Supervised learning aggregated

Cornelius Erfort

5/26/2021

Contents

1 Setting up **1**
- 1.1 Loading packages . . . 1
- 1.2 Loading document frequency matrix (dfm) . . . 2

2 Textmodels **2**
- 2.1 Naive Bayes (multinomial) . . . 3
- 2.2 Ridge regression (L2) . . . 4
- 2.3 Lasso regression (L1) . . . 4
- 2.4 Elastic Net . . . 5
- 2.5 SVM . . . 6
- 2.6 Random Forest . . . 7

3 Ensemble methods: SuperLearner **8**

4 Evaluation of textmodels **10**

5 Classification of unlabeled data **15**
- 5.1 Using the NB textmodel . . . 15
- 5.2 Aggregation of the issues categories over time and party . . . 16

1 Setting up

This script requires the files "germany_textpress.RData" and "data_joint.RDS" which are not included on GitHub. The superlearner model and the training/test csv-files are also ignored for GitHub.

At the end of this script, the file "issue_agendas_supervised.RData" is saved. It contains quarterly estimates for the share of press releases for each issue and party.

1.1 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")

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

issue	1	2	3	4	5	6	7	9	10	12	15	16	17	20	99	191	192
n	175	181	119	99	167	137	189	131	210	195	195	121	68	97	63	350	152

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

invisible(lapply(c(packages, "quanteda.classifiers"), require, character.only = T))

loadfonts()
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")
```

1.2 Loading document frequency matrix (dfm)

The sample data for Germany consists of 2,740 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.

```
load("supervised-files/train-test/dfmat.RData")
load("supervised-files/train-test/dfmat_training.RData")
load("supervised-files/train-test/dfmat_test.RData")

load("supervised-files/issue_categories.RData")


# Distribution with merged categories
table(dfmat$issue_r1) %>%
    as.data.frame() %>%
    dplyr::rename(issue = Var1, n = Freq) %>%
    t() %>%
    kbl(booktabs = T) %>%
    kable_styling(latex_options = "scale_down")
```

2 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 ten-fold cross-validation.

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

2.1 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.

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

# 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_r1, k = folds_cv, 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)))
    save(naivebayes_eval, file = filename)
}

(naivebayes_eval_aggr <- aggregate(cbind(accuracy, precision, recall, f1_score, time,
    seed) ~ prior + distribution + smooth, naivebayes_eval[, -c(1)], mean) %>%
    arrange(desc(accuracy)))[1:5, ] %>%
    mutate_at(vars(accuracy, precision, recall, f1_score), function(x) round(x, 3)) %>%
    kbl(booktabs = T)
```

prior	distribution	smooth	accuracy	precision	recall	f1_score	time	seed
uniform	multinomial	1	0.650	0.634	0.640	0.630	0.778	1622717972
termfreq	multinomial	1	0.647	0.631	0.633	0.624	0.534	1622717972
docfreq	multinomial	1	0.645	0.628	0.628	0.621	0.598	1622717972
docfreq	multinomial	2	0.643	0.647	0.613	0.612	0.596	1622717972
uniform	multinomial	2	0.640	0.635	0.614	0.610	0.514	1622717972

```r
naivebayes_eval_aggr$model <- "Naive bayes"

# Create a dataframe for all textmodels and save first row
tm_eval <- data.frame()
tm_eval <- naivebayes_eval_aggr[1, ] %>%
    rbind.fill(tm_eval)
```

Assuming a multinomial distribution of text features leads to a higher accuracy of the models compared to a Bernoulli distribution. The other benchmark parameters confirm this finding.

There is no clear pattern in regard to the effect of the priors on the quality of the models. A smoothing parameter of 1 for the feature counts seems optimal.

2.2 Ridge regression (L2)

Lasso and Ridge regression mainly differ in the penalty term. While Lasso uses an absolute value as a penalty term, Ridge uses the squared value.

We estimate the L2-regularized logistic regression for our classification task (primal and dual).

```r
# 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_r1, k = folds_cv, model = "textmodel_svm",
        fun = c("accuracy", "precision", "recall", "f1_score"), parameters = list(weight = c("uniform",
            "docfreq", "termfreq"), type = c(0, 7)))
    save(ridge_eval, file = filename)
}

(ridge_eval_aggr <- aggregate(cbind(accuracy, precision, recall, f1_score, time,
    seed) ~ weight + type, ridge_eval[, -c(1)], mean) %>%
    arrange(desc(accuracy)))[1:5, ] %>%
    mutate_at(vars(accuracy, precision, recall, f1_score), function(x) round(x, 3)) %>%
    kbl(booktabs = T)
```

weight	type	accuracy	precision	recall	f1_score	time	seed
uniform	7	0.606	0.606	0.587	0.590	4.916	1622718032
uniform	0	0.605	0.608	0.584	0.588	4.200	1622718032
docfreq	0	0.603	0.644	0.560	0.570	2.720	1622718032
termfreq	0	0.602	0.648	0.560	0.568	2.924	1622718032
termfreq	7	0.602	0.646	0.561	0.569	4.324	1622718032

```r
ridge_eval_aggr$model <- "Ridge (L2)"

# Add first row to overall dataframe
tm_eval <- ridge_eval_aggr[1, ] %>%
    rbind.fill(tm_eval)
```

The classifier does not provide a higher accuracy compared to the baseline NB model.

