Data Mining & Machine Learning

CS37300
Purdue University

September 27, 2017

Midterm

- Monday, Oct 2nd, 1:30 - 2:20pm (in-class)

Questions will be related to:

1) Basic probability knowledge (compute marginals, if two events are independent given a third compute the joint probability of all three events, ETC (other similar questions), i.e., questions similar to the ones in the homework)

2) Given a small dataset, describe a greedy algorithm to build a decision tree (greedily). Familiarize yourself with entropy gain and gini index. Give the final tree.

3) Given a small dataset, describe the algorithm to build the Naive Bayes classifier. Give the classifier for the data.

4) A question about learning curves

Predictive modeling: evaluation

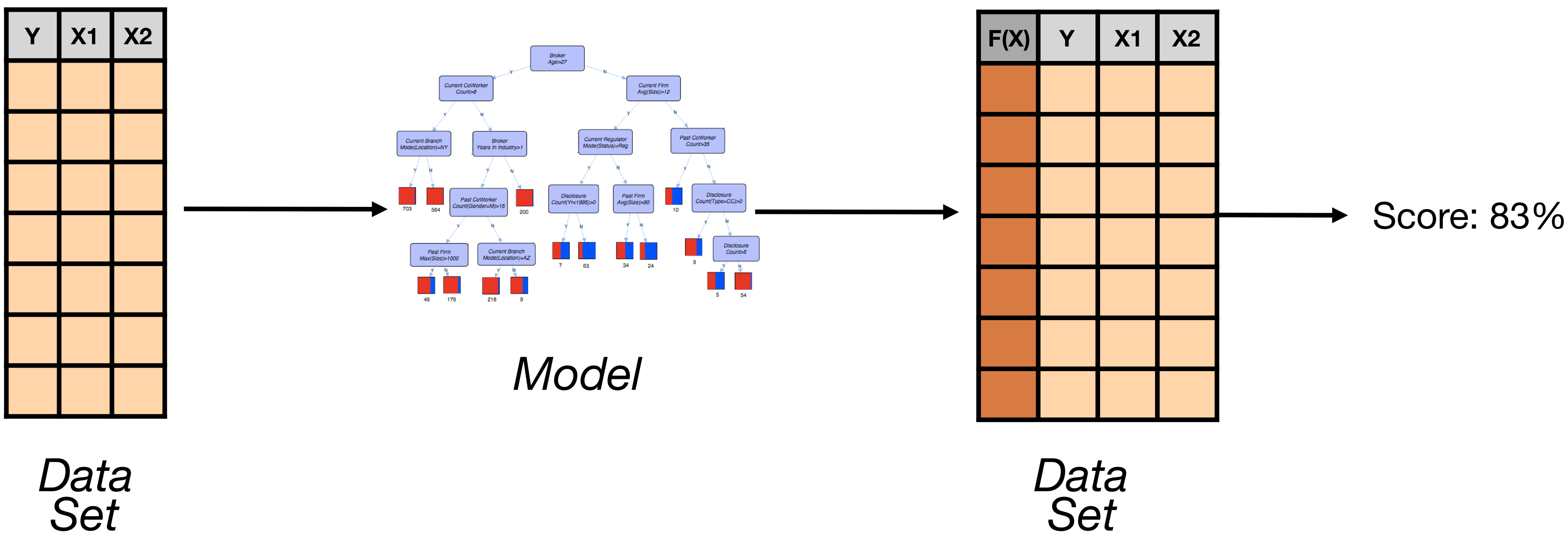
Empirical evaluation

- Given observed accuracy of a model on limited data, how well does this estimate generalize for additional examples?
- Given that one model outperforms another on some sample of data, how likely is it that this model is more accurate in general?
- When data are limited, what is the best way to use the data to both learn and evaluate a model?

