Machine learning log

Larissa Bouwknegt

2023-10-06

When loading the data in Weka a filter (filters -> unsupervised -> NumericToNominal) needs to be applied to the age atribute to make sure the age atribute is nominal instead of numeric. Saving as an arff file prevents us to have to repeat this action every time. The most common algorithms will be run with 10 fold cross validation and be added to a table to see their individual performances without changed parameters.

```
algorithm_scores <- data.frame(algorithm = c("ZeroR", "OneR"),
                               Percentage_Correct = c(64.9781, 73.9779),
                               Percentage_Wrong = c(35.0219, 26.0221)
                               )
naive_bayes <- c("Naive Bayes", 68.1409, 31.8591)
algorithm_scores <- rbind(algorithm_scores, naive_bayes)
j48 <- c("j48", 75.495, 24.505)
algorithm_scores <- rbind(algorithm_scores, j48)
IBk <- c("IBk", 70.5837, 29.4163)
algorithm_scores <- rbind(algorithm_scores, IBk)
logistic <- c("Logistic", 77.7064, 22.2936)
algorithm_scores <- rbind(algorithm_scores, logistic)
suport_vector_machine <- c("SMO", 76.4464, 23.5536)
algorithm_scores <- rbind(algorithm_scores, suport_vector_machine)
qda <- c("QDA (sex atribute removed)", 75.7521, 24.2479)
algorithm_scores <- rbind(algorithm_scores, qda)
bagging <- c("Bagging", 77.7578, 22.2422)
algorithm_scores <- rbind(algorithm_scores, bagging)
voting <- c("Vote", 64.9781, 35.0219)
algorithm_scores <- rbind(algorithm_scores, voting)
adaboost <- c("AdaBoostM1", 72.1265, 27.8735)
algorithm_scores <- rbind(algorithm_scores, adaboost)
stacking <- c("Stacking", 64.9781, 35.0219)
algorithm_scores <- rbind(algorithm_scores, stacking)
random_forest <- c("Random Forest", 77.9121, 22.0879)
algorithm_scores <- rbind(algorithm_scores, random_forest)
clustering_class <- c("ClassificationViaClustering", 62.4839, 37.5161)
algorithm_scores <- rbind(algorithm_scores, clustering_class)
knitr::kable(algorithm_scores[order(algorithm_scores$Percentage_Correct, decreasing = TRUE),], row.names
```

algorithm	Percentage_Correct	Percentage_Wrong
Random Forest	77.9121	22.0879
Bagging	77.7578	22.2422
Logistic	77.7064	22.2936
SMO	76.4464	23.5536
QDA (sex atribute removed)	75.7521	24.2479

algorithm	Percentage_Correct	Percentage_Wrong
j48	75.495	24.505
OneR	73.9779	26.0221
AdaBoostM1	72.1265	27.8735
IBk	70.5837	29.4163
Naive Bayes	68.1409	31.8591
ZeroR	64.9781	35.0219
Vote	64.9781	35.0219
Stacking	64.9781	35.0219
ClassificationViaClustering	62.4839	37.5161

The table above is sorted on the best scoring algorithms without any parameters changed except for the QDA where the gender of the crabs was removed. All the tests were done with 10 fold cross validation. 3 algorithms scored even lower then or equal to zeroR which is interesting considering that it just takes the most prominent value. When further investigating the exact 2 scores same scores of voting and stacking to ZeroR the conclusion fell that we needed to specify the algorithms there. So these 2 were rerun with 11 algorithms, all of the above except for voting and stacking self and QDA was also not included due to the fact that we want to use the sex attribute.

```
rerun <- data.frame(algorithm = c("Voting",
                                  "Stacking with j48 meta",
                                  "stacking with random forest meta"),
                    Percentage_Correct = c(76.6264, 76.4207, 78.272),
                    Percentage_Wrong = c(23.3736, 23.5793, 21.728))

knitr::kable(rerun)
```

algorithm	Percentage_Correct	Percentage_Wrong
Voting	76.6264	23.3736
Stacking with j48 meta	76.4207	23.5793
stacking with random forest meta	78.2720	21.7280

The results gained from voting and stacking seems to be much better now. The meta classifier used for stacking was a j48 tree and after that a run with a random forest meta learner. The best scoring algorithms are: Stacking with a random forest, Random forest, bagging, logistic and voting. For now we will look further into these 5 algorithms and see if we can optimize the parameters to prevent over fitting and get better results.

