50.007 Machine Learning

Ensemble Models

Roy Ka-Wei Lee
Assistant Professor, DAI/ISTD, SUTD

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

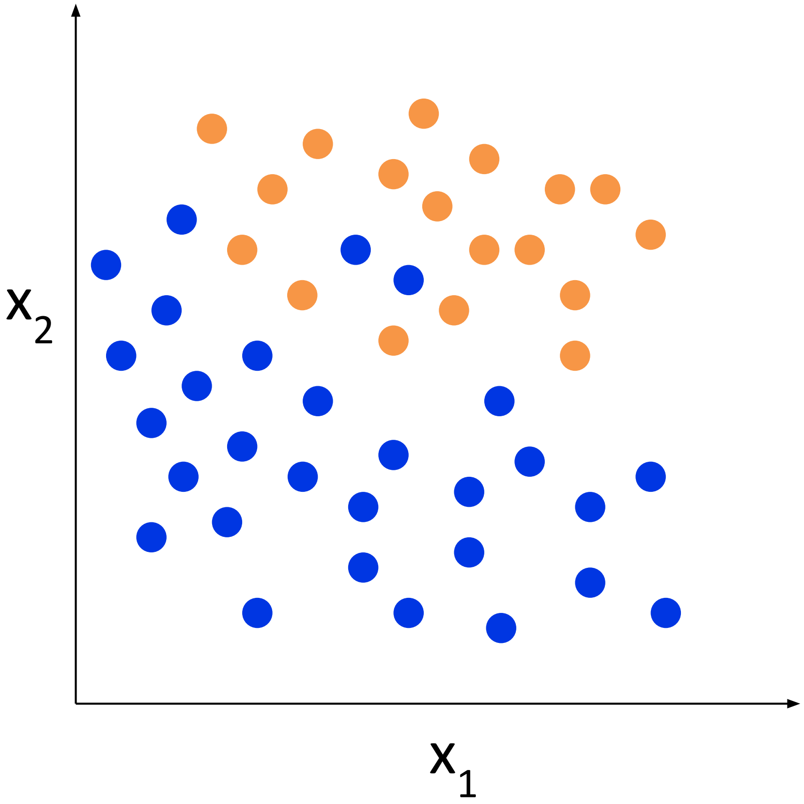
Outline

- Generalization
  - Underfitting and Overfitting
- Ensemble Classifiers
  - Bagging
    - Random Forest
  - Boosting
- Model Evaluation

Generalization

- Training data: $\{x_i, y_i\}$
  - Examples that we use to train our model
  - E.g. past records of days labelled with Roy goes/not go fishing
- Future/test data: $\{x_i, \hat{y}_i\}$
  - Examples that our model has not seen before
  - E.g. information about tomorrow to predict if Roy goes fishing
- Want to do well on future data not training data!
  - Not very useful to do well on training data; we already know the label
  - Easy to be perfect on training data
  - Doing well in training does not mean will do well on future data!

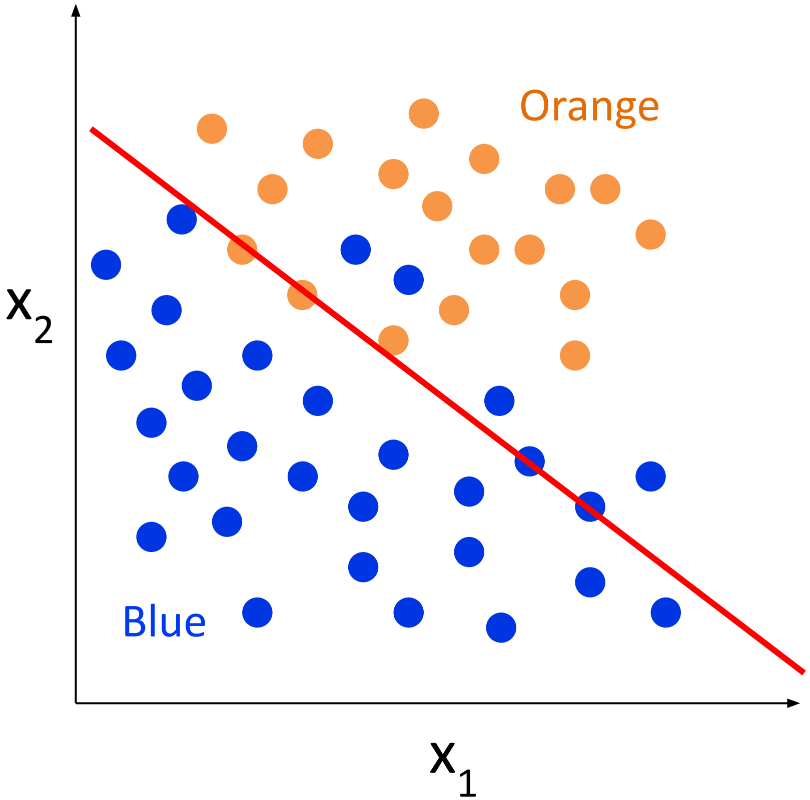
Classification Example



- Find a way to classify the points

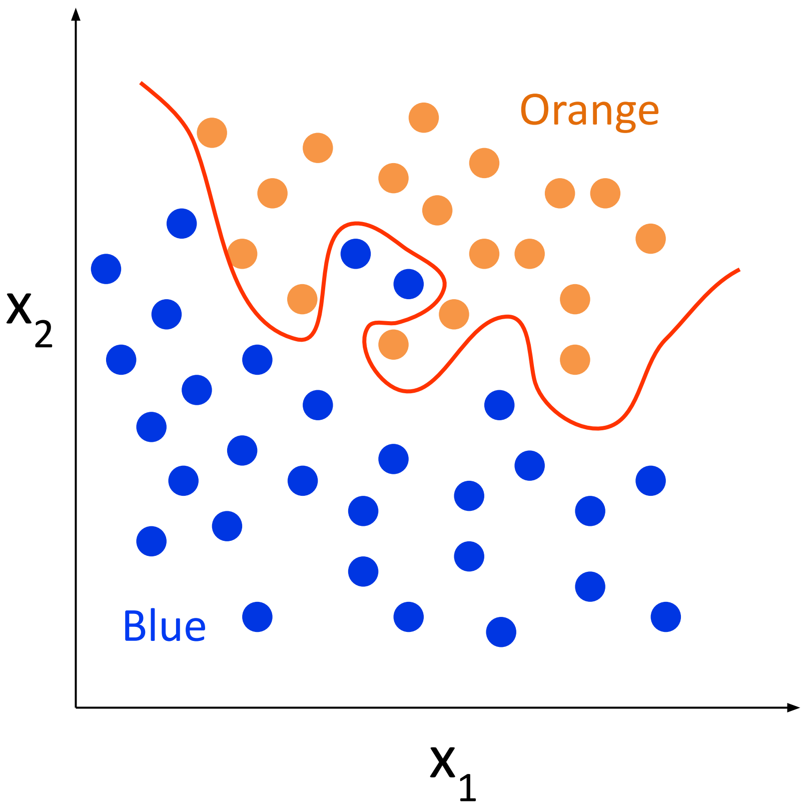
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Classification Example



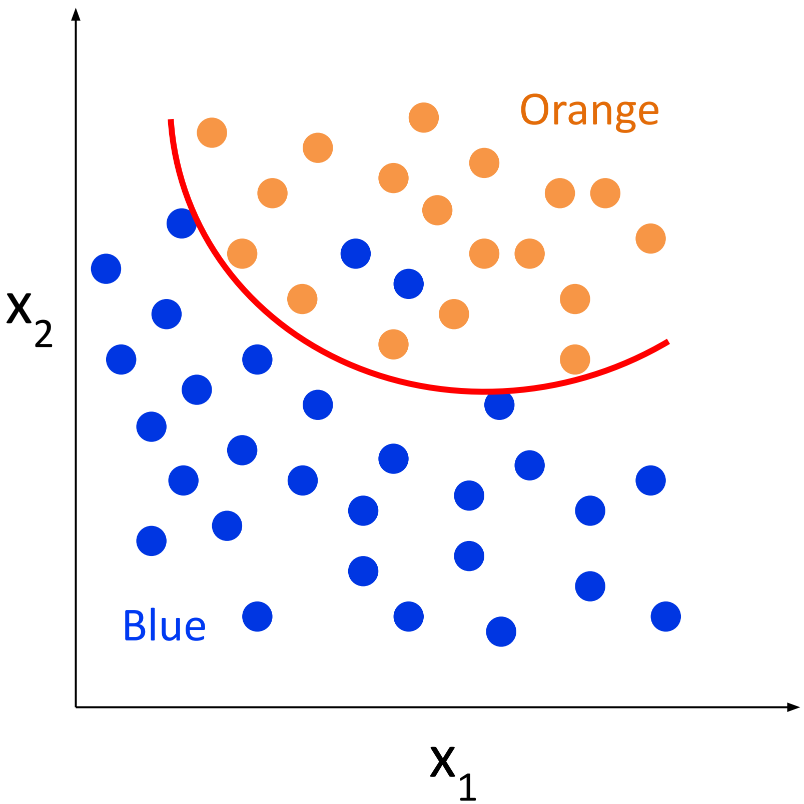
- Simple solution
- Unable to capture all salient pattern in data

Classification Example



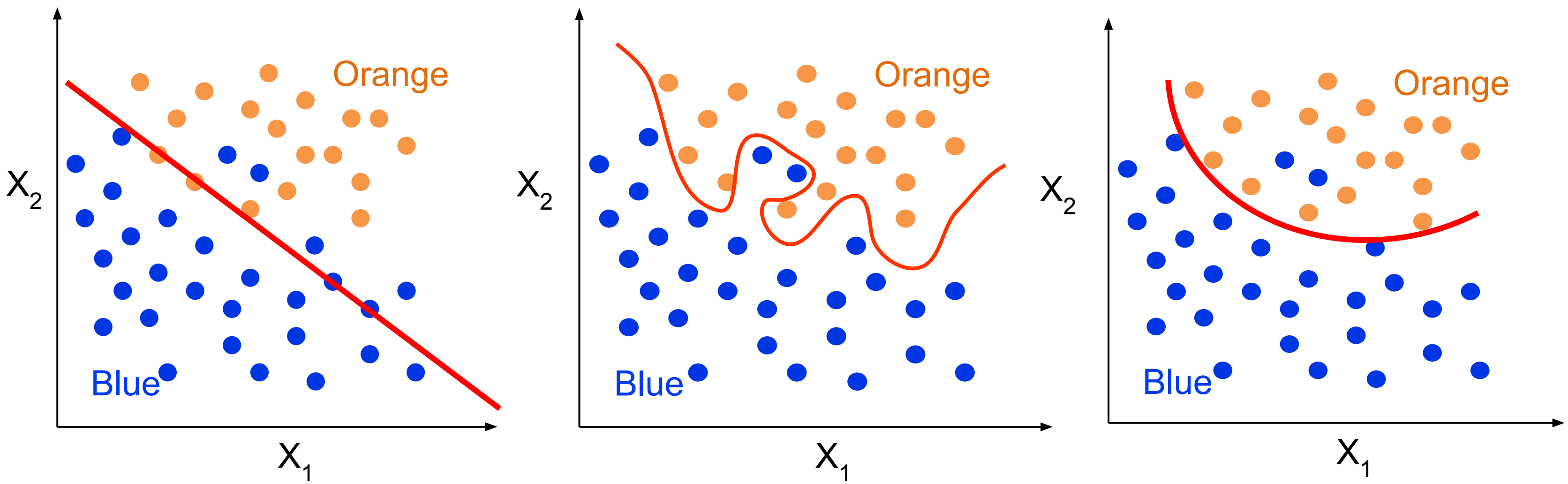
- Complex solution
- “Try too hard” to work well with training data
- Fit noise into the model
- Pattern is one-off and may not appear again in future data

Classification Example



- Good solution, able to generalize well for future data

Underfitting and Overfitting



Underfitting
(Too simple to
explain the
variance)