The calculation of the models requires much more time/computing compared to the NB models.

2.3 Lasso regression (L1)

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

```r
# 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_r1, k = folds_cv, model = "textmodel_svm",
        fun = c("accuracy", "precision", "recall", "f1_score"), parameters = list(weight = c("uniform",
            "docfreq", "termfreq"), type = 6))
    save(lasso_eval, file = filename)
}

(lasso_eval_aggr <- aggregate(cbind(accuracy, precision, recall, f1_score, time,
    seed) ~ weight, lasso_eval[, -c(1)], mean) %>%
    arrange(desc(accuracy)))[1:2, ] %>%
    mutate_at(vars(accuracy, precision, recall, f1_score), function(x) round(x, 3)) %>%
    kbl(booktabs = T)
```

weight	accuracy	precision	recall	f1_score	time	seed
uniform	0.585	0.588	0.566	0.568	3.554	1622718151
docfreq	0.428	0.456	0.403	0.400	3.070	1622718151

```r
lasso_eval_aggr$model <- "Lasso (L1)"

# Add first row to overall dataframe
tm_eval <- lasso_eval_aggr[1, ] %>%
    rbind.fill(tm_eval)
```

The classifier does not lead to a higher accuracy compared to the baseline NB model and to a slightly lower accuracy than the ridge classifier.

The calculation of the models requires much more time/computing compared to the NB models.

2.4 Elastic Net

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

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

filename <- "supervised-files/textmodels/elasticnet_eval.RData"
if (file.exists(filename)) load(filename) else {
    elasticnet_eval <- cv.glmnet(x = dfmat, y = dfmat$issue_r1, family = "multinomial",
        alpha = 0.5, nfolds = folds_cv, type.measure = "class", parallel = T, standardize = T)
    save(elasticnet_eval, file = filename)
}

elasticnet_eval$lambda.min
```

```
## [1] 0.02326881
```

```r
# Misclassification error
print(elasticnet_eval$cvm %>%
    min)
```

```
## [1] 0.3839185
```

```r
# Accuracy
print(1 - elasticnet_eval$cvm %>%
```

```
    min)
```

```
## [1] 0.6160815
```

```
seed <- 1621447882
set.seed(seed)

# Estimate model
filename <- "supervised-files/textmodels/elasticnet_mod.RData"
if (file.exists(filename)) load(filename) else {
    eval_start <- Sys.time()
    elasticnet_mod <- glmnet(x = dfmat_training, y = dfmat_training$issue_r1, family = "multinomial",
        alpha = 0.5, type.measure = "class", standardize = T)
    elasticnet_mod$time <- as.numeric(Sys.time() - eval_start)
    save(elasticnet_mod, file = filename)
}


# Get lambda with best accuracy
elasticnet_pred <- predict(elasticnet_mod, newx = as.matrix(dfmat_test), type = "class")

acc_list <- apply(elasticnet_pred, MARGIN = 2, FUN = function(x) accuracy(x, dfmat_test$issue_r1))

elasticnet_pred <- predict(elasticnet_mod, newx = as.matrix(dfmat_test), type = "class",
    s = elasticnet_mod$lambda[which(unlist(acc_list) == max(acc_list %>%
        unlist))][1])

tm_eval <- data.frame(accuracy = accuracy(elasticnet_pred, dfmat_test$issue_r1),
    precision = precision(elasticnet_pred, dfmat_test$issue_r1) %>%
        unlist() %>%
        mean(), recall = recall(elasticnet_pred, dfmat_test$issue_r1) %>%
        unlist() %>%
        mean(), f1_score = f1_score(elasticnet_pred, dfmat_test$issue_r1) %>%
        unlist() %>%
        mean(), time = elasticnet_mod$time, seed = seed, model = "Elastic net", alpha = 0.5,
    distribution = "multinomial") %>%
    rbind.fill(tm_eval)
```

The classifier does not provide a higher accuracy compared to the baseline NB model.

The calculation of the models requires much more time/computing compared to the NB models.

2.5 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_r1, k = folds_cv, model = "textmodel_svm",
        fun = c("accuracy", "precision", "recall", "f1_score"), parameters = list(weight = c("uniform",
            "docfreq", "termfreq"), type = 4))
    save(svm_eval, file = filename)
```

```
}

(svm_eval_aggr <- aggregate(cbind(accuracy, precision, recall, f1_score, time, seed) ~
    weight, svm_eval[, -c(1)], mean) %>%
    arrange(desc(accuracy)))[1:3, ] %>%
    mutate_at(vars(accuracy, precision, recall, f1_score), function(x) round(x, 3)) %>%
    kbl(booktabs = T)
```

weight	accuracy	precision	recall	f1_score	time	seed
docfreq	0.578	0.585	0.566	0.567	2.110	1622718322
termfreq	0.576	0.583	0.564	0.565	2.216	1622718322
uniform	0.574	0.575	0.562	0.562	2.218	1622718322

```
svm_eval_aggr$model <- "SVM"

# Add first row to overall dataframe
tm_eval <- svm_eval_aggr[1, ] %>%
    rbind.fill(tm_eval)
```

None of the configurations lead to a higher accuracy compared to the baseline NB model.

