Supervised learning aggregated

Cornelius Erfort

5/10/2021

Contents

1 Setting up **1**
1.1 Loading packages . . . 1
1.2 Loading data . . . 2
1.3 Merging categories . . . 2
1.4 Creating the document frequency matrix (dfm) . . . 4

2 Textmodels **5**
2.1 Naive Bayes (multinomial) . . . 5
2.2 Ridge regression (L2) . . . 6
2.3 Lasso regression (L1) . . . 7
2.4 Elastic Net . . . 8
2.5 SVM . . . 9
2.6 Random Forest . . . 10

3 Ensemble methods: SuperLearner **10**

4 Evaluation of textmodels **13**

5 Classification of unlabelled data **15**
5.1 Using the NB textmodel . . . 15
5.2 Aggregation of the issues categories over time and party . . . 16
5.3 Plotting issue attention: Examples . . . 16

1 Setting up

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

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")

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

issue	1	2	3	4	5	6	7	8	9	10	12	13	14	15	16	17	18	20	23	98	99	191	192
n	175	181	119	99	167	137	84	105	131	74	195	104	32	168	121	68	27	97	19	91	44	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))
```

1.2 Loading data

The sample data for Germany consists of 2,740 labelled press releases. The dataset is not uploaded on GitHub.

```
sample_germany <- read.dta13("data/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

sample_germany <- filter(sample_germany, date != "NA" | !is.na(text))
nrow(sample_germany)
```

```
## [1] 2740
```

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

# Distribution of issues in the hand-coded sample
table(germany_textpress$issue) %>%
    as.data.frame() %>%
    dplyr::rename(issue = Var1, n = Freq) %>%
    t() %>%
    kbl(booktabs = T) %>%
    kable_styling(latex_options = "scale_down")
```

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
                                       )
```

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	154	350	152

```
# Category descriptions
issue_categories <-
  data.frame(issue_r1 = c(1:7, 9:10, 12, 15:17, 20, 99, 191:192),
             issue_r1_descr = c("Macroeconomics", "Civil Rights",
                                "Health", "Agriculture", "Labor", "Education", "Env. & Energy",
                                "Immigration", "Welfare", "Law & Crime", "Commerce", "Defense",
                                "Technology", "Gov. Ops.", "Other", "Int. Aff.", "EU"))

issue_categories %>% dplyr::rename("Issue number" = issue_r1, "Issue name" = issue_r1_descr) %>%
  kbl(booktabs = T)
```

Issue number	Issue name
1	Macroeconomics
2	Civil Rights
3	Health
4	Agriculture
5	Labor
6	Education
7	Env. & Energy
9	Immigration
10	Welfare
12	Law & Crime
15	Commerce
16	Defense
17	Technology
20	Gov. Ops.
99	Other
191	Int. Aff.
192	EU

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

```
# Party names
party_names <- data.frame(party = c(1:8),
                          party_name = c("Bündnis 90/Die Grünen - Fraktion",
                                         "AfD - Bundesverband", "AfD - Fraktion",
                                         "FDP - Bundesverband", "FDP - Fraktion", "DIE LINKE - Fraktion
                                         "SPD - Fraktion", "CDU/CSU - Fraktion"))
germany_textpress <- merge(germany_textpress, party_names, by = "party")

# Distribution by parties
table(germany_textpress$party_name) %>% as.data.frame() %>%
  dplyr::rename(party = Var1, n = Freq) %>% kbl(booktabs = T)
```

party	n
AfD - Bundesverband	111
AfD - Fraktion	11
Bündnis 90/Die Grünen - Fraktion	482
CDU/CSU - Fraktion	393
DIE LINKE - Fraktion	639
FDP - Bundesverband	231
FDP - Fraktion	298
SPD - Fraktion	575

```
table(germany_textpress$party_name, substr(germany_textpress$date, 6, 10)) %>%
  as.data.frame.matrix() %>% kbl(booktabs = T)
```

	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
AfD - Bundesverband	0	0	0	4	22	10	21	23	19	12
AfD - Fraktion	0	0	0	0	0	0	0	11	0	0
Bündnis 90/Die Grünen - Fraktion	90	77	61	49	45	44	44	22	38	12
CDU/CSU - Fraktion	64	67	49	53	52	36	38	24	10	0
DIE LINKE - Fraktion	88	78	73	82	66	67	53	58	54	20
FDP - Bundesverband	13	14	12	5	45	33	41	39	19	10
FDP - Fraktion	62	66	64	47	0	0	0	6	33	20
SPD - Fraktion	112	90	95	63	54	51	36	32	30	12

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.

