Cite as: Ioannis Tsamardinos, Vincenzo Lagani, Automated Machine Learning and Knowledge Discovery, ECCB 2018 Tutorial

Automated Machine Learning and Knowledge Discovery

IOANNIS TSAMARDINOS

PROFESSOR, CSD, UNIVERSITY OF CRETE

GNOSIS DATA ANALYSIS, CO-FOUNDER

VINCENZO LAGANI

ILIA STATE UNIVERSITY

GNOSIS DATA ANALYSIS, CO-FOUNDER

Outline

- **Part I (45’)**
  - Introduction to the problem and the tutorial
  - Estimation of performance (single configuration)
- **Part II (45')**
  - Estimation of performance (multiple configurations)
  - Incorporating User Preferences
- **Part III (45’)**
  - Feature Selection and Knowledge Discovery
  - Hyper-parameter search strategies
- **Part IV (45’)**
  - Post-analysis interpretation and visualizations
  - AI-assisted Auto-ML (algorithm selection, pipeline synthesis, meta-learning, feature learning)
  - Putting all together – The Just Add Data Bio platform
  - Tools for Auto-ML

Tune and Estimate

Choices, choices, choices

- Multiple algorithms available and applicable for all steps of the analysis (feature selection, classification, etc.)
- Each algorithm has a set of “tuning knobs”
- Optimize choice of combinations of algorithms and their “tuning knobs”

Hyper-Parameters vs. Parameters

- A **parameter** of a <u>model</u> (e.g., linear regression) is a quantity directly estimated from the data
  - In linear regression $y = w_1 x_1 + \ldots + w_n x_n + b$, w's and b are parameters, estimated from the data
- A **hyper-parameter** of an <u>algorithm</u> is a quantity not estimated by the data but set by the user
  - Determines the sensitivity of an algorithm to detecting patterns
  - A hyper-parameter may, of course, be estimated indirectly by CV (then it becomes a parameter in the complete procedure)

Examples of Hyper-Parameters

- **K-Nearest Neighbors**: K, distance function
- **Decision Trees**: MaxPChance (level of pruning)
- **Support Vector Machines**: Cost C, kernel K (each one has its own hyper-parameters)
- **Univariate Feature Selection**: p-value threshold
- **Lasso**: regularization parameter lambda
- Gaussian processes can have dozens of hyper-parameters [C. E. Rasmussen & C. K. I. Williams. "Gaussian Processes for Machine Learning", the MIT Press, 2006]

From Hyper-Parameters to Configurations

- Which algorithm to choose can also be seen as a hyper-parameter!
- Which data representation to use is a hyper-parameter

From Hyper-Parameters to Configurations

- **Which algorithm to choose can also be seen as a hyper-parameter!**
- Which data representation to use is a hyper-parameter

From Hyper-Parameters to Configurations

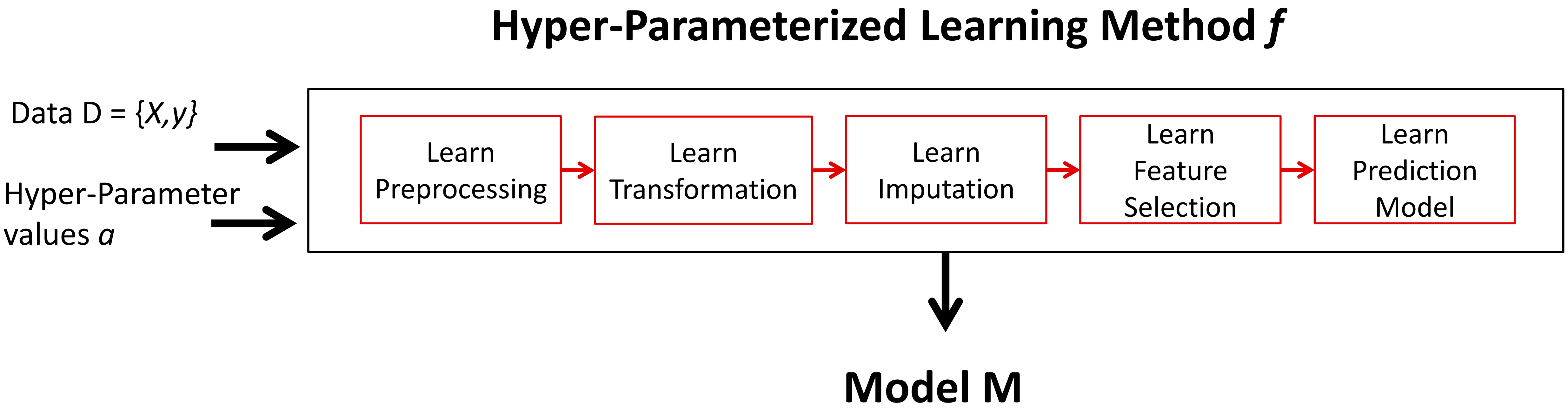
- **Which algorithm to choose can also be seen as a hyper-parameter!**
- **Which data representation to use is a hyper-parameter**

From Hyper-Parameters to Configurations

- **Which algorithm to choose can also be seen as a hyper-parameter!**
- **Which data representation to use is a hyper-parameter**
- **Point**: all our choices can be represented with a vector <u>*a*</u> of hyper-parameter values!

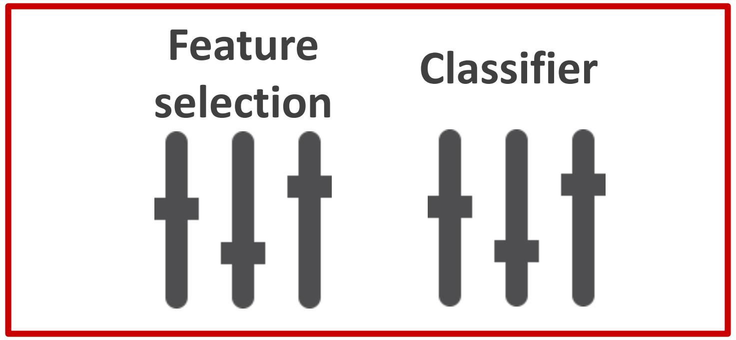
More algorithms vs. better tuning

- Personal Experience:
  - **Tuning of flexible, “good” algorithms** is more important than trying a plethora of algorithms with default values
  - *Personal choices*: SVMs, Random Forests, Gradient Boosting Trees (can represent all functions), ensemble methods
  - **Feature construction, data representation, data transformations**, more important than including more learning algorithms


Hyper-Parameterized Learning Method *f*
Data D = {*X*,*y*}
Hyper-Parameter values *a*
Learn Preprocessing
Learn Transformation
Learn Imputation
Learn Feature Selection
Learn Prediction Model
Model M

Hyper-Parameters and Configurations

- **Configuration**: an instantiation of a learning method ***f*** with specific hyper-parameter values.
- A configuration coincides with a non-hyperparameterized learning method.
- A configuration completely defines which computations to perform all the way from data to model.



Configuration	Hyper-parameter 1	Hyper-parameter 2	….	Hyper-parameter m
1	SES	0.05	…	SVM
2	Lasso	1	…	Random Forests
…	…	…	…	…
n	…	…	…	…

Tuning vs Model Selection

- **Model selection** (statistics):
  - produce several models, on all the data, select the "best"
  - Typically, the selection is manual based on some criteria (fitting + simplicity, distribution of residuals, etc.)

- **Tuning** [Tsamardinos et al. Machine Learning, 2018]
  - Tuning = **configuration selection**
  - Only one model is produced on all the data (no model selection)
  - The model is produced by the "best" configuration
  - "Best" is found by tuning the hyper-parameter

Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

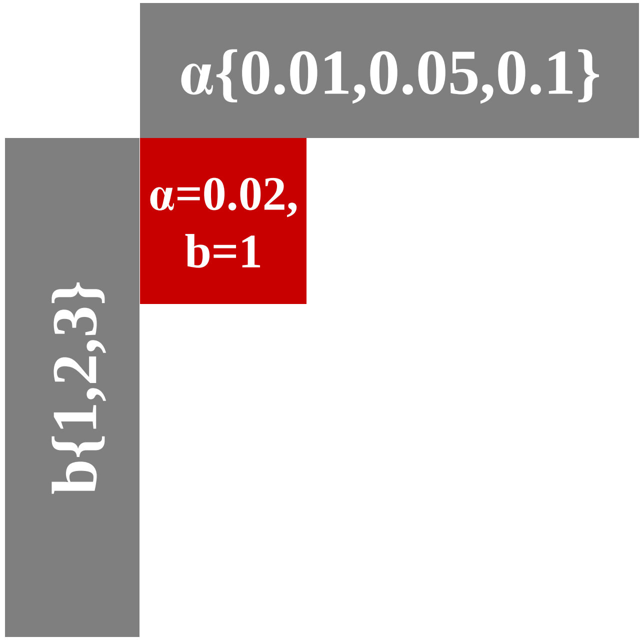
$\alpha\{0.01,0.05,0.1\}$

Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**



Grid Hyper-parameter Search

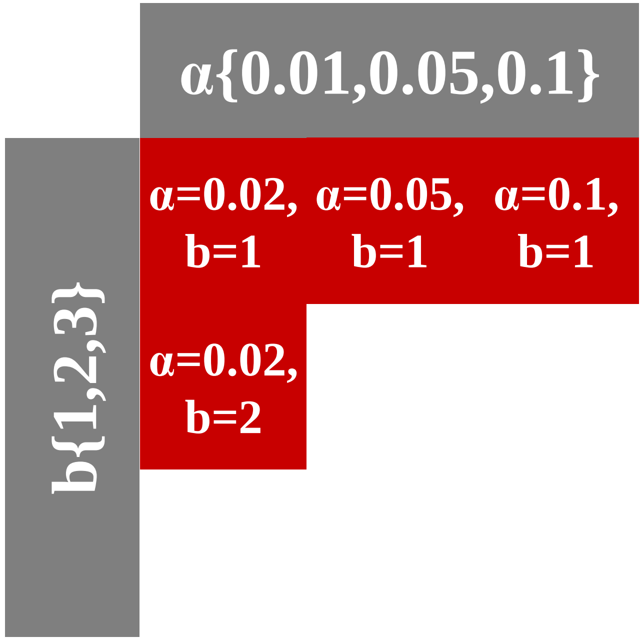
- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

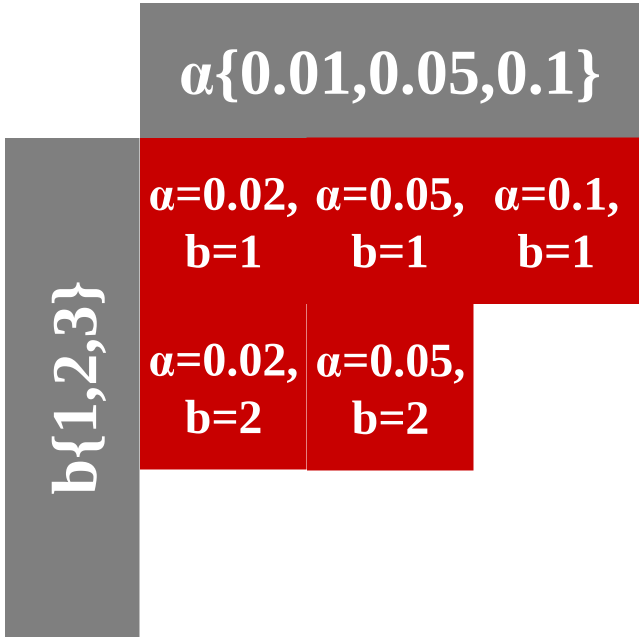
Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**



Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**



Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

	α{0.01,0.05,0.1}		
b{1,2,3}	α=0.02, b=1	α=0.05, b=1	α=0.1, b=1
	α=0.02, b=2	α=0.05, b=2	α=0.1, b=2

Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

	α{0.01,0.05,0.1}		
b{1,2,3}	α=0.02, b=1	α=0.05, b=1	α=0.1, b=1
	α=0.02, b=2	α=0.05, b=2	α=0.1, b=2
	α=0.02, b=3		

Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

	α{0.01,0.05,0.1}		
b{1,2,3}	α=0.02, b=1	α=0.05, b=1	α=0.1, b=1
	α=0.02, b=2	α=0.05, b=2	α=0.1, b=2
	α=0.02, b=3	α=0.05, b=3	

Grid Hyper-parameter Search

- A priori decide which algorithms to try in each step
- A priori decide the values to try for each hyper-parameter
- Try all combinations (full-factorial)
- Called **Grid Search**
  - Try values {0.01, 0.05, 0.1} for hp *a*
  - Try values {1, 2, 3} for hp *b*
- **Static hyper-parameter search strategies predetermine the configurations to try**

	α {0.01,0.05,0.1}		
b{1,2,3}	α=0.02, b=1	α=0.05, b=1	α=0.1, b=1
	α=0.02, b=2	α=0.05, b=2	α=0.1, b=2
	α=0.02, b=3	α=0.05, b=3	α=0.1, b=3

Example of Tune-n-Estimate
(the wrong way)

- Construct all models from each configuration f_i, i=1, …, 100
- Select Best
- Report its estimated performance

for each configuration f_i

  $\langle Perf_i , model_i \rangle = \text{Hold-Out}(D, f_i)$

end for

  $j = \text{argmax } Perf_i$

return $\langle Perf_i , model_i \rangle$

Construct all Models, Select Best

Train	Test

Algorithm	Parameter	Performance (Loss)
K-NN	K=1	0.81
	K=2	0.84
	K=5	0.88
DT	MaxPChance=0.01	0.83
	MaxPChance=0.05	0.9
	MaxPChance=0.1	0.81
SB	l = 0	0.75
	l=1	0.83

Construct all Models, Select Best

Train	Test

Algorithm	Parameter	Performance (Loss)
K-NN	K=1	0.81
	K=2	0.84
	K=5	0.88
DT	MaxPChance=0.01	0.83
	MaxPChance=0.05	0.9
	MaxPChance=0.1	0.81
SB	l = 0	0.75
	l=1	0.83

Construct all Models, Select Best

Train	Test

Algorithm	Parameter	Performance (Loss)
K-NN	K=1	0.81
	K=2	0.84
	K=5	0.88
DT	MaxPChance=0.01	0.83
	MaxPChance=0.05	0.9
	MaxPChance=0.1	0.81
SB	l = 0	0.75
	l=1	0.83

Selected model

Construct all Models, Select Best

Train	Test

Algorithm	Parameter	Performance (Loss)
K-NN	K=1	0.81
	K=2	0.84
	K=5	0.88
DT	MaxPChance=0.01	0.83
	MaxPChance=0.05	0.9
	MaxPChance=0.1	0.81
SB	l = 0	0.75
	l=1	0.83

Returned Estimate (WRONG WAY)

Selected model

Construct all Models, Select Best

for each configuration f_i

$\langle \mathrm{Perf}_i , \mathrm{model}_i \rangle = \text{Hold-Out2}(D, f_i)$

end for

$j = \mathrm{argmax}\ \mathrm{Perf}_i$

return $\langle \mathrm{Perf}_i , \mathrm{model}_i \rangle$

Construct all Models, Select Best

for each configuration f_i

$\langle Perf_i , model_i \rangle = \text{Hold-Out2}(D, f_i)$

end for

$j = \text{argmax } Perf_i$

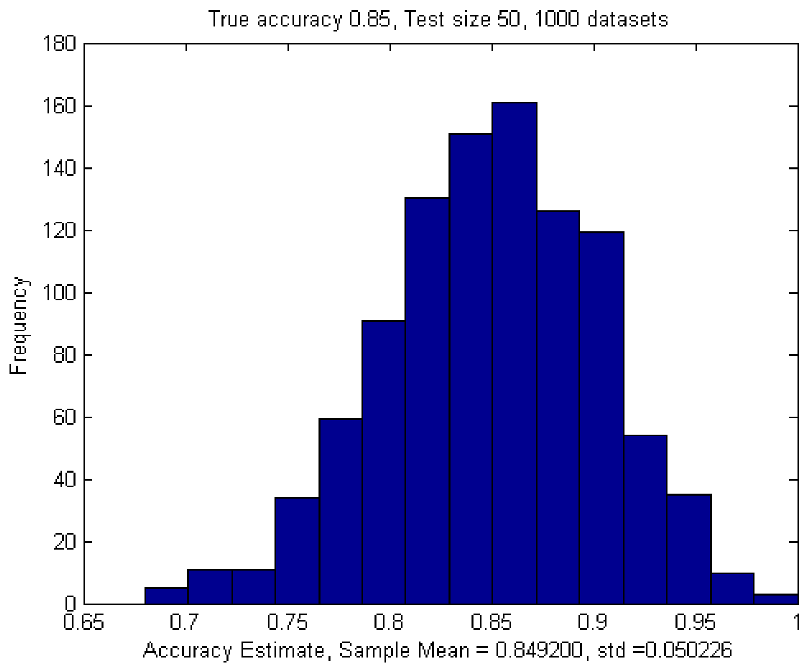
return $\langle Perf_j , model_j \rangle$

It peeks in the test cases to select the final model: **violation** of **Golden Rule**