There is no clear pattern regarding the choice of weights.

The calculation of the models requires more time/computing compared to the NB models.

2.6 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"
if (file.exists(filename)) load(filename) else {
    eval_start <- Sys.time()
    randomforest_eval <- randomForest(x = as.matrix(dfmat_training), y = as.factor(dfmat_training$issue_
        xtest = as.matrix(dfmat_test), ytest = as.factor(dfmat_test$issue_r1), importance = T,
        mtry = 20, ntree = 100, keep.forest = T, type = "class")
    randomforest_eval$time <- as.numeric(Sys.time() - eval_start)
    save(randomforest_eval, file = filename)
}

randomforest_pred <- predict(randomforest_eval, newdata = as.matrix(dfmat_test),
    type = "class")

# Accuracy
print(accuracy(randomforest_pred, dfmat_test$issue_r1))
```

```
## $accuracy
## [1] 0.5996169
```

```
tm_eval <- data.frame(accuracy = accuracy(randomforest_pred, dfmat_test$issue_r1),
    precision = precision(randomforest_pred, dfmat_test$issue_r1) %>%
        unlist() %>%
        mean(), recall = recall(randomforest_pred, dfmat_test$issue_r1) %>%
```

```
        unlist() %>%
        mean(), f1_score = f1_score(randomforest_pred, dfmat_test$issue_r1) %>%
        unlist() %>%
        mean(), time = randomforest_eval$time, seed = seed, model = "Random forest",
    alpha = 0.5, distribution = "multinomial") %>%
    rbind.fill(tm_eval)

eval_start <- Sys.time()
```

Random forest yields a lower accuracy than the baseline model and the computing time is much longer.

3 Ensemble methods: SuperLearner

Run SuperLearner in Python. (Additionally running AdaBoost.)

```
# Load libraries
import os
import pandas as pd
import numpy as np
import pickle as pk
from sklearn.metrics import accuracy_score # Load sklearn tools
from mlens.ensemble import SuperLearner # Load SuperLearner

# Load classifiers
from sklearn.naive_bayes import MultinomialNB # 1
from sklearn.naive_bayes import BernoulliNB
# 2, 3 and 4 (logistic regression L2/L1 penalty and elastic net)
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC # 5
from sklearn.ensemble import RandomForestClassifier # 6
from sklearn.ensemble import AdaBoostClassifier # 7

# Set params
super_folds = 5
super_tol = .005

# Create a list of base-models
def get_models():
    models = list()
    models.append(MultinomialNB())
    models.append(BernoulliNB())
    models.append(LogisticRegression(solver = 'liblinear',
    max_iter = 1000, tol = super_tol, penalty = "l2"))
    models.append(LogisticRegression(solver = 'liblinear',
    max_iter = 1000, tol = super_tol, penalty = "l1"))
    models.append(LogisticRegression(solver = 'saga', max_iter = 1000,
    penalty = 'elasticnet', l1_ratio = .5, multi_class = 'multinomial',
    random_state = np.random.seed(3027), tol = super_tol))
    models.append(SVC(probability = True, tol = super_tol))
    models.append(RandomForestClassifier())
    models.append(AdaBoostClassifier())
    return models

# Create the superlearner
```

```python
def get_super_learner(X):
    ensemble = SuperLearner(scorer = None, folds = super_folds, shuffle = True,
    random_state = np.random.seed(3027), sample_size = len(train_val),
    n_jobs = 1, verbose = True)
    models = get_models() # Add base models
    ensemble.add(models, proba = True)
    ensemble.add_meta(LogisticRegression(solver = 'lbfgs',
    max_iter = 1000, tol = super_tol), proba = False) # Add the meta model
    return ensemble

# Load press release data (training and test)
train = pd.read_csv("supervised-files/train-test/train.csv", index_col = 0).values
train_val = np.asarray([int(i) for i in pd.read_csv("supervised-files/train-test/train_val.csv",
index_col = 0).values])

test = pd.read_csv("supervised-files/train-test/test.csv", index_col = 0).values
test_val = np.asarray([int(i) for i in pd.read_csv("supervised-files/train-test/test_val.csv",
index_col = 0).values])

# Create the super learner
ensemble = get_super_learner(train)

# Fit the super learner
filename = "supervised-files/textmodels/superlearner.PyData"
if os.path.exists(filename) != True:
  ensemble.fit(train, train_val)
  pk.dump(ensemble, open(filename, 'wb'))
else:
  ensemble = pk.load(open(filename, 'rb'))

# Summarize base learners
print(ensemble.data)

# Make predictions on test data
filename = "supervised-files/textmodels/super_pred.PyData"
if os.path.exists(filename) != True:
  super_pred = ensemble.predict(test)
  pk.dump(super_pred, open(filename, 'wb'))
  np.savetxt("supervised-files/textmodels/super_pred.csv", super_pred, delimiter = ",")
else:
  super_pred = pk.load(open(filename, 'rb'))
```

