Constructive Meta-Learning with Machine Learning Method Repositories

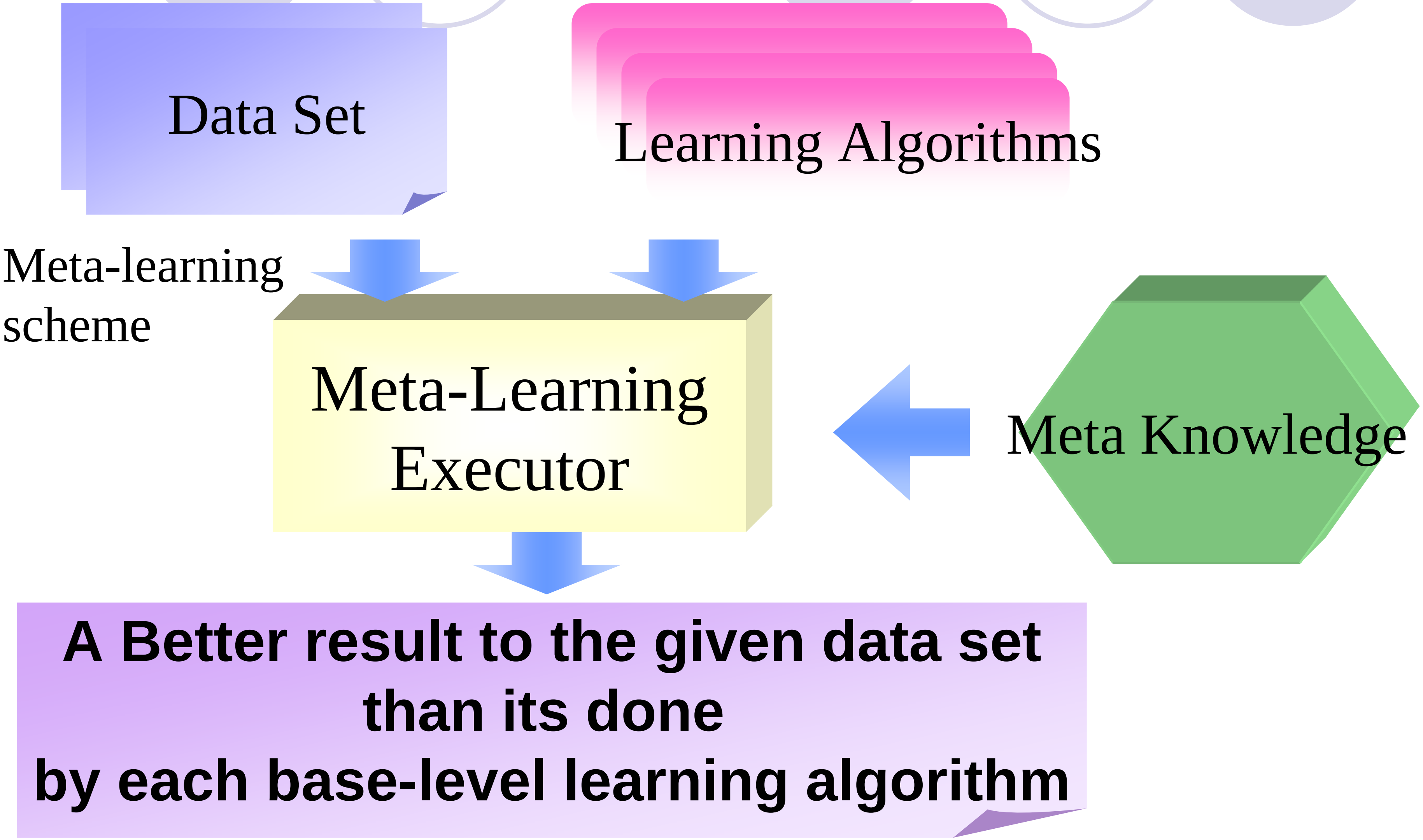
Hidenao Abe
Shizuoka University, JAPAN
hidenao@ks.cs.inf.shizuoka.ac.jp

Takahira Yamaguchi
Keio University, JAPAN
yamaguti@ae.keio.ac.jp

Contents

- Introduction
- Constructive meta-learning based on method repositories
- Experiment with common data sets
- Conclusion

Overview of meta-learning scheme



Selective meta-learning scheme and our motivation

- Integrating base-level classifiers, which are learned with different training data sets generating by
  - "Bootstrap Sampling" (bagging)
  - weighting ill-classified instances (boosting)
- Integrating base-level classifiers, which are learned from different learning algorithms
  - simple voting (voting)
  - constructing meta-level classifier with a meta-level training data set (stacking, cascading)

They don't work well, when no base-level learning algorithm works well to the given data set !!

-> It is time to de-compose base-level learning algorithms and re-construct a proper algorithm to the given data set.

Contents

- Introduction
- Constructive meta-learning based on method repositories
  - Basic idea of our constructive meta-learning
  - An implementation of the constructive meta-learning called CAMLET
  - Parallel execution and search for CAMLET
- Experiment with common data sets
- Conclusion

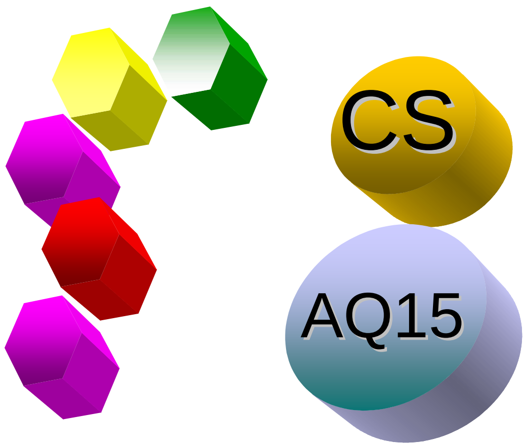
Basic Idea of our Constructive Meta-Learning