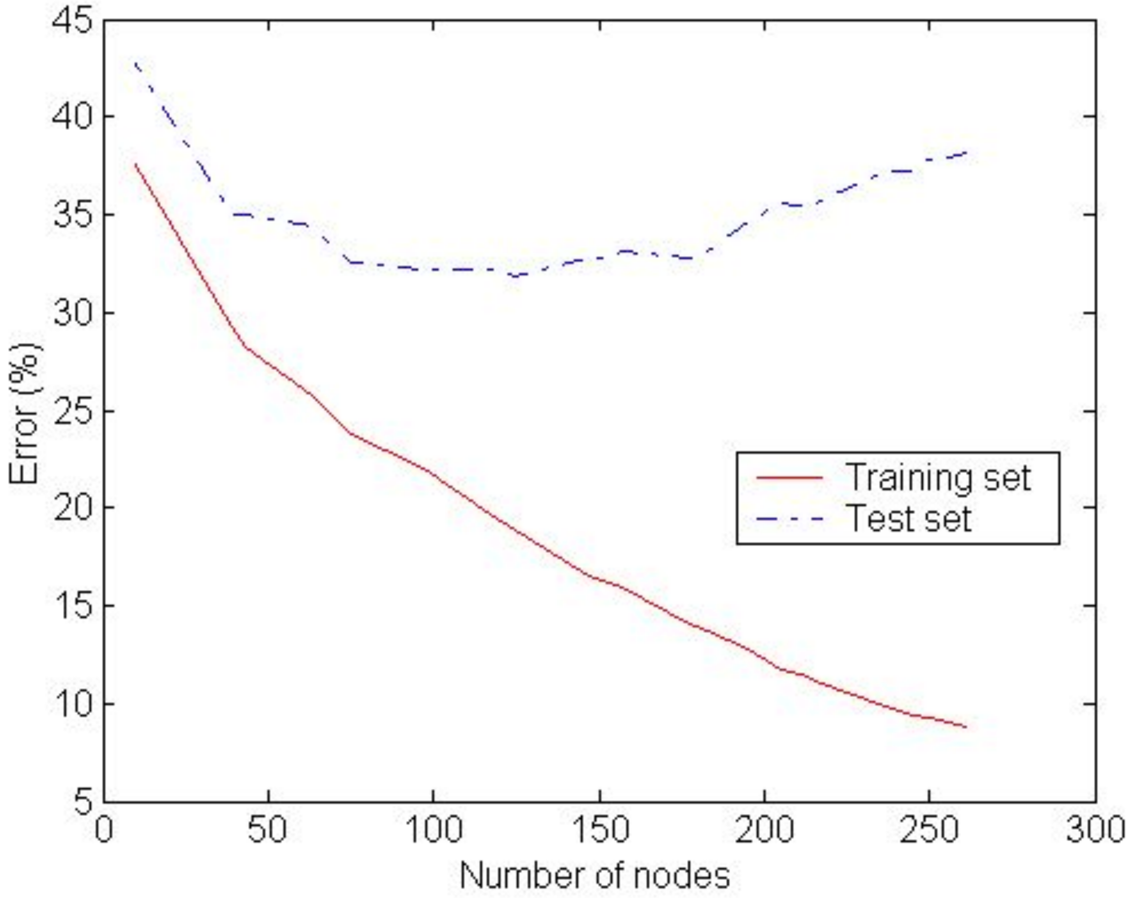
Overfitting
(Too complex
- try too hard)

Appropriate
Fitting

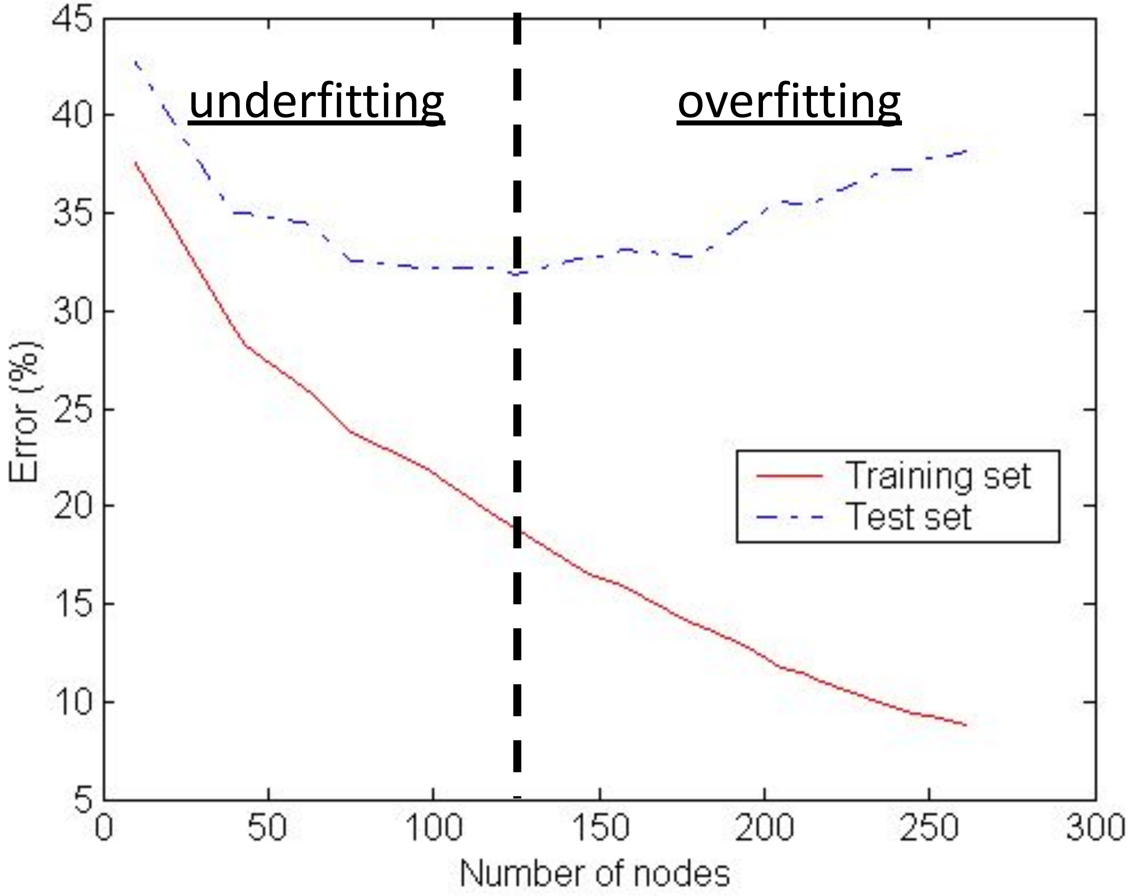
Underfitting and Overfitting

- Overfitting
  - Model is too complex (too flexible)
  - Fit noise in training data
  - Pattern is one-off and might not appear in future data
  - Model F overfits the data if:
    - We can find another model F'
    - Which makes more mistakes in training data: $E_{Train}(F') > E_{Train}(F)$
    - But less mistake in future data: $E_{Test}(F) > E_{Test}(F')$
- Underfitting
  - Model is too simple
  - Not powerful enough to capture the salient pattern
  - Can find another model F' with smaller $E_{Test}(F')$ and $E_{Train}(F')$

Underfitting and Overfitting

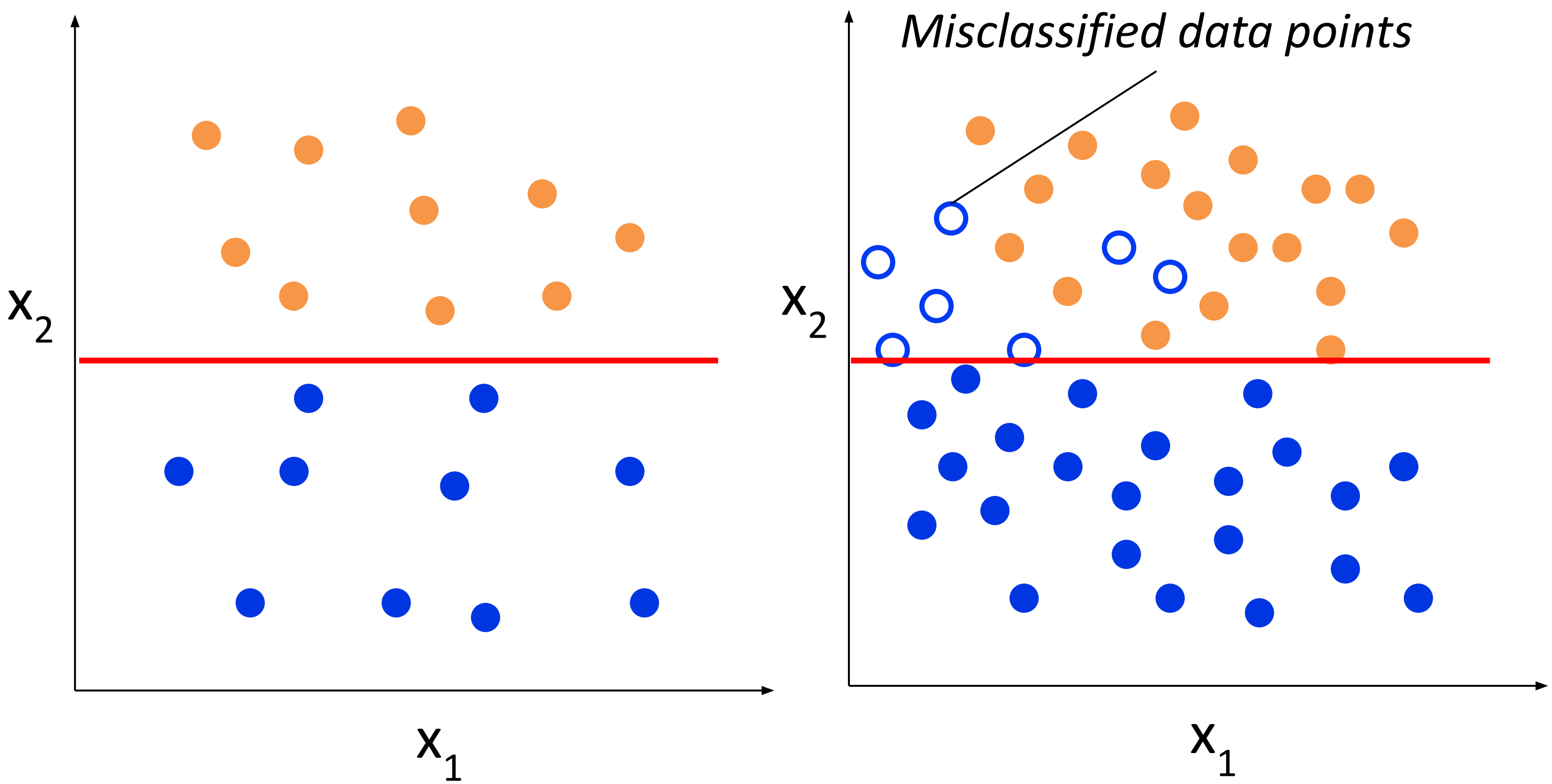


Underfitting and Overfitting



What Causes Overfitting?

