Supervised Learning

Evaluation

Evaluation Issues

- What measures should we use?
  - Classification accuracy might not be enough
- How reliable are the predicted values?
- Are errors on the training data a good indicator of performance on future data?
  - If the classifier is computed from the very same training data, any estimate based on that data will be optimistic
  - New data probably not exactly the same as the training data

Road Map

1. Evaluation Metrics
2. Evaluation Methods
3. How To Improve Classifiers Accuracy

Confusion Matrix

- The confusion matrix is a table of at least m×m size. An entry **$CM_{i,j}$** indicated the number of tuples of class **i** that were **labeled as** class **j**

Real class \ Predicted class	**$Class_1$**	**$Class_2$**	**…**	**$Class_m$**
$Class_1$	$CM_{1,1}$	CM_{12}	…	$CM_{1,m}$
$Class_2$	$CM_{2,1}$	$CM_{2,2}$	…	$CM_{2,m}$
…	…	…	…	…
$Class_m$	$CM_{m,1}$	$CM_{m,2}$	…	$CM_{m,m}$

- Ideally, most of the tuples would be represented along the diagonal of the confusion matrix

Confusion Matrix

- For a targeted class

		Predicted Class	
		Yes	**No**
Target Class	**Yes**	True positives	False negatives
	No	False positives	True negatives

- **True positives**: positive tuples correctly labeled
- **True negatives**: negative tuples correctly labeled
- **False positives**: negative tuples incorrectly labeled
- **False negatives**: positive tuples incorrectly labeled

Accuracy

		Predicted Class	
		Yes	**No**
Target Class	**Yes**	True positives (TP)	False negatives (FN)
	No	False positives (FP)	True negatives (TN)

- Most widely used metric

$$Accuracy = \frac{\#\ correctly\ classified\ tuples}{total\ \#\ tuples}$$

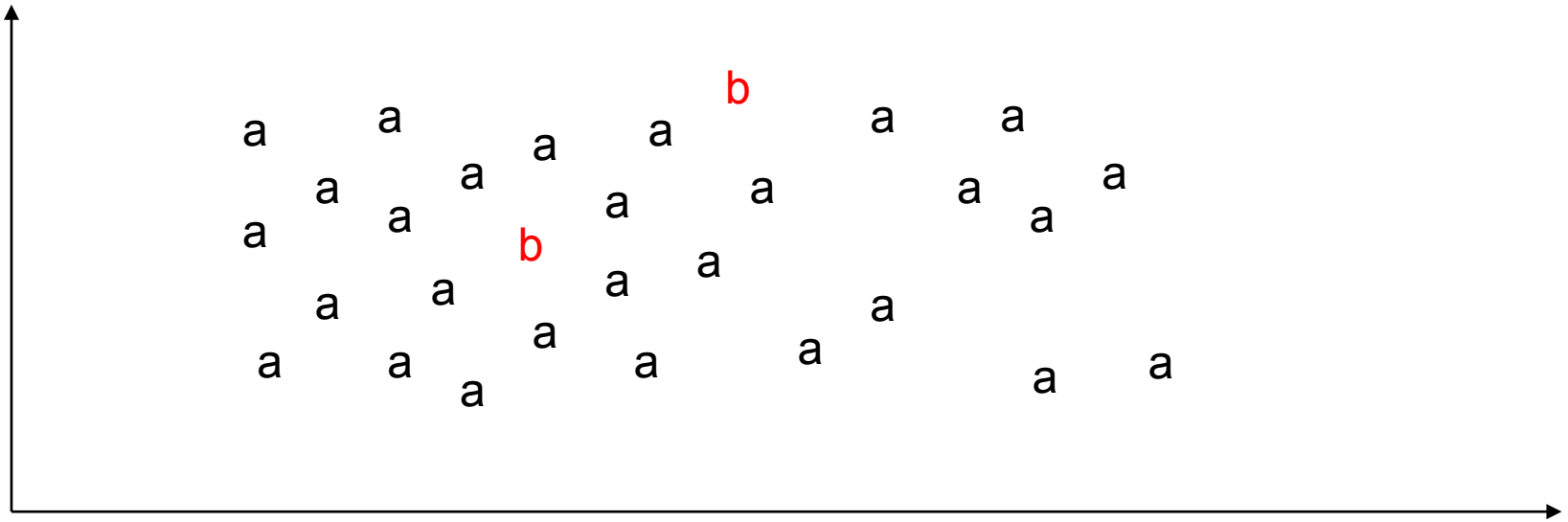
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy

- Accuracy is better measured using test data that was not used to build the classifier
- Referred to as the **overall recognition rate of the classifier**
- Error rate or misclassification rate: **1-Accuracy**
- When training data are used to compute the accuracy, the error rate is called **resubstituion error**

Sometimes Accuracy is not Enough

- Consider a 2-class problem



- If a model predicts always **class a**, accuracy is 28/30= **93%**
- Accuracy is misleading because the model does not detect any tuple of **class b**

