K-Nearest Neighbour (Continued) and Model Evaluation

Dr. Xiaowei Huang

https://cgi.csc.liv.ac.uk/~xiaowei/

Up to now,

- Two machine learning algorithms
  - Decision tree learning
  - K-nearest neighbour
    - What is k-nearest-neighbor classification
    - How can we determine similarity/distance
    - Standardizing numeric features (leave this to you)
    - K-NN regression
    - Distance-weighted nearest neighbor
    - Speeding up k-NN
      - edited nearest neighbour
      - k-d trees for nearest neighbour identification

Topics

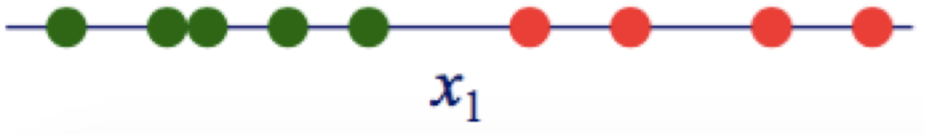
- Locally weighted regression to handle irrelevant features
- Inductive bias

- Test sets revisited
- learning curves
- multiple training/test partitions
  - stratified sampling
  - cross validation
- confusion matrices
  - TP, FP, TN, FN
- ROC curves

Locally weighted regression for Irrelevant features

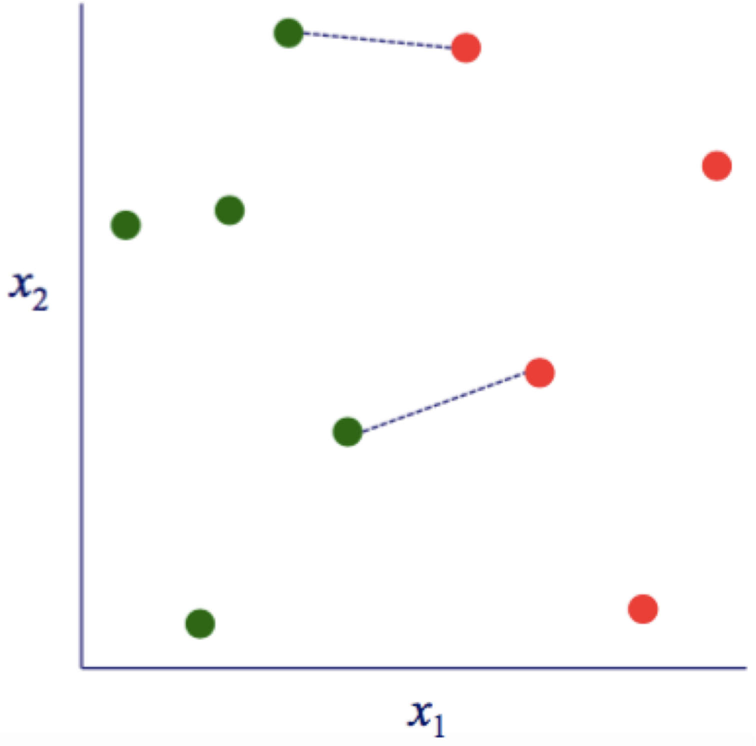
Irrelevant features in instance-based learning

here’s a case in which there is one relevant feature x_1 and a 1-NN rule classifies each instance correctly



Can you find a point (a,b) which is red, if classified only according to feature x1, but is green, if classified according to both features?

consider the effect of an irrelevant feature x_2 on distances and nearest neighbors



Locally weighted regression

- one way around this limitation is to weight features differently
- *locally weighted regression* is one nearest-neighbor variant that does this

- prediction task
  - **given**: an instance $x^{(q)}$ to make a prediction for
  - find the k training-set instances $(\mathbf{x}^{(1)}, y^{(1)}) \ldots (\mathbf{x}^{(k)}, y^{(k)})$ that are most similar to $x^{(q)}$
  - return the value $f(x^{(q)})$

What's function f ?

Locally weighted regression

- Determining function f
  - Assume that f is a linear function over the features, i.e.,

$$f(x^{(i)}) = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots + w_n x_n^{(i)}$$

  - find the weights w_i for each $x^{(q)}$ by minimizing

$$\arg \min_{w_0, w_1, \ldots, x_n} \sum_{i=1}^{k} (f(x^{(i)}) - y^{(i)})^2$$

can do this using gradient descent (to be covered soon)

  - After obtaining weights, for $x^{(q)}$, we have $f(\mathbf{x}^{(q)}) = w_0 + w_1 x_1^{(q)} + w_2 x_2^{(q)} + \ldots + w_n x_n^{(q)}$

Discussions

Strengths of instance-based learning

- simple to implement
- “training” is very efficient
- adapts well to on-line learning
- robust to noisy training data (when k > 1)
- often works well in practice

Limitations of instance-based learning

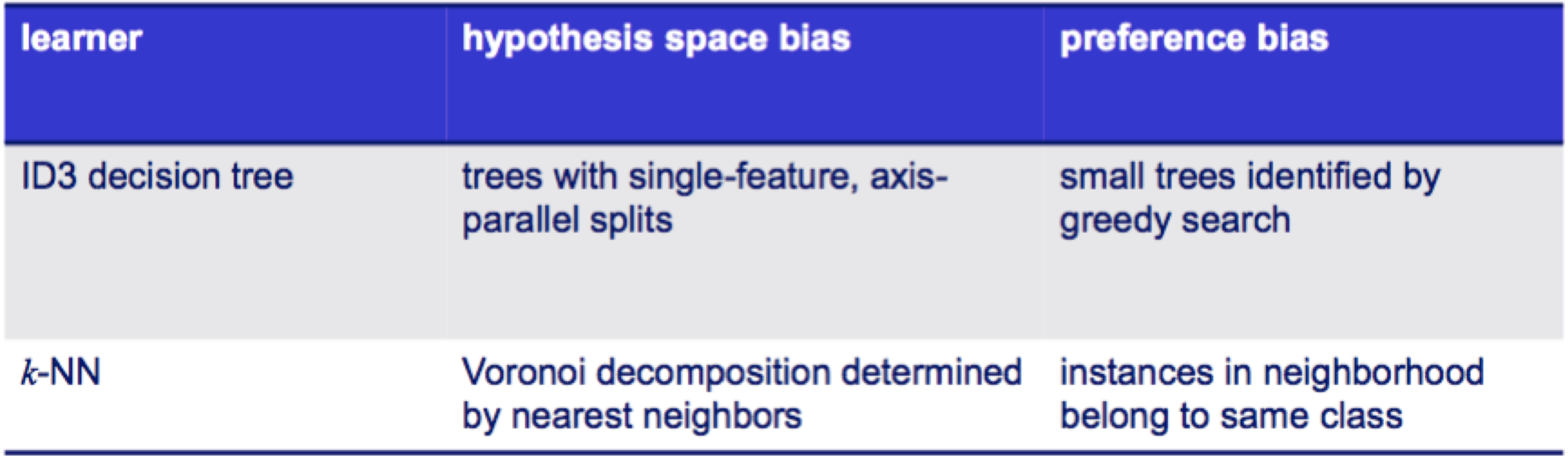
- sensitive to range of feature values
- sensitive to irrelevant and correlated features, although ...
    - there are variants (such as locally weighted regression) that learn weights for different features
- classification/prediction can be inefficient, although …
    - edited methods and k-d trees can help alleviate this weakness
- doesn’t provide much insight into problem domain because there is no explicit model

Inductive bias

Inductive bias

- *inductive bias* is the set of assumptions a learner uses to be able to predict y_i for a previously unseen instance x_i
- two components
  - *hypothesis space bias*: determines the models that can be represented
  - *preference bias*: specifies a preference ordering within the space of models
- in order to *generalize* (i.e. make predictions for previously unseen instances) a learning algorithm must have an inductive bias