- Noise in training data
- Insufficient data points in training data



Further Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary

- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records

- Need new ways for estimating errors

Estimating Generalization Errors

- Training errors: error on training (Σ e(t))
- Generalization errors: error on testing data (Σ e'(t))
- Methods for estimating generalization errors:
  - Optimistic approach: simply use training error
    - e'(t) = e(t)
  - Pessimistic approach:
    - For each leaf node: e'(t) = (e(t)+0.5)
    - Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
    - Example: For a tree with 30 leaf nodes and 10 errors on training data (out of 1000 training data instances):
      - Training error = 10/1000 = 1%
      - Generalization error = (10 + 30×0.5)/1000 = 2.5%
  - Reduced error pruning (REP):
    - uses validation data set to estimate generalization error

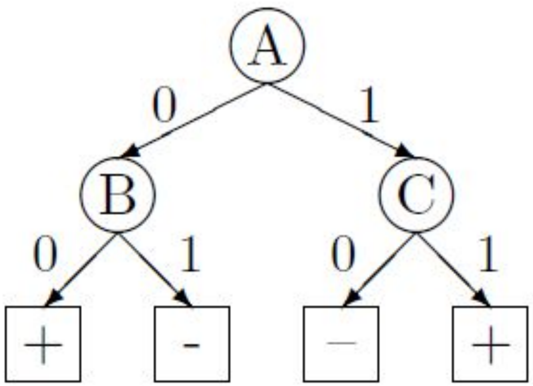
Estimating Generalization Errors

Training

S/N	A	B	C	Label
1	0	0	0	+
2	0	0	1	+
3	0	1	0	+
4	0	1	1	-
5	1	0	0	+

Testing

S/N	A	B	C	Label
6	0	0	0	+
7	0	1	1	+
8	1	1	0	+
9	1	0	1	-
10	1	0	0	+



Estimating Generalization Error

Optimistic: ?

Pessimistic: ?

Actual Generalization Error : ?

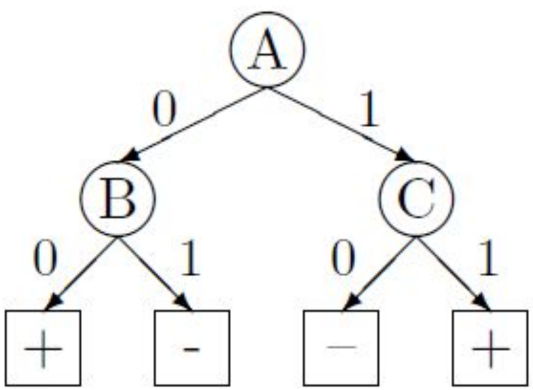
Estimating Generalization Errors

Training

S/N	A	B	C	Label	
1	0	0	0	+	✓
2	0	0	1	+	✓
3	0	1	0	+	✗
4	0	1	1	-	✓
5	1	0	0	+	✗

Testing

S/N	A	B	C	Label	
6	0	0	0	+	✓
7	0	1	1	+	✗
8	1	1	0	+	✗
9	1	0	1	-	✗
10	1	0	0	+	✗



Estimating Generalization Error

Optimistic: ?

Pessimistic: ?

Actual Generalization Error : ?

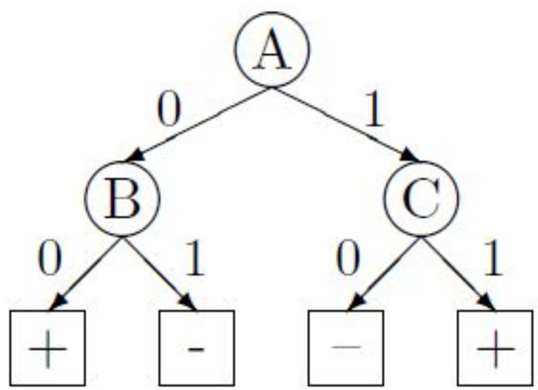
Estimating Generalization Errors

Training

S/N	A	B	C	Label	
1	0	0	0	+	✓
2	0	0	1	+	✓
3	0	1	0	+	✗
4	0	1	1	-	✓
5	1	0	0	+	✗

Testing

S/N	A	B	C	Label	
6	0	0	0	+	✓
7	0	1	1	+	✗
8	1	1	0	+	✗
9	1	0	1	-	✗
10	1	0	0	+	✗



Estimating Generalization Error

Optimistic: e'(t) = e(t) = 2 / 5 = 0.4

Pessimistic: e'(t) = (e(t)+0.5) = (2 + 4 * 0.5)/5

= 0.8

Actual Generalization Error : e'(t) = 4 / 5 = 0.8

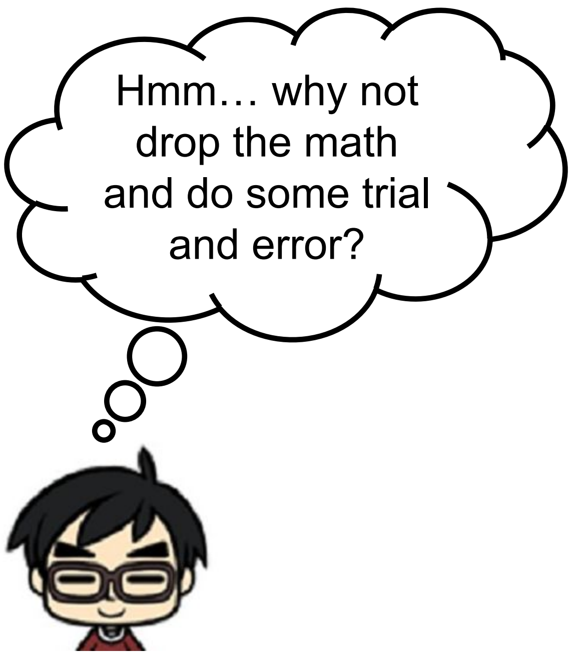
Occam's Razor

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
- For complex models, there is a greater chance that it was fitted accidentally by errors in data
- Therefore, one should include model complexity when evaluating a model

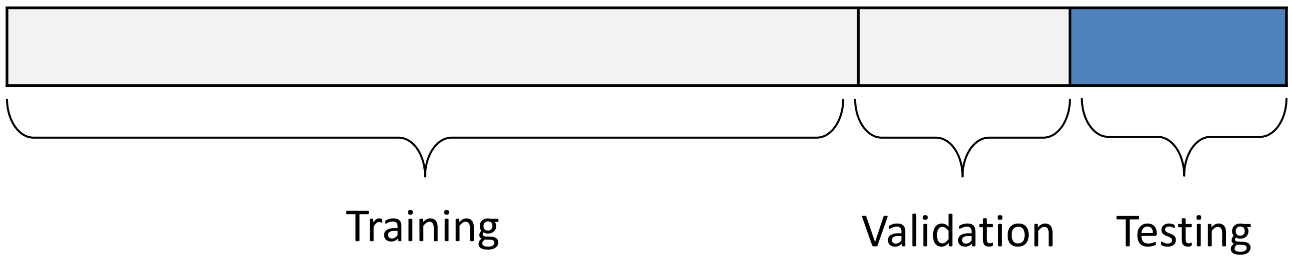
How to Address Overfitting

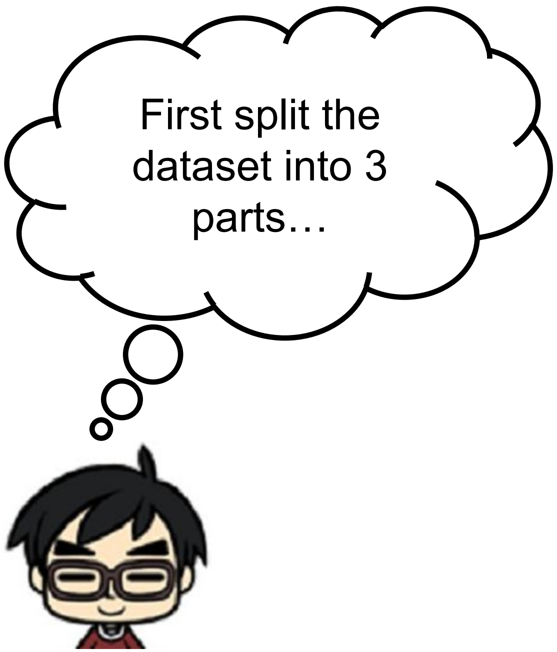
- **Pre-Pruning (Early Stopping Rule)**
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
    - More restrictive conditions: E.g., Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).
- **Post-Pruning**
  - Grow decision tree to its entirety
  - Trim the nodes of the decision tree in a bottom-up fashion
    - If generalization improves after trimming, replace sub-tree by a leaf node
  - Class label of leaf node is determined from majority class of instances in the sub-tree

How will lazy Roy do it?

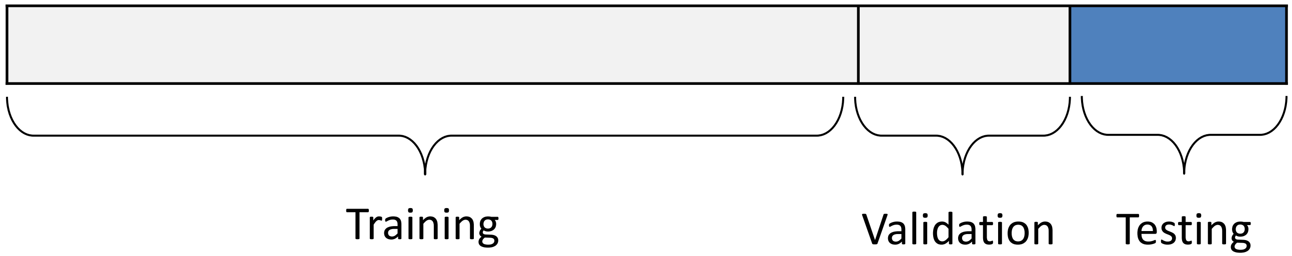


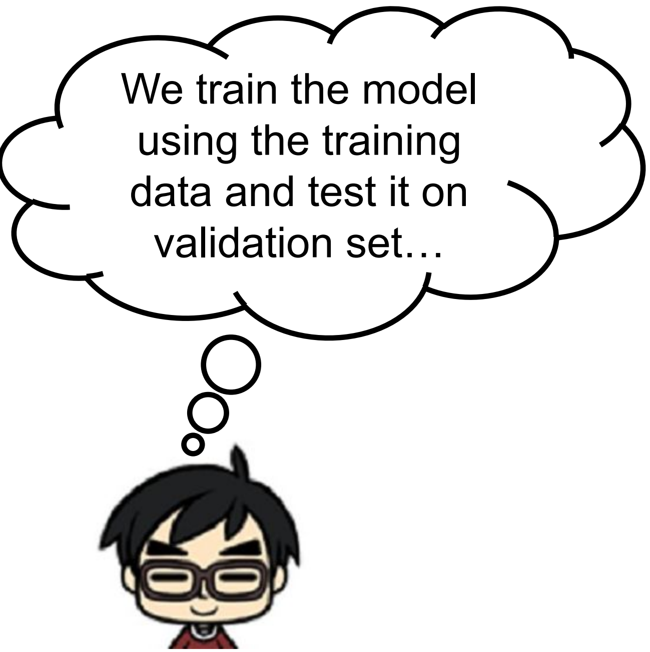
How will lazy Roy do it?





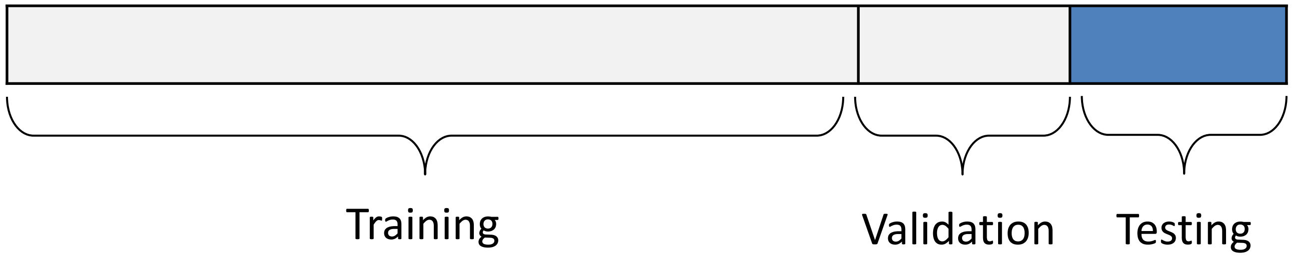
How will lazy Roy do it?

