Hyperparameter Optimization for Software Effort Estimation

Tianpei Xia

txia4@ncsu.edu

Advisor: Tim Menzies

Nov 29th, 2018

Overview

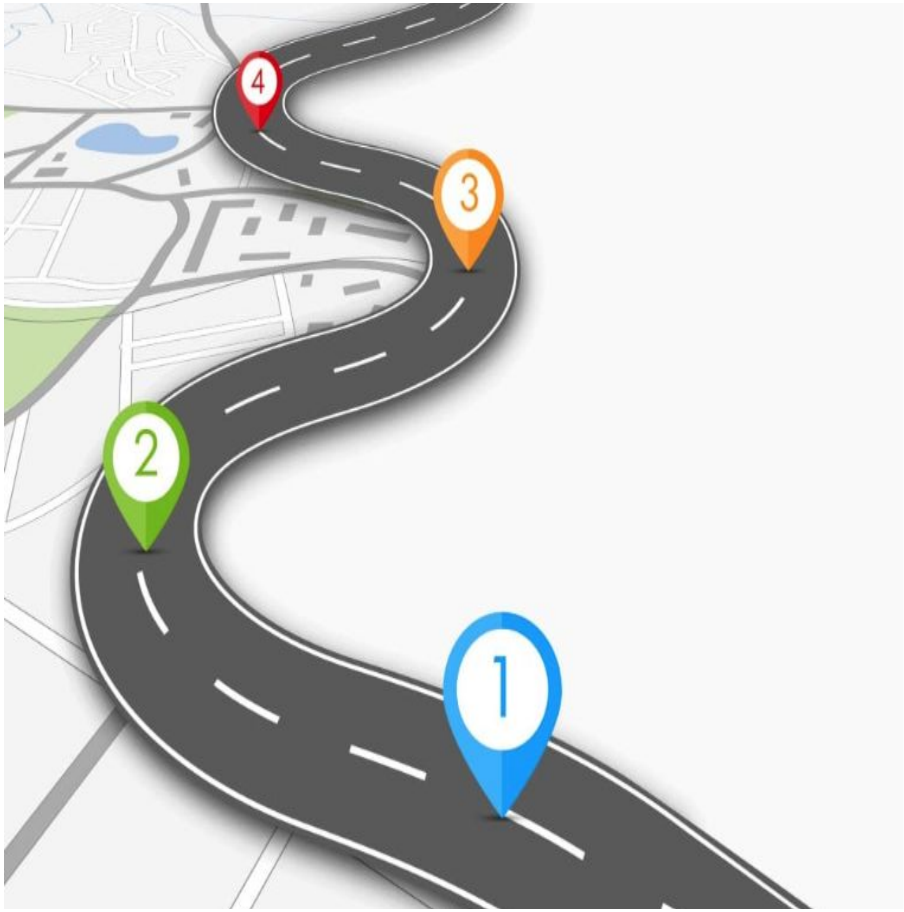
Proposal (supported by current evidence):

RATE (Rapid Automatic Tuning for Effort-estimation)

- A novel configuration tool for effort estimation
- Based on the combination of regression tree learners and optimizers (e.g. Different Evolution [**Storn' 1997**], FLASH [**Nair' 2017**])

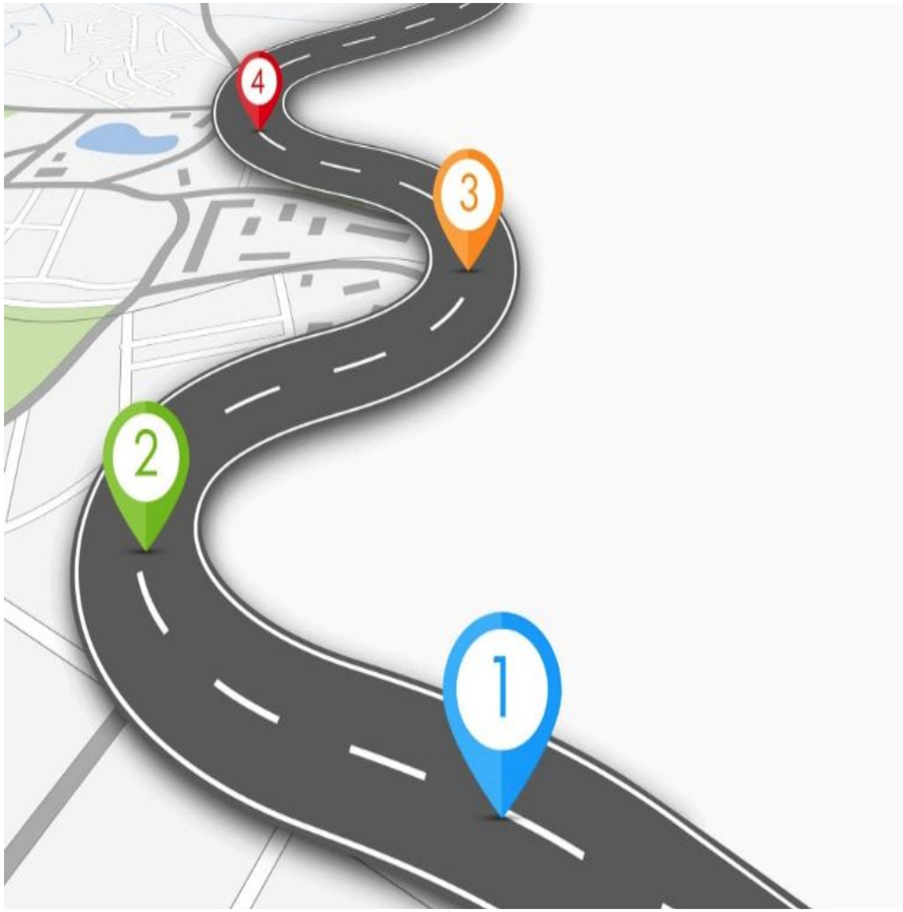
Roadmap

- Introduction
  - Effort Estimation
- Background
  - Hyperparameter Tuning
- Empirical Study
  - RATE
  - Dataset
  - Experiment Design
- Results
- Future Work



Roadmap

- Introduction
    - Effort Estimation
- Background
    - Hyperparameter Tuning
- Empirical Study
    - RATE
    - Dataset
    - Experiment Design
- Results
- Future Work



Introduction

What is effort estimation?

- Predicting human effort to plan and develop software projects.
- Based on the information collected in previous related software projects.
- Usually expressed in terms of hours, days or months of human work.

Introduction

Why effort estimation?

Can be used for

- Request for proposal
- Contract negotiation
- Resource allocation
- Monitoring and control

Introduction

Ways to estimate effort

- Human-based Methods [**Jørgensen' 2004**]

- Algorithm-based Methods
  - COCOMO [**Boehm' 1981**]
  - ATLM [**Whigham' 2015**]
  - LP4EE [**Sarro' 2018**]
  - Regression Trees (e.g. CART)
  - Analogy-Based Estimation (ABE)

Introduction

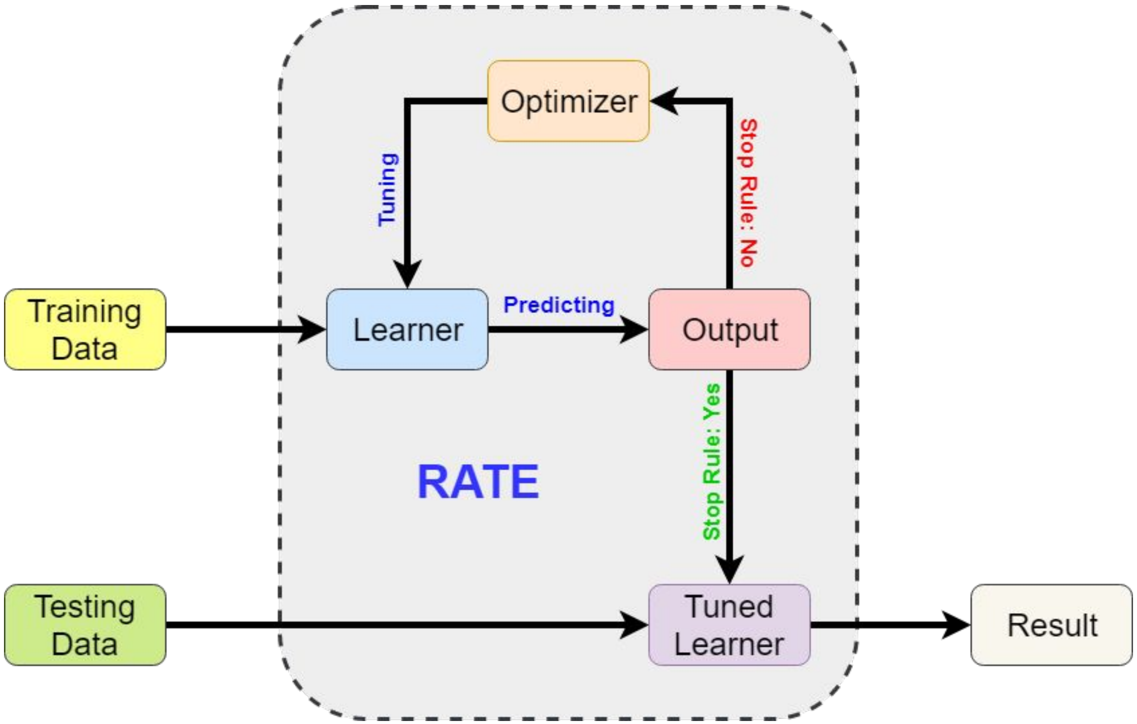
Being precise is critical

Inaccurate resource allocation can cause a considerable waste of resource and time.

- NASA canceled its Launch Control System project after the initial $200M estimate was exceeded by another $200M. [**Cowing' 2002**]

Introduction

How to improve the accuracy?



We propose RATE

(Rapid Automatic Tuning for Effort-estimation)

A **hyperparameter tuning** based method.

RATE will be discussed later in the talk.

Introduction

Performance improvement

Some SE communities (e.g. defect prediction) endorse hyperparameter tuning to improve the performance:

- For open source JAVA system datasets, tunings improved the defect detection precision from 0% to 60%. [**Fu' 2016**]
- For defect prediction models, tuning improves 40% of AUC performance. [**Tantithamthavorn' 2016**]

So why not use it on effort estimation?

Introduction

So why not effort estimation?

CPU intensive and running time?

- Arcuri et al reported their tuning may require weeks, or more, of CPU time. [**Arcuri' 2011**]
- Wang et al. needed 15 years of CPU to explore 9.3 million candidate configurations for software clone detectors. [**Wang' 2013**]

Introduction

So why not effort estimation?

Are these default hyperparameter values already good enough?

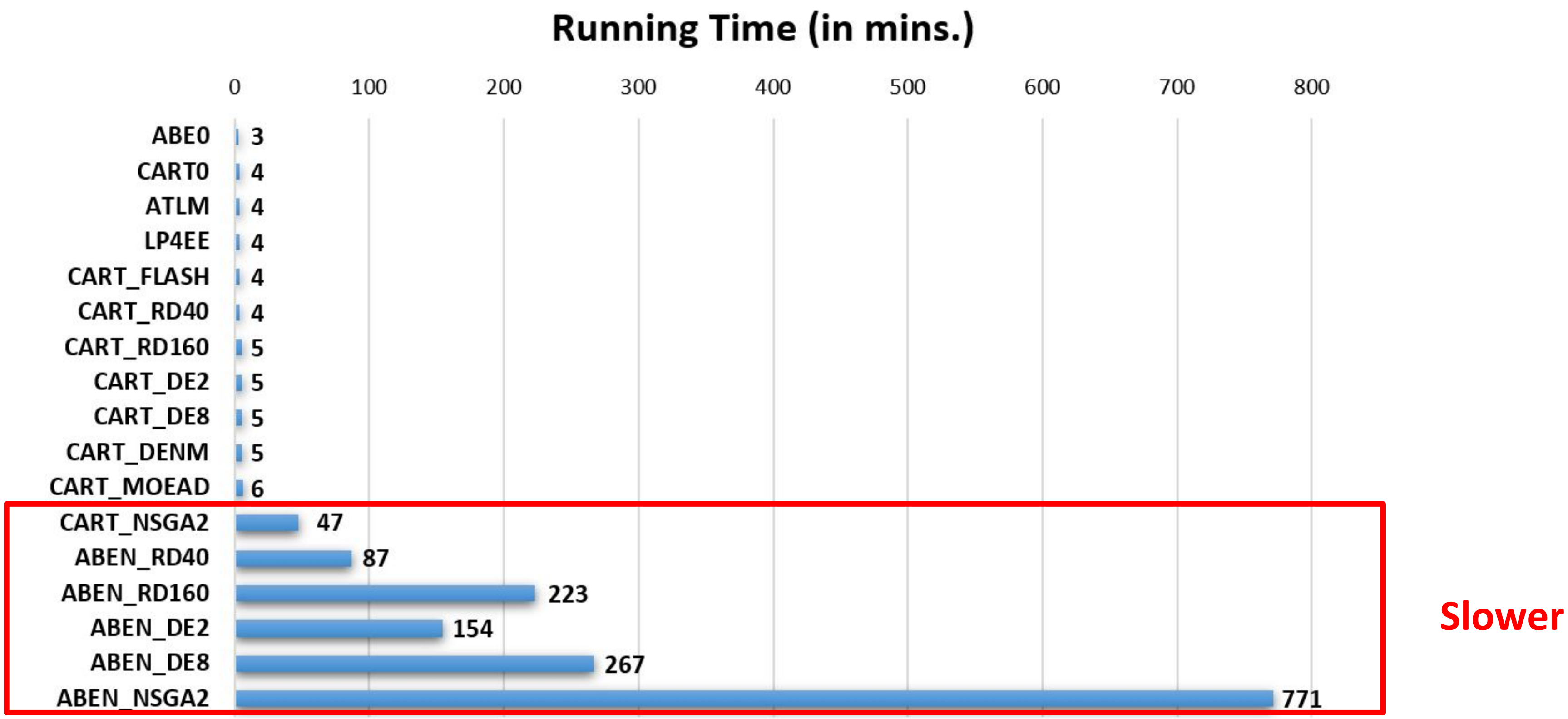
Introduction

Applying hyperparameter tuning to effort estimation

Open issues:

- **Is it best to just use "off-the-shelf" defaults?**
- **Can we replace the old default values with new values?**
- **Can we find what methods to use for effort estimation?**
- **Can we avoid the slow methods?**

Can we avoid slow hyperparameter optimization?