Consider the inductive bias of DT and k-NN learners

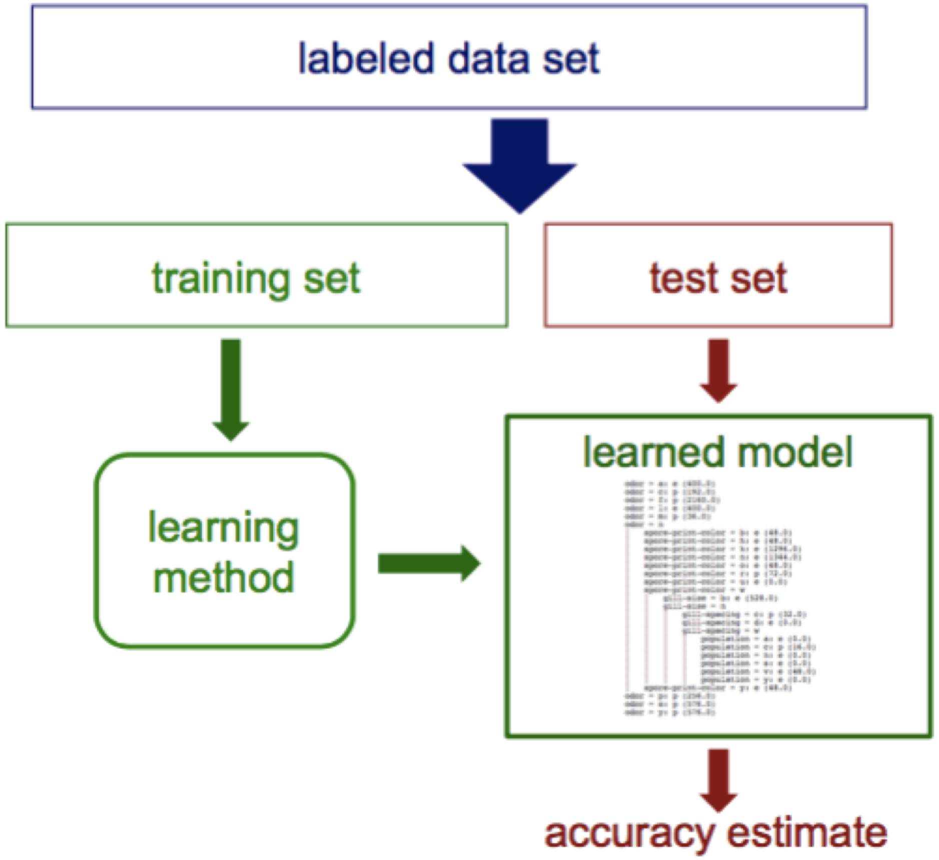


learner	hypothesis space bias	preference bias
ID3 decision tree	trees with single-feature, axis-parallel splits	small trees identified by greedy search
k-NN	Voronoi decomposition determined by nearest neighbors	instances in neighborhood belong to same class

Test sets revisited

Test sets revisited

- How can we get an unbiased estimate of the accuracy of a learned model?



labeled data set

training set

test set

learning method

learned model

accuracy estimate

Test sets revisited

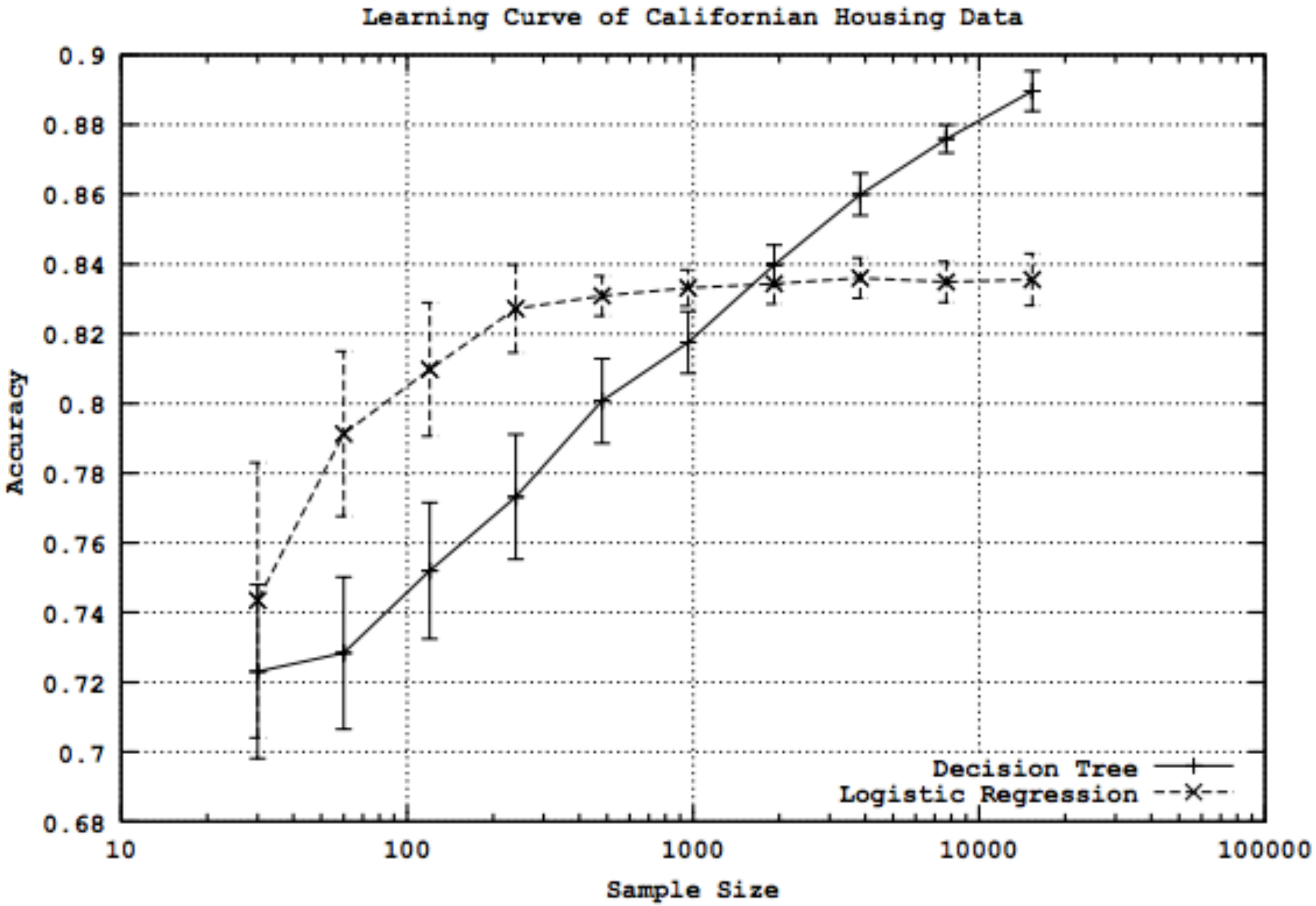
- How can we get an unbiased estimate of the accuracy of a learned model?
  - when learning a model, you should pretend that you don’t have the test data yet (it is “in the mail”)*
  - if the test-set labels influence the learned model in any way, accuracy estimates will be biased

* In some applications it is reasonable to assume that you have access to the feature vector (i.e. x) but not the y part of each test instance.

Learning Curve

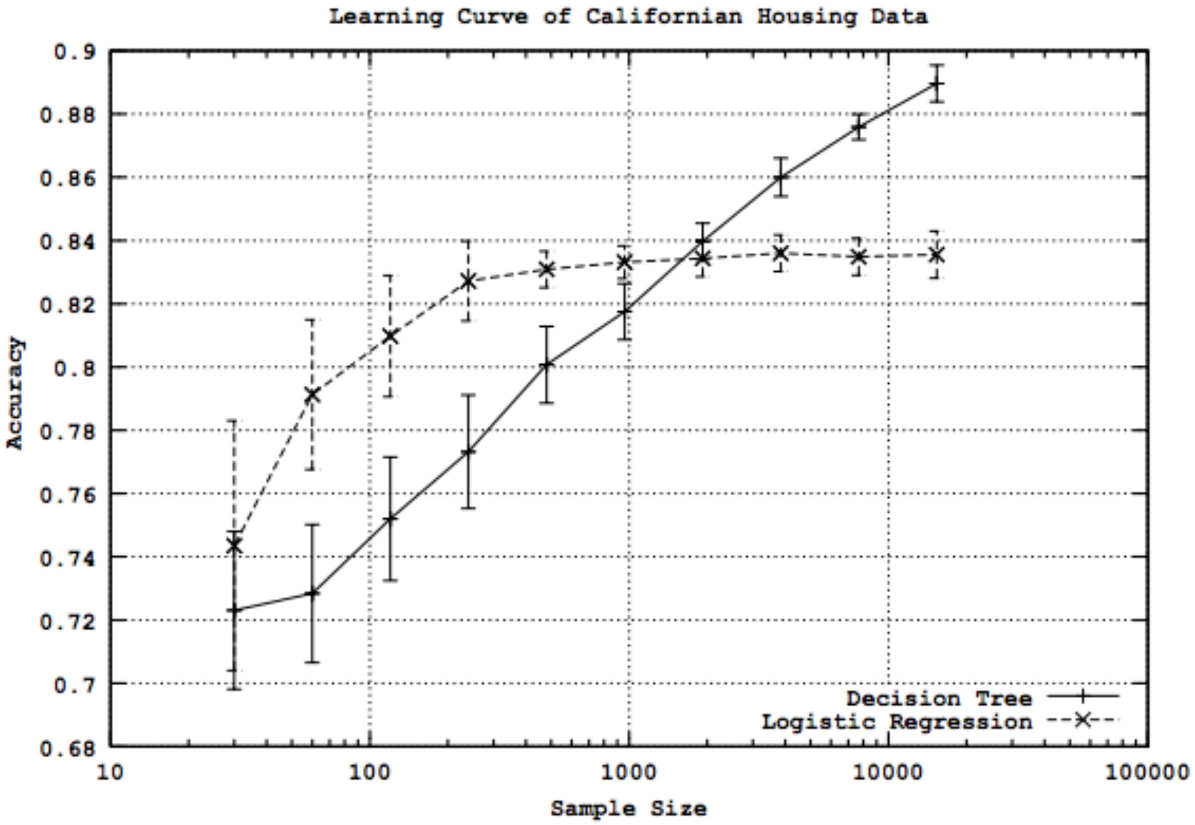
Learning curves

- How does the accuracy of a learning method change as a function of the training-set size?
  - this can be assessed by plotting *learning curves*



