Supervised learning aggregated

Cornelius Erfort

5/10/2021

Contents

1 Setting up

This script requires the files "sample_germany.dta" and "data_joint.RDS" in the parent directory. It also writes and reads two csv-files from the parent directory.

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", "randomForest", "adabag", "kableExtra")

# Using readstata13 because haven causes crash
```

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

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

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

1.2 Loading data

The sample data for Germany consists of 2,742 labelled press releases. The dataset is not on GitHub and is loaded from the parent directory here.

```
sample_germany <- read.dta13("../sample_germany.dta", convert.factors = F)

# Correcting classification for three documents
sample_germany$issue[sample_germany$id == 229] <- 191
sample_germany$issue[sample_germany$id == 731] <- 7
sample_germany$issue[sample_germany$id == 902] <- 10

# Subset to relevant vars
germany_textpress <- sample_germany %>%
    select("header", "text", "issue", "position", "id", "party")

# Distribution of issues in the hand-coded sample
table(germany_textpress$issue) %>%
    kbl(booktabs = T)
```

Var1	Freq
1	175
2	181
3	119
4	99
5	167
6	137
7	84
8	105
9	131
10	74
12	195
13	104
14	32
15	168
16	121
17	68
18	27
20	97
23	19
98	91
99	46
191	350
192	152

1.3 Merging categories

In order to improve the classification, similar topics are 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.

```
germany_textpress$issue_r1 <- as.numeric(germany_textpress$issue)

germany_textpress <- germany_textpress %>% mutate(issue_r1 = recode(issue_r1,
                           `8`  = 7,  # Environment & Energy
                           `13` = 10, # Transportation & Welfare
                           `14` = 10, # Housing & Welfare
                           `18` = 15, # Foreign Trade and Domestic Commerce
                           `98` = 99, # Non-thematic & Other
                           `23` = 99) # Culture: Too few observations
                                                )
# Category descriptions
issue_categories <- data.frame(issue_r1 = c(1:7, 9:10, 12, 15:17, 20, 99, 191:192), issue_r1_descr = c(

# Distribution with merged categories
table(germany_textpress$issue_r1) %>% kbl(booktabs = T)
```

Var1	Freq
1	175
2	181
3	119
4	99
5	167
6	137
7	189
9	131
10	210
12	195
15	195
16	121
17	68
20	97
99	156
191	350
192	152

```
# Party names
party_names <- data.frame(party = c(1:8), party_name = c("Bündnis 90/Die Grünen - Fraktion", "AfD - Bun
germany_textpress <- merge(germany_textpress, party_names, by = "party")

# Distribution by parties
table(germany_textpress$party_name) %>% kbl(booktabs = T)
```

Var1	Freq
AfD - Bundesverband	111
AfD - Fraktion	11
Bündnis 90/Die Grünen - Fraktion	484
CDU/CSU - Fraktion	393
DIE LINKE - Fraktion	639
FDP - Bundesverband	231
FDP - Fraktion	298
SPD - Fraktion	575

1.4 Creating the document frequency matrix (dfm)

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.

```
corp_press <- str_c(germany_textpress$header, " ", germany_textpress$text) %>% corpus()
```

```
## Warning: NA is replaced by empty string
```

```r
# Add id var to corpus
docvars(corp_press, "id") <- germany_textpress$id
docvars(corp_press, "issue_r1") <- germany_textpress$issue_r1
docvars(corp_press, "party_name") <- germany_textpress$party_name

# Create random sample for test dataset (size: 1/5 of all classified documents)
set.seed(300)
id_test <- sample(docvars(corp_press, "id"),
                  round(length(docvars(corp_press, "id"))/5, 0), replace = FALSE)

# Create dfm
dfmat <- corpus_subset(corp_press) %>%
  dfm(remove = stopwords("de"), # Stem and remove stopwords, punctuation etc.
      stem = T,
      remove_punct = T,
      remove_number = T,
      remove_symbols = T,
      remove_url = T) %>%
  dfm_trim(min_docfreq = 0.005, # Remove words occurring <.5% or > 80% of docs
           max_docfreq = .9,
           docfreq_type = "prop") %>%
  suppressWarnings()

# Create training and test set
dfmat_training <- dfm_subset(dfmat, !(id %in% id_test))
dfmat_test <- dfm_subset(dfmat, id %in% id_test)

