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

# Conflict of Interest Declaration

- Some of the research and algorithmic results are commercially exploited by Gnosis Data Analysis PC

# Slides, Graphics, Visuals

- Kleanthi Lakiotaki
- Kleio-Maria Verrou

# 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

# 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

# Introduction to AutoML Tutorial

# What is Automated Machine Learning

- "***Automated machine learning*** *(**AutoML**) is the process of automating the end-to-end process of applying machine learning to real-world problems.*" Wikipedia
- In this tutorial focus on:
  - Predictive and Diagnostic Modeling (Supervised learning)
  - Feature Selection (Knowledge Discovery, Biosignature Discovery)
  - No Deep Learning
- Very hot area of research!

# AutoML

Input

Predictors / features

| ID | $x_1$ | $x_2$ | $x_3$ | $x_4$ | …. | $x_m$ | target |
|---|---|---|---|---|---|---|---|
| 1 | 26 | 0 | 0.3 | 0.06 | … | 2 | yes |
| 2 | 52 | 1 | 2.3 | 0.1 | … | 2 | no |
| … | … | … | … | … | … | … | |
| *n* | 34 | 0 | 5.8 | 0.04 | … | 3 | no |

# AutoML

## Input

Predictors / features

| ID | $x_1$ | $x_2$ | $x_3$ | $x_4$ | .... | $x_m$ | target |
|---|---|---|---|---|---|---|---|
| 1 | 26 | 0 | 0.3 | 0.06 | … | 2 | yes |
| 2 | 52 | 1 | 2.3 | 0.1 | … | 2 | no |
| … | … | … | … | … | … | … | |
| *n* | 34 | 0 | 5.8 | 0.04 | … | 3 | no |

instances

# AutoML

## Input

Predictors / features

| ID | $x_1$ | $x_2$ | $x_3$ | $x_4$ | …. | $x_m$ | target |
|---|---|---|---|---|---|---|---|
| 1 | 26 | 0 | 0.3 | 0.06 | … | 2 | yes |
| 2 | 52 | 1 | 2.3 | 0.1 | … | 2 | no |
| … | … | … | … | … | … | … | |
| *n* | 34 | 0 | 5.8 | 0.04 | … | 3 | no |

instances

## Auto-ML System

# AutoML

## Input

Predictors / features

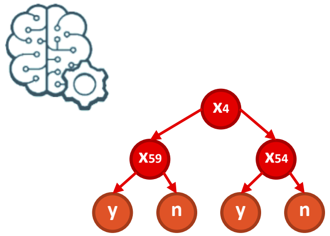
| ID | $x_1$ | $x_2$ | $x_3$ | $x_4$ | .... | $x_m$ | target |
|---|---|---|---|---|---|---|---|
| 1 | 26 | 0 | 0.3 | 0.06 | … | 2 | yes |
| 2 | 52 | 1 | 2.3 | 0.1 | … | 2 | no |
| … | … | … | … | … | … | … | |
| *n* | 34 | 0 | 5.8 | 0.04 | … | 3 | no |

instances

## Auto-ML System

## Output

**Model**



# Auto-ML problems

- Selection of algorithms
- Performance estimation
- Hyper-parameter optimization
- Feature Selection
- Generation of ML pipelines
- Detection of problems and pipeline execution monitoring
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- **Selection of algorithms**
- Performance estimation
- Hyper-parameter optimization
- Feature Selection
- Generation of ML pipelines
- Detection of problems and pipeline execution monitoring
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- **Selection of algorithms**
- **Performance estimation**
- Hyper-parameter optimization
- Feature Selection
- Generation of ML pipelines
- Detection of problems and pipeline execution monitoring
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- **Selection of algorithms**
- **Performance estimation**
- **Hyper-parameter optimization**
- Feature Selection
- Generation of ML pipelines
- Detection of problems and pipeline execution monitoring
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- **Selection of algorithms**
- **Performance estimation**
- **Hyper-parameter optimization**
- **Feature Selection**
- Generation of ML pipelines
- Detection of problems and pipeline execution monitoring
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- **Selection of algorithms**
- **Performance estimation**
- **Hyper-parameter optimization**
- **Feature Selection**
- **Generation of ML pipelines**
- Detection of problems and pipeline execution monitoring
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- **Selection of algorithms**
- **Performance estimation**
- **Hyper-parameter optimization**
- **Feature Selection**
- **Generation of ML pipelines**
- **Detection of problems and pipeline execution monitoring**
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- **Selection of algorithms**
- **Performance estimation**
- **Hyper-parameter optimization**
- **Feature Selection**
- **Generation of ML pipelines**
- **Detection of problems and pipeline execution monitoring**
- **Explanation, visualization, report writing**
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- **Selection of algorithms**
- **Performance estimation**
- **Hyper-parameter optimization**
- **Feature Selection**
- **Generation of ML pipelines**
- **Detection of problems and pipeline execution monitoring**
- **Explanation, visualization, report writing**
- **User interfaces**
- Meta-level learning, feature learning

# Auto-ML problems

- Selection of algorithms
- Performance estimation
- Hyper-parameter optimization
- Feature Selection
- Generation of ML pipelines
- Detection of problems and pipeline execution monitoring
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

# Auto-ML problems

- Selection of algorithms
- Performance estimation
- Hyper-parameter optimization
- Feature Selection
- Generation of ML pipelines
- Detection of problems and pipeline execution monitoring
- Explanation, visualization, report writing
- User interfaces
- Meta-level learning, feature learning

**Just automate the analysis of all data and send us home**

# Goals

- Improve your skills to write analysis scripts
  - Understand the trade-offs among choices
  - Avoid methodological errors and pitfalls
- Learn how to perform feature selection
- Obtain an introduction to the field, its problems and tools
- **Become a better analyst**

# Prerequisites

- Basics of supervised machine learning
  - Modeling algorithms, feature selection
  - Types of outcomes (classification, regression, etc.)
  - Performance metrics (accuracy, AUC, mean squared error)
- Experience with supervised analysis and model building

# Estimating Performance

SINGLE CONFIGURATION

# The Predictive and Diagnostic Modeling Problem

- Given **past examples of profiles** and their actual **outcome of interest**, learn a predictive or diagnostic model for **new, unseen**, profiles

# The Predictive and Diagnostic Modeling Problem

- Micro-array gene expressions
- Methylation of CpG cites
- Next Generation Sequencing mRNA peaks
- SNP
- Copy-number variations
- Proteomics (mass spectroscopy, LC, etc.)
- Metabolomics
- Flow-cytometry
- Mass-cytometry
- Text of biomedical documents
- Clinical and Medical Quantities
- Environmental exposure factors
- **Combinations of the above**

- Given **past examples of profiles** and their actual **outcome of interest**, learn a <u>predictive or diagnostic model</u> for **new, unseen**, profiles

# The Predictive and Diagnostic Modeling Problem

- Micro-array gene expressions
- Methylation of CpG cites
- Next Generation Sequencing mRNA peaks
- SNP
- Copy-number variations
- Proteomics (mass spectroscopy, LC, etc.)
- Metabolomics
- Flow-cytometry
- Mass-cytometry
- Text of biomedical documents
- Clinical and Medical Quantities
- Environmental exposure factors
- **Combinations of the above**

- Disease status (diagnosis)
- Response to treatment
- Phenotype
- Time to death, relapse, complication
- Properties of a document

- Given **past examples of profiles** and their actual **outcome of interest**, learn a <u>predictive or diagnostic model</u> for **new, unseen**, profiles

# Examples of Multivariate Predictive or Diagnostic Models

Rule-Based Model (Decision Tree)

If AFFX-BloC-5 is Overexpressed and

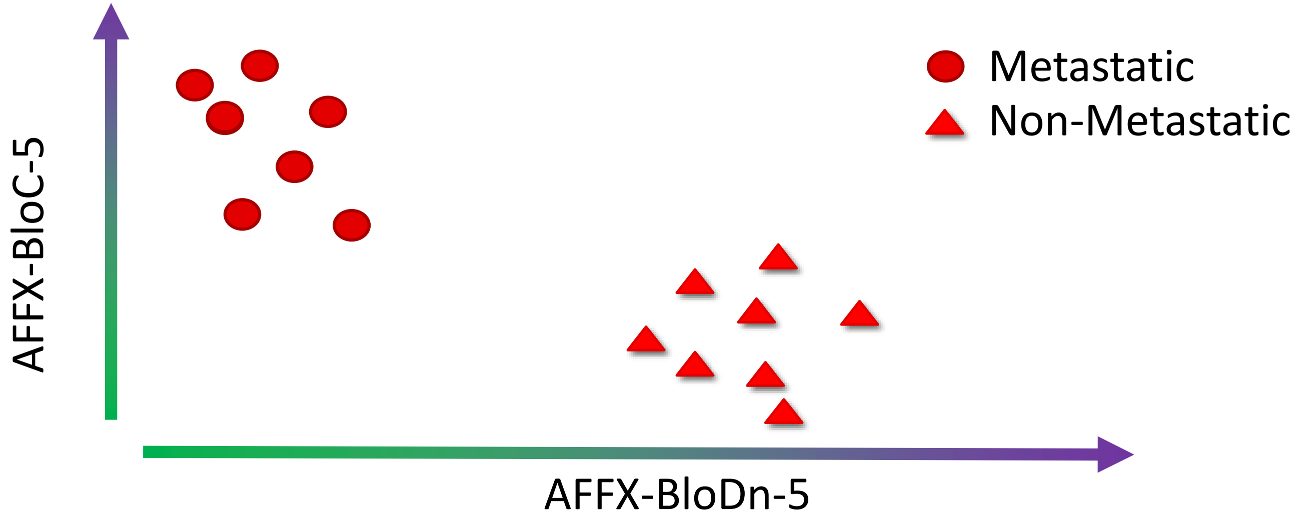
AFFX-BloDn-5 is Underexpressed

Then

Classify as Metastatic

Else

Classify as Non-Metastatic




Linear Model:

Metastatic = sign ( 0,5 × AFFX-BloC-5 – 0,5 × AFFX-BloDn-5 + 3)

Expression Values

| Sample | Genes / Probe Sets<br>AFFX-BloB-5_at | AFFX-BloB-M_at | AFFX-Blob-3_at | AFFX-BloC-5_at | ... | Affx-Bloc-3_at | AFFX-BloDn-5_at | Metastatic? |
|---|---|---|---|---|---|---|---|---|
| 1 | 123.00 | 1.00 | 2,3 | 12.00 | | 23.00 | 34.00 | Yes |
| 2 | 323.00 | 23.00 | 4,54 | 2.00 | | 21.00 | 65.00 | No |
| | | | | | | | | No |
| | | | | | | | | |
| | | | | | | | | No |
| N | 232.00 | 4,5 | 23.00 | 0,55 | | 75.00 | 343.00 | Yes |

# Examples of Multivariate Predictive or Diagnostic Models

Rule-Based Model (Decision Tree)

If AFFX-BloC-5 is Overexpressed and

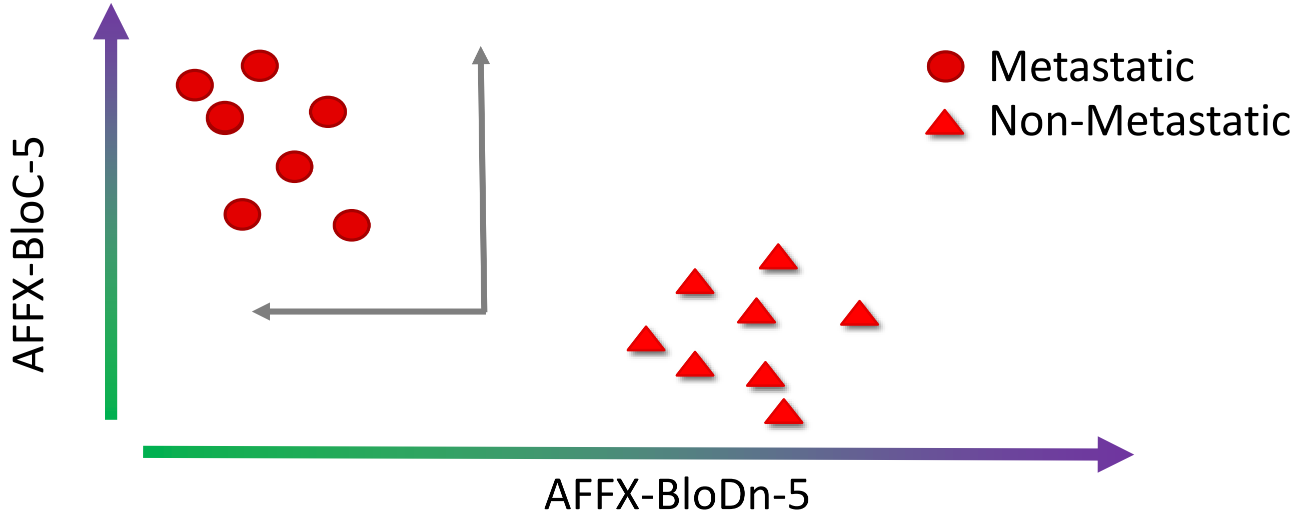
AFFX-BloDn-5 is Underexpressed

Then

Classify as Metastatic

Else

Classify as Non-Metastatic




Linear Model:

Metastatic = sign ( 0,5 × AFFX-BloC-5 – 0,5 × AFFX-BloDn-5 + 3)

Expression Values

| | | Genes / Probe Sets | | | | | | | Metastatic? |
|---|---|---|---|---|---|---|---|---|---|
| | | AFFX-BloB-5_at | AFFX-BloB-M_at | AFFX-Blob-3_at | AFFX-BloC-5_at | ... | Affx-Bloc-3_at | AFFX-BloDn-5_at | |
| Sample | 1 | 123.00 | 1.00 | 2,3 | 12.00 | | 23.00 | 34.00 | Yes |
| | 2 | 323.00 | 23.00 | 4,54 | 2.00 | | 21.00 | 65.00 | No |
| | | | | | | | | | No |
| | | | | | | | | | |
| | | | | | | | | | No |
| | N | 232.00 | 4,5 | 23.00 | 0,55 | | 75.00 | 343.00 | Yes |

# Examples of Multivariate Predictive or Diagnostic Models

Rule-Based Model (Decision Tree)

If AFFX-BloC-5 is Overexpressed and

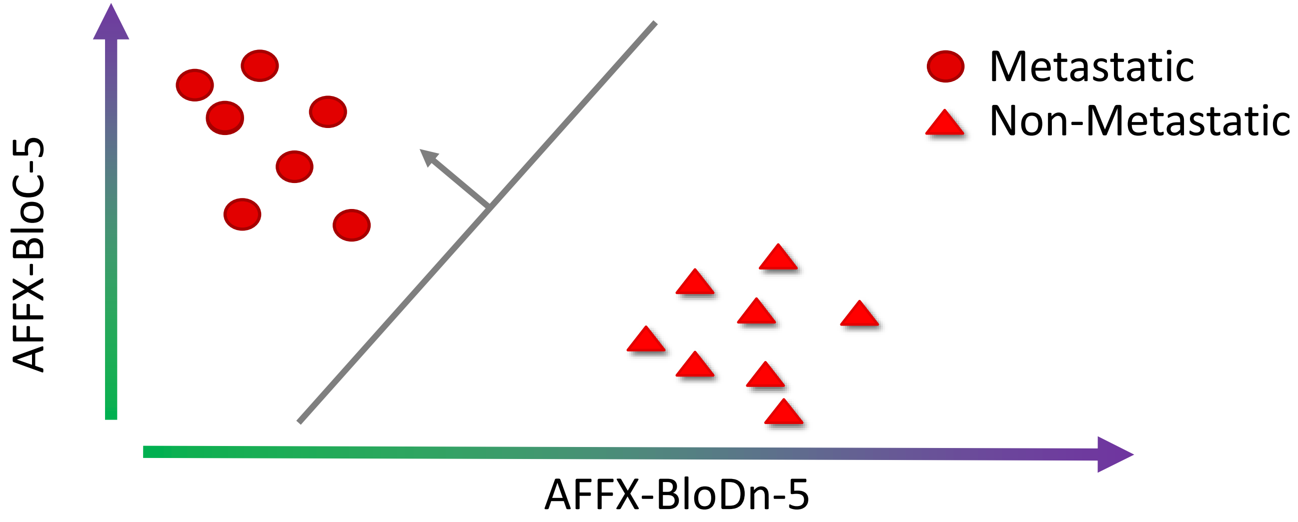
AFFX-BloDn-5 is Underexpressed

Then

Classify as Metastatic

Else

Classify as Non-Metastatic




Linear Model:

Metastatic = sign ( 0,5 × AFFX-BloC-5 – 0,5 × AFFX-BloDn-5 + 3)

**Expression Values**

| | **Genes / Probe Sets** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Sample** | **AFFX-BloB-5_at** | **AFFX-BloB-M_at** | **AFFX-Blob-3_at** | **AFFX-BloC-5_at** | **...** | **Affx-Bloc-3_at** | **AFFX-BloDn-5_at** | **Metastatic?** |
| 1 | 123.00 | 1.00 | 2,3 | 12.00 | | 23.00 | 34.00 | Yes |
| 2 | 323.00 | 23.00 | 4,54 | 2.00 | | 21.00 | 65.00 | No |
| | | | | | | | | No |
| | | | | | | | | |
| | | | | | | | | No |
| N | 232.00 | 4,5 | 23.00 | 0,55 | | 75.00 | 343.00 | Yes |

# Analysis Goals

**Given a dataset** $D = \{\langle \mathbf{x}_i, \mathrm{y}_i \rangle\}$ **in the form of a 2D matrix,** $\boldsymbol{x}_i$ the feature values, $y_i$ the true outcome.