Cost Matrix

- Useful when specific classification errors are more severe than others

		Predicted Class	
		Yes	**No**
Target Class	**Yes**	C(Yes | Yes)	C(No | Yes)
	No	C(Yes | No)	C(No | No)

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

Computing the Cost of Classification

Cost Matrix

Cost Matrix		Predicted Class	
		Yes	No
Costumer Satisfied	Yes	0	1
	No	120	0

Confusion Matrix

Model M1		Predicted Class	
		Yes	No
Costumer Satisfied	Yes	150	60
	No	40	250

Accuracy= 80%

Cost= 4860

Confusion Matrix

Model M2		Predicted Class	
		Yes	No
Costumer Satisfied	Yes	250	5
	No	45	200

Accuracy= 90%

Cost= 5405

Cost-Sensitive Measures

		Predicted Class	
		Yes	**No**
Target Class	**Yes**	True positives (TP)	False negatives (FN)
	No	False positives (FP)	True negatives (TN)

$$Precision(p) = \frac{TP}{TP + FP}$$

Biased towards C(Yes|Yes) & C(Yes|No)

The higher the precision, the lower the FPs

$$Recall(r) = \frac{TP}{TP + FN}$$

Biased towards C(Yes|Yes) & C(No|Yes)

The higher the recall, the lower the FNs

$$F1\text{-}measure = \frac{2rp}{r + p}$$

Biased towards all except C(No|No)

It is high when both p and r are high

The higher the F1, the lower the FPs & FNs

$$WeightedAccuracy = \frac{w_{TP} \times TP + w_{TN} \times TN}{w_{TP} \times TP + w_{FN} \times FN + w_{FP} \times FP + w_{TN} \times TN}$$

Other Specific Measures

- Other measures can be used when the accuracy measure is not acceptable

$$Sensitivity(sn) = \frac{TP}{\#\,Positives}$$

$$Specificity(sp) = \frac{TN}{\#\,Negatives}$$

$$Accuracy = sn\frac{\#\,Positives}{(\#\,Positives + \#\,Negatives)} + sp\frac{\#\,Negatives}{(\#\,Positives + \#\,Negatives)}$$

Predictor Error Measures

- The predictor returns continuous values
  - Measure how far the predicted value from the known value
- Compute **loss functions**

$$Absolute\ error = | y_i - y'_i |$$

$$Squared\ error = (y_i - y'_i)^2$$

y_i : the true value
y'_i: the predicted value

- The **test error** or **generalization error** is the average loss

$$Mean\ absolute\ error = \frac{\sum_{i=1}^{N} | y_i - y'_i |}{N}$$

$$Mean\ squared\ error = \frac{\sum_{i=1}^{N} (y_i - y'_i)^2}{N}$$

N is the size of the test dataset

Predictor Error Measures

- The total loss can be normalized: divid by the total loss incurred from always predicting the mean

$$Relative\ absolute\ error = \frac{\sum_{i=1}^{N} | y_i - y'_i |}{\sum_{i=1}^{N} | y_i - \bar{y} |}$$