# Make order of documents random
dfmat <- dfmat[sample(1:ndoc(dfmat), ndoc(dfmat)), ]
```

2 Textmodels

Following Barbera 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 <- 10

# Ten-fold cross-validation for every possible parameter combination
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)))
head(naivebayes_eval)
```

```
##   k  accuracy precision    recall  f1_score   prior distribution smooth time
## 1 1 0.6436364 0.6155723 0.6383209 0.6129346 uniform  multinomial      1 0.19
## 2 2 0.6094891 0.5875402 0.5986121 0.5787185 uniform  multinomial      1 0.15
## 3 3 0.6532847 0.6541336 0.6235606 0.6322591 uniform  multinomial      1 0.16
## 4 4 0.6386861 0.6216813 0.6472043 0.6216102 uniform  multinomial      1 0.24
## 5 5 0.6386861 0.6564675 0.6454688 0.6314168 uniform  multinomial      1 0.16
##         seed
## 1 1621897879
## 2 1621897879
## 3 1621897879
## 4 1621897879
## 5 1621897879
```

```
(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, ] %>%
    kbl(booktabs = T)
```

prior	distribution	smooth	accuracy	precision	recall	f1_score	time	seed
termfreq	multinomial	1	0.6454930	0.6382408	0.6363512	0.6247633	0.163	1621897879
docfreq	multinomial	1	0.6436695	0.6357764	0.6319803	0.6215195	0.196	1621897879
uniform	multinomial	1	0.6425800	0.6303078	0.6352878	0.6206623	0.173	1621897879
docfreq	multinomial	2	0.6405928	0.6451974	0.6173375	0.6134236	0.190	1621897879
termfreq	multinomial	2	0.6393046	0.6394152	0.6179870	0.6117133	0.162	1621897879

```
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).

```
# textmodel_svm, type 0 (primal) and 7 (dual)

# Ten-fold cross-validation for every possible parameter combination
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)))
head(ridge_eval)
```

```
##   k  accuracy precision    recall  f1_score  weight type time       seed
## 1 1 0.6218182 0.6282043 0.5878704 0.5963873 uniform    0 1.79 1621897918
## 2 2 0.6094891 0.6031396 0.5979235 0.5945100 uniform    0 1.57 1621897918
## 3 3 0.5912409 0.5478127 0.5528638 0.5466991 uniform    0 1.43 1621897918
## 4 4 0.5875912 0.5894278 0.5855695 0.5696273 uniform    0 1.40 1621897918
## 5 5 0.6167883 0.6251495 0.6010755 0.6074218 uniform    0 1.34 1621897918
```

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

weight	type	accuracy	precision	recall	f1_score	time	seed
docfreq	0	0.6088447	0.6407077	0.5734688	0.5803488	1.336667	1621897918
termfreq	7	0.6064881	0.6336722	0.5685160	0.5719545	2.226250	1621897918
uniform	0	0.6061327	0.6102538	0.5917826	0.5915829	1.434000	1621897918
uniform	7	0.6057691	0.6103275	0.5911772	0.5910533	2.732000	1621897918
docfreq	7	0.6056005	0.6391448	0.5702855	0.5763987	2.221111	1621897918

```
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.

```
# textmodel_svm, type 6

# Ten-fold cross-validation for every possible parameter combination
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))
head(lasso_eval)
```

```
##   k  accuracy precision    recall  f1_score  weight type time       seed
```

```
## 1 1 0.5563636 0.5370191 0.5388743 0.5287234 uniform    6 1.75 1621898035
## 2 2 0.5766423 0.5513401 0.5357696 0.5379345 uniform    6 1.75 1621898035
## 3 3 0.6204380 0.6149321 0.6002029 0.6012339 uniform    6 1.60 1621898035
## 4 4 0.6240876 0.6345778 0.6329724 0.6200666 uniform    6 1.70 1621898035
## 5 5 0.6204380 0.6131480 0.6088458 0.6033363 uniform    6 1.66 1621898035
```

```
(lasso_eval_aggr <- aggregate(cbind(accuracy, precision, recall, f1_score, time,
    seed) ~ weight, lasso_eval[, -c(1)], mean) %>%
    arrange(desc(accuracy)))[1:5, ] %>%
    kbl(booktabs = T)
```

	weight	accuracy	precision	recall	f1_score	time	seed
1	uniform	0.5984844	0.5943491	0.5849786	0.5796539	1.677	1621898035
2	docfreq	0.4744526	0.5106678	0.4219613	0.4361401	1.530	1621898035
NA	NA	NA	NA	NA	NA	NA	NA
NA.1	NA	NA	NA	NA	NA	NA	NA
NA.2	NA	NA	NA	NA	NA	NA	NA

```
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).

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

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)

elasticnet_eval$lambda.min
```

```
## [1] 0.0218569
```