1. **Produce** an **optimal** (diagnostic or predictive) model for operational use on future samples
2. **Estimate** the performance of the model
3. **Understand** which quantities are predictive (feature selection)

We have available a **single** learning method $\underline{f(D)}$ that returns models $M$

$y = M(x)$ returns predictions y for a sample x

# Ideal Performance Estimation

# Ideal Performance Estimation

1. Learn a model from samples, true outcomes in $\boldsymbol{D}$ **(train-set)**

# Ideal Performance Estimation

1. Learn a model from samples, true outcomes in $\boldsymbol{D}$ **(train-set)**
2. Install the model in its **intended operational environment**

# Ideal Performance Estimation

1. Learn a model from samples, true outcomes in $\boldsymbol{D}$ **(train-set)**
2. Install the model in its **intended operational environment**
3. Observe its operation for some time, for new cases $\mathbf{D'}$

# Ideal Performance Estimation

1. Learn a model from samples, true outcomes in $D$ **(train-set)**
2. Install the model in its **intended operational environment**
3. Observe its operation for some time, for new cases $\mathbf{D'}$
4. Label with a gold-standard y′ the cases in $\mathbf{D'}$ **(test-set)**

# Ideal Performance Estimation

1. Learn a model from samples, true outcomes in ***D*** **(train-set)**
2. Install the model in its **intended operational environment**
3. Observe its operation for some time, for new cases $\mathbf{D'}$
4. Label with a gold-standard y′ the cases in $\mathbf{D'}$ **(test-set)**
5. Estimate the performance of the model on $\mathbf{D'}$

# Ideal Performance Estimation

1. Learn a model from samples, true outcomes in $\boldsymbol{D}$ **(train-set)**
2. Install the model in its **intended operational environment**
3. Observe its operation for some time, for new cases $\mathbf{D'}$
4. Label with a gold-standard y′ the cases in $\mathbf{D'}$ **(test-set)**
5. Estimate the performance of the model on $\mathbf{D'}$

- Pros and cons?

# Simulating the Ideal

**Golden Rule:**

> Simulate: learn model from *D*, make operational, test on new samples D′

- Main point: **all decisions are made before model becomes operational and obtain D′**

# What can go wrong?

- Assumes the data distribution remains the same in the operational environment
- Example of violation:
  - Learning from case-control data (50-50% class distribution)
  - Then apply to general population (not 50-50% class distribution)
- Some performance metrics such as AUC, and Concordance-Index are invariant to class distribution changes; accuracy is not

# Why not estimate on the training set?

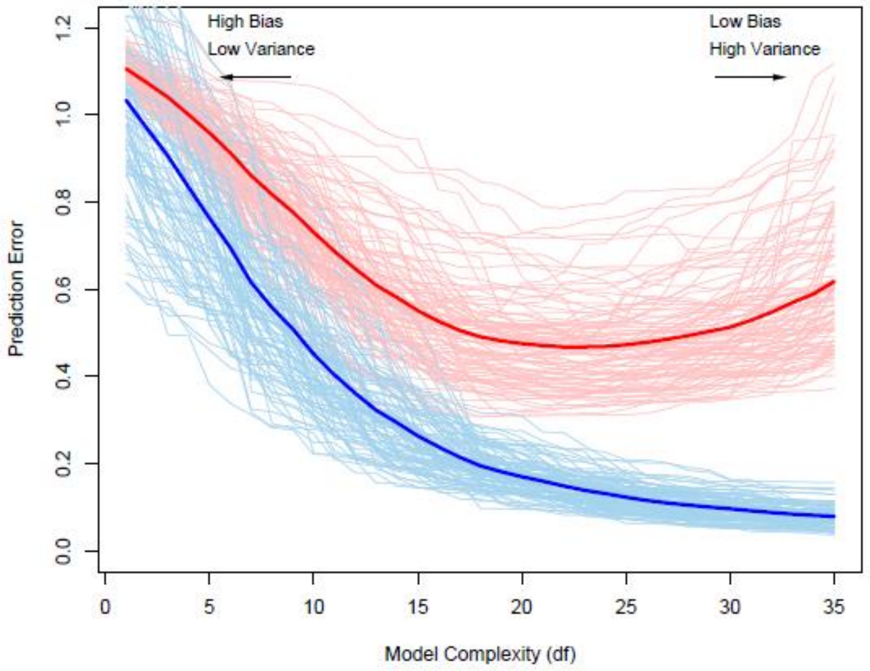
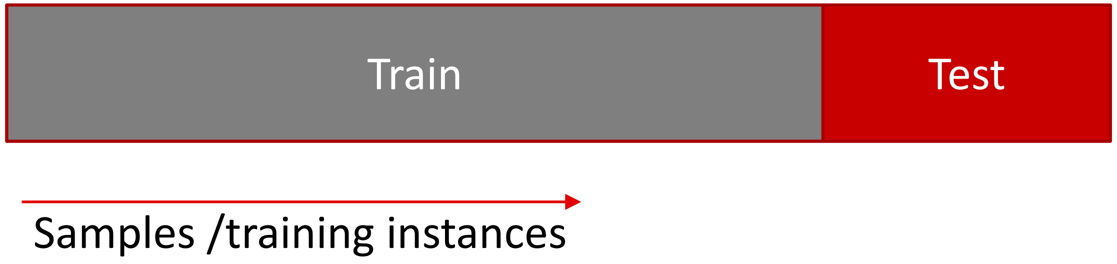



**FIGURE 7.1.** *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\mathrm{err}}$*, while the light red curves show the conditional test error* $\mathrm{Err}_{\mathcal{T}}$ *for 100 training sets of size 50 each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\mathrm{E}[\overline{\mathrm{err}}]$.

# Out and In Sample Estimators

- **Out-of-sample estimation protocols**
  - Employ the predictive performance of the model on data not seen by the learning method; ignore errors in training data
- **In-sample estimation protocols** (not covered)
  - (Also) employ the predictive performance of the model on the training data
  - Typically, they also penalize for complexity
  - Often, they only bound the performance
  - Bounds by Vapnik-Chervonenkis dimension theory

# Simulating the Ideal




- **Randomly** partition original data in terms of samples
- Learn on Train
- Estimate performance on Test
- Called hold-out estimation

# Notation

- Dataset is $D$, predictors in matrix $X$, outcome in $y$
- Rows correspond to samples, columns to features
- $X(\mathrm{indexset})$, $\mathrm{y}(\mathrm{indexset})$: selects only the **rows** of the indexset
- $l(y, p)$ the loss (error) between predictions in $p$ and true outcomes in $y$

# Hold-Out Protocol

| Train | Test |
|---|---|

**Hold-Out (Data D)**

Randomly partition row indexes to TrainIndex, TestIndex

$M = f(D(TrainIndex))$

<u>Returned Model</u>

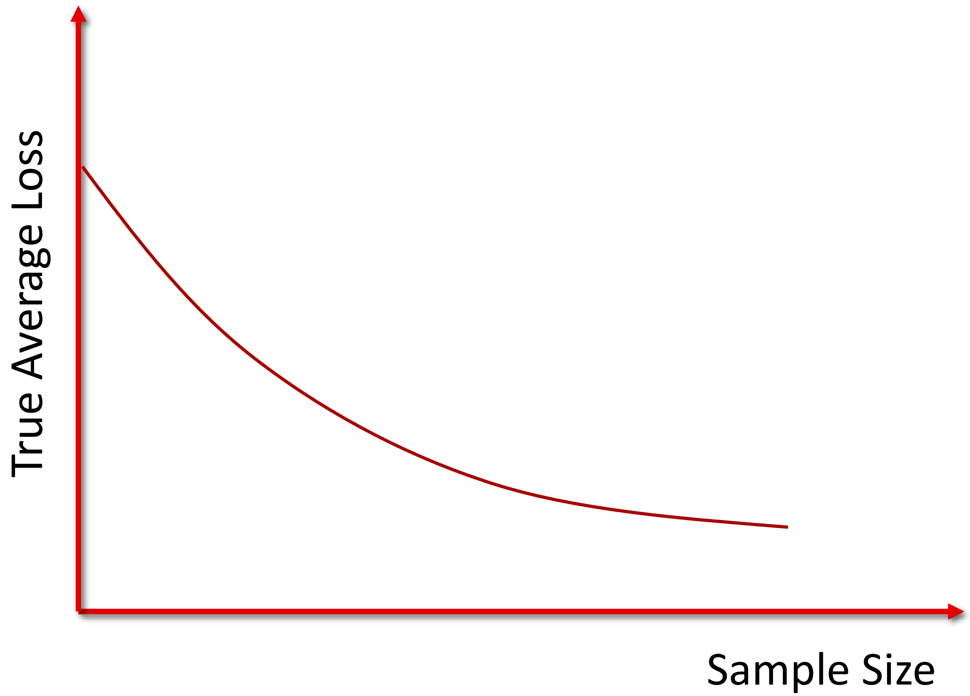
$M$

<u>Returned Estimation</u>

$l(y(TestIndex), M(X(TestIndex)))$

- Pros: simple, computationally efficient, and correct
- Pros: appropriate when data are plenty
- Cons: **some data are "lost" to estimation**