De-composition& Organization + Search & Composition

Analysis of two or more learning algorithms

Basic Idea of our Constructive Meta-Learning

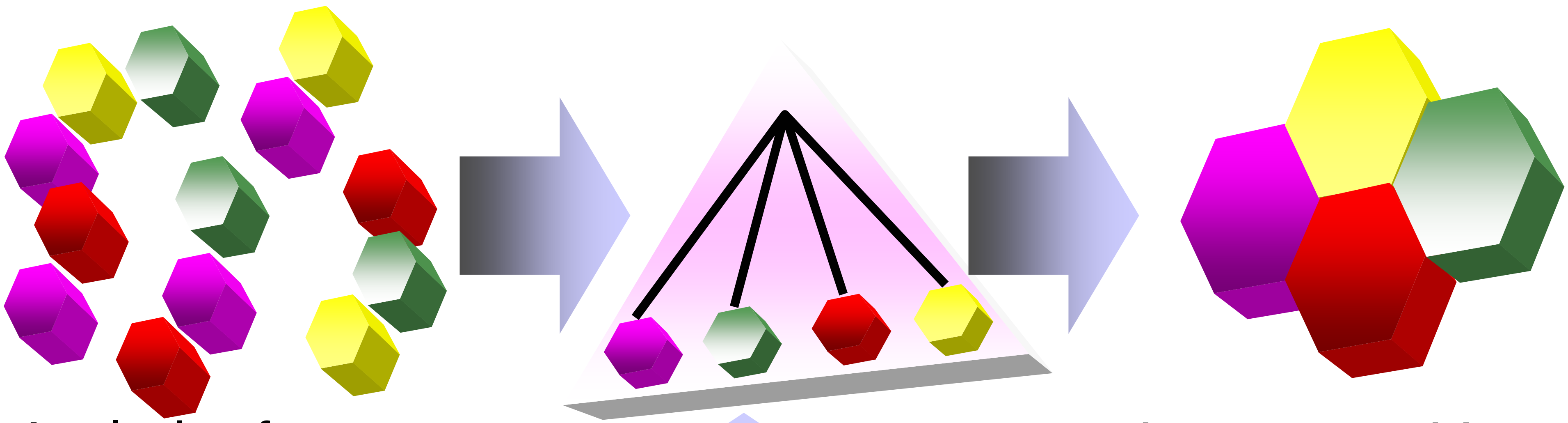
De-composition& Organization + Search & Composition



Analysis of two or more learning algorithms

Basic Idea of our Constructive Meta-Learning

De-composition& Organization + Search & Composition



Analysis of two or more learning algorithms

Automatic Composition of learning algorithms

Organizing learning methods, treated objects and control structures

Contents

- Introduction
- Constructive meta-learning based on method repositories
  - Basic idea of our constructive meta-learning
  - An implementation of the constructive meta-learning called CAMLET
  - Parallel execution and search for CAMLET
- Experiment with common data sets
- Conclusion

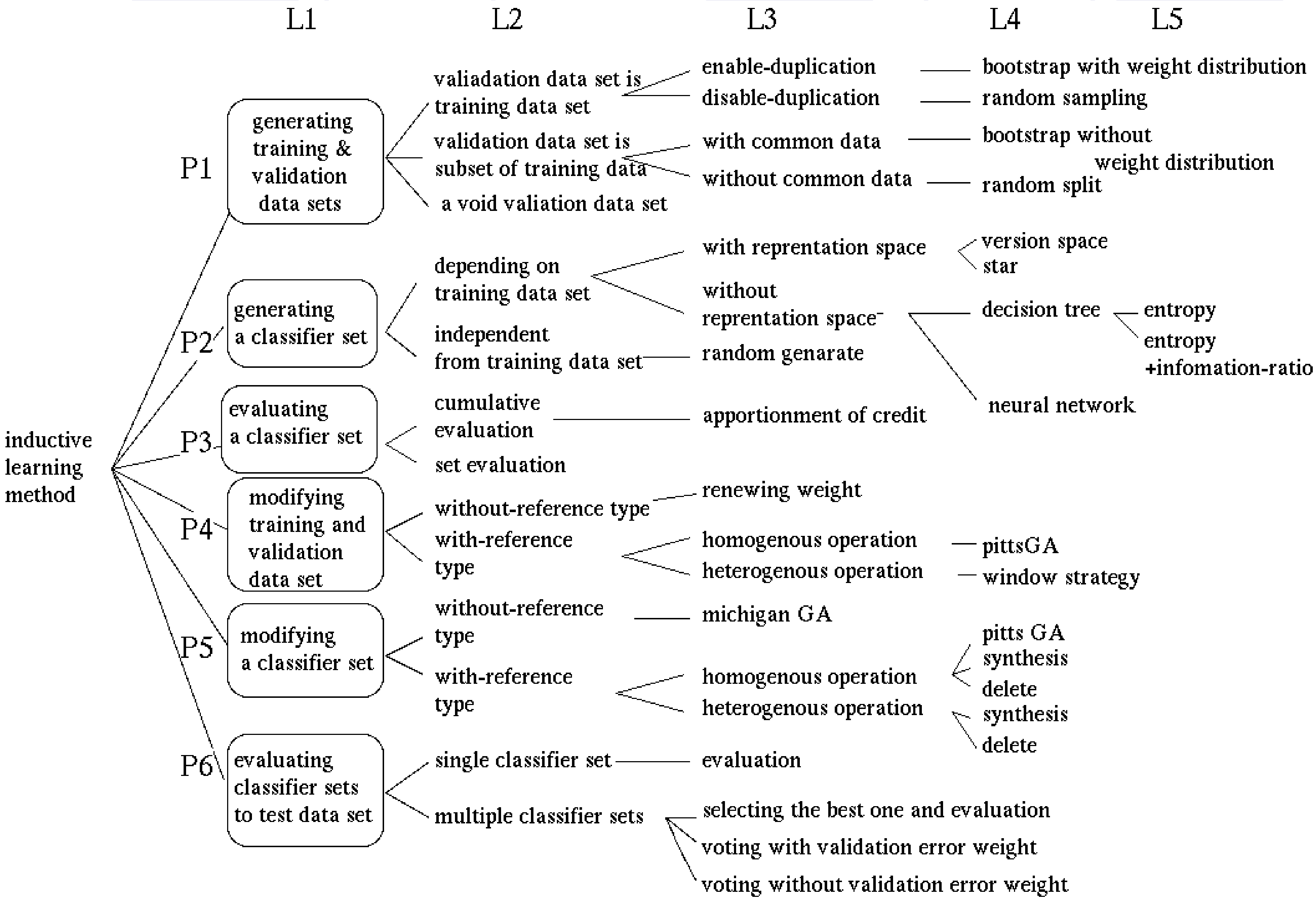
Analysis of Representative Inductive Learning Algorithms

We have analyzed the following 8 learning algorithms:

- Version Space
- AQ15
- ID3
- C4.5
- Boosted C4.5
- Bagged C4.5
- Neural Network with back propagation
- Classifier Systems

We have identified 22 specific inductive learning methods.

