_subset(dfmat,
        dfmat$cv_sample != 1)$issue, family = "multinomial", alpha = 0.5, type.measure = "class",
        standardize = T)
    elasticnet_mod$time <- as.numeric(Sys.time() - eval_start)
    elasticnet_mod$seed <- seed
    elasticnet_mod$weight <- "uniform"
    elasticnet_mod$model <- "Elastic net"
```

```
    save(elasticnet_mod, file = filename)
}

filename <- "supervised-files/textmodels/elasticnet_pred.RData"
if (file.exists(filename)) load(filename) else {
    # Get lambda with best accuracy
    elasticnet_pred <- predict(elasticnet_mod, newx = as.matrix(dfm_subset(dfmat,
        dfmat$cv_sample == 1)), type = "class")
    acc_list <- apply(elasticnet_pred, MARGIN = 2, FUN = function(x) accuracy(x,
        dfm_subset(dfmat, dfmat$cv_sample == 1)$issue))
    elasticnet_pred <- predict(elasticnet_mod, newx = as.matrix(dfm_subset(dfmat,
        dfmat$cv_sample == 1)), type = "class", s = elasticnet_mod$lambda[which(unlist(acc_list) ==
        max(acc_list %>%
            unlist))][1])

    save(elasticnet_pred, file = filename)
}

# Save accuracy to compare to alternative dfm
(benchmark_acc <- accuracy(elasticnet_pred, dfm_subset(dfmat, dfmat$cv_sample ==
    1)$issue))
```

```
## $accuracy
## [1] 0.6089494
```

```
# Estimate alternative model (bigrams and tfidf)
filename <- "supervised-files/textmodels/elasticnet_mod_alt.RData"
if (file.exists(filename)) load(filename) else {
    eval_start <- Sys.time()
    elasticnet_mod_alt <- glmnet(x = dfm_subset(dfmat_alt, dfmat_alt$cv_sample !=
        1), y = dfm_subset(dfmat_alt, dfmat_alt$cv_sample != 1)$issue, family = "multinomial",
        alpha = 0.5, type.measure = "class", standardize = T)
    elasticnet_mod_alt$time <- as.numeric(Sys.time() - eval_start)
    elasticnet_mod_alt$seed <- seed
    elasticnet_mod_alt$weight <- "tfidf"
    elasticnet_mod_alt$model <- "Elastic net"
    save(elasticnet_mod_alt, file = filename)
}

# Get lambda with best accuracy
elasticnet_pred_alt <- predict(elasticnet_mod_alt, newx = as.matrix(dfm_subset(dfmat_alt,
    dfmat_alt$cv_sample == 1)), type = "class")
acc_list <- apply(elasticnet_pred_alt, MARGIN = 2, FUN = function(x) accuracy(x,
    dfm_subset(dfmat_alt, dfmat_alt$cv_sample == 1)$issue))
elasticnet_pred_alt <- predict(elasticnet_mod_alt, newx = as.matrix(dfm_subset(dfmat_alt,
    dfmat_alt$cv_sample == 1)), type = "class", s = elasticnet_mod_alt$lambda[which(unlist(acc_list) ==
    max(acc_list %>%
        unlist))][1])

filename <- "supervised-files/textmodels/elasticnet_pred_alt.RData"
save(elasticnet_pred_alt, file = filename)

# Which model is better?
print("Optimal weight:")
```

```
## [1] "Optimal weight:"
```

```
if (accuracy(elasticnet_pred_alt, dfm_subset(dfmat_alt, dfmat_alt$cv_sample == 1)$issue) %>%
    as.numeric() > benchmark_acc %>%
    as.numeric()) {
    print(elasticnet_mod_alt$weight)
    elasticnet_opt_alt <- T
} else {
    print(elasticnet_mod$weight)
    elasticnet_opt_alt <- F
}
```

```
## [1] "uniform"
```

```
filename <- "supervised-files/textmodels/elasticnet_opt_alt.RData"
save(elasticnet_opt_alt, file = filename)
```

3.5 SVM

We estimate support vector classification (Crammer and Singer 2001) for our classification task.

(Crammer, K. & Singer, Y. (2001). On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. Journal of Machine Learning Research, 2. 265-292.)

```
# Five-fold cross-validation for every possible parameter combination
filename <- "supervised-files/textmodels/svm_eval.RData"
if (file.exists(filename)) load(filename) else {
    svm_eval <- textmodel_evaluate(dfmat, dfmat$issue, k = folds_cv, seed = seed,
        model = "textmodel_svm", fun = c("accuracy", "precision", "recall", "f1_score"),
        parameters = list(weight = c("uniform", "docfreq", "termfreq"), type = 4))

    # Add alternative dfm (with bigrams and max_docfreq 20%)
    svm_eval <- textmodel_evaluate(dfmat_alt, dfmat_alt$issue, k = folds_cv, seed = seed,
        model = "textmodel_svm", fun = c("accuracy", "precision", "recall", "f1_score"),
        parameters = list(type = 4)) %>%
        dplyr::mutate(weight = "tfidf") %>%
        rbind.fill(svm_eval)

    svm_eval$model <- "SVM"
    save(svm_eval, file = filename)
}
```

```
## Warning: 2 features in newdata not used in prediction.
```

```
## Warning: 1 feature in newdata not used in prediction.
```

```
## Warning: 2 features in newdata not used in prediction.
```

```
## Warning: 2 features in newdata not used in prediction.
```

3.6 Random Forest

We estimate a model using the random forest algorithm for classification. We forgo cross-validation because OOB avoids over-classification.

(Breiman, L. (2001), Random Forests, Machine Learning 45(1), 5-32.)

