The SECO-Framework for Rule Learning

Frederik Janssen and Markus Zopf

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Motivation
2. Separate-and-conquer Rule Learning in the SECO-Framework
   - The Separate-and-conquer strategy
   - Components of the framework
   - Example configuration
3. Capabilities of the framework
   - Rule learning algorithms
   - Weighted covering, search strategies, and heuristics
   - Evaluation Package
4. Conclusions and Future Work

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- most rule learning algorithms use the separate-and-conquer strategy
- algorithms of this kind can be decomposed into different components according to [5]
- using an abstract algorithm lets the user configure many existing and new algorithms
- it is beneficial to be able to change single components of an algorithm

The Separate-and-conquer strategy

TECHNISCHE UNIVERSITÄT DARMSTADT

1. learn a “good” rule from the given data
2. add the rule to the rule set and remove all examples covered by the rule; if positive examples are left **GOTO** 1

- what “good” means usually is measured by a *heuristic*

The Separate-and-conquer strategy

AbstractSeCo*(Examples)*

TECHNISCHE UNIVERSITÄT DARMSTADT

Theory $\leftarrow \emptyset$
Growing = SplitData *(Examples, Splitsize)*
Pruning = SplitData *(Examples, 1-Splitsize)*
while ExamplesLeftToCover*(Growing)* **do**
    Rule = FindBestRule*(Growing,Pruning)*
    Covered = Cover *(Rule,Growing)*
    if RuleStoppingCriterion *(Theory,Rule,Growing)* **then**
        exit while
    Growing = *Growing* \ *Covered*
    Theory = *Theory* $\cup$ *Rule*
Theory = PostProcess *(Theory, Growing, Pruning)*
return *(Theory)*

The Separate-and-conquer strategy

FINDBESTRULE*(Growing, Pruning)*

TECHNISCHE UNIVERSITÄT DARMSTADT

InitRule = INITIALIZERULE *(Growing)*
InitVal = EVALUATERULE *(InitRule)*
BestRule = <*InitVal, InitRule*>
Rules = {*BestRule*}
while *Rules* ≠ ∅ **do**
    Candidates = SELECTCANDIDATES *(Rules, Growing)*
    Rules = *Rules* \ *Candidates*
    for *Candidate* ∈ *Candidates* **do**
        Refinements = REFINERULE *(Candidate, Growing)*
        for *Refinement* ∈ *Refinements* **do**
            Evaluation = EVALUATERULE *(Refinement, Growing, h)*
            unless STOPPINGCRITERION *(Refinement, Evaluation, Pruning)*
            NewRule = <*Evaluation, Refinement*>
            Rules = INSERTSORT *(NewRule, Rules)*
            if *NewRule* > *BestRule* **then**
                BestRule = *NewRule*
    Rules = FILTERRULES *(Rules, Growing)*
return *(BestRule)*

Components of the framework

TECHNISCHE UNIVERSITÄT DARMSTADT

interface	purpose
BinarizationMode	used to determine the type of binarization (ordered or unordered 1-vs-all)
ClassificationMode	selects the type of classification (decision list or (weighted) voting)
RuleInitializer	used to initialize the rule (either empty or by selecting a random example)
CandidateSelector	selects the candidate rules for the next iteration (for implementing various search algorithms)
Heuristic	the heuristic to evaluate all candidate rules
RuleRefiner	creates refinements of rules (bottom up, top down, or bidirectional)
RuleFilter	filters out unpromising candidates
StoppingCriterion	used to stop the refinement of a candidate rule (usually when it does not cover any negatives)
RuleStoppingCriterion	stops the induction of rules (e.g., significance tests)
PostProcessor	implements the post processing method of Ripper [2]

- note that all components are configurable via *xml*

Example configuration of RIPPER [2]

