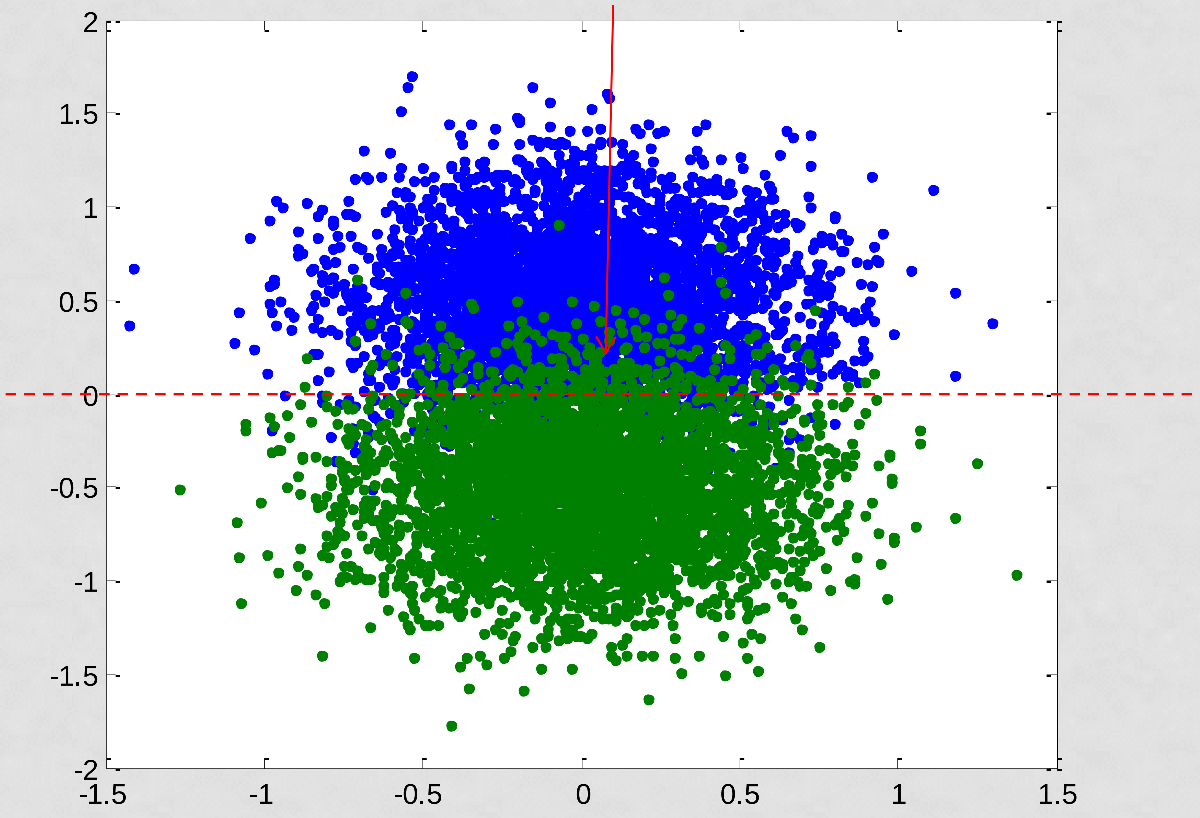
MODEL EVALUATION

EVALUATION

- Once a model has been trained, it is important to estimate its future performance (i.e. with data not used for training)
- A model performing well on the training data does not imply that the model will perform well on new data (due to overfitting to the training data)
- Intuitiion: if a student is evaluated (exam) with the same exercises s/he used for learning, probably s/he will get high grades
- But s/he is not showing that s/he has generalized beyond the training exercises. S/he might have just memorized them

EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

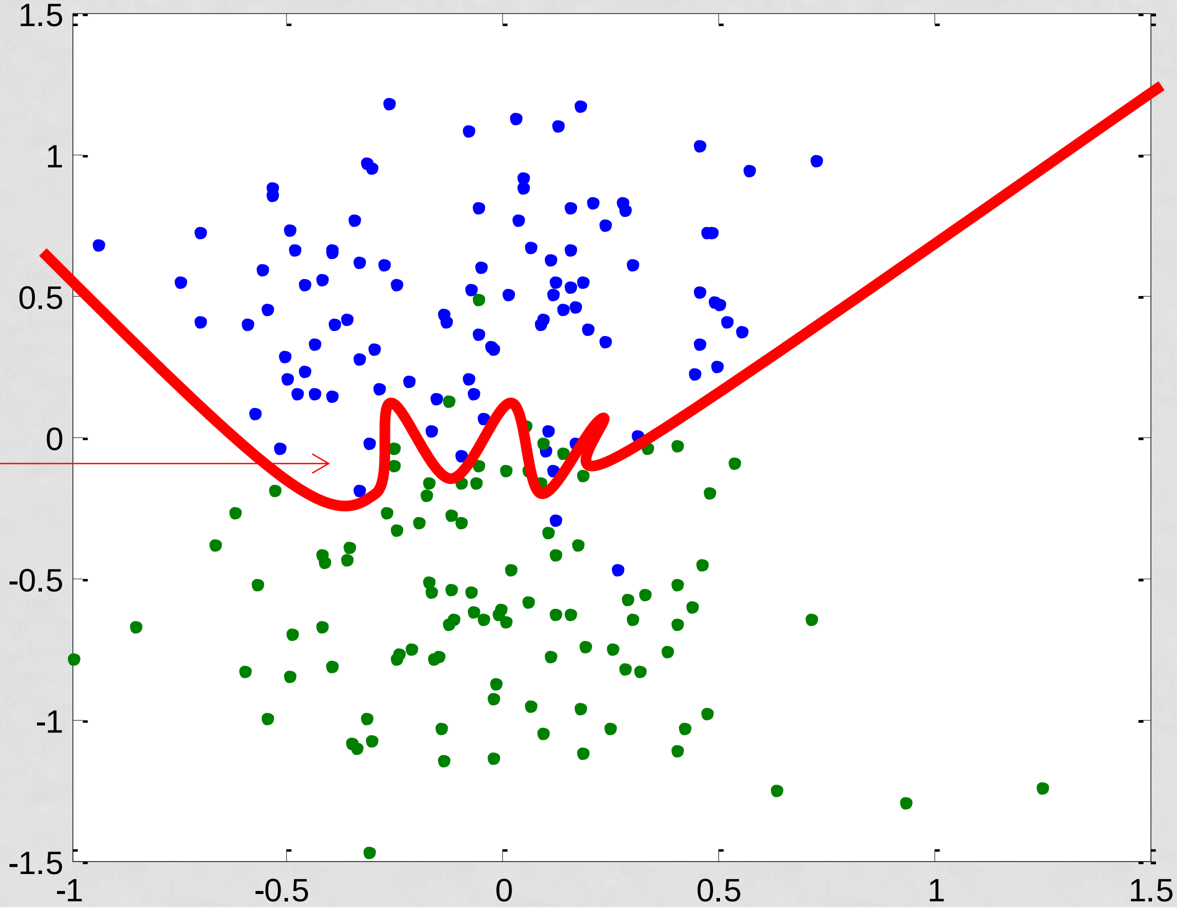
Let's suppose we want to learn a model that is able to separate class "blue" from class "green". Below we can see instance space with thousands of instances. Notice that both classes overlap in the middle



EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

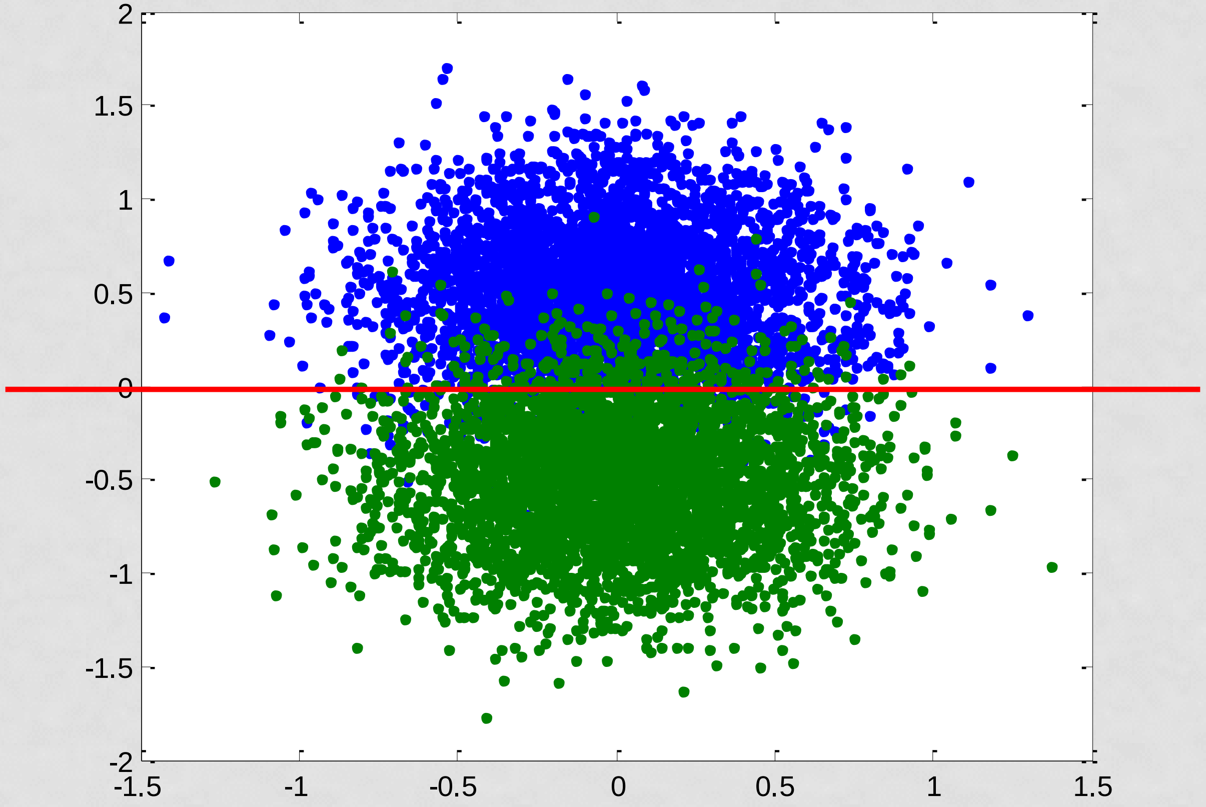
- But if we have few data for training, the following model might be learned
- The model is obviously not generalizing well. It is memorizing the data, or **overfitting** the data

This curve has been learned because there are no green instances here.
But this happened by chance. If we had more instances, probably there would be green instances in that region



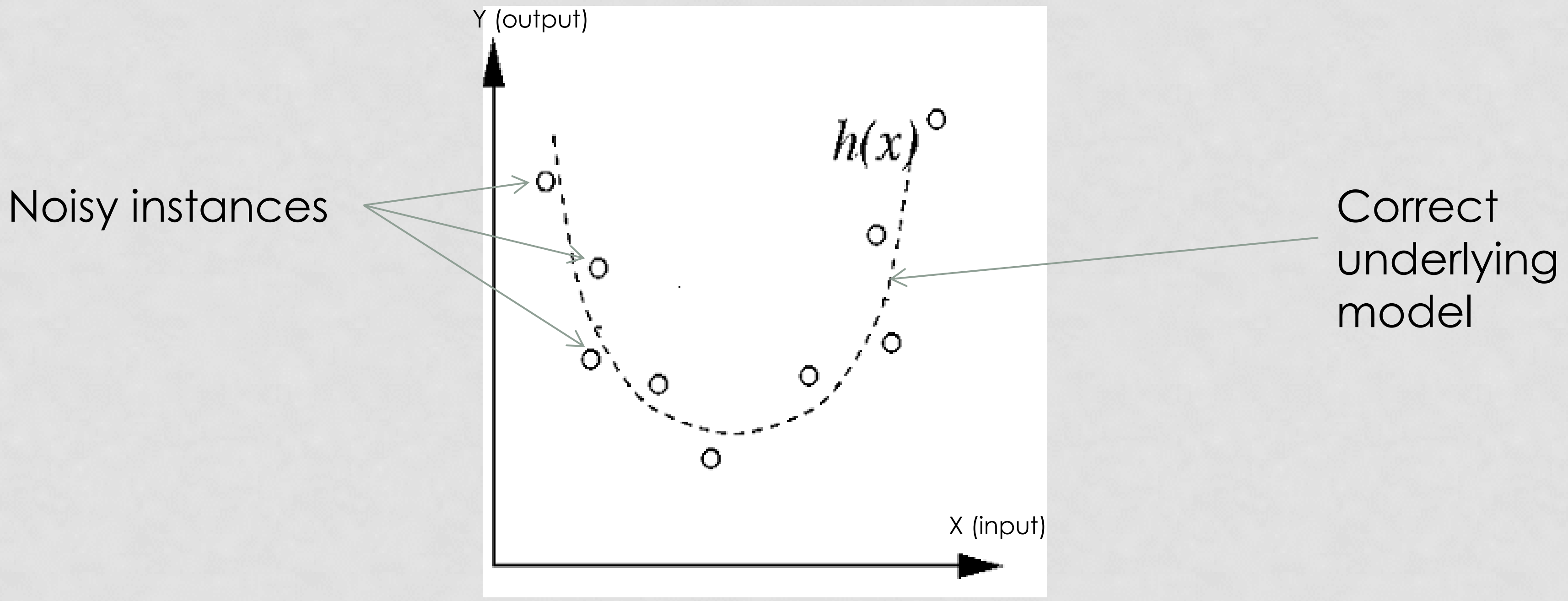
EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

