UNIVERSITY OF MANNHEIM

Data and Web Science Group
Prof. Dr. Christian Bizer
B6 – B1.15
68159 Mannheim

Data Mining – FSS 2020

Exercise 4: Classification

General notice: For X-Validation use a local seed with value "1992" to get the same results!

4.1. Rule Learning

Download the *glass.arff* data set, which is a default data mining data set, distributed e.g. by Weka, from the course website. The dataset was created during a study which was motivated by criminological investigations. At the scene of the crime, the glass left can be used as evidence, if the purpose of the glass can be classified correctly. You can read more about the different attributes within the .arff-File.

1. Import the file and store it into your repository, using the *Read ARFF* and *Store* operator.

   Solution: Use the Read ARFF and the Store operator. Select the location to store and run the process. After the process finished you will have the data in your RapidMiner Repository.

2. Now use the Rule Induction classifier to learn classification rules based on the glass dataset. Use in a first step the default setup (pureness = 0.9) and have a look at the rules.

   Solution: Retrieve the stored dataset from your repository. Set the Type attribute as label using the Set Role operator.

   Conclusion: 199 out of 213 examples will be covered correctly with the presented 19 rules. 6 of the rules are really simple (just one attribute limitation).

3. Change the pureness attribute to 1 and 0.5. How is the change reflected in the set of rules created?

   Solution: Rerun the process from 3 once with 1 and once with 0.5 as pureness.

   Conclusion:

   - Pureness 1: 179/206 correct examples with 12 rules
   - Pureness 0.5: 154/212 correct examples with 9 simple rules
   - Pureness 0.85: 198/213 correct examples with 19 rules

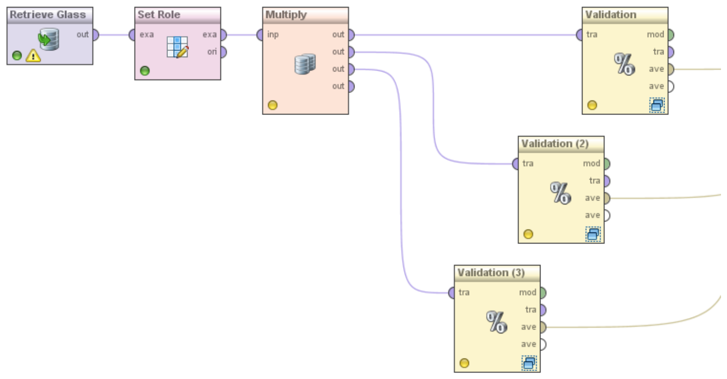
4. Replace the Rule Induction Operator by the *Tree to Rules* (nested Operator) and use a *Decision Tree* inside. Compare the results from direct and indirect Rule-Based-Classification Algorithms, based on number of rules created and complexity of the rules.

Solution: Replace the Rule Induction Operator by a Tree To Rules Operator. Include within the Tree to Rules operator a Decision Tree Operator. The Tree to Rules Operator does not offer parameters. All tunes has to be done within the Decision Tree Operator itself.

Conclusion: Running the process with default setup leads to 168 out of 214 correct examples with 16 complex rules (in comparison to Rule Induction).

5. Use the X-Validation (Classification) with a 10-fold setup and compare the accuracy of both rule learning approaches and a k-NN classification algorithm. What does work best on the data set?

   Solution: Set-up a process similar to the one from Exercise 3 using X-Validation. You can also set up 3 X-Validators in parallel using the Multiply Operator.



   Conclusion: Using the setup from 3 and 4 and K-NN with k = 1 results in K-NN with 71.02% accuracy (more than both rule based learners).

   - Tree to rules: 64.52%
   - Rule induction: 70.22%

4.2. Who should get a bank credit?

The German credit data set from the UCI data set library (http://archive.ics.uci.edu/ ml/index.html) describes the customers of a bank in respect whether they should get a bank credit or not. The data set is provided as *credit-g.arff* file in ILIAS. You need to use the RapidMiner *ARFF reader* operator to import the data set. Please also have a look at the data set documentation that is included in the file.

1. Apply the Compare ROCs Operator to the dataset and include k-NN, Decision Tree and Rules Based classification. Which classification approach looks most promising to you?

   Solution: Include the Compare ROCs operator and add the three classifiers within the operator. Based on the theory, the curves which go straight up in the beginning represent the best performance. The more it moves to a straight diagonal the worse it gets.

   Conclusion: Based on the outcome the Decision Tree performance most promising. Also the Rule Induction and K-NN with k = 1 should be considered.

2. Include the most promising classification approaches and use a 10-fold X-Validation approach. Which level of accuracy do you reach?

Solution: Remove the Compare ROCs operator and place a x-validation operator in the process. Apply the learned model and measure the performance (have a look at the classification workflow slide set in case you have troubles with this).

- Decision Tree: 69.40% Accuracy – 201 bad customers predicted good
- Rule Induction: 71.50% Accuracy – 173 bad customers predicted good
- k-NN (k=1): 61.60% Accuracy – 198 bad customers predicted good
- k-NN (k=3): 62.30% Accuracy – 230 bad customers predicted good
- k-NN (k=5): 65.60% Accuracy – 234 bad customers predicted good

3. What does the precision and recall values for the positive class “Bad Customer” tell you? Try to improve the situation by increasing the number of “bad customers” in the training set. For doing this, you first filter all bad customers from the data set and then append these customers to the original set. How does precision and recall change if you apply this procedure twice? Use the *Filter Examples* and *Append* Operators.

   Solution: There is an unbalance in the results. The bad class is predicted less often right, than the good class. To solve this add the Filter Examples with the attribute_value_filter set to “class=bad” (or custom_filters) and then use the Append Operator to increase the number of good examples in the training set. Make sure you only do this within the training set.

   Conclusion: Applying this procedure once with Rule Induction we end up with an accuracy of 69.80% and 111 bad customers classified as good. When adding the bad class two more times we end up with 68.50% accuracy but only 89 bad customers classified as good.

4. To model a use case specific evaluation, as observed in the previous example, replace the *Performance (Classification)* operator by the *Performance (Costs)* operator. Set up your cost matrix by assuming that you will lose 1 Unit if you refuse a credit to a good customer, but that you lose 100 Units if you give a bad customer a credit. Rerun the experiments from 1 and evaluate the results.

   Solution: Replace the Performance Classification operator with the Performance (Costs) operator and setup the matrix with ((0,100)(1,0)) than rerun the process and inspect the outcome.

   Conclusion: The usage of this performance measuring operator helps to directly see the cost of a misclassification which in this case is really important, as not giving a credit to somebody who can pay back is a small failure in comparison to giving a credit to somebody who will never pay something back. In the first case we lose the interests, in the second case we lose all the money.

5. As the creation of training data is mostly a manual task as humans tend to be fallible training data might include noise. Simulate this behavior by using the *Add Noise* operator and change the parameter “label noise” from 0% to 10% to 20%. Is your preferred classification approach still feasible for this situation? How does the performance of the other classifiers evolve?
   Solution: Place the Add Noise operator and change the parameter from 0.0 to 0.1 and 0.2 and observe the results.

Conclusion:

Model	Accuracy - Noise 0%	Accuracy - Noise 10%	Accuracy - Noise 20%
KNN n=1	61.80%	60.00%	58.70%
Rule Induction	68.30%	65.80%	59.40%
Decision Tree	69.80%	31.20%	30.60%