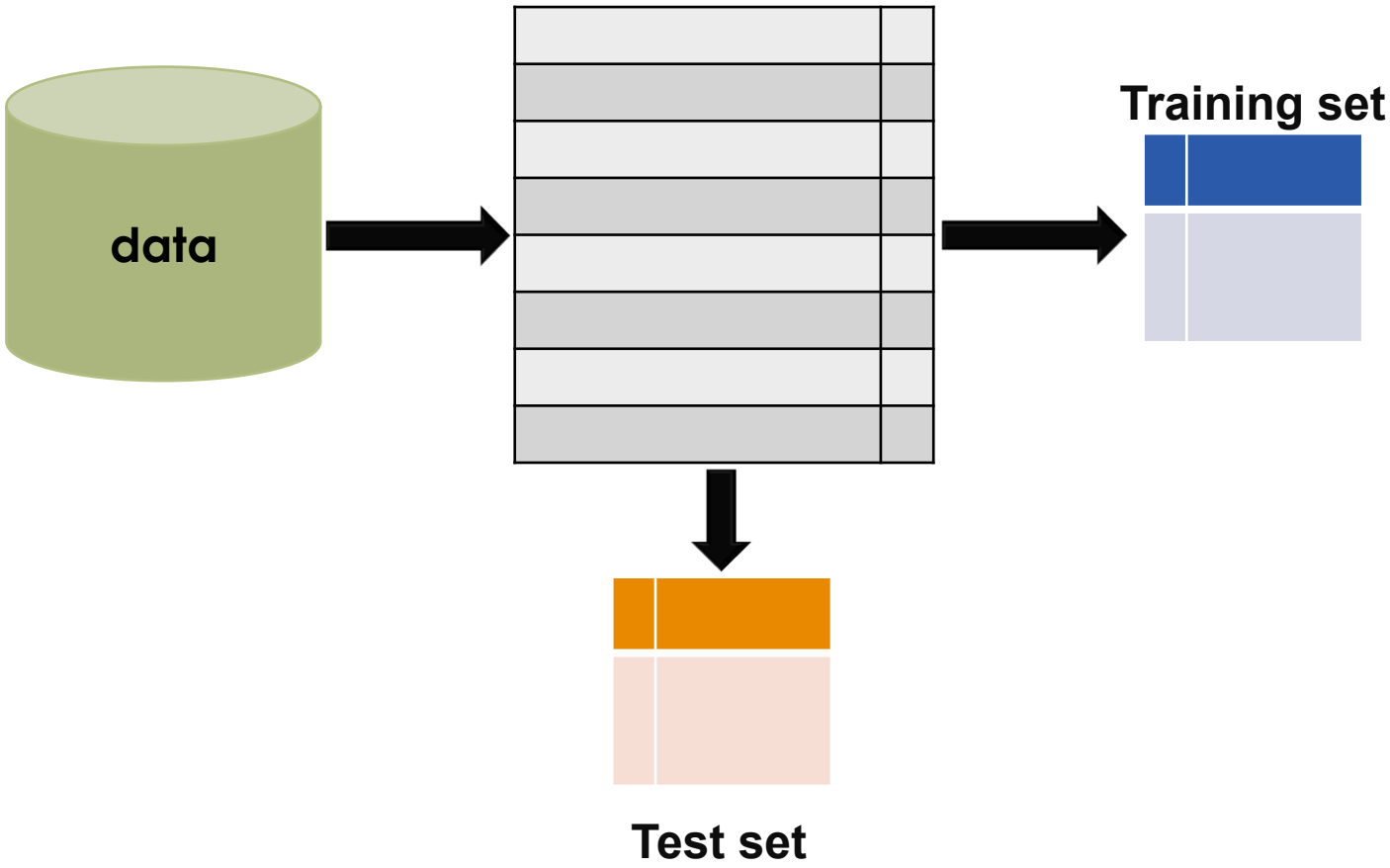
$$Relative\ squared\ error = \frac{\sum_{i=1}^{N} (y_i - y'_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}$$