TECHNISCHE UNIVERSITÄT DARMSTADT

```
<seco growingSetSize="2/3" minNo="2" weighted="false" seed="0">
    <binarizationmode classname="OrderedBinarization"/>
    <classificationmode classname="DecisionList"/>
    <ruleinitializer classname="TopDownRuleInitializer"/>
    <candidateselector classname="SelectAllCandidatesSelector"/>
    <heuristic classname="FoilGain"/>
    <rulerefiner classname="TopDownRefiner"
        nominal.cmpmode="equal"/>
    <stoppingcriterion classname="NoNegativesCoveredStop"/>
    <rulestoppingcriterion classname="MDLStoppingCriterion"/>
    <rulefilter classname="BeamWidthFilter" beamwidth="1"/>
    <postprocessor classname="PostProcessorRipper"
        optimizations="2"/>
</seco>
```

Capabilities of the framework

Rule learning algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- currently the following algorithms are implemented in the framework
  - RIPPER [3]
  - CN2 [1]
  - AQ [7]
  - SIMPLESECO [6]
- but note that nearly every separate-and-conquer rule learning algorithm can be configured
- especially the exchange of single components of a rule learning algorithm such as the heuristic can be done easily
  - often, a new paper about an algorithm simply exchanges some components of the algorithm (cf. the progress for CN2 [1,2])

Capabilities of the framework

Weighted covering, search strategies, and heuristics

TECHNISCHE UNIVERSITÄT DARMSTADT

- weighted covering: a covered example is not removed entirely but only its weight is being reduced
- search strategies
  - top-down, bottom-up, and bidirectional search
  - hill-climbing, beam, and exhaustive search
- heuristics (excerpt)
  - Laplace, Weighted Relative Accuracy, Correlation, Odds Ratio
  - Relative Cost Measure, F-Measure, m-estimate, Klösgen Measure (for all parametrized heuristics a best parameter is implemented according to [6]), LinearRegression (meta-heuristic derived in [6])

Capabilities of the framework

Evaluation package

TECHNISCHE UNIVERSITÄT DARMSTADT

- a complete package for evaluating classification algorithms is included in the framework
- different modes (configurable in *xml*): *createAndSaveModel*, *existingModelEvaluation*, *trainTestSplit*, *trainTestValidation*, *crossValidation*
- different output formats (csv, console, etc.)
- serialization of rule sets
- statistical tests
  - Friedman with post-hoc Nemenyi as suggested in [4]

Conclusions and future work

TECHNISCHE UNIVERSITÄT DARMSTADT

Conclusions

- building block architecture eases the implementation of new rule learning algorithms
- Evaluation package simplifies the comparison of different algorithms

Future Work

- implement more rule learning algorithms
- include an interactive component
- graphical user interface
- **publish the framework**

References

TECHNISCHE UNIVERSITÄT DARMSTADT

[1] P. Clark and T. Niblett. The CN2 Induction Algorithm. Machine Learning, 3(4):261-283, 1989.

[2] P. Clark and R. Boswell. Rule Induction with CN2: Some Recent Improvements. In Proceedings of the 5th European Working Session on Learning (EWSL-91), pages 151-163, Porto, Portugal, 1991. Springer-Verlag.

[3] William W. Cohen. Fast Effective Rule Induction. In Proceedings of the 12th International Conference on Machine Learning (ICML-95), pages 115-123, 1995.

[4] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. Machine Learning Research, Jan(7):1-30, 2006.

[5] Johannes Fürnkranz. Separate-and-Conquer Rule Learning. Artificial Intelligence Review, 13(1):3-54, February 1999.

[6] Frederik Janssen and Johannes Fürnkranz. On the quest for optimal rule learning heuristics. Machine Learning, 78(3):343-379, March 2010. DOI 10.1007/s10994-009-5162-2.

[7] R. S. Michalski. On the Quasi-Minimal Solution of the Covering Problem. In Proceedings of the 5th International Symposium on Information Processing (FCIP-69), volume A3 (Switching Circuits), pages 125-128, Bled, Yugoslavia, 1969.