Inductive Learning Method Repository

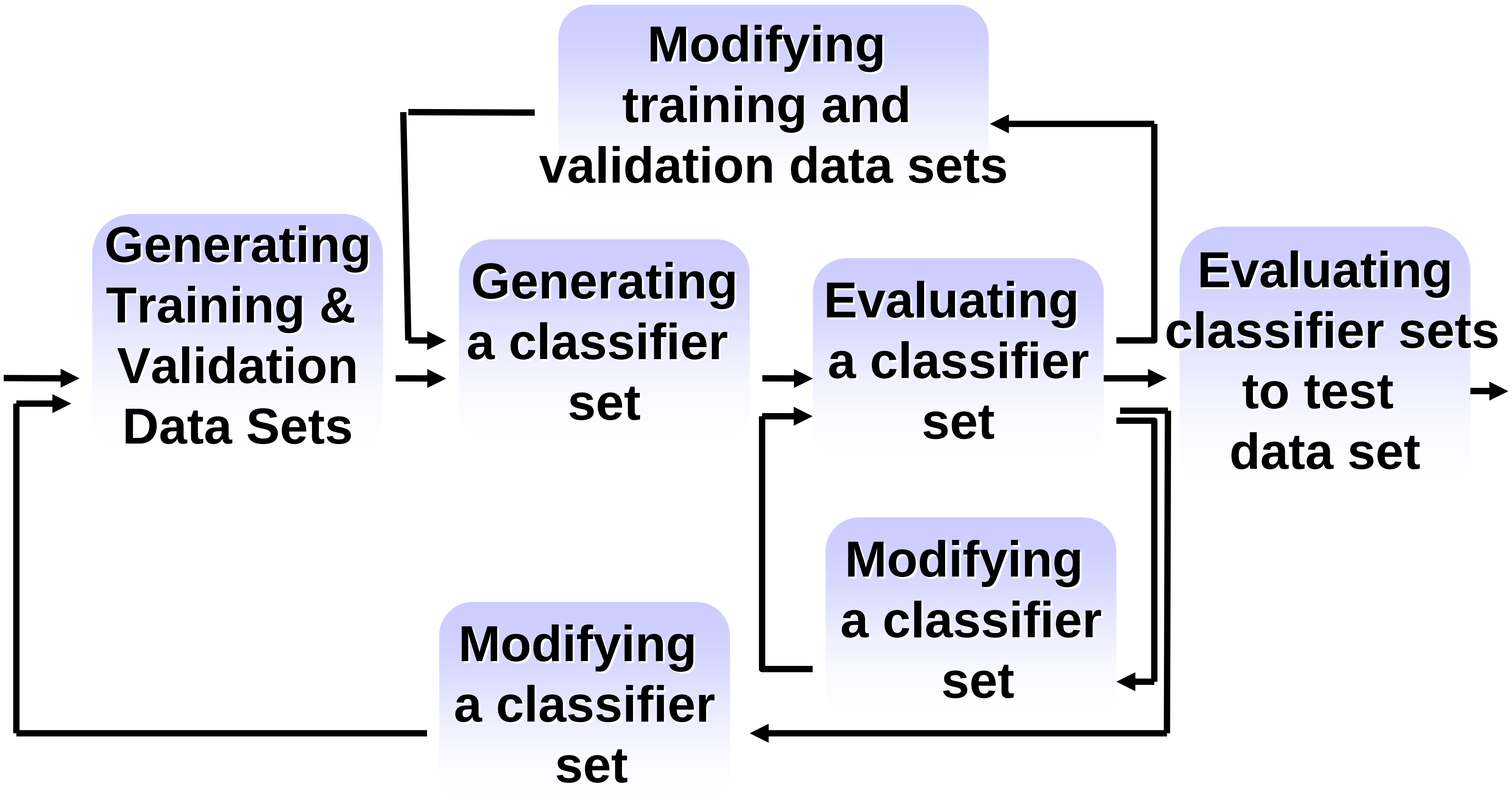


Data Type Hierarchy

Organization of input/output/reference data types for inductive learning methods

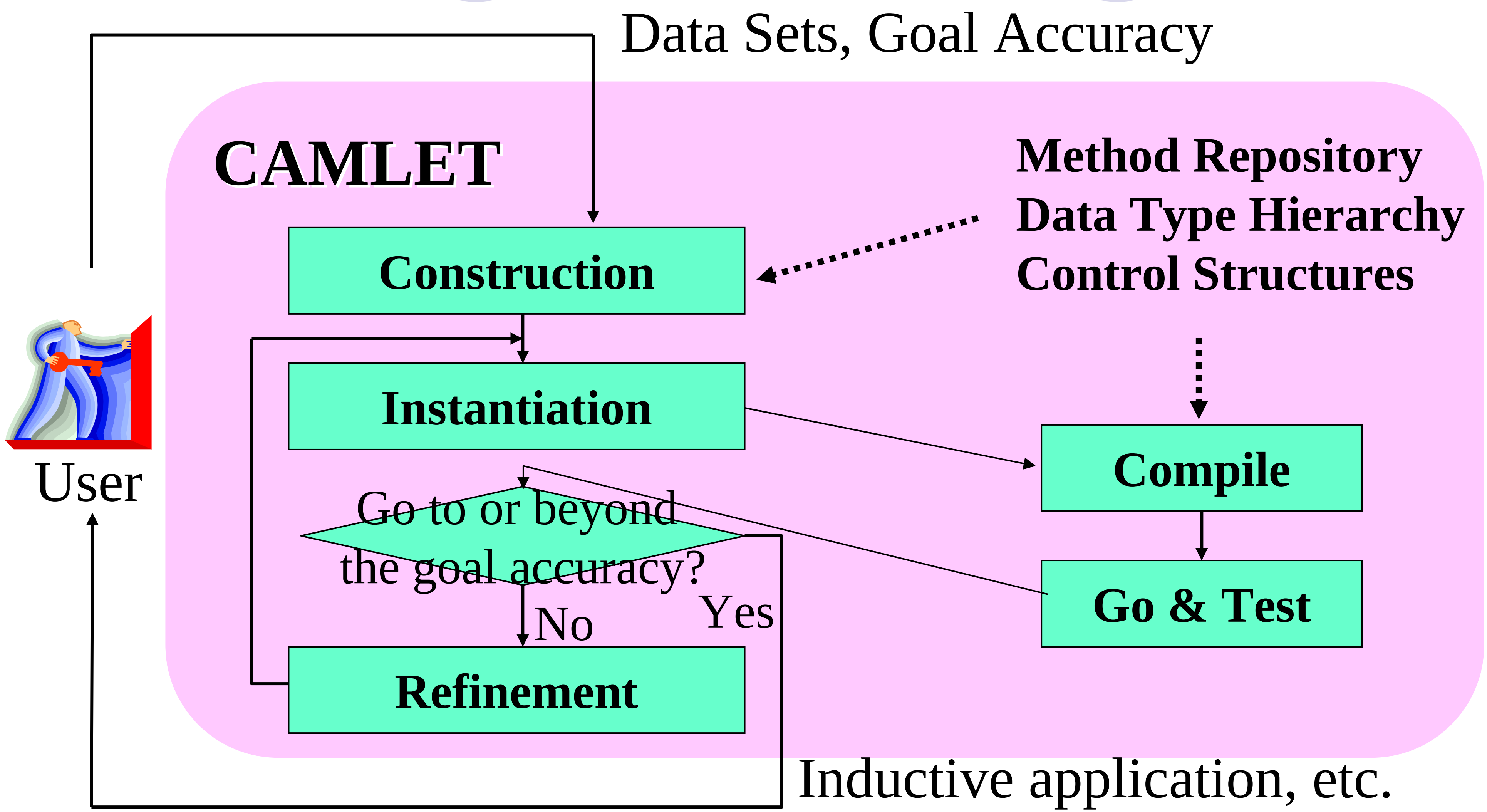
- Objects
  - classifier-set
    - If-Then rules
    - tree (Decision Tree)
    - network (Neural Net)
  - data-set
    - training data set
    - validation data set
    - test data set