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

```
## [1] 0.3832969
```

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

```
## [1] 0.6167031
```

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

# Estimate model
# elastic_net_time <- system.time(elasticnet_mod <- glmnet(x = dfmat_training,
#                     y = dfmat_training$issue_r1,
#                     family = "multinomial",
#                     alpha = 0.5,
#                     type.measure = "class",
#                     standardize = T))

elasticnet_mod <- glmnet(x = dfmat_training,
                    y = dfmat_training$issue_r1,
                    family = "multinomial",
                    alpha = 0.5,
                    type.measure = "class",
                    standardize = T)

# 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

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 = elastic_net_time,
           seed = seed,
           model = "Elastic net",
           alpha = .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.)

```
# textmodel_svm, type 4

# Ten-fold cross-validation for every possible parameter combination
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))
head(svm_eval)
```

```
##   k  accuracy precision    recall  f1_score  weight type time       seed
## 1 1 0.5854545 0.5925939 0.5792188 0.5790994 uniform    4 1.14 1621898156
## 2 2 0.5547445 0.5771948 0.5663527 0.5572186 uniform    4 1.06 1621898156
## 3 3 0.5839416 0.5817859 0.5921412 0.5798717 uniform    4 0.99 1621898156
## 4 4 0.5474453 0.5460784 0.5312947 0.5291938 uniform    4 1.04 1621898156
## 5 5 0.5839416 0.5788025 0.5786319 0.5693019 uniform    4 0.99 1621898156
```

```
(svm_eval_aggr <- aggregate(cbind(accuracy, precision, recall, f1_score, time, seed) ~
    weight, svm_eval[, -c(1)], mean) %>%
    arrange(desc(accuracy)))[1:5, ] %>%
    kbl(booktabs = T)
```

	weight	accuracy	precision	recall	f1_score	time	seed
1	docfreq	0.5886344	0.5857358	0.5802819	0.5732805	1.004	1621898156
2	termfreq	0.5882707	0.5851756	0.5804478	0.5728412	1.071	1621898156
3	uniform	0.5791493	0.5764690	0.5710138	0.5642426	1.060	1621898156
NA	NA	NA	NA	NA	NA	NA	NA
NA.1	NA	NA	NA	NA	NA	NA	NA

```
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
# randomforest_time <- system.time(randomforest_eval <-  randomForest(x = as.matrix(dfmat_training),
#                     y = as.factor(dfmat_training$issue_r1),
#                     xtest = as.matrix(dfmat_test),
#                     ytest = as.factor(dfmat_test$issue_r1),
#                     importance = T,
#                     mtry = 20,
#                     ntree = 200,
#                     keep.forest = T,
#                     type = "class"))
randomforest_eval <-  randomForest(x = as.matrix(dfmat_training),
                    y = as.factor(dfmat_training$issue_r1),
                    xtest = as.matrix(dfmat_test),
                    ytest = as.factor(dfmat_test$issue_r1),
                    importance = T,
                    mtry = 20,
                    ntree = 100,
                    keep.forest = T,
```

```
                    type = "class")

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

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

```
## $accuracy
## [1] 0.6076642
```

```
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_time,
           seed = seed,
           model = "Random forest",
           alpha = .5,
           distribution = "multinomial") %>%
  rbind.fill(tm_eval)
```

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

3 Ensemble methods: SuperLearner

Prepare the datasets for the python code chunk.

```
# Get vector indicating training sample
training <- !(1:ndoc(corp_press) %in% id_test)

# Subset training and test sample and write
if (!file.exists("../train.csv")) as.data.frame(as.matrix(dfmat_training, verbose = T)) %>%
    write.csv(., "../train.csv")
if (!file.exists("../test.csv")) as.data.frame(as.matrix(dfmat_test, verbose = T)) %>%
    write.csv(., "../test.csv")

# Get labels for training sample and write
if (!file.exists("../train_val.csv")) write.csv(dfmat_training$issue_r1, "../train_val.csv")
if (!file.exists("../test_val.csv")) write.csv(dfmat_test$issue_r1, "../test_val.csv")


super_start <- Sys.time()
```

Run SuperLearner in Python. (Additionally running AdaBoost.)