If we had lots of data, the following (correct) model would have been learned



EXAMPLE OF A REGRESSION MODEL NOT GENERALIZING WELL

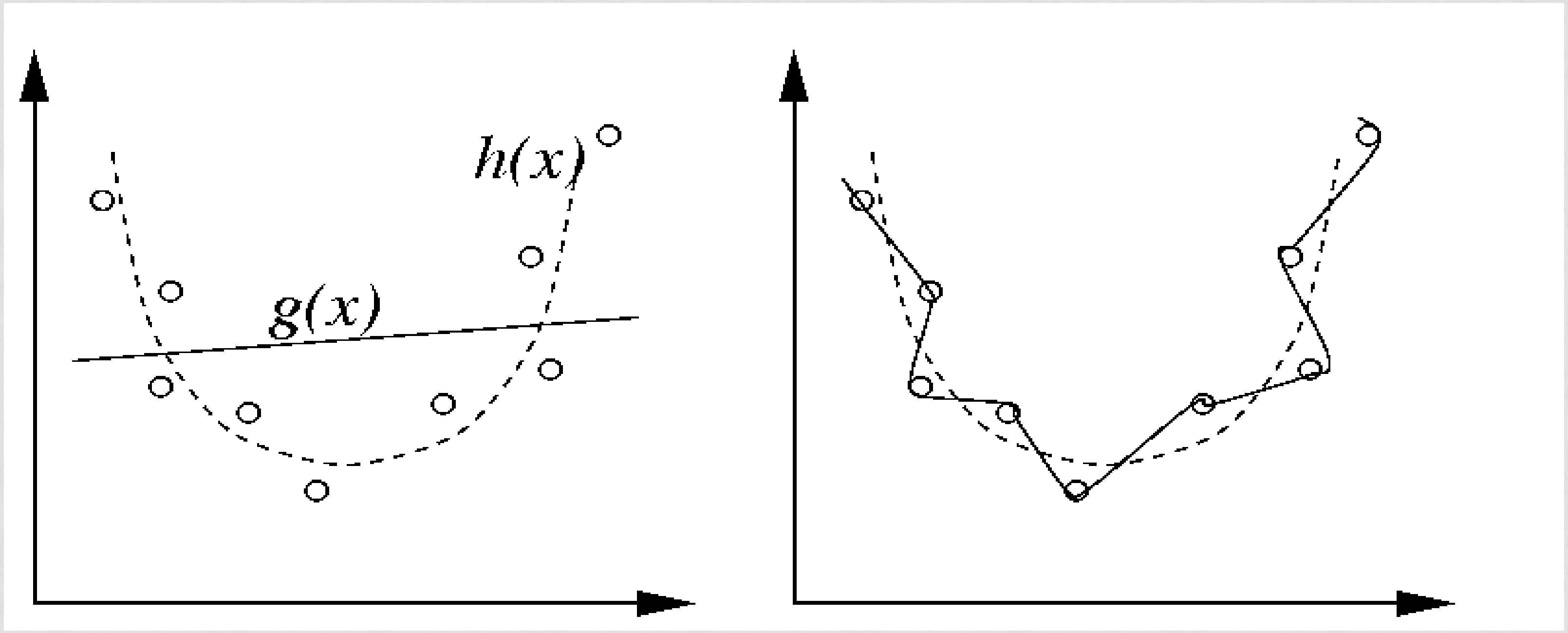
- Let's suppose that the underlying model is a parabola, but instances have some noise
  - For example, **y** might be "temperature", but the thermometer used to measure it is not very accurate



EXAMPLE OF A REGRESSION MODEL NOT GENERALIZING WELL

Underfitting

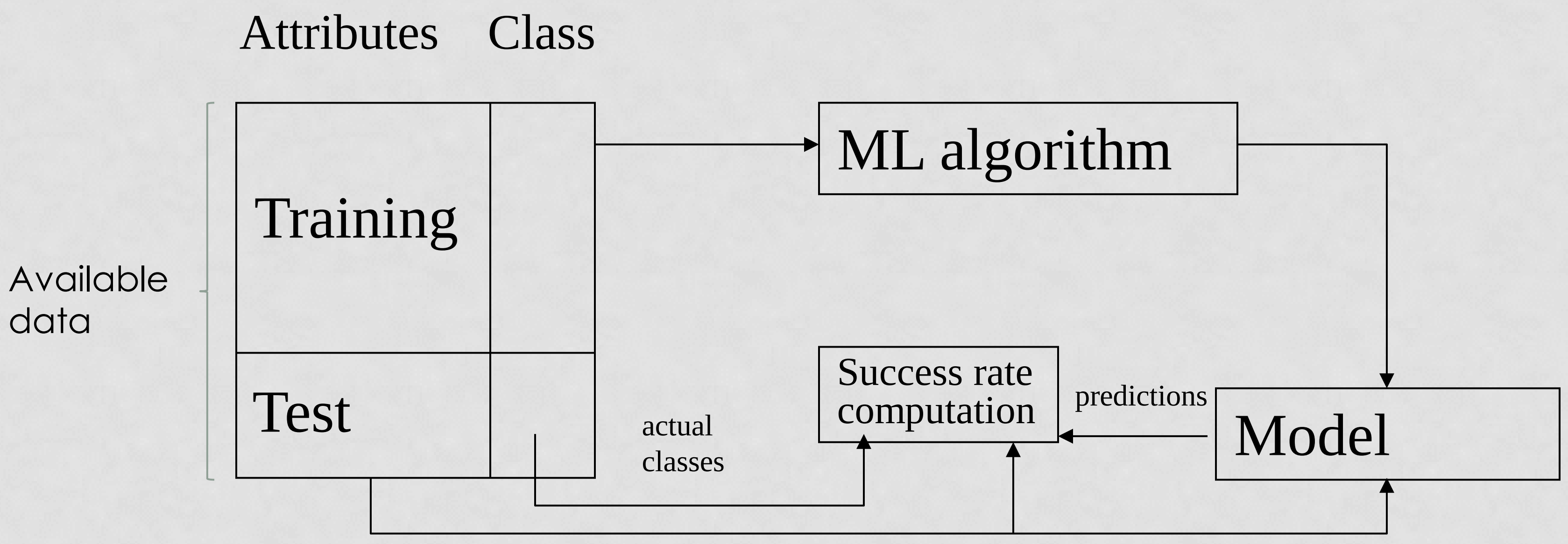
Overfitting



MODEL EVALUATION

- In summary, we want two results from the training process:
  - A model
  - An estimation of its future performance

TRAIN / TEST EVALUATION METHOD



Main problem: if available data is not very large, it is possible that the training and/or the test partitions are **biased**

Example of **bias**: There is data about whether persons like playing tennis. In the training partition, there are 51% men and 49% women and the model learns that it is more likely for men to like playing tennis. Maybe it is a random bias?