Identifying Inductive Learning Methods and Control Structures

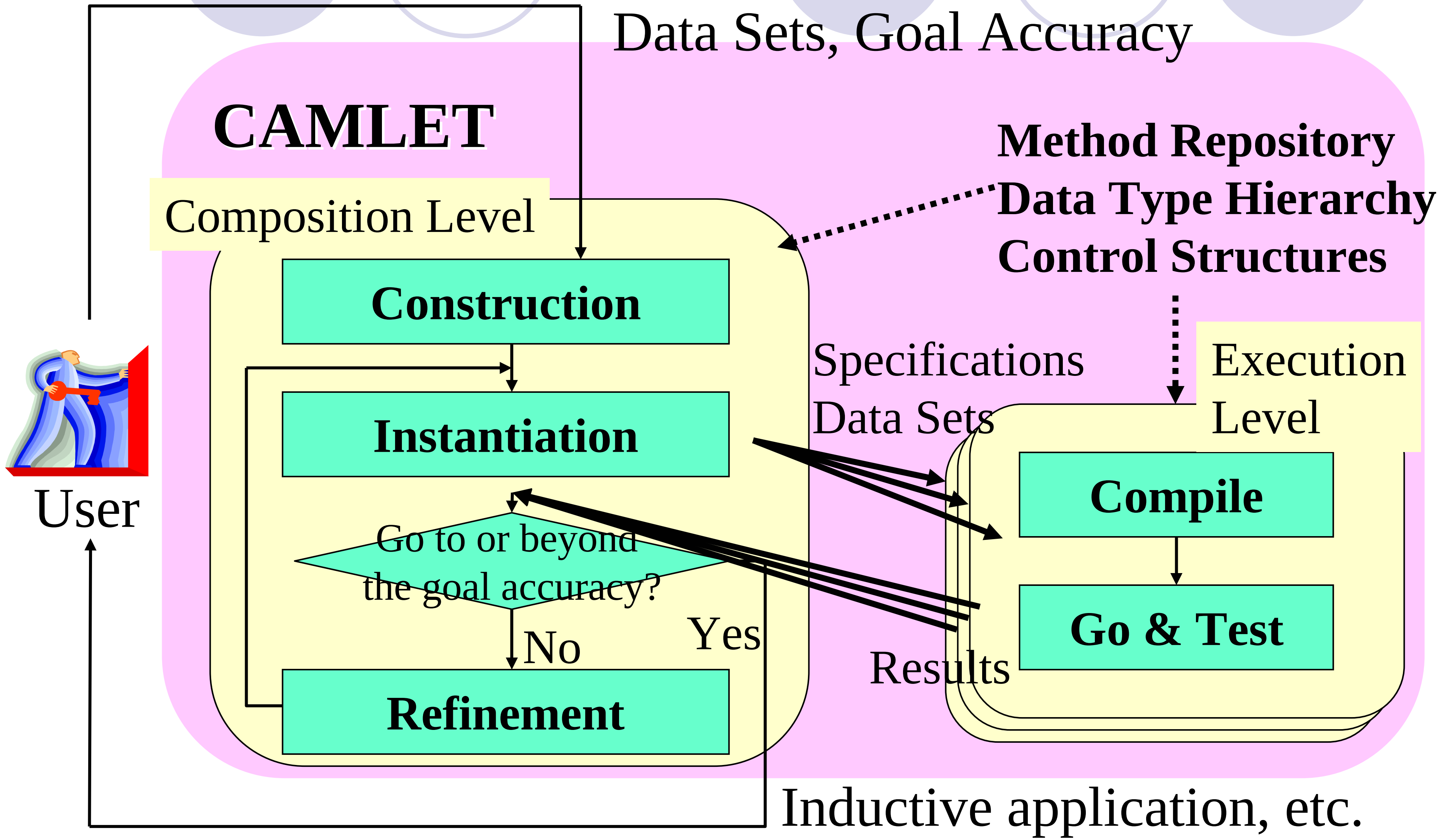


System overview of CAMLET:

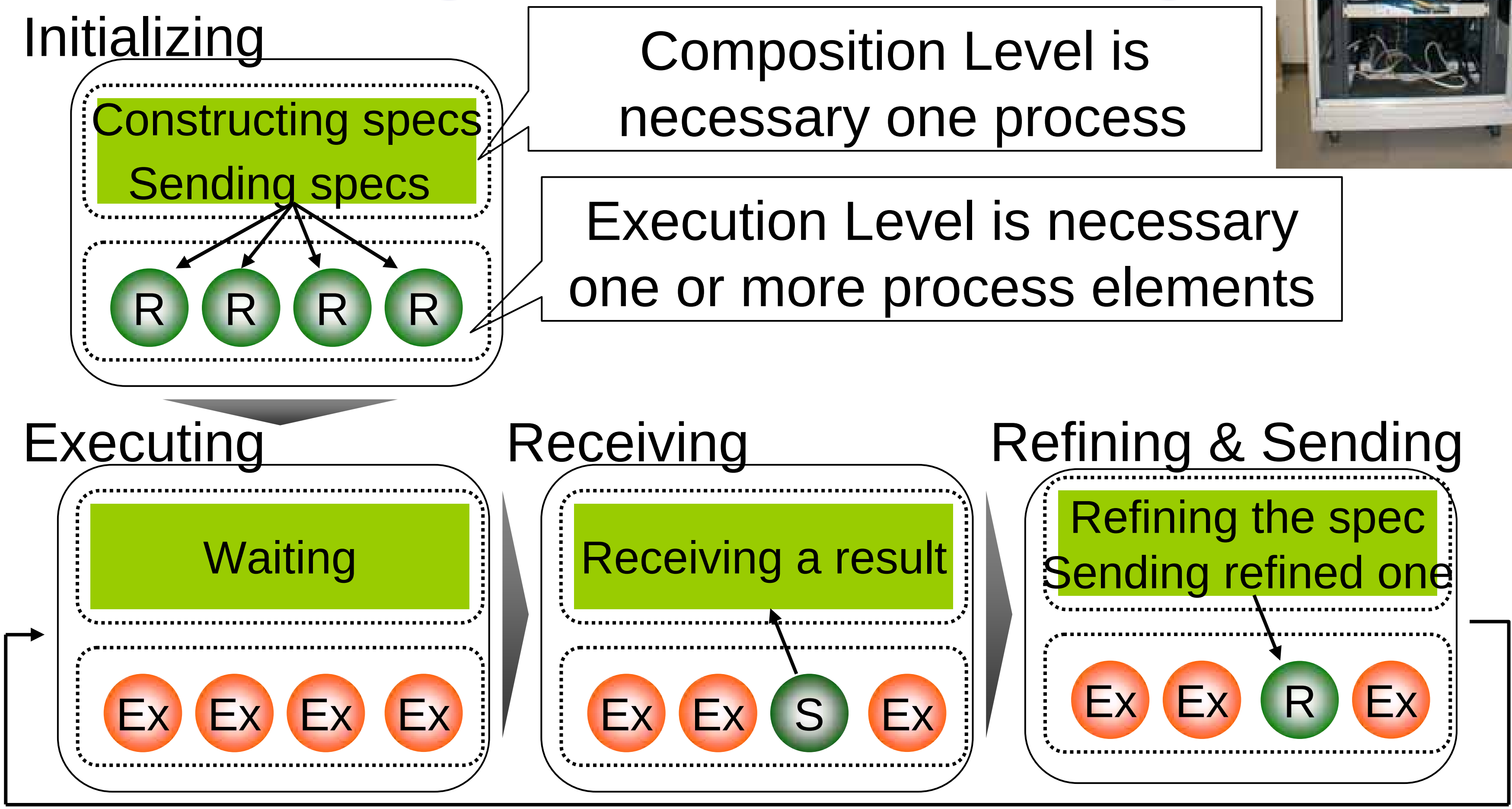
a Computer Aided Machine Learning Engineering Tool