Introduction

Applying hyperparameter tuning to effort estimation

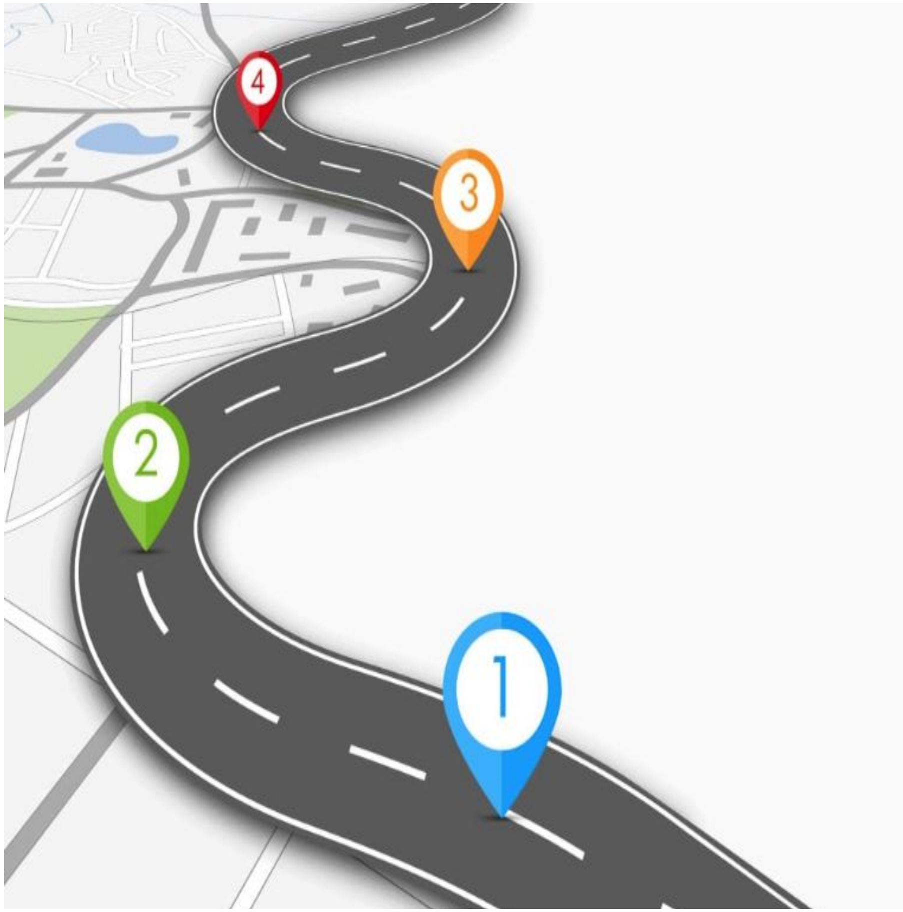
Open issues:

- **Is it best to just use “off-the-shelf” defaults?**
- **Can we replace the old defaults values with new values?**
- **Can we find what methods to use for effort estimation?**
- **Can we avoid the slow methods?**

We’ll get back to these questions later in the talk.

Roadmap

- **Introduction**
  - **Effort Estimation**
- **Background**
  - **Hyperparameter Tuning**
- **Empirical Study**
  - **RATE**
  - **Dataset**
  - **Experiment Design**
- **Results**
- **Future Work**

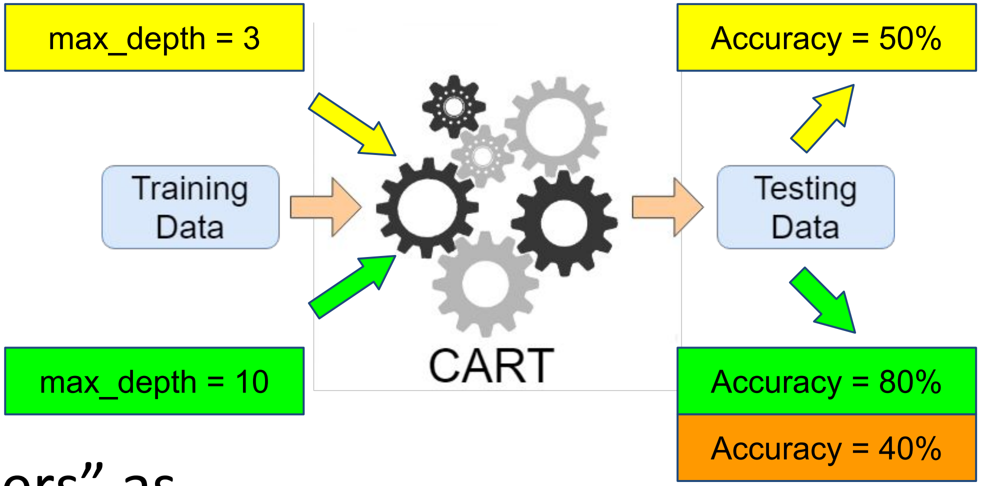


Background

What is hyperparameter?

Data mining algorithms usually have some “magic parameters ” to control their behavior.

- Decision Tree (CART):
  - max_depth
  - min_samples_leaf
  - ......



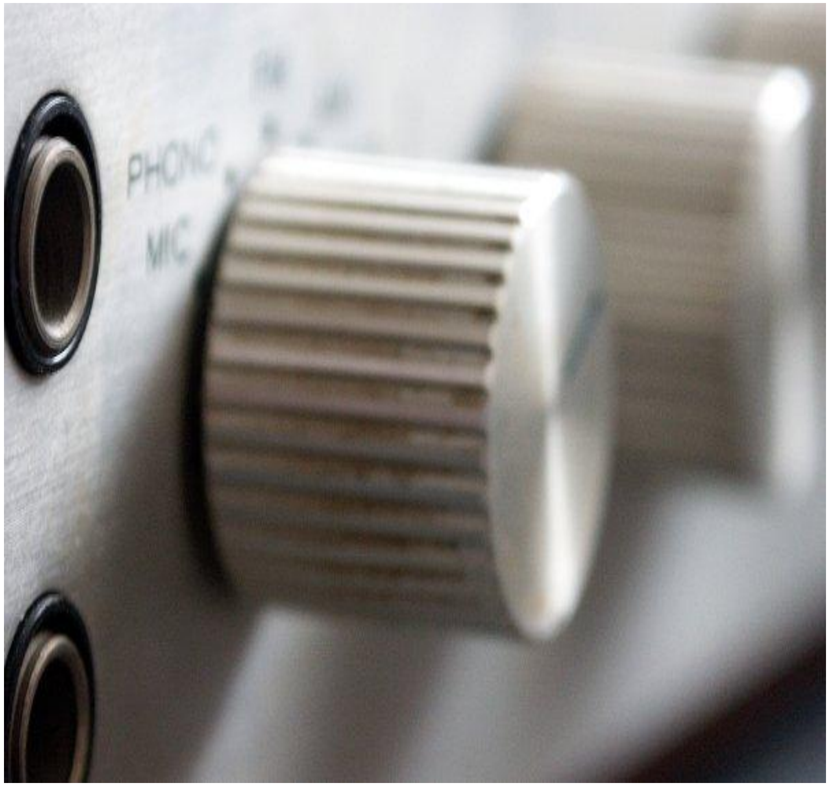
We call these “magic parameters” as hyperparameters.

Background

Tuning for Optimization

Finding suitable hyperparameters of data miners for better results.

When tuning, data miners will use different heuristics and generate different models.



Background

Ways to tune

- Mathematical Optimization
- Grid Search
- Evolutionary Algorithms
    - Differential Evolution [**Storn' 1997**]
    - NSGA-II [**Deb' 2002**]
    - MOEA/D [**Zhang' 2007**]

Background

Mathematical Optimization

Objective: $\min \quad f(x_1, x_2, x_3 \ldots)$

Constraints: $\text{s.t.} \quad x_i \leq b_i,\ i = 1, \ldots, m.$

Based on the property of objective function and constraint function:

- linear programing
- non-linear programing
- ……

Not used much in SE because of computational complexity issues. Sometime SE problem can't be modeled properly. [**Harman' 2001**]

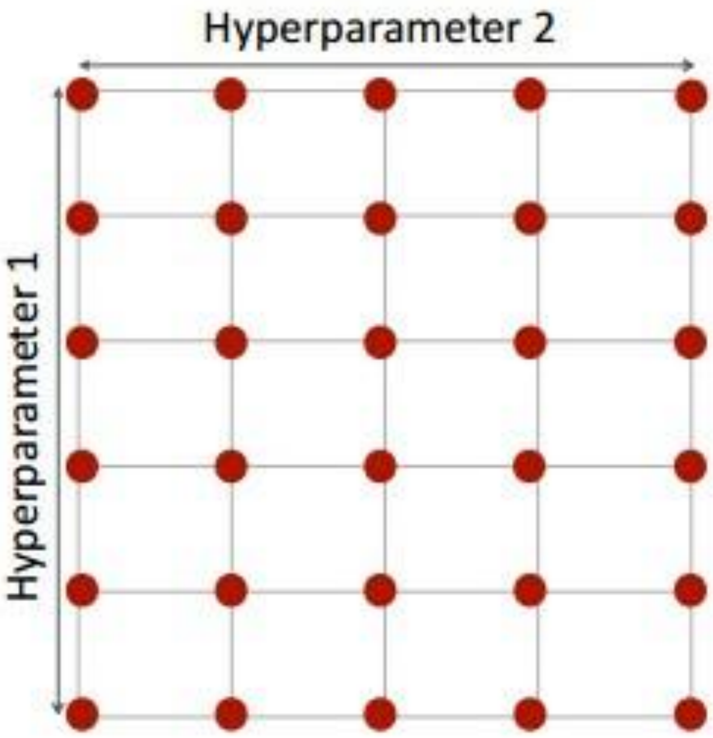
Background

Grid Search

- Evaluate every possible different hyperparameter combinations.

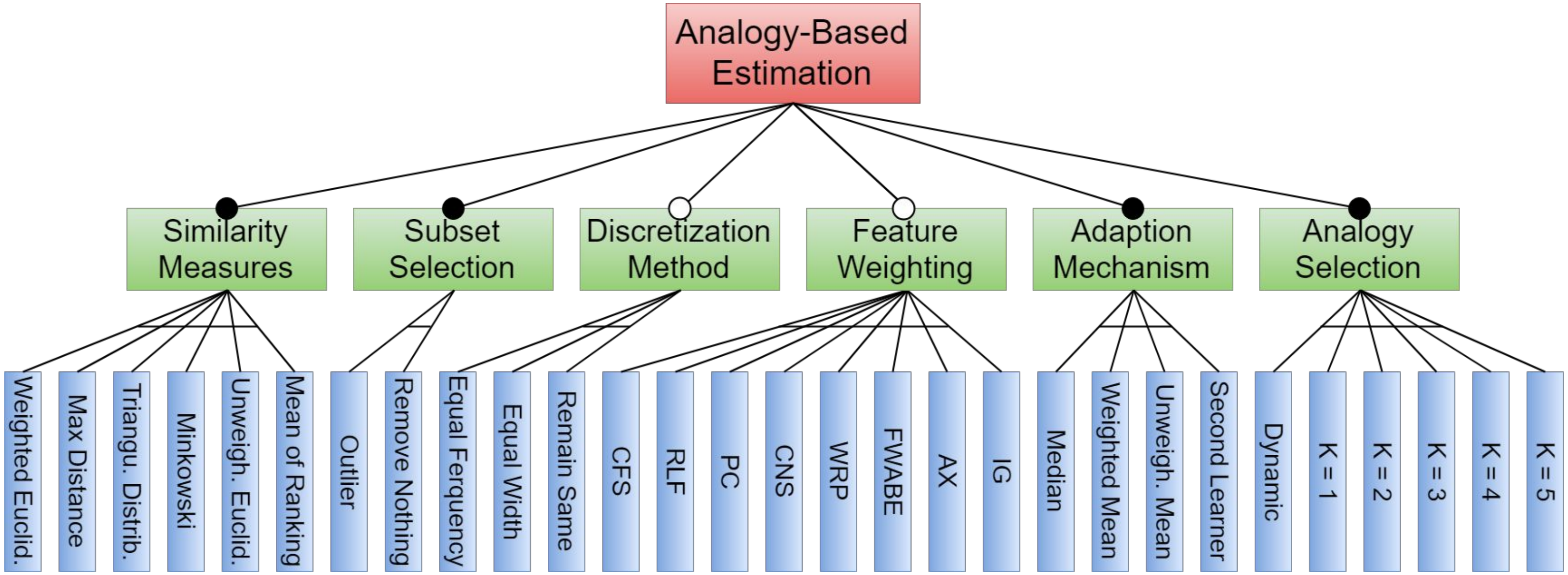
- Can be very slow!