Stratified partitions

- Stratified partitions are commonly used, so that the training and test set are representative of the problem
- This is specially true in imbalanced problems (== one of the classes has very few instances)
- Stratification enforces the same proportion of classes in the train and test set
  - Example: if the availabe dataset contains 99% negative instances and 1% positive, stratification will enforce this distribution in the training and test sets.

On the size of the test partition

- 2/3 vs. 1/3 for train and test is arbitrary but commonly used.
- Dilemma:
  - The larger the test, the more accurate the model evaluation
  - But fewer instances are available for training the model
- Alternatives:
  - Use lots of training data. The model might be better, but we will not be completely sure (because the test set will be small)
  - Use few training instances, therefore, bad quality model. The test set will be large, thefore we will be very certain about its future performance being bad.

REPEATED TRAIN / TEST

- Repeat train / test many times
- Given that biases happen randomly, biases that appear in some of the train / test partition will not appear in other train / test partitions. In general random biases will cancel each other after averaging.
- **Method:**
  - **Repeat many times:**
    1. **Sort available data randomly**
    2. **Take the first 2/3 of instances for training and train the model**
    3. **Take the last 1/3 for testing and estimate the model success rate**
  - **Average all test success rates**

REPEATED TRAIN / TEST

- Main problem: the different test partitions might overlap: they are not independent
- In an extreme (and unlikely) case, if all test partitions were exactly the same, computing the average of all of them would be useless
- This extreme case never happens, but there is always some overlap between test partitions

CROSSVALIDATION

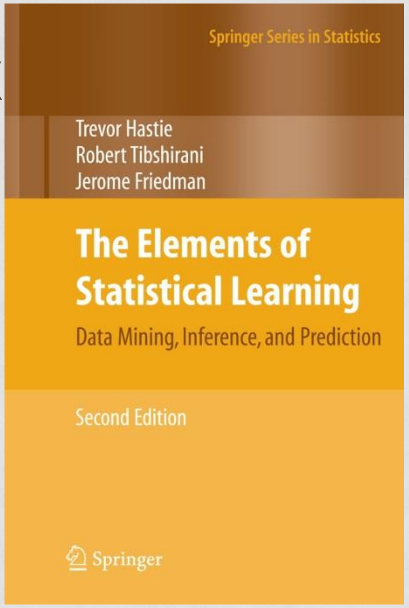
- The available data (originally called "training data") is divided into k folds (k partitions). With k=3, three partitions A, B, and C.
- The process has k steps (3 in this case):
  - Learn model with A, B, and test it with C (T1 = success rate on C)
  - Learn model with A, C, and test it with B (T2 = success rate on B)
  - Learn model with B, C and test it with A (T3 = success rate on A)
  - Success rate TX = (T1+T2+T3)/3
- The final classifier CF is learned **from the whole dataset (A, B, C)**. It is assumed that T is a good estimation of the success rate of CF
- k=10 is commonly used

CROSSVALIDATION: WHAT IS BEING EVALUATED?

- We intend to estimate the success rate of a model
- But with k-fold crossvalidation, k different models are trained!
- Actually, we are evaluating the skill of the algorithm that trains models, independently of the training sample used, rather than a specific model
- In any case, a final classifier CF is trained **with all available data** (A, B and C) and it is **assumed** that TX is an estimation of the success rate of CF

CROSSVALIDATION

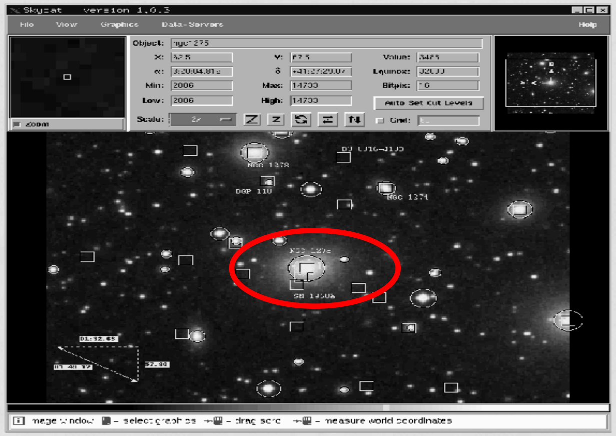
- In any case, a final classifier CF is trained **with all available data** (A, B and C) and it is **assumed** that TX is an estimation of the success rate of CF


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer

- Question: the crossvalidation success rate TX computed with crossvalidation, is it an optimistic or pesimistic estimation of CF? (in other words, is TX better or worse than the actual (unknown) success rate of CF?

- **Important**: crossvalidation is the standard evaluation method, but it is not necessary if the test set is large enough. In the latter case, train/test partitions can be used.

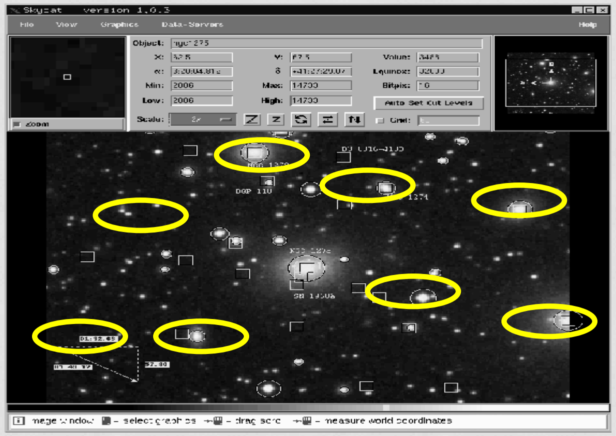
SUMMARY (SO FAR):
We started with this schema:



?