CAMLET with a parallel environment

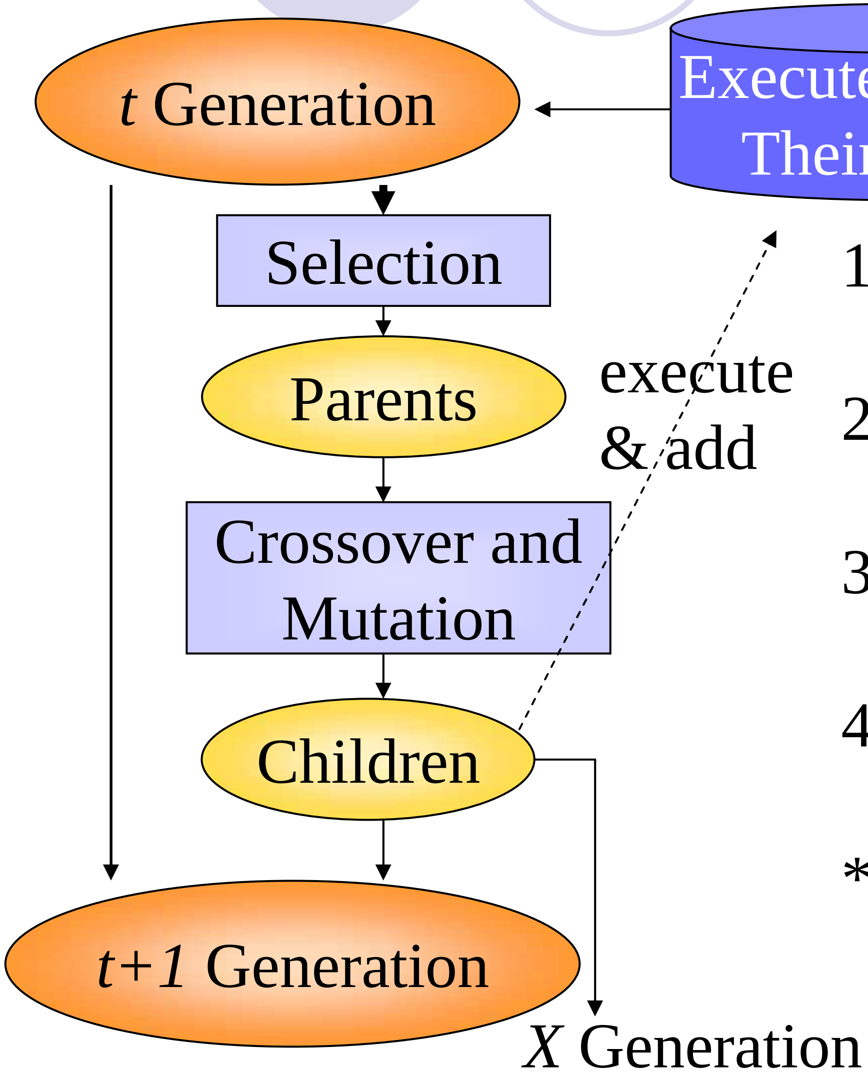


Parallel Executions of Inductive Applications



R: receiving ,Ex: executing, S: sending

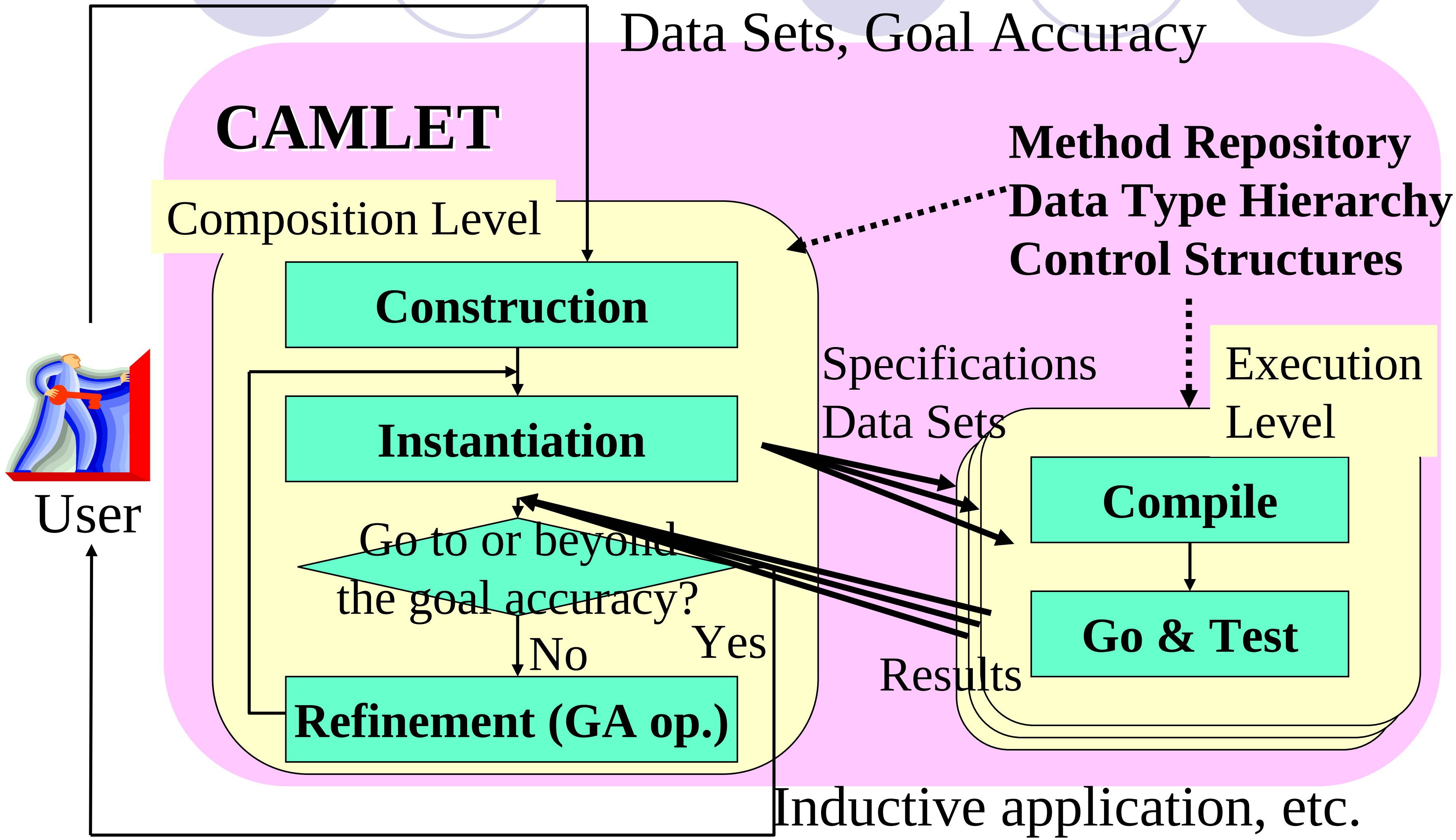
Refinement of Inductive Applications with GA