Extreme Distributions: 1 Model

Train	Test

Alg	Parameter	Loss
K-NN	K=1	



Assume: Equal true accuracies 85%

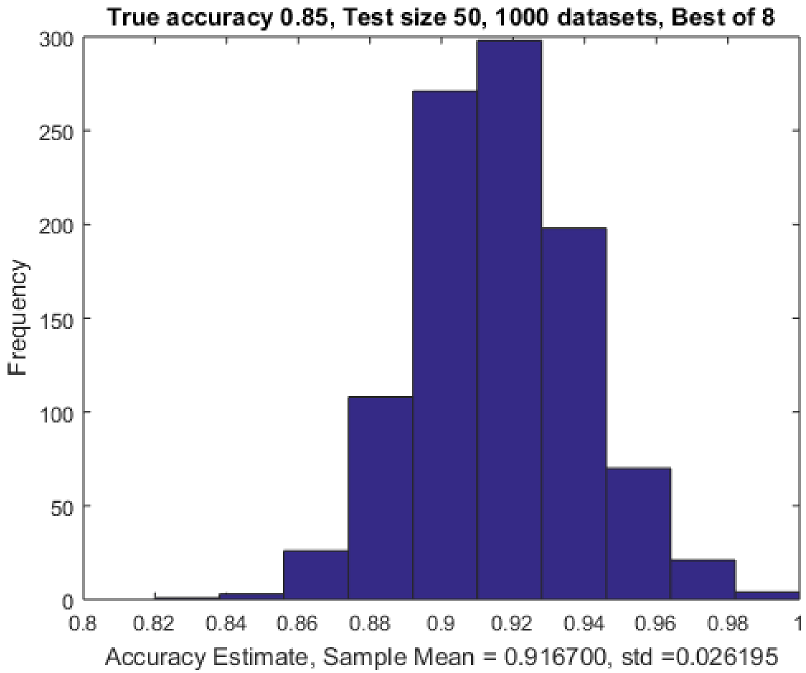
Mean = 0.85

Std = sqrt(N·p·(1-p)) / N = 0.0505

Extreme Distributions: 8 Models

Train | Test

Alg	Parameter	Loss
K-NN	K=1	
	K=2	
	K=5	
DT	MaxPChance=0.01	
	MaxPChance=0.05	
	MaxPChance=0.1	
SB	l = 0	
	l=1	



Assume: Equal true accuracies 85%Mean, Std follow an Extreme Distribution

Performance Estimation Bias

Performance Estimation Bias

- Let $\mathbf{m_{1,} \ldots, m_n}$ be the sample performances of each configuration

Performance Estimation Bias

- Let $\mathbf{m_1, \ldots, m_n}$ be the sample performances of each configuration
- Let $\mu_1, \ldots, \mu_n$ be the true performances of each configuration

Performance Estimation Bias

- Let $\mathbf{m_1, \ldots, m_n}$ be the sample performances of each configuration
- Let $\boldsymbol{\mu_1, \ldots, \mu_n}$ be the true performances of each configuration
- For unbiased estimation we have $\boldsymbol{\mu_1 = E(m_1), \ldots, \mu_n = E(m_n)}$

Performance Estimation Bias

- Let $\mathbf{m_1, \ldots, m_n}$ be the sample performances of each configuration
- Let $\boldsymbol{\mu_1, \ldots, \mu_n}$ be the true performances of each configuration
- For unbiased estimation we have $\boldsymbol{\mu_1 = E(m_1), \ldots, \mu_n = E(m_n)}$
- We return as our estimate the best sample performance $\mathbf{max(m_1, \ldots, m_n)}$

Performance Estimation Bias

- Let $m_1, \ldots, m_n$ be the sample performances of each configuration
- Let $\mu_1, \ldots, \mu_n$ be the true performances of each configuration
- For unbiased estimation we have $\mu_1 = E(m_1), \ldots, \mu_n = E(m_n)$
- We return as our estimate the best sample performance $\max(m_1, \ldots, m_n)$
- On average we return $E(\max(m_1, \ldots, m_n))$

Performance Estimation Bias

- Let $m_1, \ldots, m_n$ be the sample performances of each configuration
- Let $\mu_1, \ldots, \mu_n$ be the true performances of each configuration
- For unbiased estimation we have $\mu_1 = E(m_1), \ldots, \mu_n = E(m_n)$
- We return as our estimate the best sample performance $\max(m_1, \ldots, m_n)$
- On average we return $E(\max(m_1, \ldots, m_n))$
- True best performance is $\max(\mu_1, \ldots, \mu_n) = \max(E(m_1), \ldots, E(m_n))$

Performance Estimation Bias

- Let $m_{1}, \ldots, m_{n}$ be the sample performances of each configuration
- Let $\mu_{1}, \ldots, \mu_{n}$ be the true performances of each configuration
- For unbiased estimation we have $\mu_1 = E(m_1), \ldots, \mu_n = E(m_n)$
- We return as our estimate the best sample performance $\max(m_{1}, \ldots, m_n)$
- On average we return $E(\max(m_{1}, \ldots, m_n))$
- True best performance is $\max(\mu_{1}, \ldots, \mu_n) = \max(E(m_1), \ldots, E(m_n))$
- Our estimate on average $E(\max(m_{1}, \ldots, m_n)) \geq \max(E(m_1), \ldots, E(m_n))$ true best, by Jensen's inequality

Test set prediction matrix

Folds	C_1	C_2	....	C_n
1	0.9	0.8	…	0.7
2	0.8	0.7	…	0.6
…	…	…	…	…
K	…	…	…	…
Mean	**0.9**	**0.8**	**…**	**0.7**

Which model out of all trained should we use?

Test set prediction matrix

Folds	C_1	C_2	....	C_n
1	0.9	0.8	…	0.7
2	0.8	0.7	…	0.6
…	…	…	…	…
K	…	…	…	…
Mean	**0.9**	**0.8**	**…**	**0.7**

Which model out of all trained should we use?

Return model trained on all data using best configuration. Should be best on average

Test set prediction matrix

Folds	C_1	C_2	….	C_n
1	0.9	0.8	…	0.7
2	0.8	0.7	…	0.6
…	…	…	…	…
K	…	…	…	…
Mean	**0.9**	**0.8**	**…**	**0.7**

Test set prediction matrix

Folds	C_1	C_2	….	C_n
1	0.9	0.8	…	0.7
2	0.8	0.7	…	0.6
…	…	…	…	…
K	…	…	…	…
Mean	**0.9**	**0.8**	**…**	**0.7**

Which model out of all trained should we use?

Is its expected performance the Cross-Validated one?

Return model trained on all data using best configuration. Should be best on average

Test set prediction matrix

Folds	C_1	C_2	….	C_n
1	0.9	0.8	…	0.7
2	0.8	0.7	…	0.6
…	…	…	…	…
K	…	…	…	…
Mean	**0.9**	**0.8**	**…**	**0.7**

Which model out of all trained should we use?

Return model trained on all data using best configuration. Should be best on average

Is its expected performance the Cross-Validated one?

No! The Cross-Validated accuracy of the best configuration is **optimistic**! (multiple induction problem, Jensen 1992)

Conservatism vs. Optimism

- Each CV single-configuration estimates are **conservative**: they are based on training with fewer samples than the final model
- CV multiple-configuration estimates are **optimistic:**
- Winner depends on:
  - **Sample size**: smaller sample size optimism wins
  - **Number of configurations tried**: more configurations, optimism wins
  - **“Correlation” of configuration**: the more independent, the larger the optimism
  - **Distribution of true performances of configurations**: the less variant, the more optimism

Choose Configuration AND Estimate Performance

- How?

Choose Model AND Estimate Performance

Choose Model AND Estimate Performance

Train

- Train: used to train model

Choose Model AND Estimate Performance

- Train: used to train model
- Tune: used to choose best configuration

Choose Model AND Estimate Performance

- Train: used to train model
- Tune: used to choose best configuration
- Estimate: used to estimate performance

Choose Model AND Estimate Performance

- Train: used to train model
- Tune: used to choose best configuration
- Estimate: used to estimate performance

- Called Train-Validation-Test in the literature

Simple Train-Tune-Estimate HoldOut

Train	Tune	Estimate

STTE-Hold-Out (Data D, learning method f**,** *vectors of hps* ***a****)*

Randomly partition row indexes to TrainI, TuneI, EstI

For all hp a_i in $\boldsymbol{a}$ //Try all configurations

Create a new configuration $f_i = f(\cdot , a_i)$

$M_i = f_i (D (TrainI)), Est_i = l(y(TuneI), M_i(X(TuneI))$

End For

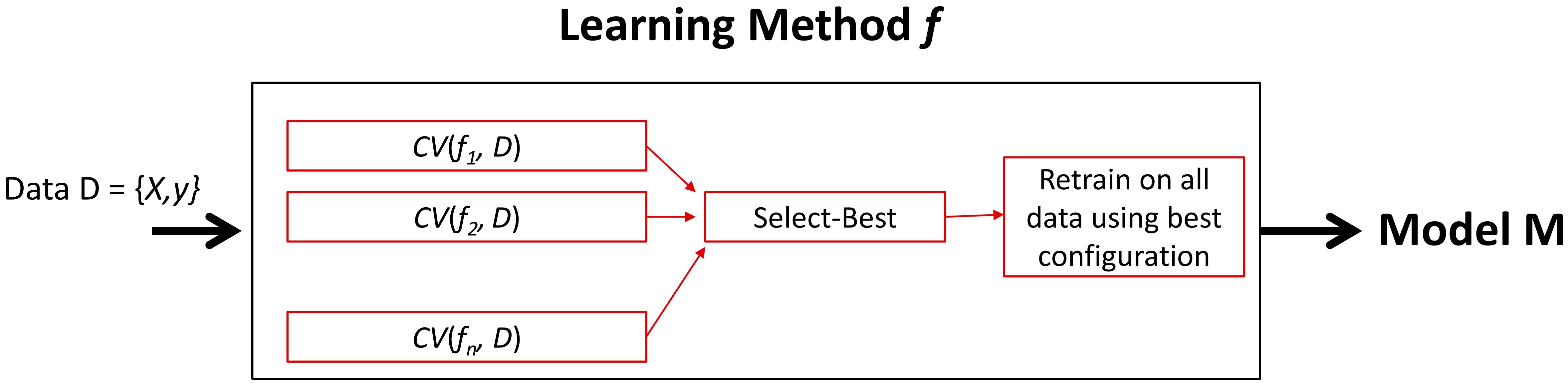
i* = argmax Est_i *// Best configuration based on Tune set*