Before tweaking the algorithms let's see if the dataset can be tweaked for optimal performace. In weka in the select atributes tab the CfsSubsetEval with exhaustive search forward and backward and no different changes (2 different runs) both give Selected attributes: 1,3,4,7,8 Sex, Diameter, Height, Viscera.Weight, Shell.Weight. A gainratio select atributes with the ranker search method gives the following results: 0.0813 8 Shell.Weight 0.0726 7 Viscera.Weight 0.0721 3 Diameter 0.0674 5 Weight 0.066 2 Length 0.0629 4 Height 0.0579 6 Shucked.Weight 0.0574 1 Sex Lets see if these attributes are also selected for the individual algorithms when WrapperSubsetEval is run for each one with BestFirst and backwards as option within BestFirst. Since Random forest are build based on Random trees we will use the random tree here for a shorter calculation time. For the reason of time consumption voting and stacking were also not run through this.

```
selected_results <- data.frame(algorithm = c("Random Tree",
                                             "Logistic",
                                             "Bagging"),
                               selected_atributes = c("Sex, Length,
                               Weight, Shucked.Weight,
                               Shell.Weight",
                               "Weight, Shucked.Weight, Shell.Weigh",
                               "Sex, Weight, Shucked.Weight,
                               Shell.Weight"))
knitr::kable(selected_results, format = "simple")
```

algorithm	selected_atributes
Random Tree	Sex, Length, Weight, Shucked.Weight, Shell.Weight
Logistic	Weight, Shucked.Weight, Shell.Weigh
Bagging	Sex, Weight, Shucked.Weight, Shell.Weight
4 sub sets are	made, based on the 3 algorithms and the cfssubseteval. These are then tested in the explorer against each o

The following are the results of a T test where the different datasets are tested with the percentage correct for the algorithms. This shows that the 4th smaller dataset scores significantly worse in all the algorithms and the 2nd and 3rd cleaned datasets when it comes to a random forest.

```
Dataset                    (1) cleaned_ | (2) 'clea (3) 'clea (4) 'clea (5) 'clea
-------------------------------------------------------------------------------
meta.Bagging '-P 100 -S 1(100)   77.34 |   77.36     76.93     77.34     75.31 *
meta.Stacking '-X 10 -M \(100)   78.08 |   77.74     77.02     77.65     75.07 *
trees.RandomForest '-P 10(100)   77.74 |   77.31     75.99 *   76.56 *   75.08 *
meta.Vote '-S 1 -B \"weka(100)   76.70 |   77.18     76.14     76.94     75.14 *
functions.Logistic '-R 1.(100)   77.76 |   77.16     77.25     77.12     74.42 *
-------------------------------------------------------------------------------
                              (v/ /*) |   (0/5/0)   (0/4/1)   (0/4/1)   (0/0/5)
```