Evaluating classifiers

- Goal: Estimate true future error rate
- When data are limited, what is the best way to use the data to both learn and evaluate a model?
- Approach 1
    - Reclassify training data to estimate error rate

Approach 1

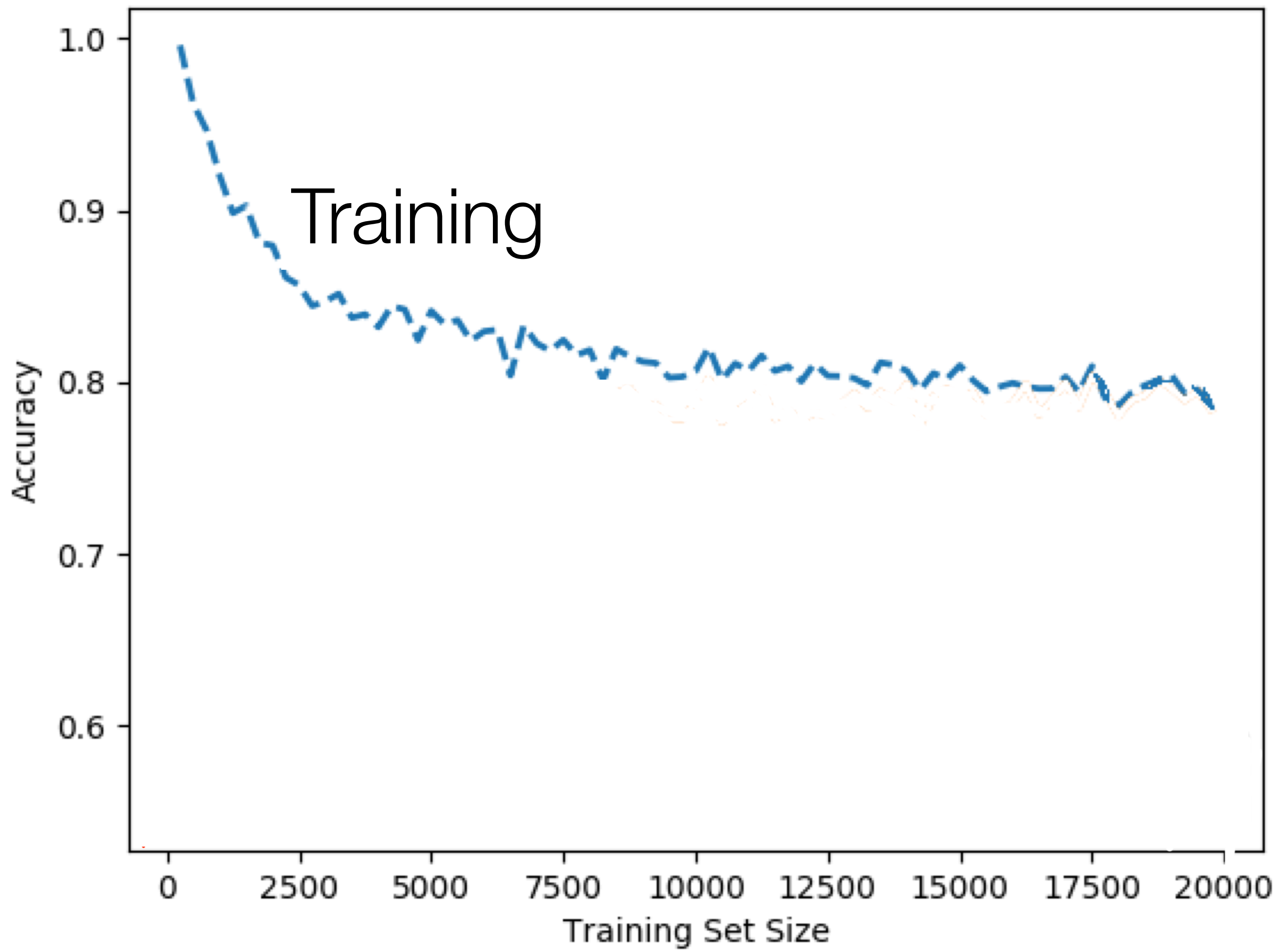


Typically produces a biased estimate of future error rate -- why?