Returned Model: M_{i*}

Returned Estimation: $l(y(EstI), M_{i*}(X(EstI))$

- $f_i = f(\cdot , a_i)$: f is called a **closure**; makes programming really easy
- Trains C models, C the number of configurations
- Correctly follows the Golden Rule, correct estimation
- Does not train on all data, as it should
- Directly estimates the loss of a model, not of the learning function
- **Pros**: computationally efficient, simple
- **Cons**: loses data to both Tune and Estimate
- Use when sample size is really large

Consider Tuning part of learning



Cross-Validation with Tuning

Algorithm 2 **CVT**(f, $D = \{F_1, \ldots, F_K\}$, Θ): Cross-Validation With Tuning

Input: Learning method f, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into about equally-sized folds F_i, set of configurations Θ

Output: Model M, Performance estimation L_{CVT}, out-of-sample predictions Π on all folds for all configurations

for $i = 1$ to $C = |\Theta|$ **do**
  // Create a closure of f (a new function) by grounding the configuration θ_i
  $f_i \leftarrow \text{Closure}(f(\cdot, \theta_i))$
  $\langle M_i, L_i, \Pi_i \rangle \leftarrow$ **CV**(f_i, D)
end for
$i^\star \leftarrow \arg\min_i L_i$
// Final Model trained by f on all available data using the best configuration
$M \leftarrow f(D, \theta_{i^\star})$
// Performance estimation; may be optimistic and should not be reported in general
$L_{CVT} \leftarrow L_{i^\star}$
// Out-of-sample predictions are used by bias-correction methods
Collect all out-of-sample predictions of all configurations in one matrix $\Pi \leftarrow [\Pi_1 \cdots \Pi_C]$
Return $\langle M, L_{CVT}, \Pi \rangle$

Nested Cross-Validation

- Cross-Validate a learning method that returns a single model, **but performs tuning internally**
- **Cross-Validate CVT**!

Algorithm 3 **NCV**(f, $D = \{F_1, \ldots, F_K\}$, Θ): Nested Cross-Validation

---

Input: Learning method f, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into about equally-sized folds F_i, set of configurations Θ

Output: Model M, Performance estimation L_{NCV}, out-of-sample predictions Π on all folds for all configurations

1: // Create closure by grounding the f and the Θ input parameters of **CVT**
2: $f' \leftarrow$ **CVT**(f, $\cdot$, Θ)
3: // Notice: final Model is trained by f' on all available data; final estimate is provided by basic CV (no tuning) since f' returns a single model each time
4: $\langle M, L_{NCV}, \Pi \rangle \leftarrow$ **CV**(f', D)
5: **Return** $\langle M, L_{NCV} \rangle$

NCV Trace: Model Production

- Configurations *a*, *b*, Folds 1, 2, 3

Train On	With Conf.	Produce	Apply on	Accuracy
1, 2	a	M_1	3	0.7
1, 3	a	M_2	2	0.8
2, 3	a	M_3	1	0.6
				$Mean_a$ = 0.7
1, 2	b	M_4	3	0.6
1, 3	b	M_5	2	0.7
2, 3	b	M_6	1	0.5
				$Mean_b$ = 0.6
Select a				
1, 2, 3	a	M_7	N/A	
Return model M_7				

NCV Trace: Estimation

- Fold 3 is held-out as an Estimation set

Train On	With Conf.	Produce	Apply on	Accuracy
1	a	M_8	2	0.7
2	a	M_9	1	0.8
				Mean$_a$ = 0.75
1	b	M_{10}	2	0.6
2	b	M_{11}	1	0.7
				Mean$_a$ = 0.65
Select a				
1, 2	a	M_{12}	**3**	**0.9**

NCV Trace: Estimation

- Fold 2 is held-out as an Estimation set

Train On	With Conf.	Produce	Apply on	Accuracy
1	a	M_{13}	3	0.6
3	a	M_{14}	1	0.7
				Mean_a = 0.65
1	b	M_{15}	3	0.7
3	b	M_{16}	1	0.8
				Mean_a = 0.75
Select b				
1, 3	b	M_{17}	**2**	**0.7**

NCV Trace: Estimation

- Fold 1 is held-out as an Estimation set

Train On	With Conf.	Produce	Apply on	Accuracy
2	a	M_{18}	3	0.8
3	a	M_{19}	2	0.6
				Mean$_a$ = 0.7
2	b	M_{20}	3	0.6
3	b	M_{21}	2	0.6
				Mean$_a$ = 0.6
Select a				
2, 3	a	M_{22}	**1**	**0.8**

Final Estimate: mean of 0.9 + 0.7 + 0.8 = **0.8**

How many models trained?

C: number of configurations

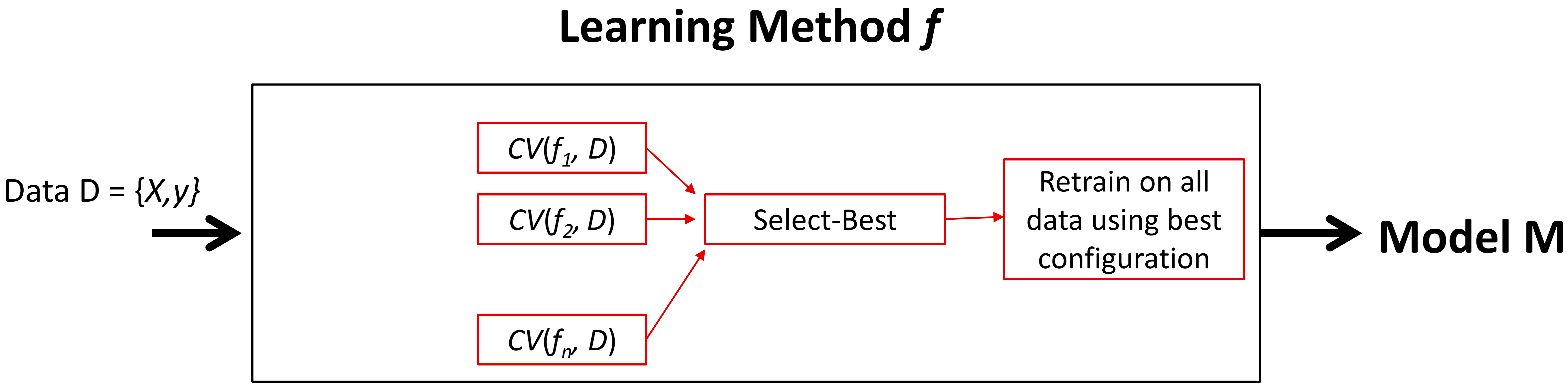
K: number of folds

- To produce the final model CVT is called with K folds
  - C configurations × K folds for estimating best configuration
  - +1 to train on the full dataset
  - $= C \times K + 1$
- To estimate its performance
  - Run CVT with K-1 folds, K times
  - $= (C \times (K-1) + 1) \times K$
- **Total number of models trained = $C \times K^2 + K + 1$**
- **Expensive**

Nested-Cross Validation

- Fold loop within CVT: **inner** CV loop
- Fold loop within CV: **outer** CV loop
- The **standard protocol** for small-sample, omics data
- Want more accurate estimation, run **Repeated-CV** instead of CV
- Want even more accurate estimation, run **Repeated-CVT** instead of CVT
- Computationally expensive $O(K^2)$ models per configuration; **Can we do better**?

Let's Focus on Selection



- Our selection strategy creates the estimation problem

Let's Focus on Selection

Learning Method f

Data D

$CV(f_1, D)$

$CV(f_2, D)$

$CV(f_n, D)$

Test set prediction matrix Π

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Select-Best

Retrain on all data using best configuration

Π_{ij} = prediction of model trained with configuration *j* on sample *i*, when *i* was **in the test set**

Choose the column with the lowest loss

Output best **configuration**

Model M

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

No need to train new models, computationally efficient

Can safely replace **nested Cross-Validation; Next standard**?

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

No need to train new models, computationally efficient

Can safely replace **nested Cross-Validation; Next standard**?