The are under the curve shows us that the area under the curve is even significantly worse with all the smaller datasets for 2 different algorithms. The last smaller dataset is worse for all algorithms.

```
Dataset                  (1) cleaned | (2) 'cle (3) 'cle (4) 'cle (5) 'cle
-------------------------------------------------------------------------
meta.Bagging '-P 100 -S 1(100)   0.84 |   0.84     0.84     0.84     0.82 *
meta.Stacking '-X 10 -M \(100)   0.85 |   0.85     0.84     0.85     0.82 *
trees.RandomForest '-P 10(100)   0.85 |   0.84 *   0.83 *   0.84 *   0.82 *
meta.Vote '-S 1 -B \"weka(100)   0.83 |   0.84     0.83     0.84     0.82 *
functions.Logistic '-R 1.(100)   0.85 |   0.84 *   0.84 *   0.84 *   0.81 *
-------------------------------------------------------------------------
                            (v/ /*) |  (0/3/2)  (0/3/2)  (0/3/2)  (0/0/5)


Key:
(1) cleaned_data-unsupervised.attribute.NumericToNominal-Rlast
(2) 'cleaned_data-unsupervised.attribute.NumericToNominal-Rlast-unsupervised.attribute.Remove-R3-4,7'
(3) 'cleaned_data-unsupervised.attribute.NumericToNominal-Rlast-unsupervised.attribute.Remove-R1-4,7'
(4) 'cleaned_data-unsupervised.attribute.NumericToNominal-Rlast-unsupervised.attribute.Remove-R2-4,7'
(5) 'cleaned_data-unsupervised.attribute.NumericToNominal-Rlast-unsupervised.attribute.Remove-R2,5-6'
```

So lets focus on how the 5 algorithms scored compared to the whole dataset. Since stacking has the highest score this algorithm is used as the comparison.

```
Dataset                  (2) meta.Sta | (1) meta. (3) trees (4) meta. (5) funct
-------------------------------------------------------------------------------
cleaned_data-weka.filters(100)   78.08 |   77.34     77.74     76.70 *   77.76
'cleaned_data-weka.filter(100)   77.74 |   77.36     77.31     77.18     77.16
'cleaned_data-weka.filter(100)   77.02 |   76.93     75.99     76.14     77.25
'cleaned_data-weka.filter(100)   77.65 |   77.34     76.56     76.94     77.12
'cleaned_data-weka.filter(100)   75.07 |   75.31     75.08     75.14     74.42
-------------------------------------------------------------------------------
                            (v/ /*) |   (0/5/0)   (0/5/0)   (0/4/1)   (0/5/0)


Key:
(1) meta.Bagging '-P 100 -S 1 -num-slots 1 -I 10 -W trees.REPTree -- -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0' -11587996
(2) meta.Stacking '-X 10 -M \"trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1\" -S 1 -num-sl
(3) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
(4) meta.Vote '-S 1 -B \"trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1\" -B \"trees.J48 -C
(5) functions.Logistic '-R 1.0E-8 -M -1 -num-decimal-places 4' 3932117032546553727
```

```
Dataset                    (2) meta.St | (1) meta (3) tree (4) meta (5) func
-------------------------------------------------------------------------------
cleaned_data-weka.filters(100)    0.85 |   0.84     0.85     0.83 *   0.85
'cleaned_data-weka.filter(100)    0.85 |   0.84     0.84     0.84 *   0.84
'cleaned_data-weka.filter(100)    0.84 |   0.84     0.83 *   0.83 *   0.84
'cleaned_data-weka.filter(100)    0.85 |   0.84     0.84 *   0.84 *   0.84
'cleaned_data-weka.filter(100)    0.82 |   0.82     0.82     0.82     0.81
-------------------------------------------------------------------------------
                              (v/ /*) |  (0/5/0)  (0/3/2)  (0/1/4)  (0/5/0)
```

Vote scores significantly worse then stacking. When looking at the area under the curve stacking scores worse as well on the original dataset.

So lets see if we can tweak the 4 algorithms without vote a bit more on the original dataset.

Let's start with the random forest, since it is build on random trees we will try to improve an individual tree before

```
tree_scores <- data.frame(tree_depth = c("unlimited",
                                         "5",
                                         "6",
                                         "7",
                                         "8"),
                          percentage_corect = c(70.6351,
                                                74.1322,
                                                75.0064,
                                                75.6236,
                                                74.775))

knitr::kable(tree_scores, format = "simple")
```

tree_depth	percentage_corect
unlimited	70.6351
5	74.1322
6	75.0064
7	75.6236
8	74.7750

tree_depth
5
6
7
8
The table sho ws the changes to the depth of the tree parameter and how they corresponded to the percentage correct wi