  (Our Colleagues spent 27.5 days on grid search in their experiment) [Fu' 2016]



Background

Grid Search



For ABEN methods, there are

6 * 2 * 3 * 8 * 4 * 6 = 6,912 different variants!

Background

Differential Evolution (DE)

Population = Pick N options at random # e.g. N =20

M times repeat : # e.g. M = 8

for Parent in Population:

- Select a, b, c = three other items in population.
- Candidate = a + f * (b - c)
- if Candidate is “better”, replace Parent.

Background

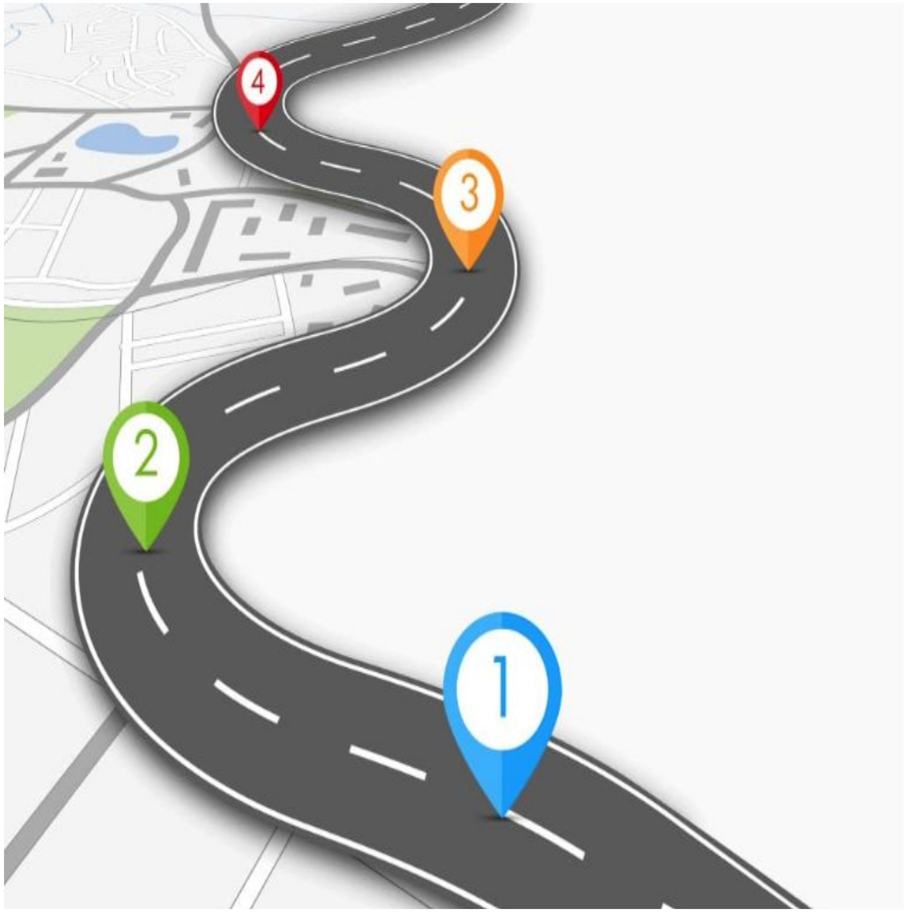
FLASH [Nair’ 2017]

```
Pick a number of data into build_pool, evaluate the build_pool, and put
the rest into rest_pool

while life > 0:
    build CART model by using build_pool
    next_point = max( model.predict( rest_pool ) )
    build_pool += next_point
    rest_pool -= next_point
    if model.evaluate( next_point ) < max( build_pool ):
        life -= 1
return max( build_pool )
```

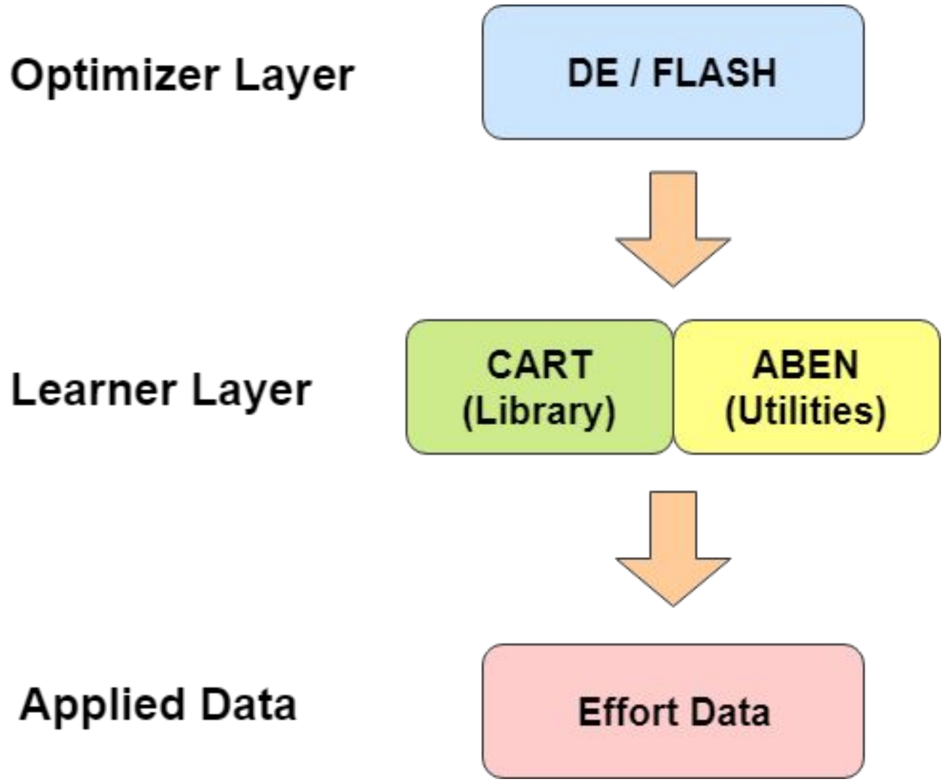
Roadmap

- Introduction
  - Effort Estimation
- Background
  - Hyperparameter Tuning
- Empirical Study
  - RATE
  - Dataset
  - Experiment Design
- Results
- Future Work



Empirical Study

RATE Architecture



Compare with non-tuning methods:

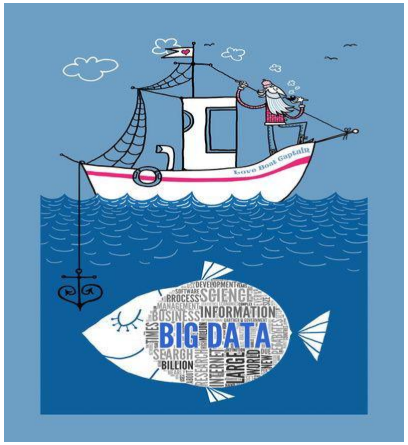
- ABE0
- CART0
- ATLM
- LP4EE

Empirical Study

Software effort dataset

SEACRAFT

A widely used research repository for SE datasets

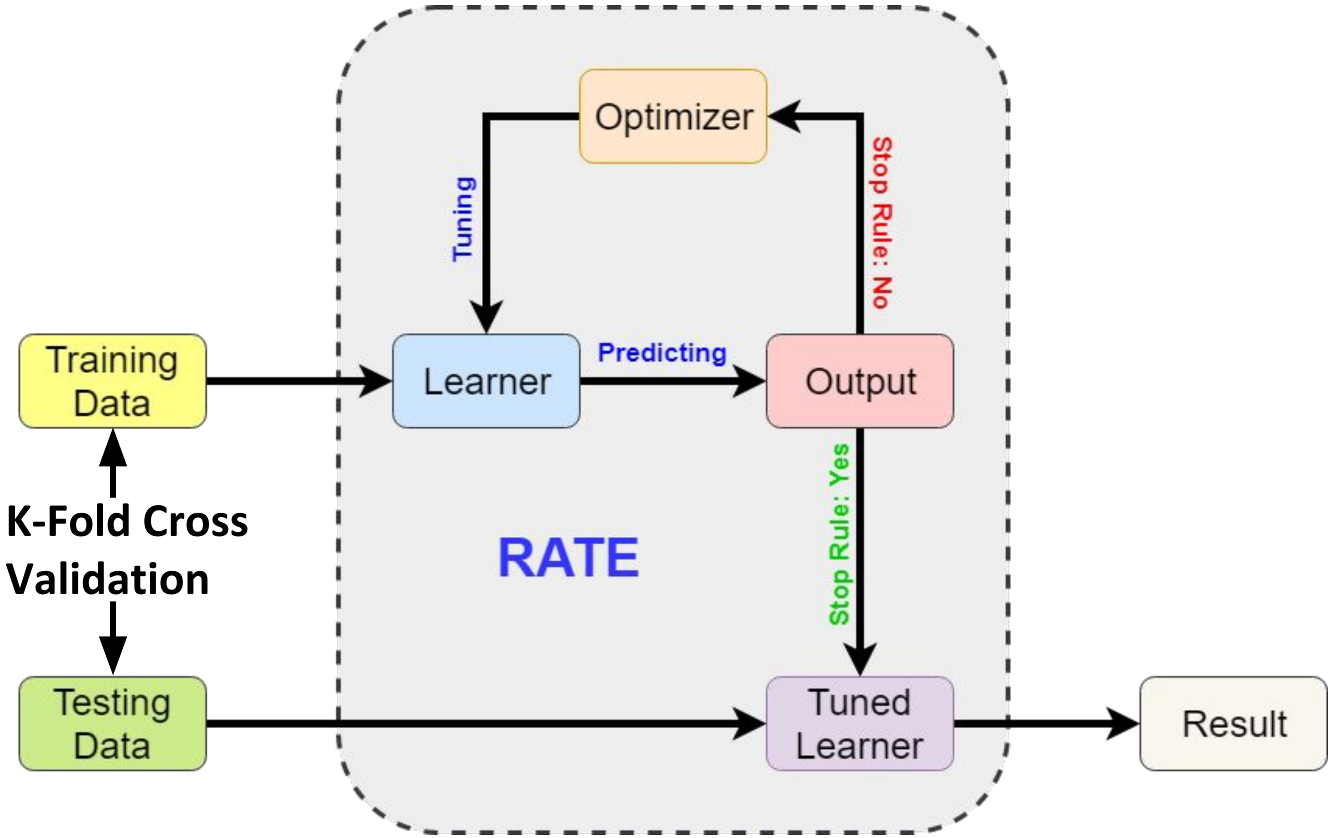


	feature	min	max	mean	std
desharnais	TeamExp	0	4	2.3	1.3
	MngExp	0	7	2.6	1.5
	Length	1	36	11.3	6.8
	Trans.s	9	886	177.5	146.1
	Entities	7	387	120.5	86.1
	AdjPts	73	1127	298.0	182.3
	Effort	546	23940	4834	4188

Datasets	#Projects	#Features
kemerer	15	6
albrecht	24	7
isbsg10	37	11
finnish	38	7
miyazaki	48	7
maxwell	62	25
desharnais	77	6
kitchenham	145	6
china	499	16
Total#	945	

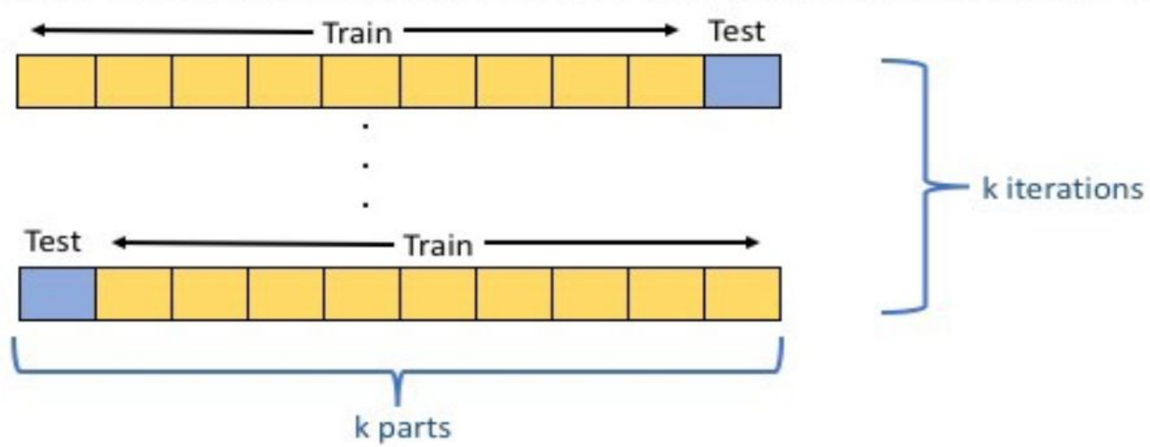
Empirical Study

Experiment design



Empirical Study

K-Folds Cross Validation



1. Divide the sample data into k parts.
2. Use k-1 of the parts for training, and 1 for testing.
3. Repeat the procedure k times, rotating the test set.
4. Determine an expected performance metric based on the results across the iterations.