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Solution:

Estimate the performance of the **configuration selection procedure:**

- Use bootstrapping or CV on the test prediction matrix!
  - Select best configuration on a subset of the matrix
  - Estimate performance of the selected configuration on the held-out set

No need to train new models, computationally efficient

Can safely replace **nested Cross-Validation; Next standard**?

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Bootstrap **B**ias **C**orrected **CV**

I. Tsamardinos, E. Greasidou, G. Borboudakis, "Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation", **Machine Learning** 2018

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
1	yes	no	…	no
…	…	…	…	…
224	no	yes	…	yes

Sample	f_1	f_2	….	f_n
2	no	yes	…	yes
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

Bootstrap **B**ias **C**orrected **CV**

I. Tsamardinos, E. Greasidou, G. Borboudakis, “Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation”, **Machine Learning** 2018

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Select best Configuration, i.e. C_1

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
1	yes	no	…	no
…	…	…	…	…
224	no	yes	…	yes

Sample	f_1	f_2	….	f_n
2	no	yes	…	yes
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

Bootstrap **B**ias **C**orrected **CV**

I. Tsamardinos, E. Greasidou, G. Borboudakis, “Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation”, **Machine Learning** 2018

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Select best Configuration, i.e. C_1

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
1	yes	no	…	no
…	…	…	…	…
224	no	yes	…	yes

Measure Performance P_1 of C_1

Sample	f_1	f_2	….	f_n
2	no	yes	…	yes
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

Bootstrap **B**ias **C**orrected **CV**

I. Tsamardinos, E. Greasidou, G. Borboudakis, "Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation", **Machine Learning** 2018

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Select best Configuration, i.e. C_1

Sample	f_1	f_2	….	f_n
1	yes	no	…	no
1	yes	no	…	no
…	…	…	…	…
224	no	yes	…	yes

Measure Performance P_1 of C_1

Sample	f_1	f_2	….	f_n
2	no	yes	…	yes
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

B=1

Bootstrap **B**ias **C**orrected **CV**

I. Tsamardinos, E. Greasidou, G. Borboudakis, "Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation", **Machine Learning** 2018

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Select best Configuration, i.e. C_1

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
1	yes	no	…	no
…	…	…	…	…
224	no	yes	…	yes

Measure Performance P_1 of C_1

Sample	f_1	f_2	....	f_n
2	no	yes	…	yes
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

B=1

…

Select best Configuration, i.e. C_2

Sample	f_1	f_2	....	f_n
2	no	yes	…	yes
2	no	yes	…	yes
…	…	…	…	…
210	no	yes	…	yes

Measure Performance P_2 of C_2

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

B=1000

Bootstrap **B**ias **C**orrected **CV**

I. Tsamardinos, E. Greasidou, G. Borboudakis, “Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation”, **Machine Learning** 2018

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Select best Configuration, i.e. C_1

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
1	yes	no	…	no
…	…	…	…	…
224	no	yes	…	yes

Measure Performance P_1 of C_1

Sample	f_1	f_2	....	f_n
2	no	yes	…	yes
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

B=1

…

Select best Configuration, i.e. C_2

Sample	f_1	f_2	....	f_n
2	no	yes	…	yes
2	no	yes	…	yes
…	…	…	…	…
210	no	yes	…	yes

Measure Performance P_2 of C_2

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

B=1000

$$\text{Performance} = \frac{\sum_{i=1}^{B} P_i}{B}$$

Bootstrap **B**ias **C**orrected **CV**

I. Tsamardinos, E. Greasidou, G. Borboudakis, “Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation”, **Machine Learning** 2018

Performance estimation bias correction

Test set prediction matrix

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
2	no	yes	…	no
…	…	…	…	…
224	no	yes	…	yes

Select best Configuration, i.e. C_1

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
1	yes	no	…	no
…	…	…	…	…
224	no	yes	…	yes

Measure Performance P_1 of C_1

Sample	f_1	f_2	....	f_n
2	no	yes	…	yes
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

B=1

…

Select best Configuration, i.e. C_2

Sample	f_1	f_2	....	f_n
2	no	yes	…	yes
2	no	yes	…	yes
…	…	…	…	…
210	no	yes	…	yes

Measure Performance P_2 of C_2

Sample	f_1	f_2	....	f_n
1	yes	no	…	no
3	yes	no	…	no
…	…	…	…	…
220	no	yes	…	yes

B=1000

Same procedure used to provide confidence intervals!

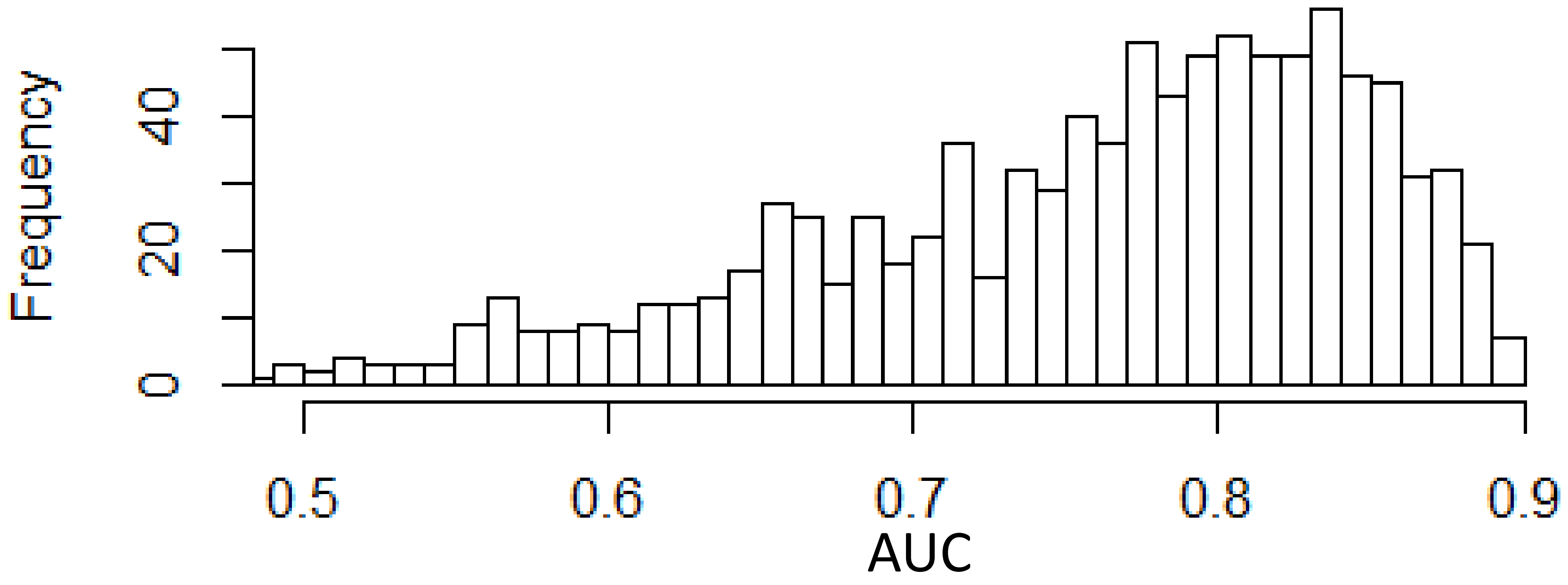
$$\textbf{Performance} = \frac{\sum_{i=1}^{B} P_i}{B}$$

