* NAME

KARL: Program Knowledge Augmented Biomarker Discovery, variable selection, discretization, and data preprocessing.

* SYNOPSIS

java [JAVA\_PARAMETERS] -jar TRLplus.jar

-lp [LEARNING\_PARAMETERS]

-dp [DATA\_PARAMETERS]

* DESCRIPTION

KARL, is a toolkit for learning Knowledge Augmented Rule Learning (KARL). The algorithms have several learning parameters with sensible default values. The program can also transform input data in various ways before learning, or even without any learning.

KARL's inputs are a lookup table contains properties of domain, in addition to, a set of training data instances specified in a data file (see DATA FILE FORMAT), where each instance is a vector of values for the input variables, and a class value. The variables can be continuous or categorical.

With the input it learns classification rule model, which comprises an unordered list of rules of the form:

IF <antecedent> THEN <consequent>

where the antecedent consists of a logical conjunction of one or more variable-value pairs (conditions), and the consequent is a prediction of the class variable. For example, a learned rule

might be:

IF ((Age=High) AND (BloodPressure=Low)) THEN Class=Control

which means “if the variable *Age* is in the *High* range, and the variable *BloodPressure* is in the *Low* range, then predict that the data instance has the class value *Control*.” Values such as *Low* and *High* represent intervals of real numbers that result from discretizing the variables before learning. A rule is said to cover or match a data instance if each variable value of the instance is in the range specified in the rule antecedent. The classifier also includes an evidence gathering method for breaking ties when several rules match a query data instance but predict different classes.

The classic RL algorithm proceeds as a heuristic beam search through the space of rules from general to specific. Starting with all rules containing no variable-value pairs, it iteratively specializes the rules by adding conjuncts to the antecedent. It evaluates the rules, calculating a certainty factor value and other statistics for each rule. It re-inserts promising rules onto the beam, while removing other rules. The beam is sorted by decreasing certainty factor value and is trimmed to a pre-defined length during each iteration. Beam search is used to limit the running time and space of the algorithm. Multiple learned rules in an RL classifier may cover the same training instance. This is unlike most other classification rule and tree learning algorithms, which cover data without replacement, so that each data instance is covered by only one rule. With small sample size data sets, covering with replacement allows RL, and its extensions, to utilize more of the available evidence for each rule when computing the generalizability of the rule. The extensions to RL , KARL, also allows transfer learning, where some “prior” rules are learned on “domain-knowledge bases”, and are then placed on the beam for learning on the “target” data set.

* DATA FILE FORMAT

Each data file comprises a table of rows (lines) and columns representing vectors of variable values. Example data file:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #ID | Age | Sex | Temperature | @Diagnosis |
| A42 | 22 | F | 37 | Healthy |
| D25 | 35 | M | 39 | Sick |
| ... | ... | ... | ... | ... |

In normally-oriented data files, each row represents a data instance vector and each column represents a variable of values for the instances; the first row is a header line specifying the names of the variables. However, the input file can be transposed, so that rows represent variables and columns represent data instances; this must be specified using the “-tpf” option.

Columns can be separated by tabs or by commas (CSV). The separator must be the same throughout the file, and is assumed to be Tab unless “,” occurs more frequently in the header line. The user can explicitly specify the separator by using the data parameters.

The data must contain exactly one class variable (output variable), and a number of input variables. The class variable is indicated by a “@” as the first character of the variable name. The class value of the first data instance that appears in the file is used in the predictive performance statistics in the output.

A data file may contain one ID input variable, indicated by a “#” as the first character of the variable name. KARL ignores the ID variable during learning but uses it to identify data instances in the output. If no ID variable is specified, the program uses data instance IDs “1”, “2” and so on, namely the index of the data instance in the data file.

Each input variable can be continuous or categorical. A continuous variable is one whose values can be parsed as a numbers, such as “100” or “1.25” (without the quotes). Categorical variables have values such as “F” and “M”. Before learning, KARL discretizes the continuous-valued variables using the specified discretizer. If you want the program to treat some numeric variable as categorical instead of continuous, and thus avoid discretizing it, you can add a single quote before each value. For example, instead of “100” (without the quotes), use “100”.