N is the size of the test dataset

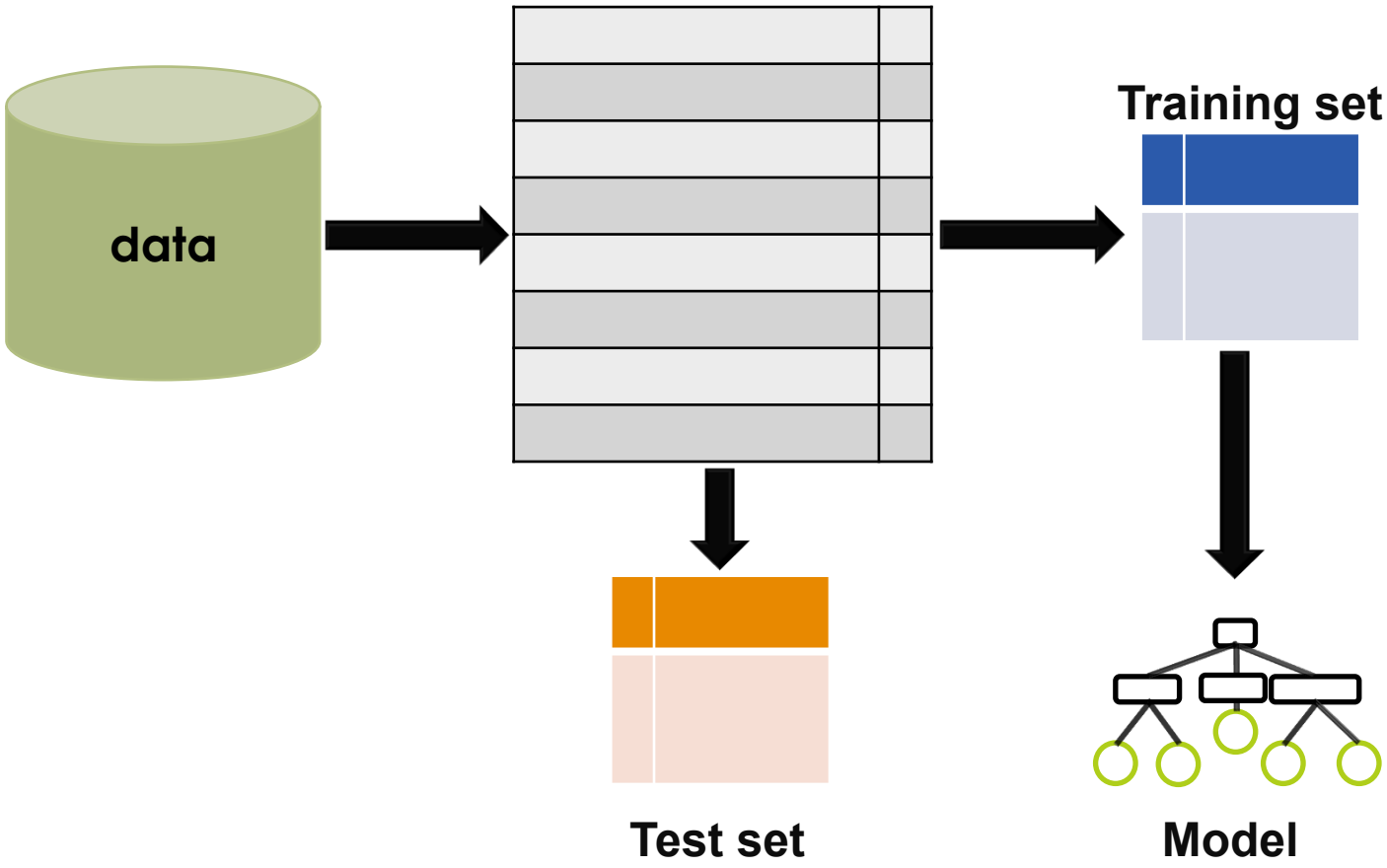
Road Map

1. Evaluation Metrics
2. Evaluation Methods
3. How to Improve Classifiers Accuracy

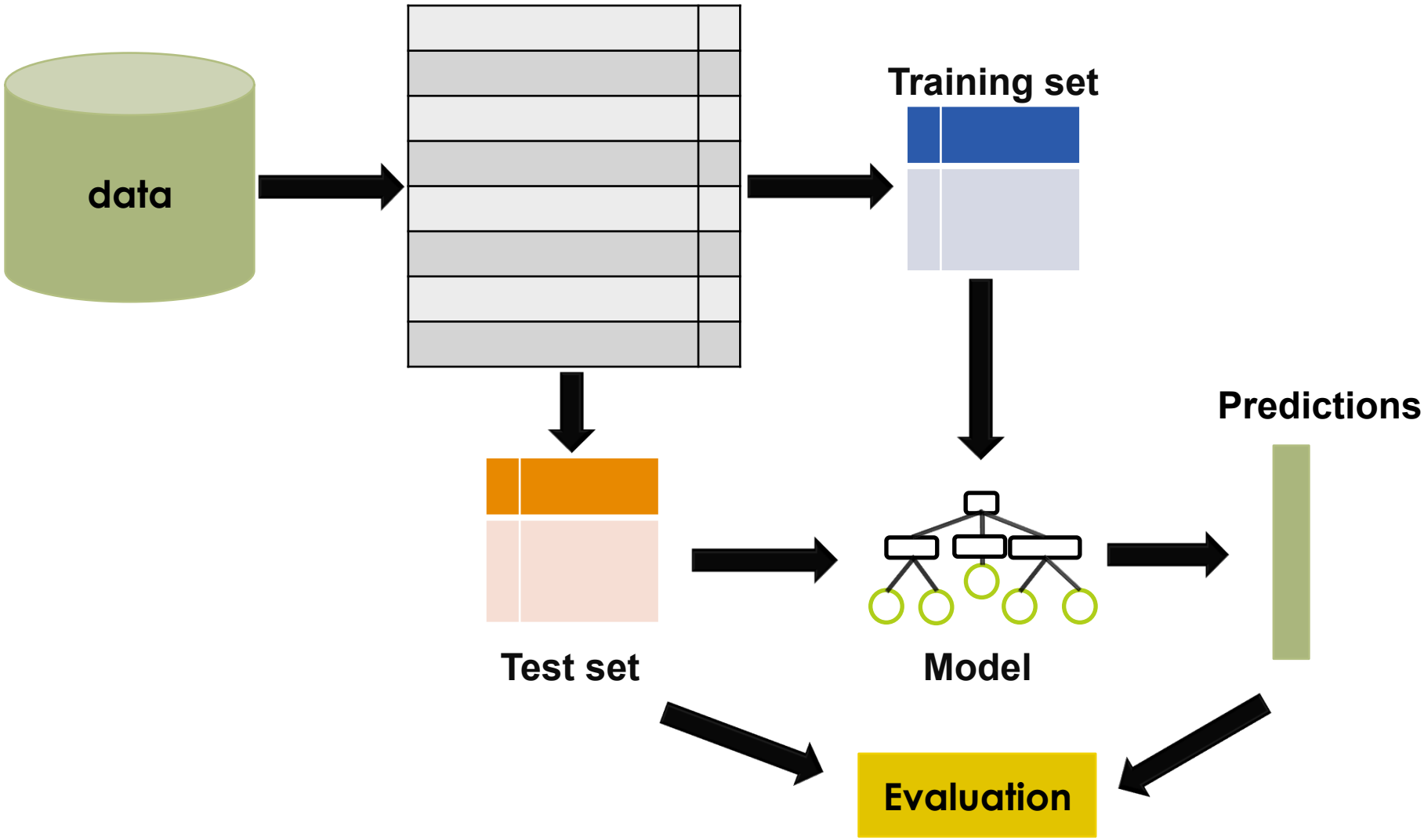
Step1: Prepare Training and Test Sets



Step2: Build The Model

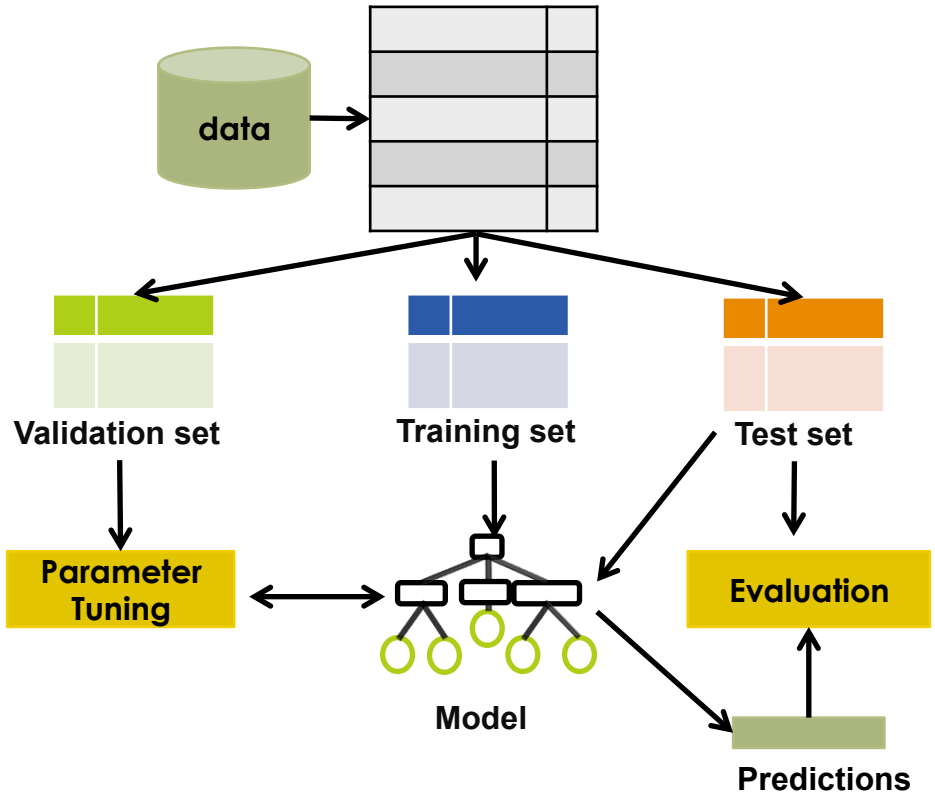


Step3: Evaluate The Model



Note on Parameter Tuning

- Some learning schemes operates in two stages:
  - Stage1: Build the basic structure of the model
  - Sage2: Optimize parameter settings
- It is important not to use test data to build the model
- The test data should not be used for parameter tuning



Methods for Performance Evaluation

- How to obtain 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

Holdout

- Typically use two-thirds of the data are allocated to training set and one-third is allocated to test set
- The estimate is pessimistic because only a portion of the initial data is used to derive the model
- For small or "unbalanced" datasets, samples might not be representative
- **Stratified sampling**: make sure that each class is represented with approximately equal proportions in both subsets

Random Subsampling

- Holdout estimate can be made more reliable by repeating the process with different subsamples (k times)
- In each iteration, a certain proportion is randomly selected for training (possibly with stratification)
- The error rates on the different iterations are averaged to yield an overall error rate