1. Transform executed specifications to chromosomes
2. Select parents with “Tournament Method”
3. Crossover the parents and Mutate one of their children
4. Execute children’s specification, transforming chromosomes to specs

* If CAMLET can execute more than two I.A at the same time, some slow I.A will be added to t+2 or later Generation.

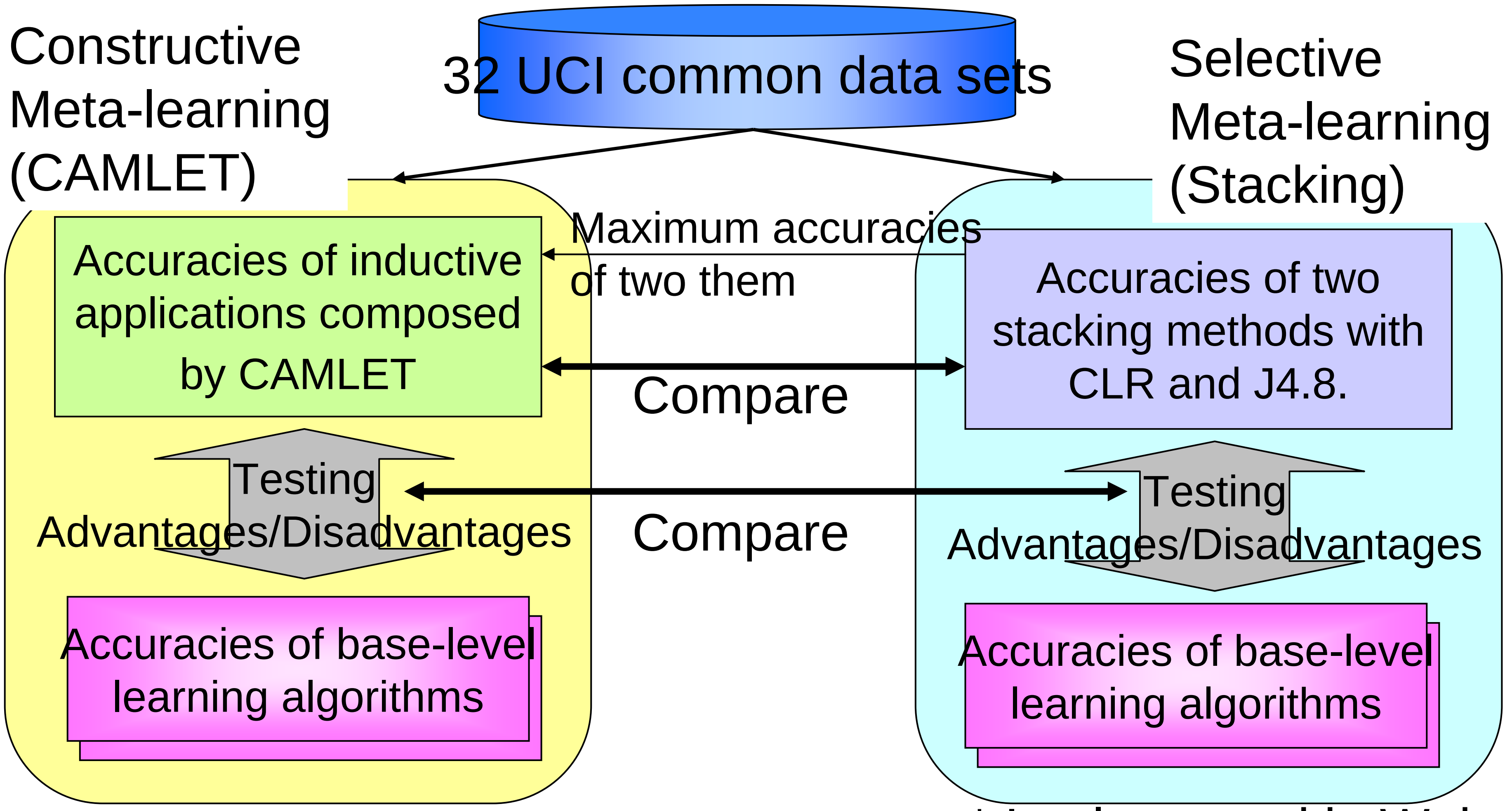
CAMLET on a parallel GA refinement



Contents

- Introduction
- Constructive meta-learning based on method repositories
- Experiments: Accuracy comparison using UCI common data sets
  - Comparing with stacking methods
  - Comparing with each base-level learning algorithms
- Conclusion

Experiment overview



* Implemented in Weka

Configuration of Stacking methods

Meta-level Learning Algorithms

J4.8
(with pruning, $C=0.25$)

Linear Regression
(with all of features)

J4.8 (with pruning, $C=0.25$)

Naïve Bayes

Part (with pruning, $C=0.25$)

IBk ($k=5$)

Bagged J4.8 (Sampling rate=55%, 10 Iterations)