```
if(!dir.exists("train-test-data")) dir.create("train-test-data") # Delete folder when training and test
if(file.exists("train-test-data/dfmat.RData") & file.exists("train-test-data/dfmat_training.RData") & fi
  load("train-test-data/dfmat.RData")
  load("train-test-data/dfmat_training.RData")
  load("train-test-data/dfmat_test.RData")
  } else {
  corp_press <- str_c(germany_textpress$header, " ", germany_textpress$text) %>% corpus()

# 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 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()

# Make order of documents random
dfmat <- dfmat[sample(1:ndoc(dfmat), ndoc(dfmat)), ]
save(dfmat, file = "train-test-data/dfmat.RData")

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

# Create training and test set
dfmat_training <- dfm_subset(dfmat, !(id %in% id_test))
save(dfmat_training, file = "train-test-data/dfmat_training.RData")
dfmat_test <- dfm_subset(dfmat, id %in% id_test)
save(dfmat_test, file = "train-test-data/dfmat_test.RData")
}
```

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 <- 10  # Ten-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.

# Ten-fold cross-validation for every possible parameter combination
if (!dir.exists("textmodels")) dir.create("textmodels")
filename <- "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	0.645	0.626	0.628	0.618	0.5370000	1621984239
termfreq	multinomial	1	0.644	0.624	0.623	0.615	0.3910000	1621984239
docfreq	multinomial	1	0.642	0.624	0.621	0.613	0.4230000	1621984239
uniform	multinomial	2	0.639	0.631	0.613	0.608	0.3033333	1621984239
termfreq	multinomial	2	0.639	0.632	0.608	0.603	0.3922222	1621984239

```
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).

```
# Ten-fold cross-validation for every possible parameter combination
filename <- "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
termfreq	0	0.620	0.646	0.581	0.584	2.448000	1621984323
docfreq	7	0.613	0.648	0.581	0.585	3.621429	1621984323
termfreq	7	0.611	0.646	0.576	0.580	3.816667	1621984323
docfreq	0	0.611	0.639	0.576	0.579	2.365000	1621984323
uniform	0	0.605	0.609	0.591	0.587	3.353000	1621984323

```
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.

```
# Ten-fold cross-validation for every possible parameter combination
filename <- "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
1	uniform	0.6	0.603	0.587	0.584	2.869	1621984528
NA	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)

filename <- "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] 0.02290444
```

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

```
## [1] 0.3781022
```

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

```
## [1] 0.6218978
```

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

# Estimate model
filename <- "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.)

```
# Ten-fold cross-validation for every possible parameter combination
filename <- "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
uniform	0.577	0.577	0.564	0.562	1.845	1621984731
docfreq	0.575	0.576	0.564	0.561	1.626	1621984731
termfreq	0.575	0.576	0.564	0.560	1.657	1621984731

```
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 <- "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] 0.6058394
```

```
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)
```

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.

```
# Write training and test samples to csv (for Python to read)
if (!file.exists("train-test-data/train.csv")) as.data.frame(as.matrix(dfmat_training,
    verbose = T)) %>%
    write.csv(., "train-test-data/train.csv")
if (!file.exists("train-test-data/test.csv")) as.data.frame(as.matrix(dfmat_test,
    verbose = T)) %>%
    write.csv(., "train-test-data/test.csv")

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

```
    "train-test-data/test_val.csv")
```

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 = 10
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
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("train-test-data/train.csv", index_col = 0).values
train_val = np.asarray([int(i) for i in pd.read_csv("train-test-data/train_val.csv",
index_col = 0).values])

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

# Create the super learner
ensemble = get_super_learner(train)

# Fit the super learner
filename = "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 = "textmodels/super_pred.PyData"
if os.path.exists(filename) != True:
  super_pred = ensemble.predict(test)
  pk.dump(super_pred, open(filename, 'wb'))
  np.savetxt("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         17.34   2.20  0.20  0.01
## layer-1  bernoullinb                 3.16   0.10  0.02  0.01
## layer-1  logisticregression-1        0.97   0.14  0.01  0.00
## layer-1  logisticregression-2        0.47   0.02  0.00  0.00
## layer-1  logisticregression-3      202.04  25.20  0.01  0.00
## layer-1  multinomialnb               2.65   0.21  0.01  0.01
## layer-1  randomforestclassifier      9.64   1.83  0.06  0.02
## layer-1  svc                       165.79  64.42  3.38  2.84
```

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

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

```
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, but the necessary computing time is much greater.