Empirical Study

Performance metrics

Magnitude of Relative Error (MRE) [**Conte' 1986**]

$$\text{MRE} = \left(\frac{|\,Actual - Predicted\,|}{Actual} \right) \times 100$$

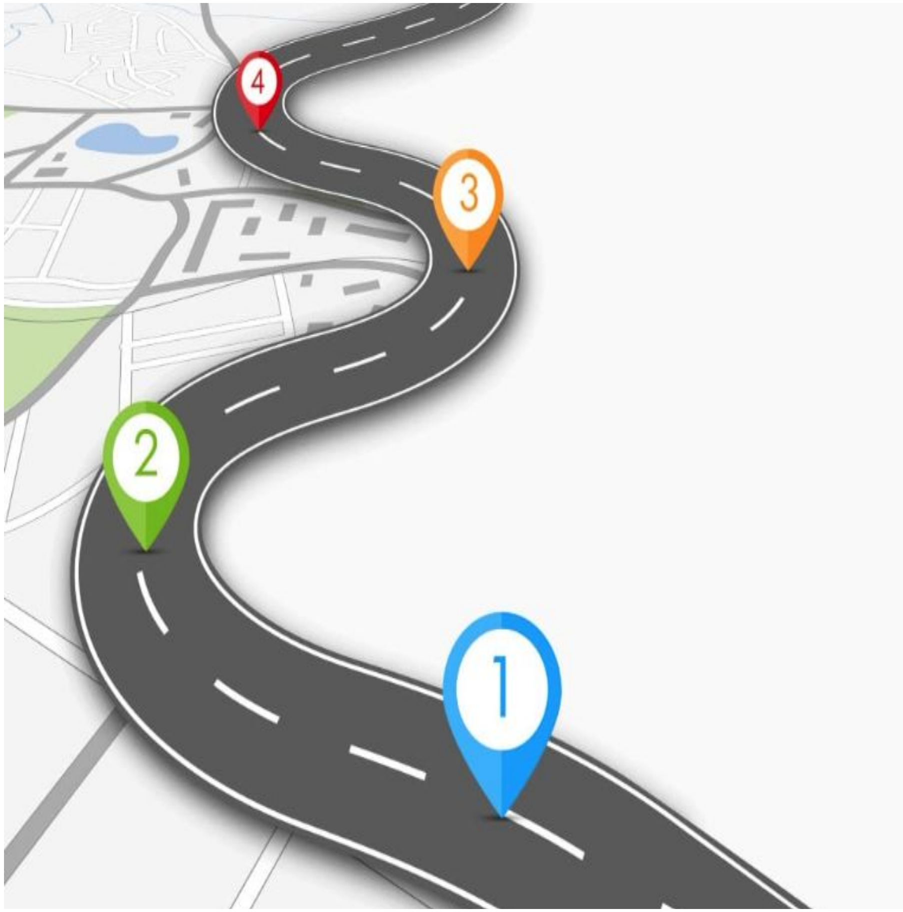
Standardized Accuracy (SA) [**Shepperd' 2012**]

$$\text{SA} = \left(1 - \frac{|\,Actual - Predicted\,|}{mean(\,Actual\,)} \right) \times 100$$

Note: MRE: Lower the better; SA: Higher the better.

Roadmap

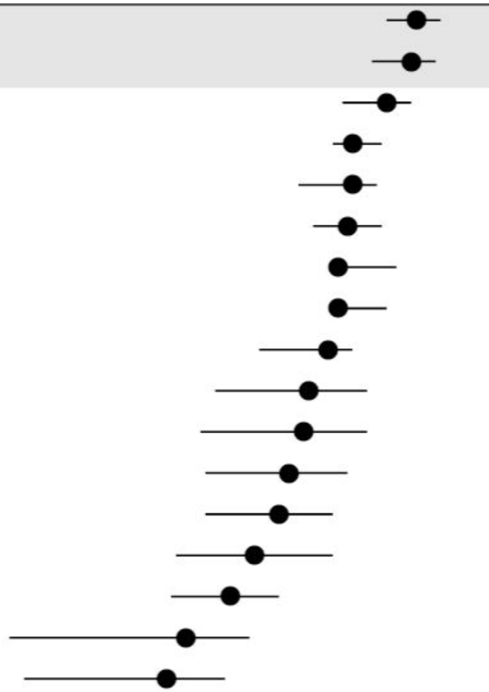
- Introduction
  - Effort Estimation
- Background
  - Hyperparameter Tuning
- Empirical Study
  - RATE
  - Dataset
  - Experiment Design
- Results
- Future Work



Results

Performance ranking (SA scores)

Rank	Methods	Med	IQR
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_RD160	74	16
2	CART_NSGA2	73	14
2	CART_RD40	71	12
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABEN_DE2	65	31
3	ABEN_DE8	64	34
3	ABEN_RD160	61	29
3	ABEN_RD40	59	26
3	ABEN_NSGA2	54	32
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41

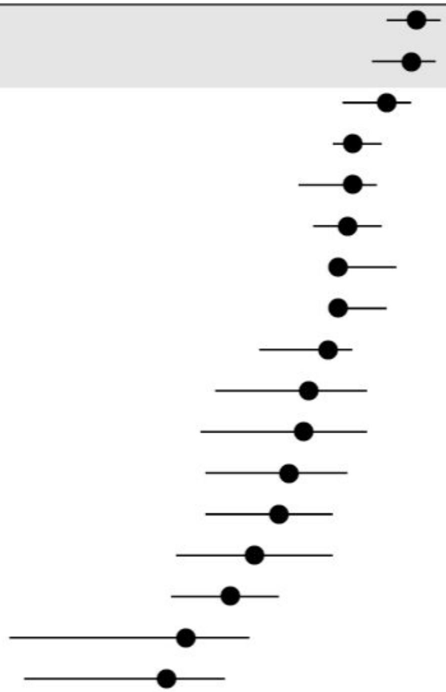


Example of SA scores for finnish dataset, using 10-way cross validation, 20 times experiment repeats. Methods recommended by [**Mittas' 2013**].

Results

Performance ranking (SA scores)

Rank	Methods	Med	IQR
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_RD160	74	16
2	CART_NSGA2	73	14
2	CART_RD40	71	12
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABEN_DE2	65	31
3	ABEN_DE8	64	34
3	ABEN_RD160	61	29
3	ABEN_RD40	59	26
3	ABEN_NSGA2	54	32
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41



Results

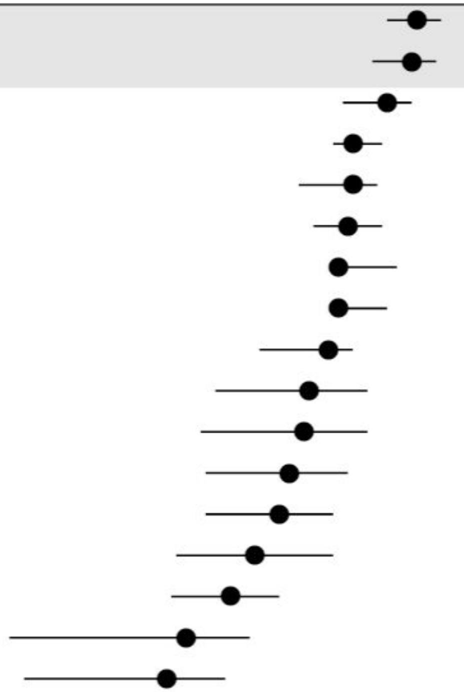
Performance ranking (SA scores)

Rank	Methods	Med	IQR
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_RD160	74	16
2	CART_NSGA2	73	14
2	CART_RD40	71	12
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABEN_DE2	65	31
3	ABEN_DE8	64	34
3	ABEN_RD160	61	29
3	ABEN_RD40	59	26
3	ABEN_NSGA2	54	32
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41

Results

Performance ranking (SA scores)

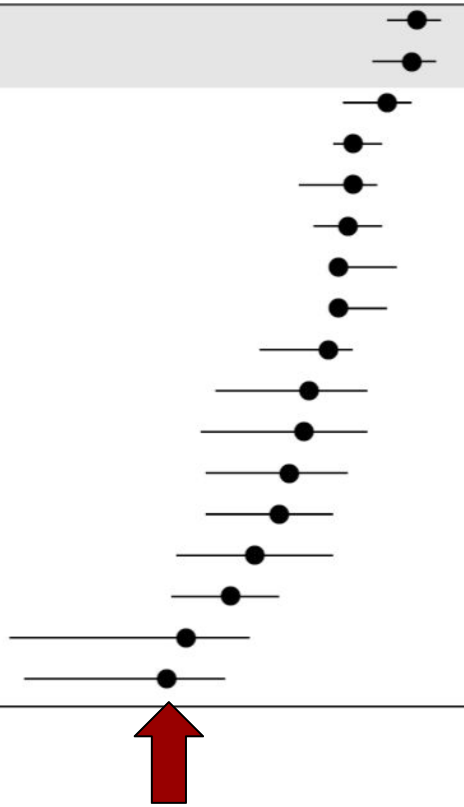
Rank	Methods	Med	IQR
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_RD160	74	16
2	CART_NSGA2	73	14
2	CART_RD40	71	12
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABEN_DE2	65	31
3	ABEN_DE8	64	34
3	ABEN_RD160	61	29
3	ABEN_RD40	59	26
3	ABEN_NSGA2	54	32
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41



Results

Performance ranking (SA scores)

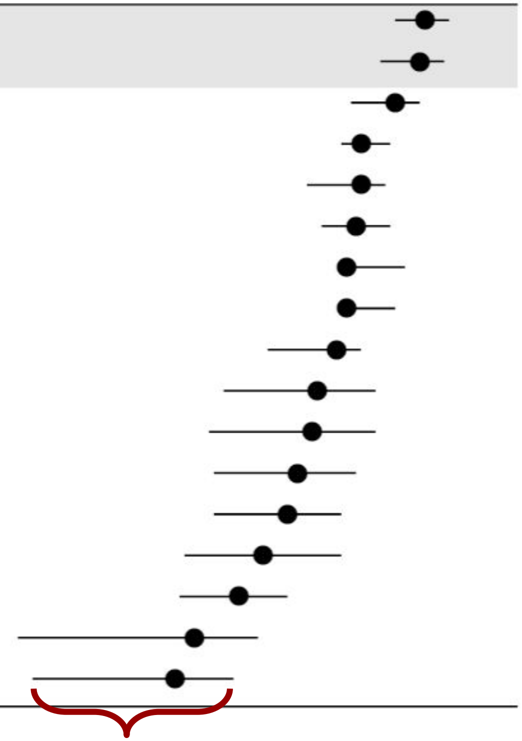
Rank	Methods	Med	IQR
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_RD160	74	16
2	CART_NSGA2	73	14
2	CART_RD40	71	12
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABEN_DE2	65	31
3	ABEN_DE8	64	34
3	ABEN_RD160	61	29
3	ABEN_RD40	59	26
3	ABEN_NSGA2	54	32
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41



Results

Performance ranking (SA scores)

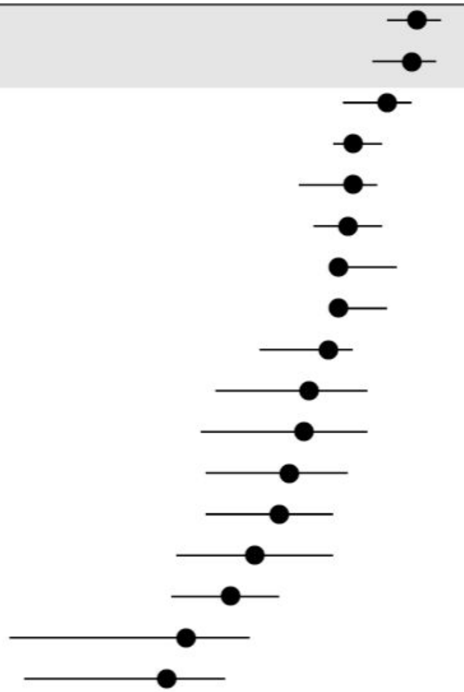
Rank	Methods	Med	IQR
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_RD160	74	16
2	CART_NSGA2	73	14
2	CART_RD40	71	12
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABEN_DE2	65	31
3	ABEN_DE8	64	34
3	ABEN_RD160	61	29
3	ABEN_RD40	59	26
3	ABEN_NSGA2	54	32
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41



