Preference and Rule Learning Documentation

Release latest

Mar 17, 2019

Contents

PRL is a preference learning algorithm which learns the maximal margin hypothesis by incrementally solving a two-player zero-sum game. The algorithm has theoretical guarantees about its convergence to the optimal solution. PRL has been presented @ AAAI 2019; the reference paper is:

13. Polato and F. Aiolli, "Interpretable preference learning: a game theoretic framework for large margin on-line feature and rule learning", AAAI 2019.

CHAPTER 1

Installing PRL with Pypi (Python 3)

PRL is available in the PyPi repository and it can be installed with

```
pip3 install prl
```

and then it can be imported in python with

```
import prl
```

CHAPTER 2

How to use the PRL module

It is possible to run PRL using the provided python script ‘run_prl.py’.

CHAPTER 3

Configuration file

In order to correctly run the script the configuration file must be initialized. An example of configuration file is provided in 'config/config.json'

```
{
    "algorithm" : "PRL",
    "feat_gen" : "GenHPolyF",
    "feat_gen_params" : [3],
    "pref_generator" : "macro",
    "columns_budget" : 1000,
    "iterations" : 10,
    "solver" : "LinProg",
    "solver_params" : [0]
}
```

The meaning of each configuration attribute is described in the following: - 'algorithm': it selects the PRL variation. Actually two PRL variations are implemented:

- 'PRL': which is the standard algorithm as presented in the paper mentioned above;
- 'PRL_ext': that is slight different from PRL, in the sense that the budget of columns is not fixed, but at each iterations 'columns_budget' number of new columns are generated. This variation guarantees that regardless of the initial budget at some iteration the number of columns will be enough to guarantee the convergence to the optimum (as the number of iterations increases).
- 'feat_gen': it indicates which feature generator will be used by PRL. Feature generators are implemented in the script 'genF.py' and at the moment the following feature generators scheme are implemented:
  - 'GenLinF': generates linear features, i.e., it randomly pick a feature from the input ones;
  - 'GenHPolyF': generates homogeneous polynomial features of the specified degree;
  - 'GenDPKF': generates dot-product polynomials features of the specified degree. It mainly differs from 'GenHPolyF' on the weighting scheme of the features;
  - 'GenConjF': it assumes binary/boolean input vectors and it generates conjunctive (logical AND) features of the specified arity;

  - 'GenRuleF': generates conjunctions of logical rules over the input attributes. The generated rules has a form like 'age >= 10 and height <= 160'. The arity of the conjunction is a parameter of the generator;
  - 'GenRuleEqF': like 'GenRuleF', but the only relation considered is the equality ('==').
- 'feat_gen_params': ordered list of parameters for the selected feature generator. For more details, please refer to the documentation of the generators;
- 'pref_generator': it indicated which preference generator will be used by PRL. The possible preference generation schemes are:
  - 'macro': a macro preference describes preferences like

    y_i is preferred to 'y_j' for the instance 'x_i', where '(x_i, y_i) in X x Y', while '(x_i, y_j) not in X x Y'. This kind of preferences are suitable for label ranking tasks.
  - 'micro': a micro preference describes preferences like

    '(x_i, y_i)' is preferred to '(x_j, y_j)', where '(x_i, y_i) in X x Y', while '(x_j, y_j) not in X x Y'. This kind of preferences are suitable for instance ranking tasks.
- 'columns_budget': the number of columns of the matrix game;
- 'iterations': number of iterations of PRL;
- 'solver': the algorithm for solving the game. Up to now the available algorithms are 'FictitiousPlay', 'AMW' and 'LinProg';
- 'solver_params': it is the ordered list of parameters of the solver. For more details, please refer to the documentation of the solvers.

CHAPTER 4

Run PRL

Once the configuration file is ready, PRL can be trained and evaluated by using the provided script

```
python3 run_prl.py [OPTIONS] dataset
```

where 'dataset' must be an svmlight file and the possible options are the following:

- '-c CONFIG_FILE', '–config_file CONFIG_FILE': 'CONFIG_FILE' specifies the path of the configuration file (default: 'config/config.json');
- '-t SIZE', '–test_size SIZE': 'SIZE' specifies the portion (in percentage, float between 0 and 1) of the dataset will be used as test set (default: 0.3);
- '-s SEED', '–seed SEED': ''SEED" specifies the pseudo-random seed. Useful for replicability purposes (default: 42);
- '-v', '–verbose': whether the output it is verbose or not;
- '-h', '–help': shows the help.

An example of run, using the configuration file as above, is:

```
python3 run_prl.py -t 0.2 -s 1 -v
```

which runs PRL using 80% of the dataset as training set and the rest as test set, using 1 as pseudo-random seed and a verbose output.

CHAPTER 5

Evaluation

The evaluation is computed in terms of *accuracy*, *balanced accuracy* and it also shows the *confusion matrix*.

CHAPTER 6

Version

0.94b

CHAPTER 7

Requirements

PRL requires the following python modules:

- CVXOPT
- Numpy
- Scikit-learn
- Scipy