# Learning Curve

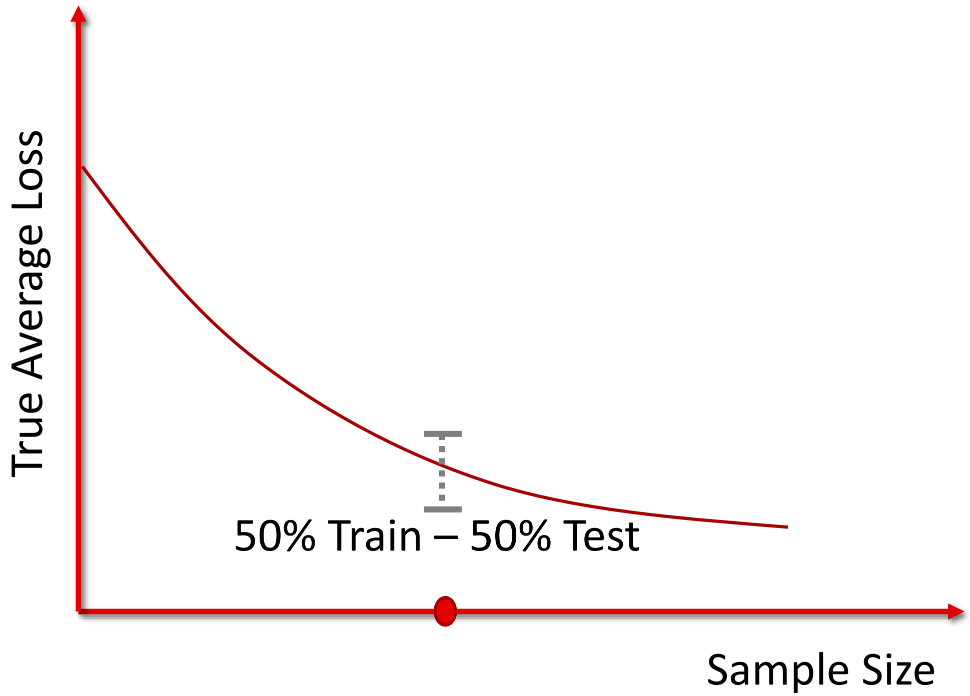



Variance (uncertainty) of estimation

Variance due to:

- Random sampling of the dataset from the whole population
- Random partition to train and test
- Size of test set

# Learning Curve

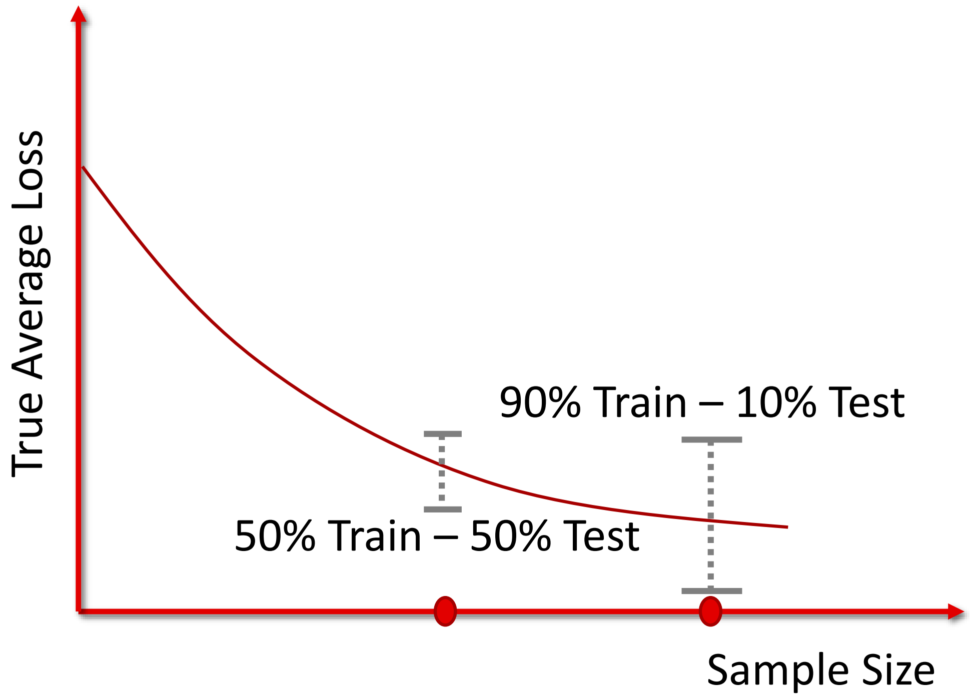



Variance (uncertainty) of estimation

Variance due to:

- Random sampling of the dataset from the whole population
- Random partition to train and test
- Size of test set

# Learning Curve

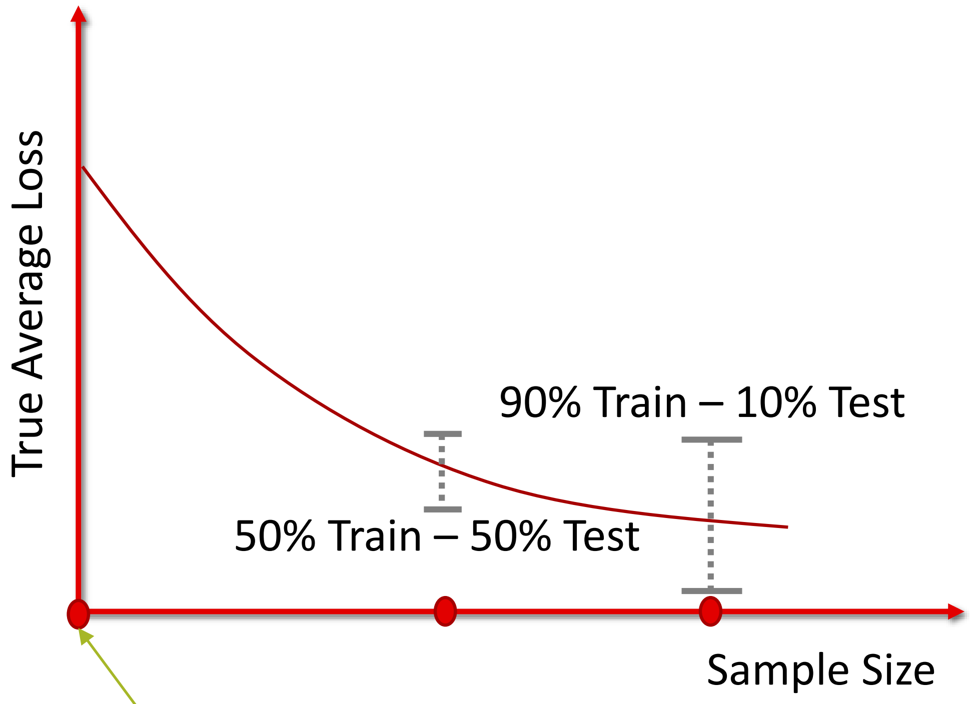




Variance due to:

- Random sampling of the dataset from the whole population
- Random partition to train and test
- Size of test set

# Learning Curve



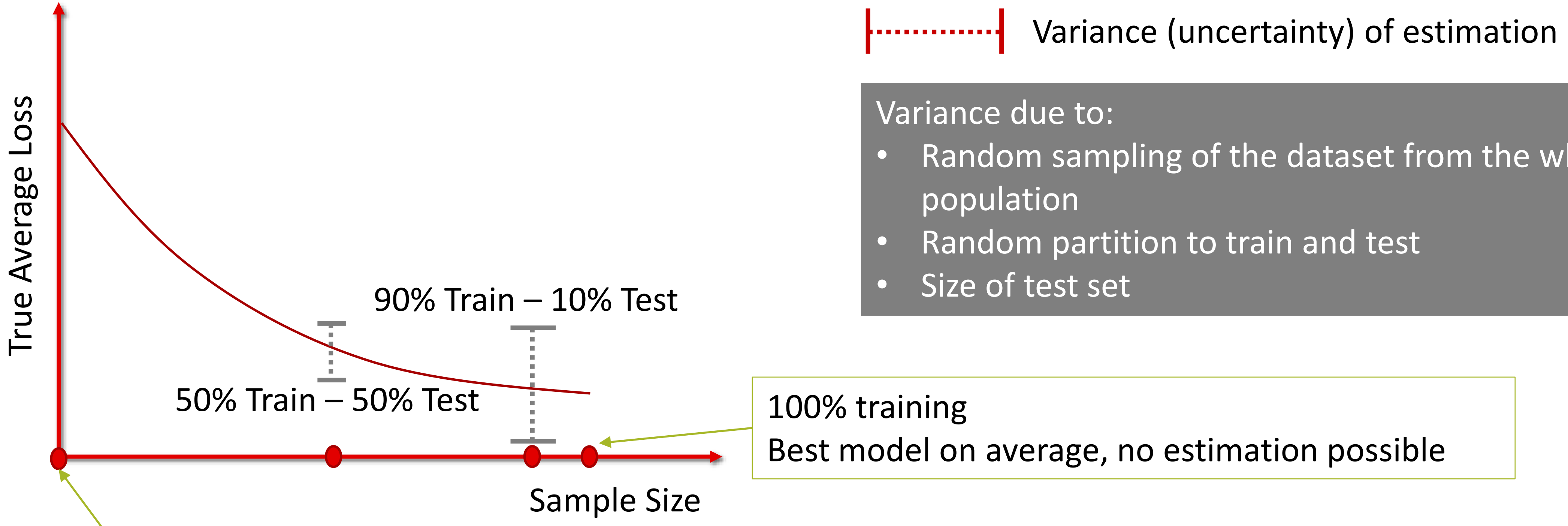

0% training
Worst model on average, best estimation possible

Variance (uncertainty) of estimation

Variance due to:

- Random sampling of the dataset from the whole population
- Random partition to train and test
- Size of test set

# Learning Curve

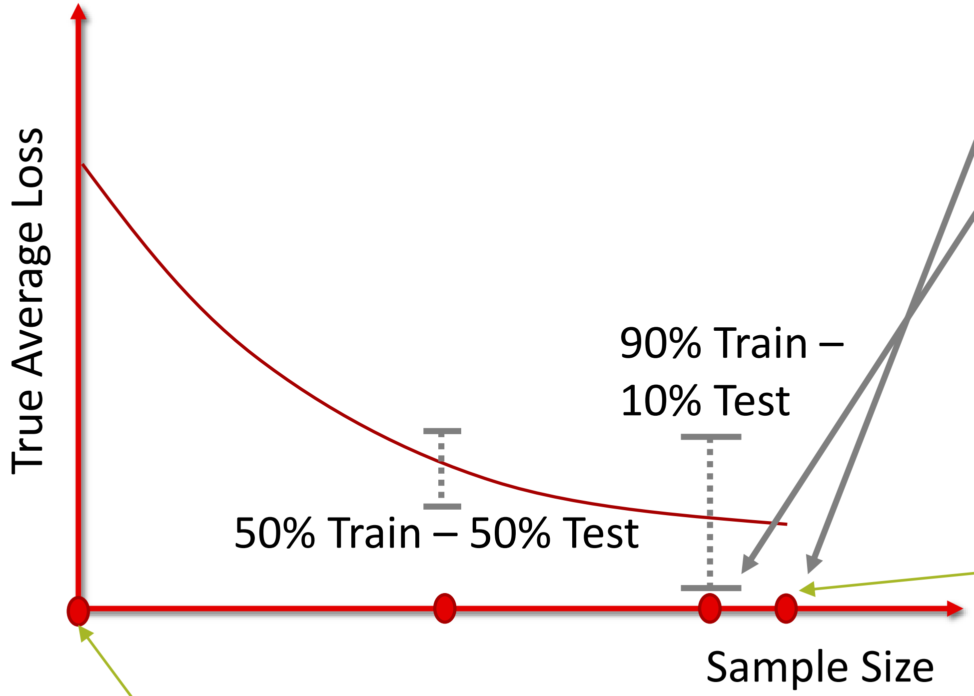



Variance due to:
- Random sampling of the dataset from the whole population
- Random partition to train and test
- Size of test set

100% training
Best model on average, no estimation possible

0% training
Worst model on average, best estimation possible

# Learning Curve




Return best possible model (on average) using **all data as training**
Use the estimation of some other model as a proxy