Learning curves

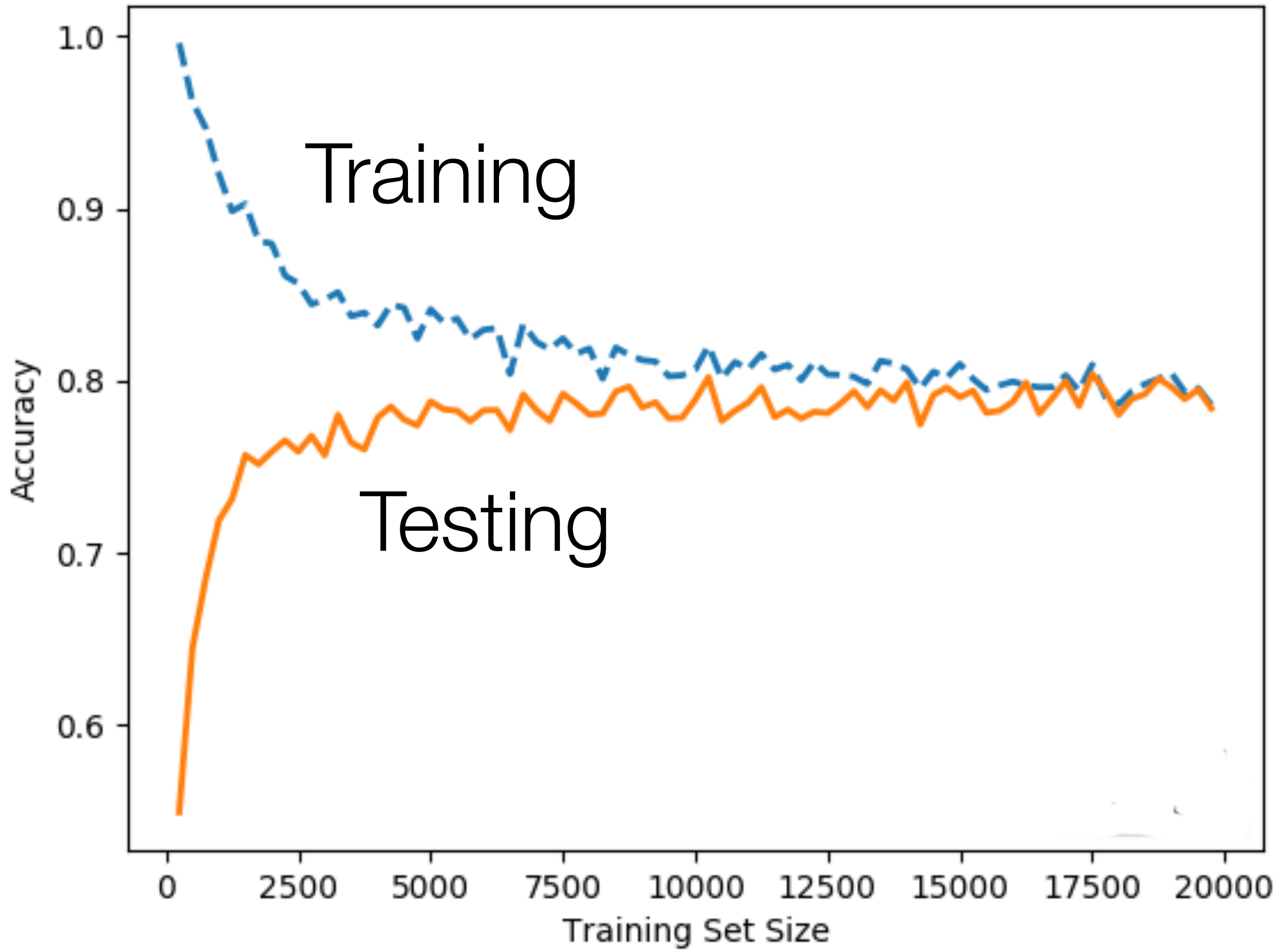
- **Goal**: See how performance improves with additional training data

