Evaluation of Classification and Regression Algorithms

João Gama
jgama@fep.up.pt

LIAAD-INESC TEC, University of Porto, Portugal

October 2019

Outline

Introduction

Evaluation is important

- to users
  - Model Selection:
    It enables to determine which ML system (classifier) should be used
  - Trustability:
    It enables to determine whether the outputs can be trusted,
- to the designers of ML algorithms
  - It enables to determine whether the outputs can be trusted,
  - It enables to determine which ML system (classifier) should be used, Importance to designers:
  - It enables them to design better systems (incorporate those subsystems that lead to better performance)

Evaluation Criteria

Different measures provide different perspective:

- Error rate (or success rate),
- Learning time (train time),
- Test time (time in using the model)
- Size of the model generated,
- Model interpretability.

The focus here is on **Error rate (or success rate)**

Loss Functions

- Assume we have an unknown function $f(.)$ that label the examples: $y = f(x)$.
  A labelled example is a pair of the form (x_i, y_i);
- Using a set of labelled examples, we learn an approximation function $\hat{f}(x)$.
- A loss function measures the goodness of that approximation.
  How to quantify the goodness of fit?

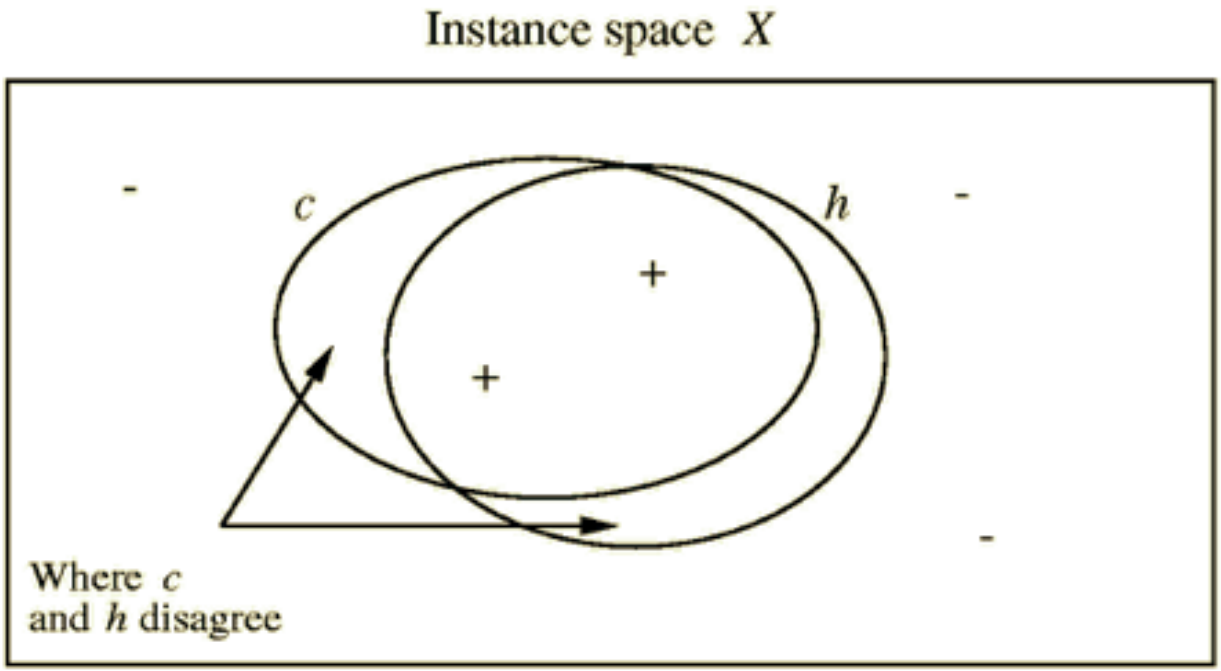
Loss Functions

- Assume an observation x.
  The true label (unknown) is $y = f(x)$.
- The learned function assigns the label $\hat{y} = \hat{f}(x)$.
- Dependent on the domain of y, we use different loss functions to quantify the matching between y and $\hat{y}$.
- For classification problems, $y \in \{y_1, \ldots, y_k\}$, for regression problems $y \in \Re$.
  - Classification: 0-1 loss function:
    $loss(y, \hat{y}) = 1$ iif $y \neq \hat{y}$; 0 iif $y = \hat{y}$
  - Regression:
    - Squared error: $loss(y, \hat{y}) = (y - \hat{y})^2$
    - Absolute error: $loss(y, \hat{y}) = |y - \hat{y}|$

Sample versus True Error

It is useful to distinguish between:

- True error
- Measured error on a data sample
  (provides basis for estimating the true error)



Sample Error

It is useful to distinguish between:

- Resubstitution error
  (error measured on the train data / biased estimator),
- Generalization error
  (error measured on the test data / unbiased estimator).

Golden Rule:

Error estimates should be measured on test data independent from the training data.

Score Functions for Predictive Models

Zero-One Loss (Misclassification error):

$$\frac{\sum_{i=1}^{N} I(y_i \neq \hat{y}_i)}{N}$$

where

- $I(exp)$ is the identity function which is 1, if exp is true and 0 otherwise
- y_i represents the observed value of the class variable of i-th example,
- $\hat{y}_i$ represents the predicted value of the class variable of i-th example,
- N represents the number of examples.

Different Objectives of Evaluation

- **We have 1 algorithm (classifier) and 1 dataset:**
  We want to know the estimate of the generalization error
- **We have 2 algorithms (classifiers) and 1 dataset:**
  We want to determine whether
  - one is better than another (in terms of the generalization error),
  - both have a comparable generalization error.
- **We have N algorithms (classifiers) and M datasets**:
  We want to order (rank) the algorithms taking into account the generalizations errors on all datasets

Methods for Estimating the Error

Methods oriented for 1 algorithm on 1 dataset:

- Train and test (holdout)
  (is useful if we have many cases)
- Cross-validation (CV)
  (is useful is we have fewer cases)
- Leave-one-out CV
- Bootstrap

In each case, we consider the estimation of:

- The most likely value (mean)
- Interval of confidence

Method Train and Test (Holdout)

Train and test method:

- Select a part of data for training (e.g. 70%), (stratified selection is better (see below))
- Select the remainder for testing (e.g. remaining 30%),
- Train the algorithm (classifier) on the train data,
- Use the algorithm (classifier) to classify cases using the test data,
- Calculate the error rate.

