CHAPTER 19:

EXPERİMENT DESİGN WITH CROSS VALIDATION Section 19.1-19.7

Introduction

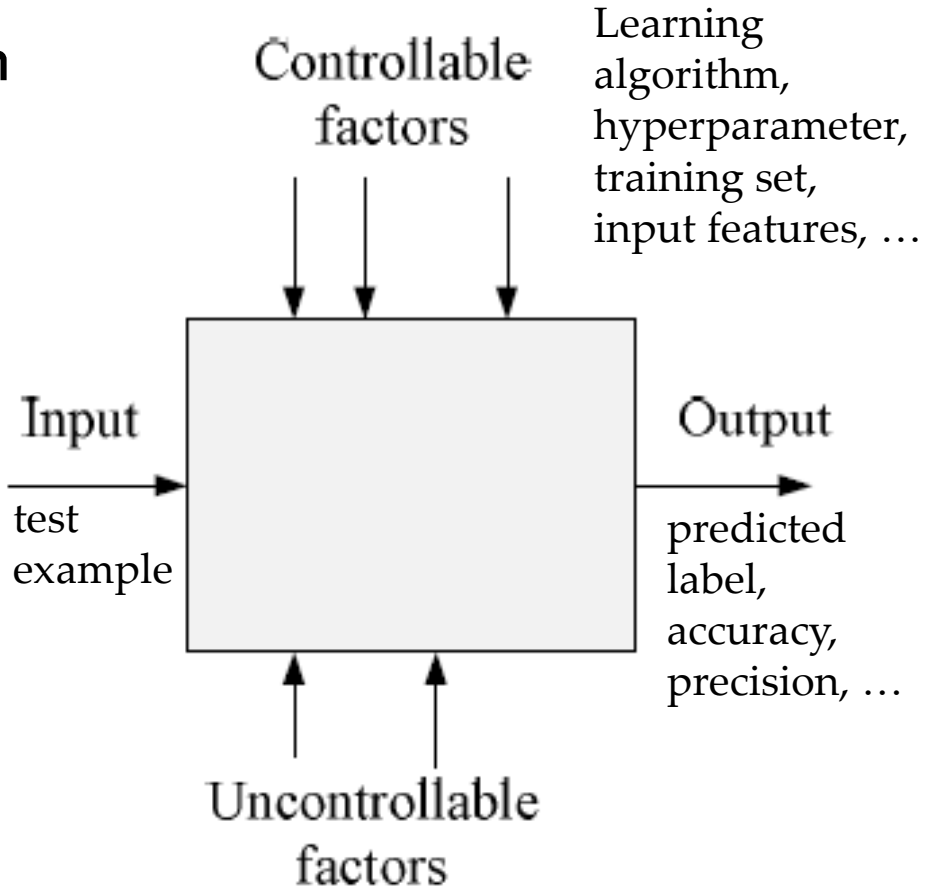
- Questions:
  - Assessment of the expected error of a learning algorithm: If training error is 3%, is the error rate on test data < 2%?
  - Comparing the expected errors of two algorithms: Is *k*-NN more accurate than deep neural networks? 1-NN v.s. 5-NN
- Need multiple runs
  - Outlier, randomness
- Training/validation/test sets
- Resampling methods: *K*-fold cross-validation

Algorithm Preference

- Criteria (Application-dependent):
  - Misclassification error, or risk (loss functions)
  - Training time/space complexity
  - Testing time/space complexity
  - Interpretability
  - Easy programmability
- Cost-sensitive learning

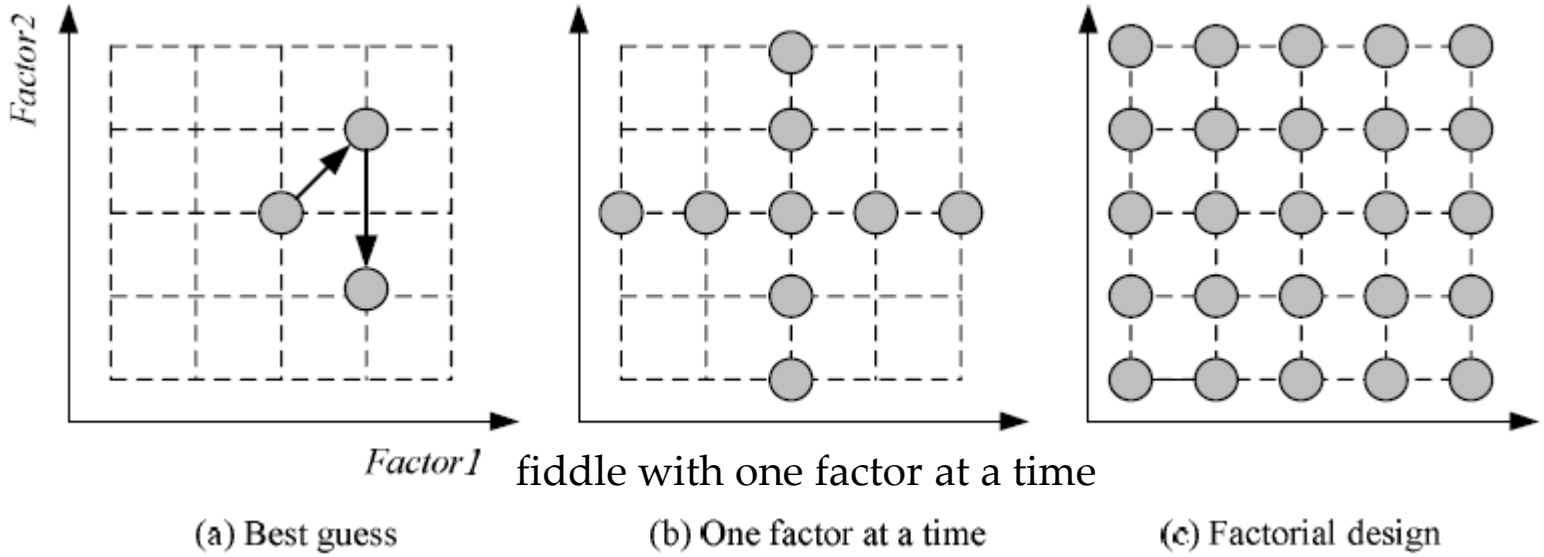
ML Experiment: Factors and Response

- Response function based on output to be maximized
- Depends on controllable factors
- Uncontrollable factors introduce randomness
- Aim: Find the configuration of controllable factors that optimizes response and *minimally affected by uncontrollable factors*



Strategies of Experimentation

How to search the factor space?



fiddle with one factor at a time

assume no interaction btw factors

complexity?