- From dataset set S, where |S|=n
  - For i=[10, 20, ... ,100]
    - Randomly sample i% of S to construct sample S'
    - Learn model on S'
    - Evaluate model
  - Plot training set size vs. accuracy



Training
Accuracy
1.0
0.9
0.8
0.7
0.6
0
2500
5000
7500
10000
12500
15000
17500
20000
Training Set Size

How does performance change when measured on disjoint test set?


1.0
0.9
0.8
0.7
0.6
Accuracy
Training
Testing
0
2500
5000
7500
10000
12500
15000
17500
20000
Training Set Size

Overfitting

- Consider a **distribution D** of data representing a population and a **sample D_S** drawn from D, which is used as training data

- Given a model space M, a score function S, and a learning algorithm that returns a model $m \in M$:

  The learning algorithm **overfits** the training data D_S if:
  $\exists\, m' \in M$ such that $S(m,D_S) > S(m',D_S)$ but $S(m,D) < S(m',D)$

  In other words, there is another model (m') that is better on the full data distribution and if we had learned from all the data we would have selected it instead

Example learning problem

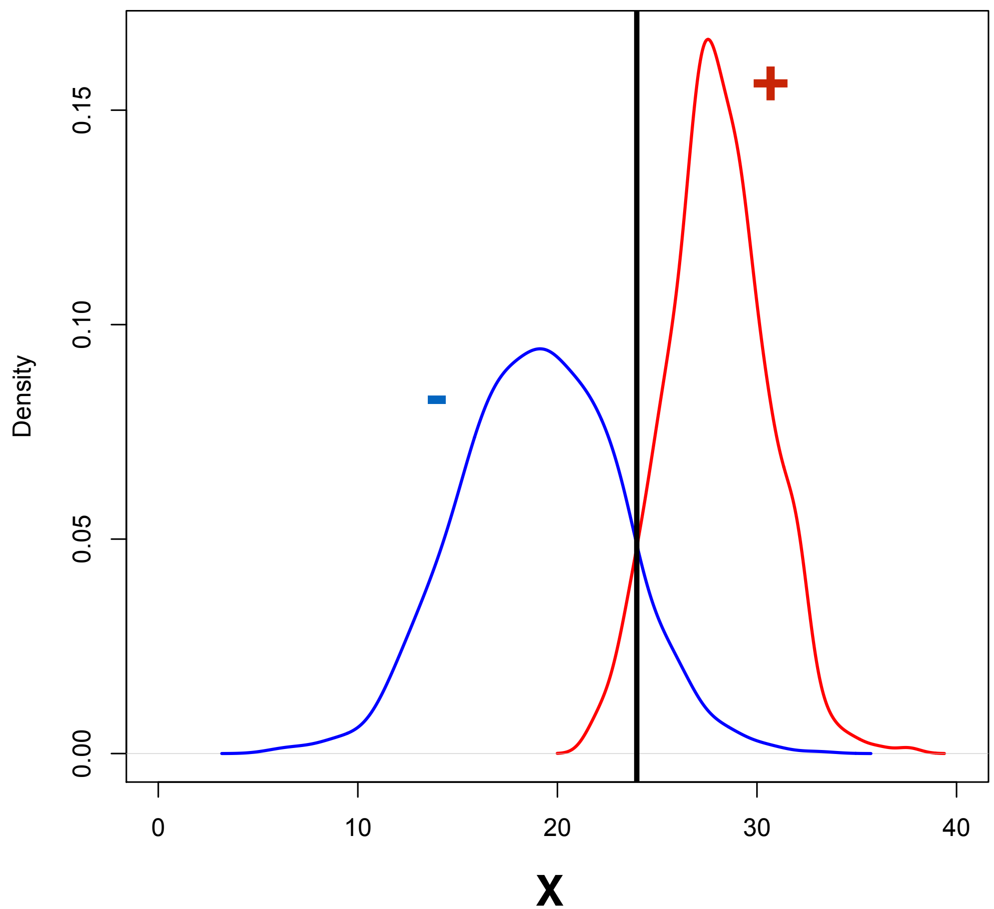
Task: Devise a rule to classify items based on the attribute ***X***

Knowledge representation:
If-then rules

Example rule:
If x > 25 then +
Else -

What is the model space?