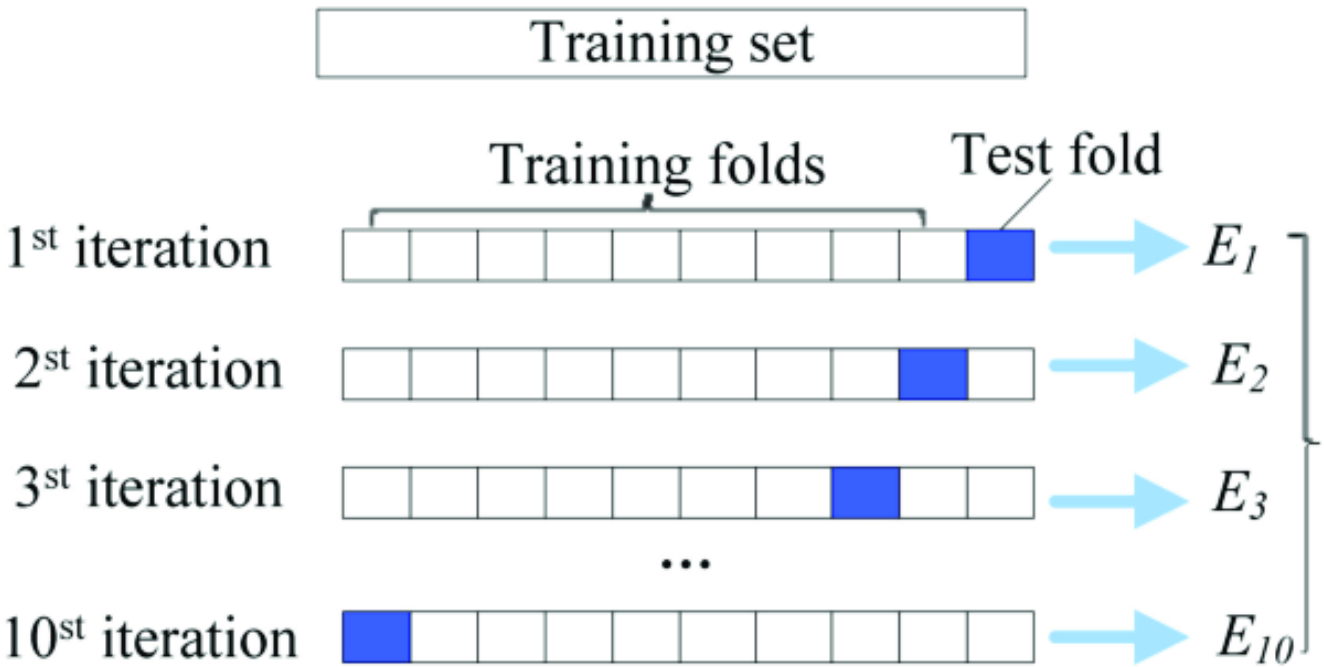
Stratified selection for samples: Maintain similar proportions of the cases of each class as in the full dataset

Cross Validation (CV)

Divide the data into N (e.g. $N = 10$) partitions (stratified selection is better)

- For $i = 1$ to N
  - Use all partitions except i for training,
  - Keep i_{th} partition for testing,
  - Train the algorithm to learn a classifier from the training data,
  - Use the classifier to classify cases using the test data,
  - Calculate the error rate.
- Repeat
- Stratified selection for samples:
  Maintain similar proportions of the cases of each class as in the full dataset
- An interesting property: each example will appear once in the test set

Cross Validation (10-CV)



$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

Leave-One-Out and Bootstrap

- **Leave-One-Out**
  - A special case of cross-validation.
  - Used when we have a small number N of cases (e.g. N=30),
  - the data is divided into N partitions,
    and so the test partition contains always just one case.
- **Bootstrap**
  - Useful if we have a small number of cases.
  - The train set is augmented by including some cases more than once (sampling with replacement).
  - The test set contains all cases not used for training. (Both sets should be disjoint, however)

Example: Evaluating Classifiers in R

Example using dataset `iris`:

```
> data(iris)
> str(iris)
'data.frame': 150 obs. of  5 variables:
$ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
$ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
$ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1
```

Evaluating Classifiers in R

Generate train and test set without permutation:

```
> last <-0.7*nrow(iris)
> train <-iris[1:last,]
> test <-iris[-(1:last),]
```

Generate train and test set, while permuting data:

```
> permute.index <-sample(1:nrow(iris), 0.7*nrow(iris))

> train <-iris[ permute.index, ]
> test <-iris[ -permute.index, ]
```

Generating predictions

Train the algorithm (classifier) on the train data,

```
> library(rpart)
> arv <-rpart(Species ~ . , train)
> arv
n= 105

node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 105 69 setosa (0.3428571 0.3142857 0.3428571)
2) Petal.Length< 2.45 36  0 setosa (1.0000000 0.0000000 0.0000000) *
3) Petal.Length>=2.45 69 33 virginica (0.0000000 0.4782609 0.5217391)
6) Petal.Length< 4.75 29  0 versicolor (0.0000000 1.0000000 0.0000000)
7) Petal.Length>=4.75 40  4 virginica (0.0000000 0.1000000 0.9000000) *
```

Ploting the Decision Tree

```
> plot(árvore)
> text(árvore)
```

Improved tree plot:

```
> show.tree <-function(arvore) {
+ plot(arvore,uniform=T,branch=0)
+ text(arvore,digits=3,cex=0.65,
+ font=10, pretty=0,fancy=T,fwidth=0,
+ fheight=0)
+}
> show.tree(arvore)
```

Other improvements:

```
> help(plot.rpart)
```

Generating predictions

Predict the classifier on the test data,

```
> preds <- predict(arv, test, type="class")
> table(preds)
preds
setosa versicolor  virginica
14          16          15
```

Generate confusion matrix:

```
> mc <- table(preds,test[,5])
> mc
preds         setosa versicolor virginica
setosa          14          0         0
versicolor       0         15         1
virginica        0          2        13
```

Calculating Error

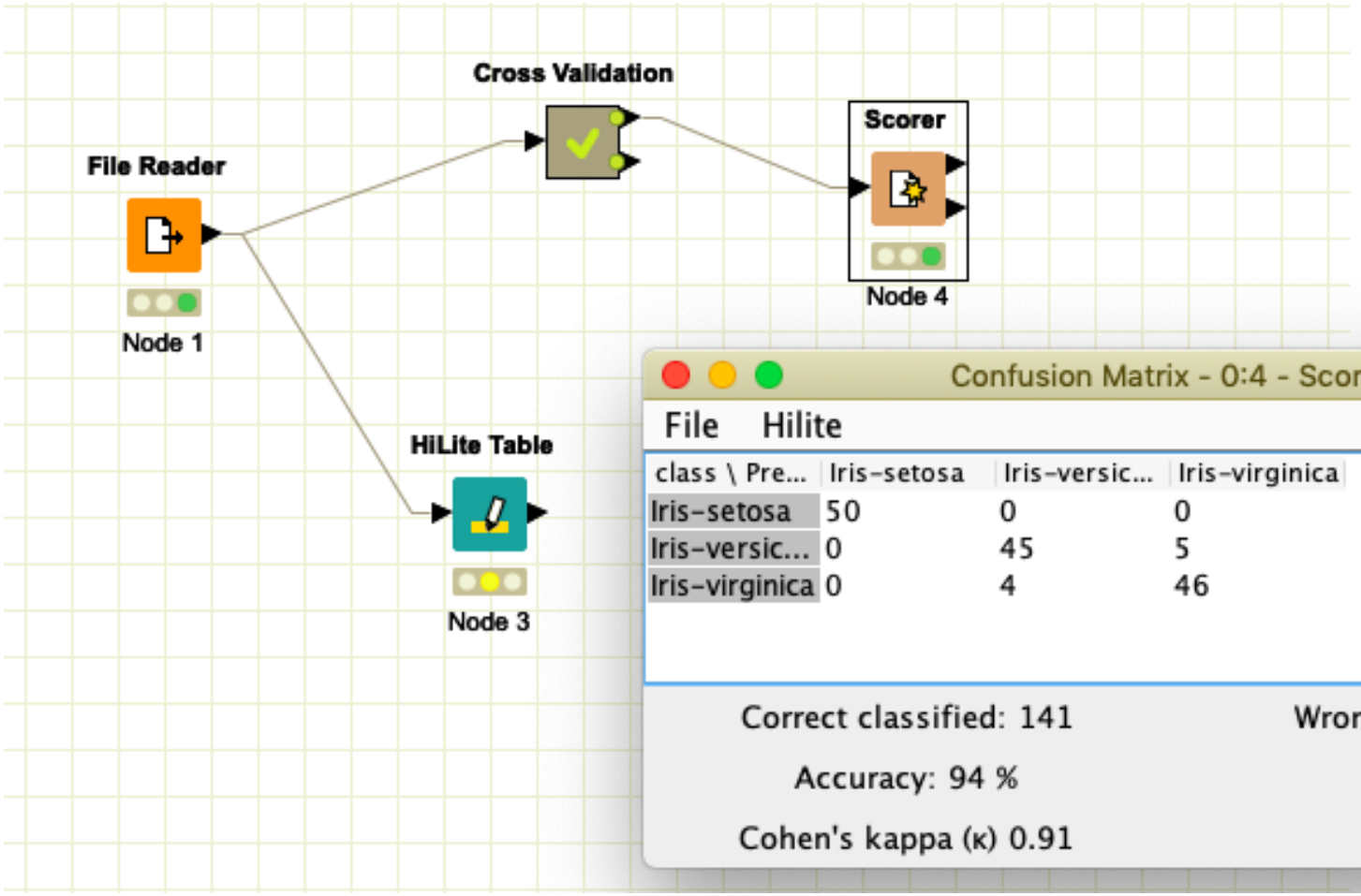
```
> mc
preds          setosa versicolor virginica
setosa             14          0         0
versicolor          0         15         1
virginica           0          2        13
```