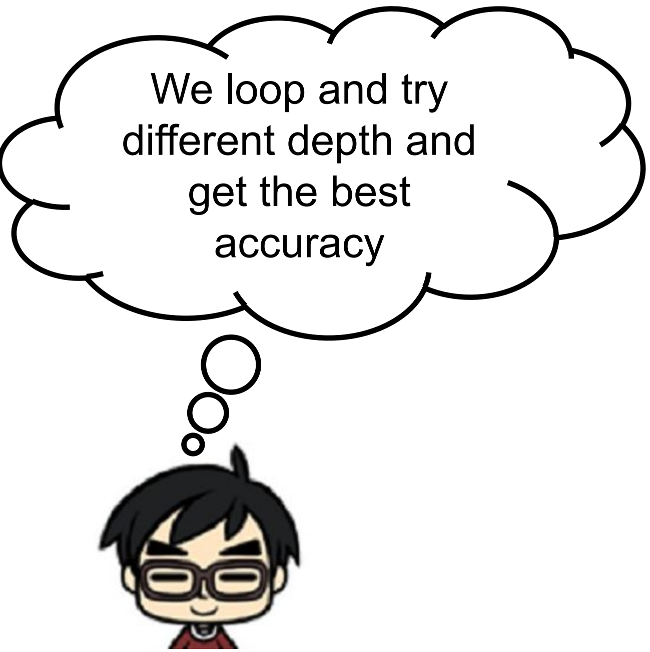





```
DT = DecisionTreeClassifier(depth=d)
DT.fit(training data)
Pred = DT.predict(validation data)
Acc = ComputeAcc(Pred)
```

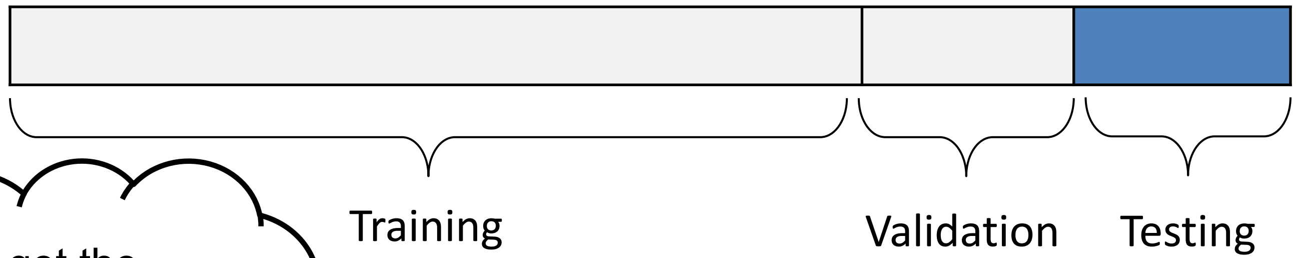
How will lazy Roy do it?

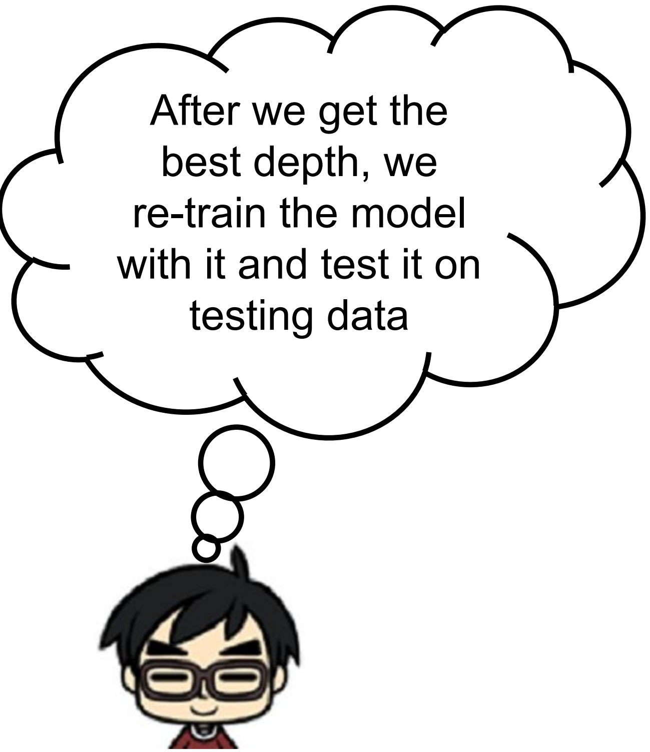





```
For d in range(101,1,-1)
    DT = DecisionTreeClassifier(depth=d)
    DT.fit(training data)
    Pred = DT.predict(validation data)
    Acc = ComputeAcc(Pred)
    if BestAcc < Acc:
        BestAcc = Acc
        BestDepth = d
```

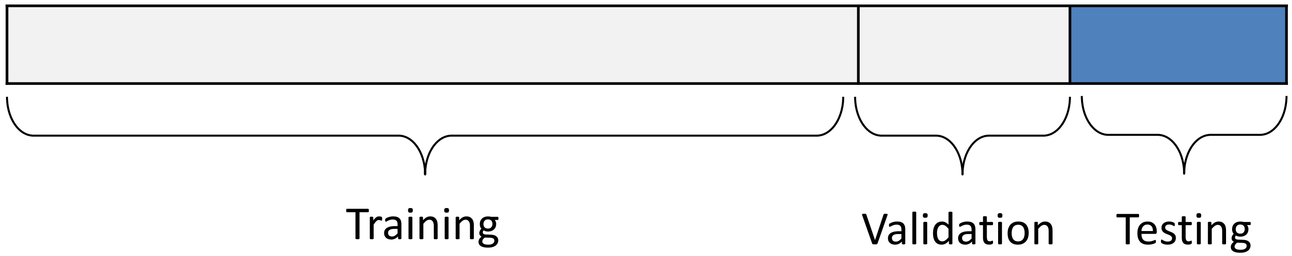
How will lazy Roy do it?

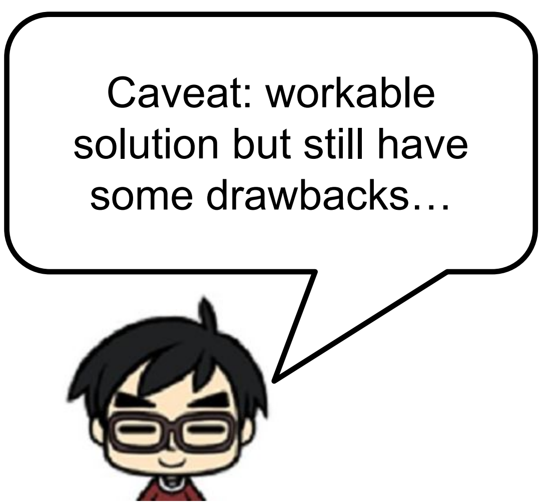





```
For d in range(101,1,-1)
    DT = DecisionTreeClassifier(depth=d)
    DT.fit(training data)
    Pred = DT.predict(validation data)
    Acc = ComputeAcc(Pred)
    if BestAcc < Acc:
        BestAcc = Acc
        BestDepth = d
DT = DecisionTreeClassifier(depth=BestDepth)
DT.fit(training data)
Pred = DT.predict(testing data)
Acc = ComputeAcc(Pred)
```

How will lazy Roy do it?



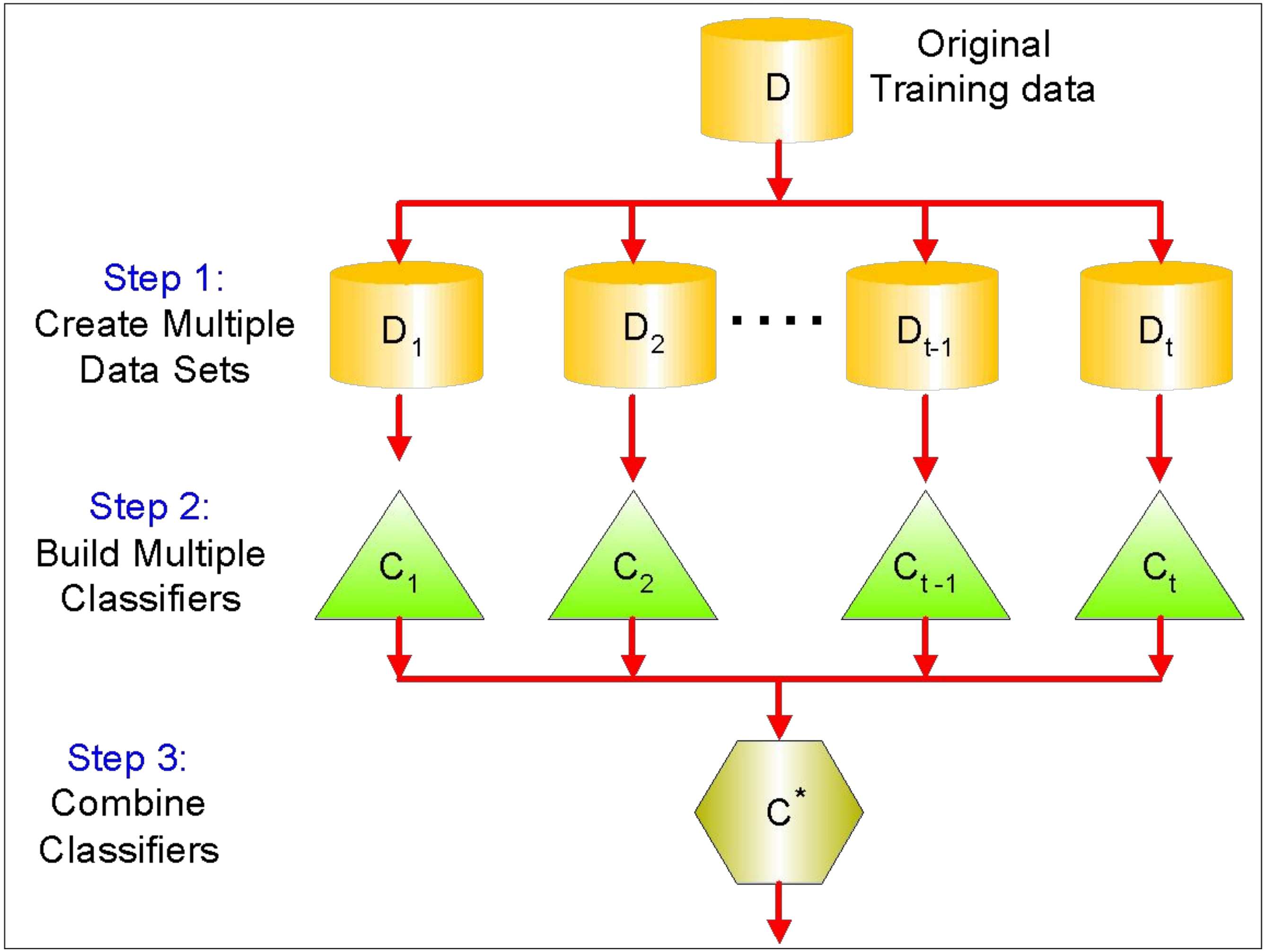



```
For d in range(101,1,-1)
   DT = DecisionTreeClassifier(depth=d)
   DT.fit(training data)
   Pred = DT.predict(validation data)
   Acc = ComputeAcc(Pred)
   if BestAcc < Acc:
      BestAcc = Acc
      BestDepth = d
DT = DecisionTreeClassifier(depth=BestDepth)
DT.fit(training data)
Pred = DT.predict(testing data)
Acc = ComputeAcc(Pred)
```

Ensemble Methods

- Construct a set of classifiers from the training data
- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers
- Assumption:
  - Individual classifiers (voters) could be lousy (stupid), but the aggregate (electorate) can usually classify (decide) correctly.