Response surface design for approximating and maximizing the response function in terms of the controllable factors

Guidelines for ML experiments

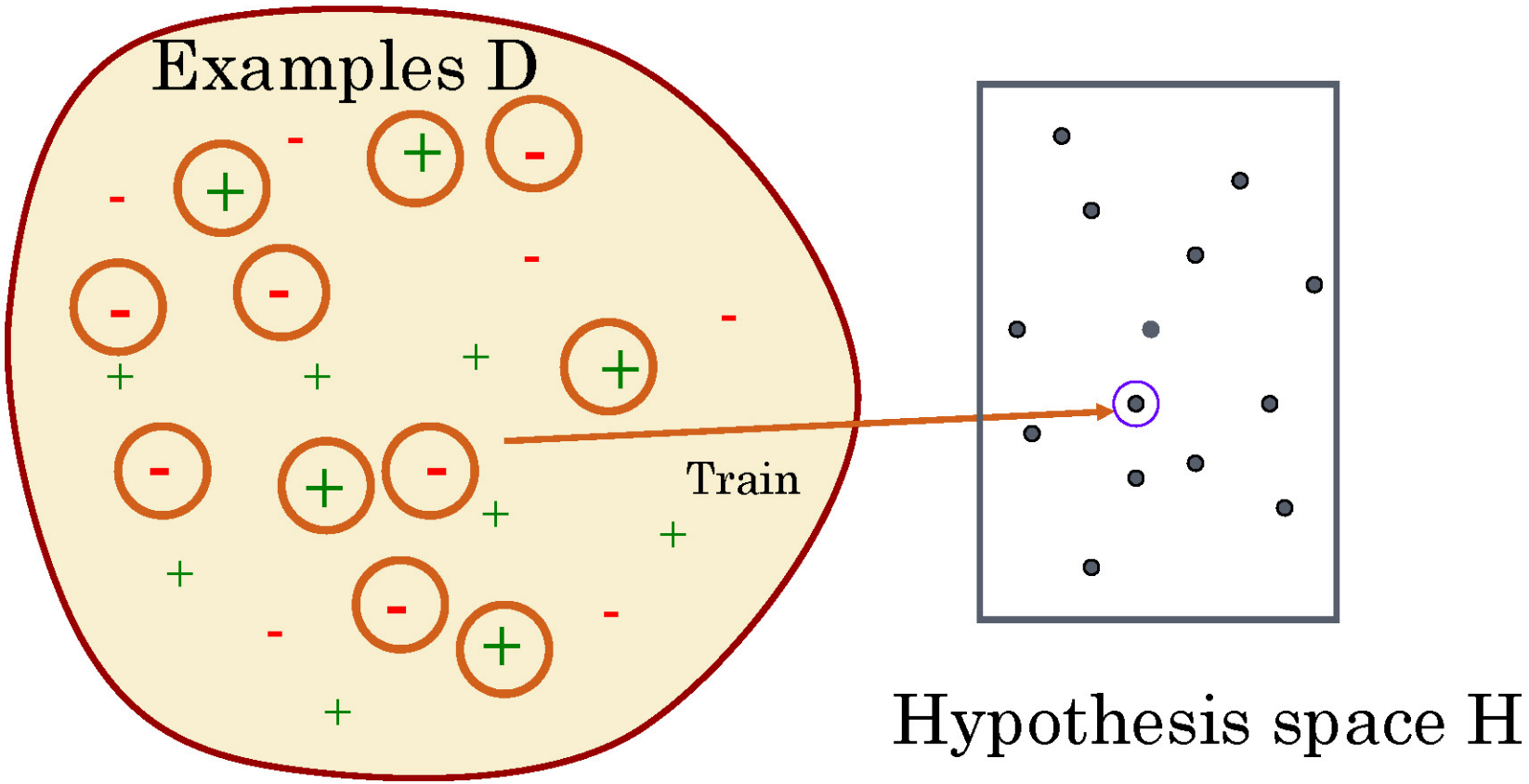
A. Aim of the study
B. Selection of the response variable
C. Choice of factors and levels
D. Choice of experimental design
E. Performing the experiment
F. Statistical analysis of the data
G. Conclusions and recommendations