- Number of examples classified
  $> sum(mc)$
- Calculate the total of correct classifications on the diagonal of the confusion matrix:
  $> diag(mc)$
  $> sum(diag(mc))$
- Calculate the accuracy rate:
  $> sum(diag(mc))/sum(mc)$
- Calculate the error rate:
  $> 1 - sum(diag(mc))/sum(mc)$

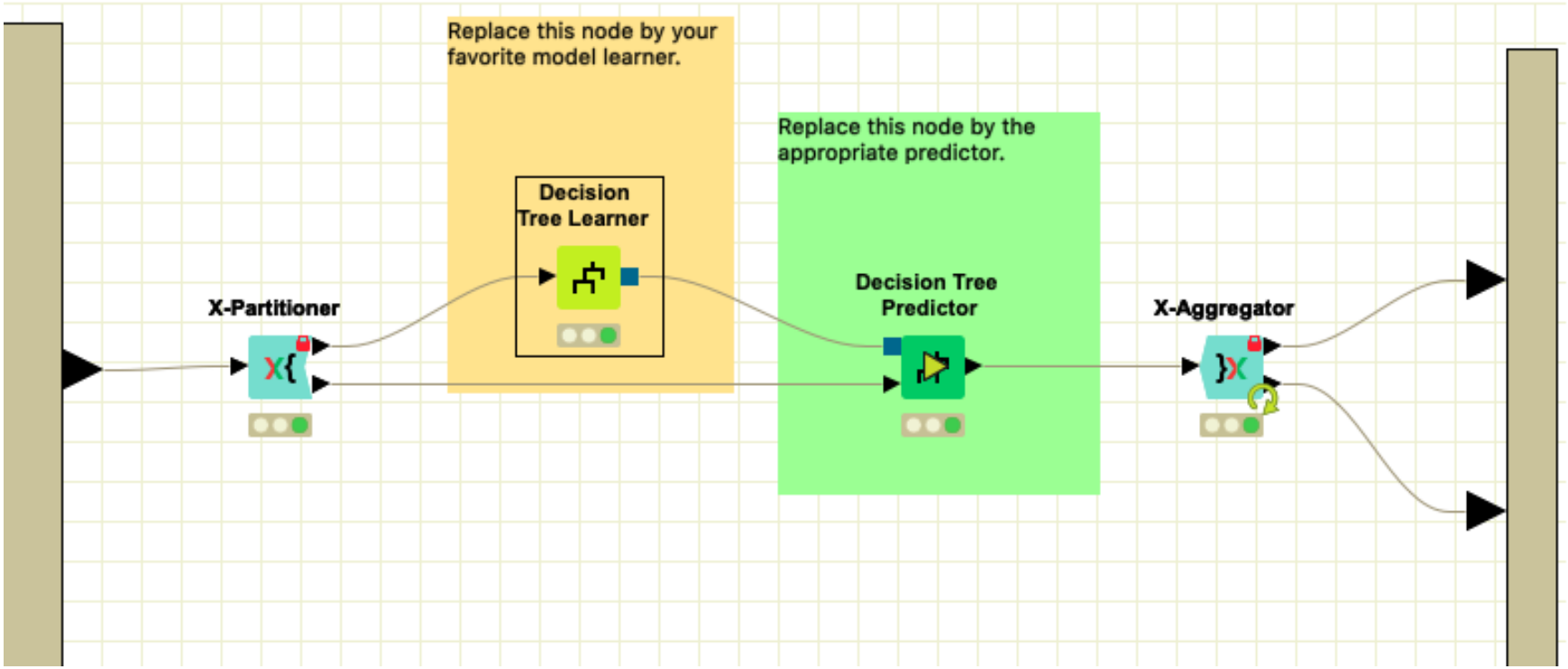
Implementing 10-fold CV

```
#Randomly shuffle the data
data<-iris[sample(nrow(iris)),]

#Create 10 equally size folds
folds <- cut(1:nrow(data),breaks=10,labels=FALSE)

#Perform 10 fold cross validation
for(i in 1:10){
testData <- data[folds == i, ]
trainData <- data[folds != i, ]

% #Use the test and train data partitions however you desi
. . .
}
```

Cross-validation in KNIME



Cross-validation in KNIME



Estimating Intervals of Confidence

- Consider some hypothesis h(x).
- Given a sample S of size n, we can calculate the $e = error(h|S)$.
- What can be infered about the error in the population $p = error(h|P)$?
- We cannot compute p, but can deduce an interval that contains p for a given confidence level.

Given that error rate follows a binomial distribution:

Confidence Interval

$$CI = e \pm z \times \sqrt{\frac{e \times (1-e)}{n}}$$

where z can be looked up in a table of the binomial distribution and depends on the confidence level.

Intervals of Confidence for Train / Test

Computing the sample error, $error(h|S)$, using Train and Test.
$error(h|P) \approx error(h|S)$, and
$var(error(h|P)) = \frac{error(h|S) \times (1 - error(h|S))}{n}$
The interval of confidence can be derived analytically:

Interval of Confidence

Given a confidence level α, $error(h|P)$ is contained in the interval

$$error(h|P) = error(h|S) \pm z_\alpha \sqrt{\frac{error(h|S) \times (1 - error(h|S))}{n}}$$

If the confidence level is 95%, then $z_\alpha = 1.26$

Example

- e=10/100 (10 errors in 100 (n=100)
- The given confidence level is 95%
  This determines $z = 1.96$
- z value that can be looked up in a table for binomial distribution and given confidence level (e.g. 95%)
- The confidence interval is: $CI = e \pm z \times \sqrt{\frac{e \times (1-e)}{n}}$
- $CI = 0.1 \pm 1.96 \times \sqrt{\frac{0.1 \times (1-0.1)}{100}}$
- 0.1 ± 0.058

Outline

Comparing the Performance of 2 Classifiers

One possible objective mentioned earlier: We have 2 algorithms (classifiers) and 1 dataset:

We want to determine whether

- one algorithm is better than another (in terms of the generalization error),
- both have a comparable generalization error.