```
# Load pandas and numpy
import pandas as pd
import numpy as np

# Load sklearn tools
from sklearn.metrics import accuracy_score

# Load classifiers
from sklearn.naive_bayes import MultinomialNB # 1
from sklearn.naive_bayes import BernoulliNB
```

```
from sklearn.linear_model import LogisticRegression # 2, 3 and 4 (logistic regression L2/L1 penalty and
from sklearn.svm import SVC # 5
from sklearn.ensemble import RandomForestClassifier # 6
from sklearn.ensemble import AdaBoostClassifier # 7

# Load SuperLearner
from mlens.ensemble import SuperLearner

# Set params
# super_folds = r.folds_cv
super_folds = 10
# super_tol = .005
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 =
    models.append(LogisticRegression(solver = 'liblinear', max_iter = 1000, tol = super_tol, penalty =
    models.append(LogisticRegression(solver = 'saga', max_iter = 1000, penalty = 'elasticnet', l1_ratio
    models.append(SVC(probability = True, tol = super_tol))
    models.append(RandomForestClassifier())
    models.append(AdaBoostClassifier())
    return models

# Create the superlearner
def get_super_learner(X):
    ensemble = SuperLearner(scorer = None, folds = super_folds, shuffle = True, random_state = np.rando
    models = get_models() # Add base models
    ensemble.add(models, proba = True)
    ensemble.add_meta(LogisticRegression(solver = 'lbfgs', max_iter = 1000, tol = super_tol), proba = Fa
    return ensemble

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

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

# Create the inputs and outputs
# Split data
print('Train', train.shape, train_val.shape, 'Test', test.shape, test_val.shape)

# Create the super learner
ensemble = get_super_learner(train)

# Fit the super learner
ensemble.fit(train, train_val)

# Summarize base learners
```

```
print(ensemble.data)

# Make predictions on test data
super_pred = ensemble.predict(test)

print('Super Learner: %.3f' % (accuracy_score(test_val, super_pred) * 100))
```

```
## Train (2194, 3623) (2194,) Test (548, 3623) (548,)
##
## Fitting 2 layers
## Fit complete                         | 00:45:27
##                                       ft-m  ft-s  pt-m  pt-s
## layer-1  adaboostclassifier           8.99  0.06  0.10  0.00
## layer-1  bernoullinb                  2.71  0.05  0.01  0.00
## layer-1  logisticregression-1         0.56  0.02  0.00  0.00
## layer-1  logisticregression-2         0.31  0.02  0.00  0.00
## layer-1  logisticregression-3       138.54  5.25  0.00  0.00
## layer-1  multinomialnb                2.66  0.03  0.00  0.00
## layer-1  randomforestclassifier       5.65  0.58  0.05  0.08
## layer-1  svc                         82.09  0.45  1.35  0.02
##
##
## Predicting 2 layers
## Predict complete                     | 00:00:04
## [MLENS] backend: threading
## Super Learner: 65.328
```

```
# 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 = as.numeric(Sys.time() - super_start), model = 'SuperLearner ensemble')
# %>% rbind.fill(tm_eval)
```

The SuperLearner does not significantly increase the accuracy.

4 Evaluation of textmodels

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