Assessing Performance of a Learning Algorithm

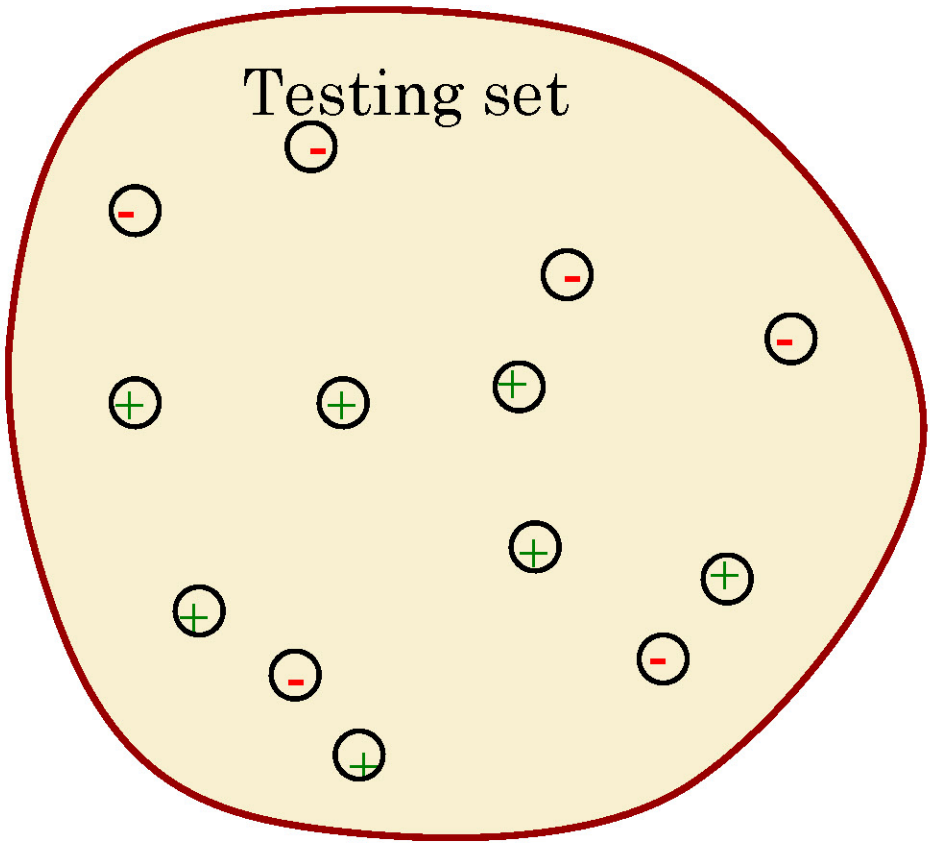
- Dataset is typically given: cannot request more
- Task: compare which learning algorithm is better
- Partition the dataset into two parts:
  - Training set: train the two algorithms on it, get two models
  - Test set: make predictions on it using the two learned models
  - Check which model yields lower test error
  - Error rate:

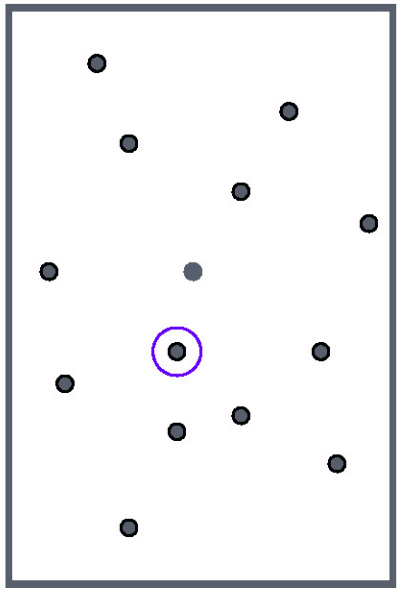
$$\frac{1}{\#test} \sum_{i=1}^{\#test} \delta(\text{PredictLabel}(i) \neq \text{TrueLabel}(i))$$

