MTAT.03.227 Machine Learning
Spring 2016 / Exercise session II
Nominal score: 10p
Maximum score: 15p
Deadline: 2rd of March 16:15 EET

1. This exercise illustrates shortcomings of rule detection. For that you have to find rules that determine the species of iris plants based on the shape of its flower. The input data is stored in the file `iris-discretised.Rdata` and can be loaded using `load` command. Rules can be found with `arules` package. The file `analysis-of-rules.R` contains some examples how rules can be learnt from the data and how to manipulate rules.

   (a) The dataset contains three species: *iris-setosa*, *iris-versicolor* and *iris-virginica*. Modify the code so that you get a list of rules for each species: `rules.setosa`, `rules.versicolor` and `rules.virginica`. Sort each list according to their support in decending order. (**1p**)

   (b) Next choose one of these lists and compute overall recall and precision for the first i rules. For that you must find the number of true positives, false positives and false negatives. To simplify the analysis, assume that a row that is matched by any of the first i rules is predicted as positive. Tabulate recall and precision for all possible values of i. Draw the corresponding scatter plots for all three lists and interpret results. What is the maximal achievable recall if we can reliably detect only rules with support more than 5%. (**1p**)

2. Load the file `iris-discretised.Rdata` as an input and modify the file `id3-algorithm.R` in order to implement the ID3 algorithm.

   (a) Write a function `ComputeEntropy` that tabulates occurrences of the target variable, computes frequencies and outputs the entropy of the target variable. **Hint:** You can use `table` function. (**1p**)

   (b) Write a function `ComputeInformationGain` that splits the target vector into sub-vectors according to some attribute vector and computes the corresponding information gain. As all attributes have discretised levels which can be ordered, you should split the attribute vector into two parts according some threshold. As there are five potential thresholds, there are five different information gains and you should choose the best. Write a function `ComputeInformationGain` whichs return both the best threshold and the corresponding gain. (**1p**)

   (c) Write a recursive function `RecursiveID3` which uses the function `ComputeInformationGain` to find the best attribute for splitting and then splits the data according the best threshold into two data frames and then recursively applies `RecursiveID3` on both parts. The function should output the best prediction to the target value if the

dataset cannot split further. The function should output the textual description of the corresponding decision tree. You should use and improve functions `TreeNodeLine` and `PrintLeafNodeInfo`. (**1p**)

(d) Modify the function `RecursiveID3` so that it would give out predictions both on the training and on the test data. For that you do not have to create an explicit description of the decision tree. It is sufficient to split training and test data simultaneously based on the split criterion found on the training data. Split the original data randomly int two parts such that training set consists of 100 and the test set of 50 samples. Report prediction accuracy on the training and test data in each leaf node and on average over both datasets separately. Interpret results. (**1p**)

3. Use the Iris Data Set from UCI Machine Learning repository available as `iris-original.csv` to implement ID3 algorithm for continuous values.

   (a) Write a function `ComputeInformationGain` that splits the target vector into two sub-vectors according to a real-valued attribute vector and computes the corresponding information gain. You should split the attribute vector into two parts according some real-valued threshold. As there are many potential thresholds and you should choose the best. Write a function `ComputeInformationGain` whichs return both the best threshold and the corresponding gain. (**1p**)

   (b) Implement a naive statistical test for determining relevance of a split. Do at least 100 time the following procedure (simulation). Randomly reorder the target variable (use `sample`). Compute the resulting information gains for the split. Do the split only if the original information gain is bigger than at least 95 information gains obtained in the simulations. (**1p**)

   (c) Write a corresponding function `RecursiveID3` so that it would give out predictions both on the training and on the test data. Split the original data randomly int two parts such that training set consists of 100 and the test set of 50 samples. Report prediction accuracy on the training and test data in each leaf node and on average over both datasets separately. Interpret results. (**1p**)

4. Load the file `iris-discretised.Rdata` as an input and modify the file `rule-prioritisation.R` in order to implement rule prioritisation algorithm. To simplify the task, you can use `rpart` package to generate the decision tree. The file `rule-prioritisation.R` implements some convenience methods for rule generation and rule matching.

   (a) Split the data randomly into 100 element training set and 50 element test set. Train a decision tree on the training set without any pruning restrictions. Extract all rules corresponding to the leaf nodes and compute confidence levels for each rule (**1p**).

(b) For each rule perform iterative weakening. Find a term in the left-hand side which can be dropped with minimal consequences on the test set. Let c_0 be the confidence level of the initial rule and c_i the confidence level of the tule where the ith condition is dropped. Then you should find the index i which minimises $\Delta c = c_0 - c_i$.

If Δc is smaller than 10% from the initial confidence c_0, you should drop the ith condition and continue on with the weakening. Otherwise, you should stop. Note that c_0 should be always the confidence of the initial rule to avoid gradual drift.

List all unique weakened rules with corresponding confidence levels on the test and data. (**1p**)

(c) Build a predicting algorithm that uses weakened rules to predict the labels. If several rules match the input, you should output the prediction of the rule with the hughes confidence. Find out the accuracy of the original decision tree and your new algorithm on the training and test data. For the original model, you can use `predict(model, data, type="class")` to get labels. Interpret the results. (**1p**)

(d) Repeat the same experiment with many random splits and draw corresponding boxplots for the accuracy on the test and training data. Interpret results. Why is the comparison scheme unfair for the default algorithm? What should we do to get fair comparison? (**1p**)

5. Study the effect of pruning and complexity penalty on the prediction error. Use Breast Cancer Wisconsin dataset for benchmarking. Split the data randomly so that training set consists of 66% and test set of 34% samples. Use standard implementations like `rpart` or `tree` and study how corresponding `rpart.control` and `prune` functions work. On strategy should be conservative and seriously limit the complexity of a tree, while the other should allow almost maximal complexity. Report and interpret the results. (**3p**)

6. Study the effect of classification errors on the prediction error. Use Iris or Breast Cancer Wisconsin dataset in experiments or some other easily classifiable dataset like 3D checker board pattern. Split the data randomly so that training set consists of 66% and test set of 34% samples. On the training data randomly flip 1, 5, 10, 25, 50, 100% labels of the target value. Foe two class case, the flipping reversal of the label. If you consider more than two class labels then describe the flipping strategy, as well. Use two decision tree learning algorithms one with limited complexity (e.g. severely limited tree depth) and other without limitations. Do several experiments and present the average behaviour together with some variance estimate (e.g. `boxplot`). Visualise and interpret results. (**3p**)

7. The main drawback of a decision tree are rectangular decision borders. For continuous variables, such borders do not make much sense. Hence, one could be more intelligent and use more complex borders. Use linear

SVM-s (`e1071` package) as a black-box method for splitting the data into halves according to class labels. After that train a new SVM on each part separately until all elements are correctly classified or some other stopping criterion is met. The resulting decision tree should work much better on datasets with complex borders. Build such a decision tree and test it on 2D examples: 45° tilted checker board and ball patterns. Compare your algorithm with ID3 algorithm. Report and interpret results. (**5p**)