4 Evaluation of textmodels

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

```
# Comparison of textmodels
tm_eval[, c("accuracy", "precision", "recall", "f1_score")] <- apply(tm_eval[, c("accuracy",
    "precision", "recall", "f1_score")], MARGIN = 2, function(x) round(x, 3))
tm_eval[order(tm_eval$accuracy, decreasing = T), c("model", "accuracy", "precision",
    "recall", "f1_score", "distribution")] %>%
    kbl(booktabs = T, row.names = F)
```

model	accuracy	precision	recall	f1_score	distribution
SuperLearner ensemble	0.682	0.700	0.670	0.674	NA
Naive bayes	0.645	0.626	0.628	0.618	multinomial
Elastic net	0.633	0.626	0.601	0.609	multinomial
Ridge (L2)	0.620	0.646	0.581	0.584	NA
Random forest	0.606	0.637	0.562	0.566	multinomial
Lasso (L1)	0.600	0.603	0.587	0.584	NA
SVM	0.577	0.577	0.564	0.562	NA

```
# 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   23  0  0  0  2  1  1  0  2  0  1  0  2  0  1   1   0
##   2    1 11  0  0  2  0  0  2  5  3  0  0  1  3  1   1   1
##   3    0  1 13  2  2  2  2  0  0  0  0  0  0  0  0   0   0
##   4    0  0  0 13  0  0  1  0  0  0  0  0  0  0  0   0   0
##   5    3  1  0  0 22  1  0  1  0  0  2  0  0  0  0   0   0
##   6    0  0  0  0  0 21  0  0  3  0  0  0  0  0  0   0   0
##   7    0  0  0  0  0  0 36  0  1  1  1  0  0  0  1   1   0
##   9    0  0  0  0  0  2  0 15  0  0  1  0  0  0  0   1   1
##   10   1  1  0  0  5  2  5  0 22  0  1  0  0  0  2   0   1
##   12   1  1  0  0  0  0  0  1  0 36  2  1  0  1  1   1   2
##   15   0  0  0  2  0  1  1  0  2  3 24  0  1  1  1   3   2
##   16   0  0  0  0  0  0  0  0  0  4  0 16  0  0  0   6   1
##   17   3  1  0  0  0  0  0  0  0  1  2  0  6  0  0   0   0
##   20   1  2  0  0  0  0  0  0  2  1  1  1  0  7  1   0   1
##   99   0  2  0  0  2  5  0  0  0  0  0  0  2  4 15   1   2
```

```
##   191  1  2  0  2  0  1  3  0  0  0  1  0  0  0  1  65   3
##   192  1  0  0  0  0  0  0  3  0  0  2  0  0  0  3   2  24
##
## Overall Statistics
##
##                Accuracy : 0.6734
##                  95% CI : (0.6323, 0.7125)
##     No Information Rate : 0.1496
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6479
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.65714  0.50000  1.00000  0.68421  0.62857  0.58333