We will orient our analysis to CV.

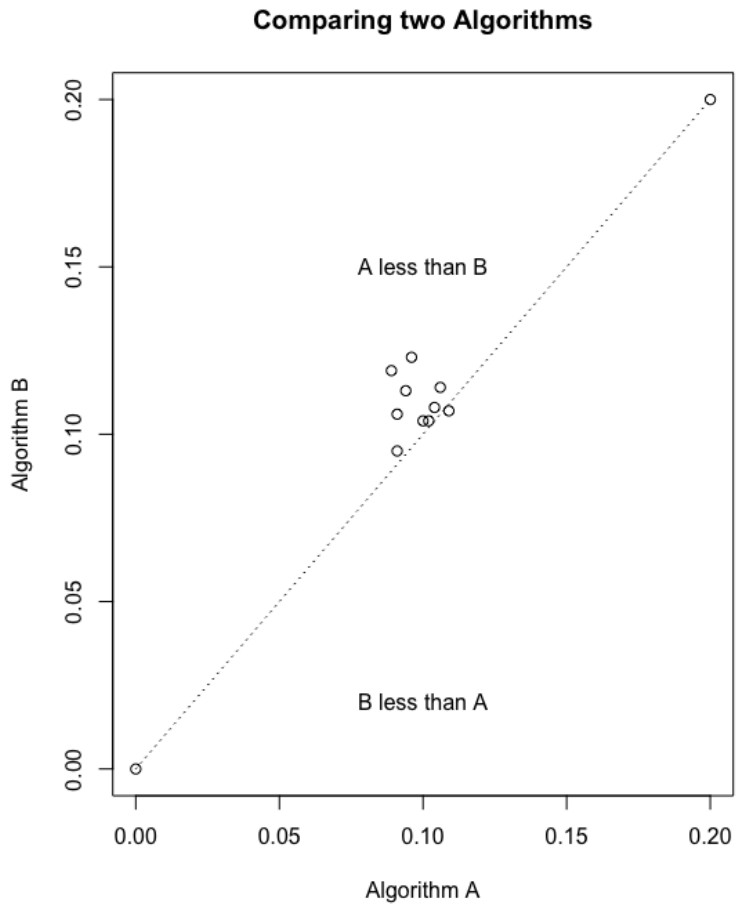
Example

Consider the following example:

- Errors of classifier A:
  0.100 0.094 0.109 0.091 0.096 0.104 0.102 0.089 0.091 0.106
- Errors of classifier B:
  0.104 0.113 0.107 0.106 0.123 0.108 0.104 0.119 0.095 0.114

Is A better than B? Or vice versa?
Or are they comparable?

Example



Dotted line represents equal performance of both algorithms.

Hypothesis Test

- We cannot simply compare the means, as the difference might be only by chance and are not statistically significant.
- The proper way is to conduct a hypothesis test that should answer the question whether the two samples belong to two identical populations (null hypothesis) or to two different populations (alternative hypothesis).
- Let μ_A and μ_B represent the mean errors of two models in CV evaluation. We can formulate:
  - Null hypothesis: $H0 = \mu_A - \mu_B = 0$
  - Alternative hypothesis: $H1 = \mu_A - \mu_B \neq 0$
- The objective of a statistical test is to accept / reject the null hypothesis (and hence accept the alternative hypothesis) Different statistical tests that can be used.

Statistical Tests

- **Parametric tests** make assumptions about the underlying distribution (e.g. that it is normal)
  Example: (`t-test`)
- **Non-Parametric tests** `do not make any assumptions about the underlying distribution.`
  `Example: Wilcoxon signed-rank test, McNemar test`

With CV, always use the variant of **matched pairs**:

- Match errors on corresponding folds of CV.
- Both algorithms are trained and evaluated in the same conditions (same train and test set)
- This test has greater statistical power.

Use **two-sided** tests.

Tests based on Student Distribution

To determine whether two means are statistically different, calculate:

- the differences of errors $d_i = e_{a_i} - e_{b_i}$
- the mean of all differences d
- Student distribution $t = d/(\sqrt{\sigma^2/k})$
- Use table for t-distribution with $k-1$ degrees of freedom (nº of observations) to establish the limit z for a given confidence level
  If confidence level is 95%, z is 1.83.
- Determine whether t exceeds the limit z
  (either $t > z$ or $t < -z$)
  If it does, the means are significantly different.

Conducting t-test in R: Example

```
> a
[1] 0.100 0.094 0.109 0.091 0.096 0.104 0.102 0.089  0.091 0.106
> b
[1] 0.104 0.113 0.107 0.106 0.123 0.108 0.104 0.119 0.095 0.114
> t.test(a,b,alternative="two.sided", conf.level = 0.95)
Welch Two Sample t-test
data:  a and b
t = -3.2721, df = 17.598, p-value = 0.004333
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-0.018238664 -0.003961336
sample estimates:
mean of x mean of y
0.0982    0.1093
```

Wilcoxon signed-rank tests (non-param.)

Wilcoxon signed rank test:

- Calculate differences of performance measure of models A,B
- Calculate the absolute value of the differences
- Rank the absolute values values
- Can we distinguish the values of A from those of B?

Data	A	B	Diff	Abs. dif	Rank
Pulmão	0.583	0.583	0	0	1.5
Fungo	0.583	0.583	0	0	1.5
Atmosfera	0.882	0.888	+0.006	0.006	3.0
Mama	0.599	0.591	-0.008	0.008	4.0

Wilcoxon signed-rank tests

- Calculate the mean value $R+$ of ranks in which B is better than A
  (average ranks of ties)
- Repeat for $R-$ where B is worse than A.
  (average ranks of ties)
- Determine S, the minimum of $R+$ and $R-$.
- Look up a critical value z in a table for given S, N. or alternatively if $N > 25$, use a formula:
  $z = \frac{S - 1/4N(N-1)}{\sqrt{1/24N(N+1)(2N+1)}}$
- Assuming confidence level of 95% (a=0.05)
  reject null hypothesis, if $z < -1.96$.

(see Gama et al.: E.C.D., p.204 for more details)

Wilcoxon signed-rank tests in R

```
> a <- c(0.100,0.094,0.109,0.091,0.096,0.104,0.102,0.089,0.091,0.106)
> b <- c(0.104,0.113,0.107,0.106,0.123,0.108,0.104,0.119,0.095,0.114)
> ?wilcox.test
> wilcox.test(a,b)

        Wilcoxon rank sum test with continuity correction

data:  a and b
W = 14.5, p-value = 0.008008
alternative hypothesis: true location shift is not equal to 0
```

Problem with repeated tests

- If we repeat tests, there is some chance that the test will return a wrong result.
- There are two possibilities (Type I and Type II errors):
  - The method should reject a null hypothesis, but it did not.
  - The method should not reject a null hypothesis, but it did.
- These errors arise due to the statistical nature of the test.
- If the given confidence is, say, 95%, we can expect that in approximately 5% of cases the test will go wrong.
- So, if we repeat test, we need to carry out a Bonferroni adjustment.