- Still not optimum since the different test sets overlap

Cross Validation

- Avoids overlapping test sets
  - Data is split into k mutually exclusive subset, or **folds**, of equal size: D_1, D_2,…, D_k
  - Each subset in turn is used for testing and the remainder for training:

    First iteration: use D_2,…D_k and training and D_1 as test
    Second iteration: use D_1,D_3,…,D_k as training and D_2 as test
    …
- This is called k-fold cross validation
- Often the subsets are stratified before cross-validation is performed
- The error estimates are averaged to yield an overall error estimate

Cross Validation

- Standard method for evaluation stratified 10-fold cross validation
- Why 10? Extensive experiments have shown that this is the best choice to get an accurate estimate
- Stratification reduces the estimate's variance
- Even better: repeated cross-validation. E.g. ten-fold cross-validation is repeated ten times and results are averaged (reduces the variance)

Leave-One-Out Cross Validation

- A special case of k-fold cross-validation
  - K is set to the initial number of tuples
  - Only one sample is left out at a time for the test set
- Makes best use of the data
- Involves no random subsampling
- Computationally expensive
- Disadvantage: stratification is not possible
- Extreme example: random dataset split equally into two classes
  - Model predicts majority class
  - 50% accuracy on the whole data
  - Leave-One-Out-CV estimate is 100% error