```
# Comparison of textmodels
tm_eval[order(tm_eval$accuracy), ] %>%
    kbl(booktabs = T)
```

	accuracy	precision	recall	f1_score	seed	model	alpha	distribution	weight	
2	0.5886344	0.5857358	0.5802819	0.5732805	1621898156	SVM	NA	NA	docfreq	1.00
4	0.5984844	0.5943491	0.5849786	0.5796539	1621898035	Lasso (L1)	NA	NA	uniform	1.67
1	0.6076642	0.5977692	0.5681454	0.5586247	1621447882	Random forest	0.5	multinomial	NA	
5	0.6088447	0.6407077	0.5734688	0.5803488	1621897918	Ridge (L2)	NA	NA	docfreq	1.33
3	0.6094891	0.6246391	0.5824033	0.5950890	1621447882	Elastic net	0.5	multinomial	NA	
6	0.6454930	0.6382408	0.6363512	0.6247633	1621897879	naive bayes	NA	multinomial	NA	0.16

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

# Confusion matrix and overall statistics
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   19  1  0  0  3  0  0  0  1  0  3  0  0  0  1   0   3
##   2    0 17  0  0  1  2  0  1  5  6  0  0  0  1  1   2   0
##   3    0  2 21  1  1  1  0  0  0  1  0  0  0  0  0   0   0
##   4    0  0  1 15  0  0  2  0  0  0  2  0  0  0  0   0   1
##   5    2  0  3  0 24  0  0  0  0  1  2  0  0  0  0   0   0
##   6    0  1  0  0  1 21  0  0  1  0  0  0  1  1  1   0   0
##   7    2  1  1  1  0  0 20  0  0  0  2  1  0  1  1   0   2
##   9    0  1  0  0  1  2  0 22  0  4  1  0  0  0  0   2   0
##   10   0  1  0  2  2  2  3  1 21  0  3  1  1  0  1   0   1
##   12   0  1  0  1  0  1  0  1  0 23  3  0  0  1  0   1   1
##   15   4  0  1  3  0  2  0  0  1  1 20  0  3  2  2   3   1
##   16   0  1  0  0  0  0  0  0  0  2  1 13  0  1  1   6   0
##   17   0  1  0  0  0  2  0  1  0  1  1  0  2  0  1   1   1
##   20   0  0  0  0  1  2  1  0  2  2  1  3  1  8  0   0   1
##   99   0  1  1  0  0  2  0  0  1  1  1  0  1  3 11   1   1
##   191  0  1  0  1  0  1  5  0  0  2  0  3  1  0  0  63   3
##   192  0  0  0  0  0  0  0  1  0  0  0  0  0  0  1   7  23
##
## Overall Statistics
##
##                Accuracy : 0.6259
##                  95% CI : (0.5839, 0.6666)
##     No Information Rate : 0.1569
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5975
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.70370  0.58621  0.75000  0.62500  0.70588  0.55263
## Specificity           0.97697  0.96339  0.98846  0.98855  0.98444  0.98824
## Pos Pred Value        0.61290  0.47222  0.77778  0.71429  0.75000  0.77778
## Neg Pred Value        0.98453  0.97656  0.98656  0.98292  0.98062  0.96737
## Precision             0.61290  0.47222  0.77778  0.71429  0.75000  0.77778
## Recall                0.70370  0.58621  0.75000  0.62500  0.70588  0.55263
## F1                    0.65517  0.52308  0.76364  0.66667  0.72727  0.64615
## Prevalence            0.04927  0.05292  0.05109  0.04380  0.06204  0.06934
## Detection Rate        0.03467  0.03102  0.03832  0.02737  0.04380  0.03832
## Detection Prevalence  0.05657  0.06569  0.04927  0.03832  0.05839  0.04927
## Balanced Accuracy     0.84034  0.77480  0.86923  0.80677  0.84516  0.77043
```

```
##                      Class: 7 Class: 9 Class: 10 Class: 12 Class: 15 Class: 16
## Sensitivity           0.64516  0.81481   0.65625   0.52273   0.50000   0.61905
## Specificity           0.97679  0.97889   0.96512   0.98016   0.95472   0.97723
## Pos Pred Value        0.62500  0.66667   0.53846   0.69697   0.46512   0.52000
## Neg Pred Value        0.97868  0.99029   0.97839   0.95922   0.96040   0.98470
## Precision             0.62500  0.66667   0.53846   0.69697   0.46512   0.52000
## Recall                0.64516  0.81481   0.65625   0.52273   0.50000   0.61905
## F1                    0.63492  0.73333   0.59155   0.59740   0.48193   0.56522
## Prevalence            0.05657  0.04927   0.05839   0.08029   0.07299   0.03832
## Detection Rate        0.03650  0.04015   0.03832   0.04197   0.03650   0.02372
## Detection Prevalence  0.05839  0.06022   0.07117   0.06022   0.07847   0.04562
## Balanced Accuracy     0.81098  0.89685   0.81068   0.75144   0.72736   0.79814
##                      Class: 17 Class: 20 Class: 99 Class: 191 Class: 192
## Sensitivity            0.20000   0.44444   0.52381     0.7326    0.60526
## Specificity            0.98327   0.97358   0.97533     0.9632    0.98235
## Pos Pred Value         0.18182   0.36364   0.45833     0.7875    0.71875
## Neg Pred Value         0.98510   0.98099   0.98092     0.9509    0.97093
## Precision              0.18182   0.36364   0.45833     0.7875    0.71875
## Recall                 0.20000   0.44444   0.52381     0.7326    0.60526
## F1                     0.19048   0.40000   0.48889     0.7590    0.65714
## Prevalence             0.01825   0.03285   0.03832     0.1569    0.06934
## Detection Rate         0.00365   0.01460   0.02007     0.1150    0.04197
## Detection Prevalence   0.02007   0.04015   0.04380     0.1460    0.05839
## Balanced Accuracy      0.59164   0.70901   0.74957     0.8479    0.79381
```

5 Classification of unlabelled data

5.1 Using the NB textmodel

We trained the models using a set of 2,742 labelled documents. In order to obtain aggregated measures of issue attention, we predict the issue categories of all 47,111 labelled and unlabelled press releases in our sample.