Performance measured on new samples each time

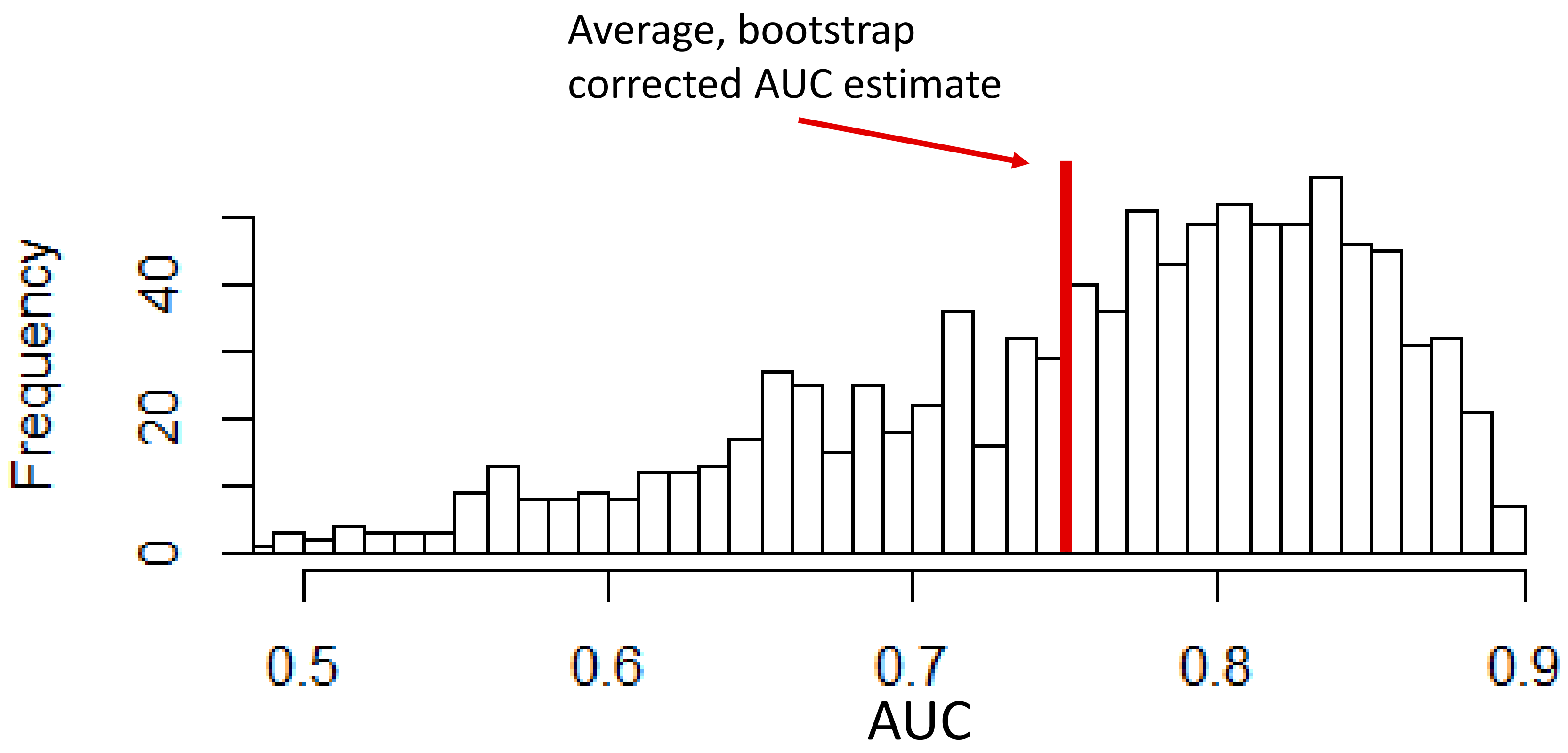
Bootstrap **B**ias **C**orrected **CV**

I. Tsamardinos, E. Greasidou, G. Borboudakis, “Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation”, **Machine Learning** 2018

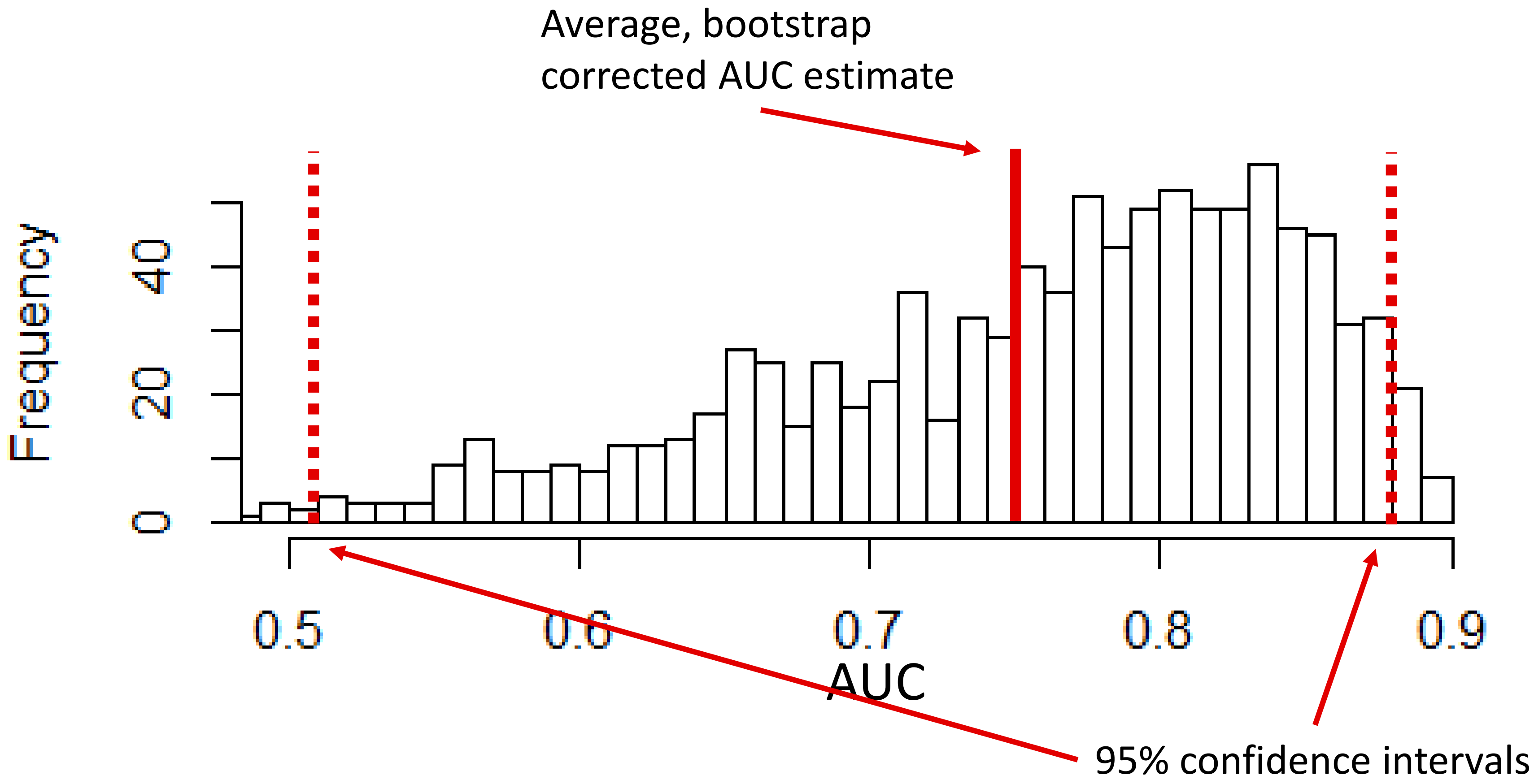
Generation of CIs



Generation of CIs



Generation of CIs

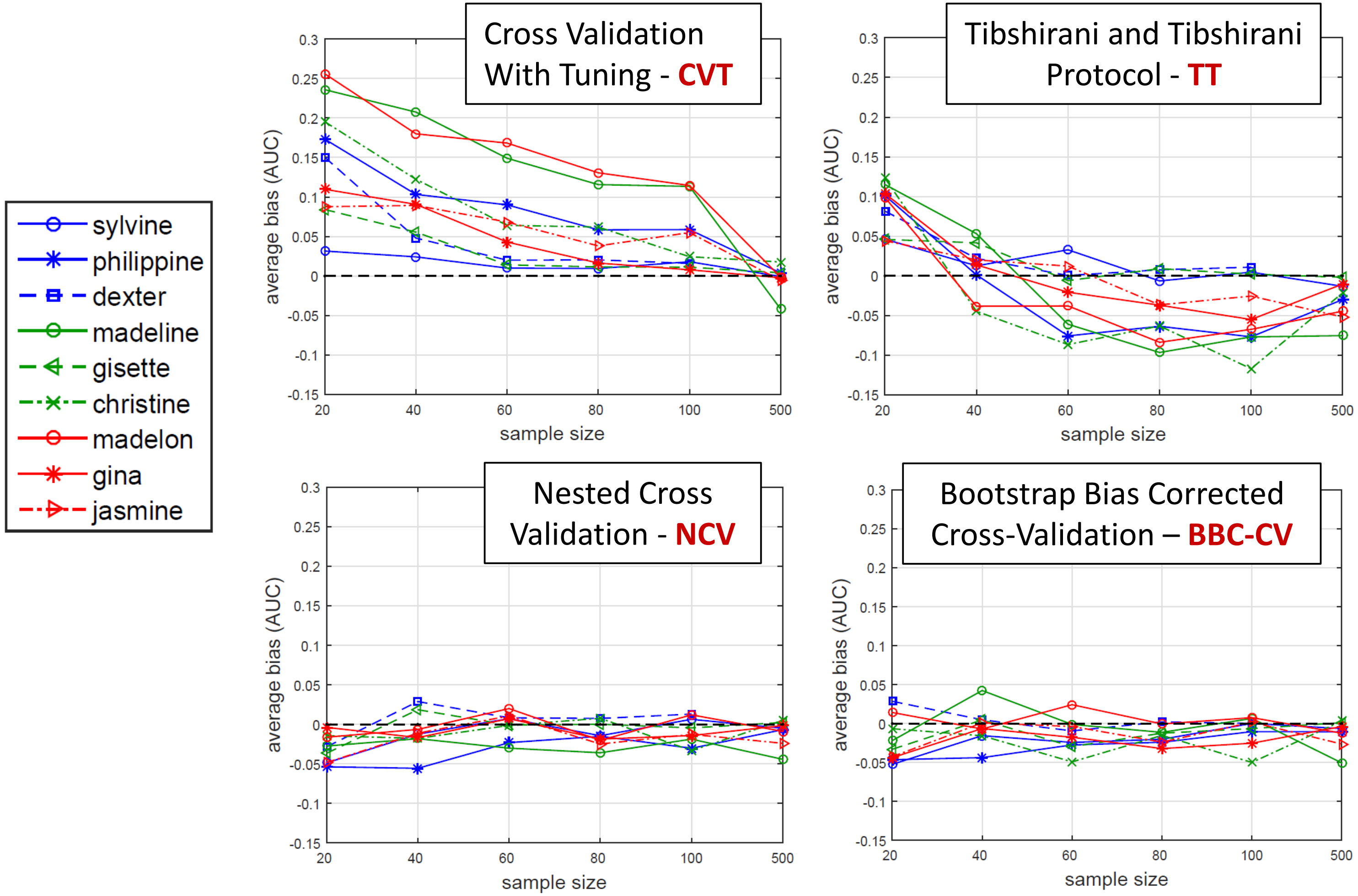


Algorithm 5 BBC-CV(f, $D = \{F_1, \ldots, F_K\}$, Θ): Cross-Validation with Tuning, Bias removal using the BBC method