All possible thresholds



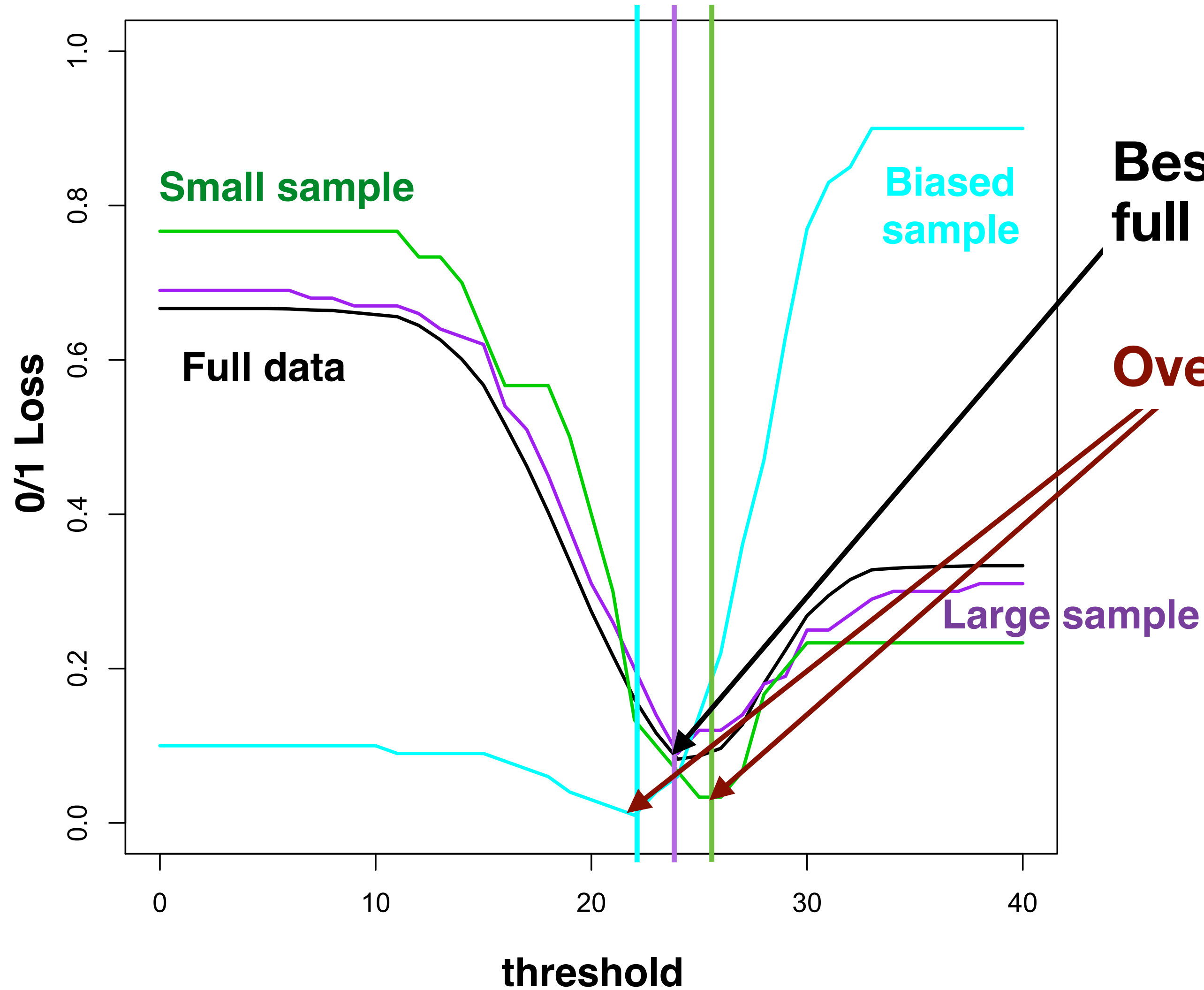
What score function?