Adjustment for Multiple Tests

Bonferroni adjustment is used to adjust the confidence level:

$$\alpha_n = 1 - (1 - a)^n$$

where n is the number of repetitions.
Ex. If the test is repeated twice, we need to adjust 95% to:
$\alpha = 1 - (1 - 0.95)^2 = 0.9975$

Problems with t-tests

Although t-tests are commonly used, the test is being criticized as not somewhat problematic:

- The training data used in different folds of CV is not independent,
- The test assumes normal distribution

Some authors suggest

- Using 10*10-fold CV, with permutation of the data in each run,
- Wilcoxon matched-pairs signed-rank test.

Outline

Comparing N Classifiers

Suppose our task it use N classifiers on M different datasets and determine which one(s) provide the best performance.
There are different types of answers we may seek to respond:

- Identify the best classifier,
- Identify the best classifier and all equivalent ones within critical distance,
- Provide a ranking of classifiers,
- Provide a ranking of groups of classifiers.

How should we proceed?

Approach based on Ranks of Performance

The approach currently used is based on ranks of performance (following Demsar, 2006)

- Construct a table where :
  - columns represent classifiers,
  - rows represent datasets,
  - value $< i, j >$ represents a rank of performance of classifier$j$ on dataset i
- obtained as a result of evaluation (e.g. running CV)
- Elaborate the global measure for each algorithm (column): **mean rank**
- Elaborate a ranking of the mean ranks.
- This permits to decide which classifier is best overall.

Approach based on Ranks

- The first question that arises is: Are the results of the classification algorithms significantly different?
- This can be determined by Friedman test (non-parametric).
  The null hypothesis is that there is no difference among the classifiers. If the null hypothesis is rejected, we can proceed with a post-hoc tests.
- The process involves calculation a Friedman statistic F_F, which is a function of :
  N the number of datasets,
  A is the number of algorithms
  R_J mean ranks of algorithms.

$$F_F = \frac{(N-1)\chi_F^2}{N(A-1)-\chi_F^2} \qquad \chi_F^2 = \frac{12N}{A(A+1)}[\sum_j R_j^2 - \frac{A(A+1)^2}{4}]$$

Approach based on Ranks

- The null hypothesis is rejected
  if the statistic F_F is greater than $F_{A-1,(A-1)(N-1)}$
  where $A-1$ and $(A-1)(N-1)$ represent the degrees of freedom.
- The value of $F_{A-1,(A-1)(N-1)}$ can be retrieved from books on statistics.
- If the null hypothesis is rejected, we can proceed with post-hoc test.

Approach based on Ranks

- Post-hoc tests can determine whether the performance of two classifiers is significantly different.

  In post-hoc tests we can use:

  - Nemenyi test for all pairs of classifiers,
  - Bonferoni-Dunn test, where all classifiers are compared to a control classifier.

- We can calculate critical distance CD (function of A, N and q_a) which can be used to determine whether two algorithms are significantly different. This happens if the differences of mean ranks exceed CD:

$$CD = q_\alpha \sqrt{A(A+1)/(6N)}$$

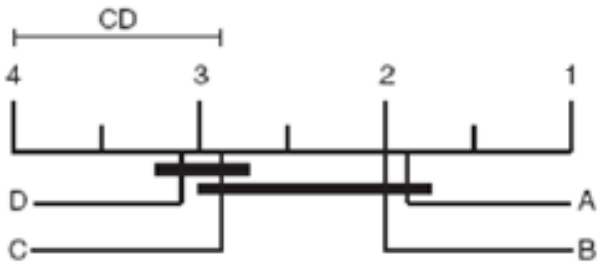
Values of $q_0.05$ for different post-tests and different values of A:

A	2	3	4	5	6	7	8	9	10
Nemenyi	1,960	2,343	2,569	2,728	2,850	2,949	3,031	3,102	3,164
Bonferroni-Dunn	1,960	2,241	2,394	2,498	2,576	2,648	2,690	2,724	2,773

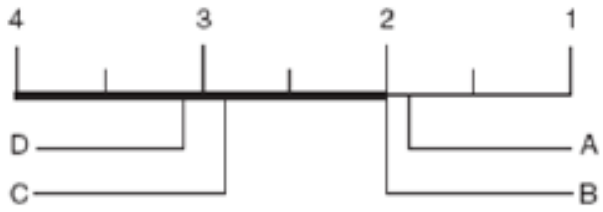
Approach based on Ranks

	C4.5	C4.5+m	C4.5+cf	C4.5+m+cf
Classifier	D	B	C	A
average rank	3.14	2.00	2.89	1.96

- Comparisons using a Nemenyi test is shown in fig. (a). The best ranked algorithm (A) is shown. Groups of classifiers that are not significantly different are connected.
- Comparisons using a Bonferoni-Dunn test is shown in (b). It assumes that classifiers A,B,C are compared to D.



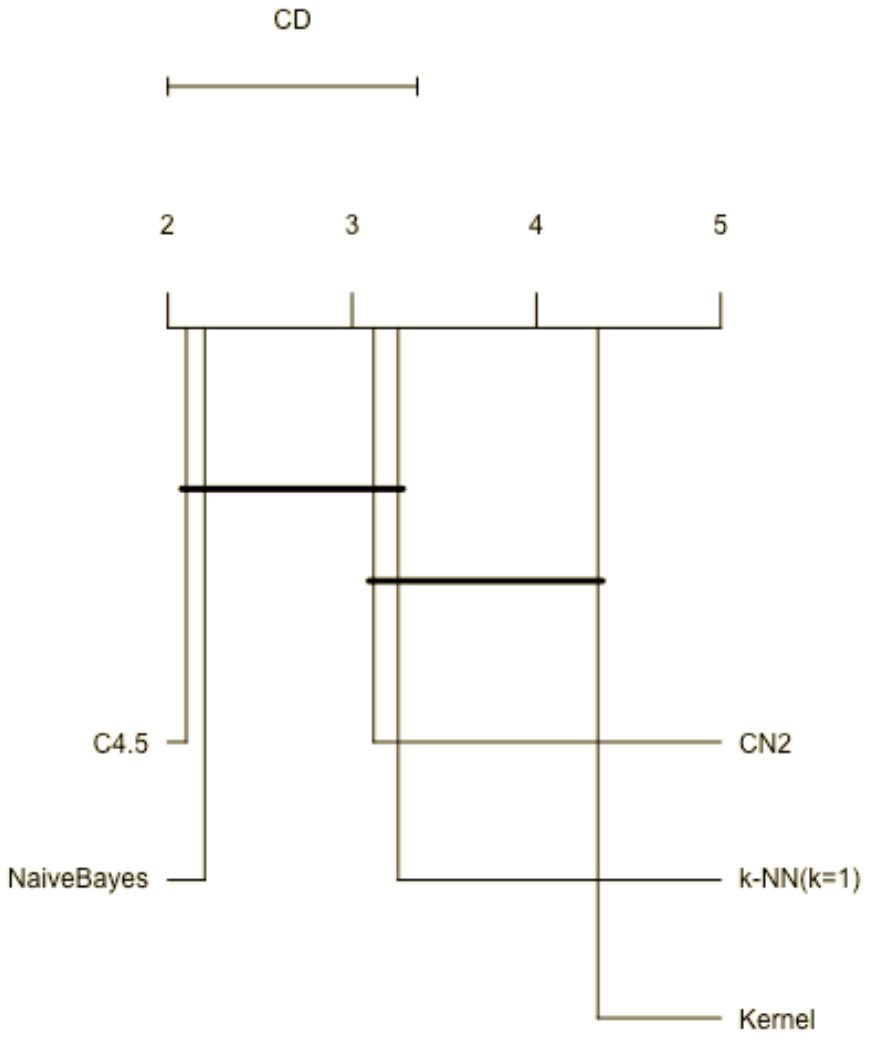
(a) Nemenyi



(b) Bonferroni-Dunn

Example in R

$> library('scmamp')$
$> data(data\_gh\_2008)$
$> plotCD(data.gh.2008, alpha = 0.01)$