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

# Loading full dataset from parent dir
all_germany <- read_rds("../data_joint.RDS") %>% select(c(header, text.x, date.x, issue, party.x, id))

# Constructing the document frequency matrix
dfmat_all <- corpus(str_c(all_germany$header, " ", all_germany$text.x)) %>%
  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()

# Adding docvars
docvars(dfmat_all, "party") <- all_germany$party.x
docvars(dfmat_all, "date") <- all_germany$date.x
docvars(dfmat_all, "id") <- all_germany$id

# Subsetting to features in the training data
```

```r
dfmat_all <- dfm_match(dfmat_all, features = featnames(dfmat_training))

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

table(dfmat_all$issue_r1) %>% kbl(booktabs = T)
```

Var1	Freq
1	2763
2	2746
3	1696
4	1941
5	2985
6	3146
7	3334
9	2348
10	3065
12	3635
15	2992
16	2011
17	1241
20	1541
99	2648
191	6012
192	3007

5.2 Aggregation of the issues categories over time and party

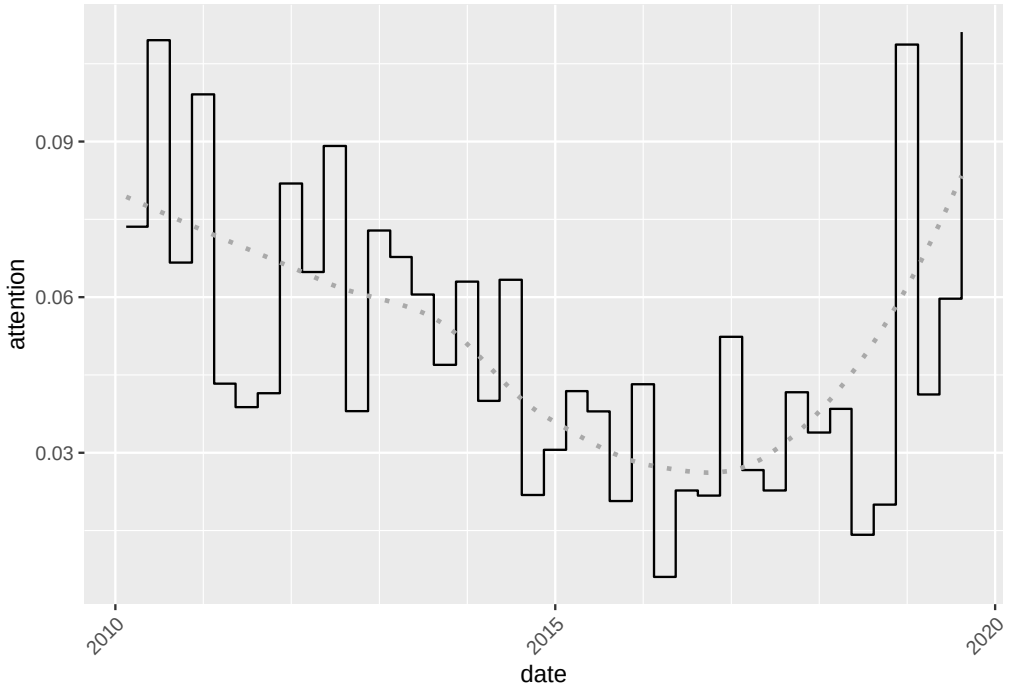
To measure parties' evolving issue agendas, we aggregate the category counts over time.

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

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

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

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

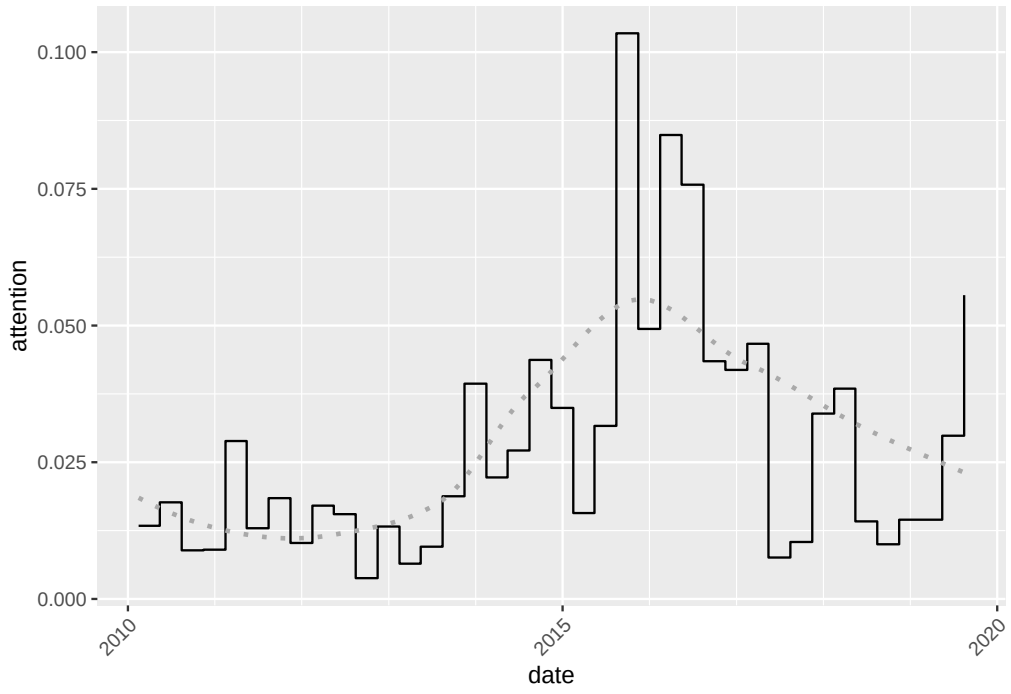
```r
issue_agendas$attention <- issue_agendas$freq/issue_agendas$party_sum

