Forest-Based Binary Classification Study

Kevin Tittel (481044)

September 22, 2020

Step 1

We present our computational study with Random Forest (RF), MIRCO and Decision Tree (DT) on the German Credit Dataset. We have implemented all three algorithms in Python. The dataset corresponds to binary classification, where one is determined to be qualified for a credit or not.

We use two measures to compare the three methods against each other. Firstly, the average performance of the methods is reported, which boils down to the accuracy. Secondly, the average number of rules used by all three methods is reported, which corresponds to the extent of interpretability of the methods. We apply 10-fold nested cross validation in all experiments. Stratified sampling is used in order to preserve class fractions which may be present in the data set. Moreover, 10-fold nested cross validation is used to reduce variability of the results. Validation results are then averaged over the folds to give an estimate of the models' predictive performances. The following options are used for hyperparameter tuning: $maxdepth \in \{5, 10, 20\}$ for DT and RF, $numberoftrees \in \{10, 50, 100\}$ for RF.

The results of the numerical study are provided in Table 1. We observe that the number of rules provided by MIRCO is higher than the number of rules provided by DT (27.9% higher), while the accuracy increases by 1.39%. The resulting set of rules obtained by DT is more interpretable, since it produces a fewer set of rules. However, the results obtained by MIRCO are not too far off, considering that the accuracy is increased with MIRCO.

Table 1: Results steps 1 and 2

	Performance	Number of rules	Missed points	Runtime
Decision Tree	0.717	38	-	-
Random Forest	0.761	14065	-	-
MIRCO	0.738	63	0.081	42.41
MIRCO adapted	0.738	63	0.081	00.37

[a] Runtime given in minutes

Step 2

The greedy algorithm used to solve MIRCO is known for its accurate results, but on the other hand its many iterations over all rules induce a long completion time. One algorithm that combats this long computation time is the List-and-Remove (LAR) algorithm (Yang et al., 2015). The LAR algorithm maintains a list that indicates how many uncovered samples are covered per rule that is not selected yet. Once a rule is selected, this list updates the amount of uncovered samples for the remaining unselected rules. Straightforwardly, LAR iteratively selects the rule that covers the highest amount of uncovered samples. Since we efficiently use and update this list, a large amount of iterations over each and every rule to select the best one in terms of costs is to be avoided. This makes the running time significantly lower as can be seen when looking at this adapted MIRCO in Table 1. Interestingly, it can also be concluded that the results equal those of the original MIRCO, such that apparently the increase in efficiency is not traded for a decrease in accuracy.

Step 3

The set of rules obtained by the MIRCO algorithm currently inhibit the problem that the rules do not necessarily cover all the feature space. We have proposed Algorithm 1 that makes the MIRCO algorithm a classification algorithm. The objective should still be to minimize the number of rules and total amount of costs. However, the new constraint becomes that the total feature space should be covered by the rules such that the classification of any test sample can be ensured. Hence, we need to add back rules of the Random Forest, that were previously rejected, up until the point that the feature space is completely covered again, after which we can possibly subtract any redundant rules.

Firstly, we introduce some of the notation used. Recall that the sets $\mathcal{I}$ and $\mathcal{J}$ represent the set of samples and the set of rules, respectively. In addition, let $\mathcal{X}(j)$ denote the feature subspace that is covered by rule $j \in \mathcal{J}$. Suppose that the MIRCO Algorithm introduced by Birbil et al. (2020) results in a set of rules $\hat{\mathcal{J}}$. Hence, if we denote the complete, universal feature space by $\mathcal{U}$, then $\mathcal{O} = \mathcal{U} \setminus \{\cup_{j \in \hat{\mathcal{J}}} \mathcal{X}(j)\}$ is the feature space that is not yet covered by any of the rules in $\hat{\mathcal{J}}$. Our objective is to cover $\mathcal{O}$ by adding rules out of $\mathcal{J} \setminus \hat{\mathcal{J}}$, and eventually subtracting redundant rules as we aim to minimize the total number of rules and impurity cost. In particular, subtracting redundant rules is done as follows: once we have obtained our set $\mathcal{J}^{Upd}$ which covers all of $\mathcal{U}$, its rules are sorted in ascending order of their costs in a list. Then, starting off with the rules at the end of this list and going back to the rule at the start of the list with minimal cost, a rule $j \in \mathcal{J}^{Upd}$ is removed if and only if $\mathcal{U} \setminus \mathcal{X}(j) = \mathcal{U}$, that is, removing the rule does not limit the universal feature space $\mathcal{U}$. Ultimately, the algorithm proposed in Algorithm 1 turns MIRCO into a classifier.