100% training
Best model on average, no estimation possible

0% training
Worst model on average, best estimation possible

# Hold-Out-New Protocol

| Train | Test |
| --- | --- |

**Hold-Out-New (Data D)**

Randomly partition row indexes to TrainIndex, TestIndex

$M = f(D(TrainIndex))$

$M_{all} = f(D)$

<u>Returned Model</u>

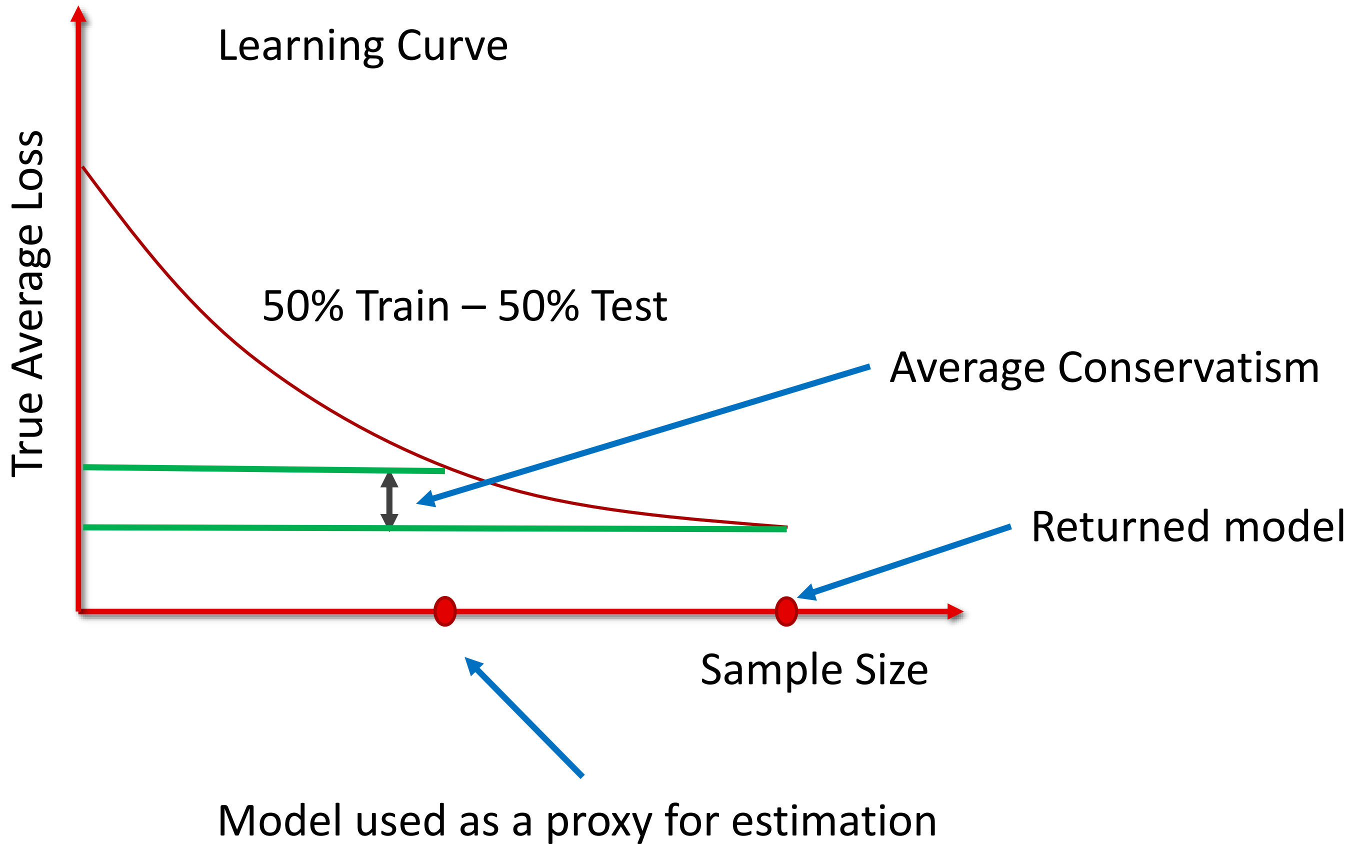
$\boldsymbol{M_{all}}$

<u>Returned Estimation</u>

$l(y(TestIndex), M_{train}(X(TestIndex)))$

- Trains 2 models, instead of 1
- Estimation is **conservative** on average

# Conservatism



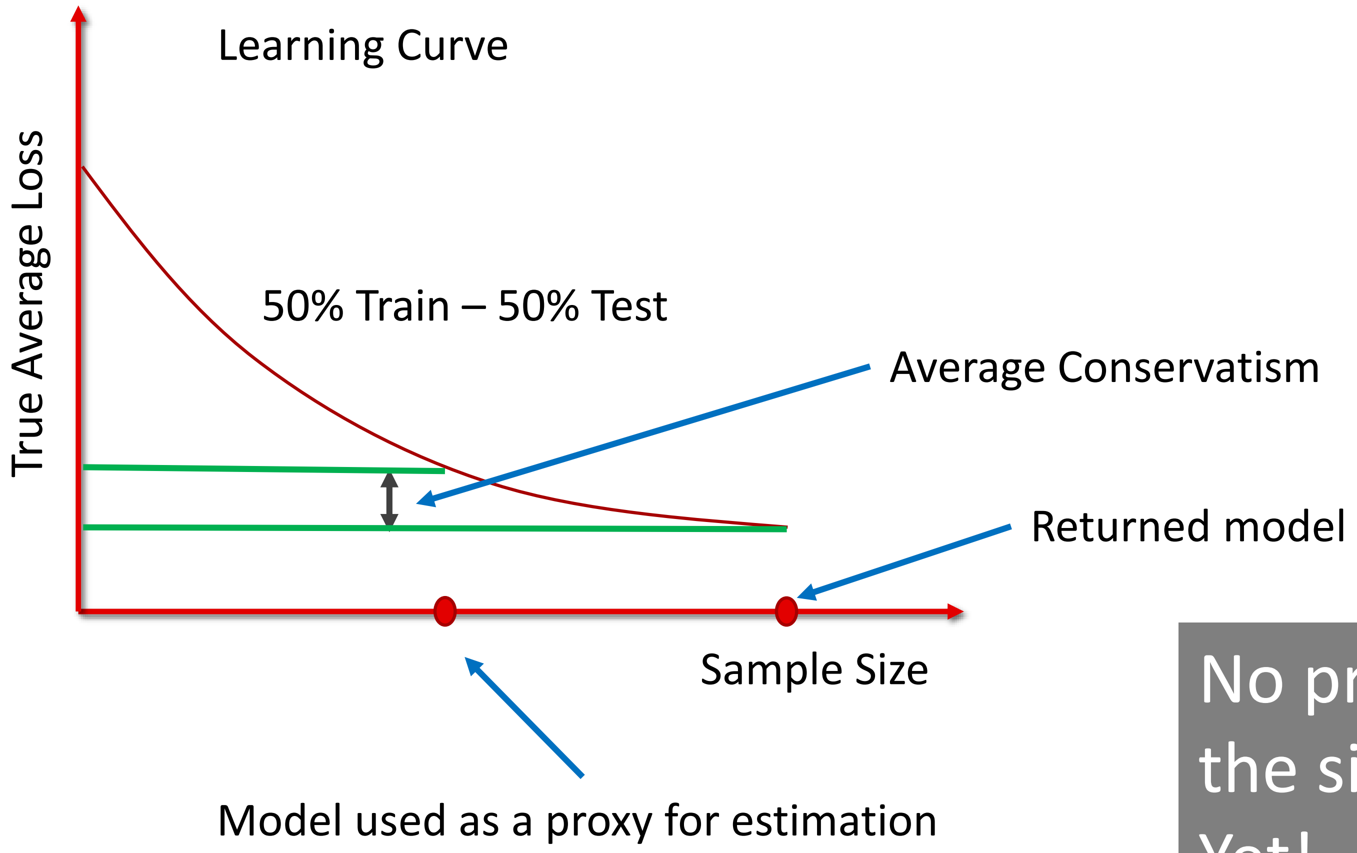

Small Train –Large Test

Estimate more conservative

Estimate more reliable

Typical splits: Train set is 66%, 75%, 80%, 90% of the data

# Conservatism




Small Train –Large Test

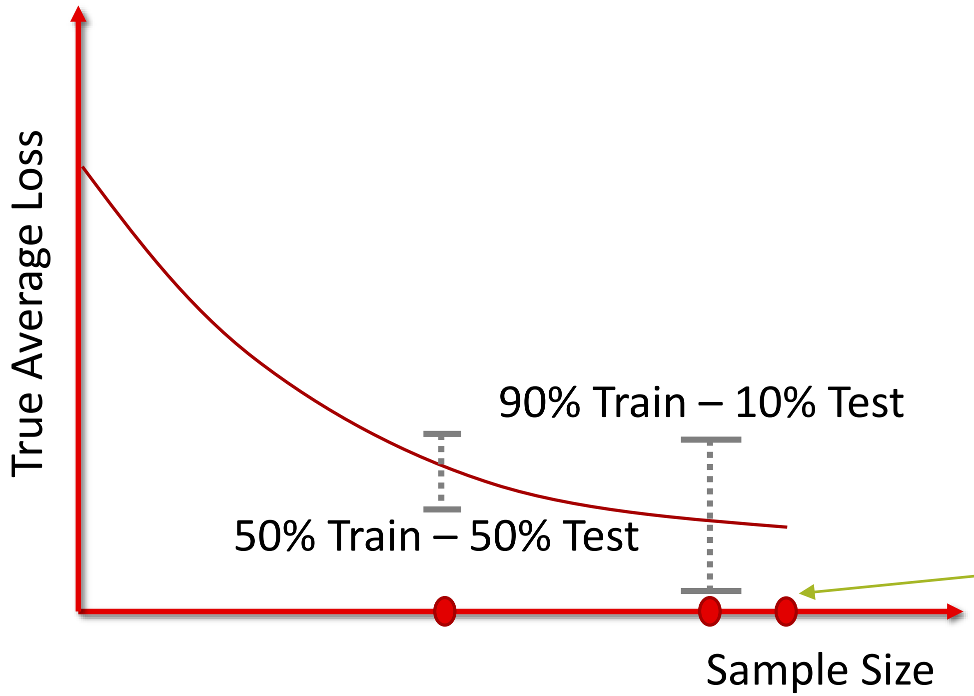
Estimate more conservative

Estimate more reliable

Typical splits: Train set is 66%, 75%, 80%, 90% of the data

No principled method for choosing the size of the test set.
Yet!

# Learning Curve



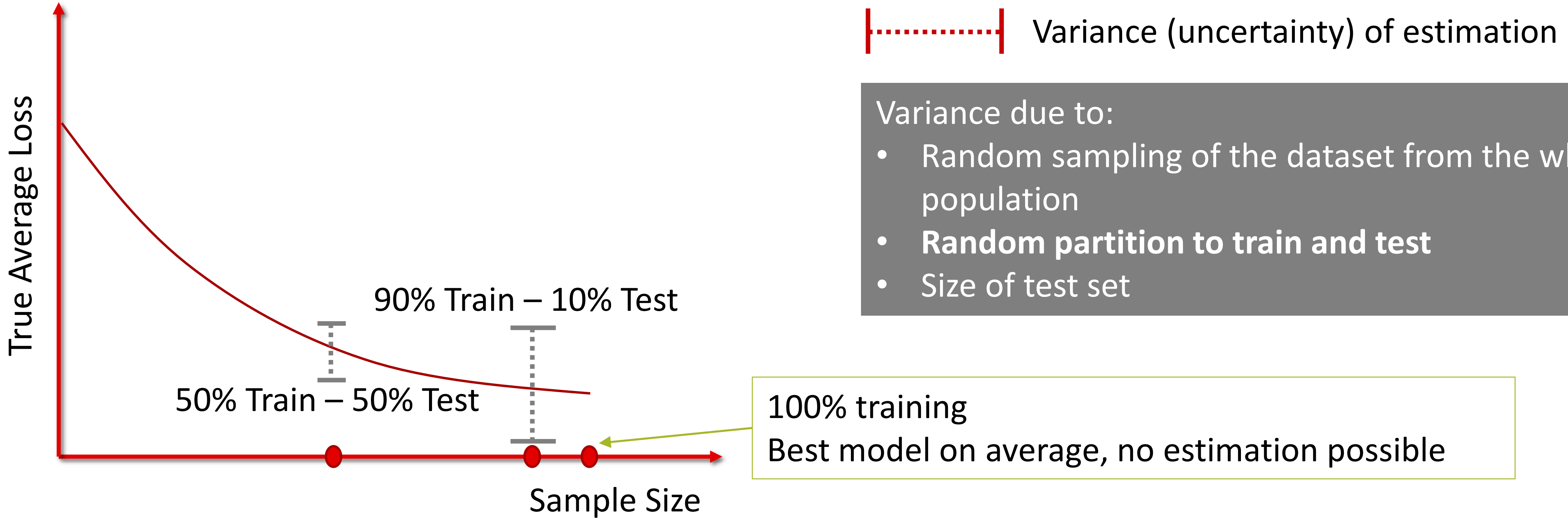

Variance (uncertainty) of estimation

Variance due to:

- Random sampling of the dataset from the whole population
- **Random partition to train and test**
- Size of test set

100% training
Best model on average, no estimation possible

# Learning Curve




Variance due to:
- Random sampling of the dataset from the whole population
- **Random partition to train and test**
- Size of test set

100% training
Best model on average, no estimation possible

**Repeat the process several times and average out**

# Hold-Out Protocol


**Repeated Hold-Out (Data D, nrepetitions)**

**For** r = 1 to nrepetitions

Randomly partition row indexes to TrainIndex, TestIndex

$M = f(D(TrainIndex))$

$l_r = l(y(TestIndex), Mtrain(X(TestIndex)))$

**End For**

$M_{all}=f(D)$

<u>Returned Model</u>

$M_{all}$

$l = 1/nrepetitions\ \Sigma l_r$

- Trains nrepetitions+1 models
- Simulates the Golden Rule several times
- Reduces the uncertainty of estimation
- Still conservative estimation

# Perspective Shift

- Hold-Out:
  - Returns model $M_{Train}$
  - Estimates its performance by applying **the same** $M_{Train}$ to test data
- Repeated Hold-Out and Hold-Out-New
  - Returns model $M_{all}$
  - Applies **other models** $M_{train}$ to estimate performance!
- What just happened?