- Split original set of examples, train



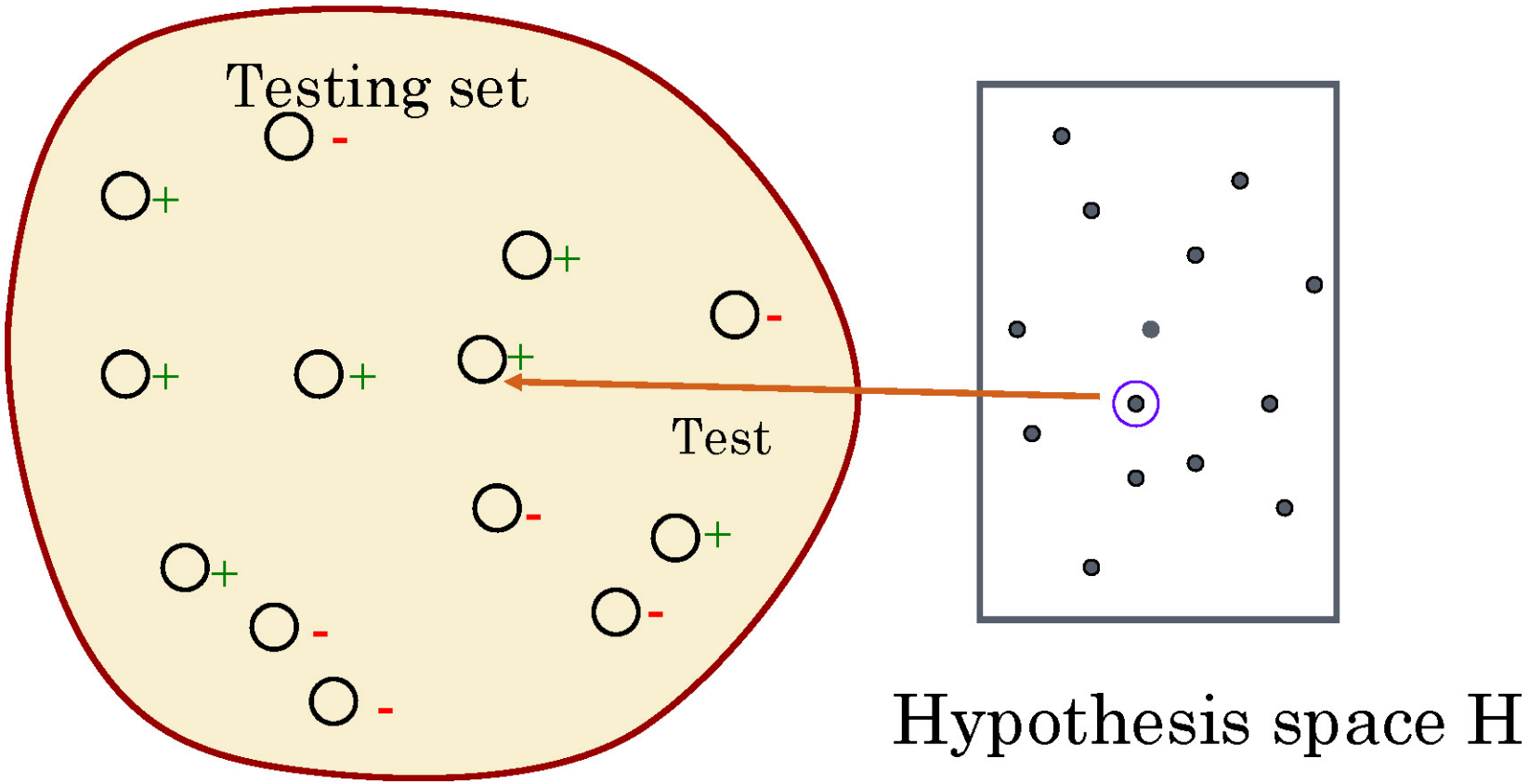
- Evaluate model on testing set



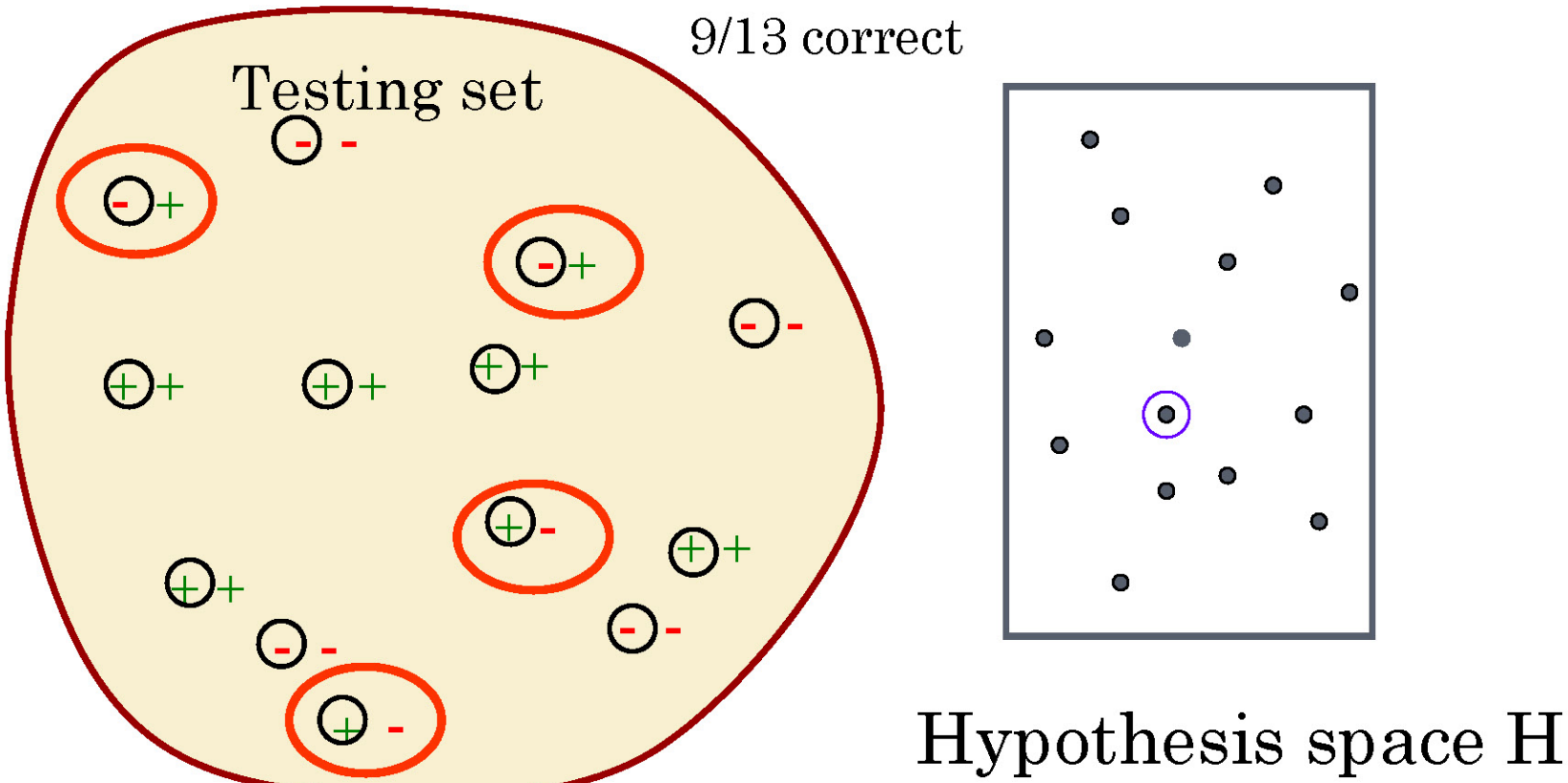


Hypothesis space H

- Evaluate/Predict labels on testing set



- Compare true label against prediction

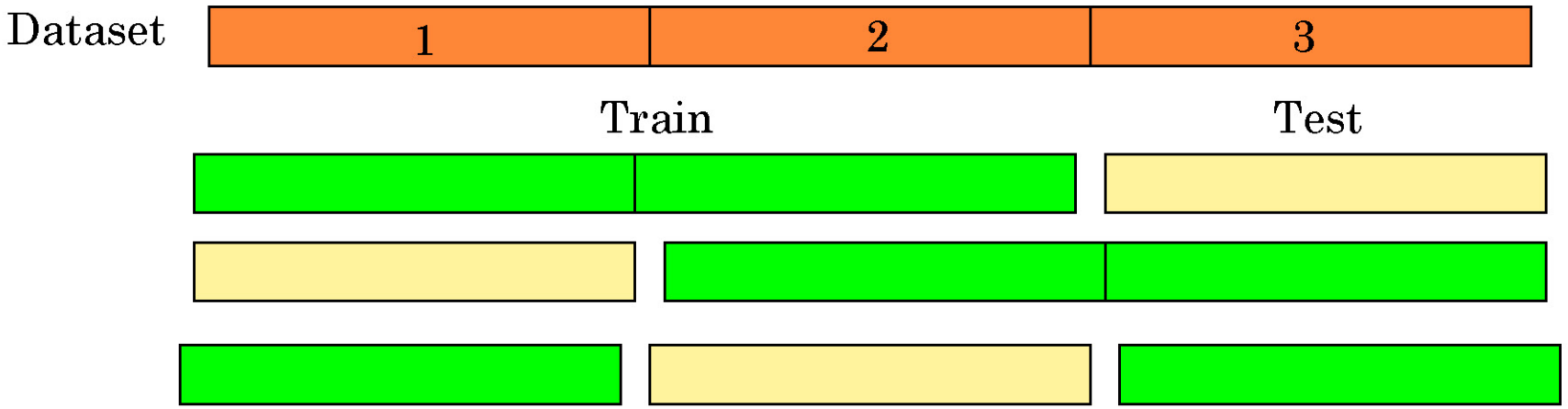


Assessing Performance of a Learning Algorithm

- Dataset is typically given: cannot request more
- Task: compare which learning algorithm is better
- Partition the dataset into two parts:
  - Training set: train the two algorithms on it, get two models
  - Test set: make predictions on it using the two learned models