Prediction error rate

Score function over model space

Search procedure?

Try all thresholds, select one with lowest score



Best result on full data

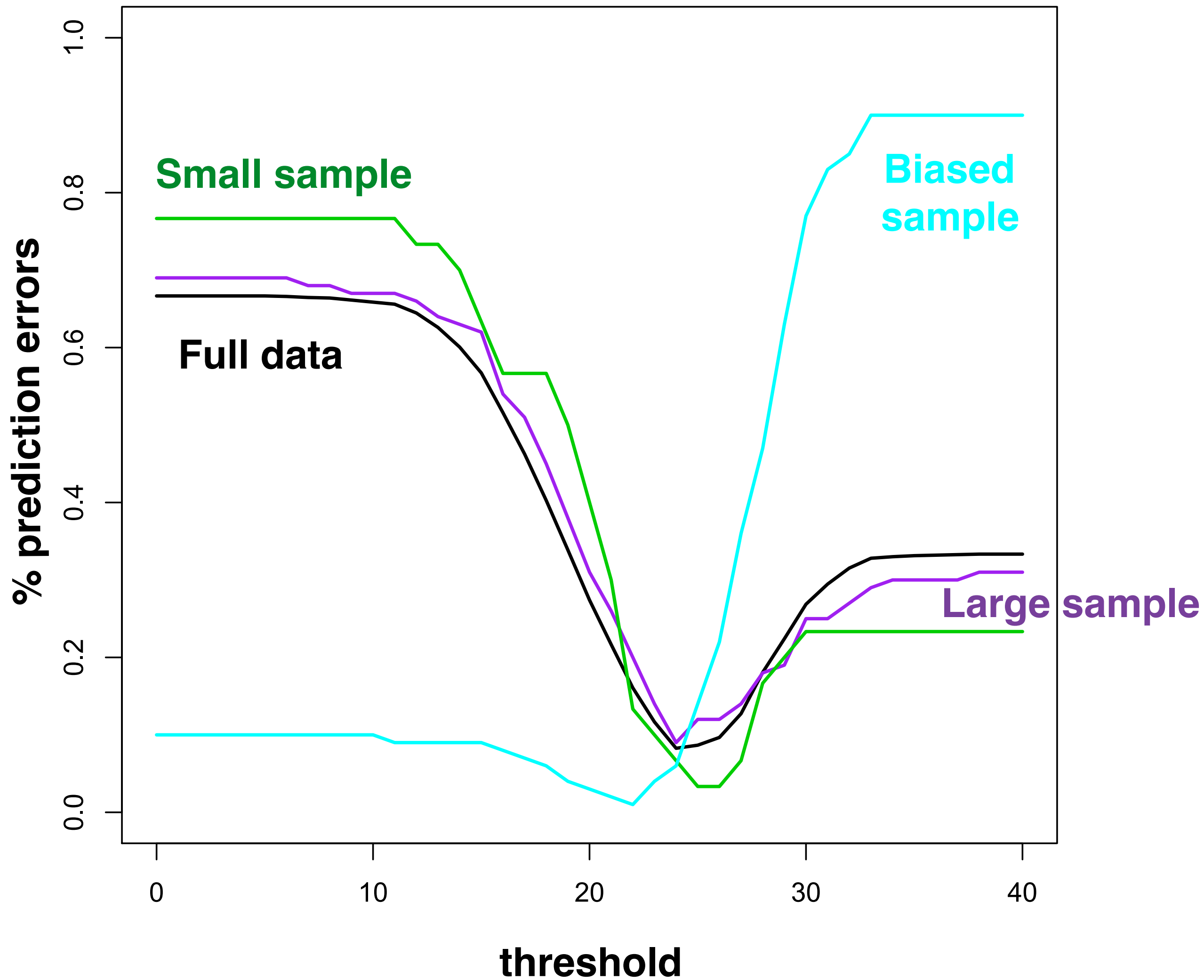
Overfitting

Approaches to avoid overfitting

- Regularization (e.g., smoothing probability estimates)
  - e.g., Laplace correction in NBC
- Hold out evaluation set, used to adjust structure of learned model