## Specificity           0.97856  0.96198  0.98318  0.99811  0.98441  0.99414
## Pos Pred Value        0.67647  0.35484  0.59091  0.92857  0.73333  0.87500
## Neg Pred Value        0.97665  0.97872  1.00000  0.98876  0.97490  0.97137
## Precision             0.67647  0.35484  0.59091  0.92857  0.73333  0.87500
## Recall                0.65714  0.50000  1.00000  0.68421  0.62857  0.58333
## F1                    0.66667  0.41509  0.74286  0.78788  0.67692  0.70000
## Prevalence            0.06387  0.04015  0.02372  0.03467  0.06387  0.06569
## Detection Rate        0.04197  0.02007  0.02372  0.02372  0.04015  0.03832
## Detection Prevalence  0.06204  0.05657  0.04015  0.02555  0.05474  0.04380
## Balanced Accuracy     0.81785  0.73099  0.99159  0.84116  0.80649  0.78874
##                      Class: 7 Class: 9 Class: 10 Class: 12 Class: 15 Class: 16
## Sensitivity           0.73469  0.68182   0.59459   0.73469   0.63158   0.88889
## Specificity           0.98998  0.99049   0.96477   0.97796   0.96667   0.97925
## Pos Pred Value        0.87805  0.75000   0.55000   0.76596   0.58537   0.59259
## Neg Pred Value        0.97436  0.98674   0.97047   0.97405   0.97239   0.99616
## Precision             0.87805  0.75000   0.55000   0.76596   0.58537   0.59259
## Recall                0.73469  0.68182   0.59459   0.73469   0.63158   0.88889
## F1                    0.80000  0.71429   0.57143   0.75000   0.60759   0.71111
## Prevalence            0.08942  0.04015   0.06752   0.08942   0.06934   0.03285
## Detection Rate        0.06569  0.02737   0.04015   0.06569   0.04380   0.02920
## Detection Prevalence  0.07482  0.03650   0.07299   0.08577   0.07482   0.04927
## Balanced Accuracy     0.86234  0.83616   0.77968   0.85632   0.79912   0.93407
##                      Class: 17 Class: 20 Class: 99 Class: 191 Class: 192
## Sensitivity            0.50000   0.43750   0.55556     0.7927    0.63158
## Specificity            0.98694   0.98120   0.96545     0.9700    0.97843
## Pos Pred Value         0.46154   0.41176   0.45455     0.8228    0.68571
## Neg Pred Value         0.98879   0.98305   0.97670     0.9638    0.97271
## Precision              0.46154   0.41176   0.45455     0.8228    0.68571
## Recall                 0.50000   0.43750   0.55556     0.7927    0.63158
## F1                     0.48000   0.42424   0.50000     0.8075    0.65753
## Prevalence             0.02190   0.02920   0.04927     0.1496    0.06934
## Detection Rate         0.01095   0.01277   0.02737     0.1186    0.04380
## Detection Prevalence   0.02372   0.03102   0.06022     0.1442    0.06387
## Balanced Accuracy      0.74347   0.70935   0.76050     0.8813    0.80501
```