# Add issue descriptions
issue_agendas <- merge(issue_agendas, issue_categories, by = "issue_r1")
```

5.3 Plotting issue attention: Examples

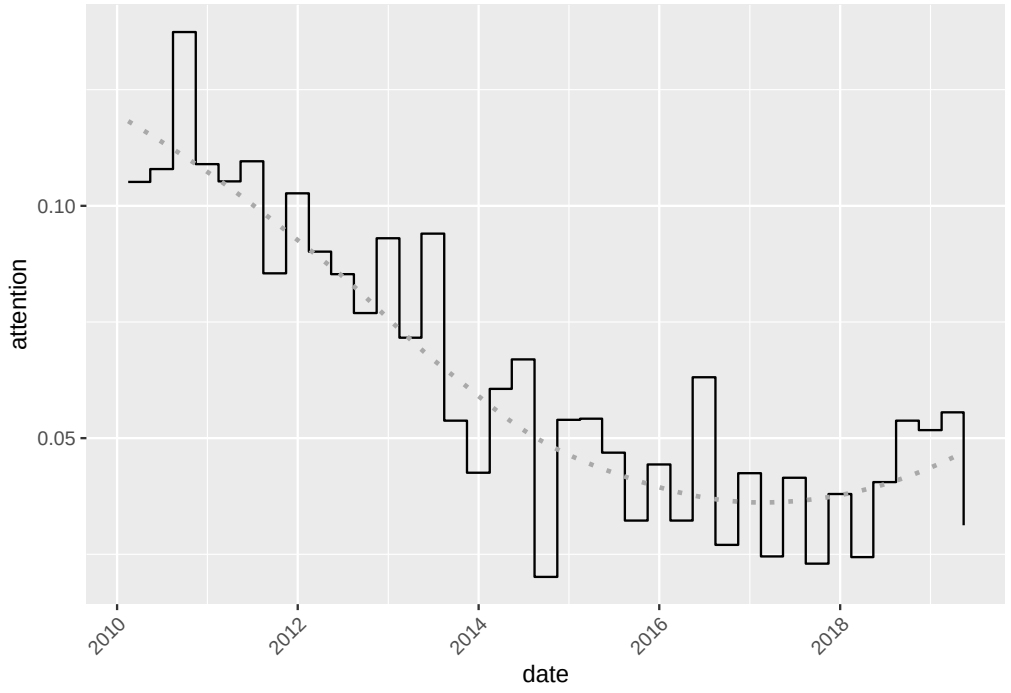
```r
if (!dir.exists("plots")) dir.create("plots")