Learning curves

- given training/test set partition
  - for each sample size s on learning curve
    - (optionally) repeat n times
      - randomly select s instances from training set
      - learn model
      - evaluate model on test set to determine accuracy a
      - plot (s, a) or (s, avg. accuracy and error bars)



multiple training/test partitions

Limitations of using a single training/test partition

- we may not have enough data to make sufficiently large training and test sets
  - a larger test set gives us more reliable estimate of accuracy (i.e. a lower variance estimate)
  - but... a larger training set will be more representative of how much data we actually have for learning process

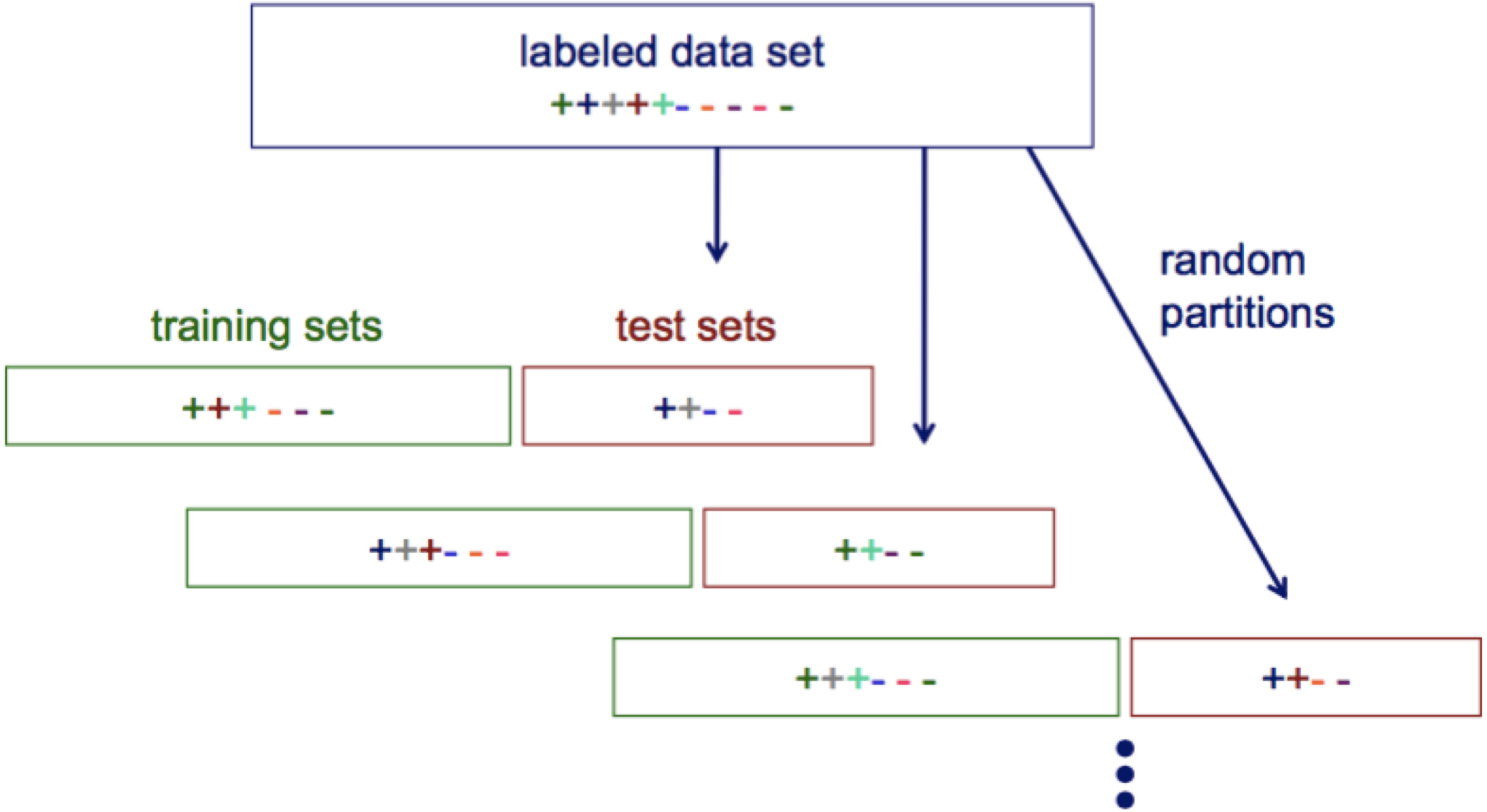
- a single training set doesn't tell us how sensitive accuracy is to a particular training sample

Using multiple training/test partitions

- two general approaches for doing this
  - random resampling
  - cross validation

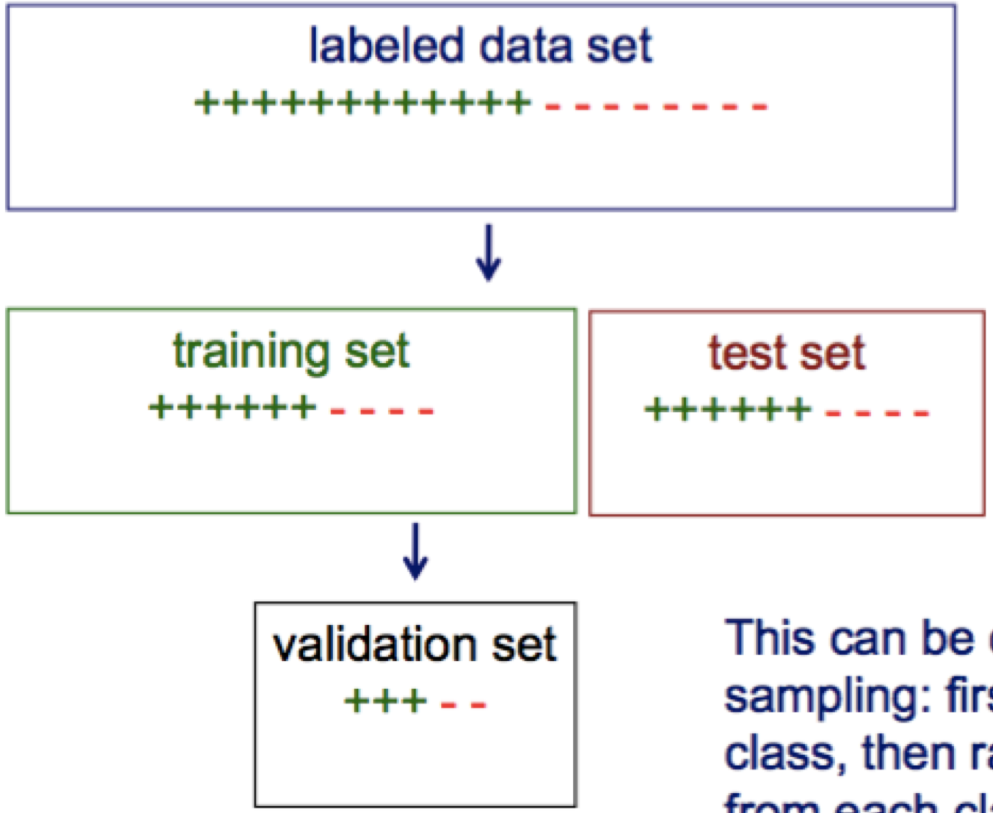
Random resampling

- We can address the second issue by repeatedly randomly partitioning the available data into training and test sets.