```
## [MLENS] backend: threading
##                                    ft-m    ft-s  pt-m  pt-s
## layer-1  adaboostclassifier        16.86    5.31  0.44  0.11
## layer-1  bernoullinb                3.38    0.21  0.05  0.02
## layer-1  logisticregression-1       1.86    0.55  0.02  0.01
## layer-1  logisticregression-2       0.62    0.02  0.01  0.00
## layer-1  logisticregression-3     693.71  311.28  0.10  0.06
## layer-1  multinomialnb              3.41    0.03  0.01  0.00
## layer-1  randomforestclassifier    14.90    0.34  0.18  0.04
## layer-1  svc                      159.87   67.02  5.32  2.05
```

```r
# Load superlearner prediction and add row to evaluation table
super_pred <- read_csv("supervised-files/textmodels/super_pred.csv", col_names = F)[[1]] %>% as.numeric
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double()
## )
```

```r
tm_eval <- data.frame(accuracy = accuracy(super_pred, dfmat_test$issue_r1),
           precision = precision(super_pred, dfmat_test$issue_r1) %>% unlist() %>% mean(),
           recall = recall(super_pred, dfmat_test$issue_r1) %>% unlist() %>% mean(),
           f1_score = f1_score(super_pred, dfmat_test$issue_r1) %>% unlist() %>% mean(),
           time = 67.78454, # as.numeric(Sys.time() - eval_start), # Calculated manually
           model = "SuperLearner ensemble") %>%
  rbind.fill(tm_eval)
```

The SuperLearner does not significantly increase the accuracy, but the necessary computing time is much greater.

4 Evaluation of textmodels

In this section, we present a table for comparison of our textmodels.

```r
# Comparison of textmodels
tm_eval[, c("accuracy", "precision", "recall", "f1_score", "time")] <- apply(tm_eval[,
    c("accuracy", "precision", "recall", "f1_score", "time")], MARGIN = 2, function(x) round(x,
    3))

tm_eval[order(tm_eval$accuracy, decreasing = T), c("model", "accuracy", "precision",
    "recall", "f1_score", "distribution", "time")] %>%
    kbl(booktabs = T, row.names = F)
```

model	accuracy	precision	recall	f1_score	distribution	time
SuperLearner ensemble	0.676	0.687	0.656	0.661	NA	67.785
Naive bayes	0.650	0.634	0.640	0.630	multinomial	0.778
Elastic net	0.623	0.655	0.591	0.609	multinomial	19.158
Ridge (L2)	0.606	0.606	0.587	0.590	NA	4.916
Random forest	0.600	0.667	0.562	0.565	multinomial	31.171
Lasso (L1)	0.585	0.588	0.566	0.568	NA	3.554
SVM	0.578	0.585	0.566	0.567	NA	2.110

```r
latex_out <- capture.output(tm_eval[order(tm_eval$accuracy, decreasing = T), c("model",
    "accuracy", "precision", "recall", "f1_score", "time")] %>%
    dplyr::rename(Model = model, Accuracy = accuracy, Precision = precision, Recall = recall,
        `F1 score` = f1_score) %>%
    stargazer(out = "tables/tm_eval.tex", summary = F, rownames = F, title = "Summary of classifier per
        label = "tab:tm_eval"))

# Naive Bayes baseline model
baseline_nb <- textmodel_nb(dfmat_training, dfmat_training$issue_r1, distribution = "multinomial")