  - e.g., pruning in decision trees
- Statistical tests during learning to only include structure with significant associations
  - e.g., pre-pruning in decision trees
- Penalty term in classifier scoring function
  - i.e., change score function to prefer simpler models

How to interpret overfitting avoidance methods?

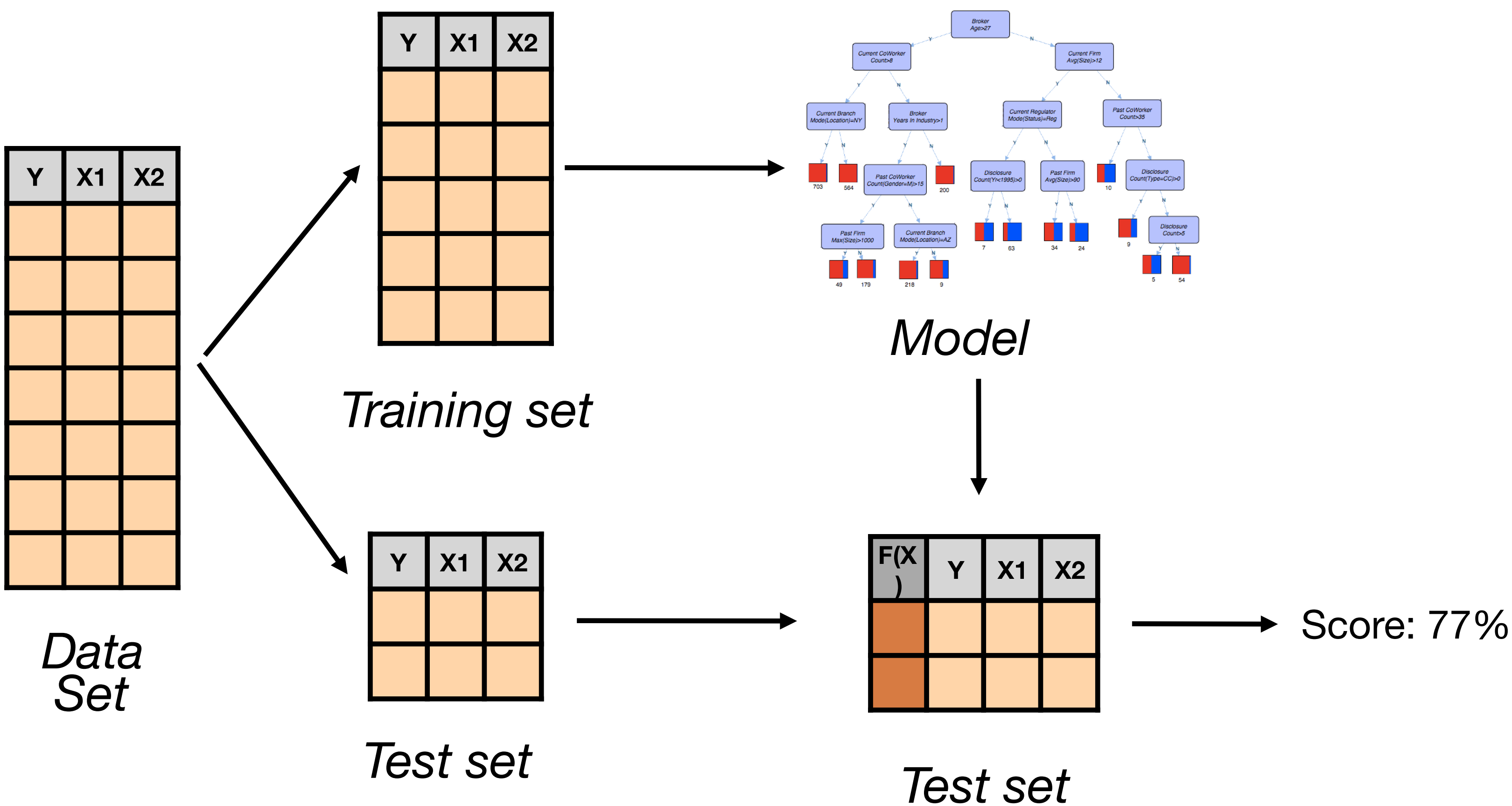
Modification of score function… to better represent model value