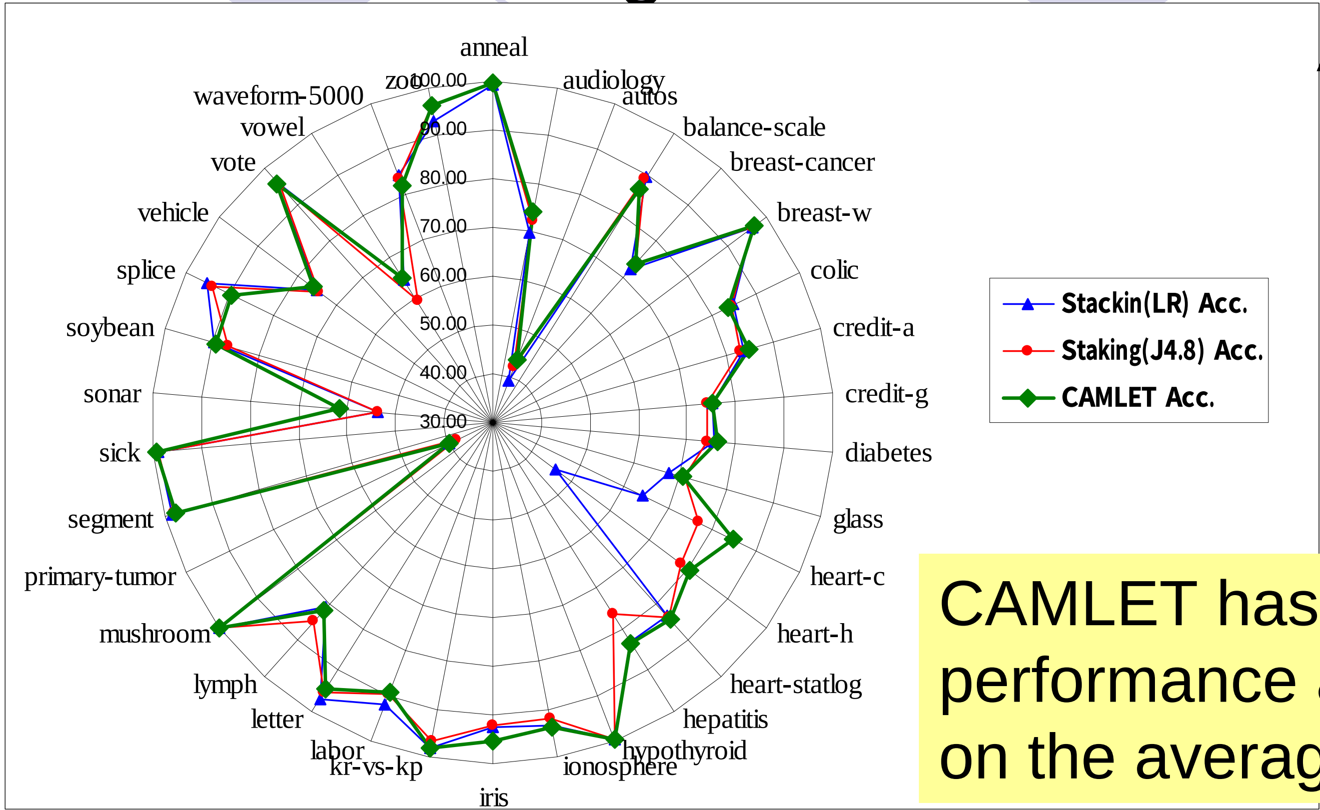
Boosted J4.8 (10 Iterations)

Base-level Learning Algorithms

Configuration of CAMLET

- We have input goal accuracies to each data set as the requirement of CAMLET.
  - These accuracies have been the maximum accuracies of the two stacking methods.
- CAMLET has output just one specification of the inductive application to each data set and its accuracy.
  - CAMLET has searched about 6,000 inductive applications for the best one, executing <u>up to one hundred inductive applications</u>.
- 10 “Execution Level” PEs have been used to each data set.

Evaluation on 10-CV accuracies of CAMLET and two stacking methods



Accuracy(%)= (#correctly classified/ #total test instances) *100

CAMLET has achieved as good performance as the given goals on the average.

	Stacking			CAMLET
	LR	J4.8	Max of two	
Average	81.52	82.38	83.34	83.58

Comparing test results of each base-level learning algorithms

1. We have got accuracies to 32 UCI data sets with 10-fold CV.
2. We have tested performances of meta-learning schemes and their base-level learning algorithms with t-test.

CAMLET vs. its base-level algorithms						
	C4.5(unpruned)	ID3(unpruned)	CS	NeuralNet	Boosted C4.5	Bagged C4.5
CAMLET	12:0	15:0	21:0	19:0	7:0	5:0
Stacking vs. its base-level algorithms						
	j4.8	Part	NB	Ibk(k=5)	Boosetd j4.8	Bagged j4.8
CLR	6:0	8:1	12:3	10:2	3:0	5:1
j4.8	7:0	6:0	10:1	9:0	2:0	3:0

Number of win/loss(#win:#loss) where algorithm in row significantly outperforms algorithms in column

Summary of the result

- CAMLET works well as a meta-learning scheme compared with the two stacking methods.
  - CAMLET has been able to compose proper inductive applications to given data sets and goals to them.
  - The inductive applications composed by CAMLET to 32 UCI data sets significantly outperform its base-level learning algorithms.

Contents

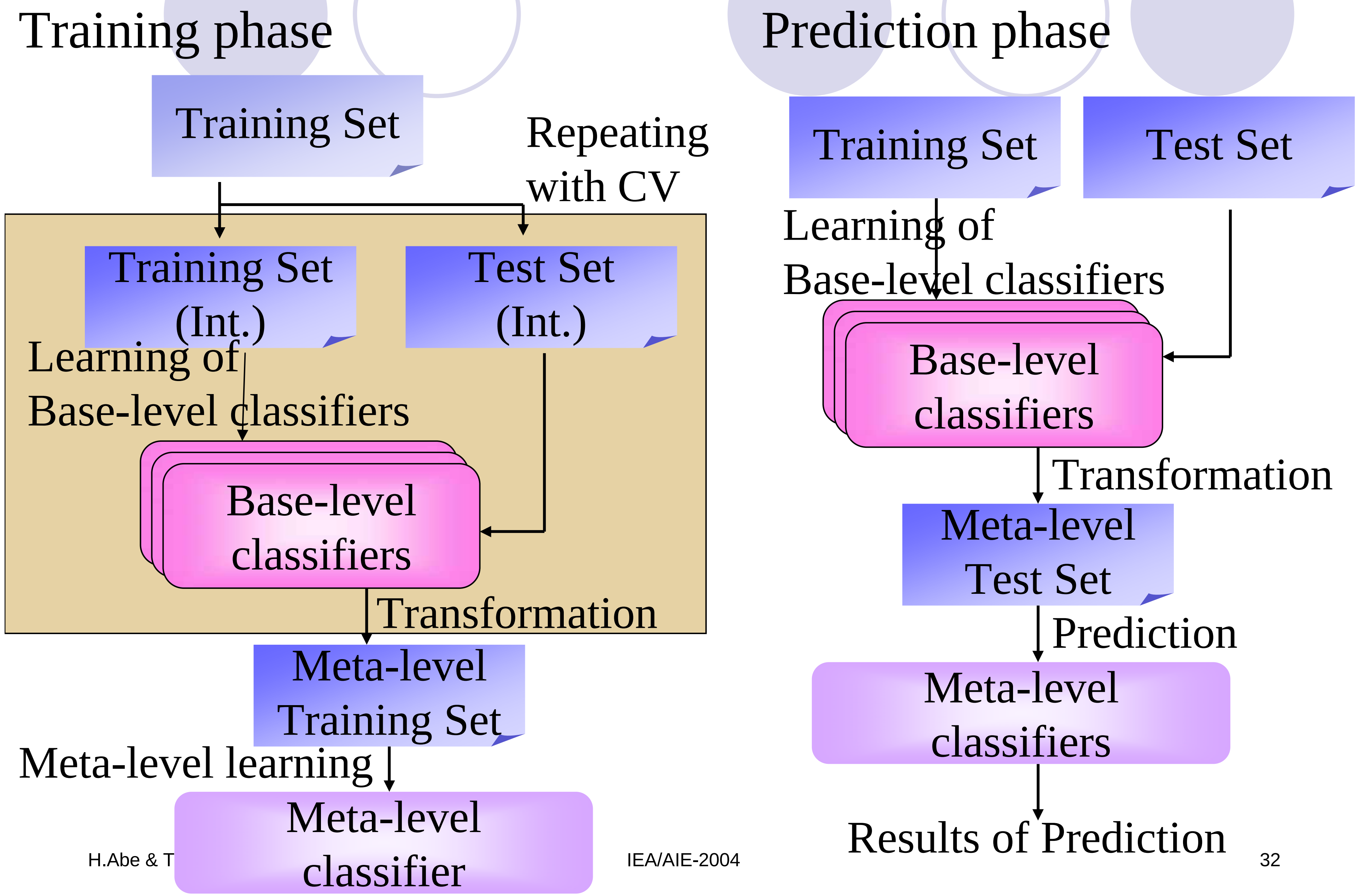
- Introduction
- Constructive meta-learning based on method repositories
- Case Study with common data sets
- Conclusion