The 0.632 Bootstrap

- Sample training tuples uniformly with replacement: each time a tuple is selected, it is equally to be selected again and re-added to the training set
- An instance has a probability of 1-1/n of not being picked
- Thus its probability of ending up in the test data is:

$$\left(1-\frac{1}{n}\right)^{n} \approx e^{-1} = 0.368$$

- This means the training data will contain approximately 63.2% of the instances

Estimating Error using Bootstrap

- The error estimate on the test data will be very pessimistic, since training was on just ~63% of the instances
- Therefore, combine it with the resubstitution error:

$$Err(M) = \sum_{i=1}^{k} (0.632 \times Err(M_i)_{test\_set} + 0.368 \times Err(M_i)_{train\_set})$$

- The resubstitution error gets less weight than the error on the test data
- Repeat process several times with different replacement samples; average the results

More on Bootstrap

- Probably the best way of estimating performance for very small datasets
- However, it has some problems
- Consider a random dataset with a **50%** class distribution
  - A model that memorizes all the training data will achieve **0%** resubstituion error on the training data and **~50%** error on test data
  - Bootstrap estimate for this classifier:
  - Err=0.632×0.5+0.368×0=**31.6%**
  - True expected error: **50%**

Road Map

1. Evaluation Metrics
2. Evaluation Methods
3. How to Improve Classifiers Accuracy

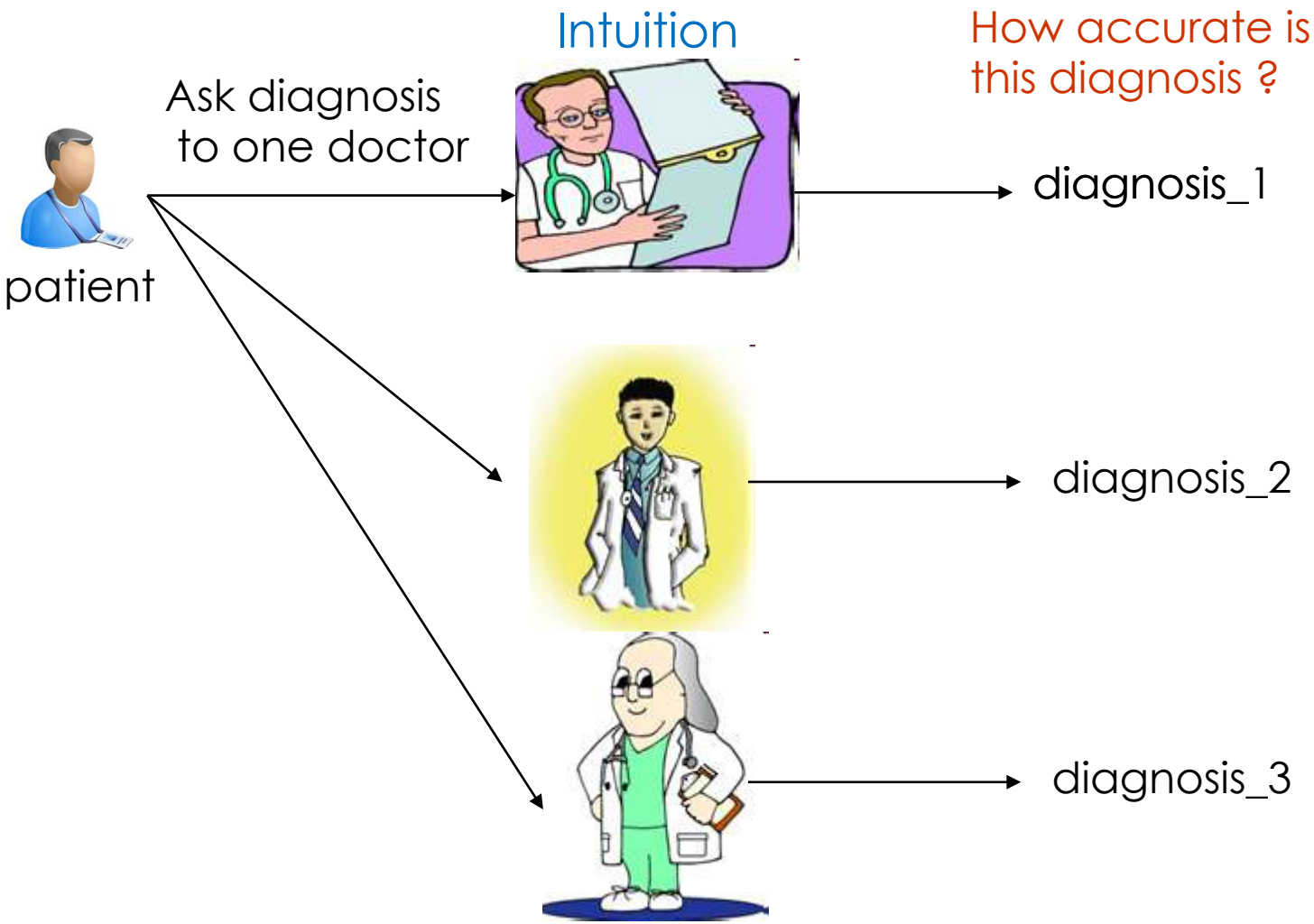
Increasing the Accuracy

- We have seen that pruning improves the accuracy of decision trees by reducing the overfitting effect