  - Check which model yields lower test error
  - Error rate: $$\frac{1}{\#test} \sum_{i=1}^{\#test} \delta(\text{PredictLabel}(i) \neq \text{TrueLabel}(i))$$
- Problem:
  - Test set needs to be large enough (wastes training examples)
  - *Solution: Cross-validation*
    repeated use of the same dataset, but split differently

Common Splitting Strategies

- k-fold cross-validation



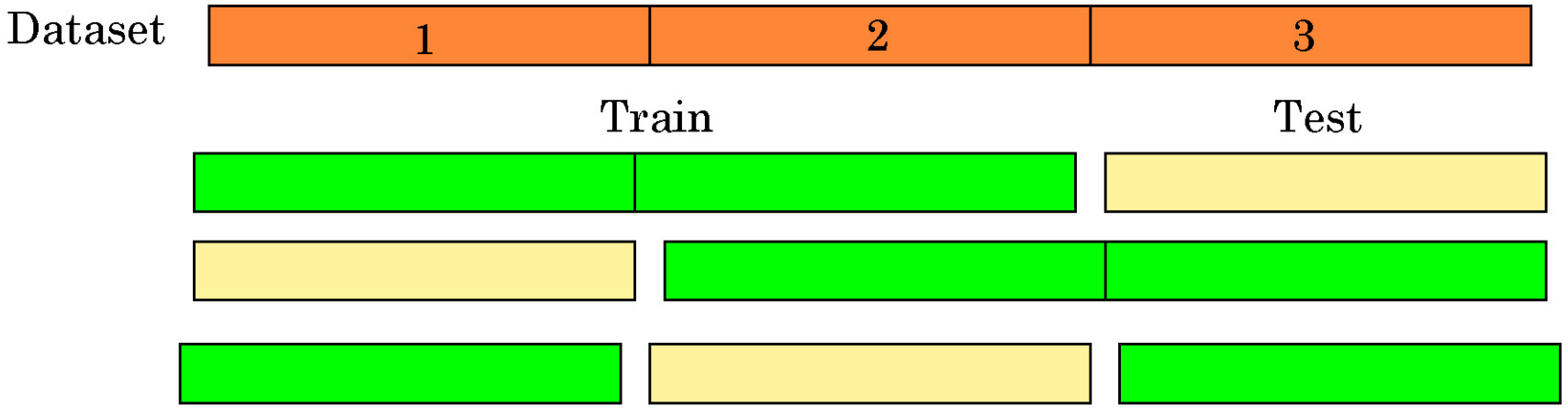
- One error for each fold
- Average error reported

Benefits:

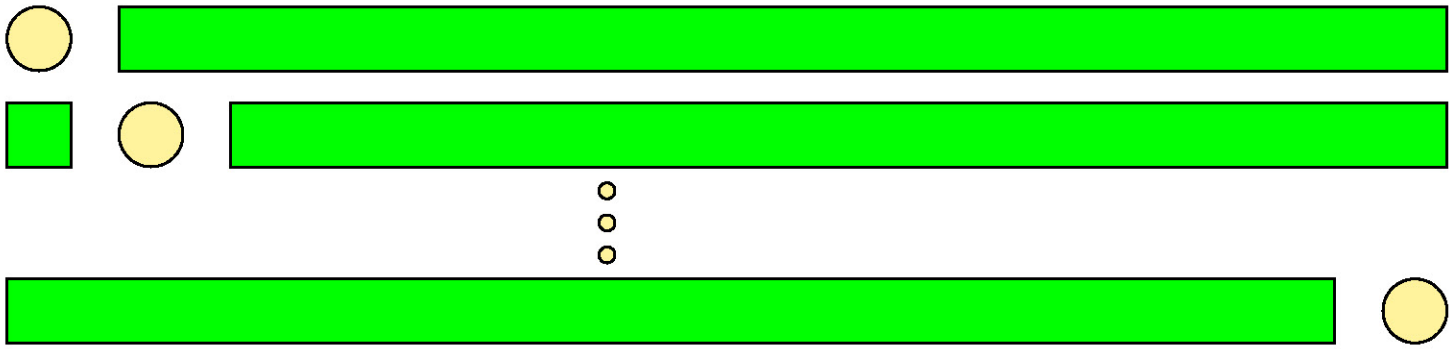
1. Keep training/validation sets as large as possible
2. Keep the overlap between different sets as small as possible