Outline

"No Free Lunch" Theorems

$Acc_G(L)$ = Generalization accuracy of learner L
= Accuracy of L on non-training examples
$\mathcal{F}$ = Set of all possible concepts, $y = f(\mathbf{x})$

Theorem: For any learner L, $\frac{1}{|\mathcal{F}|}\sum_{\mathcal{F}} Acc_G(L) = \frac{1}{2}$
(given any distribution $\mathcal{D}$ over $\mathbf{x}$ and training set size n)

Proof sketch: Given any training set S:
For every concept f where $Acc_G(L) = \frac{1}{2} + \delta$,
there is a concept f' where $Acc_G(L) = \frac{1}{2} - \delta$.
$\forall \mathbf{x} \in S, f'(\mathbf{x}) = f(\mathbf{x}) = y. \quad \forall \mathbf{x} \notin S, f'(\mathbf{x}) = \neg f(\mathbf{x}).$

Corollary: For any two learners L_1, L_2:
If $\exists$ learning problem s.t. $Acc_G(L_1) > Acc_G(L_2)$
Then $\exists$ learning problem s.t. $Acc_G(L_2) > Acc_G(L_1)$

What Does This Mean in Practice?

- Don't expect your favorite learner to always be best
- Try different approaches and compare
- But how could (say) a multilayer perceptron be less accurate than a single-layer one?

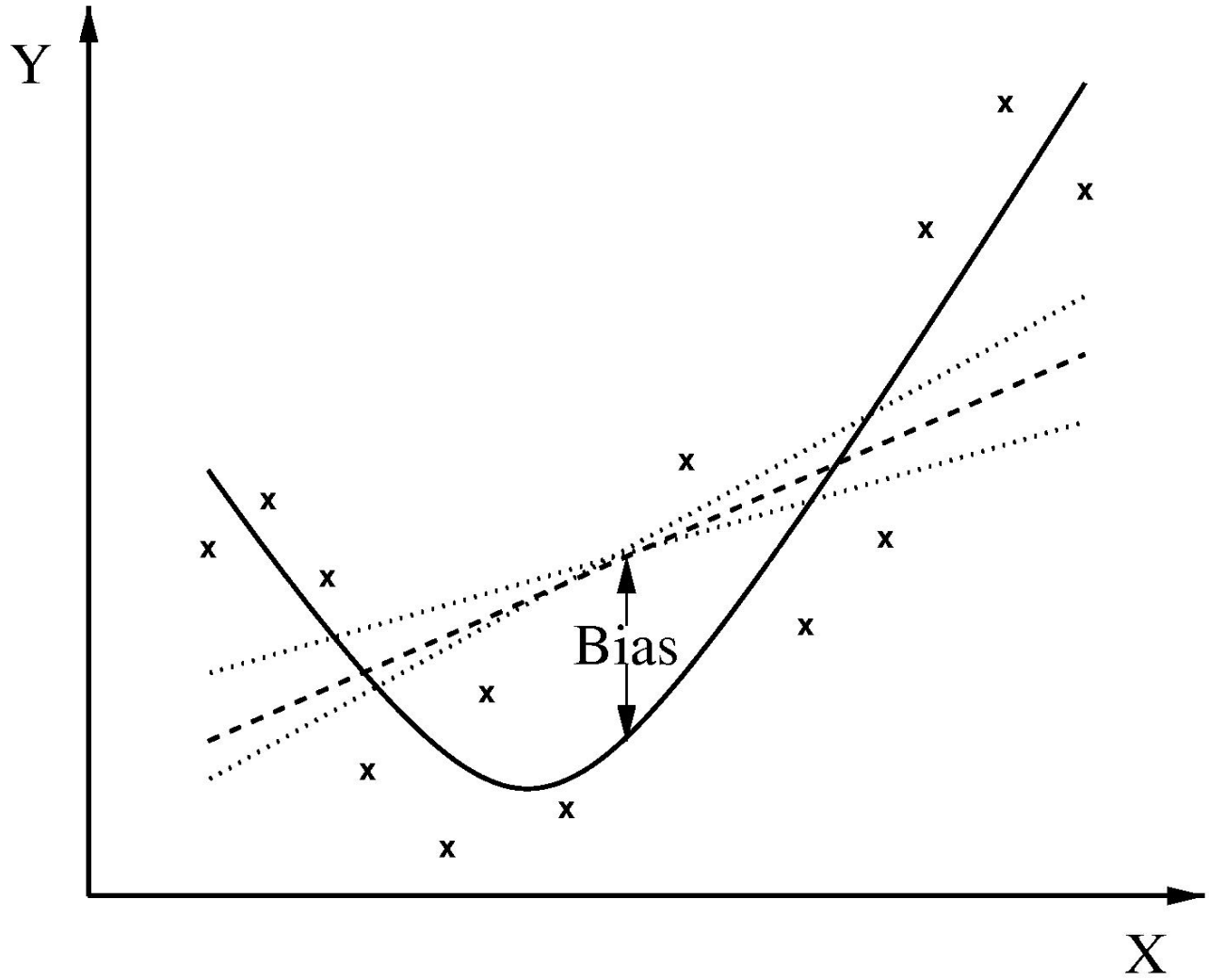
Bias and Variance

- Bias-variance decomposition is key tool for understanding learning algorithms
- Helps explain why simple learners can outperform powerful ones
- Helps explain why model ensembles outperform single models
- Helps understand & avoid overfitting
- Standard decomposition for squared loss
- Can be generalized to zero-one loss