By bagging we can change the number of iterations:

```
bagging_itter <- data.frame(itterations = c("10",
                                            "100",
                                            "1000"),
                            percentage_correct = c(77.7578,
                                                   78.2206,
                                                   78.1435))

knitr::kable(bagging_itter, format = "simple")
```

itterations
10
100
1000
Changing the n umber of iterations improves it when it is set to a 100 but decreases when set to a 1000. A 1000 iteration

Stacking seems to improve when using less and simpler algorithms. Also with j48 as a meta learner. After some more experimenting with different algorithms and settings the highest percentage seems to be 79.0177 correct with an AUC of 0,817. The setting used are the following base algorithms: Random forest with depth of 7, logistic, IBk with 3 neighbours, bagging with 100 iterations and OneR. The meta learner used was J48 without any changes to the settings. Using less algorithms from different categories and improving their settings led to this.

Since the 2 group in age is under represented a costsentive run was also tried where the matrix was changed to give 2 points to a 2 predicted as a 1 instead of a penalty of 1. This seemed to lower the overall score so no further experimentations were done on this area since a penalty of 2 was not much but lowered the score and did not change much in the actual group 2 predictions.

The only thing that makes sense to change with the logistic algorithm is the number of iterations, yet this does not improve the algorithm. A lower iteration maximum decreases the score and eventually you reach a threshold where the score is the same as not changing the maximum.

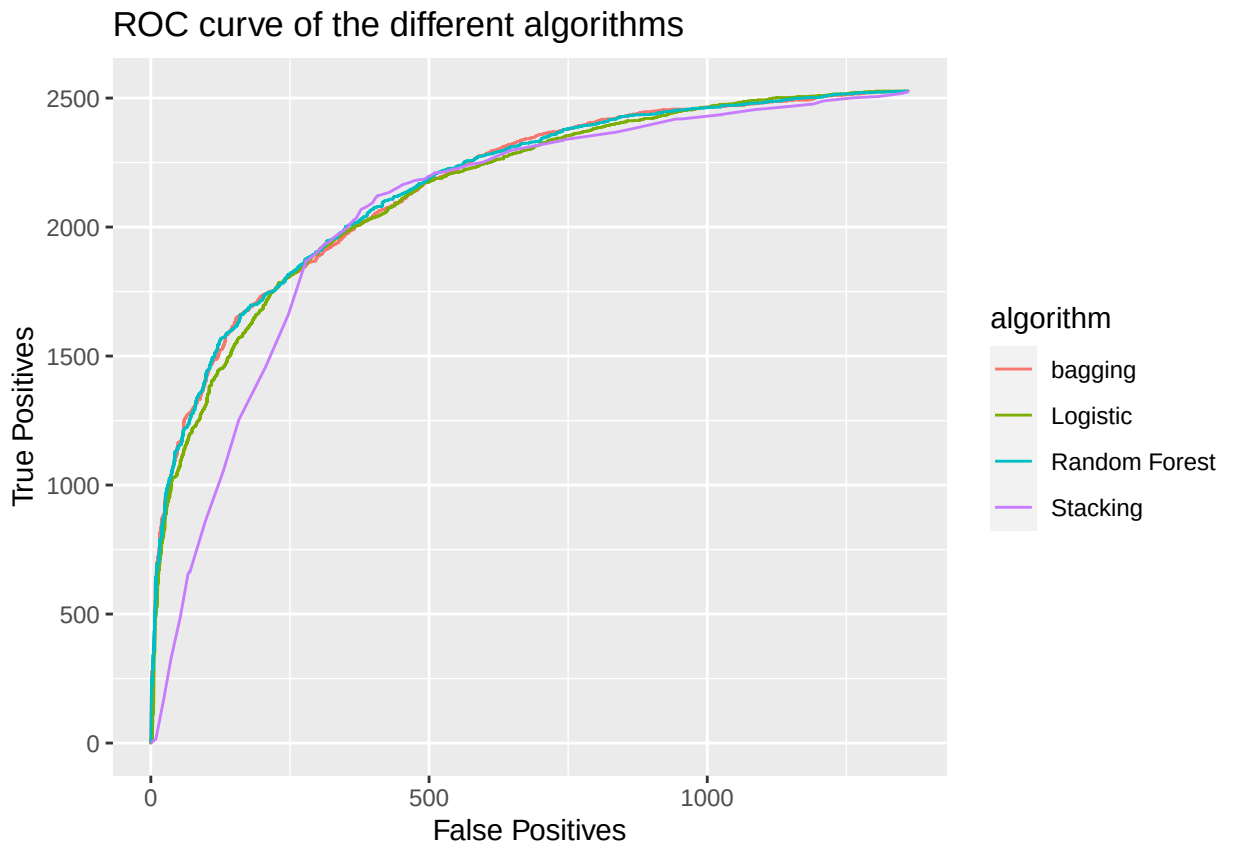
An ROC curve is made of the 4 best algorithms with the changed variables.

```
bagging_auc <- read.arff("Bagging_AUC.arff")
rf_auc <- read.arff("RF_AUC.arff")
logistic_auc <- read.arff("Logistic_AUC.arff")
stacking_auc <- read.arff("Stacking_AUC.arff")

subsetdata <- function(dataframe, value){
  subsetted <- subset(dataframe, select=c(`True Positives`, `False Positives`))
  subsetted$algorithm <- value
  return(subsetted)
```

```
}
all_auc <- rbind(subsetdata(bagging_auc, "bagging"),
                 subsetdata(rf_auc, "Random Forest"),
                 subsetdata(logistic_auc, "Logistic"),
                 subsetdata(stacking_auc, "Stacking"))

ggplot(all_auc, aes(x=`False Positives`, y = `True Positives`, group = algorithm)) +
  geom_line(aes(color=algorithm)) +
  ggtitle("ROC curve of the different algorithms")
```