- There are some general strategies for improving the accuracy of classifiers and predictors

- Bagging and Boosting are some of these strategies
  - Ensemble methods: use a combination of models
  - Find an an improved composite model M*
  - Combine a series of learned classifiers $M_1, M_2, \ldots, M_k$
  - Find an an improved composite model M*

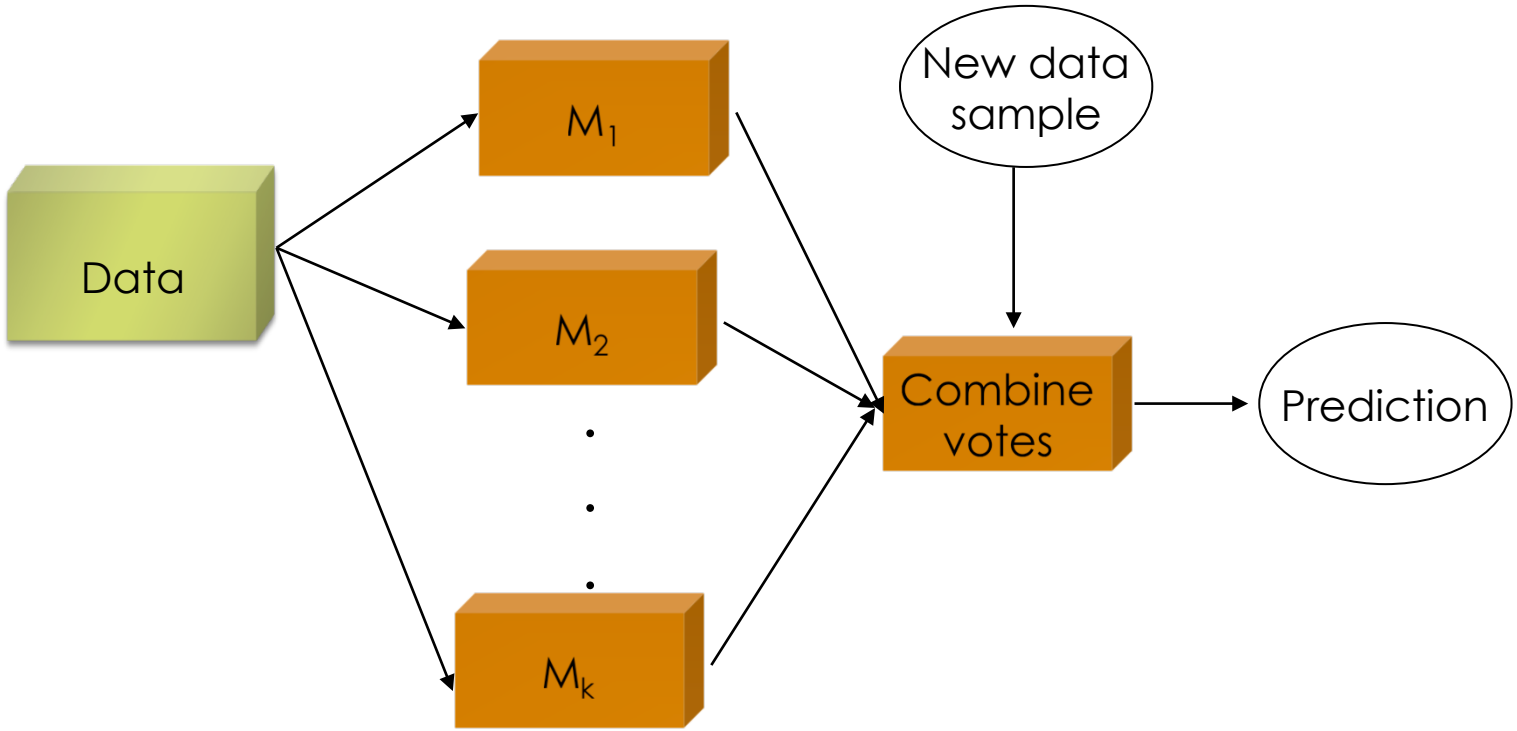
Bagging



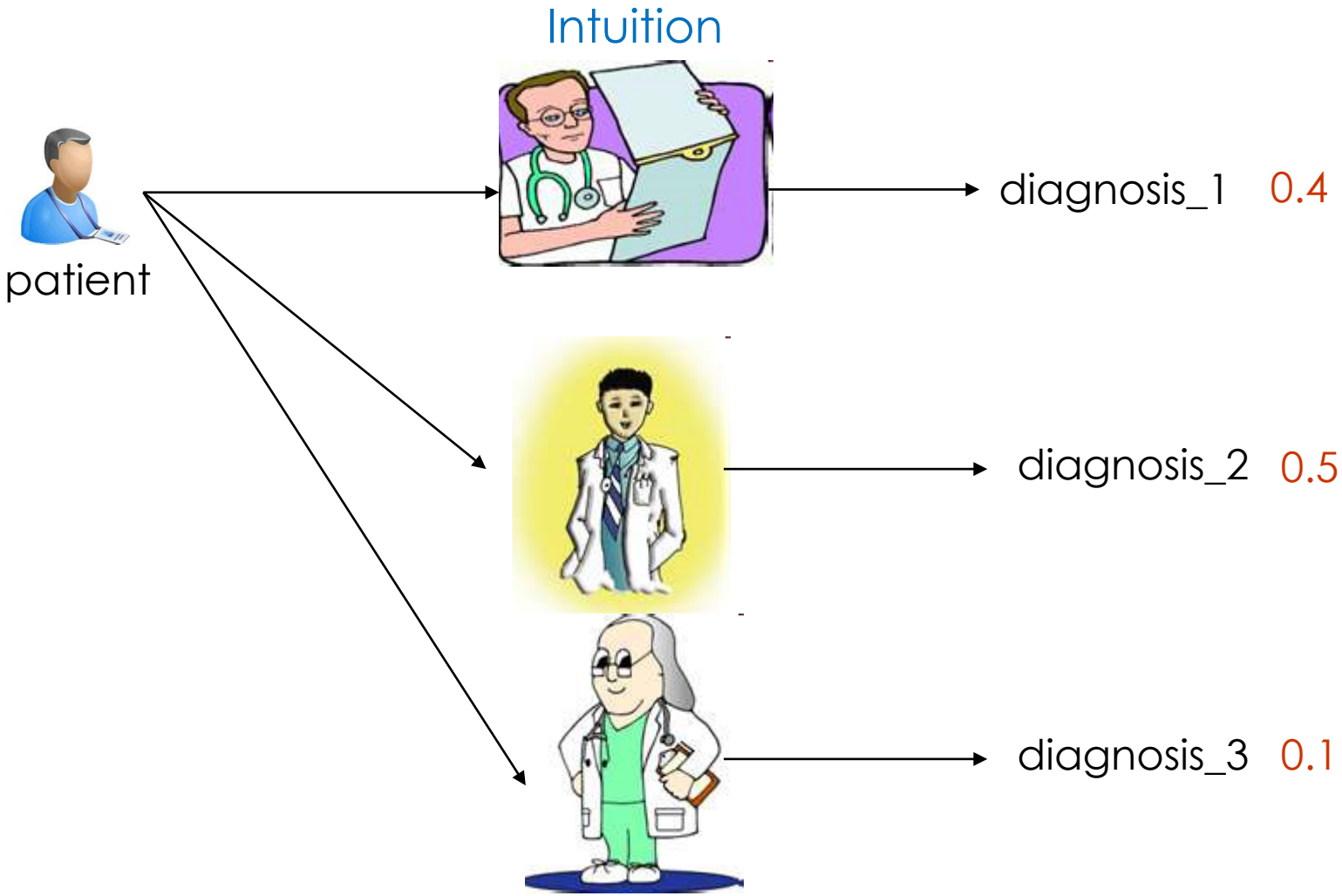
Choose the diagnosis that occurs more than any of the others

Bagging

- K iterations
- At each iteration a training set D_i is sampled with replacement
- The combined model M* returns the most frequent class in case of classification, and the average value in case of prediction



Boosting



Assign different weights to the doctors based on the accuracy of their previous diagnosis

Boosting

- Weights are assigned to each training tuple
- A series of k classifiers is iteratively learned
- After a classifier M_i is learned, the weights are adjusted to allow the subsequent classifier to pay more attention to training tuples misclassified by M_i
- The final boosted classifier M^* combines the votes of each individual classifier where the weight of each classifier is a function of its accuracy
- This strategy can be extended for the prediction of continuous values

Example: The Adaboost Algorithm

- Given a set of d class-labeled tuples (X1, y1), …, (Xd, yd)
- Initially, all the weights of tuples are the same: 1/d
- Generate k classifiers in k rounds.
- At round i, tuples from D are sampled (with replacement) to form a training set Di of the same size
- Each tuple's chance of being selected depends on its weight
- A classification model Mi is derived and tested using Di
- If a tuple is misclassified, its weight increases, otherwise it decreases (use err(Mi)/(1-err(Mi)))

Example: The Adaboost Algorithm

- Error rate err(X_i) is the misclassification error of tuple X_i
- Classifier M_i error rate is the sum of the weights of the misclassified tuples

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

  - Tuple correctly classified: err(Xi)=0
  - Tuple incorrectly classified: err(Xi)=1

- The weight of classifier Mi's vote is

$$\log \frac{1 - error(M_i)}{error(M_i)}$$

Summary

- **Accuracy** is used to assess **classifiers**
- **Error measures** are used to assess **predictors**
- **Stratified 10-fold cross validation** is recommended for estimating accuracy
- **Bagging** and **boosting** are used to improve the the accuracy of classifiers and predictors