* JAVA PARAMETERS

Because KARL is a java program, it must be run on a Java virtual machine. Java can take a number of parameters, which are described in the Java manual. In addition, it uses some of the Weka program libraries, so Weka must be installed on the system and the Java *classpath* must include the Weka classes (jar file). The *classpath* can be set in the CLASSPATH environment variable, or using the “-classpath” Java parameter. To provide enough memory on the Java virtual machine, use the “-Xmx” Java parameter. For example, “java -Xmx1000m”.

* LEARNING PARAMETERS

Learning parameters are command line attributes that bias the learning process of the algorithms, and are initiated by the flag “-lp”. They can be specified in any order after the Java parameters and must precede the data parameters flag, “-dp”, on the command line. The learning parameters, including their meanings, are presented below.

-cftype INDEX\_VAL

Function to compute the certainty factor value for each rule.

|  |  |
| --- | --- |
| 0 | Positive predictive value (default):  TP / (TP + FP) |
| 1 | Positive predictive value with Yates correction:  (TP + 0.05) / (TP + FP) if TP > FP,  (TP - 0.05) / (TP + FP) if TP < FP,  TP / (TP + FP) otherwise. |
| 2 | Positive predictive value, normalized for asymmetric class distributions:  1 if FP = 0  0 if TP + FP = 0  TP / (TP + FP \* Pos / Neg) |
| 3 | Laplace estimate:  (TP + 1) / (TP + FP + num\_of\_classes) |
| 4 | Laplace extended:  (TP + k\*m) / (TP + FP + k),  where  k = number of target values  m = Pos / (Pos + Neg) |
| 5 | Laplace extended with bias for short rules:  (TP + c \* q) / (TP + FP + k),  where  c = 1 + number of conjuncts in the rule  q = TP / (TP + TN) |
| 6 | F-Measure:  2 \* precision \* recall / (precision +recall)  where  precision = TP / (TP + FP)  recall = TP / (TP + FN) |
| 7 | Laplace estimate, normalized for asymmetric class distributions:  (TP + 1) / (TP + FP \* Pos / Neg + num\_of\_classes) |
| 8 | P-Value, log likelihood ratio test:  2 \* (TP \* log2(TPfreq/PosFreq) + n  FP \* log2(FPfreq/NegFreq))  Where  TPfreq = TP / (TP + FP)  FPfreq = FP / (TP + FP)  PosFreq = Pos / (Pos + Neg)  NegFreq = Neg / (Pos + Neg) |

-inftype INDEX\_VAL

The “inference type” or "evidence-gathering" function that is to be used during inference to make a prediction from a given data instance from the learned set of rules. It combines the predictions of all rules that match the given instance.

|  |  |
| --- | --- |
| 0 | Weighted voting (default). Predict the highest-weighted class, where the weight of each class is the sum of certainty factors of rules predicting that class. If there is a tie, predicts class 0. |
| 1 | Maximum likelihood ratio |
| 2 | "Combine CF" |
| 3 | Lowest p-value: use the rule with the lowest p-value |
| 4 | Single best: use the rule with the highest certainty factor |
| 5 | Minimum weighted voting. Like weighted voting, but use only  the highest k rules to calculate the weight of each class,  where k is the minimum number of rules voting for any  class. |
| 6 | Single best specific: use the rule with the highest worth  (certainty divided by cost) and the highest number of  conjuncts |
| 7 | Most specific single best: use the rule with the most  conjuncts among rules with the highest certainty factor. |
| 8 | Highest Coverage: use the rule with the highest coverage. |

-mincf NUMBER

The minimum certainty factor value that any rule in the model will have. The default is 0.80.

-minconj NUMBER

The minimum number of conjuncts in any rule in the model. The default is 1.

-maxconj NUMBER

The maximum number of conjuncts in any rule in the model. The default is 5.

-specialize

If this option is specified, when a rule is added to the model, RL will also check if some specializations of this rule should also be added to the model. If the option is omitted (default), RL stops specialization of a rule once it is found to satisfy the search constraints.