This shows that all the algorithms have almost the same curve except for stacking, that one is slightly worse then the rest. Let's see if this corresponds with what the Weka experimenter says.

```
Dataset                   (1) function | (2) trees (3) meta. (4) meta.
-------------------------------------------------------------------------
cleaned_data-weka.filters(100)   77.76 |   78.43     78.15     78.48
-------------------------------------------------------------------------
                               (v/ /*) |   (0/1/0)   (0/1/0)   (0/1/0)


Key:
(1) functions.Logistic '-R 1.0E-8 -M -1 -num-decimal-places 4' 3932117032546553727
(2) trees.RandomForest '-P 100 -I 1000 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1 -depth 7'
(3) meta.Bagging '-P 100 -S 1 -num-slots 1 -I 100 -W trees.REPTree -- -M 2 -V 0.001 -N 3
(4) meta.Stacking '-X 10 -M \"trees.J48 -C 0.25 -M 2\" -S 1 -num-slots 1 -B \"rules.OneR
```

```
Dataset                   (1) functio |  (2) tree (3) meta (4) meta
-----------------------------------------------------------------
cleaned_data-weka.filters(100)   0.85 |    0.86 v   0.86     0.79 *
-----------------------------------------------------------------
                               (v/ /*) |  (1/0/0)  (0/1/0)  (0/0/1)


Key:
(1) functions.Logistic '-R 1.0E-8 -M -1 -num-decimal-places 4' 3932117032546553727
(2) trees.RandomForest '-P 100 -I 1000 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1 -depth 7' 1
(3) meta.Bagging '-P 100 -S 1 -num-slots 1 -I 100 -W trees.REPTree -- -M 2 -V 0.001 -N 3 -
(4) meta.Stacking '-X 10 -M \"trees.J48 -C 0.25 -M 2\" -S 1 -num-slots 1 -B \"rules.OneR -
```

This shows that percentage correct is the highest with stacking, but only slightly higher then a random tree. But when looking at the AUC stacking is significantly worse. So for that reason the random forest with a tree depth of 7 is the chosen algorithm to continue with. Also cause of time consumption, the tree is faster then the stacking but also when looking at the categories like percentage correct and AUC a better choice.