General Idea



Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, ε = 0.35
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction (i.e.,*13 out of the 25 classifiers misclassified*):

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

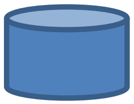
Ensemble Methods

- How to generate an ensemble of classifiers? (list below is non exhaustive!)
  - Bagging
  - Boosting

Bagging

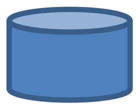
- **B**ootstrap **Agg**regating (Bagging)
- Multiple models of same learning algorithm trained with subset of dataset randomly sampled (with replacement) from the training dataset
- Each sample has probability $1-(1 - 1/n)^n$ of being selected
  - This value tends to be **0.63** for large n

Bagging



Training Dataset
(Size *N*)

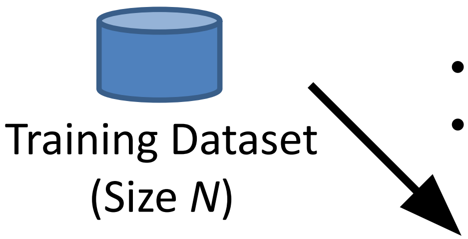
Bagging



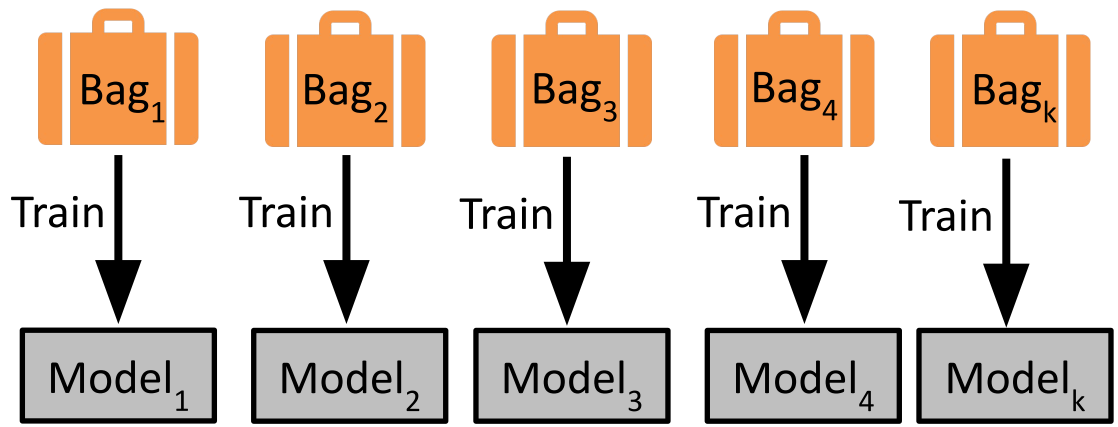
Training Dataset
(Size *N*)

- Randomly select data points into the bags (repetition allow)
- Each bag has M data points, M<N

Bagging

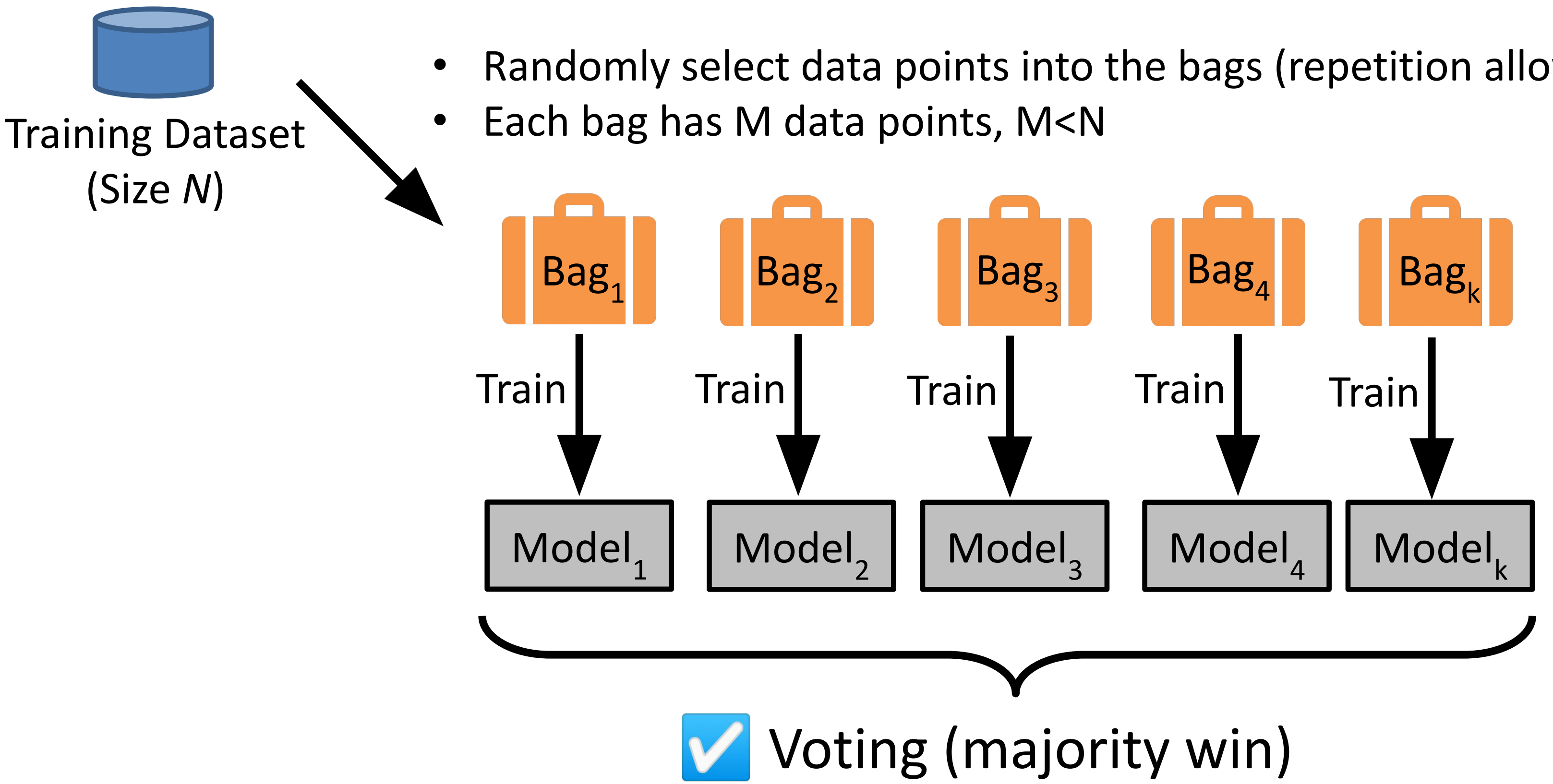


- Randomly select data points into the bags (repetition allow)
- Each bag has M data points, M<N



Bagging

- Randomly select data points into the bags (repetition allow)
- Each bag has M data points, M<N



Random Forest

- Train a lot of decision trees and use bagging
- Wisdom of the crowd!
  - Uncorrelated trees are preferred

Random Forest

- Train a lot of decision trees and use bagging
- Wisdom of the crowd!
  - Uncorrelated trees are preferred

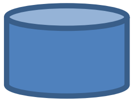
Random Forest

- Bootstrapping + Feature Randomness

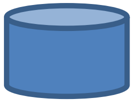
Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, sampling weights may change at the end of boosting round
    - Records that are wrongly classified will have their weights increased
    - Records that are correctly classified will have their weights decreased
  - **Caveat**: boosting show better predictive accuracy than bagging but also tends to over-fits the training data

Boosting

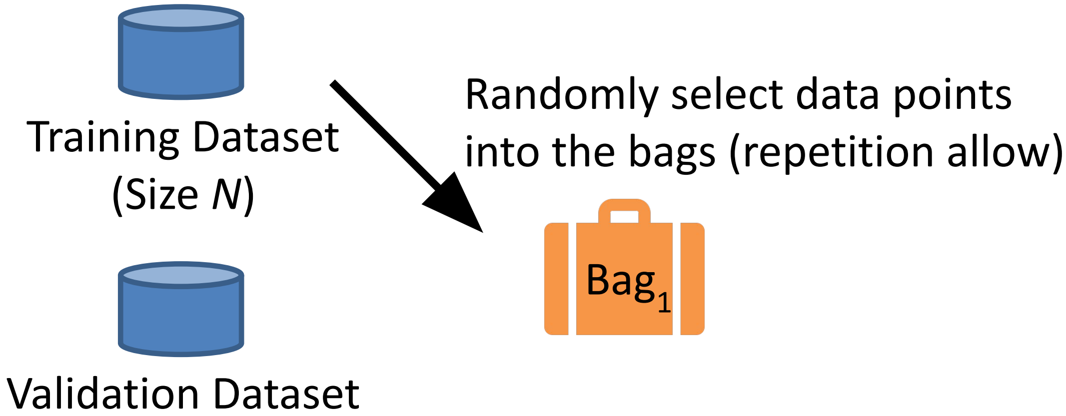


Training Dataset
(Size *N*)