Results

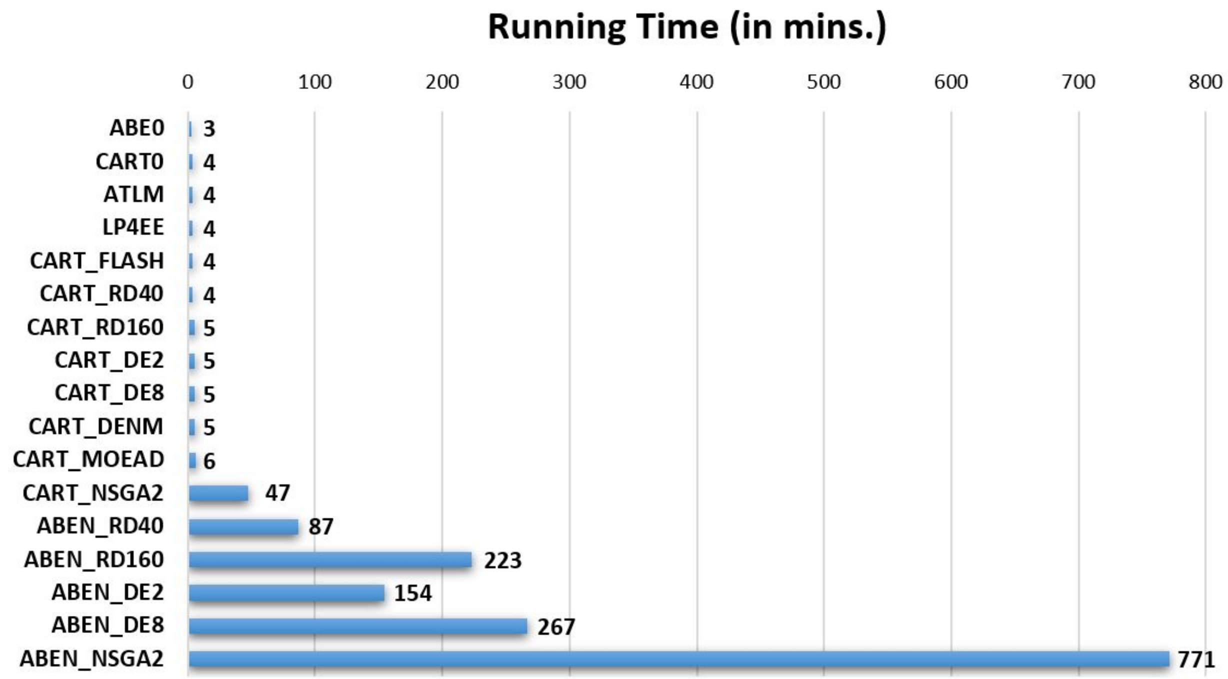
Performance ranking (SA scores)

Rank	Methods	Med	IQR
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_RD160	74	16
2	CART_NSGA2	73	14
2	CART_RD40	71	12
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABEN_DE2	65	31
3	ABEN_DE8	64	34
3	ABEN_RD160	61	29
3	ABEN_RD40	59	26
3	ABEN_NSGA2	54	32
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41



Results

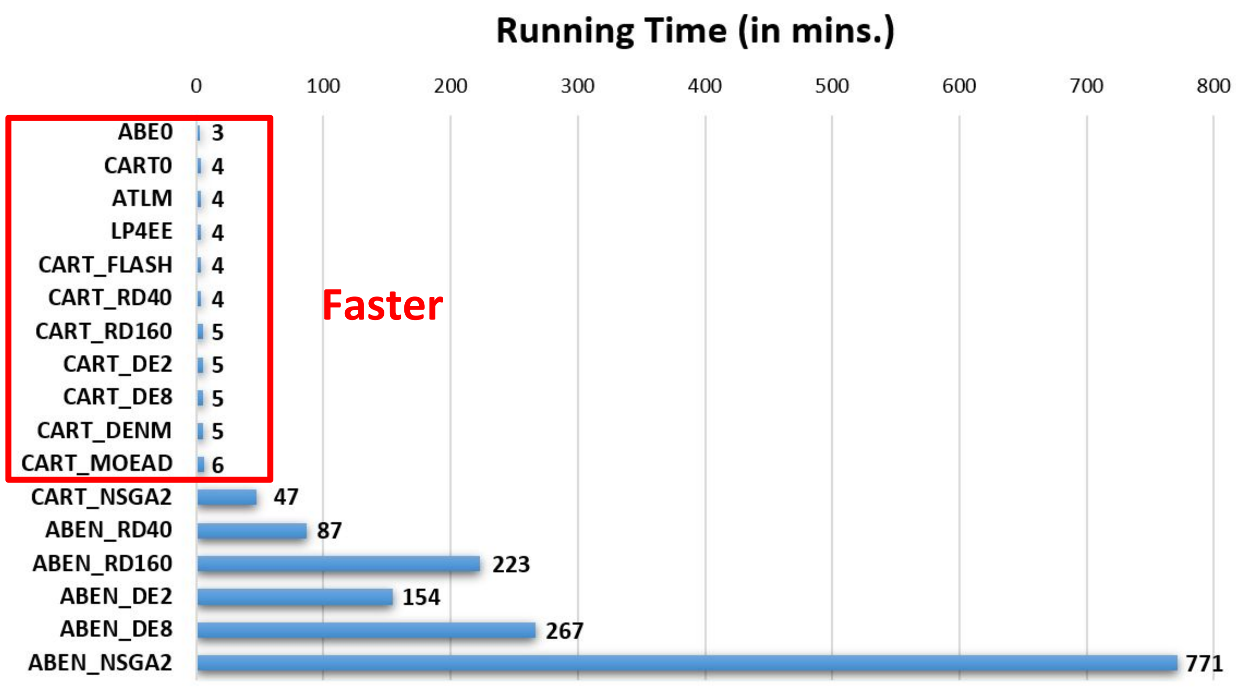
Running time



Average runtime (in minutes), for one-way out of an N*M cross-validation experiment, in a local machine.

Results

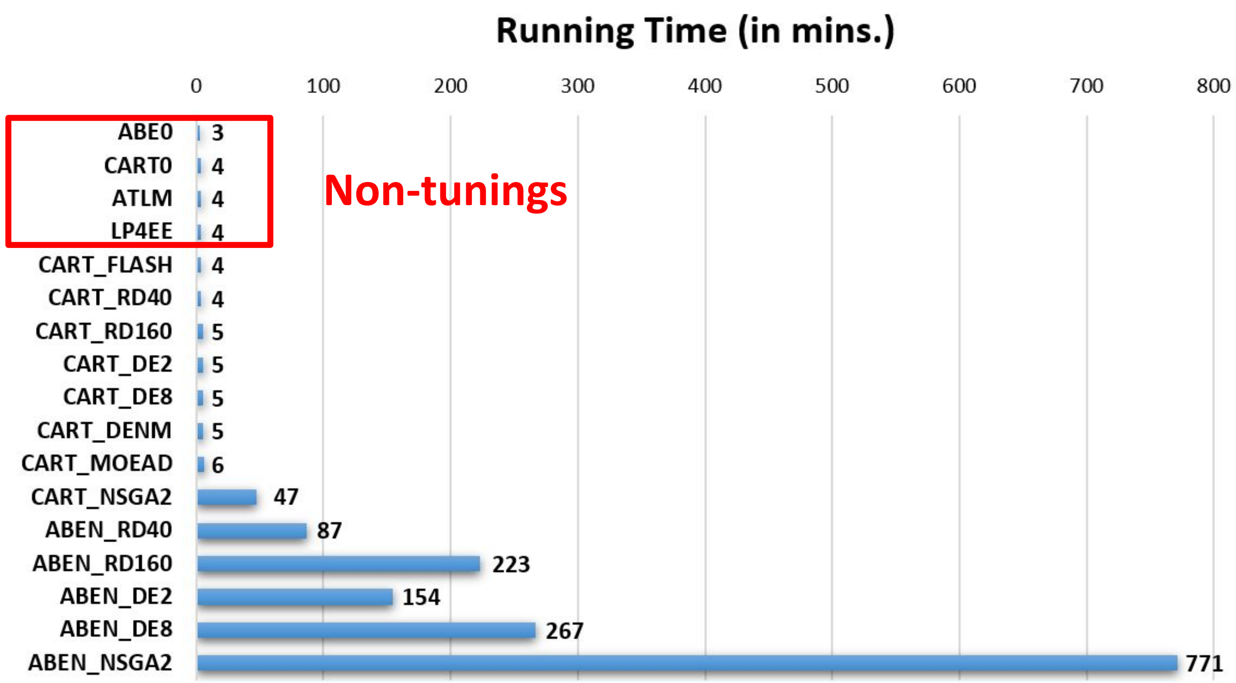
Running time



Average runtime (in minutes), for one-way out of an N*M cross-validation experiment, in a local machine.

Results

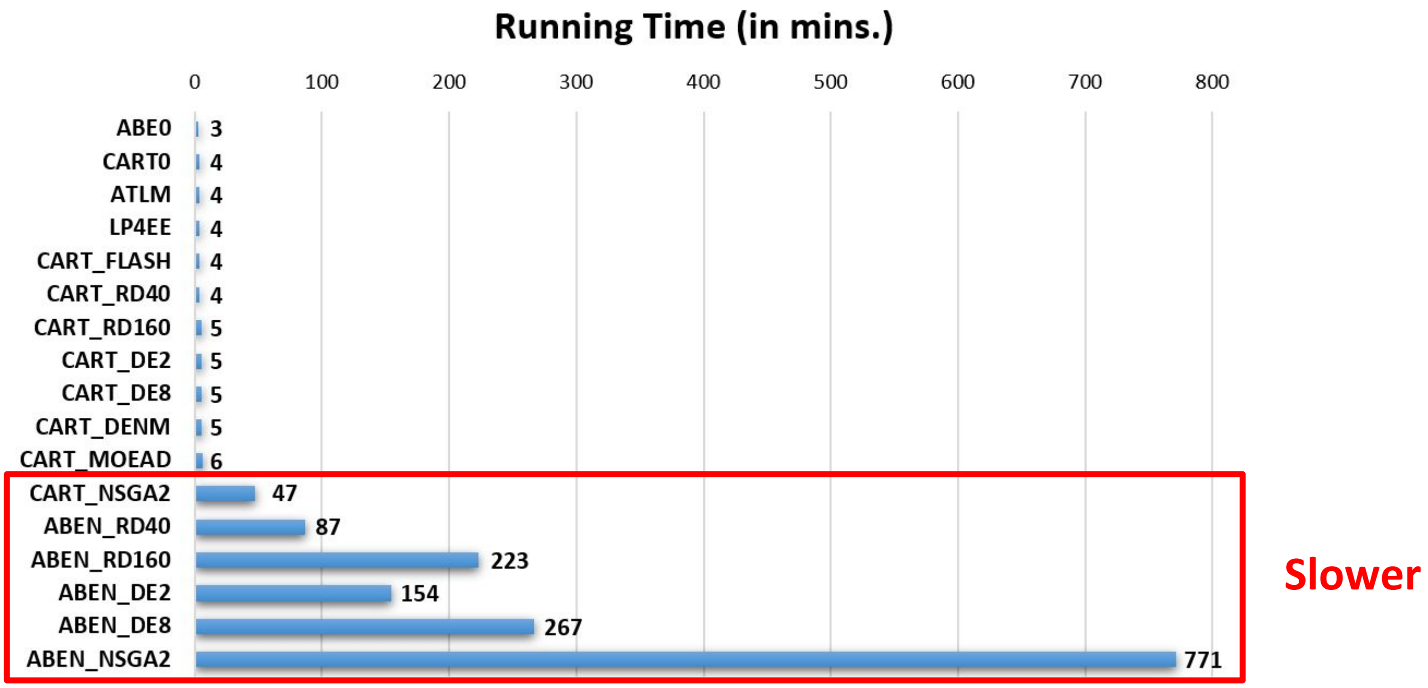
Running time



Average runtime (in minutes), for one-way out of an N*M cross-validation experiment, in a local machine.

Results

Running time



Average runtime (in minutes), for one-way out of an N*M cross-validation experiment, in a local machine.

Results

Performance ranking (MRE scores)

Rank	Methods	Med	IQR
1*	CART_FLASH	46	21
1	ABEN_RD160	48	29
1	ABEN_DE2	48	20
1*	ABE0	51	28
1	ABEN_DE8	51	30
1*	CART_DENM	51	33
1*	CART_DE8	52	28
2	CART_DE2	56	23
2	CART_MOEAD	56	23
2	CART0	59	21

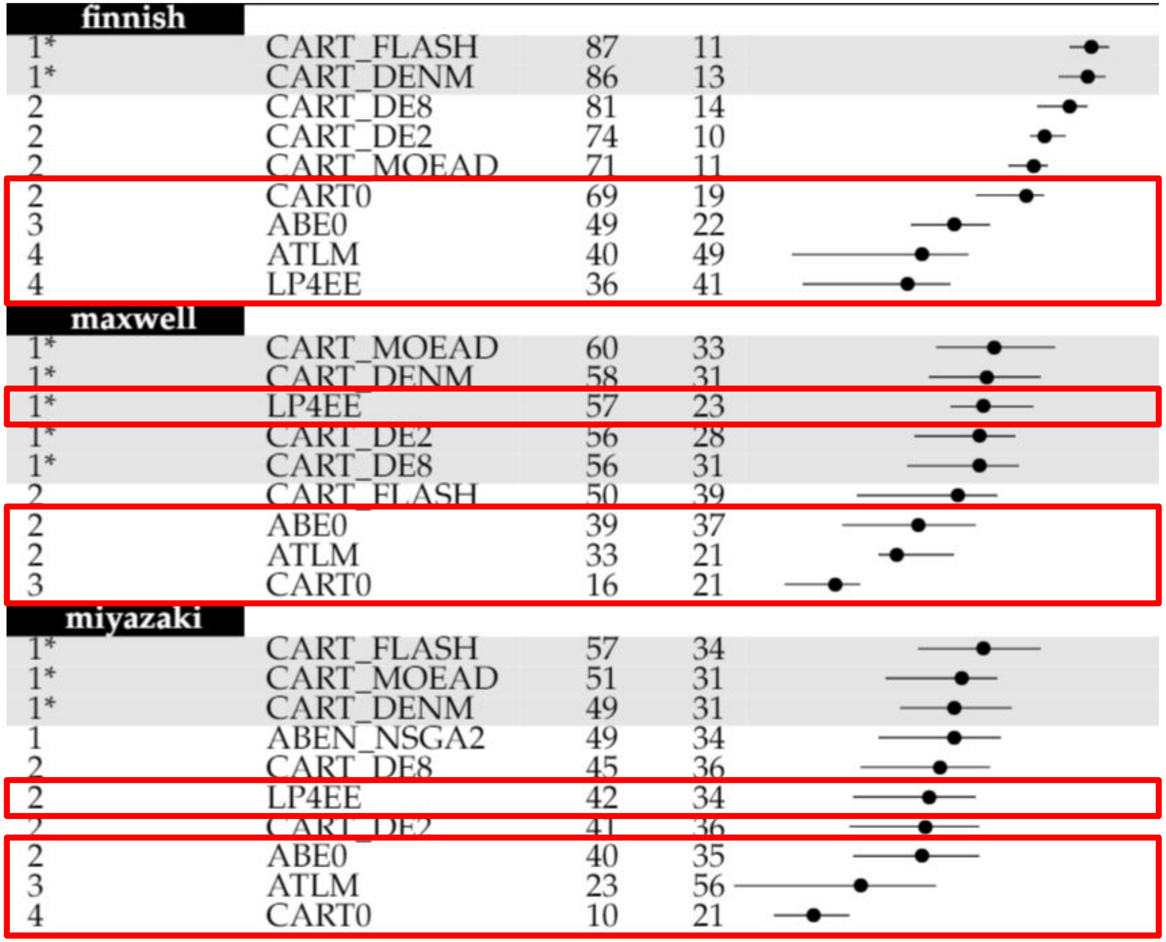
Research Questions

- Is it best to just use “off-the-shelf” defaults?
- Can we replace the old default values with new values?
- Can we find what methods to use for effort estimation?
- Can we avoid the slow methods?