Conclusion & Future Work

- CAMLET has been implemented as a tool for “Constructive Meta-Learning” scheme based on method repositories.
- CAMLET shows us a significant performance as a meta-learning scheme.
  - We are extending the method repository to construct data mining application of whole data mining process.
    - Task description of data mining process
    - Modeling method of criteria to describe a user requirement

Thank you!

Stacking: training and prediction



How to transform base-level attributes to meta-level attributes

Learning with Algorithm 1

Classifier by Algorithm 1

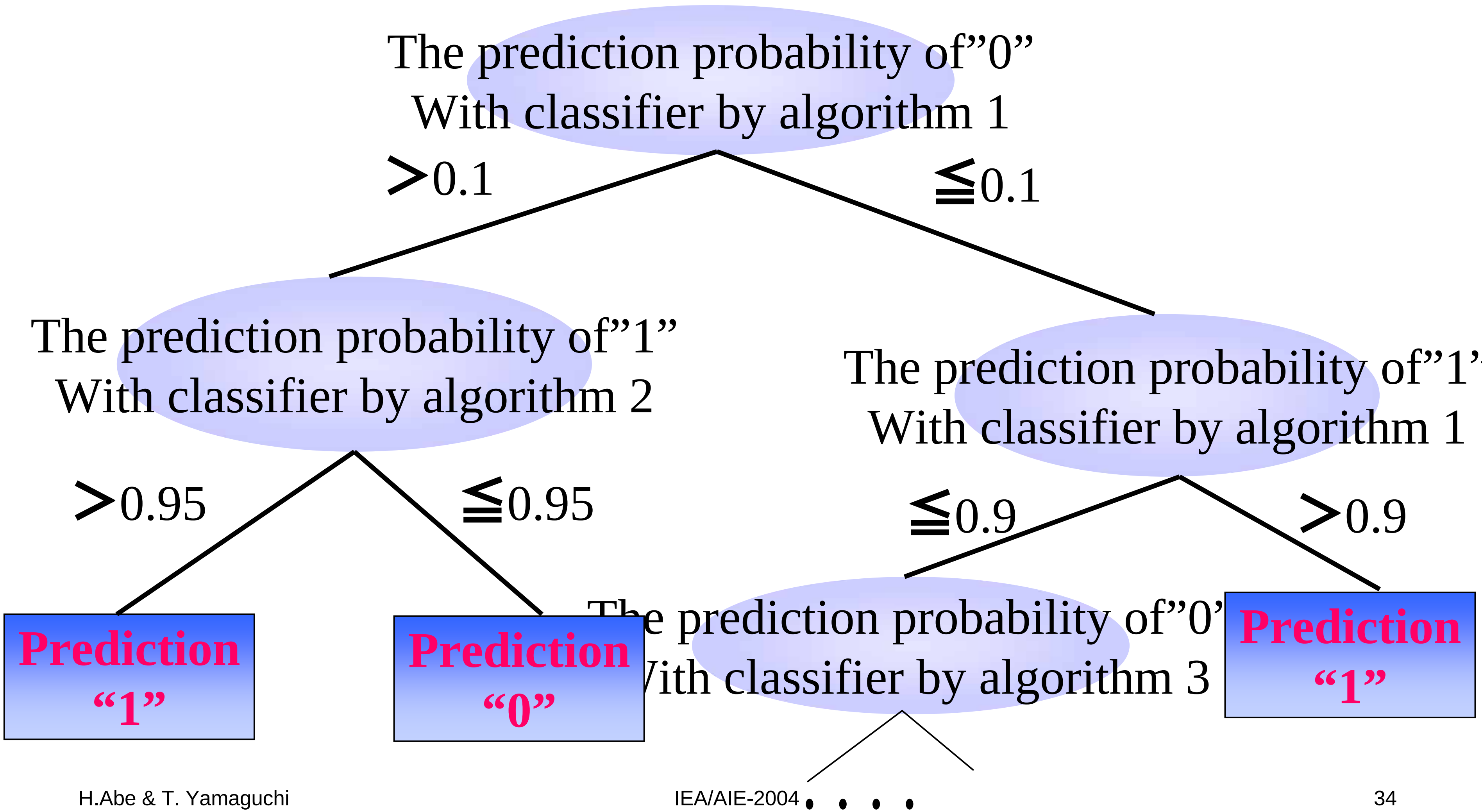
Att. A	Att. B	Att. C	Classes
a1	0.3	c3	0
a2	5	c2	1
…	…	…	…

1. Getting prediction probability to each class value with the classifier.
2. Adding base-level class value of each base-level instance

Algorithm 1		
Prob. Of Predicting "0	Prob. Of Predicting "1	Classes
0.99	0.01	0
0.4	0.6	1
…	…	…

A algorithm
C class values
#Meta-level Att. = *AC*

Meta-level classifiers (Ex.)



Accuracies of 8 base-level learning algorithms to Statlog data sets

Stacking
Base-level algorithms

Algorithms	Avg.(10 data sets)
J4.8	84.84
IBk(5)	84.32
Part	84.10
NaiveBayes	76.53
Bagging(5)	85.29
Bagging(10)	86.17
Boosting(5)	85.56
Boosting(10)	85.94
Max	87.58

Accuracy (%)

CAMLET
Base-level algorithms

Algorithms	Avg.(10 data sets)
C4.5 DT	81.07
ID3 DT	81.76
NeuralNetwork	63.09
ClassifierSystems	64.85
Bagging(5)	84.28
Bagging(10)	85.17
Boosting(5)	84.52
Boosting(10)	85.59
Max	87.04

Accuracy (%)

C4.5 Decision Tree without pruning (reproduction of CAMLET)

Generating training and validation data sets
with a void validation set

↓

Generating a classifier set (decision tree)
with entropy + information ratio

↓

Evaluating a classifier set
with set evaluation

↓

Evaluating classifier sets to test data set
with single classifier set evaluation

1. Select just one control structure
2. Fill each method with specific methods
3. Instantiate this spec.
4. Compile the spec.
5. Execute its executable code
6. Refine the spec. (if needed)