Common Splitting Strategies

- k-fold cross-validation



- Leave-one-out (*n*-fold cross validation)



Computational complexity

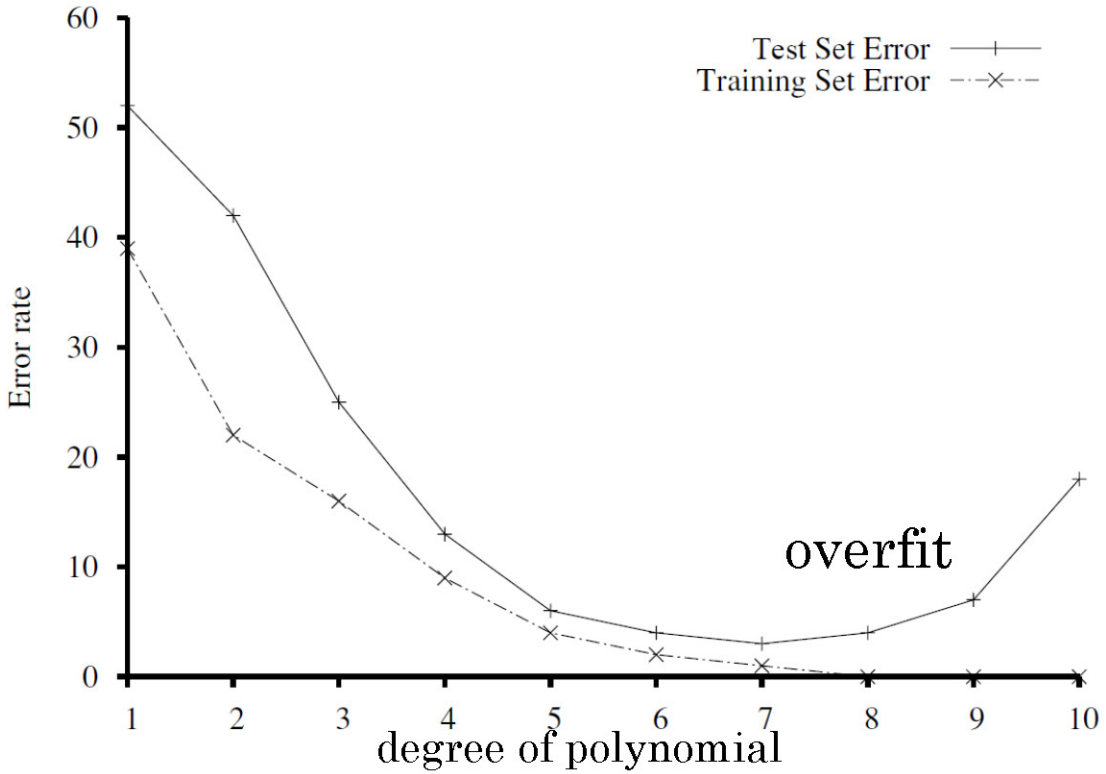
- k-fold cross validation requires
  - Train for k times, each time on n(k-1)/k data points
  - Test for k times, each time on n/k data points
  - (There are efficient ways of computing L.O.O. estimates for some models)
- Average results reported
- Stratified CV
  - Each partition has label proportions similar to that of the whole dataset. Suppose 20% examples are positive in the whole data set.

Partitions			
	20% positive	20% positive	20% positive

  - Randomly permute the set of positive and negative examples **separately**
  - First partition = first 1/3 of positive + first 1/3 of negative
  - Second partition = second 1/3 of positive + second 1/3 of negative

Model selection

- Many algorithms have a hyper-parameter to tune
  - E.g. k-nearest neighbor, power of polynomial
  - Must be chosen wisely

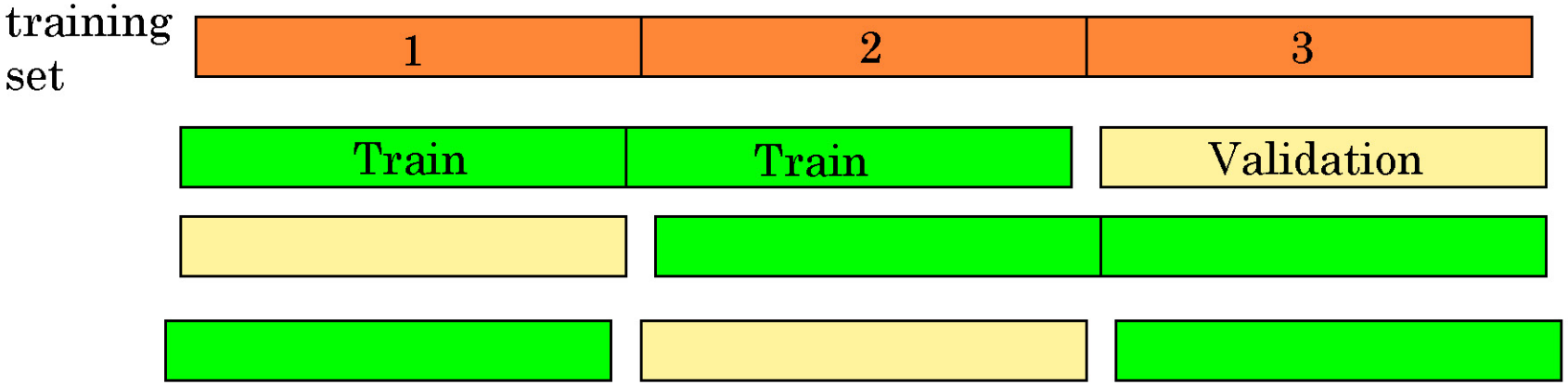


Model selection

- Many algorithms have a hyper-parameter to tune
  - E.g. Depth of tree, max #nodes, power of polynomial
- How to pick that number?
  - Try different hyper-parameter numbers on a training set
  - Each yields a model for a hyper-parameter value
  - Apply the models to a test set
  - Get test errors (one for each model)
  - Pick the model with the lowest test error
  - Peeeeek
  - Solution: CV for model selection