# Confusion matrix and overall statistics
```

```r
table(dfmat_test$issue_r1, predict(baseline_nb, newdata = dfmat_test)) %>%
    confusionMatrix(mode = "everything")
```

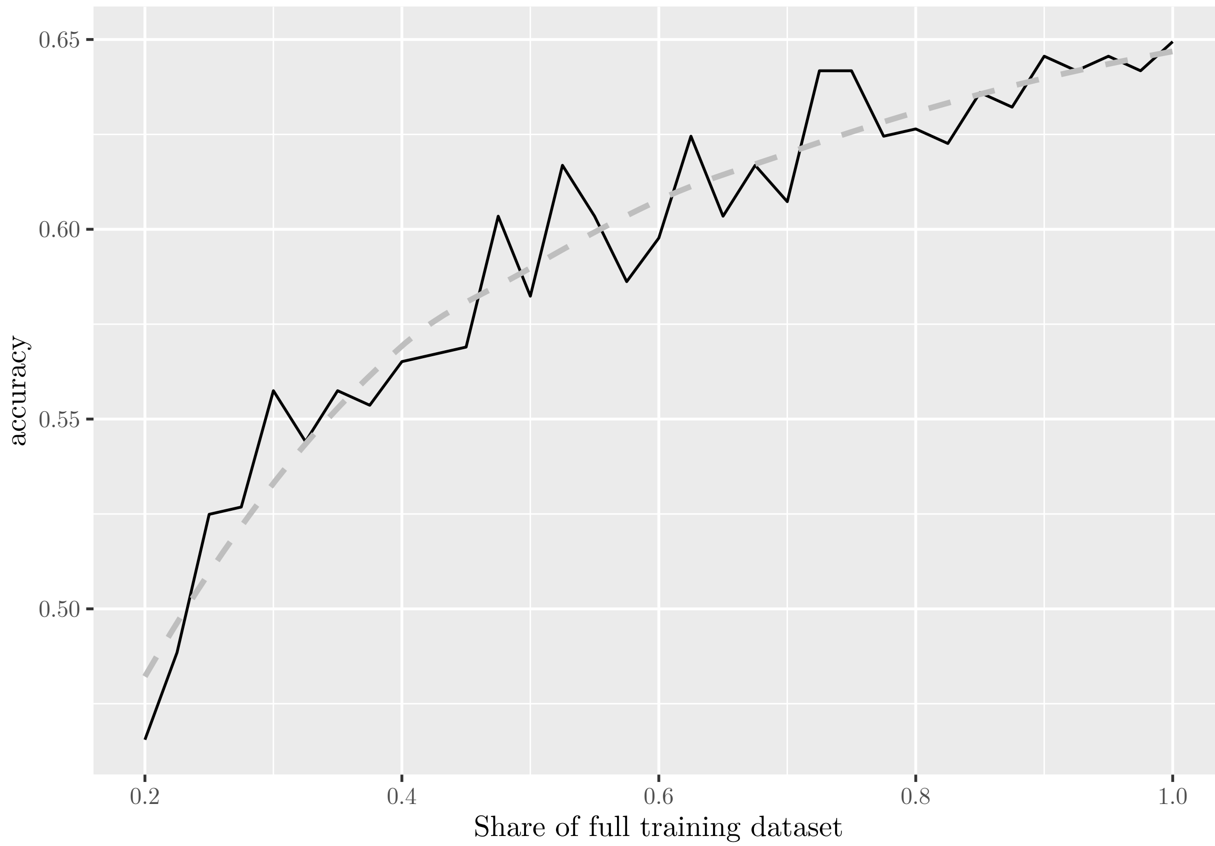
```
## Confusion Matrix and Statistics
##
##
##         1  2  3  4  5  6  7  9 10 12 15 16 17 20 99 191 192
##   1    26  0  0  1  2  1  2  0  0  0  1  0  1  0  2   0   0
##   2     0 14  1  0  2  1  0  0  0  7  2  1  0  0  1   1   0
##   3     1  3 20  3  2  1  0  0  0  0  0  0  1  0  0   0   0
##   4     0  0  0 24  0  0  1  0  0  0  2  0  0  0  0   0   0
##   5     2  0  0  0 20  0  1  1  1  1  0  0  0  0  0   0   1
##   6     0  1  0  0  0 15  0  0  2  0  1  0  0  0  0   0   0
##   7     1  0  1  4  0  0 27  0  1  0  0  1  1  0  0   1   2
##   9     0  1  0  0  0  1  0 22  0  4  1  0  0  0  0   0   1
##   10    2  1  0  0  3  3  4  0 27  2  2  0  1  1  0   0   0
##   12    1  4  0  0  0  0  1  0  0 24  3  1  0  0  3   1   0
##   15    1  1  0  4  0  0  2  0  2  1 23  0  3  0  1   2   5
##   16    0  1  0  0  0  0  0  0  0  1  0 16  0  1  0   6   1
##   17    2  0  0  0  0  1  0  0  1  1  0  0  9  0  0   0   0
##   20    0  0  0  0  0  1  0  1  5  3  3  1  0  6  1   1   1
##   99    0  1  0  0  1  2  0  0  0  0  1  0  0  0  5   0   0
##   191   1  1  0  0  0  0  3  1  0  0  0  4  0  1  0  43   1
##   192   0  0  0  0  0  0  0  0  0  1  1  0  0  1  1   4  18
##
## Overall Statistics
##
##                Accuracy : 0.6494
##                  95% CI : (0.6068, 0.6904)
##     No Information Rate : 0.113
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6246
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.70270  0.50000  0.90909  0.66667  0.66667  0.57692
## Specificity           0.97938  0.96761  0.97800  0.99383  0.98577  0.99194
## Pos Pred Value        0.72222  0.46667  0.64516  0.88889  0.74074  0.78947
## Neg Pred Value        0.97737  0.97154  0.99593  0.97576  0.97980  0.97813
## Precision             0.72222  0.46667  0.64516  0.88889  0.74074  0.78947
## Recall                0.70270  0.50000  0.90909  0.66667  0.66667  0.57692
## F1                    0.71233  0.48276  0.75472  0.76190  0.70175  0.66667
## Prevalence            0.07088  0.05364  0.04215  0.06897  0.05747  0.04981
## Detection Rate        0.04981  0.02682  0.03831  0.04598  0.03831  0.02874
## Detection Prevalence  0.06897  0.05747  0.05939  0.05172  0.05172  0.03640
## Balanced Accuracy     0.84104  0.73381  0.94355  0.83025  0.82622  0.78443
##                      Class: 7 Class: 9 Class: 10 Class: 12 Class: 15 Class: 16
## Sensitivity           0.65854  0.88000   0.69231   0.53333   0.57500   0.66667
## Specificity           0.97505  0.98390   0.96066   0.97065   0.95436   0.97992
## Pos Pred Value        0.69231  0.73333   0.58696   0.63158   0.51111   0.61538
```

