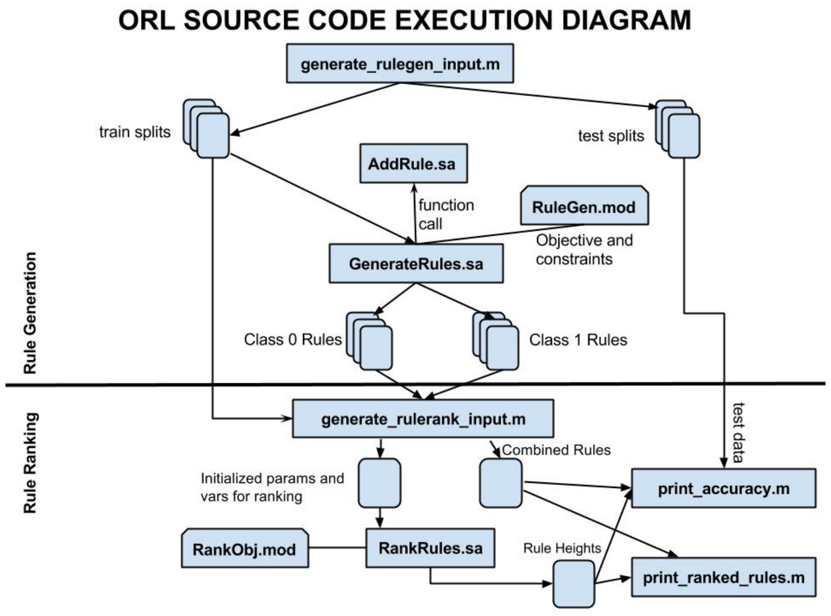
README for ORL Package

This package contains the data and code for running ORL experiments, associated with the paper Learning Customized and Optimized Lists of Rules with Mathematical Programming by Cynthia Rudin and Seyda Ertekin.

In the github repository `https://github.com/SeydaErtekin/ORL`, the code for the first phase of ORL (Rule Generation) is under the Rule_Generation directory, and code for the second phase (Ranking of the discovered rules) is under the Rule_Ranking directory. We provide two of the datasets that we used in our experiments, namely Haberman's Survival and TicTacToe, under the Datasets directory.

In the package, we provide two shell scripts for running experiments with Haberman and TicTacToe datasets. The first script, run_haberman.sh, uses Haberman's sample train/test split under Datasets/processed/ and invokes the sequence of codes for generating and ranking rules, followed by displaying the ranked rules. With the default settings, the script generates the ranked rules shown in Table 5 in the paper. For TicTacToe, we use the toy ruleset under Rule_Generation/rules, so run_tictactoe.sh only runs the rule ranking and displaying routines. This ruleset and corresponding results form the basis of our discussion in Section 2.1. Note that both scripts require Matlab and AMPL with Gurobi solver to be installed on the local machine.

An overview of the order of execution and the dependencies of the code is given in the diagram below.



In this package, we also provide a sample train/test split for both datasets, as well as the rules (under Rule_Generation/rules directory), the data input for rule ranking and the ranked rules (under Rule_Ranking/rules directory). The script print_ranked_rules.m can be used to view the ordered rule lists for these splits. For the Haberman's Survival dataset, the set of rules include all rules discovered with a particular setting of the input parameters. For the tictactoe dataset, we provide the toy ruleset (that we discuss in Section 2.1 in the paper) that is a trimmed version of all discovered rules. This toy ruleset includes eight rules for the 1 class, three rules for the 0 class, and two default rules (one of each class). The input data for tictactoe used for ranking (under Rule_Ranking/rules/tictactoe_binary_train12_rank_input.dat) only initializes the necessary parameters required for ranking; it does not need to precompute the values of the variables because the number of rules is already pretty small and the optimization completes within a few seconds.

Directory Structure

Datasets: .csv files of the original datasets. If you'd like to generate brand new train/test splits for the datasets, you can use the script generate_rulegen_input.m to generate up to 3 train/test splits by chunking the dataset into 3 equal sized chunks. Files for each split is suffixed with 12, 13, or 23, indicating which chunks were used for training. For example, the files with suffix 12 indicate that first and second chunks are in the train set and chunk 3 is in test set.

Note that due to the random shuffling of the examples, any newly generated train/test splits will be different than what we provided, hence may yield different results. If you'd like to use the existing splits that we reported results for in the paper, you can use the files under Datasets/processed.

Datasets/processed: Directory that contains train/test sets (files with .txt extension) and the train sets in ampl data format (with .dat extension). The former files are used for performance evaluation whereas the latter files are used in rule generation.

Rule_Generation: Contains generate_rulegen_input.m script for generating files under Datasets/processed, and the ampl code(s) that implement rule generation routines. GenerateRules.sa is the main implementation of the rule generation routine and AddRule.sa is a helper script (called from GenerateRules.sa) that's responsible for writing discovered rules to the output file as well as adding the rule to the list of constraints so we don't discover the same rule again in subsequent iterations. The objective and constraints for rule generation are specified in a model file called RuleGen.mod.

Rule_Generation/rules: Contains the files for the discovered rules for both classes in the datasets. We provide representative rules for both datasets in this directory. Files with "one" and "zero" suffixes include rules for one and zero classes, respectively. The file with "all" suffix is the aggregate of both files and default rules for both classes.

Rule_Ranking: Contains matlab script generate_rulerank_input.m for aggregating the rules for both classes under Rule_Generation/rules. The aggregate rules are written to Rule_Generation/rules (with "all" suffix and .txt extension) and an ampl formatted version is written under the rules subdirectory. The Rule_Ranking directory also includes the ampl code RankRules.sa that implements the rule ranking routine and the model file RankObj.mod.

Rule_Ranking/rules: Contains the data input used for rule ranking as well as the ranking output (the π vector of rule heights). This directory contains the ranked rules for both dataset at obtained for different C and C_1 settings. Running print_ranked_rules.m (up in the Rule_Ranking directory) prints the ranked rules for the specified dataset/experiment in human-readable form. print_accuracy.m similarly computes the accuracy on train or test set (controlled within the code) for the specified dataset/experiment.