Cross validation for Model selection

- Partition the data set into two sets: training + test
- Partition the **training set** into *k* folders (test set not shown below)



- Train on (k-1) folders, using each value of hyper-parameter
- Compute the error on the validation set, averaged over k folds
- Pick the hyper-parameter that gives the lowest validation error
- Use the hyper-parameter to train on the **whole training set**
- Report the **test set** as the performance score of the algorithm

function CROSS-VALIDATION-WRAPPER($Learner, k, examples$) **returns** a model

local variables: $errT$, an array, indexed by $size$, storing training-set error rates
$errV$, an array, indexed by $size$, storing validation-set error rates

for $size = 1$ **to** ∞ **do** (size can be the degree of polynomial)
$errT[size], errV[size] \leftarrow$ CROSS-VALIDATION($Learner, size, k, examples$)
if $errT$ has converged **then do**
$best\_size \leftarrow$ the value of $size$ with minimum $errV[size]$
return $Learner(best\_size, examples)$

finally train on all examples except test examples

---

function CROSS-VALIDATION($Learner, size, k, examples$) **returns** two values:
average training set error rate, average validation set error rate

$fold\_errT \leftarrow 0$; $fold\_errV \leftarrow 0$
for $fold = 1$ *to* k *do*
$training\_set, validation\_set \leftarrow$ PARTITION($examples, fold, k$)
$h \leftarrow Learner(size, training\_set)$
$fold\_errT \leftarrow fold\_errT +$ ERROR-RATE($h, training\_set$)
$fold_errV \leftarrow fold_errV +$ERROR-RATE($h, validation\_set$)
return $fold\_errT/k, fold\_errV/k$

Figure 18.8 of AIMA 3rd ed

Performance Measures

confusion matrix

	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

(authentication problem of log in by voice, imposter)

- Extension to more than two classes

		Predicted Class		
		1	2	3
True class	1			
	2	count of "predict 1 but 2 indeed"	count of "correctly predicting 2"	
	3			

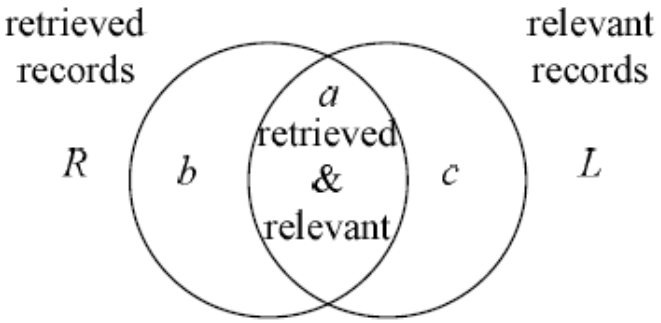
Performance Measures

confusion matrix

	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

- Error rate = # of errors / # of instances = (FN+FP) / N
- Recall = # of found positives / # positives
  = TP / (TP+FN) = sensitivity = tp-rate = hit rate
- Precision = # of found positives / # found
  = TP / (TP+FP)
- False alarm rate = fp-rate = FP / (FP+TN)
- Specificity = 1 – False alarm rate
  = # of found negative / # negative = TN / (TN+FP)