-cover NUMBER

The minimum number of training examples that any rule in the model will cover. The default is 4.

-minTP DECIMAL

The minimum true positive rate that any rule in the model will have. The default is 0.05. Valid values are in the range [0, 1].

-maxFP DECIMAL

The maximum false positive rate that any rule in the model will have. This option is not set by default. Valid values are in the range [0, 1].

-indStr NUMBER

Inductive strengthening: the minimum number of previously uncovered examples that each new rule must cover. The default is 1. The smaller this number, the larger the overlap of instances covered by different rules. Because RL covers data with replacement, using some non-zero inductive strengthening helps to learn a more generalizable model.

-beam WIDTH

The number of rules kept at any time to be specialized in the next iteration. The default is 2500.

-cv NUM\_FOLDS

Stratified cross-validation. If the option is not specified, no cross-validation is performed. In any case, a classifier is learned on all the training (target) data.

-d Discretize.

The parameters are as described in PREPROCESSING PARAMETERS, but the discrete intervals for each variable are computed based on the training data only, then applied to the test data. If cross-validation is specified, the discretization is computed on the training subset separately for each fold.

* TRANSFER LEARNING PARAMTERS

For the transfer learning algorithms to be invoked the “-tr” flag must first be specified, else the program runs RL by default. Below are the specific transfer learning parameters, including descriptions of their usage.

-tr INDEX\_VAL

INDEX\_VAL should be set to 2 for running the KARL method.

-lookuptable FILENAME

This flag invokes the KARL algorithm, where prior rules are generated from abstracted evidence of domain-knowledge contained in the FILENAME. The FILENAME is a csv file, which contains domain variables and their evidence (from literature) of association with pertinent domain processes. Below is a snippet of a lookup table file which contains abstracted information from the domain, cancer.

|  |  |  |  |
| --- | --- | --- | --- |
| GeneID | Findings | #Evidence | Function |
| COL1A1 | -1 | 8 | 1 |
| IGFBP2 | 1 | 9 | 1 |
| VEGFA | 1 | 31 | 2 |
| CAV1 | -1 | 24 | 3 |
| ... | ... | ... | ... |

In the example above, “GeneID” referes to the name of a domain variable (gene); “Findings” indicates the general consensus and evidence from literature on whether the variable decreases (-1), increases(+1), or affects (0) specific domain processes; “Function” denotes specific domain processes like cell death(1), cell invasion (2), or cell proliferation (3); and “#Evidence” indicates the number of literature references that attest to the functional evidence of the variable. Note that each lookup table file is specific to the training or target data set and must be processed outside the program.

* PREPROCESSING PARAMETERS

These are optional parameters that specify operations to be performed on all the data before any rule learning. They can appear in any order after the "-lp" flag and before the "-dp" flag. Only operations specified will be performed.

-d DISC\_MECHOD DISC\_VALUE

Discretize using DISC\_METHOD with specified parameter

|  |  |  |
| --- | --- | --- |
| PARAMETER | DISC\_METHOD | PARAMETER2 |
| 0 | GaussianU | Number of bins |
| 1 | EqualWidthU | Number of bins |
| 2 | EqualFreqU | Number of bins |
| 3 | OneR | Number of instances |
| 4 | ErrorBased | Max number of bins |
| 5 | D2S | (none - max number of bins is set to 8) |
| 6 | FayyadIraniMDL | Number of bins |
| 7 | HEBD | c structure prior (use value "1") |
| 8 | MODL | none? |
| 9 | EBD lambda | Prior |

**Example:** EBD (2011) discretization with default parameter:

-d 9 0.5

-r Remove trivial variables after discretization

-chi NUM\_OF\_VARS\_TO\_SELECT

Chi-squared variables selection: select the top NUM\_VARS\_TO\_SELECT variables.

-s SCALING\_METHOD ...