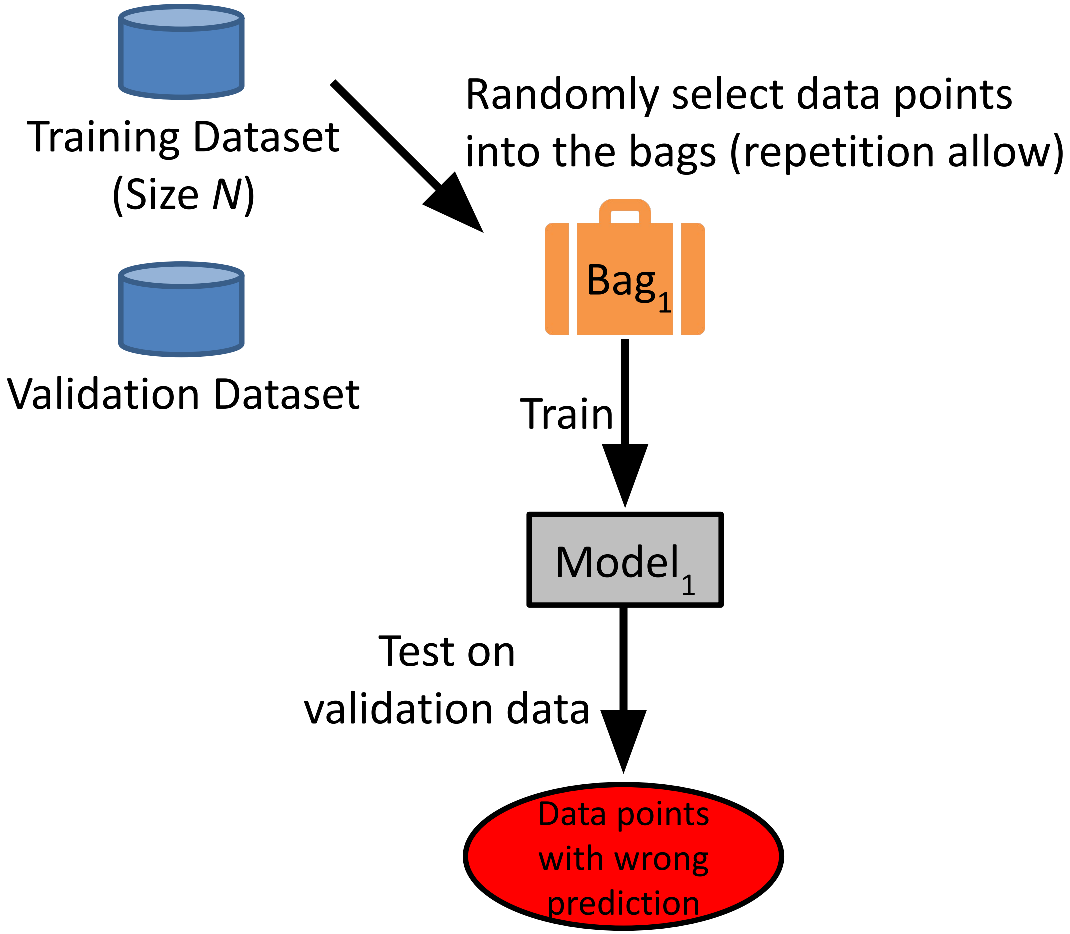


Validation Dataset

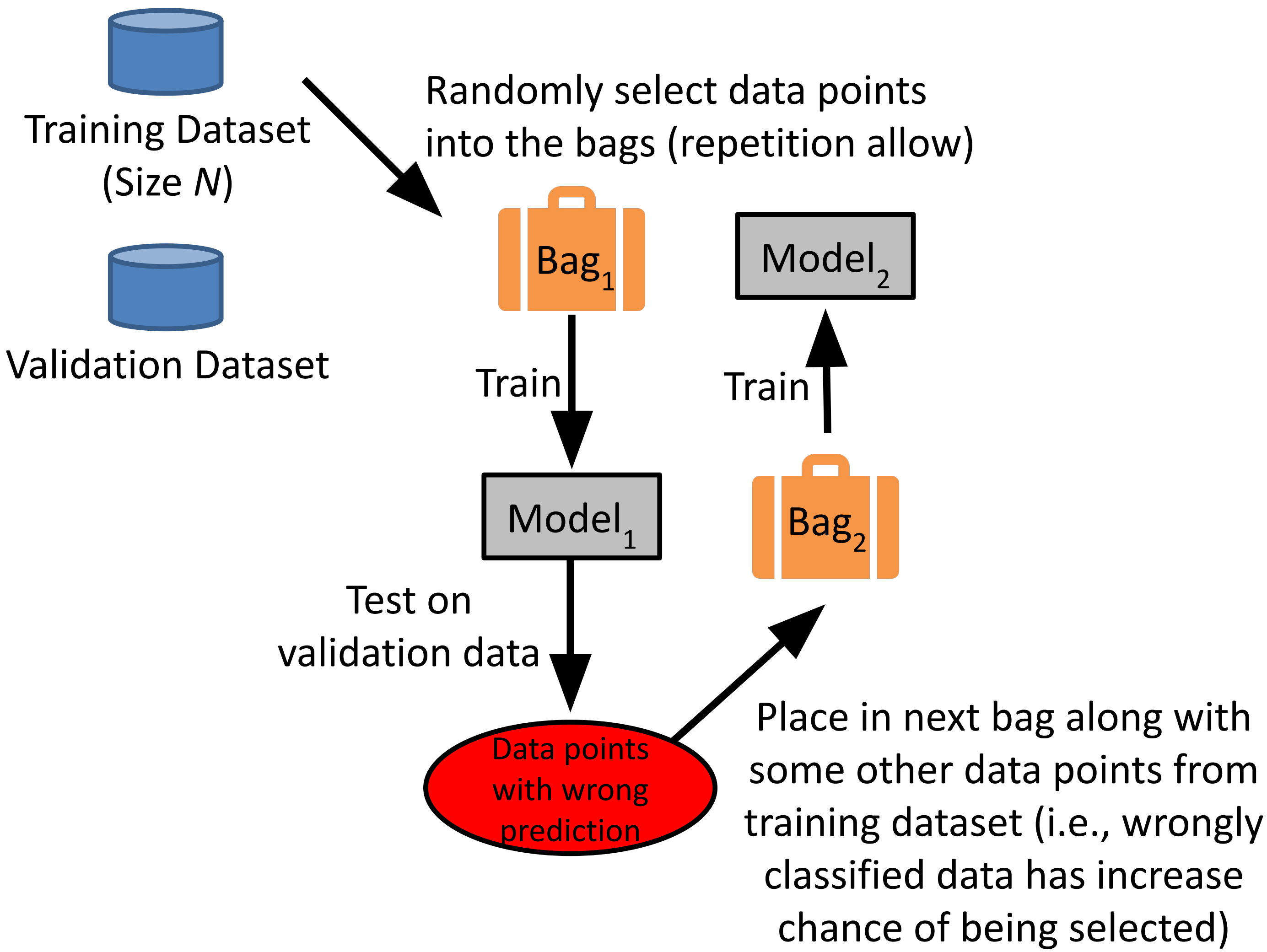
Boosting


Training Dataset
(Size *N*)
Validation Dataset
Randomly select data points
into the bags (repetition allow)
Bag1

Boosting

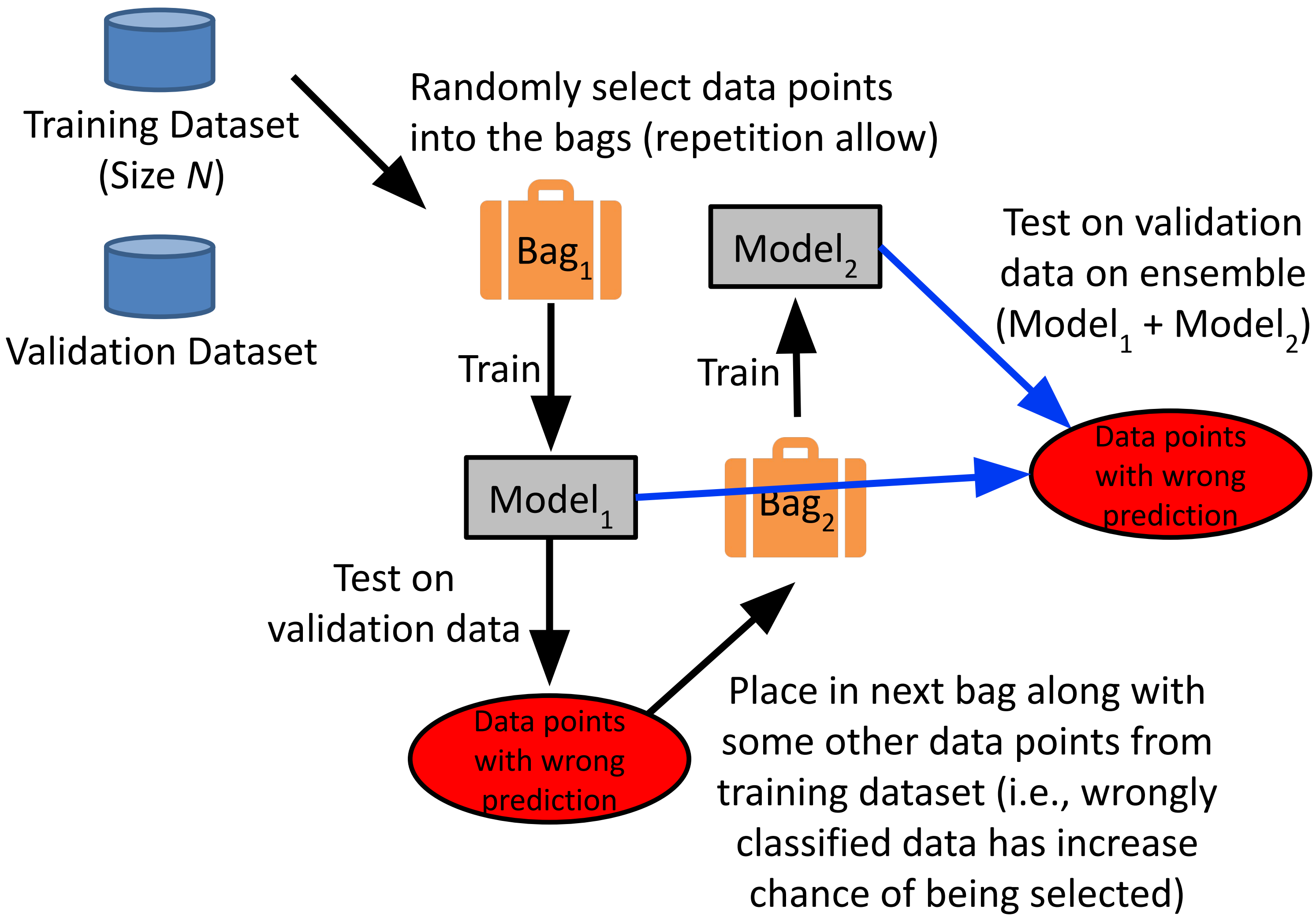

Training Dataset
(Size *N*)
Validation Dataset
Randomly select data points
into the bags (repetition allow)
Bag$_1$
Train
Model$_1$
Test on
validation data
Data points
with wrong
prediction

Boosting


Training Dataset
(Size *N*)
Validation Dataset
Randomly select data points
into the bags (repetition allow)
Bag$_1$
Model$_2$
Train
Train
Model$_1$
Bag$_2$
Test on
validation data
Data points
with wrong
prediction
Place in next bag along with
some other data points from
training dataset (i.e., wrongly
classified data has increase
chance of being selected)

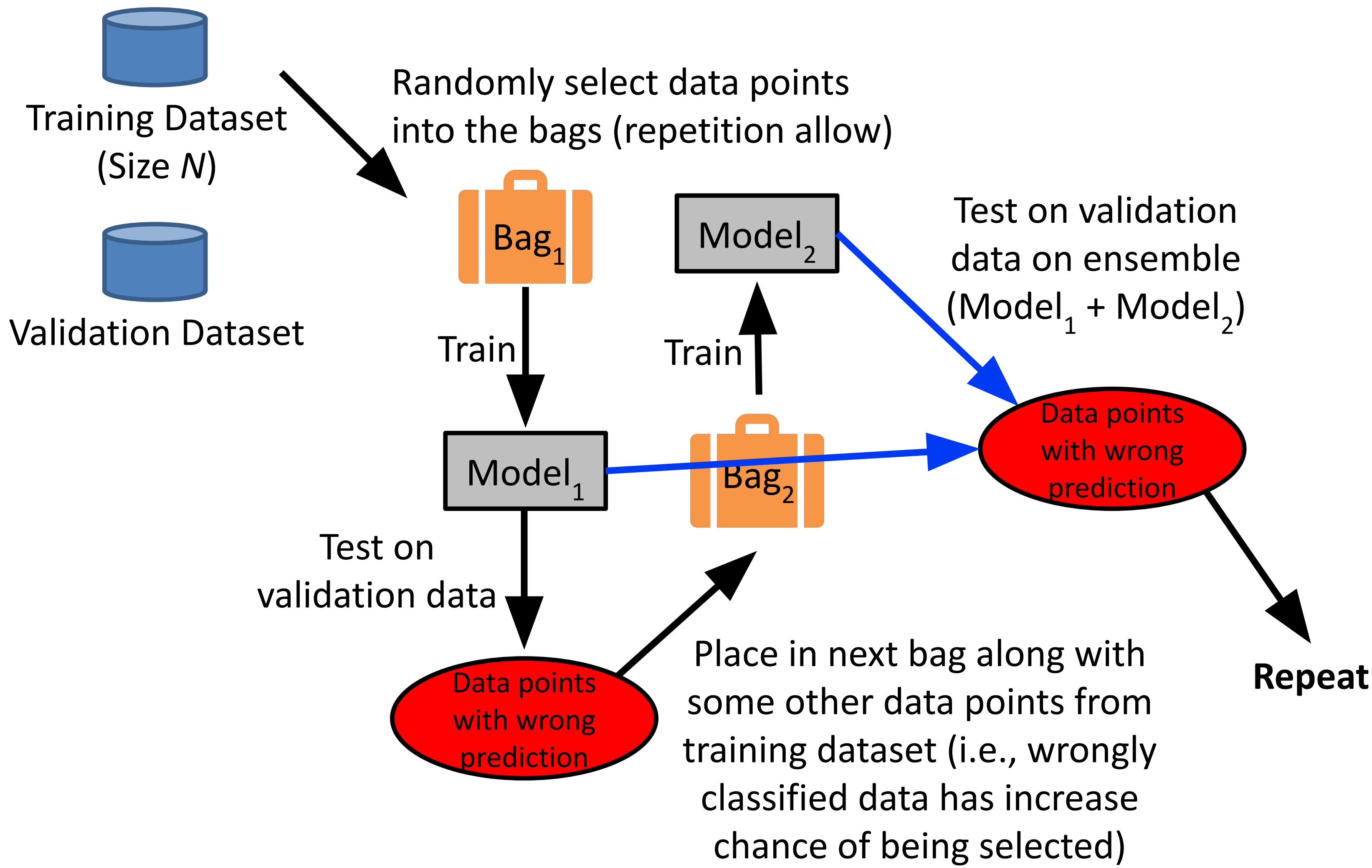
42

Boosting



Training Dataset
(Size N)
Validation Dataset
Randomly select data points
into the bags (repetition allow)
Bag1
Model2
Test on validation
data on ensemble
(Model1 + Model2)
Train
Train
Model1
Bag2
Data points
with wrong
prediction
Test on
validation data
Data points
with wrong
prediction
Place in next bag along with
some other data points from
training dataset (i.e., wrongly
classified data has increase
chance of being selected)