```
## Neg Pred Value        0.97101  0.99390  0.97479  0.95661  0.96436  0.98387
## Precision             0.69231  0.73333  0.58696  0.63158  0.51111  0.61538
## Recall                0.65854  0.88000  0.69231  0.53333  0.57500  0.66667
## F1                    0.67500  0.80000  0.63529  0.57831  0.54118  0.64000
## Prevalence            0.07854  0.04789  0.07471  0.08621  0.07663  0.04598
## Detection Rate        0.05172  0.04215  0.05172  0.04598  0.04406  0.03065
## Detection Prevalence  0.07471  0.05747  0.08812  0.07280  0.08621  0.04981
## Balanced Accuracy     0.81679  0.93195  0.82649  0.75199  0.76468  0.82329
##                      Class: 17 Class: 20 Class: 99 Class: 191 Class: 192
## Sensitivity            0.56250   0.60000  0.357143    0.72881    0.60000
## Specificity            0.99012   0.96680  0.990157    0.97408    0.98374
## Pos Pred Value         0.64286   0.26087  0.500000    0.78182    0.69231
## Neg Pred Value         0.98622   0.99198  0.982422    0.96574    0.97581
## Precision              0.64286   0.26087  0.500000    0.78182    0.69231
## Recall                 0.56250   0.60000  0.357143    0.72881    0.60000
## F1                     0.60000   0.36364  0.416667    0.75439    0.64286
## Prevalence             0.03065   0.01916  0.026820    0.11303    0.05747
## Detection Rate         0.01724   0.01149  0.009579    0.08238    0.03448
## Detection Prevalence   0.02682   0.04406  0.019157    0.10536    0.04981
## Balanced Accuracy      0.77631   0.78340  0.673650    0.85145    0.79187
```

```r
# Accuracy
accuracy(dfmat_test$issue_r1, predict(baseline_nb, newdata = dfmat_test))
```

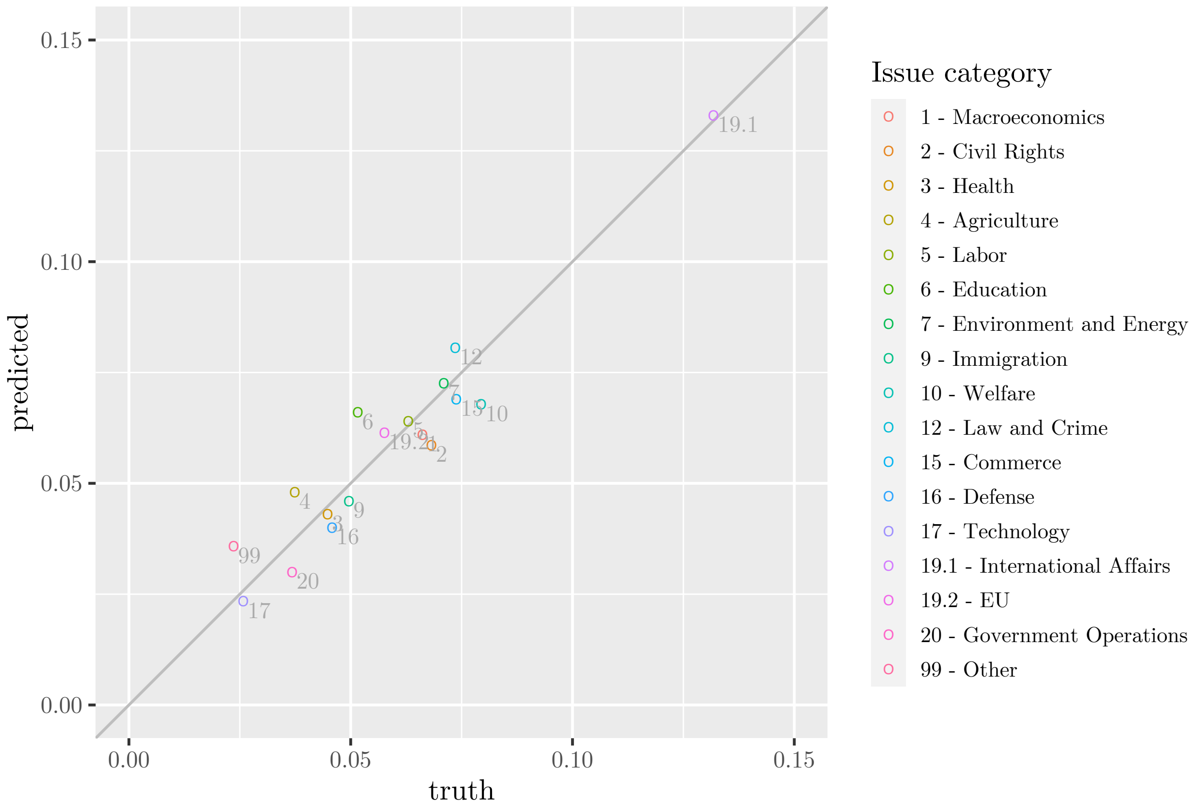
```
## $accuracy
## [1] 0.6494253
```