Stratified sampling

- When randomly selecting training or validation sets, we may want to ensure that class proportions are maintained in each selected set

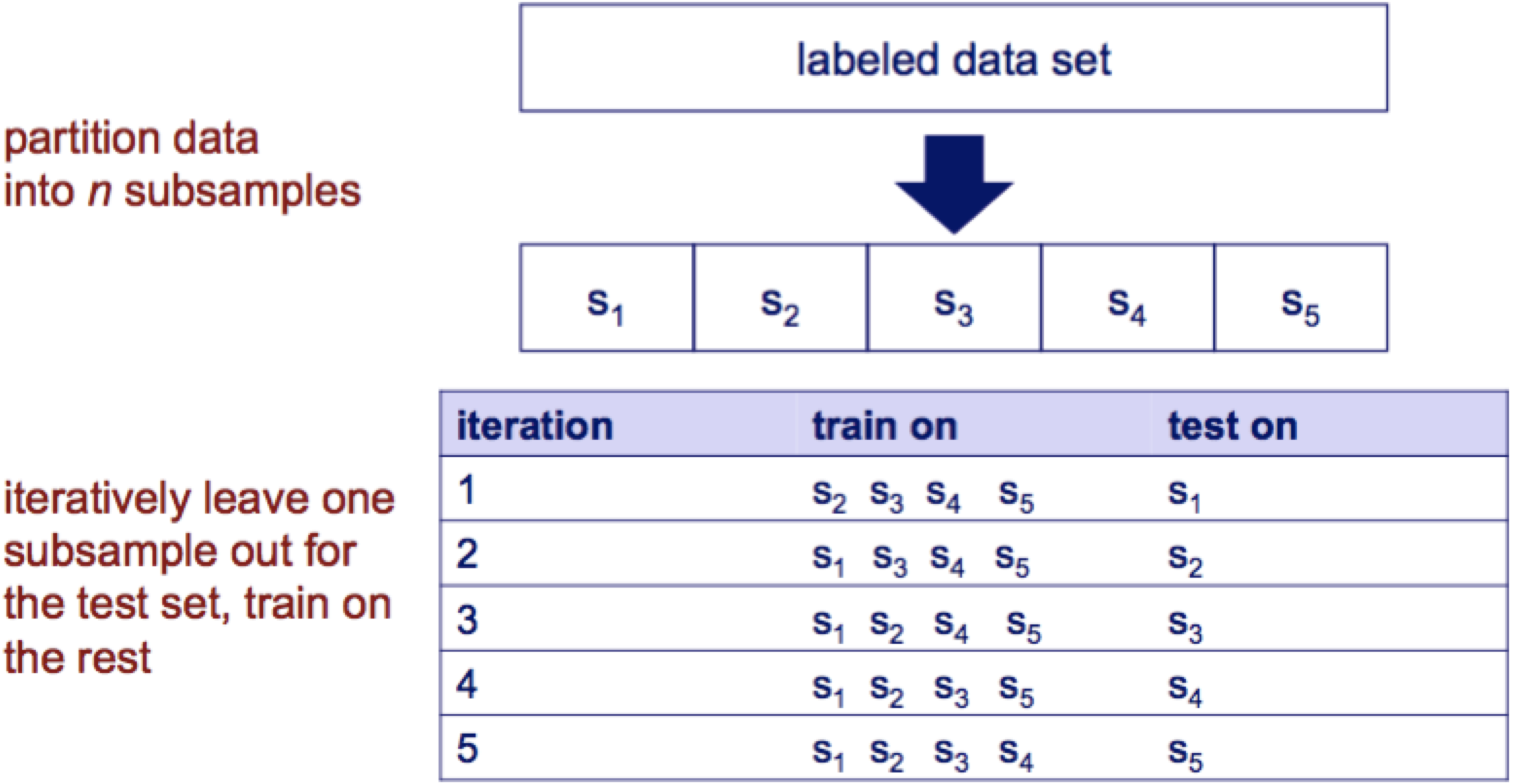


This can be done via stratified sampling: first stratify instances by class, then randomly select instances from each class proportionally.

Recall: a *validation set* (a.k.a. *tuning set*) is a subset of the training set that is held aside

Validation datasets can be used for regularization by early stopping: stop training when the error on the validation dataset increases, as this is a sign of overfitting to the training dataset

Cross validation



partition data
into *n* subsamples

labeled data set

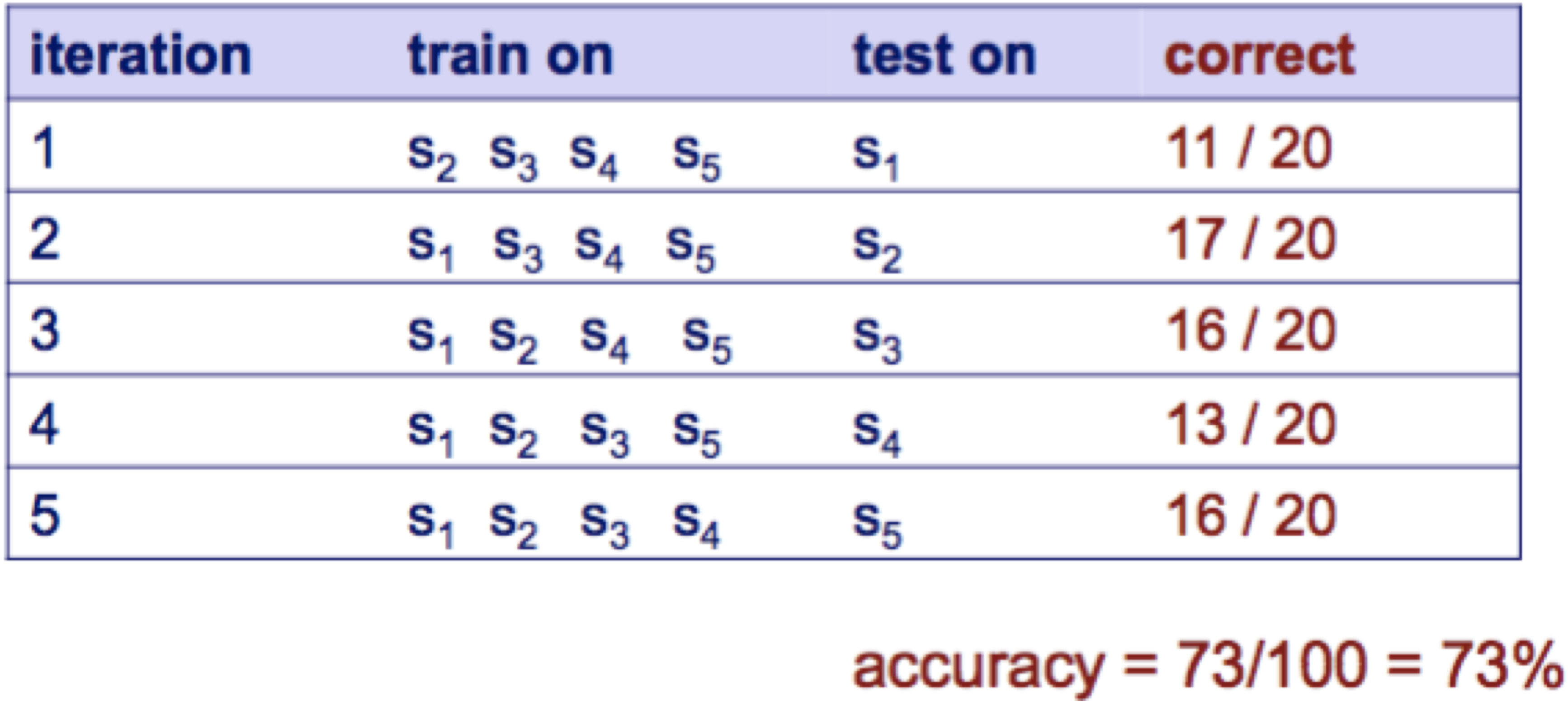
s_1 s_2 s_3 s_4 s_5

iteratively leave one subsample out for the test set, train on the rest

iteration	train on	test on
1	s_2 s_3 s_4 s_5	s_1
2	s_1 s_3 s_4 s_5	s_2
3	s_1 s_2 s_4 s_5	s_3
4	s_1 s_2 s_3 s_5	s_4
5	s_1 s_2 s_3 s_4	s_5

Cross validation example

- Suppose we have 100 instances, and we want to estimate accuracy with cross validation



iteration	train on	test on	correct
1	s_2 s_3 s_4 s_5	s_1	11 / 20
2	s_1 s_3 s_4 s_5	s_2	17 / 20
3	s_1 s_2 s_4 s_5	s_3	16 / 20
4	s_1 s_2 s_3 s_5	s_4	13 / 20
5	s_1 s_2 s_3 s_4	s_5	16 / 20