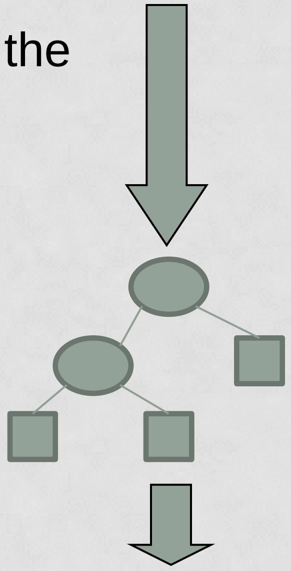
Use of the model

Training data



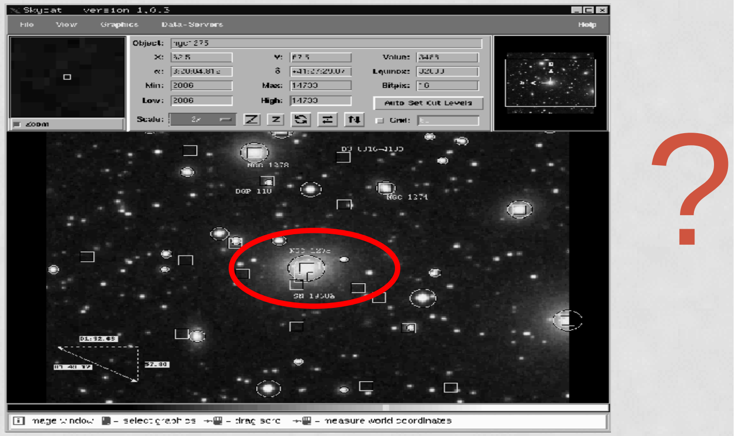
Algorithm

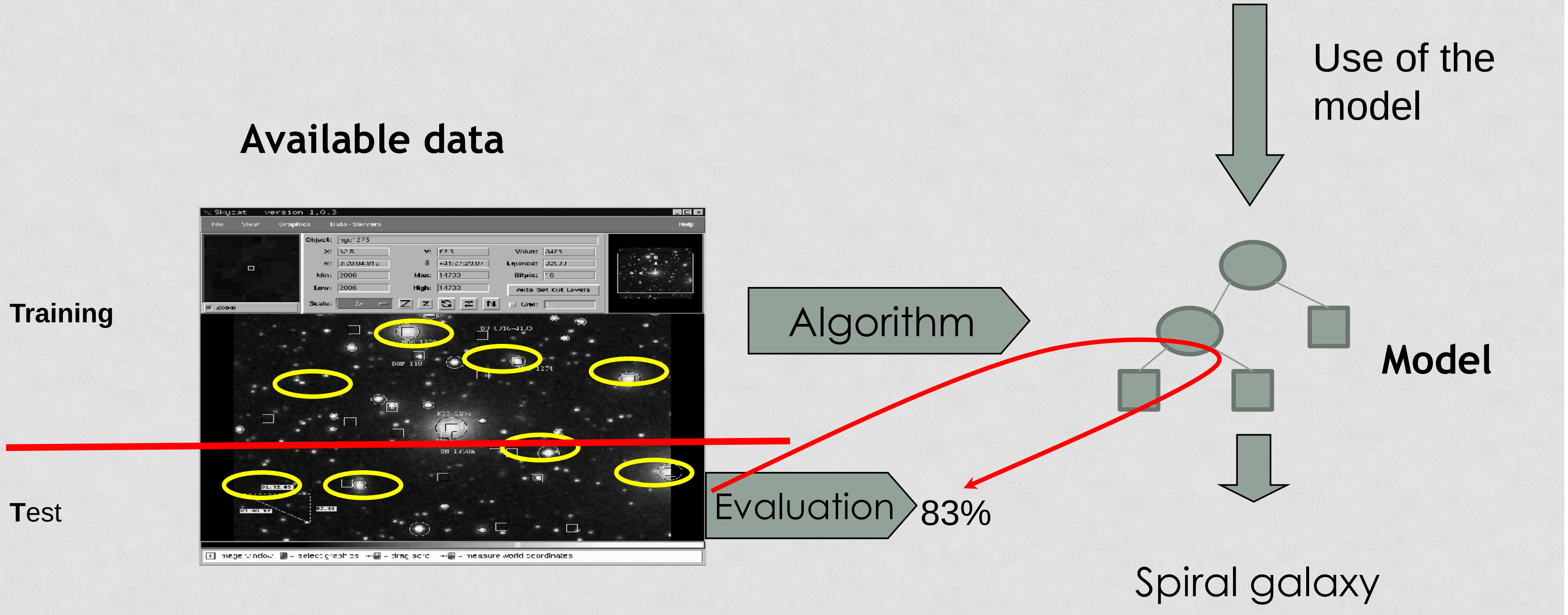
Model



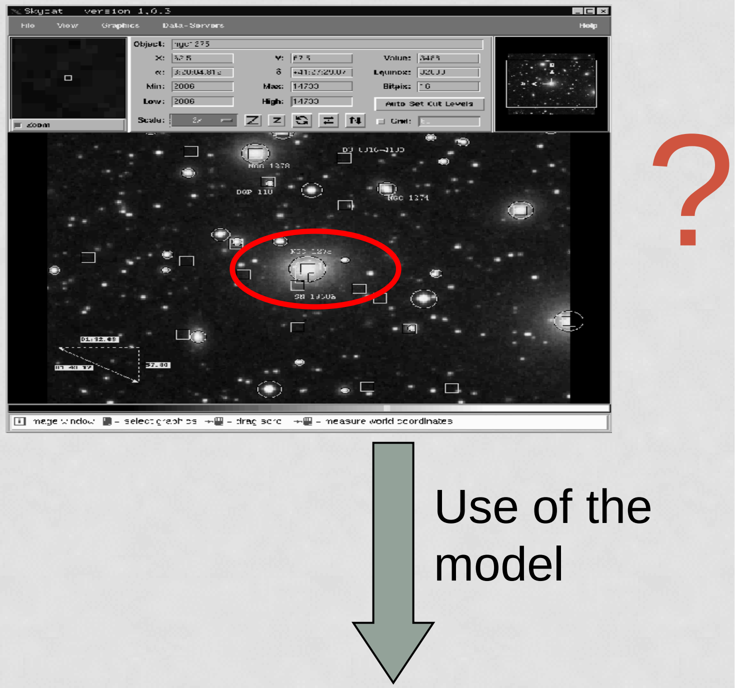
Spiral galaxy

SUMMARY (SO FAR): train/test
Now, in addition to training the model with the training partition, the model is evaluated with the test partition.