# Function for plotting parties' issue attention over time
plot_issue_party <- function(plot_issue, plot_party) ggplot(issue_agendas %>%
    filter(issue_r1 == plot_issue & party == plot_party), aes(x = date, y = attention)) +
    geom_step() + geom_smooth(method = "loess", formula = "y ~ x", color = "dark grey",
    lty = 3, se = F) + theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    scale_x_date(date_minor_breaks = "1 year") + # ggtitle('Share of press releases for issue per quart
# issue_categories$issue_r1_descr[issue_categories$issue_r1 == plot_issue], ' -
# ', plot_party, ')')) +
ggsave(str_c("plots/", plot_issue, " - ", issue_categories$issue_r1_descr[issue_categories$issue_r1 ==
    plot_issue], "_", plot_party, ".pdf"), device = cairo_pdf, width = 5 * 2^0.5,
    height = 5) + ggsave(str_c("plots/", plot_issue, " - ", issue_categories$issue_r1_descr[issue_categ
    plot_issue], "_", plot_party, ".png"), width = 5 * 2^0.5, height = 5)


# Plot quarterly issue attention for category '7 Environment & Energy' for
# 'union_fraktion'
plot_issue_party(7, "union_fraktion")  # There seems to be a decline since Fukushima in 2011.
```



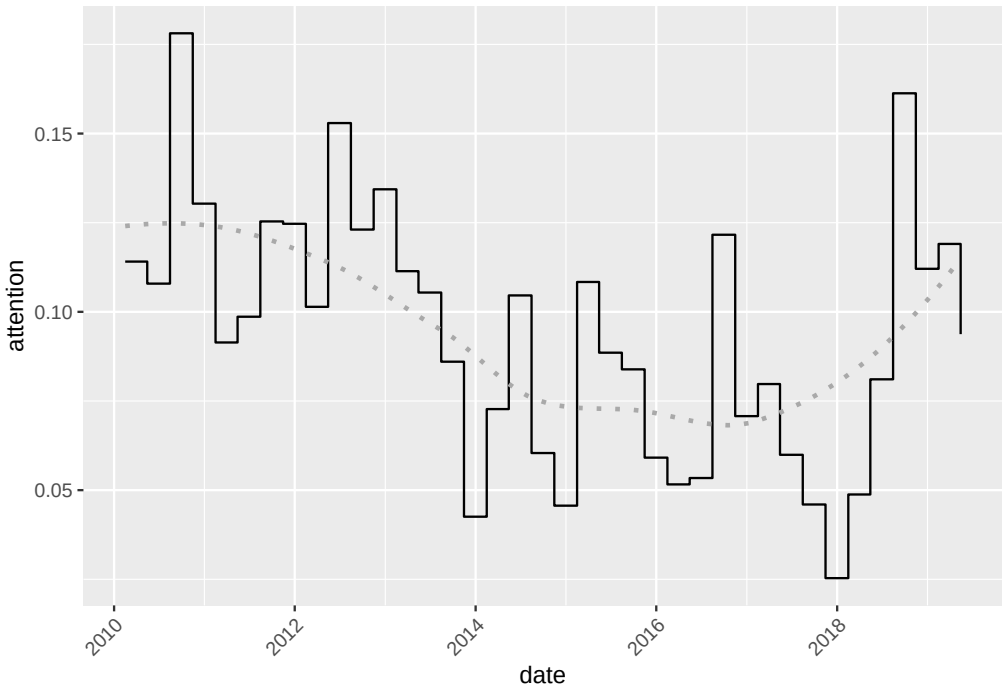

```r
# Plot quarterly issue attention for category '9 Immigration' for
# 'union_fraktion'
plot_issue_party(9, "union_fraktion")  # There seems to be a peak around the so-called refugee crisis i
```




```
# Plot quarterly issue attention for category '7 Environment & Energy' for
# 'spd_fraktion'
plot_issue_party(7, "spd_fraktion")  # There seems to be a decline since Fukushima in 2011.
```




```
# Plot quarterly issue attention for category '10 Welfare' for 'spd_fraktion'
plot_issue_party(10, "spd_fraktion")  # There seems to be a lower emphasis on welfare after the entry i
```




```
# Time needed to run script
print(Sys.time() - start_time)
```

```
## Time difference of 1.199129 hours
```