Input: Learning method f, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into approximately equally-sized folds F_i, set of configurations Θ
Output: Model M, Performance estimation L_{BBC}, 95% confidence interval $[lb, ub]$

```
// Notice: the final Model is the same as in CVT
⟨M, L_CVT, Π⟩ ← CVT(f, D, Θ)
for b = 1 to B do
   Π^b ← sample with replacement N rows of Π
   Π^\b ← Π \ Π^b // get samples in Π and not in Π^b
   // Apply the configuration selection method on the bootstrapped out-of-sample predictions
   j ← ccs(Π^b, y^b)
   // Estimate the error of the selected configuration on predictions not selected by this bootstrap
   L_b ← l(y^\b, Π(:, j)^\b)
end for
L_BBC = (1/B) Σ_{b=1}^{B} L_b
// Compute 95% confidence interval; L_(k) denotes the k-th value of L_b's in ascending order
[lb, ub] = [L_(0.025·B), L_(0.975·B)]
Return ⟨M, L_BBC, [lb, ub]⟩
```

Cross Validation bias correction

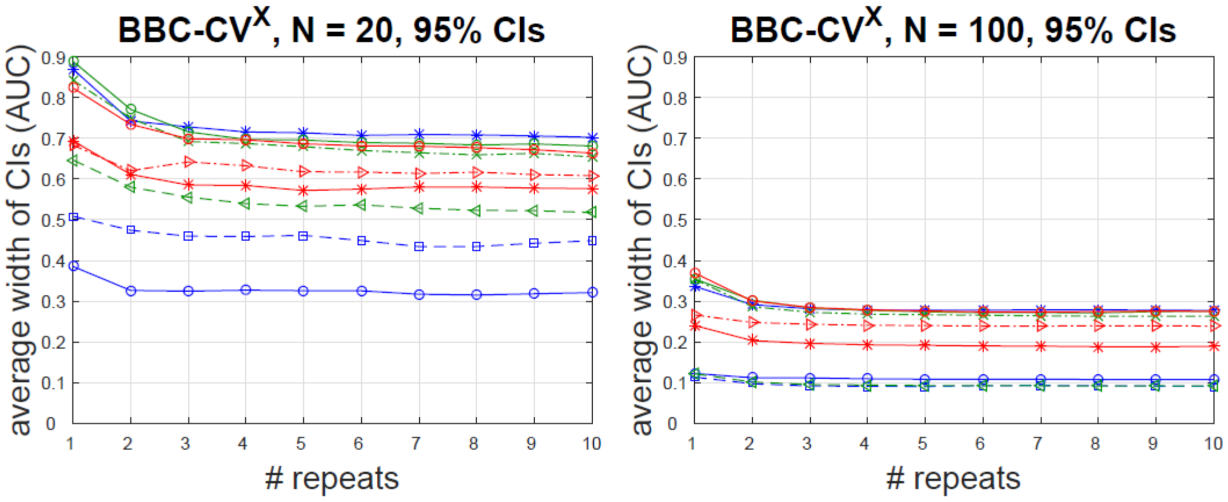


Average estimated bias (over 20 sub-datasets for each original dataset) of the **CVT, TT, NCV** and **BBC-CV** estimates of performance.

- **CVT** is optimistically biased for sample size N ≤ 100.
- **NCV** and **BBC-CV**, both have low bias though results vary with dataset.

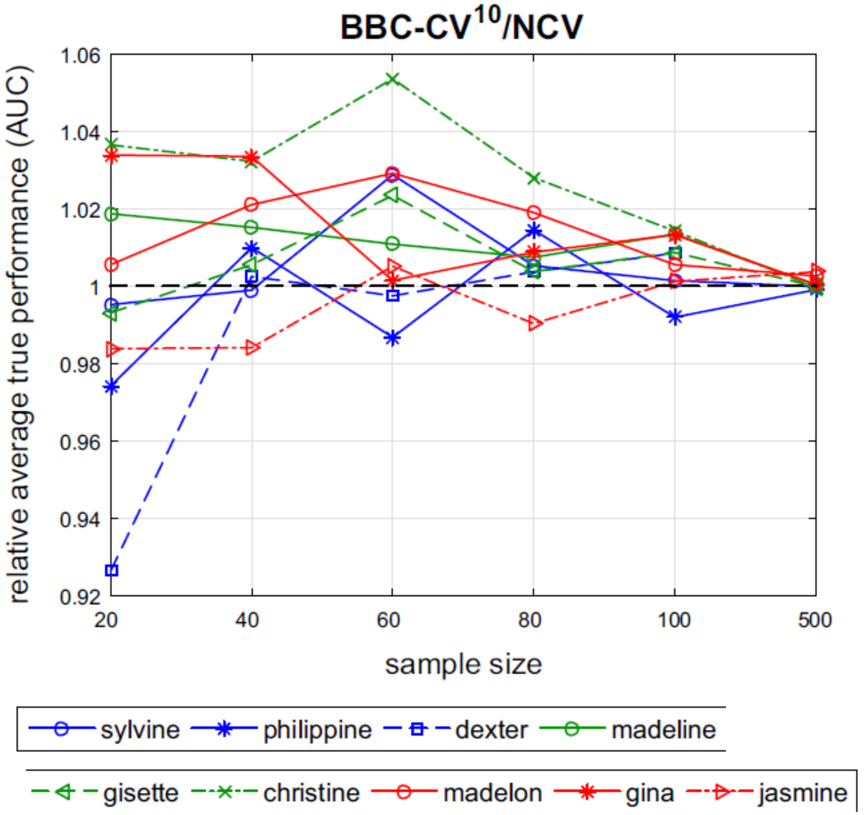
Multiple Repeats with Different Fold Partitions

- Different splits to folds, give different estimates
- **Repeat** the cross validation procedure with different splits
- Reduces **confidence intervals**
- **Improves selection** of best configuration



NCV vs Repeated BBC

- Is it better to use NCV with 10 folds or BBC with 10 Repeats?
  - same number of trained models
- BBC-CV^{10} returns on average better models for small sample sizes



BBC-CV

- **Pros**: Generally applicable to any type of data, any type of outcome, any performance metric
- **Pros**: Reduces complexity from $O(K^2)$ models per configuration to $O(K)$
- **Pros**: Generation of Confidence Intervals comes for free
- **Pros**: better than NCV for the same budget

- **Cons**: Requires a predetermined set of configurations; does not work with dynamic search strategies

BBCD: BBC with Dropping

- Do we really need to train models for all folds for all configurations?
- Can't we detect the inferior configurations with after just a few folds?
- And stop training further models?