Research Questions

- **Is it best to just use “off-the-shelf” defaults?**
- Can we replace the old default values with new values?
- Can we find what methods to use for effort estimation?
- Can we avoid the slow methods?

Is it best to just use "off-the-shelf" defaults?



finnish			
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41
maxwell			
1*	CART_MOEAD	60	33
1*	CART_DENM	58	31
1*	LP4EE	57	23
1*	CART_DE2	56	28
1*	CART_DE8	56	31
2	CART_FLASH	50	39
2	ABE0	39	37
2	ATLM	33	21
3	CART0	16	21
miyazaki			
1*	CART_FLASH	57	34
1*	CART_MOEAD	51	31
1*	CART_DENM	49	31
1	ABEN_NSGA2	49	34
2	CART_DE8	45	36
2	LP4EE	42	34
2	CART_DE2	41	36
2	ABE0	40	35
3	ATLM	23	56
4	CART0	10	21

"Off-the-shelf" defaults rarely have good performance.

Using just default values should be deprecated.

Research Questions

- **Is it best to just use "off-the-shelf" defaults?**

NO

- Can we replace the old default values with new values?
- Can we find what methods to use for effort estimation?
- Can we avoid the slow methods?

Research Questions

- **Is it best to just use "off-the-shelf" defaults?**

NO

- **Can we replace the old default values with new values?**
- Can we find what methods to use for effort estimation?
- Can we avoid the slow methods?

Can we replace the old defaults with new values?

Learner: CART

Data Set: desharnais

Optimizer: Differential Evolution

Hyperparameter: min_samples_leaf

Value Range	**(0, 3]**	**(3, 6]**	**(6, 9]**	**(9, 12]**
Number of case	**63**	**52**	**48**	**37**

An example of hyperparameter selections for CART_DE8,
using 10-way cross validation, 20 times experiment repeats

There are no “best” default settings.

Research Questions

- **Is it best to just use "off-the-shelf" defaults?**

NO

- **Can we replace the old default values with new values?**

NO

- Can we find what methods to use for effort estimation?
- Can we avoid the slow methods?

Research Questions

- **Is it best to just use "off-the-shelf" defaults?**

NO

- **Can we replace the old default values with new values?**

NO

- **Can we find what methods to use for effort estimation?**
- Can we avoid the slow methods?

Can we find what methods to use for effort estimation?

In all 18 experiment cases (9 datasets, 2 metrics for each),

If we use only one method, after counting the frequencies in the top-ranks (Rank as “1*”), then we get:

9/18: CART_DE2, CART_DENM

10/18: CART_DE8

12/18: CART_FLASH

Can we find what methods to use for effort estimation?

In all 18 experiment cases (9 datasets, 2 metrics for each),

If we use two methods combinations, then we get:

16/18: CART_DE2 + CART_FLASH

16/18: CART_DE8 + CART_FLASH

Can we find what methods to use for effort estimation?

In all 18 experiment cases (9 datasets, 2 metrics for each),

And if we use three methods combinations, then:

17/18: ABE0 + CART_DE2 + CART_FLASH

17/18: ABE0 + CART_DE8 + CART_FLASH

17/18: ATLM + CART_DE2 + CART_FLASH

17/18: ATLM + CART_DE8 + CART_FLASH

17/18: CART_MOEAD + CART_FLASH + CART_DE2

17/18: CART_NSGA2 + CART_FLASH + CART_DE2

17/18: CART_MOEAD + CART_FLASH + CART_DE8

17/18: CART_NSGA2 + CART_FLASH + CART_DE8

Can we find what methods to use for effort estimation?

- To simplify the implementation of our methods, we will avoid CART_DE8, MOEA/D and NSGA-II.
- Since 2-methods combo have similar performance as 3-methods combo (16/18 vs 17/18) with faster running time, the recommended choice will be:

CART_DE2 + CART_FLASH

Hence, for effort datasets, try a combination of CART with the optimizers Differential Evolution and FLASH.

Research Questions

- **Is it best to just use "off-the-shelf" defaults?** NO
- **Can we replace the old default values with new values?** NO
- **Can we find what methods to use for effort estimation?** YES
- Can we avoid the slow methods?

Research Questions

- **Is it best to just use "off-the-shelf" defaults?**

  NO

- **Can we replace the old default values with new values?**

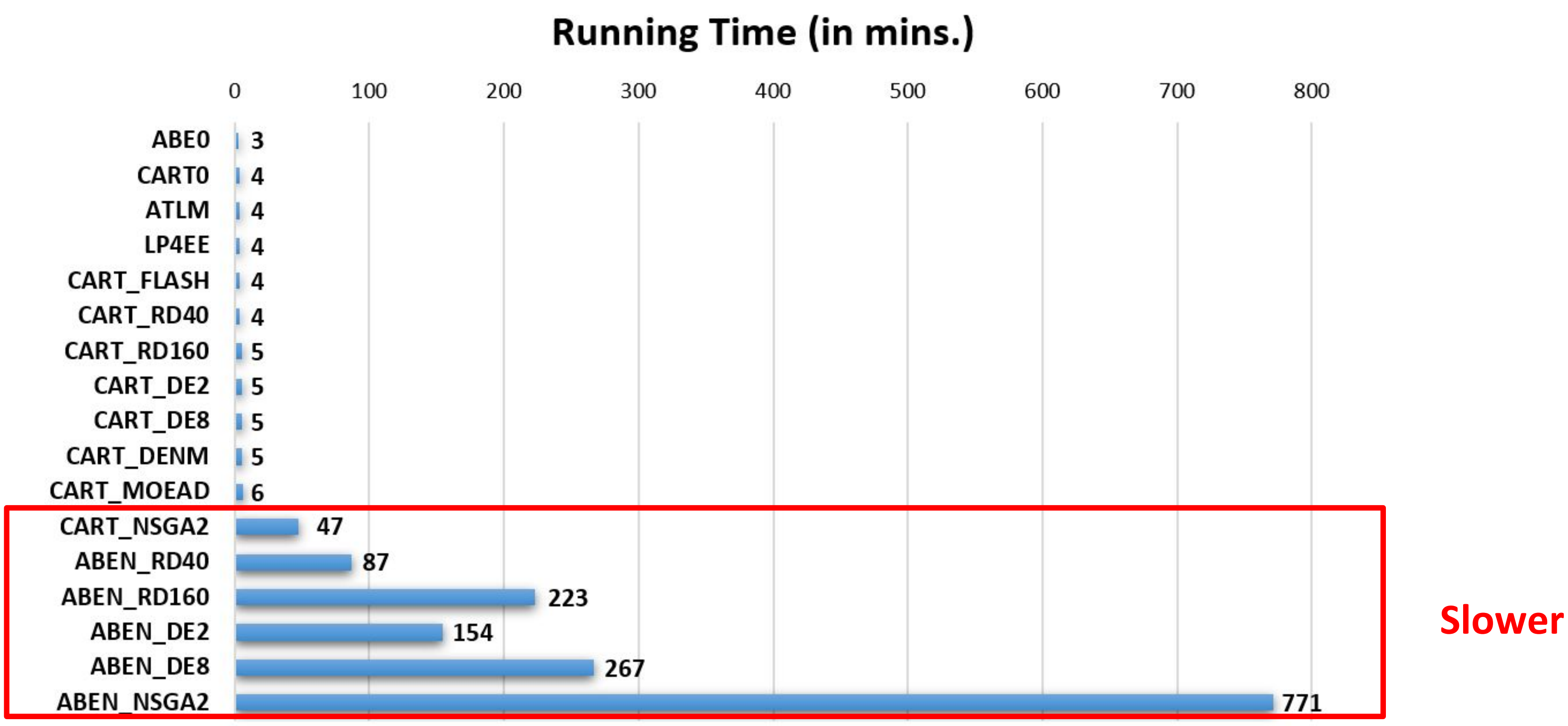
  NO

- **Can we find what methods to use for effort estimation?**

  YES

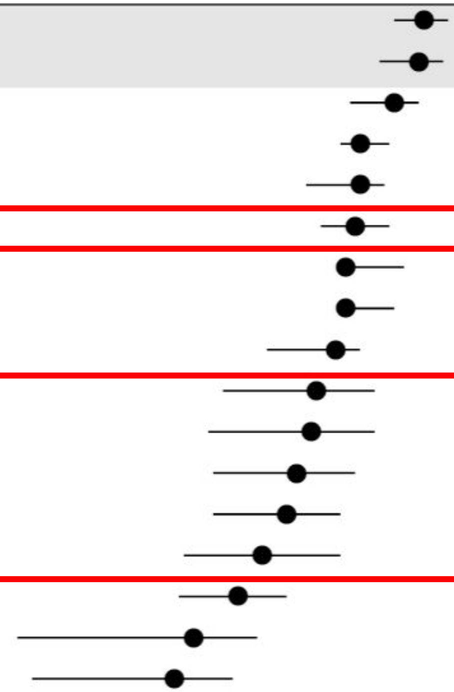
- **Can we avoid the slow methods?**

Can we avoid slow methods?



Can we avoid slow methods?

Rank	Methods	Med	IQR
1*	CART_FLASH	87	11
1*	CART_DENM	86	13
2	CART_DE8	81	14
2	CART_DE2	74	10
2	CART_RD160	74	16
2	CART_NSGA2	73	14
2	CART_RD40	71	12
2	CART_MOEAD	71	11
2	CART0	69	19
3	ABEN_DE2	65	31
3	ABEN_DE8	64	34
3	ABEN_RD160	61	29
3	ABEN_RD40	59	26
3	ABEN_NSGA2	54	32
3	ABE0	49	22
4	ATLM	40	49
4	LP4EE	36	41



Overall, our slowest optimizers perform no better than certain faster ones.

In all cases (9 datasets * 2 metrics), slower ones only win (Rank as “1*”) in 1/18.