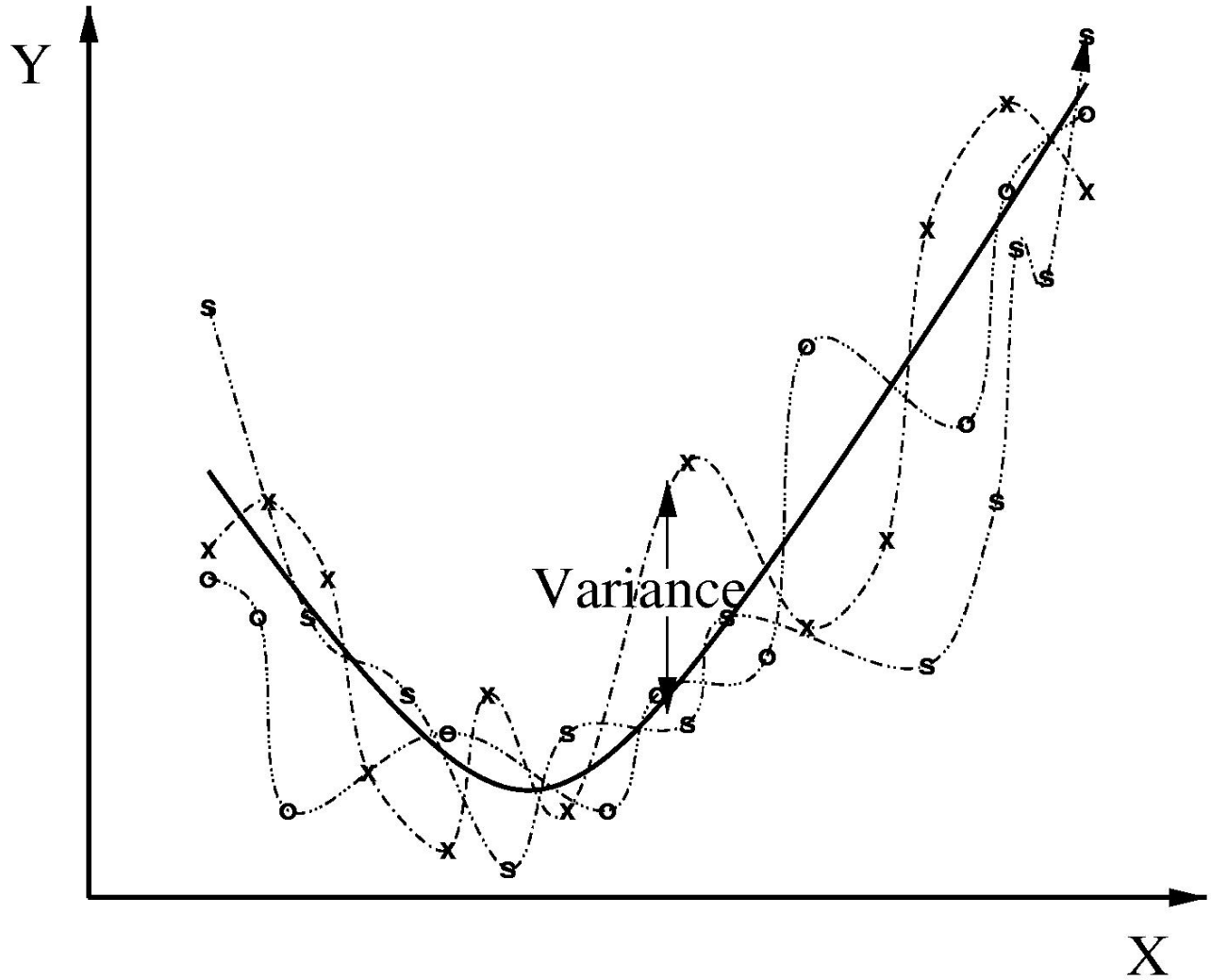
Definitions

- Given training set: $\{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_n, t_n)\}$
- Learner induces model: $y = f(\mathbf{x})$
- Loss measures quality of learner's predictions
  - Squared loss: $L(t, y) = (t - y)^2$
  - Absolute loss: $L(t, y) = |t - y|$
  - Zero-one loss: $L(t, y) = 0$ if $y = t$, 1 otherwise
  - Etc.
- Loss = Bias + Variance + Noise
  (This lecture: ignore noise; see paper)

Bias



Variance



Decomposition for squared loss

$$
\begin{aligned}
(t-y)^2 &= (t-\overline{y}+\overline{y}-y)^2 \\
&= (t-\overline{y})^2 + (\overline{y}-y)^2 + 2(t-\overline{y})(\overline{y}-y)
\end{aligned}
$$

$$
\begin{aligned}
E[(t-y)^2] &= (t-\overline{y})^2 + E[(\overline{y}-y)^2] \\
\text{Exp. loss} &= \text{Bias} \quad + \text{Variance}
\end{aligned}
$$

(Expectations are over training sets)

How to generalize this to other loss funcs?

$$E[(t-y)^2] = (t-\overline{y})^2 + E[(\overline{y}-y)^2]$$

$$\begin{array}{rcll}
(a-b)^2 & \rightarrow & L(a,b) & \\
E[(t-y)^2] & \rightarrow & E[L(t,y)] & \text{(Exp. loss)} \\
(t-\overline{y})^2 & \rightarrow & L(t,\overline{y}) & \text{(Bias)} \\
E[(\overline{y}-y)^2] & \rightarrow & E[L(\overline{y},y)] & \text{(Variance)}
\end{array}$$

But what should $\overline{y}$ be?

Define **Main Prediction**:
Prediction with min average loss relative to all predictions

$$\overline{y}_L = \underset{y'}{\operatorname{argmin}}\, E[L(y, y')]$$

- Squared loss: $\overline{y}$ = Mean
- Absolute loss: $\overline{y}$ = Median
- Zero-one loss: $\overline{y}$ = Mode

Generalized definitions

Bias = Loss incurred by main prediction = $L(t, \overline{y})$

Variance = Average loss incurred by prediction relative to main prediction = $E[L(\overline{y}, y)]$

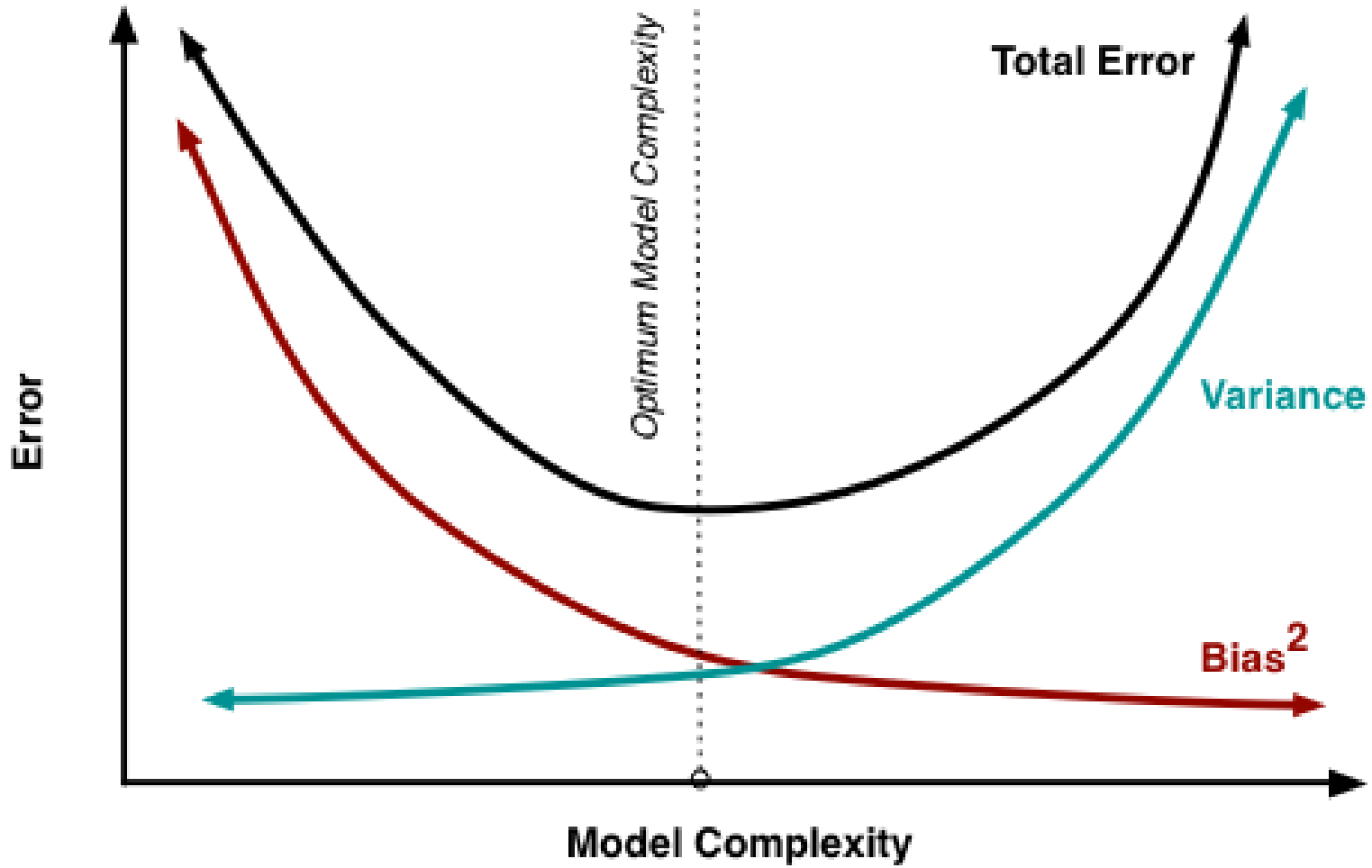
These definitions have all the required properties.