The Naive Bayes model reaches a high accuracy of ~.64 which can only be slightly improved using an ensemble

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

of classifiers (~.68). 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 labelled 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.

5 Classification of unlabelled data

5.1 Using the NB textmodel

We trained the models using a set of 2,740 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 hidden dir
all_germany <- read_rds("data/data_joint.RDS") %>% select(c(header, text.x, date.x, issue, party.x, id))
nrow(all_germany)
```

```
## [1] 47111
```

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

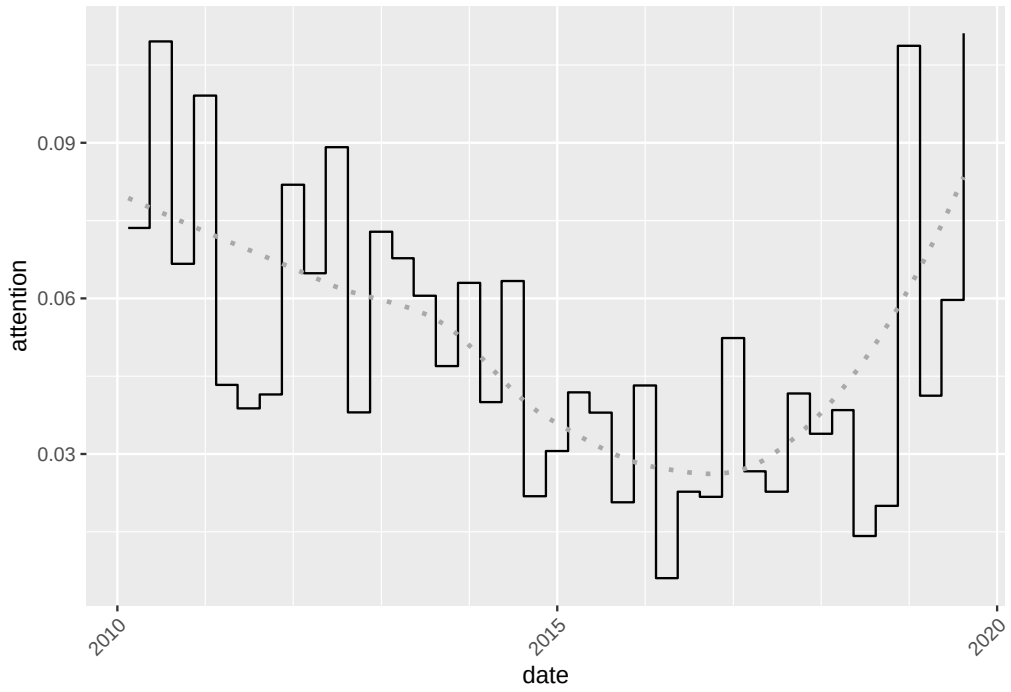
5.2 Aggregation of the issues categories over time and party

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

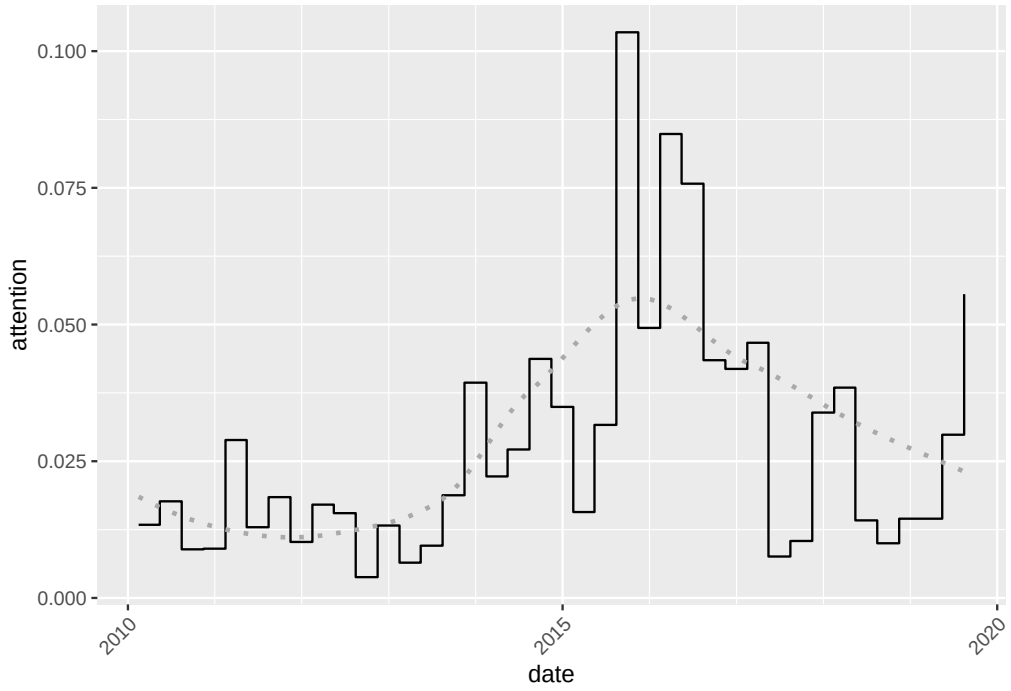
```
# Create dataframe from dfm
issue_agendas <- data.frame(date = docvars(dfmat_all, "date"), party = docvars(dfmat_all,
    "party"), issue_r1 = docvars(dfmat_all, "issue_r1"))

# 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)

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

```
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") + 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$issue_r1_descr[issue_categories$issue_r1
        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")
```