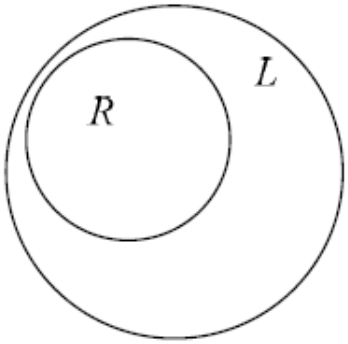
Precision and Recall



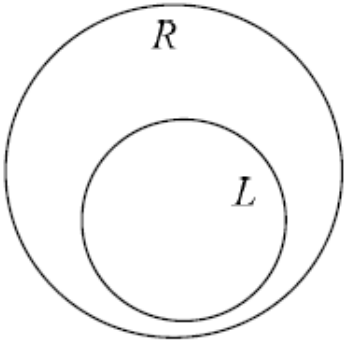
$$\text{Precision: } \frac{a}{a + b}$$

$$\text{Recall: } \frac{a}{a + c}$$

(a) Precision and recall



(b) Precision = 1



(c) Recall = 1

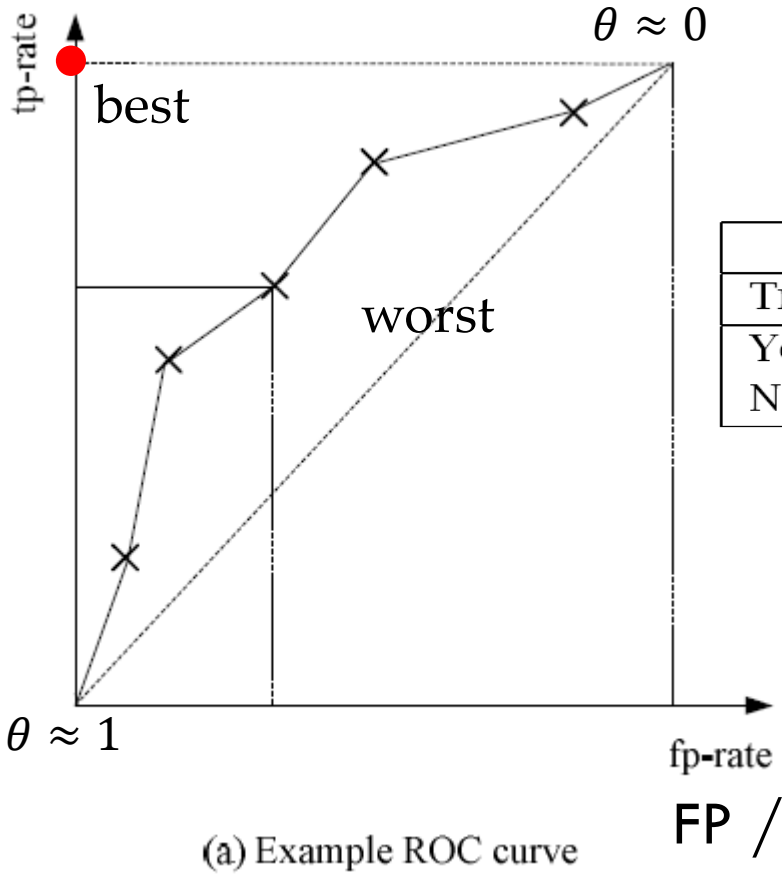
Receiver operating characteristics (ROC) Curve, Area under curve (AUC)

system returns $\hat{P}(C_1|x)$

choose "positive" if $\hat{P}(C_1|x) > \theta$

$\theta \approx 1$: hardly positive

TP / #positive



(a) Example ROC curve

FP / #negative

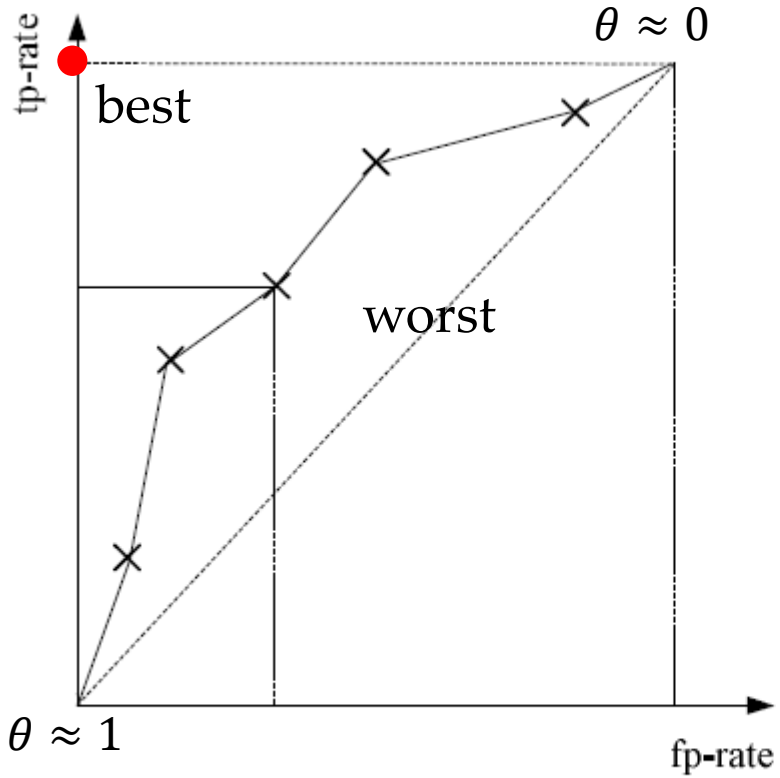
	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

Receiver operating characteristics (ROC) Curve, Area under curve (AUC)

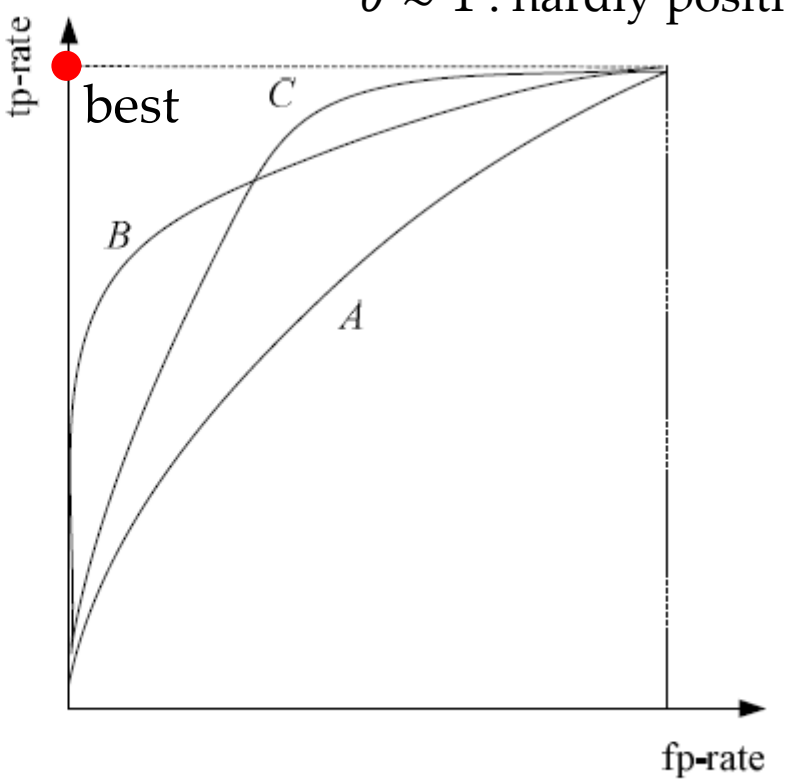
system returns $\hat{P}(C_1|x)$

choose "positive" if $\hat{P}(C_1|x) > \theta$

$\theta \approx 1$: hardly positive



(a) Example ROC curve



(b) Different ROC curves for different classifiers