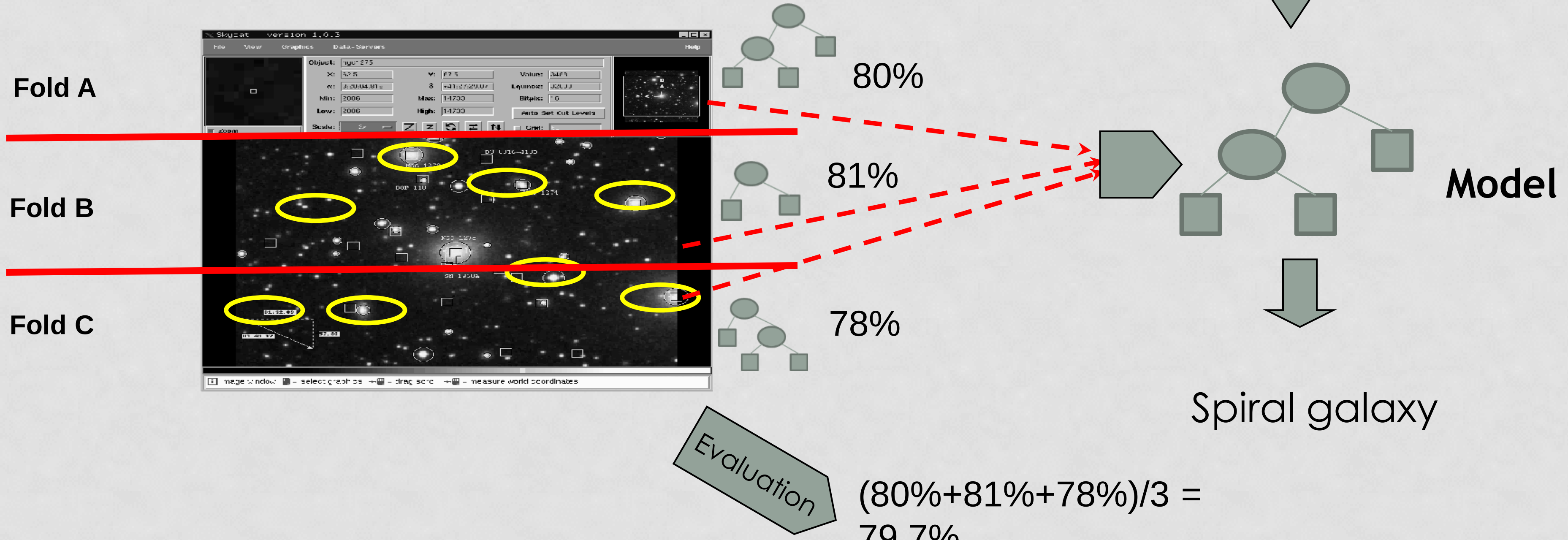




SUMMARY (SO FAR): xvalidation
If evaluation is done with 3-fold crossvalidation:



Available data



(80%+81%+78%)/3 = 79.7%

BASIC CRITERIA FOR EVALUATING CLASSIFICATION MODELS

- In classification problems with 2 classes:
  - What is the minimum success rate that our model should have in order to be considered useful?
  - Why?
  - What if there are M classes?

BASIC CRITERIA FOR EVALUATING CLASSIFICATION MODELS

- For two classes, a useful model has to obtain a success rate larger than 50%.
- Otherwise, tossing a coin would give better results
- For M classes, the baseline result is 1/M * 100

BASIC CRITERIA FOR EVALUATING CLASSIFICATION MODELS

- Imbalanced datasets contain much more data for one of the classes (the majority class) than the other. E.g. most people do not have cancer.
- Let be an imbalanced dataset:
  - 990 negative instances (99%)
  - 10 positive instances (1%)
- What is the minimum success rate that our model should have in order to be considered useful?

BASIC CRITERIA FOR EVALUATING CLASSIFICATION MODELS

- Let be a problem with an imbalanced dataset:
  - 990 negative instances (99%)
  - 10 positive instances (1%)
- What is the minimum success rate that our model should have in order to be considered useful?

- In imbalanced classification problems, a successful model has to obtain a success rate larger that the percentage of the majority class.
- A **trivial classifier** that always says “No” would already obtain 99% success rate!! (by doing nothing)
- Our model has to do better than that

THE CONFUSION MATRIX

- The success rate is a global measure
- But sometimes it is interesting to break down the success rate by class:
  - Success rate of the positive class (+)
  - Success rate of the negative class (-)
- Notice that the success rate = (TP+TN) / (TP+TN+FP+FN)

	Clasiffied as +	Classified as -
Instances actually +	**TP (true positive)**	FN (false negative)
Instances actually -	FP (false positive)	**TN (true negative)**

THE CONFUSION MATRIX

- If "+" = cancer and "-" = not-cancer, which of the two models is more appropriate?
- Notice that the success rate is the same in both cases = (90+60)/200 = 0.75

Decision tree

	Clasiffied as +	Classified as -
Instances actually +	**TP 90**	FN 10
Instances actually -	FP 40	**TN 60**

KNN

	Clasiffied as +	Classified as -
Instances actually +	**TP 60**	FN 40
Instances actually -	FP 10	**TN 90**

THE CONFUSION MATRIX

- Typically, confusion matrices are normalized by dividing by the amount of positive instances and amount of negative instances.
- TPR and TNR can be interpreted as the success rates of the positive and the negative classes, respectively.
- FPR and FNR can be interpreted as the error rates of the negative and the positive classes, respectively

Confusion matrix

	Clasiffied as +	Classified as -
Instances actually +	**TP 90**	FN 10
Instances actually -	FP 40	**TN 60**

Normalized confusion matrix

	Clasiffied as +	Classified as -
Instances actually +	**TPR = 90/(90+10) = 0.9**	**FNR = 10/(90+10) = 0.1**
Instances actually -	**FPR = 40/(40+60) = 0.4**	**TNR = 60/(40+60) = 0.6**

PERFORMANCE MEASURES

- What is meant by "error" (or "success"):
  - In classification problems, **success rate**: count how many times the model gives the correct answer and divide by the number of test data.
    - Error rate = 1 – success rate
    - There are others like area under the ROC curve (AUC or AUROC), Kappa, etc.
  - In regression problems: **root mean squared quadratic error**
    - …

PERFORMANCE MEASURES FOR REGRESSION

- Real values: $\{y_1, \ldots, y_n\}$
- Model predictions: $\{p_1, \ldots, p_n\}$

MSE = Mean Squared Error
RMSE = Root-Mean Squared Error

$$MSE : \frac{(p_1 - y_1)^2 + \ldots + (p_n - y_n)^2}{n}; \quad RMSE = \sqrt{MSE}$$

- RMSE is very common
- But instances with large errors have too much weight on the average.

PERFORMANCE MEASURES FOR REGRESSION

MAE = Mean Absolute Error

$$MAE: \frac{|p_1 - y_1| + \ldots + |p_n - y_n|}{n}$$

- With MAE, instances with large errors do not so much weight on the average (compared to RMSE).
- But both RMSE and MAE have the problem that their scale is relative to the scale of the output variable (y_i)
  - E.g.: if the output variable unit is meters, the RMSE and MAE will be 1000 larger than if the unit is km. That does not mean that the model is 1000 times worse. Just the scale is different.
- This does not happen in classification, where the success rate is an absolute measure.

PERFORMANCE MEASURES FOR REGRESSION

- Relative measures compare our model with the simplest possible regression model: predicting with a constant, the mean of the output variable.
- Relative measures usually range from 1(the model is equivalent to predicting with the mean). But they can be larger than 1 if the model is extremely bad.

RSE = Relative Squared Error
RRSE = Root Relative Squared Error
RAE = Root Absolute Error

$$RSE : \frac{(p_1 - y_1)^2 + \ldots + (p_n - y_n)^2}{(\bar{y} - y_1)^2 + \ldots + (\bar{y} - y_n)^2}; \quad RRSE = \sqrt{RSE}; \quad \bar{y} = \frac{y_1 + \ldots + y_n}{n}$$

$$RAE = \frac{|p_1 - y_1| + \ldots + |p_n - y_n|}{|\bar{y} - y_1| + \ldots + |\bar{y} - y_n|}$$

Note: the coefficient of determination is R^2 = 1- RRSE

Performance measure	Formula
mean-squared error	$\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{n}$
root mean-squared error	$\sqrt{\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{n}}$
mean absolute error	$\frac{\lvert p_1-a_1\rvert+\ldots+\lvert p_n-a_n\rvert}{n}$
relative squared error	$\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{(a_1-\bar{a})^2+\ldots+(a_n-\bar{a})^2}$, where $\bar{a}=\frac{1}{n}\sum_i a_i$
root relative squared error	$\sqrt{\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{(a_1-\bar{a})^2+\ldots+(a_n-\bar{a})^2}}$
relative absolute error	$\frac{\lvert p_1-a_1\rvert+\ldots+\lvert p_n-a_n\rvert}{\lvert a_1-\bar{a}\rvert+\ldots+\lvert a_n-\bar{a}\rvert}$
correlation coefficient	$\frac{S_{PA}}{\sqrt{S_P S_A}}$, where $S_{PA}=\frac{\sum_i(p_i-\bar{p})(a_i-\bar{a})}{n-1}$, $S_p=\frac{\sum_i(p_i-\bar{p})^2}{n-1}$, and $S_A=\frac{\sum_i(a_i-\bar{a})^2}{n-1}$

* p are predicted values and a are actual values.