# Perspective Shift

- Hold-Out estimates the performance of **the actual model** $M_{Train}$ to use operationally
- Repeated Hold-Out estimates the performance of the **learning method** $f$ that will produce the final model
- **Perspective shift:** from estimating the performance of a specific model to estimating the performance of a learning method

# K-Fold Cross Validation

Each repetition of **Repeated Hold-Out** produces a set of predictions of a model produced by $f$ on a test set

**Fact:** the uncertainty of estimation is reduced the most, when these predictions are on independent samples

Random partitioning to Train-n-Test produces overlapping test sets …

# K-Fold Cross Validation

Each repetition of **Repeated Hold-Out** produces a set of predictions of a model produced by *f* on a test set

**Fact:** the uncertainty of estimation is reduced the most, when these predictions are on independent samples

Random partitioning to Train-n-Test produces overlapping test sets …

When re-partitioning **force test sets to be disjoint** and cover all samples

K-Fold Cross-Validation = Repeated Hold-Out with *K* disjoint test sets covering the full dataset


**Algorithm 1** **CV**($f$, $D = \{F_1, \ldots, F_K\}$): Basic K-Fold Cross-Validation

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

**Output**: Model $M$, Performance estimation $L_{CV}$, out-of-sample predictions $\Pi$ on all folds

Define $D_{\setminus i} \leftarrow D \setminus F_i$
// Obtain the indexes of each fold
$I_i \leftarrow indexes(F_i)$
// Final Model trained by $f$ on all available data
$M \leftarrow f(D)$
// Performance estimation: learn from $D_{\setminus i}$, estimate on $F_i$
$L_{CV} \leftarrow \frac{1}{K} \sum_{i=1}^{K} l(y(I_i), f(F_i, D_{\setminus i}))$
// Out-of-sample predictions are used by bias-correction methods
Collect out-of-sample predictions $\Pi = [f(F_1, D_{\setminus 1}); \cdots ; f(F_K, D_{\setminus K})]$
**Return** $\langle M, L_{CV}, \Pi \rangle$

Model learnt from $D_{\setminus i}$ applied on fold $F_i$

# K-Fold Cross Validation

- Trains $K+1$ models
- As always: best model to use operationally is the one trained on all data!
- Still **conservative**: estimates the performance of the average model produced by $f$ on training sets of size $N = S\,(1 - 1 / K\,)$, $S$ the total sample size
- Typical values for $K = 3, 5, 10$, or maximum $S$ called **Leave-One-Out Cross-Validation** or **LOO CV**

# Cross-Validation Variants

- Can I further reduce the variance of estimation?
  - Yes! There is still variance due to the specific partitioning to folds.
  - **Repeated Cross-Validation**: repeat CV with many partitions to folds and average. Use as many repetitions as possible! <u>It works, it's important for small sample sizes</u>.
- I only have time for K=3, but leaving out 33% of the data each time is too much!
  - Partition to K=10 (or whatever) and perform only the first 3 iterations of the Cross-Validation
  - **Incomplete Cross-Validation**

# Failure of Cross-Validation

- Leave-One-Out CV should be the least conservative, less variant estimate, but …

# Failure of Cross-Validation

- Leave-One-Out CV should be the least conservative, less variant estimate, but …
- There is evidence that LOO-CV is not always the best [Kohavi, R. 1995]

# Failure of Cross-Validation

- Leave-One-Out CV should be the least conservative, less variant estimate, but …
- There is evidence that LOO-CV is not always the best [Kohavi, R. 1995]
- Example: 25 positives and 25 negative samples. Classifier learns to predict the majority class in the training data. Question: what's the estimate of accuracy of LOO-CV?

# Failure of Cross-Validation

- Leave-One-Out CV should be the least conservative, less variant estimate, but …
- There is evidence that LOO-CV is not always the best [Kohavi, R. 1995]
- Example: 25 positives and 25 negative samples. Classifier learns to predict the majority class in the training data. Question: what's the estimate of accuracy of LOO-CV?
- Answer: 0% ! A complete break down

# Failure of Cross-Validation

- Leave-One-Out CV should be the least conservative, less variant estimate, but …
- There is evidence that LOO-CV is not always the best [Kohavi, R. 1995]
- Example: 25 positives and 25 negative samples. Classifier learns to predict the majority class in the training data. <u>Question</u>: what's the estimate of accuracy of LOO-CV?
- <u>Answer</u>: 0% ! A complete break down
- Leave-one-Out forces an extreme difference between the class distribution in the original dataset and each test set
- Test sets without any samples from some classes maybe problematic.

# Stratified Cross-Validation

- Randomly split to folds, while maintaining the distribution of the classes as close as possible to the one in the full dataset
- Highly recommended when some classes are rare
- **Suggestion**: All folds should have at least 1 sample from each class, thus <u>K is at most #samples-of-rarest-class</u>
- For **regression**, similar ideas should be applied (e.g., partition to folds with the same **variance** as the original dataset)

# Personal Advise

- For a **single learning** method, when sample size is low and computational time is no issue use:
  - **Stratified, Repeated K-fold Cross Validation**
  - **K = #samples-of-rarest-class** (each fold has samples from all classes)

# Pitfalls of Cross-Validation

**Golden Rule:**

Simulate: learn model from $D$, make operational, test on new samples $D'$

- Scale data so that each variable has zero mean and standard deviation of 1
- Remove variables independent of the target
- $\langle \textit{model}, \textit{estimate} \rangle = \text{Cross-Validation}(f, \mathbf{D})$
- Claim to the reviewers that *model* is expected to have loss *estimate*

# Pitfalls of Cross-Validation

It peeks in the test cases!!!

**Golden Rule:**

<u>Simulate: learn model from $D$, make operational, test on new samples D′</u>

- Scale data so that each variable has zero mean and standard deviation of 1
- Remove variables independent of the target
- $\langle \textit{model}, \textit{estimate} \rangle = \text{Cross-Validation}(f, \mathbf{D})$
- Claim to the reviewers that <u>*model*</u> is expected to have loss <u>*estimate*</u>

# Pitfalls of Cross-Validation

It peeks in the test cases!!!

**Golden Rule:**

Simulate: learn model from $D$, make operational, test on new samples D′

- Scale data so that each variable has zero mean and standard deviation of 1
- Remove variables independent of the target
- ⟨*model, estim*
- Claim to the … to have loss *estimate*

Scaling and variable selection is part of the learning method; they **also** have to be CVed

# Correct CV

**Algorithm 1** **CV**($f$, $D = \{F_1, \ldots, F_K\}$): Basic K-Fold Cross-Validation

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

**Output**: Model $M$, Performance estimation $L_{CV}$, out-of-sample predictions $\Pi$ on all folds

```
Define $D_{\setminus i} \leftarrow D \setminus F_i$
// Obtain the indexes of each fold
$I_i \leftarrow indexes(F_i)$
// Final Model trained by $f$ on all available data
$M \leftarrow f(D)$
// Performance estimation: learn from $D_{\setminus i}$, estimate on $F_i$
$L_{CV} \leftarrow \frac{1}{K} \sum_{i=1}^{K} l(y(I_i), f(F_i, D_{\setminus i}))$
// Out-of-sample predictions are used by bias-correction methods
Collect out-of-sample predictions $\Pi = [f(F_1, D_{\setminus 1}); \cdots; f(F_K, D_{\setminus K})]$
Return $\langle M, L_{CV}, \Pi \rangle$
```

The learner *f* is creating a new function (model) with several steps. This is easier in languages where functions are first class objects, e.g., R, Matlab, python, but not C

**f(Data Train)**

1. Normalize **Train**, store normalizing parameters **normpar**
2. Identify the most important variable-set **S** from **Train**
3. Project **Train** on **S** only
4. Learn a decision tree TR from **Train** data
5. Return a model **M(x)**
   - Normalizes **x** according to **normpar**
   - Retain only variables S from vector **x**
   - Return the output of TR on (modified vector) **x**

# Correct CV

**Learning function containing *all steps***

---

**Algorithm 1** **CV**($f$, $D = \{F_1, \ldots, F_K\}$): Basic K-Fold Cross-Validation

---

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

**Output**: Model $M$, Performance estimation $L_{CV}$, out-of-sample predictions $\Pi$ on all folds

```
Define D_{\i} ← D \ F_i
// Obtain the indexes of each fold
I_i ← indexes(F_i)
// Final Model trained by f on all available data
M ← f(D)
// Performance estimation: learn from D_{\i}, estimate on F_i
L_CV ← (1/K) Σ_{i=1}^{K} l(y(I_i), f(F_i, D_{\i}))
// Out-of-sample predictions are used by bias-correction methods
Collect out-of-sample predictions Π = [f(F_1, D_{\1}); ··· ; f(F_K, D_{\K})]
Return ⟨M, L_CV, Π⟩
```

---

**f(Data Train)**

1. Normalize **Train**, store normalizing parameters **normpar**
2. Identify the most important variable-set **S** from **Train**
3. Project **Train** on **S** only
4. Learn a decision tree TR from **Train** data
5. Return a model **M(x)**
   - Normalizes **x** according to **normpar**
   - Retain only variables S from vector **x**
   - Return the output of TR on (modified vector) **x**

The learner *f* is creating a new function (model) with several steps. This is easier in languages where functions are first class objects, e.g., R, Matlab, python, but not C

# Correct CV

**Algorithm 1** **CV**($f$, $D = \{F_1, \ldots, F_K\}$): Basic K-Fold Cross-Validation

**Input**: Learning method $f$, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into about equally-sized folds $F_i$
**Output**: Model $M$, Performance estimation $L_{CV}$, out-of-sample predictions $\Pi$ on all folds

Define $D_{\setminus i} \leftarrow D \setminus F_i$
// Obtain the indexes of each fold
$I_i \leftarrow indexes(F_i)$
// Final Model trained by $f$ on all available data
$M \leftarrow f(D)$
// Performance estimation: learn from $D_{\setminus i}$, estimate on $F_i$
$L_{CV} \leftarrow \frac{1}{K} \sum_{i=1}^{K} l(y(I_i), f(F_i, D_{\setminus i}))$
// Out-of-sample predictions are used by bias-correction methods
Collect out-of-sample predictions $\Pi = [f(F_1, D_{\setminus 1}); \cdots; f(F_K, D_{\setminus K})]$
**Return** $\langle M, L_{CV}, \Pi \rangle$