accuracy = 73/100 = 73%

Cross validation

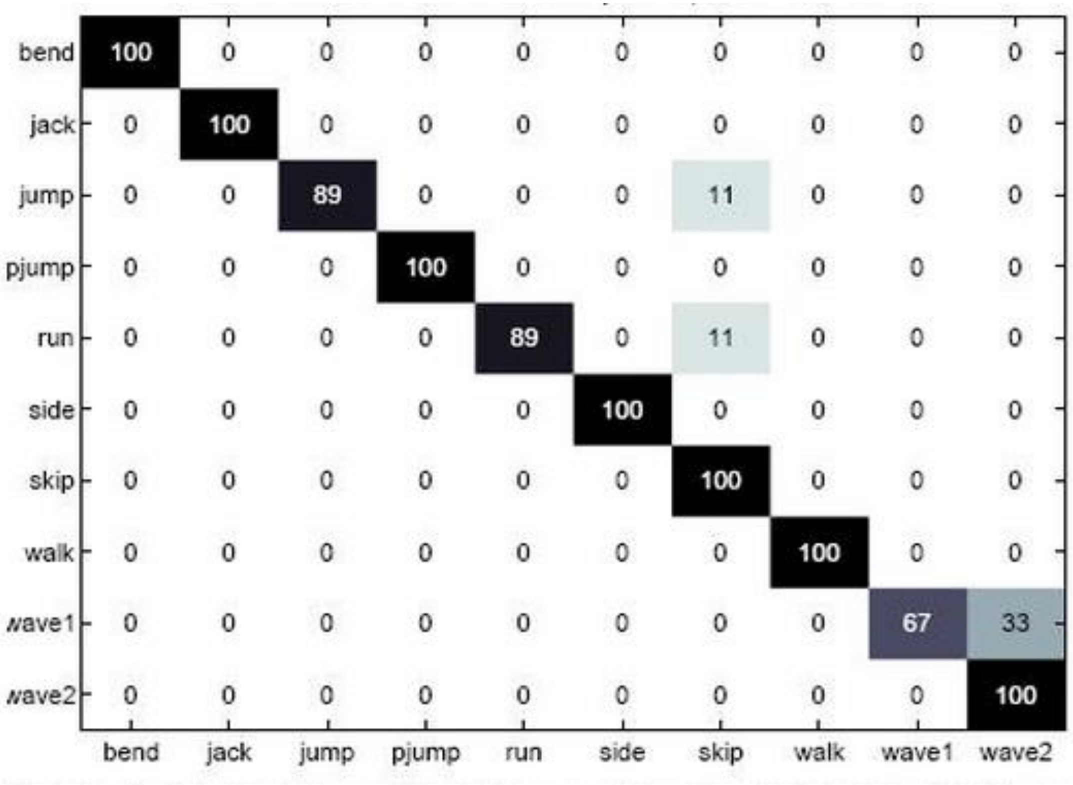
- 10-fold cross validation is common, but smaller values of *n* are often used when learning takes a lot of time
- in *leave-one-out* cross validation, *n* = # instances
- in *stratified* cross validation, stratified sampling is used when partitioning the data
- Cross validation makes efficient use of the available data for testing

Confusion matrices

Confusion matrices

- How can we understand what types of mistakes a learned model makes?

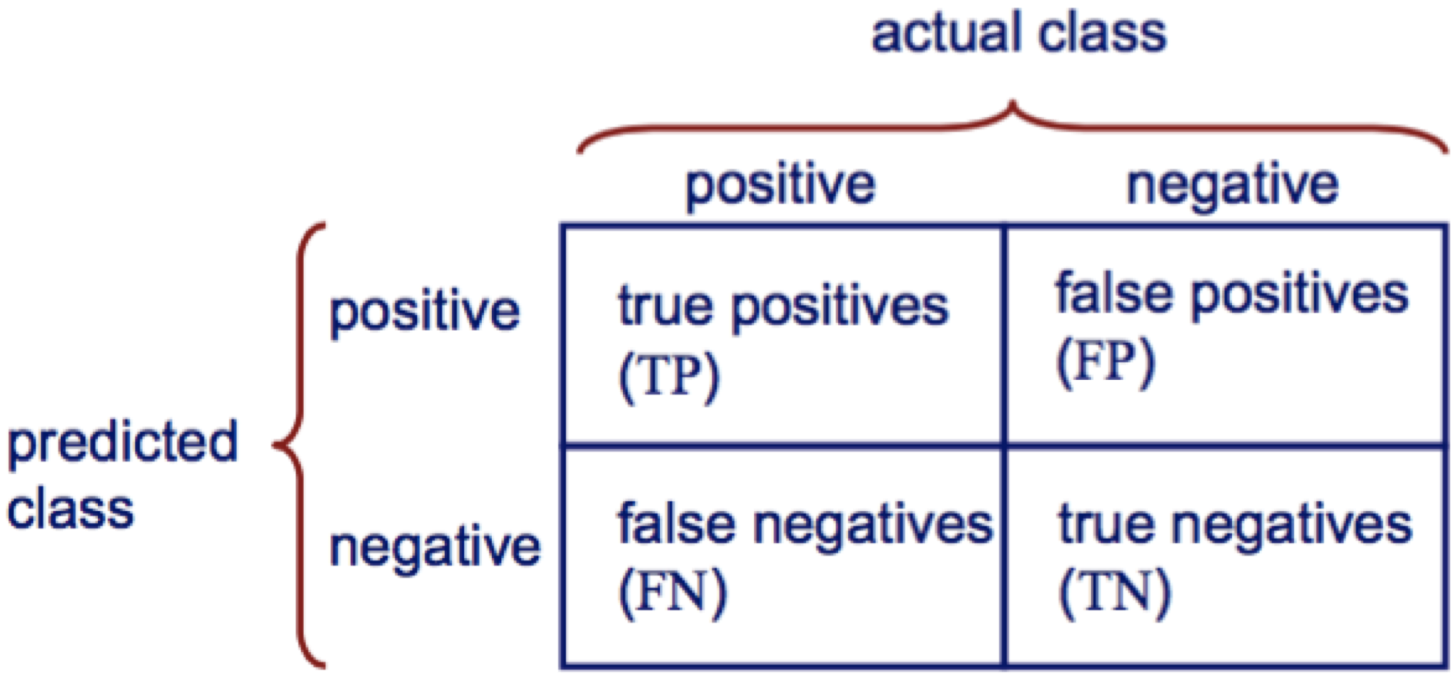
actual class



	bend	jack	jump	pjump	run	side	skip	walk	wave1	wave2
bend	100	0	0	0	0	0	0	0	0	0
jack	0	100	0	0	0	0	0	0	0	0
jump	0	0	89	0	0	0	11	0	0	0
pjump	0	0	0	100	0	0	0	0	0	0
run	0	0	0	0	89	0	11	0	0	0
side	0	0	0	0	0	100	0	0	0	0
skip	0	0	0	0	0	0	100	0	0	0
walk	0	0	0	0	0	0	0	100	0	0
wave1	0	0	0	0	0	0	0	0	67	33
wave2	0	0	0	0	0	0	0	0	0	100

predicted class

Confusion matrix for 2-class problems



		actual class	
		positive	negative
predicted class	positive	true positives (TP)	false positives (FP)
	negative	false negatives (FN)	true negatives (TN)

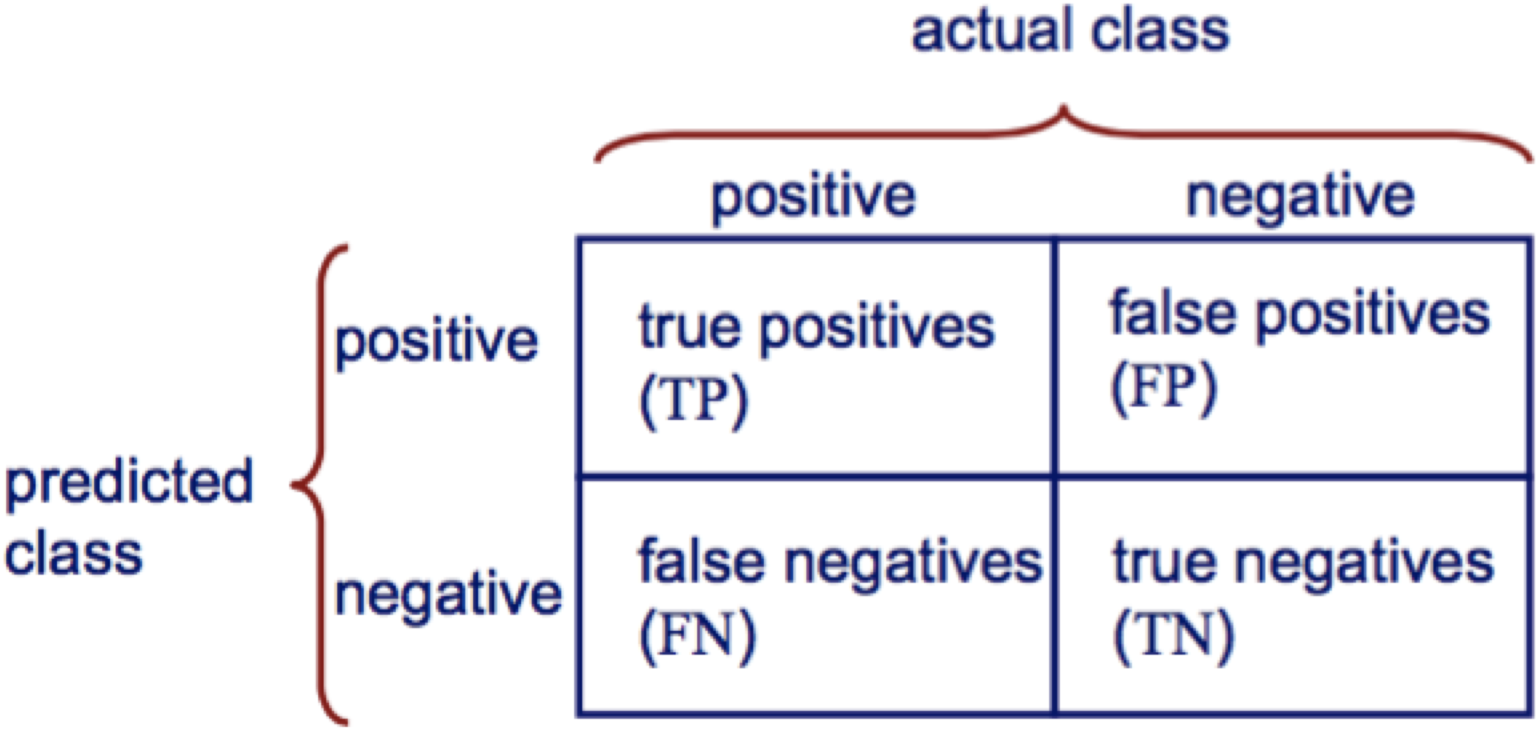
$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

$$\text{error} = 1 - \text{accuracy} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

Is accuracy an adequate measure of predictive performance?