Scale each variable by the specified SCALING\_METHOD in turn

|  |  |
| --- | --- |
| 0 | 0-1 scaling |
| 1 | Subtract local minimum |
| 2 | Subtract global minimum |
| 3 | Log2 |
| 4 | Square root |
| 5 | Exponent 2 |
| 6 | Square |
| 7 | Normalize to mean 0 and standard deviation 1 |

-ctr

Combine technical replicates. The samples must have the same name, with ‘#’ next to it

DATA PARAMETERS

Data parameters specify the input and output files and their format. The training data file is a mandatory data parameter and must be the last parameter. Data parameters must be preceded by the “-dp” flag. The “-dp” flag and the data parameters must appear after the “-lp” flag and any learning parameters, and after the.

-itrncsv

Training data file is comma-separated

-itstcsv

Test data file is comma-separated

-c CSV\_DATA\_FILE

Convert the csv-delimited file to tab-delimited or vice versa.

-tpf DATA\_FILE

Transpose the file data file; that is, make the rows be columns (variables) and columns be rows (instances).

-dtr TRAINING\_DIRECTORY

Directory containing the training data files, one file for each training data instance. Each file contains two columns: variable and value. Within the directory files grouped by class folder; e.g. inside the training directory, there are two folders: "disease" and "control". There should be no trailing "/" in TRAINING\_DIRECTORY name.

-tst TEST\_FILE

Specify a test data file

-dtst TEST\_DIRECTORY

Similar to -dtr.

-od OUTPUT\_DIRECTORY

The output directory where to write the result files. The directory is automatically created if it does not already exist.

-o OUTPUT\_DATA\_FORMAT

csv Comma-separated values format. The default is tab-separated.

-rand SEED

Specifies a seed for creating random folds for running multiple runs of RL with cross-validation. SEED is an integer. On Unix-like systems and on Windows, a random integer is provided by the RANDOM environment variable. If this option is not specified, the default seed is 1.

-cmbf DIRECTORY

Combine the files in DIRECTORY. Each file represents one training example (such as a mass spectrum), and contains two comma-separated columns. The first column contains the names of the variables (such as M/Z values). The second column contains the values for those variables (e.g., intensity values).

-src

This parameter flags the source data file(s) for learning rules for transfer in TRL, MS-TRL, and iTRL frameworks. One (for TRL) or more (for MS-TRL or iTRL) files can be specified after this flag. Note that this flag is mandatory for the aforementioned transfer rule learning frameworks.

* TRAINING\_FILE

The training data file is specified as the last argument. This argument must ALWAYS be specified. For all transfer learning algorithms, this data file is designated as the target dataset.

* OUTPUT

The program prints to standard output a log of its working that includes the program parameters (and learning parameters), classifier learned, predictive performance statistics, starting time and total running time. For each rule, the log includes the following statistics: CF, CF/cost, p-value, true positive (TP) count, false positive (FP) count, and test TP and test FP. These last two statistics are the number of test examples for which the rule was applied correctly (TP) or incorrectly (FP) when using the whole model. When applying the model, a rule may not fire even if it matches a test example, because of interaction with other rules. (See the discussion of evidence gathering under the “-inftype” parameter.) The program also creates some output files containing any pre-processed data, rules learned on the whole training data set, prior rules learned from source(s), rules learned from each cross-validation fold (if cross-validation was used), predictions on the data instances used in validation, and the predictive performance of the model calculated from the predictions. The files are in the output directory, which by default is named as TRL\_run\_YYYY-MM-DD-hhmmss where the last part of the file name is the time when the program was run. A dedicated output directory can be specified using the “-od” parameter.

* EXAMPLES

Running KARL on the target dataset, GEO6956.txt by using the lookup table, GEO6956\_LTable.csv. As the discretization method EBD with default lambda parameter, 0.5, has been set as the command arguments:

-lp -tr 2 -lookuptable Demo-data/GEO6956\_LTable.csv -d 9 0.5 n -dp Demo-data/GEO6956.txt

* AUTHORS

Jonathan Lustgarten, 2006 - 2009

Philip Ganchev, 2009 - 2010

Jeya Balaji Balasubramanian, 2011 - 2015

Henry Ato Ogoe, 2014 - 2016