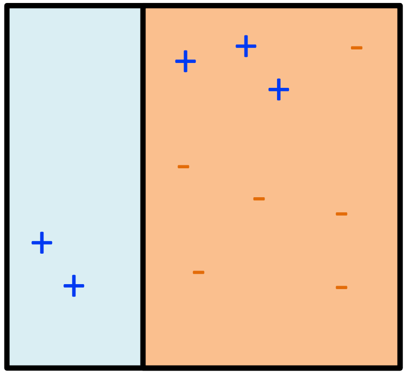
Boosting



SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

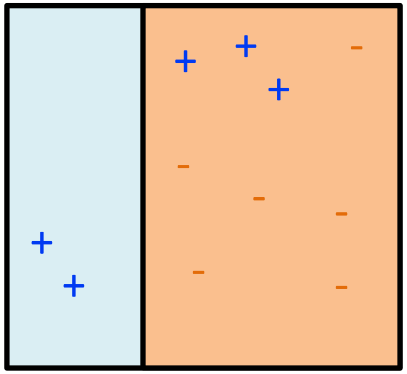
AdaBoost

Iteration 1

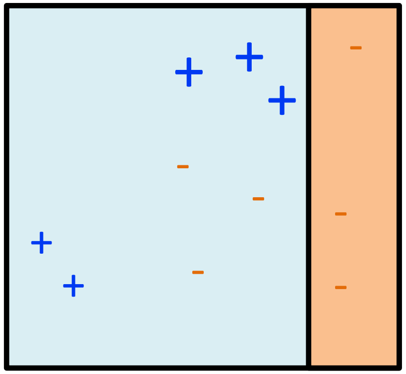


AdaBoost

Iteration 1

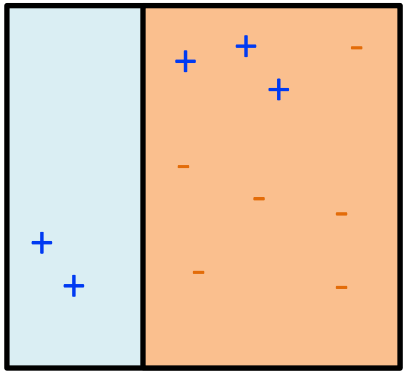


Iteration 2

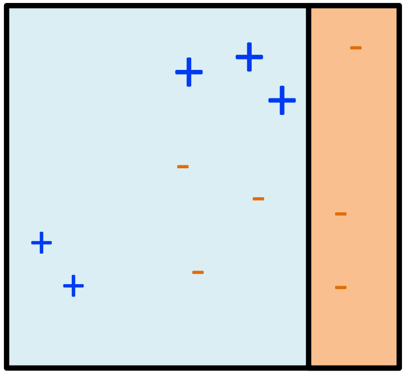


AdaBoost

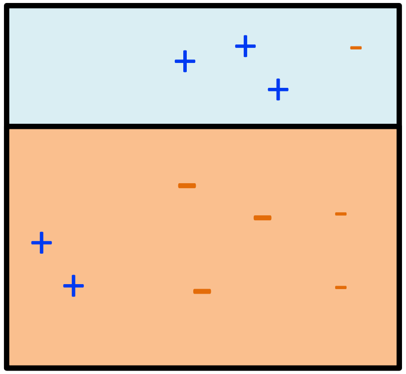
Iteration 1



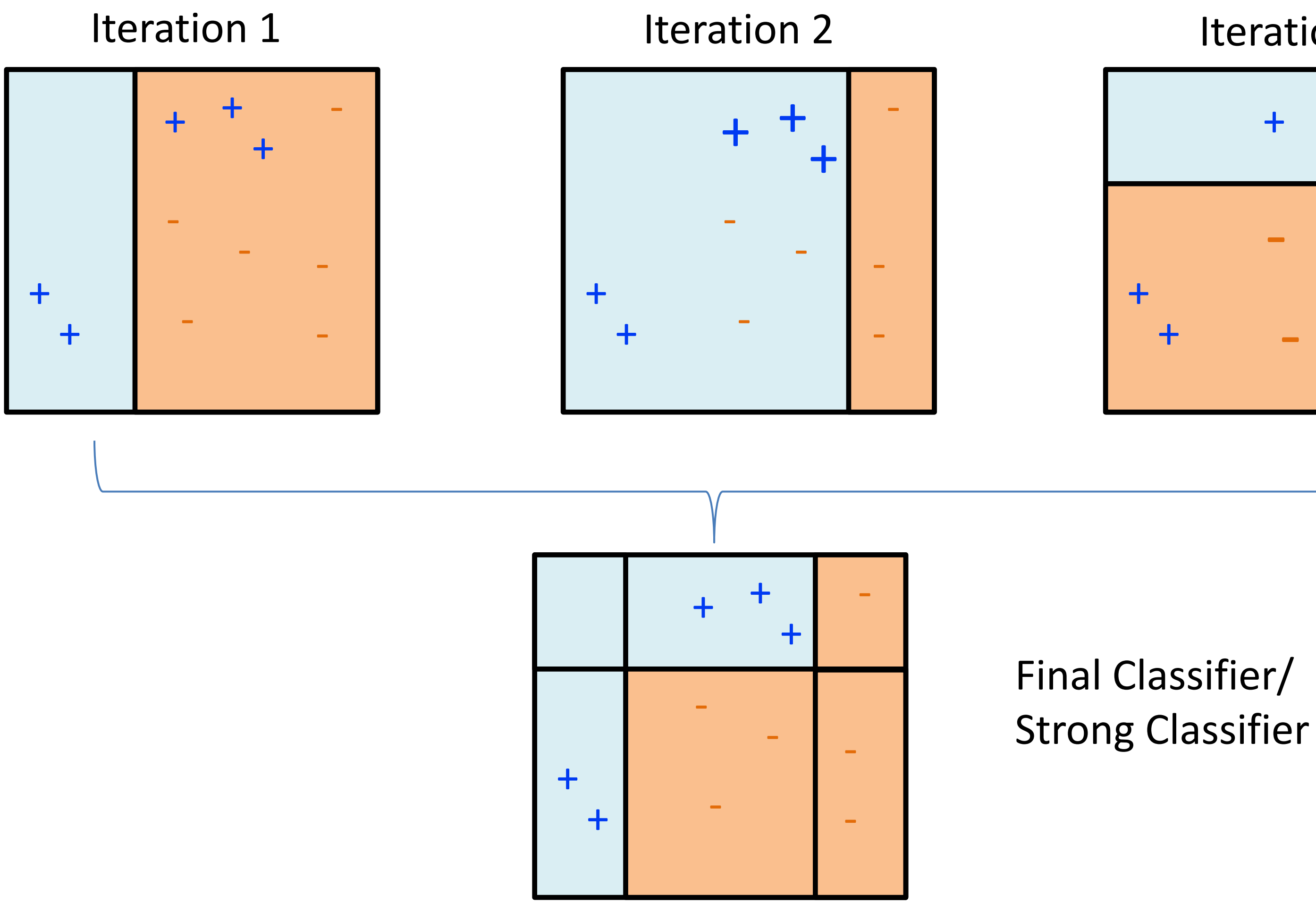
Iteration 2



Iteration 3



AdaBoost


Iteration 1
Iteration 2
Iteration 3
Final Classifier/
Strong Classifier

AdaBoost Algorithm

- Initialize observation weights $w(x_i, y_i) = 1/n,\ i=1,\ldots,n$
- Base classifiers: $C_1, C_2, \ldots, C_T$

AdaBoost Algorithm

- Initialize observation weights $w(x_i,y_i) = 1/n,\ i=1,\ldots,n$
- Base classifiers: $C_1, C_2, \ldots, C_T$
- For $i=1$ to T:
  - Fit a classifier $C_i(x)$ to training data
  - Compute error rate of C_i:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

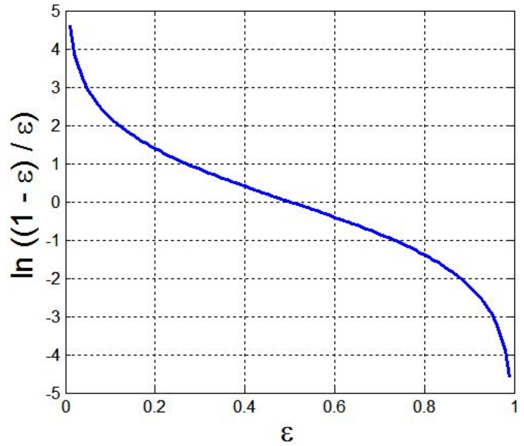
AdaBoost Algorithm

- Initialize observation weights $w(x_i, y_i) = 1/n,\ i=1,\ldots,n$
- Base classifiers: $C_1, C_2, \ldots, C_T$
- For $i=1$ to T:
  - Fit a classifier $C_i(x)$ to training data
  - Compute error rate of C_i:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

  - Compute importance of classifier C_i:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



When the error rate goes beyond 0.5, we assign the classifier a negative weight (i.e., we don't want them!)

AdaBoost Algorithm

- Initialize observation weights $w(x_i, y_i) = 1/n,\ i=1,\ldots,n$
- Base classifiers: $C_1, C_2, \ldots, C_T$
- For $i=1$ to T:
  - Fit a classifier $C_i(x)$ to training data
  - Compute error rate of C_i:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

  - Compute importance of classifier C_i:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

  - Update the weights

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j}\begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

- $\exp^{-\alpha}$ is always lesser than 1 when the data point is correctly classified, thus giving it a smaller weight

- $\exp^{\alpha}$ is bigger than 1 when it is misclassified, assigning this data point a greater weight

AdaBoost Algorithm

- Initialize observation weights $w(x_i, y_i) = 1/n,\ i=1,\ldots,n$
- Base classifiers: $C_1, C_2, \ldots, C_T$
- For $i=1$ to T:
  - Fit a classifier $C_i(x)$ to training data
  - Compute error rate of C_i:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

  - Compute importance of classifier C_i :

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

  - Update the weights

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j}\begin{cases}\exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i\end{cases}$$

where Z_j is the normalization factor

- Final Classifier:

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?

- Methods for Performance Evaluation
  - How to obtain reliable estimates?

- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix:

	PREDICTED CLASS	
ACTUAL CLASS	**Class=Yes**	**Class=No**
Class=Yes	a	b
Class=No	c	d

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Metrics for Performance Evaluation

	PREDICTED CLASS	
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Limitation of Accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10
- If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any class 1 example

Cost Matrix

- *C(i|j)*: Cost of misclassifying class *j* example as class *i*

	PREDICTED CLASS	
	$C(i\|j)$	**Class=Yes**	**Class=No**
ACTUAL CLASS	**Class=Yes**	C(Yes|Yes) TP	C(No|Yes) FN
	Class=No	C(Yes|No) FP	C(No|No) TN

TP: True Positive

TN: True Negative

FN: False Negative

FP: False Positive

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	+	-
	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	150	40
	-	60	250

Accuracy = ?

Cost = ?

Model M_2	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	250	45
	-	5	200

Accuracy = ?

Cost = ?