BBCD: BBC with Dropping

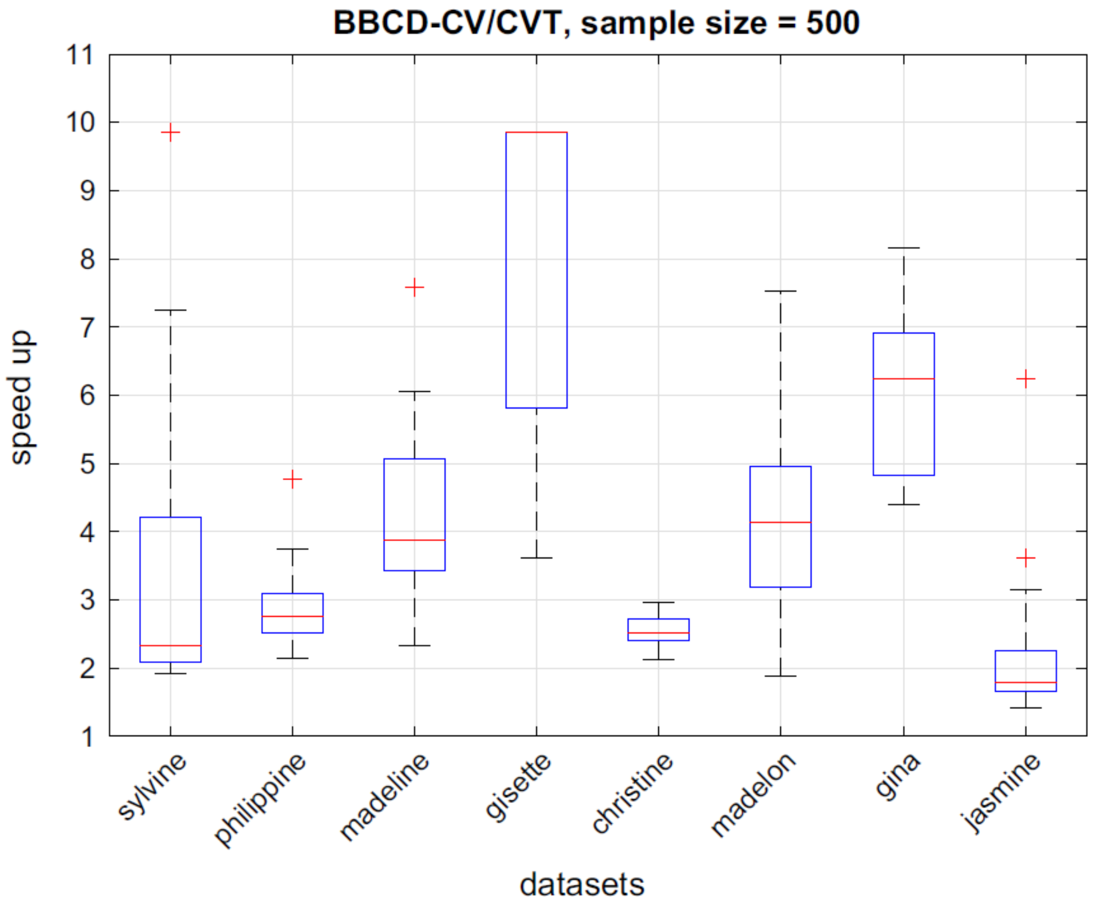
Test set prediction matrix

	Sample	f_1	f_2	….	f_n	y
Samples of fold F_1	1	yes	no	…	no	No
	2	no	yes	…	no	Yes
	3	Yes	Yes	…	No	No
Samples of fold F_2	4	(empty)	(empty)	…	(empty)	No
	…					Yes
						No
Samples of fold F_3						
	224					Yes

- After first CV iteration:
  - Training data all folds but F_1
  - Test data F_1
- Models produced with configuration f_1 seem inferior
- Perform a statistical test to determine inferiority
- If true, drop f_1 from subsequent CV iterations

BBCD

- Selects equally good models when samples size > 500
- Speed up of 5-6 times for 10-fold CV
- Total speed up vs. 10-fold NCV about 40-50



Practical advise for Tune-n-Estimate

- For **samples sizes < 250 per class** use **BBC** with multiple repeats
- For **sample 250 < sizes < 2000** use **BBCD**
- For larger sample sizes use hold-out

User Preferences

Preference Dimensions

Preference Dimensions

- Different analyses, different needs

Preference Dimensions

- Different analyses, different needs
- Not everybody cares only for predictive performance!

Preference Dimensions

- Different analyses, different needs
- Not everybody cares only for predictive performance!
- What are the different criteria for a successful analysis?

User Preferences Trade-Offs

- **Predictive performance**
  - Important for models to put in operational use
  - E.g., models for translational medicine
  - Use maximum number of folds, several repeats, more algorithms, more hyper-parameter values

User Preferences Trade-Offs

- **Knowledge Discovery** (in the form of Feature Selection)
  - Important when trying to get intuition into the mechanisms (causality) of the data generating mechanism
  - Or, when trying to reduce cost of measuring the features (E.g., models in molecular biology)
  - Try configurations with feature selection only
  - Try feature selection hyper-parameters that force the selection of few features

User Preferences Trade-Offs

- **Interpretability**
  - Important when gaining intuition how the features determine the outcome (E.g., medicine)
  - Enforce both feature selection and humanly-interpretable models
  - Generalized linear models
  - Decision Trees

User Preferences Trade-Offs

- **Speed of analysis**
  - Important for initial estimation of the information-value
  - Use fewer algorithms, fewer hyper-parameter values, less expensive algorithms

User Preferences Trade-Offs

- **Speed of Model Execution**
  - Important for real-time predictions (E.g., text classification models on a popular web-server)
  - Enforce only fast-executing models, e.g,. generalized linear models, decision trees

Trade-off estimation

- When restricting search to
  - only interpretable models
  - only with feature selection
  - only with fast-executing models

- Compare against the unrestricted search results to estimate the performance loss

Summary

Summary

- **Tuning** is very **important** for predictive performance

Summary

- **Tuning** is very **important** for predictive performance
- **Tuning** (trying multiple configurations)leads to **overestimations**; it requires special estimation protocols

Summary

- **Tuning** is very **important** for predictive performance
- **Tuning** (trying multiple configurations)leads to **overestimations**; it requires special estimation protocols
- **Beware of "choosing the best of"** decisions in general! Lead to overestimation!

Summary

- **Tuning** is very **important** for predictive performance
- **Tuning** (trying multiple configurations)leads to **overestimations**; it requires special estimation protocols
- **Beware of "choosing the best of"** decisions in general! Lead to overestimation!
- **NCV is the current standard**: it cross-validates a learner that CVs configurations and chooses the best

Summary

- **Tuning** is very **important** for predictive performance
- **Tuning** (trying multiple configurations)leads to **overestimations**; it requires special estimation protocols
- **Beware of "choosing the best of"** decisions in general! Lead to overestimation!
- **NCV is the current standard**: it cross-validates a learner that CVs configurations and chooses the best
- **BBC** bootstraps the configuration strategy

Summary

- **Tuning** is very **important** for predictive performance
- **Tuning** (trying multiple configurations)leads to **overestimations**; it requires special estimation protocols
- **Beware of “choosing the best of”** decisions in general! Lead to overestimation!
- **NCV is the current standard**: it cross-validates a learner that CVs configurations and chooses the best
- **BBC** bootstraps the configuration strategy
- **BBCD** drops early inferior configurations

Summary

- **Tuning** is very **important** for predictive performance
- **Tuning** (trying multiple configurations)leads to **overestimations**; it requires special estimation protocols
- **Beware of “choosing the best of”** decisions in general! Lead to overestimation!
- **NCV is the current standard**: it cross-validates a learner that CVs configurations and chooses the best
- **BBC** bootstraps the configuration strategy
- **BBCD** drops early inferior configurations
- **BBC and BBCD** a faster, better proposed alternative

Summary

- **Tuning** is very **important** for predictive performance
- **Tuning** (trying multiple configurations)leads to **overestimations**; it requires special estimation protocols
- **Beware of "choosing the best of"** decisions in general! Lead to overestimation!
- **NCV is the current standard**: it cross-validates a learner that CVs configurations and chooses the best
- **BBC** bootstraps the configuration strategy
- **BBCD** drops early inferior configurations
- **BBC and BBCD** a faster, better proposed alternative
- Different analysis preferences require adjusting the pipeline

Checkpoint: You should know

Checkpoint: You should know

- ✓ The concepts of hyper-parameters and configurations

Checkpoint: You should know

- ✓ The concepts of hyper-parameters and configurations
- ✓ The Golden Rule of estimation

Checkpoint: You should know

- ✓ The concepts of hyper-parameters and configurations
- ✓ The Golden Rule of estimation
- ✓ Common pitfalls of analysis:
  - ✓ Not CVing all steps of the analysis
  - ✓ Reporting the best of CVed performances

Checkpoint: You should know

- ✓ The concepts of hyper-parameters and configurations
- ✓ The Golden Rule of estimation
- ✓ Common pitfalls of analysis:
  - ✓ Not CVing all steps of the analysis
  - ✓ Reporting the best of CVed performances
- ✓ Grid Search in the space of hyper-parameters

Checkpoint: You should know

- ✓ The concepts of hyper-parameters and configurations
- ✓ The Golden Rule of estimation
- ✓ Common pitfalls of analysis:
  - ✓ Not CVing all steps of the analysis
  - ✓ Reporting the best of CVed performances
- ✓ Grid Search in the space of hyper-parameters
- ✓ NCV and BBC

Checkpoint: You should know

- ✓ The concepts of hyper-parameters and configurations
- ✓ The Golden Rule of estimation
- ✓ Common pitfalls of analysis:
  - ✓ Not CVing all steps of the analysis
  - ✓ Reporting the best of CVed performances
- ✓ Grid Search in the space of hyper-parameters
- ✓ NCV and BBC
- ✓ Should be enough to construct a basic, but quite general and **correct** automated pipeline

References

- C. E. Rasmussen & C. K. I. Williams. "Gaussian Processes for Machine Learning", the MIT Press, 2006
- I. Tsamardinos, E. Greasidou, G. Borboudakis, "Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation", **Machine Learning** 2018

End of Part II