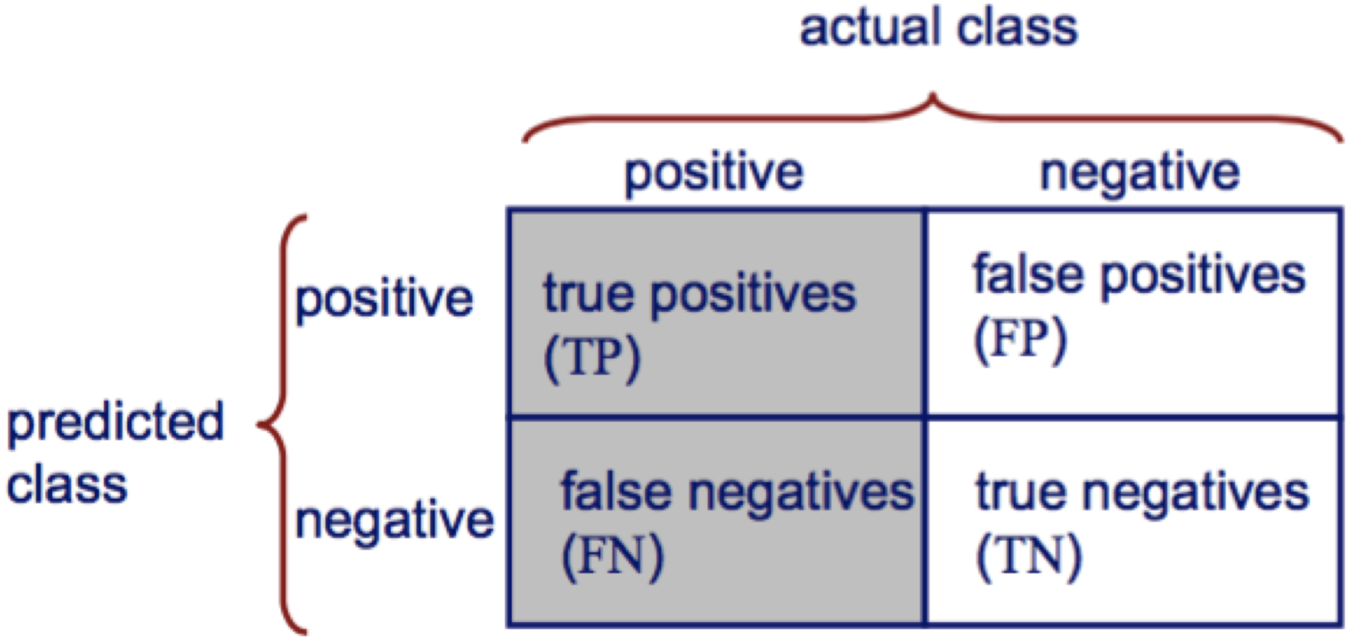
- accuracy may not be a useful measure in cases where
  - there is a large class skew
    - Is 98% accuracy good when 97% of the instances are negative?
  - there are differential misclassification costs – say, getting a positive wrong costs more than getting a negative wrong
    - Consider a medical domain in which a false positive results in an extraneous test but a false negative results in a failure to treat a disease
- we are most interested in a subset of high-confidence predictions

Other accuracy metrics



		actual class	
		positive	negative
predicted class	positive	true positives (TP)	false positives (FP)
	negative	false negatives (FN)	true negatives (TN)

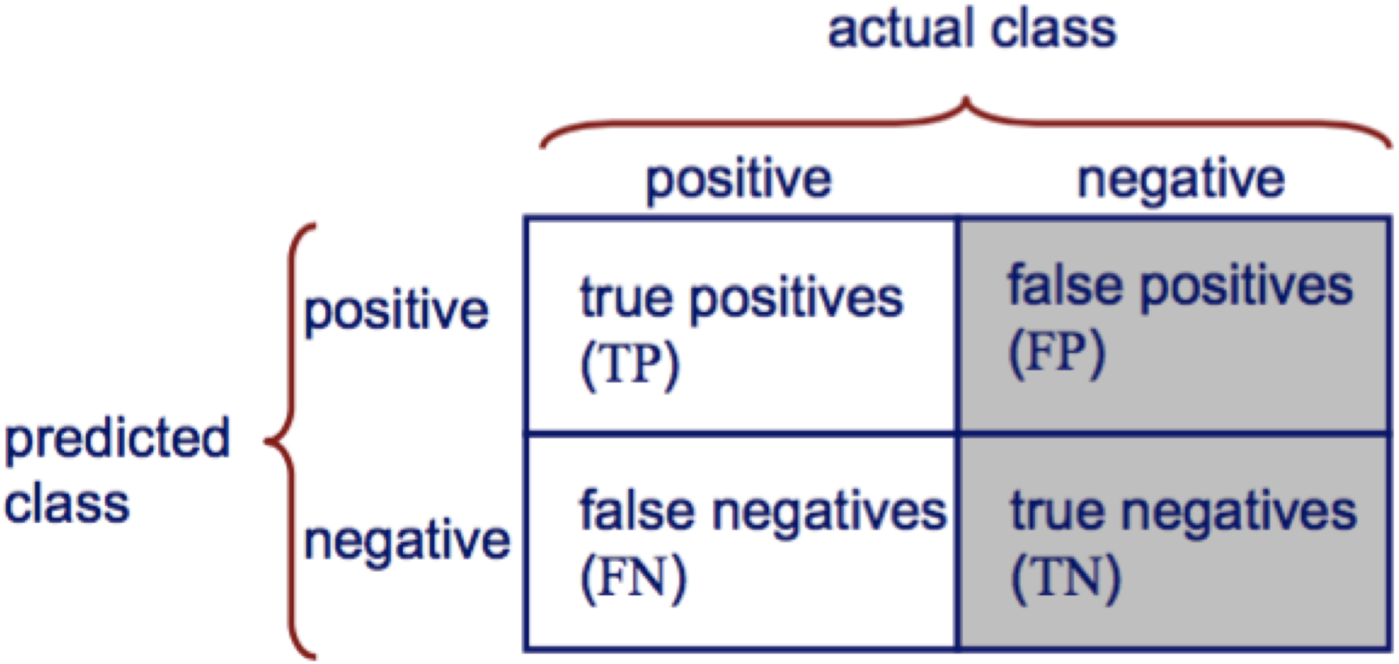
Other accuracy metrics



		actual class	
		positive	negative
predicted class	positive	true positives (TP)	false positives (FP)
	negative	false negatives (FN)	true negatives (TN)

$$\text{true positive rate (recall)} = \frac{\text{TP}}{\text{actual pos}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Other accuracy metrics



		actual class	
		positive	negative
predicted class	positive	true positives (TP)	false positives (FP)
	negative	false negatives (FN)	true negatives (TN)