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	+	-
	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	150	40
	-	60	250

Accuracy = 150 + 250/150+250+40+60

= 80%

Cost = 150(-1) + 40(100) + 60(1) + 250(0)

= 3910

Model M_2	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	250	45
	-	5	200

Accuracy = 250 + 200/250+200+45+5

= 90%

Cost = 250(-1) + 45(100) + 5(1) + 200(0)

= 4255

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Precision, Recall & F1

	PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

$$\text{Precision (p)} = \frac{a}{a+c} \qquad \frac{TP}{TP + FP}$$

$$\text{Recall (r)} = \frac{a}{a+b} \qquad \frac{TP}{TP + FN}$$

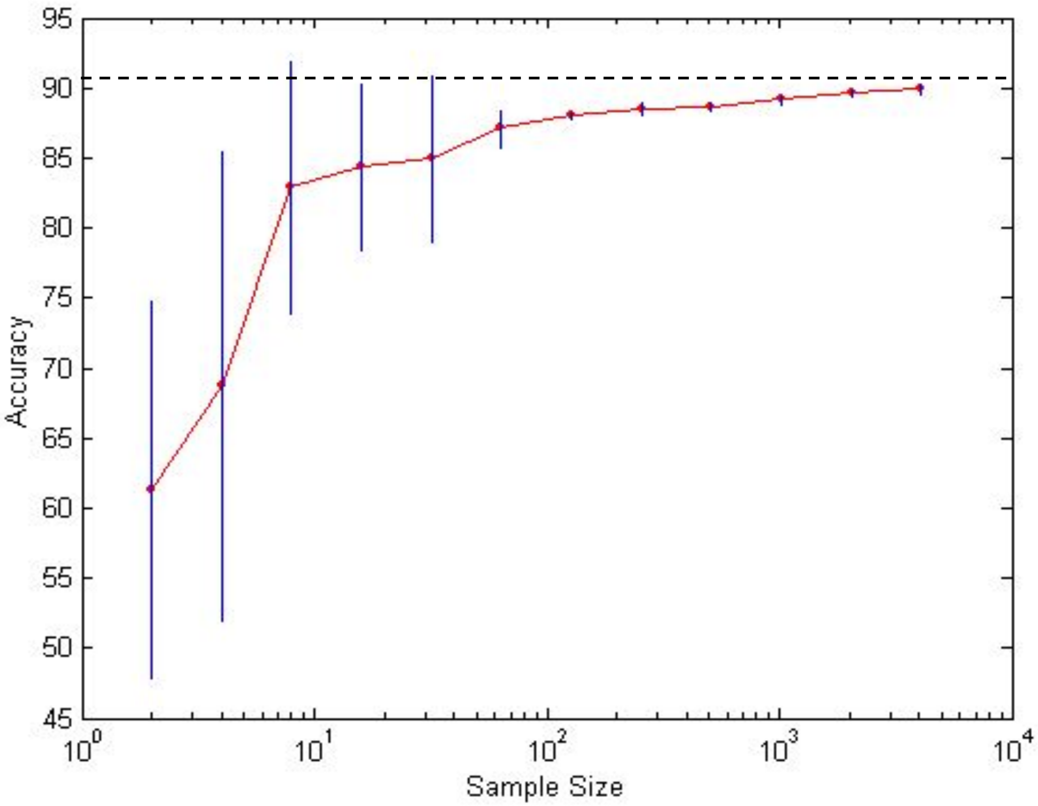
$$\text{F - measure (F)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

Methods for Performance Evaluation

- How to obtain a reliable estimate of performance?

- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets

Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Requires a sampling schedule for creating learning curve:
  - Arithmetic sampling (Langley, et al.)
  - Geometric sampling (Provost et al,)

- Effect of small sample size:
  - Bias in the estimate
  - Variance of estimate

Methods of Estimation

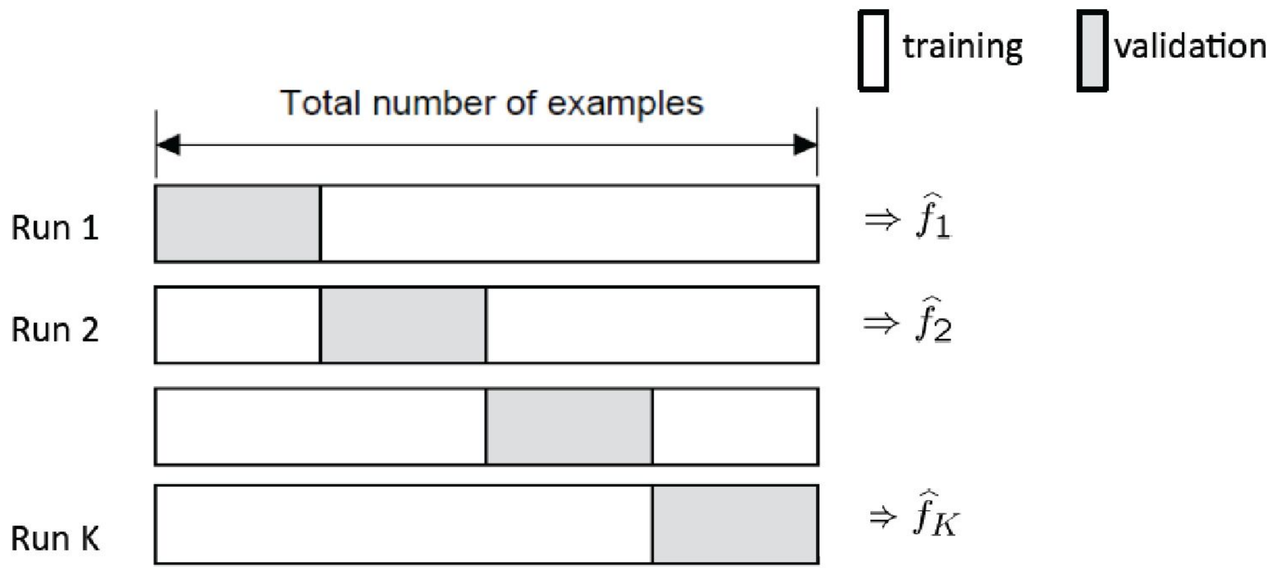
- Holdout
  - Reserve 2/3 for training and 1/3 for test
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n
- Random subsampling
  - Repeated holdout

Train, Test and Validate

- **Training Set**: The actual dataset that we use to train the model (weights and biases in the case of Neural Network). The model sees and learns from this data.
- **Validation Set**: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.
- **Testing Set**: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

Cross Validation

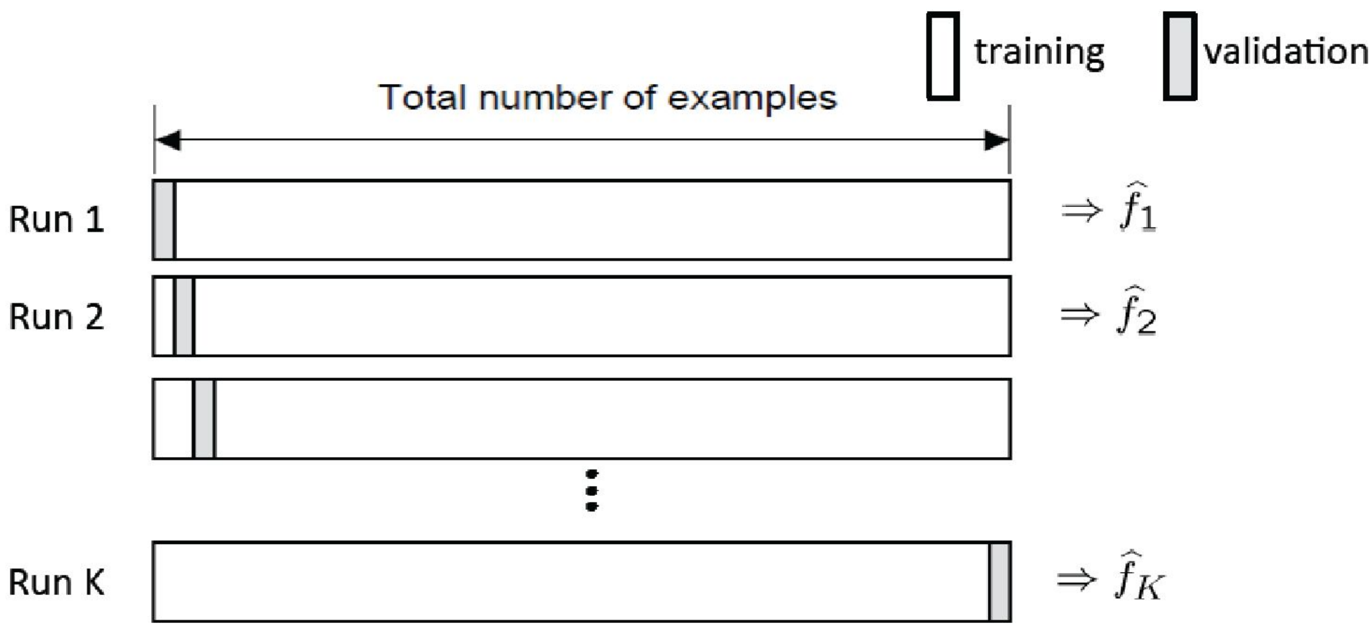
- K-fold cross-validation
  - Create K-fold partition of the dataset.
  - Form K hold-out predictors, each time using one partition as validation and rest K - 1 as training datasets.
  - Final predictor is average/majority vote over the K hold-out estimates.



SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

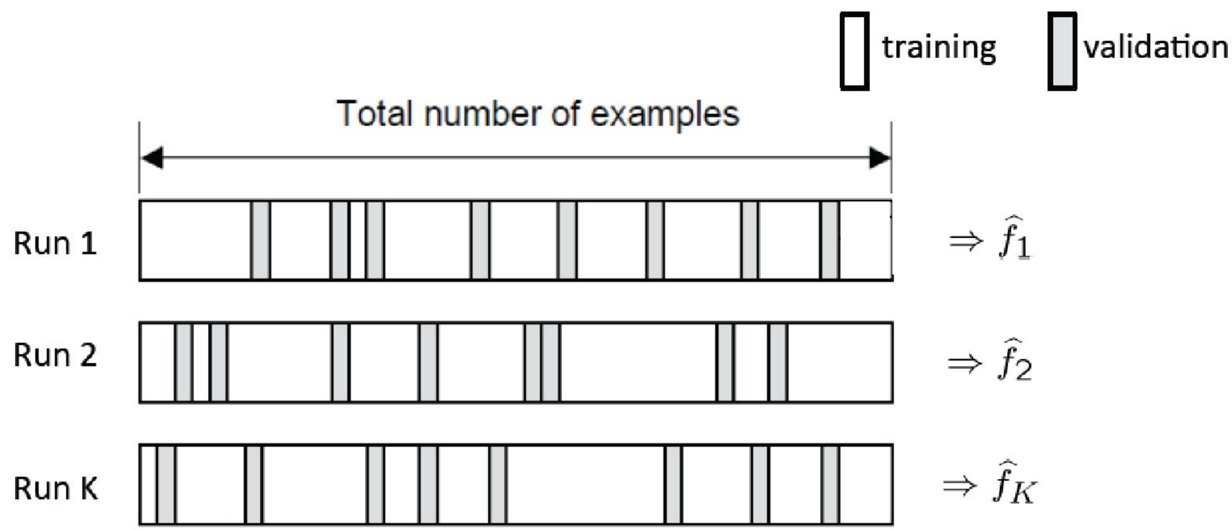
Cross Validation

- Leave-one-out (LOO) cross-validation
  - Special case of K-fold with K=n partitions
  - Equivalently, train on n-1 samples and validate on only one sample per run for n runs



Random Subsampling

- Randomly subsample a fixed fraction αn ($0 < \alpha < 1$) of the dataset for validation.
- Train the model with remaining data as training data.
- Repeat K times
- Compute the average metrics for the K hold-out estimates.



Summary

- Generalization
  - Underfitting and Overfitting
- Ensemble Classifiers
  - Bagging
    - Random Forest
  - Boosting
- Model Evaluation