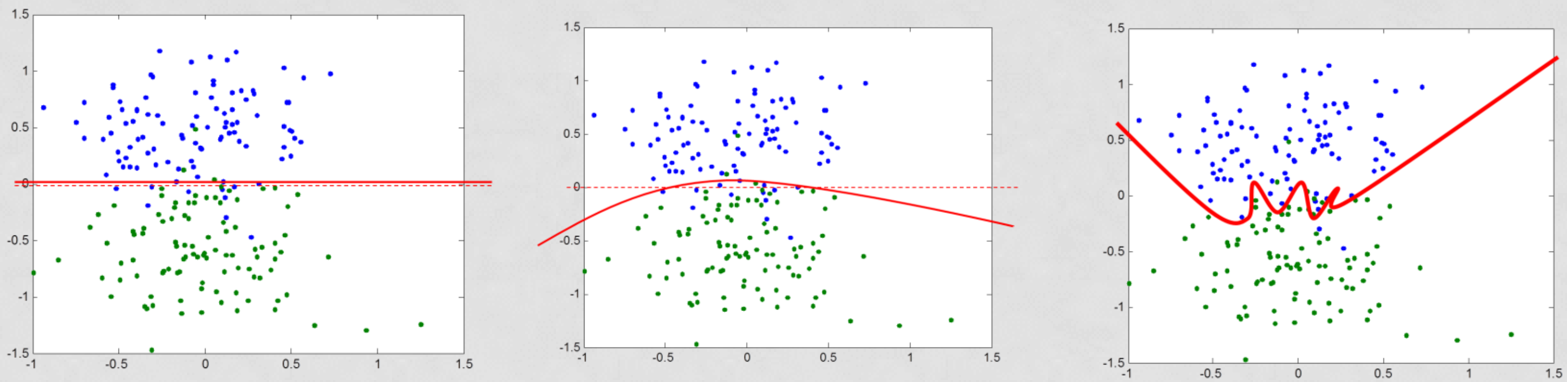
HYPERPARAMETER TUNING

HYPER-PARAMETERS

- Each machine learning algorithm has one or several parameters (called hyper-parameters)
- For instance, KNN has *K* (the number of neighbors)
- For instance, decision trees have:
  - *max_depth*: the maximum depth of the tree
  - *min_samples_split*: the minimum number of instances to split a node (the default value is 2: if a node contains fewer than 2 instances, the node is not split)
- Finding the correct value of a hyper-parameter may result in improved performance of the classifier

HYPER-PARAMETERS

- Finding the correct value of a hyper-parameter may result in improved performance of the classifier
- Hyperparameters control, directly or indirectly, the complexity of a classifier
- In general, the more complex a model is, the more likely it will overfit the data (but if it is not complex enough, it will underfit the data)
- Example of a complex classifier (the curve is allowed to turn around many times):

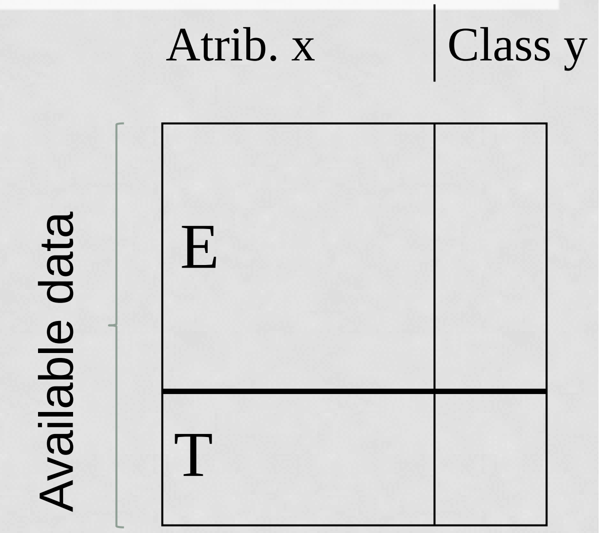


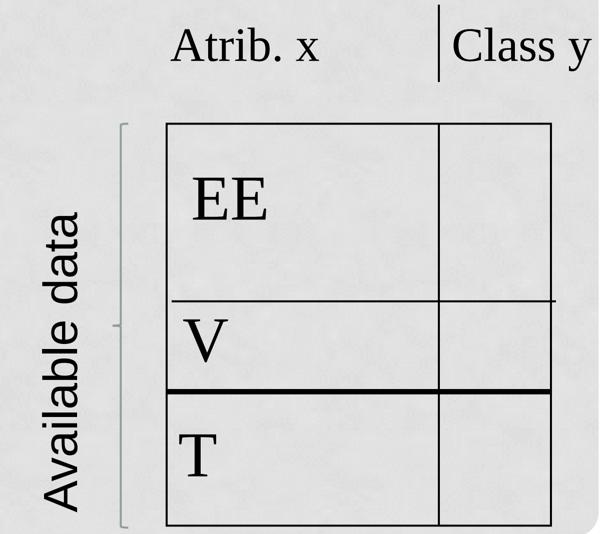
HYPER-PARAMETER TUNING

- In decision trees, if *max_depth* is very large or *min_samples_split* is very small, the resulting tree will be large, and therefore, complex
- We can try to give appropriate values to the hyper-parameters by hand (trying and testing)
- But there is an automatic way of tuning the hyperparameters which is called **grid-search**

HYPER-PARAMETER TUNING

- Let's suppose that the train/test methodology is used for evaluation
- Let's suppose that our algorithm has just one hyper-parameter: *maximum depth of the tree*.
- First, a set of values has to be determined:
  - e.g. (2, 4, 6, 8, 10).
- What is the best depth? Different models with different depths will be trained and evaluated. The best one will be kept.
- But the test set cannot be used for evaluating hyperparameters, because it is reserved to evaluate the final model.
- Therefore E will be divided into training (EE) and validation (V). V will be used to evaluate each of the max depth values.
- Once the best depth has been determined, a final model will be trained with E=EE+V and evaluated with T.