```r
# Simulating models with smaller training datasets (20% to 100% of our training
# set)
nb_opt <- data.frame()
for (train_size in seq(0.2, 1, 0.025)) {
    sub_train <- sample(1:ndoc(dfmat_training), train_size * ndoc(dfmat_training))
    nb_opt_mod <- textmodel_nb(dfmat_training[sub_train, ], dfmat_training$issue_r1[sub_train],
        distribution = "multinomial")
    nb_opt <- data.frame(train_size = train_size, train_size_abs = length(sub_train),
        accuracy = accuracy(dfmat_test$issue_r1, predict(nb_opt_mod, newdata = dfmat_test))) %>%
        rbind.fill(nb_opt)
}


ggplot(nb_opt, aes(x = train_size, y = accuracy)) + geom_line() + geom_smooth(method = "loess",
    formula = "y ~ x", se = F, lty = 2, color = "grey") + xlab("Share of full training dataset") +
    ggsave(str_c("plots/training-size-simulation.pdf"), device = cairo_pdf, width = 5 *
        2^0.5, height = 5) + ggsave(str_c("plots/training-size-simulation.png"),
    width = 5 * 2^0.5, height = 5)
```




```
# Aggregate Accuracy of predicted proportions (five-fold cross-validation)

nb_agg_cv <- list()
for (i in unique(dfmat$cv_sample)) {
    nb_agg_cv[[i]] <- textmodel_nb(dfmat[dfmat$cv_sample != i, ], dfmat$issue_r1[dfmat$cv_sample !=
        i], distribution = "multinomial")
}

# Prediction estimate and truth in %
agg_eval <- data.frame()
for (i in unique(dfmat$cv_sample)) {

    agg_eval <- data.frame(issue_r1 = table(dfmat$issue_r1) %>%
        rownames, predicted = (table(predict(nb_agg_cv[[i]], newdata = dfmat[dfmat$cv_sample ==
        i, ]))/sum(table(predict(nb_agg_cv[[i]], newdata = dfmat[dfmat$cv_sample ==
        i, ])))) %>%
        as.vector(), truth = (table(dfmat$issue_r1[dfmat$cv_sample == i])/sum(table(dfmat$issue_r1[dfma
        i]))) %>%
        as.vector(), cv_sample = i) %>%
        rbind.fill(agg_eval)

}

# Difference in percentage points (positive values indicate an inflated
# prediction, i.e. we estimate a higher share for the category compared to the
# truth)
agg_eval$difference <- agg_eval$predicted - agg_eval$truth
```

```r
agg_eval[, c(2:3, 5)] <- apply(agg_eval[, c(2:3, 5)], MARGIN = 2, function(x) round(x,
    3))

# Change and order labels
agg_eval$issue_r1[agg_eval$issue_r1 == 191] <- 19.1
agg_eval$issue_r1[agg_eval$issue_r1 == 192] <- 19.2
agg_eval$issue_r1 <- as.factor(as.numeric(agg_eval$issue_r1))
levels(agg_eval$issue_r1) <- str_c(levels(agg_eval$issue_r1), " - ", issue_categories[c(1:13,
    16:17, 14:15), 2])

save(agg_eval, file = "supervised-files/agg_eval.RData")

# Write latex table
if (!dir.exists("tables")) dir.create("tables")
latex_out <- capture.output(agg_eval %>%
    dplyr::group_by(issue_r1) %>%
    dplyr::summarise(predicted = mean(predicted), truth = mean(truth)) %>%
    dplyr::rename(issue = issue_r1) %>%
    as.data.frame() %>%
    stargazer(out = "tables/agg_eval_nb.tex", summary = F, rownames = F, title = "Evaluation of aggrega
        label = "tab:agg_eval_nb", notes = "Mean values from five-fold cross-validation."))


if (!dir.exists("plots")) dir.create("plots")

plot_agg_eval(agg_eval %>%
    dplyr::group_by(issue_r1) %>%
    dplyr::summarise(predicted = mean(predicted), truth = mean(truth)), "supervised")
```



The plot shows the mean values from a five-fold cross-validation.

The Naive Bayes model reaches a high accuracy which can only be slightly improved using an ensemble of classifiers. We thus rely on the NB classifier for the classification of press releases.

Regarding the issues, the classifier works better for specific issue categories. While some have a higher than average sensitivity (e.g. 3 - Health, 5 - Labor, 6 - Education, 7 - Env. & Energy, 9 - Immigration, 191 - Int. Affairs), others fare worse than average (e.g. 15 - Commerce, 17 - Technology, 20 - Gov. Ops., 99 - Other). Unsurprisingly, the rather unspecifict categories 20 - Gov. Ops. and 99 - Other are difficult to predict.

The better sensitivity for other categories is likely the result of a more specific use of words. Category 17 - Technology may feature a worse accuracy because it is underrepresented in the labeled data. Category 15 - Commerce is often misclassified as 1 - Macroeconomics, 4 - Agriculture, 17 - Technology and 191 - Int. Affairs: All categories where a press release may contain similar words.

2 - Civil Rights is often misclassified as 10 - Welfare or 12 - Defense. One might argue here, that press releases concerned with civil rights often also refer to military operations or the welfare of the persons involved.

10 - Welfare, on the other hand, is often categorized as 5 - Labor by the algorithm, a connection that is common in politics.

5 - Labor is often misclassified as 1 - Macroeconomics, a link that is hardly surprising.

16 - Defense is often misclassified as 191 - International Affairs.

5 Classification of unlabeled data

5.1 Using the NB textmodel

We trained the models using a set of 2,740 labeled documents. In order to obtain aggregated measures of issue attention, we predict the issue categories of all 47,111 labeled and unlabeled press releases in our sample.

issue	1	2	3	4	5	6	7	9	10	12	15	16	17	20	99	191	192
n	2775	2868	1703	1939	2979	3160	3348	2372	3038	3651	3030	2017	1290	1672	1604	6067	3042

We drop 556 manually identified non-thematic press releases. They can be easily identified because they feature recurrent themes such as obituaries.