```
# Estimate model (100 trees)
filename <- "supervised-files/textmodels/randomforest_eval.RData"
```

```r
if (file.exists(filename)) load(filename) else {
    eval_start <- Sys.time()
    randomforest_eval <- randomForest(x = as.matrix(dfm_subset(dfmat, dfmat$cv_sample !=
        1)), y = as.factor(dfm_subset(dfmat, dfmat$cv_sample != 1)$issue), xtest = as.matrix(dfm_subset
        dfmat$cv_sample == 1)), ytest = as.factor(dfm_subset(dfmat, dfmat$cv_sample ==
        1)$issue), importance = T, mtry = 20, ntree = 100, keep.forest = T, type = "class")
    randomforest_eval$time <- as.numeric(Sys.time() - eval_start)
    randomforest_eval$seed <- seed
    randomforest_eval$model <- "Random forest"
    randomforest_eval$weight <- "uniform"
    save(randomforest_eval, file = filename)
}

randomforest_pred <- predict(randomforest_eval, newdata = as.matrix(dfm_subset(dfmat,
    dfmat$cv_sample == 1)), type = "class")

filename <- "supervised-files/textmodels/randomforest_pred.RData"
save(randomforest_pred, file = filename)

# Save accuracy to compare to alternative dfm
(benchmark_acc <- accuracy(randomforest_pred, dfm_subset(dfmat, dfmat$cv_sample ==
    1)$issue))
```

```
## $accuracy
## [1] 0.5856031
```

```r
# Estimate alternative model (bigrams and tfidf)
filename <- "supervised-files/textmodels/randomforest_eval_alt.RData"
if (file.exists(filename)) load(filename) else {
    eval_start <- Sys.time()
    randomforest_eval_alt <- randomForest(x = as.matrix(dfm_subset(dfmat_alt, dfmat_alt$cv_sample !=
        1)), y = as.factor(dfm_subset(dfmat_alt, dfmat_alt$cv_sample != 1)$issue),
        xtest = as.matrix(dfm_subset(dfmat_alt, dfmat_alt$cv_sample == 1)), ytest = as.factor(dfm_subse
            dfmat_alt$cv_sample == 1)$issue), importance = T, mtry = 20, ntree = 100,
        keep.forest = T, type = "class")
    randomforest_eval_alt$time <- as.numeric(Sys.time() - eval_start)
    randomforest_eval_alt$seed <- seed
    randomforest_eval_alt$model <- "Random forest"
    randomforest_eval_alt$weight <- "tfidf"
    save(randomforest_eval_alt, file = filename)
}

randomforest_pred_alt <- predict(randomforest_eval_alt, newdata = as.matrix(dfm_subset(dfmat_alt,
    dfmat_alt$cv_sample == 1)), type = "class")

filename <- "supervised-files/textmodels/randomforest_pred_alt.RData"
save(randomforest_pred_alt, file = filename)


# Which model is better?
print("Optimal weight:")
```

```
## [1] "Optimal weight:"
```

```
if (accuracy(randomforest_pred_alt, dfm_subset(dfmat_alt, dfmat_alt$cv_sample ==
    1)$issue) %>%
    as.numeric() > benchmark_acc %>%
    as.numeric()) {
    print(randomforest_eval_alt$weight)
    randomforest_opt_alt <- T
} else {
    print(randomforest_eval$weight)
    randomforest_opt_alt <- F
}
```

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

```
filename <- "supervised-files/textmodels/randomforest_opt_alt.RData"
save(randomforest_opt_alt, file = filename)
```

4 Ensemble methods: SuperLearner

The SuperLearner is run in Google Colab (Python).

See: https://colab.research.google.com/drive/1vUF7fGXenLG1kXqWsWnSTrwAHSmnaGUG?usp=sharing

Maximum accuracy:

5 Semi-supervised models

The semi-supervised models are run in Google Colab (Python).

See: https://colab.research.google.com/drive/1JR6pVtnIwVK4AMu-IN4UN43ABWflCUUO?usp=sharing

Maximum accuracy:

```
# Time needed to run script (much shorter when textmodels are just loaded from
# file) The estimation time for the single textmodels can found in the table
# above.

print(Sys.time() - start_time)
```

```
## Time difference of 39.37045 mins
```

```
# In total, the script needs about 2-3h to run.
```