Learning function containing *all steps*

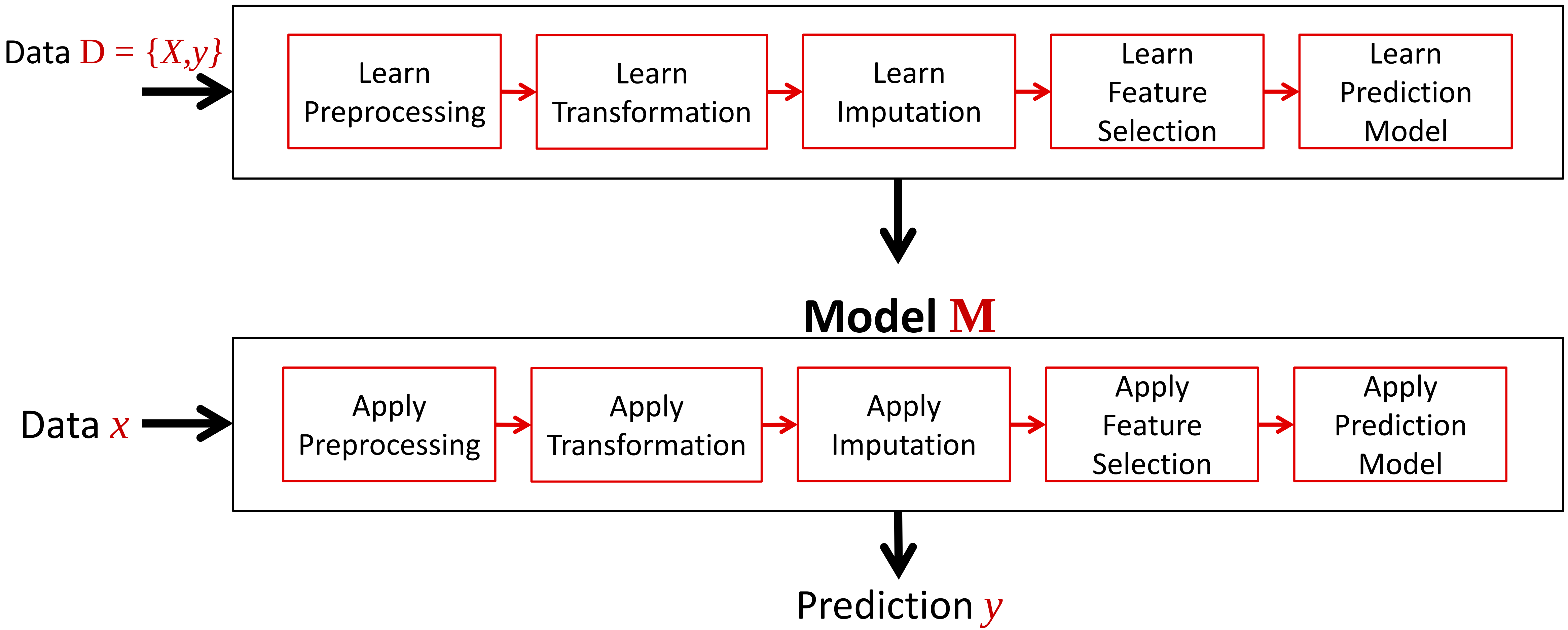
**f(Data Train)**

1. Normalize **Train**, store normalizing parameters **normpar**
2. Identify the most important variable-set **S** from **Train**
3. Project **Train** on **S** only
4. Learn a decision tree TR from **Train** data
5. Return a model **M(x)**
   - Normalizes **x** according to **normpar**
   - Retain only variables S from vector **x**
   - Return the output of TR on (modified vector) **x**

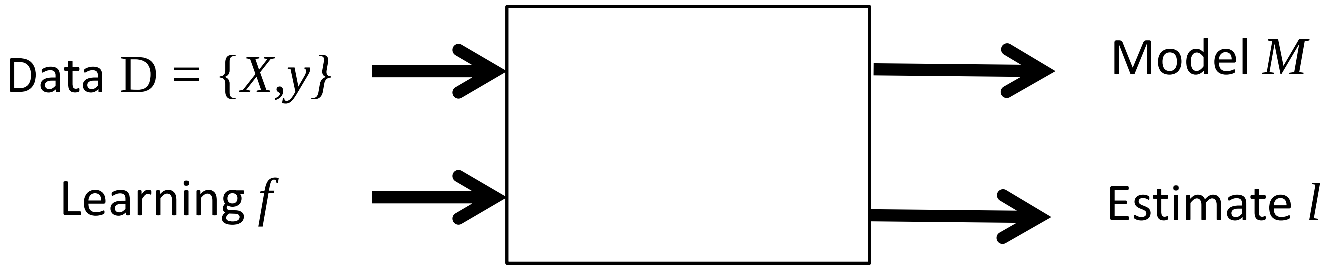
Learnt Model applying all steps

The learner *f* is creating a new function (model) with several steps. This is easier in languages where functions are first class objects, e.g., R, Matlab, python, but not C

# Learning Method $f$



# Estimation Protocol



# Example of Overfitting due to Bad CV

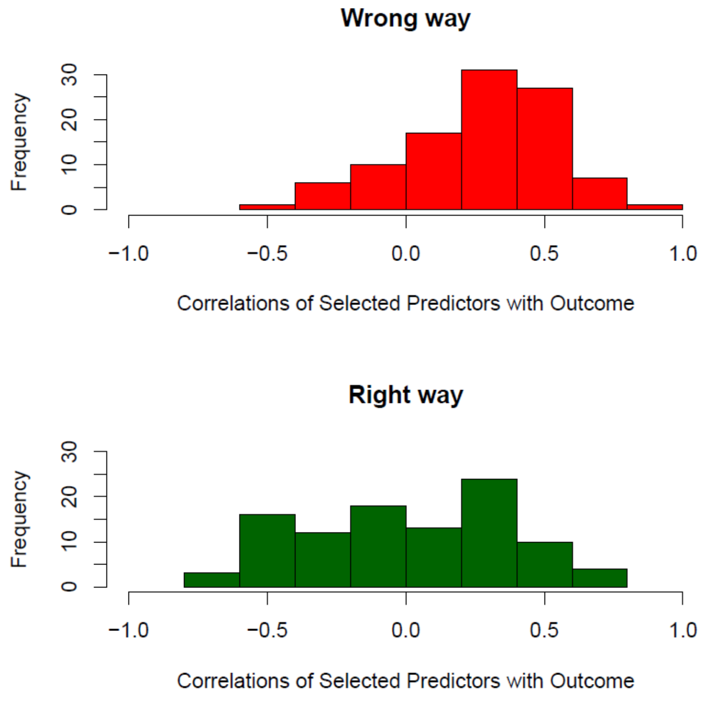
Consider a scenario with N = 50 samples in two equal-sized classes, and p = 5000 quantitative predictors (standard Gaussian) that are independent of the class labels. The true (test) error rate of any classifier is 50%.

**Wrong way**

1. Choose 100 predictors having highest correlation with the class labels
2. Use a 1-nearest neighbor classifier, based on just these 100 predictors
3. Average CV error of 1-KK rate on 50 simulations: 3%!!!

**Right way**

1. Divide the samples into K cross-validation folds (groups) at random.
2. For each fold k = 1, 2, . . . ,K
   a) Find a subset of "good" predictors that show fairly strong (univariate) correlation with the class labels, using all of the samples except those in fold k.
   b) Using just this subset of predictors, build a multivariate classifier, using all of the samples except those in fold k.
   c) Use the classifier to predict the class labels for the samples in fold k.



# Summary

- Always **follow the Golden Rule** in performance estimation.
- **All steps** of the analysis are **part of the learning method**, not just the classifier (regressor, etc.)
- The final model applies **all** what was **learnt** in all steps of the analysis **to new data**
- Perspective shift from estimating the performance of a model, to **estimating the performance of a learning method**
- Use **Stratified, Repeated K-fold Cross Validation**, K = #samples-of-rarest-class for small sample sizes and a single learning method

# Summary

Let's all **stop** overfitting (overestimating performance)

- Always **follow the Golden Rule** in performance estimation.
- **All steps** of the analysis are **part of the learning method**, not just the classifier (regressor, etc.)
- The final model applies **all** what was **learnt** in all steps of the analysis **to new data**
- Perspective shift from estimating the performance of a model, to **estimating the performance of a learning method**
- Use **Stratified, Repeated K-fold Cross Validation**, K = #samples-of-rarest-class for small sample sizes and a single learning method

# References


- (2018, September 4 ).Automated machine learning. Retrieved September 7, 2018, from https://en.wikipedia.org/wiki/Automated_machine_learning
- Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai*, *14*, 1137–1145.
- Hastie, Tibshirani, Friedman. Elements of Statistical Learning, p. 245, second edition.

# End of Part I