```
# Naive Bayes with full labeled data
tm_naivebayes <- textmodel_nb(dfmat, dfmat$issue_r1, distribution = "multinomial")

# Loading full dataset (not on GitHub)
all_germany <- read_rds("data/data_joint.RDS") %>% select(c(header, text.x, date.x, issue, party.x, id))
nrow(all_germany)
```

```
## [1] 46555
```

```
# Constructing the document frequency matrix
dfmat_all <- corpus(str_c(all_germany$header, " ", all_germany$text.x),
                    docvars = select(all_germany, c(party, date, id))) %>%
  dfm(remove = stopwords("de"), # Stem and remove stopwords, punctuation etc.
      stem = T,
      remove_punct = T,
      remove_number = T,
      remove_symbols = T,
      remove_url = T) %>% suppressWarnings()

# Subsetting to features in the training data
dfmat_all <- dfm_match(dfmat_all, features = featnames(dfmat))

# Predicting the issue category for all documents
dfmat_all$issue_r1 <- predict(tm_naivebayes, newdata = dfmat_all)

table(dfmat_all$issue_r1) %>% as.data.frame() %>%
  dplyr::rename(issue = Var1, n = Freq) %>% t() %>% kbl(booktabs = T) %>%
  kable_styling(latex_options = "scale_down")
```

```
table(dfmat_all$issue_r1) / ndoc(dfmat_all)
```

```
##
##          1          2          3          4          5          6          7
## 0.05960692 0.06160455 0.03658039 0.04164966 0.06398883 0.06787670 0.07191494
##          9         10         12         15         16         17         20
## 0.05095049 0.06525615 0.07842337 0.06508431 0.04332510 0.02770916 0.03591451
##         99        191        192
## 0.03445387 0.13031898 0.06534207
```

5.2 Aggregation of the issues categories over time and party

To measure parties' evolving issue agendas, we aggregate the category counts over time.

```
# Create dataframe from dfm
issue_agendas_supervised <- data.frame(date = docvars(dfmat_all, "date"), party = docvars(dfmat_all,
    "party"), issue_r1 = docvars(dfmat_all, "issue_r1"))

# Make date quarterly
issue_agendas_supervised$date <- as.character(issue_agendas_supervised$date) %>%
    substr(1, 8) %>%
```

```r
    str_c("15") %>%
    str_replace_all(c(`-01-` = "-02-", `-03-` = "-02-", `-04-` = "-05-", `-06-` = "-05-",
        `-07-` = "-08-", `-09-` = "-08-", `-10-` = "-11-", `-12-` = "-11-")) %>%
    ymd()

# Add variable for counting
issue_agendas_supervised$freq <- 1

# Unite parties
issue_agendas_supervised$party <- issue_agendas_supervised$party %>%
    str_replace_all(c(union_fraktion = "CDU/CSU", spd_fraktion = "SPD", `90gruene_fraktion` = "B'90/Die
        fdp_bundesverband = "FDP", fdp_fraktion = "FDP", linke_fraktion = "DIE LINKE",
        afd_bundesverband = "AfD", afd_fraktion = "AfD"))

# Aggregate by party, date and issue
issue_agendas_supervised <- aggregate(freq ~ party + date + issue_r1, issue_agendas_supervised,
    sum)

# Add observations with zero documents
for (thisparty in unique(issue_agendas_supervised$party)) {
    for (thisdate in unique(issue_agendas_supervised$date[issue_agendas_supervised$party ==
        thisparty])) {
        for (thisissue in unique(issue_agendas_supervised$issue_r1)) {
            if (nrow(issue_agendas_supervised[issue_agendas_supervised$party == thisparty &
                issue_agendas_supervised$date == thisdate & issue_agendas_supervised$issue_r1 ==
                thisissue, ]) == 0 & nrow(issue_agendas_supervised[issue_agendas_supervised$party ==
                thisparty & issue_agendas_supervised$date == thisdate, ]) != 0) {
                issue_agendas_supervised <- data.frame(party = thisparty, date = thisdate,
                  issue_r1 = thisissue, freq = 0) %>%
                  rbind.fill(issue_agendas_supervised)
            }
        }
    }
}

# Add var for total press releases per party and month
issue_agendas_supervised$party_sum <- ave(issue_agendas_supervised$freq, issue_agendas_supervised$date,
    issue_agendas_supervised$party, FUN = sum)

issue_agendas_supervised$attention <- issue_agendas_supervised$freq/issue_agendas_supervised$party_sum

# Add issue descriptions
issue_agendas_supervised <- merge(issue_agendas_supervised, issue_categories, by = "issue_r1") %>%
    select(-c(freq))

save(issue_agendas_supervised, file = "supervised-files/issue_agendas_supervised.RData")
```

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

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

```
## Time difference of 2.317703 mins
```

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