Algorithm 1: MIRCO Classifier

```
Result: Classifier rules $\mathcal{J}^{Upd}$
$\mathcal{J}^{Upd} = \hat{\mathcal{J}}$;
while $\mathcal{O} \neq \emptyset$ do
    $j^* = \arg\min_{j \in \mathcal{J} \setminus \mathcal{J}^{Upd}} \left\{ \frac{1+w_j}{|\mathcal{I}(j)|} : \mathcal{O} \cap \mathcal{X}(j) \neq \emptyset \right\}$;
    $\mathcal{J}^{Upd} \leftarrow \mathcal{J}^{Upd} \cup j^*$;
    $\mathcal{O} \leftarrow \mathcal{O} \setminus \mathcal{X}(j^*)$;
end
Remove redundant columns from $\mathcal{J}^{Upd}$;
Return $\mathcal{J}^{Upd}$;
```

Step 4

The output of MIRCO concerns a set of rules which may not necessarily be concise. The function in step4.py simplifies the resulting set of rules obtained with MIRCO by compacting the clauses forming a rule in two different ways. Firstly the notation within a set is simplified by compacting rules concerning the same variables.

Furthermore, another simplification concerns denoting the complete set of rules more compactly. That is, starting off with a set of $|\hat{\mathcal{J}}|$ rules obtained by the MIRCO algorithm, we would like to deliver a more compact set of size $\lceil |\hat{\mathcal{J}}|/2 \rceil$, which contains pairs of rules that share the most feature space and, hence, have in common as many clauses as possible. Trivially, each pair of rules has disjoint clauses, which will also be highlighted. This could be extended by making this an iterative process, in which the pairs of the previous iteration are subsequently combined with the pair that has in common the most up until the point that all the rules are combined in the total set, $\hat{\mathcal{J}}$. Hence, a tree can be graphed of pairs of sets of rules that have the most feature space in common, which gives a compact overview of the structure of the rules we have obtained. We ourselves have implemented the first iteration of this idea.

Appendix

We have written the following codes to implement our methods:

- **step1.py**: This code implements Random Forest (RF), MIRCO and Decision Tree (DT) algorithms on the given credit dataset, and reports (i) the average performance of all three methods, (ii) the average number of missed points by MIRCO, and (iii) the average number of rules used by all three methods in Python.
- **step2.py**: This code implements our proposed improvement on Chvatal's greedy set covering heuristic used in MIRCO in Python. The definition of the function 'greedySCP' is altered.
- **step4.py**: This code implements our proposed simplifications in reporting the outcome of the MIRCO algorithm in Python. The functions 'exportRulesSimplified1' and 'exportRulesSimplified2' are added as simplification. 'exportRulesSimplified1' is the implementation of the simplification given in the assignment. 'exportRulesSimplified2' is the implementation of the simplification provided above.

References

Birbil, S.I., Edali, M., Yüceoğlu B. (2020) Rule Covering for Interpretation and Boosting. Under review.

Yang, Q., Nofsinger, A., McPeek, J., Phinney, J., Knuesel, R. (2015). A complete solution to the set covering problem. In Proceedings of the International Conference on Scientific Computing (CSC) (p. 36). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).