For zero-one loss:

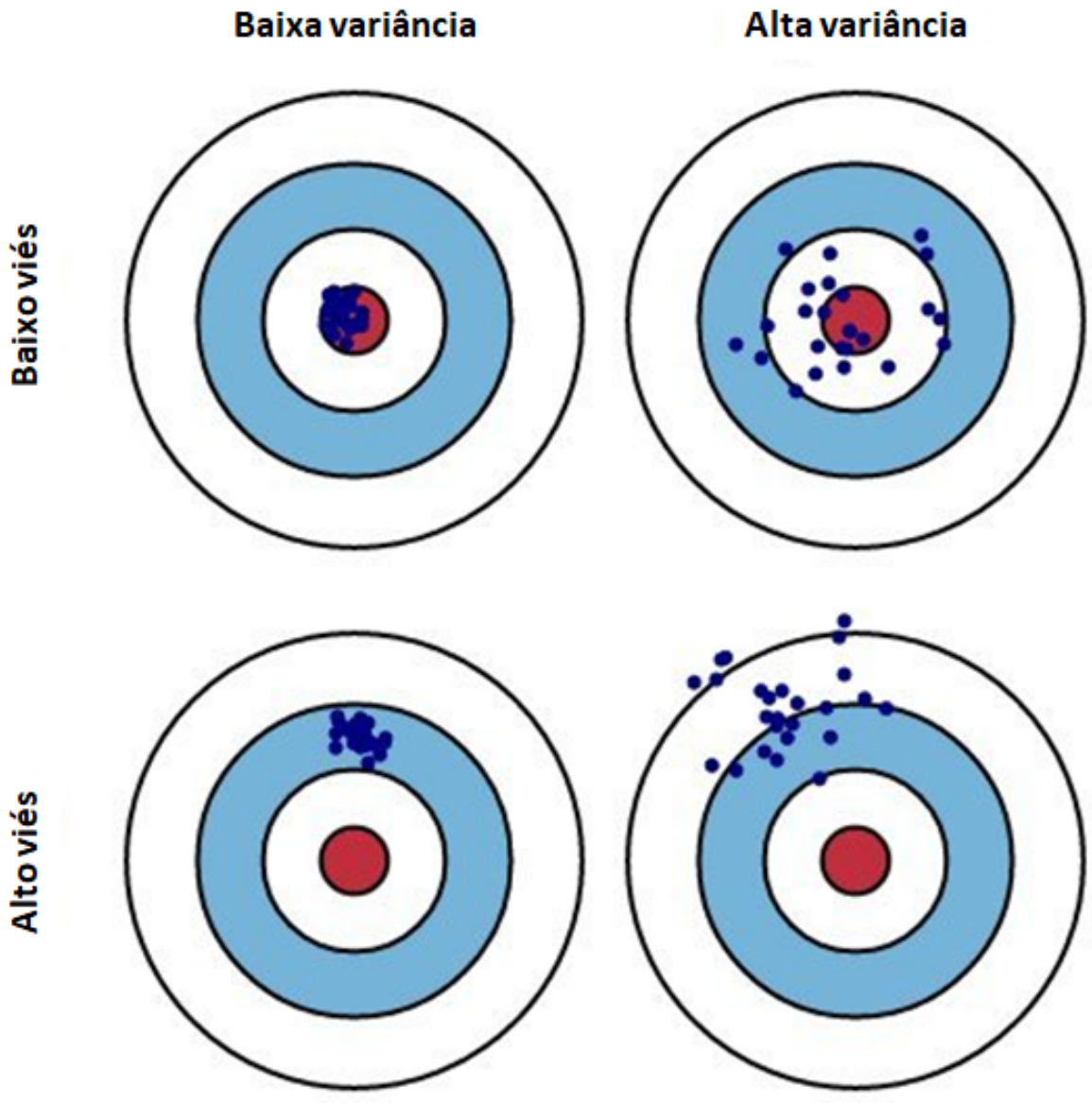
$$\textbf{Bias} = \begin{cases} 0 \text{ if main prediction is correct} \\ 1 \text{ otherwise} \end{cases}$$

Variance = Prob(Prediction $\neq$ Main pred) = $P(y \neq \overline{y})$

Bias-Variance Tradeoff



Bias-Variance Intuition



Bias-Variance Tradeoff

Typical behaviour:

- High bias, Low variance
  Linear Discriminants, Naive Bayes
- Low bias, High variance
  Decision Trees, Neural Networks

If we increase the number of degrees of freedom of the model:

- Bias will diminish
- Variance will increase
- To minimize the expected error, we need establish a compromise between the two components.

Outline

Further Reading

- J.Gama, A.de Carvalho, K.Facelli, A. C. Lorena, M.Oliveira: Extração de Conhecimento de Dados, Cap. 9, Ed. Sílabo, 2017.
- T. Mitchell: Machine Learning, McGrawHill, 1997 Chapter 5 Evaluationg Hypotheses
- D.Hand, H. Manilla, P. Smyth: Principles of Data Mining, MIT Press, 2001 Section 7.3 Predictive versus Descriptive Score Functions; Section 10.2 Evaluating and Comparing Classifiers
- M.Berthold,D.Hand: Intelligent Data Analysis, Section 2.5 Prediction and Prediction Error; Section 2.6 Resampling
- J Demsar: Statistical Comparisons of Classifiers over Multiple Data Sets, The Journal of Machine Learning Research, Vol. 7 , 2006, Pages: 1–30.