Research Questions

- **Is it best to just use "off-the-shelf" defaults?**
  NO
- **Can we replace the old defaults with new defaults?**
  NO
- **Can we find what methods to use for effort estimation?**
  YES
- **Can we avoid the slow methods?**
  YES

Contributions

- **A demonstration that default settings are not the best way to perform effort estimation.**
- **A recognition of the inherent difficulty associated with effort estimation.**
- **A new criterion for assessing effort estimators.**
- **The identification of a combination of learner and optimizer that works good.**
- **An extensible open-source architecture.**

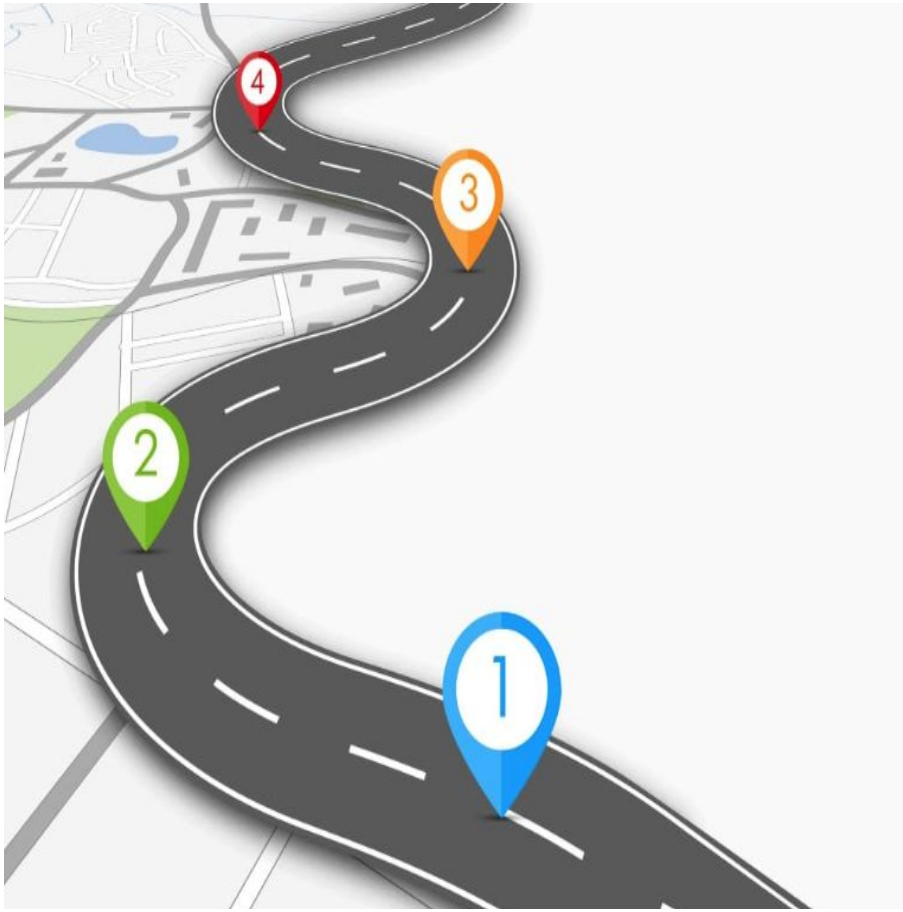
Submission

- A part of my work has been submitted to Empirical Software Engineering (EMSE)
  https://arxiv.org/pdf/1805.00336.pdf

- Open source tool is available on:
  https://github.com/ai-se/magic101

Roadmap

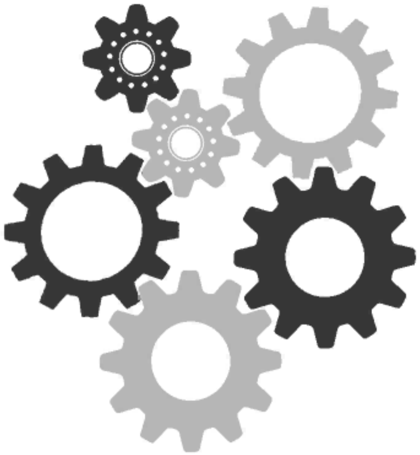
- Introduction
  - Effort Estimation
- Background
  - Hyperparameter Tuning
- Empirical Study
  - RATE
  - Dataset
  - Experiment Design
- Results
- Future Work



Future Work

More Exploration on RATE

- Better Optimizers
- Better Learners



Future Work

New Data Collections

- Github
- Stack Overflow
- ……

Future Work

Other Domains

In a 2018 TSE paper [**Huang' 2018**]:

IEEE TRANSACTIONS ON
SOFTWARE ENGINEERING

Automating Intention Mining

TABLE 11
The accuracy achieved by different approaches using the email and issue report dataset.

Approaches	Email	Email to Issue	Issue to Email
Ours	**0.791**	**0.579**	**0.658**
CNN	0.710	0.522	0.569
NLP	0.575	0.460	0.483
SMO	0.707	0.370	0.450
LibSVM	0.273	0.231	0.173
NBM	0.607	0.349	0.347
RF	0.693	0.329	0.383
kNN	0.604	0.239	0.370
DE-SVM	0.714	0.374	0.402
Auto-Weka	0.707	0.349	0.329

TABLE 12
Training time cost of each approach under different experiment setting

Approach	Issue	Email	Issue to Email	Email to Issue
Our (with BN)	30min	6min	38min	7min
Our (without BN)	5.6h	1h	7.2h	1.2h
Kim's CNN	20min	4min	24min	5min
NLP	4min	1min	6min	1min
SMO	11s	2s	12s	3s
LibSVM	9s	1s	10s	2s
NBM	1s	1s	1s	1s
RF	3min	1min	4min	1min
kNN	1s	1s	1s	1s
DE-SVM	20min	10min	16min	11min
Auto-Weka	16min	15min	15min	15min

Future Work

Beyond Hyperparameter

- The optimizers like DE or FLASH in RATE also have their own magic parameters

- Hyper-hyperparameter optimization could be a challenge.

Thank You!

Reference

[Storn’ 1997] Storn, R. and Price, K., 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*(4), pp.341-359.

[Nair’ 2017] Nair, V., Yu, Z. and Menzies, T., 2017. Flash: A faster optimizer for sbse tasks. *arXiv preprint arXiv:1705.05018*.

[Jørgensen’ 2004] Jørgensen, M., 2004. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, *70*(1-2), pp.37-60.

[Boehm’ 1981] Boehm, B.W., 1981. *Software engineering economics* (Vol. 197). Englewood Cliffs (NJ): Prentice-hall.

[Whigham’ 2015] Whigham, P.A., Owen, C.A. and Macdonell, S.G., 2015. A baseline model for software effort estimation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *24*(3), p.20.

[Sarro’ 2018] Sarro, F. and Petrozziello, A., 2018. Linear Programming as a Baseline for Software Effort Estimation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *27*(3), p.12.

[Cowing’ 2002] Cowing, K., 2002. NASA to Shut Down Checkout & Launch Control System. SpaceRef.

[Fu’ 2016] Fu, W., Menzies, T. and Shen, X., 2016. Tuning for software analytics: Is it really necessary?. *Information and Software Technology*, *76*, pp.135-146.

[Tantithamthavorn’ 2016] Tantithamthavorn, C., McIntosh, S., Hassan, A.E. and Matsumoto, K., 2016, May. Automated parameter optimization of classification techniques for defect prediction models. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on* (pp. 321-332). IEEE.

[Arcuri’ 2011] Arcuri, A. and Fraser, G., 2011, September. On parameter tuning in search based software engineering. In *International Symposium on Search Based Software Engineering* (pp. 33-47). Springer, Berlin, Heidelberg.

Reference

[Wang' 2013] Wang, T., Harman, M., Jia, Y. and Krinke, J., 2013, August. Searching for better configurations: a rigorous approach to clone evaluation. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (pp. 455-465). ACM.

[Dev' 2002] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.A.M.T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, *6*(2), pp.182-197.

[Zhang' 2007] Zhang, Q. and Li, H., 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, *11*(6), pp.712-731.

[Harman' 2001] Harman, M. and Jones, B.F., 2001. Search-based software engineering. *Information and software Technology*, *43*(14), pp.833-839.

[Fu' 2016] Fu, W., Nair, V. and Menzies, T., 2016. Why is Differential Evolution Better than Grid Search for Tuning Defect Predictors?. *arXiv preprint arXiv:1609.02613*.

[Conte' 1986] Conte, S.D., Dunsmore, H.E. and Shen, Y.E., 1986. *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc..

[Shepperd' 2012] Shepperd, M. and MacDonell, S., 2012. Evaluating prediction systems in software project estimation. *Information and Software Technology*, *54*(8), pp.820-827.

[Mittas' 2013] Mittas, N. and Angelis, L., 2013. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Transactions on software engineering*, *39*(4), pp.537-551.

[Huang' 2018] Huang, Q., Xia, X., Lo, D. and Murphy, G.C., 2018. Automating Intention Mining. *IEEE Transactions on Software Engineering*.

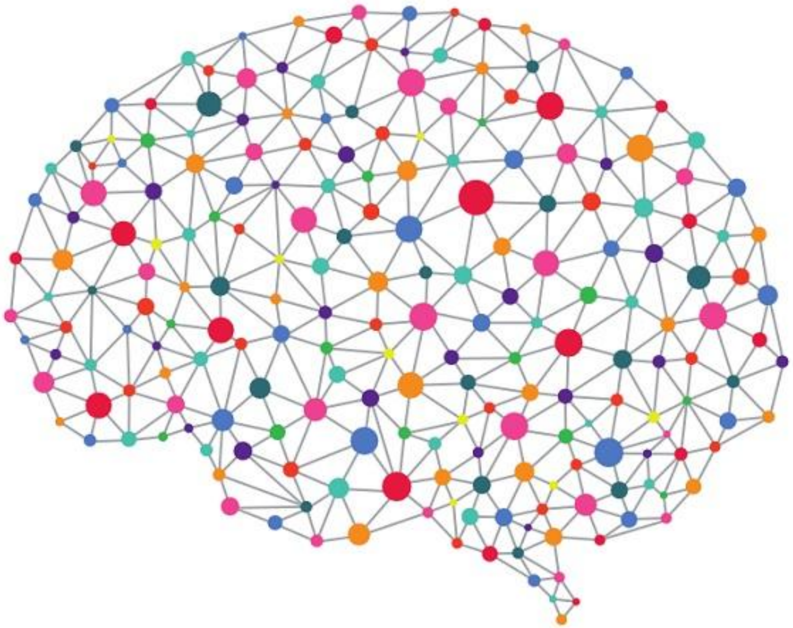
Backup Slides

Slides for extra explanation.

Future Work

Deep Learning

- State-of-the-art but computing intensive
- What to tune?
  - number of hidden layer
  - learning rate
  - amount of regularization
  - ......



Results

Running time

	faster											slower						
	ABE0	CART0	ATLM	LP4EE	FLASH_CART	CART_RD40	CART_RD160	CART_DE2	CART_DE8	CART_DENM	CART_MOEAD	CART_NSGA2	ABEN_RD40	ABEN_RD160	ABEN_DE2	ABEN_DE8	ABEN_NSGA2	total
kemerer	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	5	2	5	2	4	17	46
albrecht	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	5	2	8	3	6	24	59
finnish	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	5	3	10	4	8	30	71
miyazaki	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	5	4	14	5	10	37	86
desharnais	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	4	8	19	12	22	64	140
isbsg10	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	5	4	11	4	95	33	163
maxwell	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	6	13	38	16	29	111	224
kitchenham	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	6	15	32	26	48	126	264
china	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	<1	6	36	86	82	131	329	681
total	3	4	4	4	4	4	5	5	5	5	6	47	87	223	154	267	771	

Average runtime (in minutes), for one-way out of an N*M cross-validation experiment, in a local machine.

Can we replace the old defaults with new values?

	max_features (selected at random; 100% means "use all")				max_depth (of trees)				min_sample_split (continuation criteria)				min_samples_leaf (termination criteria)			
	25%	50%	75%	100%	$\leq$03	$\leq$06	$\leq$09	$\leq$12	$\leq$5	$\leq$10	$\leq$15	$\leq$20	$\leq$03	$\leq$06	$\leq$09	$\leq$12
kemerer	18	32	23	27	57	37	05	00	95	02	03	00	92	02	05	02
albrecht	13	23	20	43	63	28	08	00	68	32	00	00	83	15	02	00
isbsg10	12	35	28	25	57	33	08	00	47	23	15	15	60	27	10	03
finnish	07	03	27	63	32	56	12	00	73	18	05	03	78	17	05	00
miyazaki	10	22	27	40	31	46	20	03	42	24	18	16	78	13	07	02
maxwell	04	16	40	40	18	60	20	02	44	27	17	12	50	33	14	04
desharnais	25	23	27	25	40	46	11	02	36	26	13	25	32	26	24	19
kitchenham	01	12	32	56	03	42	45	10	43	30	17	10	48	35	12	04
china	00	04	25	71	00	00	25	75	56	30	10	02	68	28	04	00

KEY: 10 20 30 40 50 60 70 80 90 100 %

Tunings discovered by hyperparameter selections for CART_DE8

There are no "best" default settings.

Statistical Methods

All treatments are recursively bi-clustered into ranks. At each level, the treatments are split at the point where the expected values of the treatments after the split is most different to before, effect and significance tests are called to check that the two splits are actually different.

As a result, dozens of treatments end up generating just a handful of ranks.

Effect Test

Check the difference between the treatments is not some small effect.

Parametric Test:

Cohen's rule, Hedge's rule

Non-parametric Test:

Cliff's Delta, A12

Significance Test

Check the observed effect is not due to noise.

Parametric Test:

T-test

Non-parametric Test:

Bootstrap (slower, but safer)

Cliff’s Delta [Cliff’ 1996]

Non-parametric Effects Test

```
def cliffsDelta(lst1, lst2, small=0.147): # assumes all samples are nums
    "Cliff's delta between two list of numbers i,j."
    lt = gt = 0
    for x in lst1:
      for y in lst2 :
        if x > y: gt += 1
        if x < y: lt += 1
    z = abs(lt - gt) / (len(lst1) * len(lst2))
    return z < small # true is small effect in difference
```

$$A12 = \left(\sum_{x \in M, y \in N} \begin{cases} 1 & \text{if } x > y \\ 0.5 & \text{if } x == y \end{cases} \right) / (mn)$$

Bootstrap [Efron’ 1979]

Non-parametric Significance Test

To check if two populations X, Y are different using the bootstrap, sampling many times with replacement from both to generate (x1, y1), (x2, y2), (x3, y3)... etc.

For all those samples, check if some *testStatistic* in the original pair hold for all the other pairs. If it does more than (say) 95% of the time, then we are 95% confident in that the populations are the different.

In a bootstrap hypothesis test, *“testStatistic”* is the difference between the two populations, muted by the joint standard deviation of the populations.