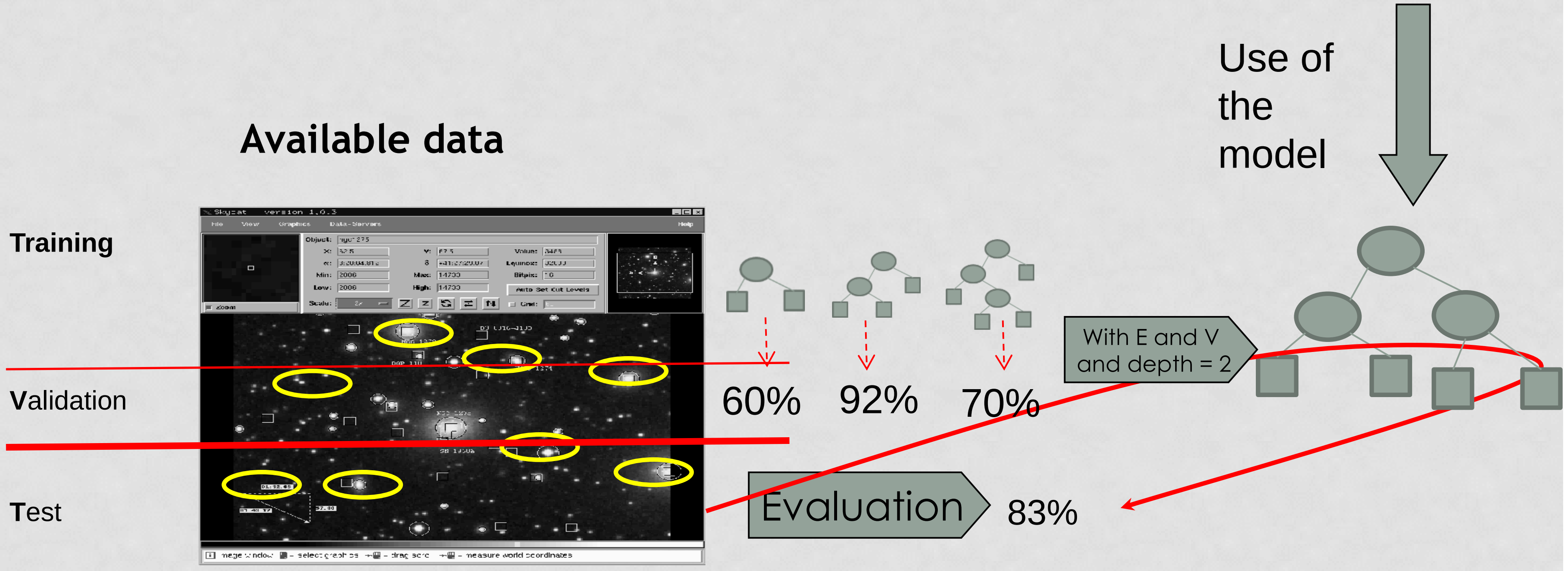




SUMMARY SO FAR:

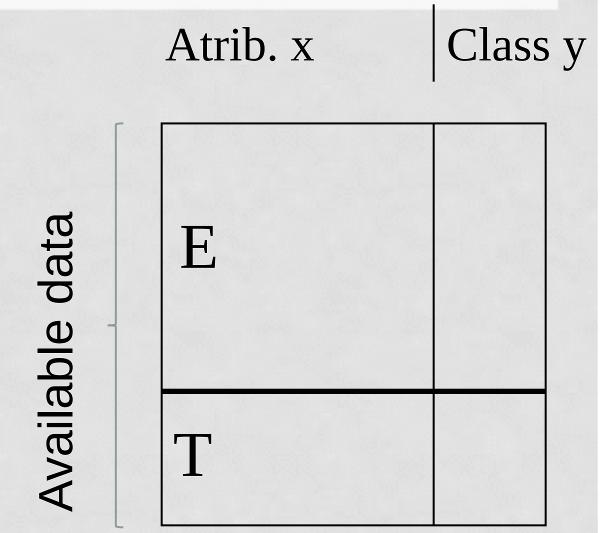
- Let's suppose we use the Train/Test methodology
- Hyper-parameter tuning with validation partition:

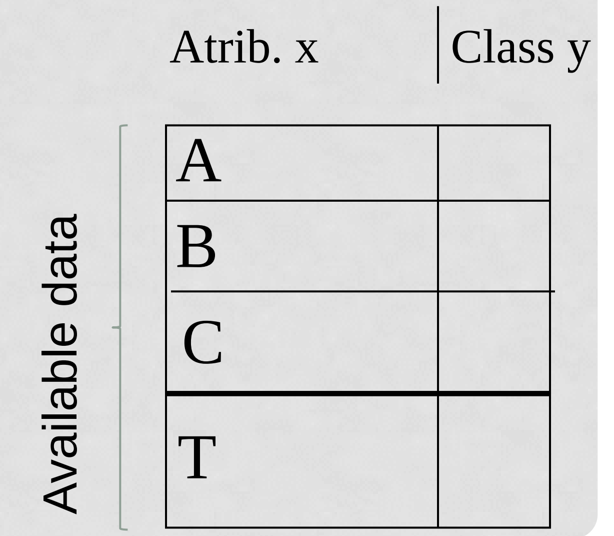
Available data



HYPER-PARAMETER TUNING

- A more accurate method (but computationally very expensive) is to use crossvalidation to determine the best depth.
- The figure shows how E has been split into three folds: A, B, and C.
- Thefore:
  - For max depth 2:
    - A model is trained with A+B and evaluated with C
    - A model is trained with A+C and evaluated with B
    - A model is trained with B+C and evaluated with A
    - The average of the three evaluations is computed
  - Similarly for max depth 4
  - …
  - Similarly for max depth 10
  - The best max depth is chosen, and a final model is trained with E=A+B+C
  - The model is finally evaluated with T





HYPER-PARAMETER TUNING

- If there is more than one hyper-parameter, **grid search** is typically used.
- All possible combinations of hyper-parameters is checked.
- Computationally expensive.

GRID SEARCH

MAX_DEPTH	2	4	6	8
MIN_SAMPLES				
2	(2,2)	(2,4)	(2,6)	(2,8)
4	(4,2)	(4,4)	(4,6)	(4,8)
6	(6,2)	(6,4)	(6,6)	(6,8)

Grid search means: try all possible combinations of values for the two (or more) hyper-parameters. For each one, carry out a train/validation or a crossvalidation, and obtain the success rate. Select the combination of hyper-parameters with best success-rate.

MAX_DEPTH	2	4	6	8
MIN_SAMPLES				
2	70%	75%	76%	68%
4	72%	73%	**81%**	70%
6	68%	70%	71%	67%

RANDOM SEARCH OF HYPER-PARAMETERS

- Grid-search is obviously computationally expensive
- Other possibilities:
  - Random search: combination of hyper-parameters are checked randomly (i.e. not all possible combinations are checked).
  - Tune the first hyper-parameter, leaving the other fixed with reasonable values. Then tune the second one, and son on.