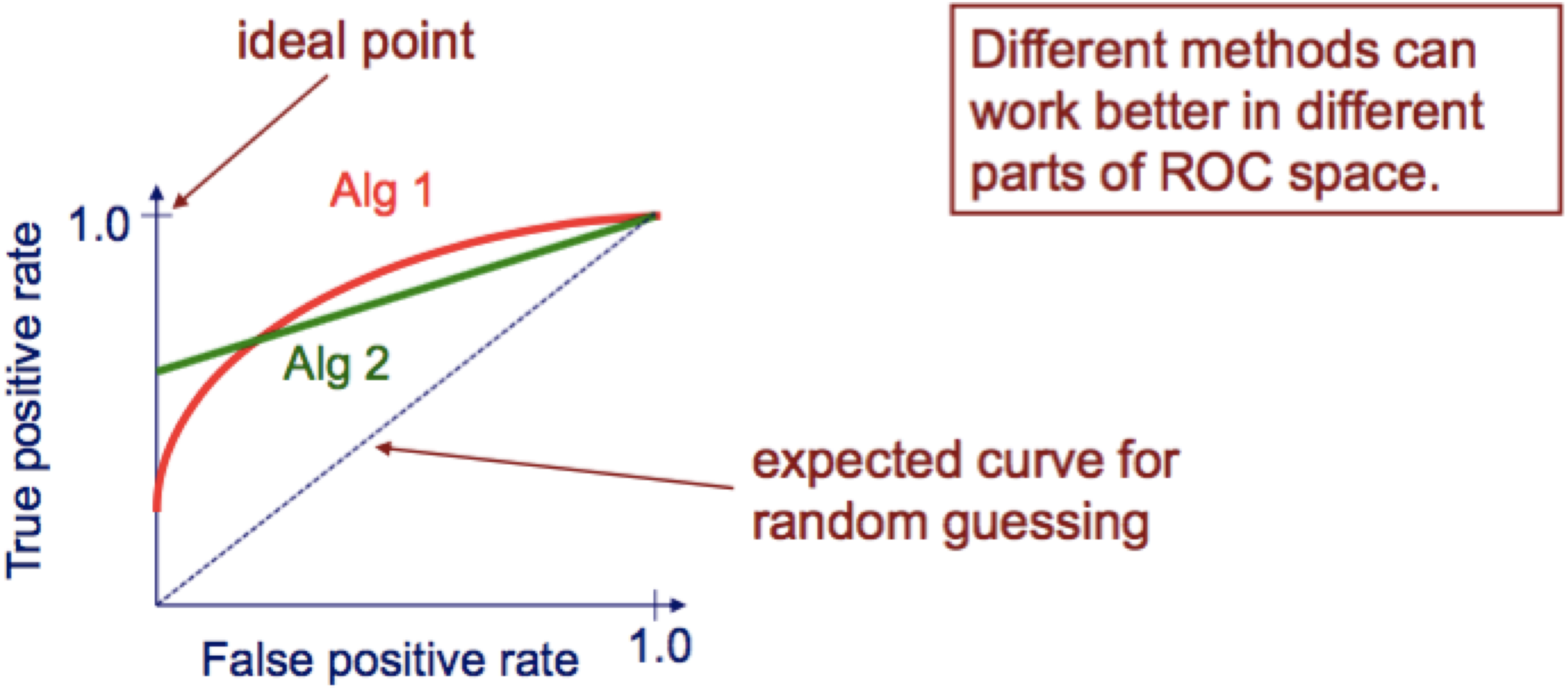
$$\text{true positive rate (recall)} = \frac{\text{TP}}{\text{actual pos}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{false positive rate} = \frac{\text{FP}}{\text{actual neg}} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

ROC curves

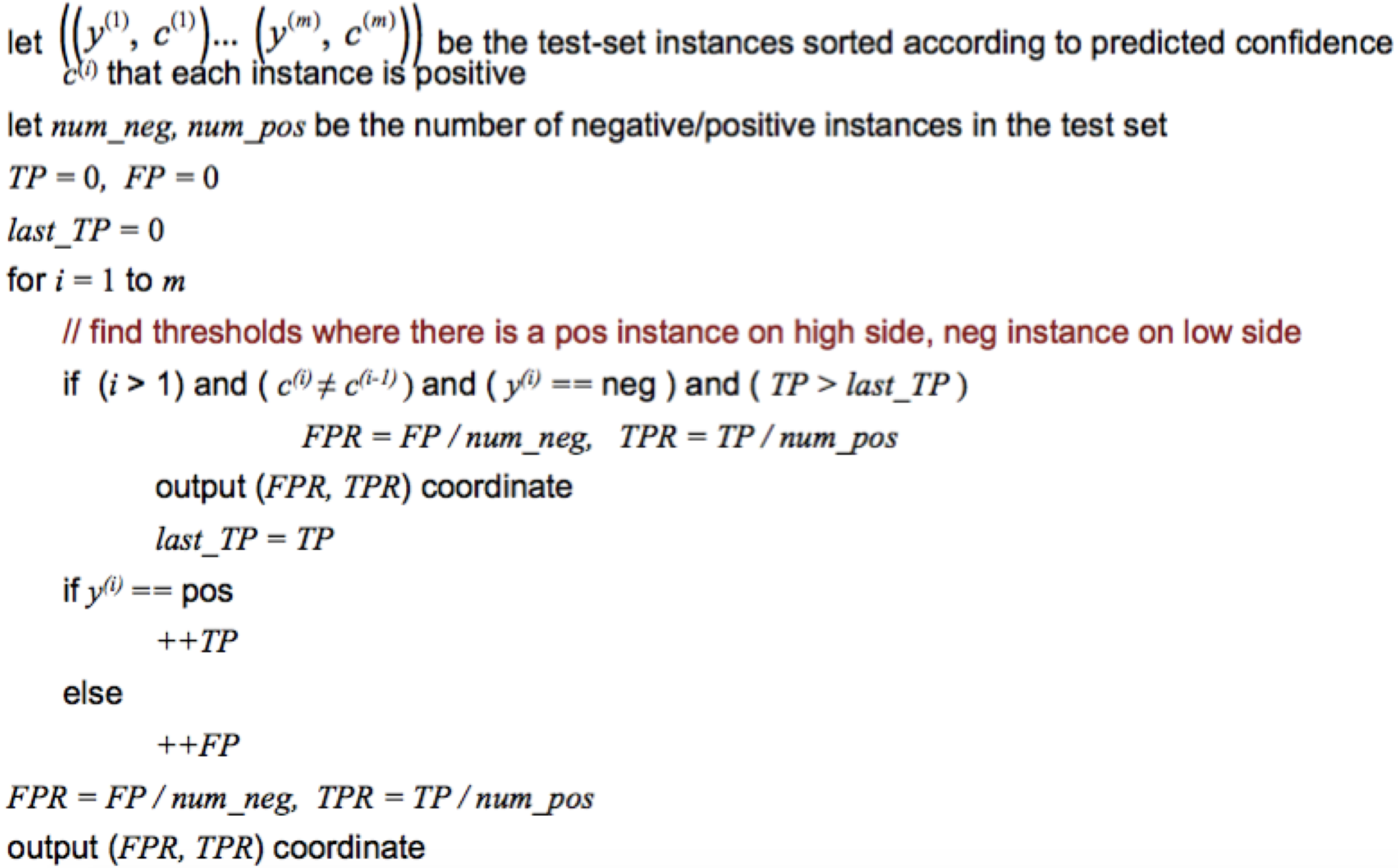
ROC curves

- A *Receiver Operating Characteristic* (*ROC*) curve plots the TP-rate vs. the FP-rate as a threshold on the confidence of an instance being positive is varied



Different methods can work better in different parts of ROC space.

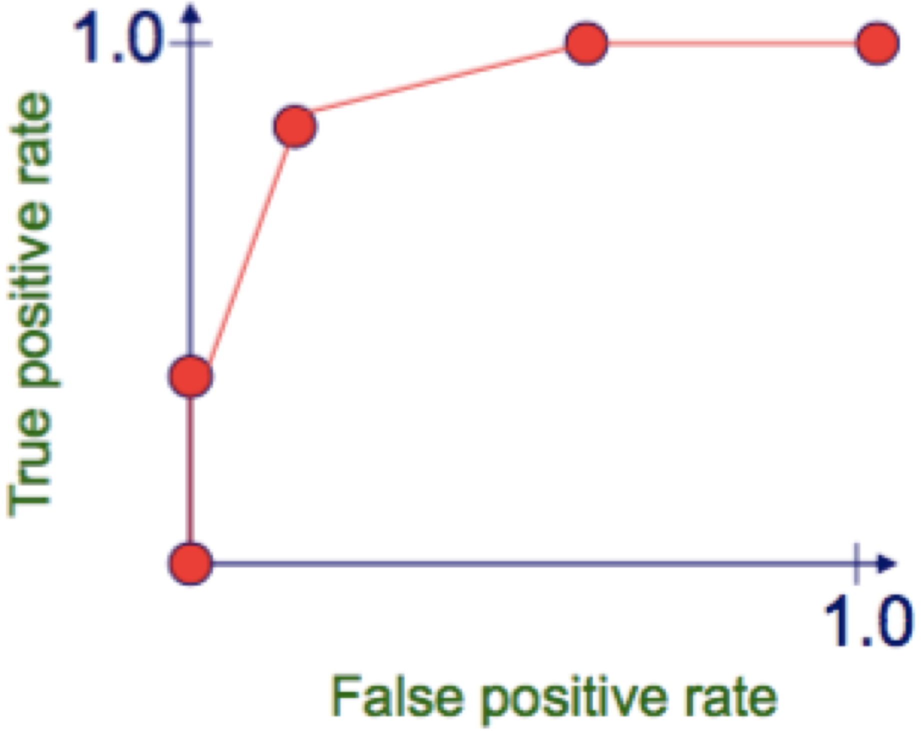
Algorithm for creating an ROC curve



```
let $((y^{(1)}, c^{(1)}) \dots (y^{(m)}, c^{(m)}))$ be the test-set instances sorted according to predicted confidence
    $c^{(i)}$ that each instance is positive
let $num\_neg$, $num\_pos$ be the number of negative/positive instances in the test set
$TP = 0$,  $FP = 0$
$last\_TP = 0$
for $i = 1$ to $m$
    // find thresholds where there is a pos instance on high side, neg instance on low side
    if ($i > 1$) and ( $c^{(i)} \neq c^{(i-1)}$ ) and ( $y^{(i)}$ == neg ) and ( $TP > last\_TP$ )
                    $FPR = FP / num\_neg$,   $TPR = TP / num\_pos$
        output ($FPR$, $TPR$) coordinate
        $last\_TP = TP$
    if $y^{(i)}$ == pos
        ++$TP$
    else
        ++$FP$
$FPR = FP / num\_neg$,  $TPR = TP / num\_pos$
output ($FPR$, $TPR$) coordinate
```