Bootstrap [Efron' 1979]

Bootstrap Hypothesis Testing

- Observed Sample 1 of size n: $\{x_{obs,1}, x_{obs,2}, \dots, x_{obs,n}\} \Rightarrow \overline{x}_{obs}$
- Observed Sample 2 of size m: $\{y_{obs,1}, y_{obs,2}, \dots, y_{obs,m}\} \Rightarrow \overline{y}_{obs}$
- Observed sample mean difference: $t^*_{obs} = \overline{x}^*_{obs} - \overline{y}^*_{obs}$
- H_0: Both samples are from the same population
- H_1: Both samples are NOT from the same population and $\mu_x > \mu_y$
- $\alpha = 0.05$

Bootstrap [Efron' 1979]

Bootstrap Hypothesis Testing

- Step 0: Merge the two observed samples into one sample of $(n+m)$ observations
- Step 1: Draw a *bootstrap* sample of $(n+m)$ observations *with replacement* from the merged sample.
- Step 2: Calculate the mean of the first n observations and call it $\overline{x}^*$, calculate the mean of the remaining m observations and call it $\overline{y}^*$, and finally, evaluate the test statistic:

$$t^* = \overline{x}^* - \overline{y}^* \tag{1}$$

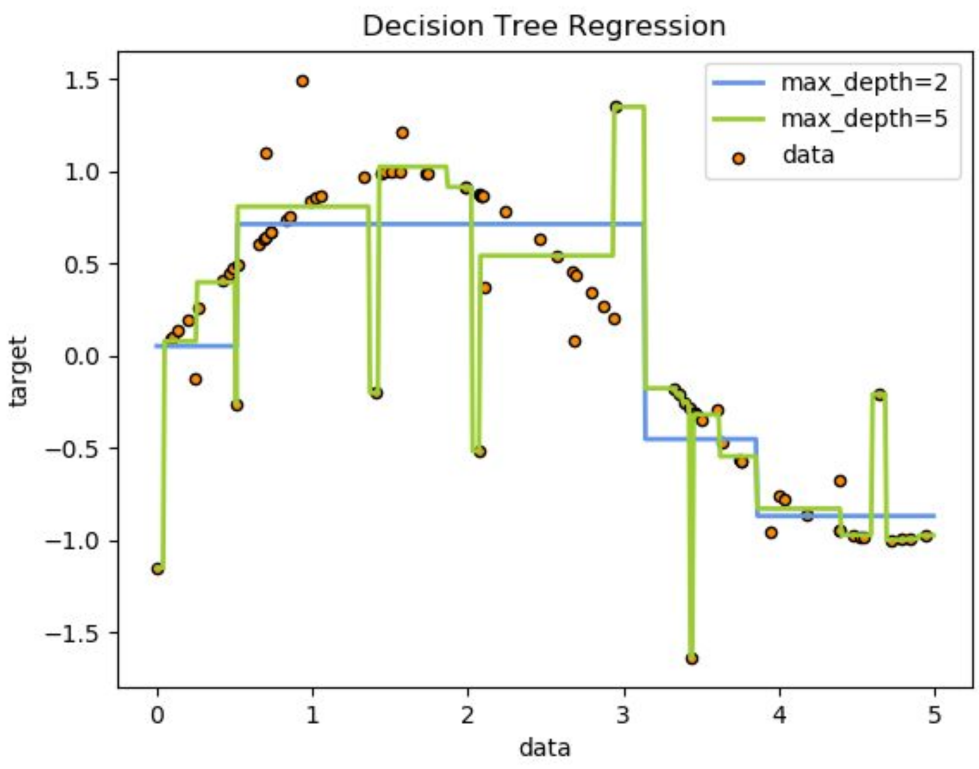
- Step 3: Repeat Step 1 and Step 2 for B (e.g., 3000) times and obtain B values of the test statistic.
- Step 4: The desired p-value is then estimated as

$$p - value \cong \frac{number\ of\ times\{t^* > t^*_{obs}\}}{B} \tag{2}$$

Reject H_0 if $p - value < \alpha$ and retain H_0 otherwise.

CART

Classification and Regression Trees



MOEA/D

Multiobjective Evolutionary Algorithm Based on Decomposition

MOEA/D

Multiobjective optimization problem:

$$maximize\ F(x) = (f_1(x), \dots, f_m(x))^T$$

$$subject\ to\ x \in \Omega$$

It is well known that a Pareto optimal solution to MOP under mild conditions, could be an optimal solution to a **scalar optimization problem** where the objective is **an aggregation** of all the f_i's

Decompose →

Three Decomposition Approaches:

1. Weight Sum Approach

$$maximize\ g^{ws}(x|\boldsymbol{\lambda}) = \sum_{i=1}^{m} \lambda_i f_i(x)$$

$$subject\ to\ x \in \Omega$$

[1]$\lambda_i \geq 0$
[2] $\sum_{i=1}^{m} \lambda_i = 1$

2. Tchebycheff Approach

$$maximize\ g^{te}(x|\boldsymbol{\lambda}, z^*) = \max_{1 \le i \le m} \{\lambda_i |f_i(x) - z_i^*|\}$$

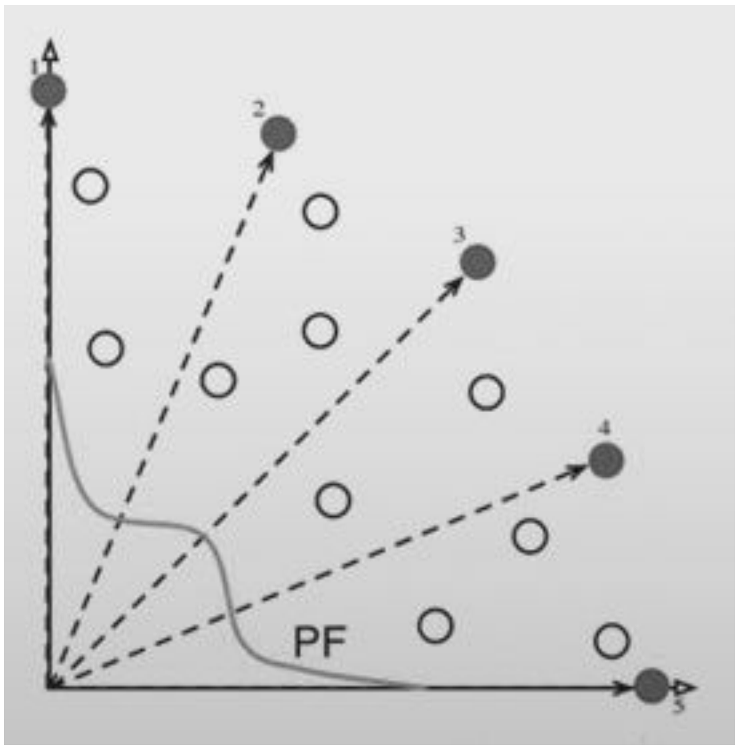
$$subject\ to\ x \in \Omega$$

[1]$\lambda_i \geq 0$ [3]$z^* = (z_1^*, \dots, z_m^*)^T$
[2] $\sum_{i=1}^{m} \lambda_i = 1$ [4] $z_i^* = \max\{f_i(x) | x \in \Omega\}$

3. Boundary Intersection Approach

MOEA/D

Multiobjective Evolutionary Algorithm Based on Decomposition



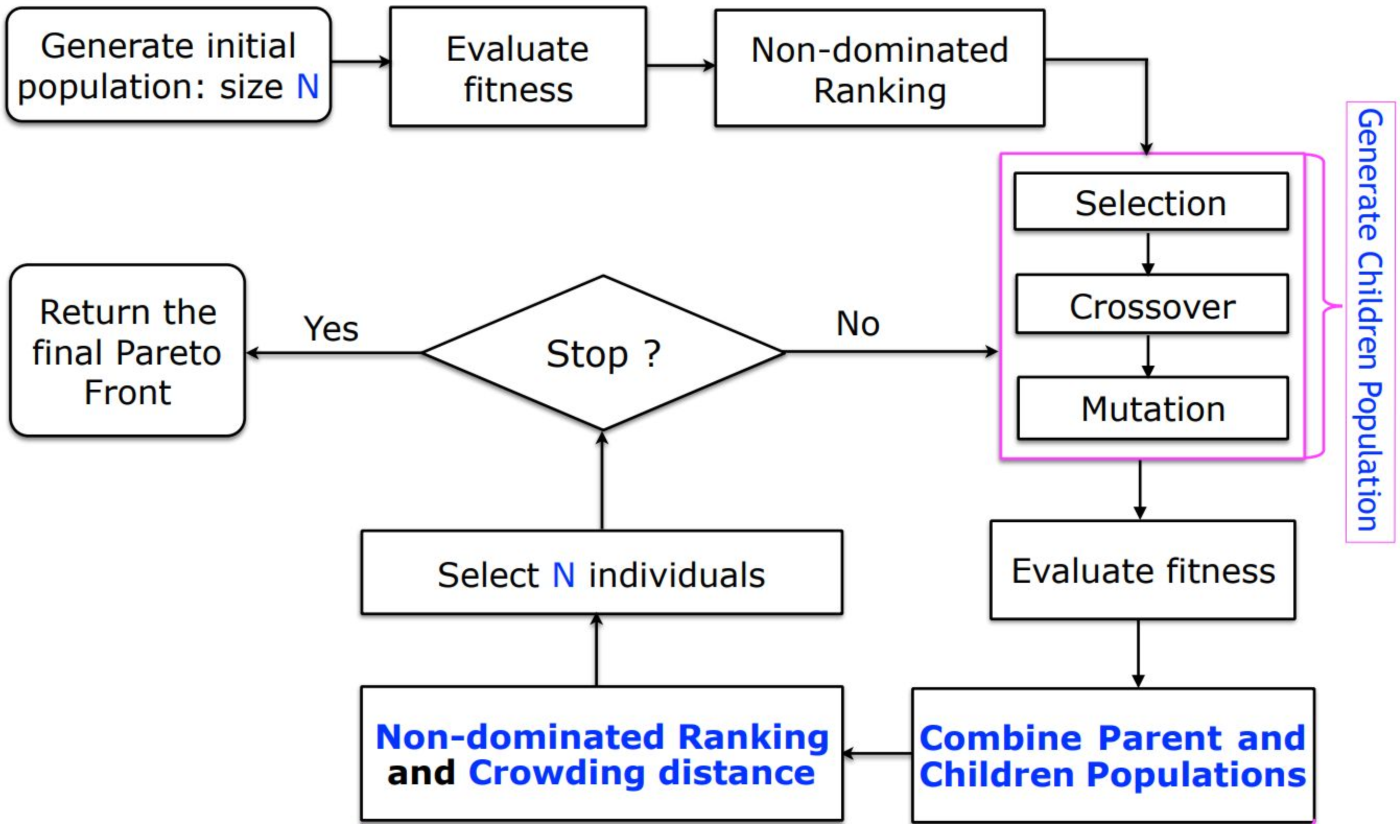
Algorithm 1: General Framework of MOEA/D

Input:
N: the number of subproblems to be decomposed;
W: a well-distributed set of weight vectors $\{\mathbf{w}^1, \ldots, \mathbf{w}^N\}$;
T: the neighborhood size.
Output:
P: the final approximation to PS.

$\mathbf{z} = (z_1 = +\infty, \ldots, z_k = +\infty)^\intercal$;
Generate a random set of solutions $P = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ in Ω;
for $i = 1, \ldots, N$ **do**
  $B_i \leftarrow \{i_1, \ldots, i_T\}$, such that: $\mathbf{w}^{i_1}, \ldots, \mathbf{w}^{i_T}$ are the T closest weight vectors to $\mathbf{w}^i$;
  $z_j \leftarrow \min(z_j, f_j(\mathbf{x}^i))$; // for $j = 1, \ldots, k$
while *stopping criterion is not satisfied* **do**
  for $\mathbf{x}^i \in P$ **do**
    Reproduction: Randomly select two indexes k, l from B_i, and then generate a new solution $\mathbf{y}$ from $\mathbf{x}^k$ and $\mathbf{x}^l$ by using genetic operators.
    Mutation: Apply a mutation operator on $\mathbf{y}$ to produce $\mathbf{y}'$.
    Update of $\mathbf{z}$: $z_j \leftarrow \min(z_j, f_j(\mathbf{x}^i))$; // for $j = 1, \ldots, k$
    Update of Neighboring Solutions: For each index $j \in B_i$, if $g(\mathbf{y}'|\mathbf{w}^j, \mathbf{z}) < g(\mathbf{x}^i|\mathbf{w}^j, \mathbf{z})$, then set $\mathbf{x}^j = \mathbf{y}'$;
return $P = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$;

NSGA-II

Non-dominated Sorting Genetic Algorithm II



COCOMO

Constructive Cost Model

COCOMO is a model that allows one to estimate the cost and effort when planning a new software development activity.

The model parameters are derived from fitting a regression formula using data from historical projects

ATLM

Automatically Transformed Linear Baseline Model [Whigham' 2015]

A multiple linear regression model which calculate the effort as:

$$effort = \beta_0 + \sum_i \beta_i \times a_i + \varepsilon_i$$

ai are explanatory attributes and ei are errors to the actual value, bi are prediction weights determined by using least square error estimation.

LP4EE

Linear Programming for Effort Estimation
[Sarro’ 2018]

This model minimises the Sum of Absolute Residual (SAR), when a new project is presented to the model, LP4EE predicts the effort as:

$$effort = a_1 \times x_1 + a_2 \times x_2 + \cdots + a_1 \times x_1 + a_n \times x_n$$

x is the value of given project feature and a is the corresponding coefficient evaluated by linear programming.