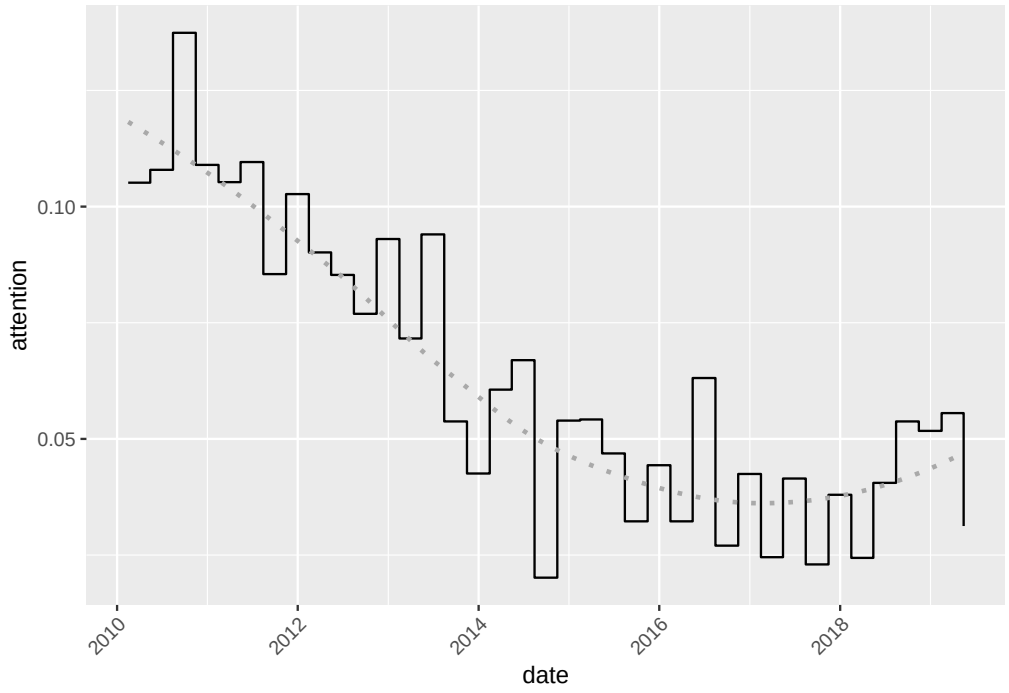



```
# There seems to be a decline since Fukushima in 2011.

# Plot quarterly issue attention for category '9 Immigration' for
# 'union_fraktion'
plot_issue_party(9, "union_fraktion")
```

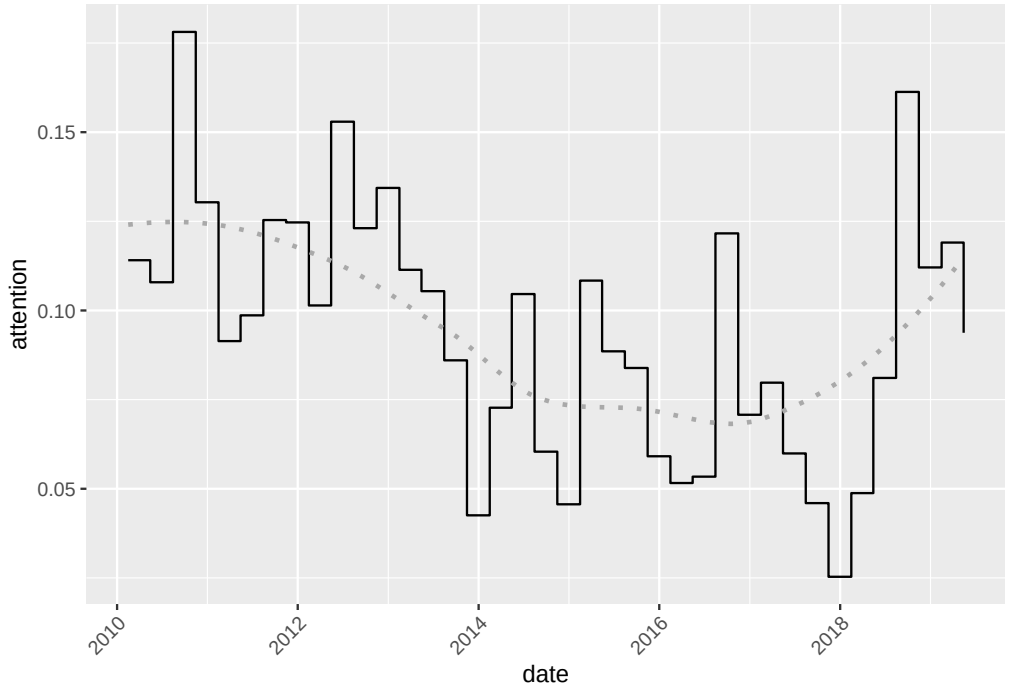



```
# There seems to be a peak around the so-called refugee crisis in 2015.

# 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 into government
# in fall 2013.
```

```
# Time needed to run script (much shorter when textmodels are just loaded from a
# file)
print(Sys.time() - start_time)
```

```
## Time difference of 1.653533 mins
```