Plotting an ROC curve

instance	confidence positive		correct class
Ex 9	.99		+
Ex 7	.98	TPR= 2/5, FPR= 0/5	+
Ex 1	.72		-
Ex 2	.70		+
Ex 6	.65	TPR= 4/5, FPR= 1/5	+
Ex 10	.51		-
Ex 3	.39		-
Ex 5	.24	TPR= 5/5, FPR= 3/5	+
Ex 4	.11		-
Ex 8	.01	TPR= 5/5, FPR= 5/5	-



ROC curve example

task: recognizing genomic units called operons

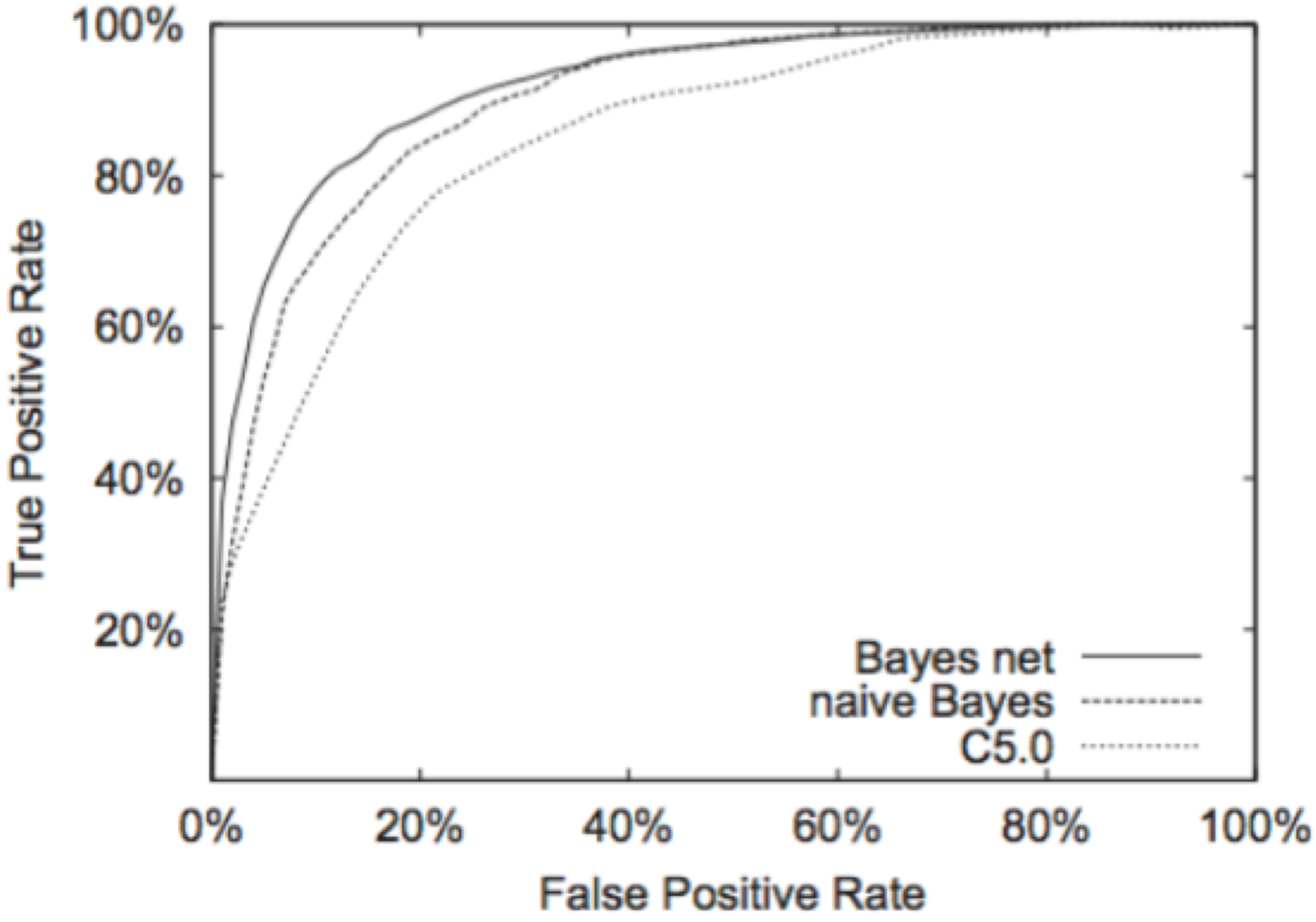
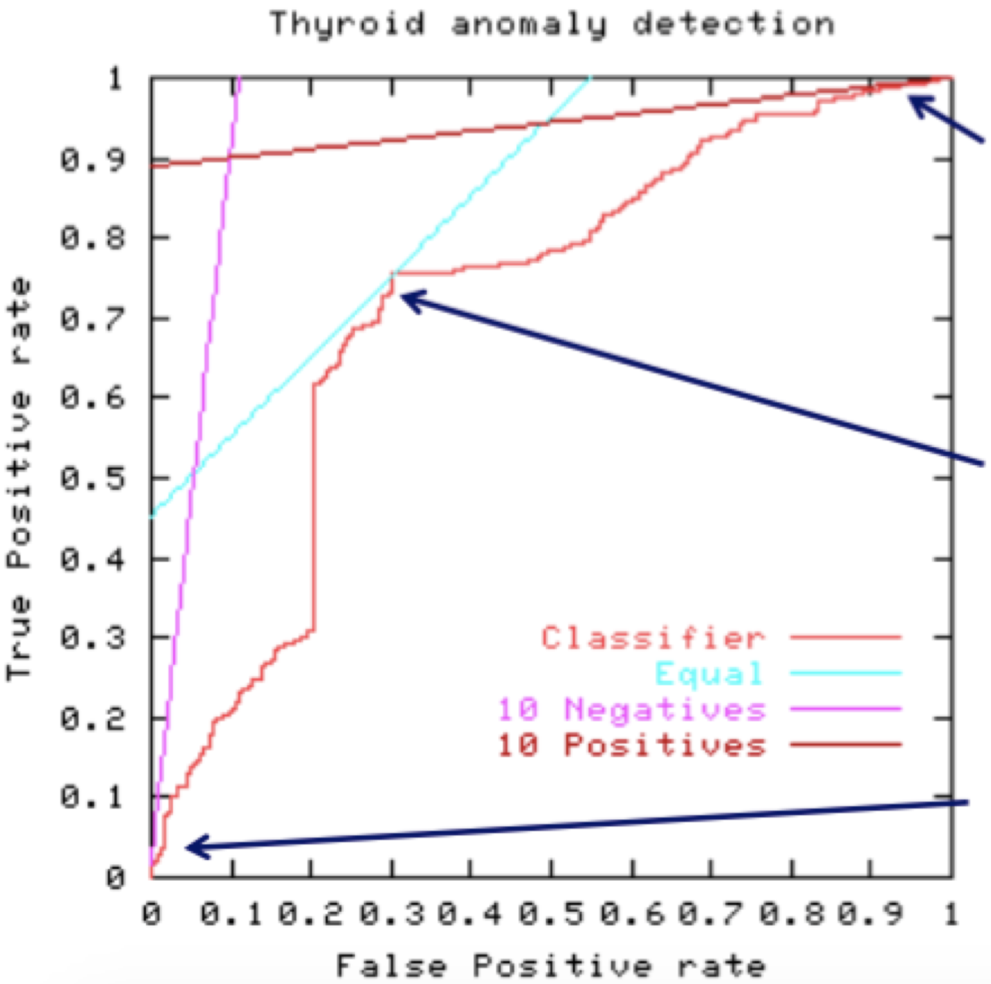


figure from Bockhorst et al., *Bioinformatics* 2003

ROC curves and misclassification costs

The best operating point depends on the relative costs of FN and FP misclassifications



best operating point when FN costs 10× FP

best operating point when cost of misclassifying positives and negatives is equal

best operating point when FP costs 10× FN