Evaluating classifiers

- Approach 2
    - Classify **disjoint** test set to estimate generalization rate

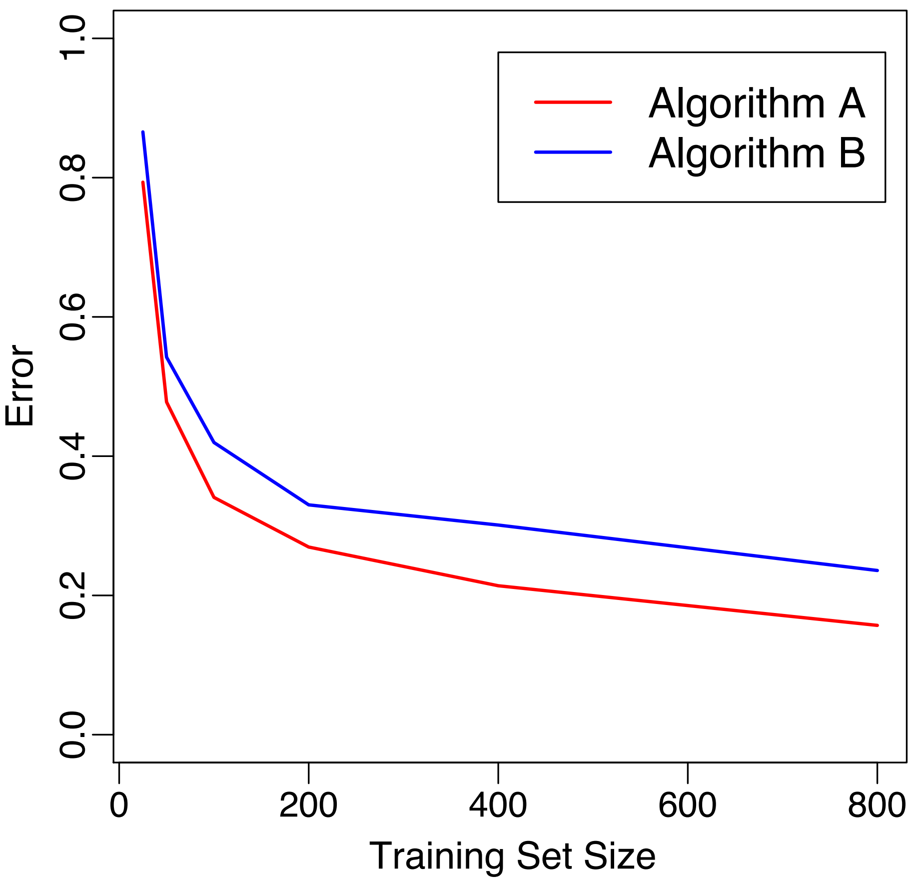
Approach 2



Estimate will vary due to size and makeup of test set

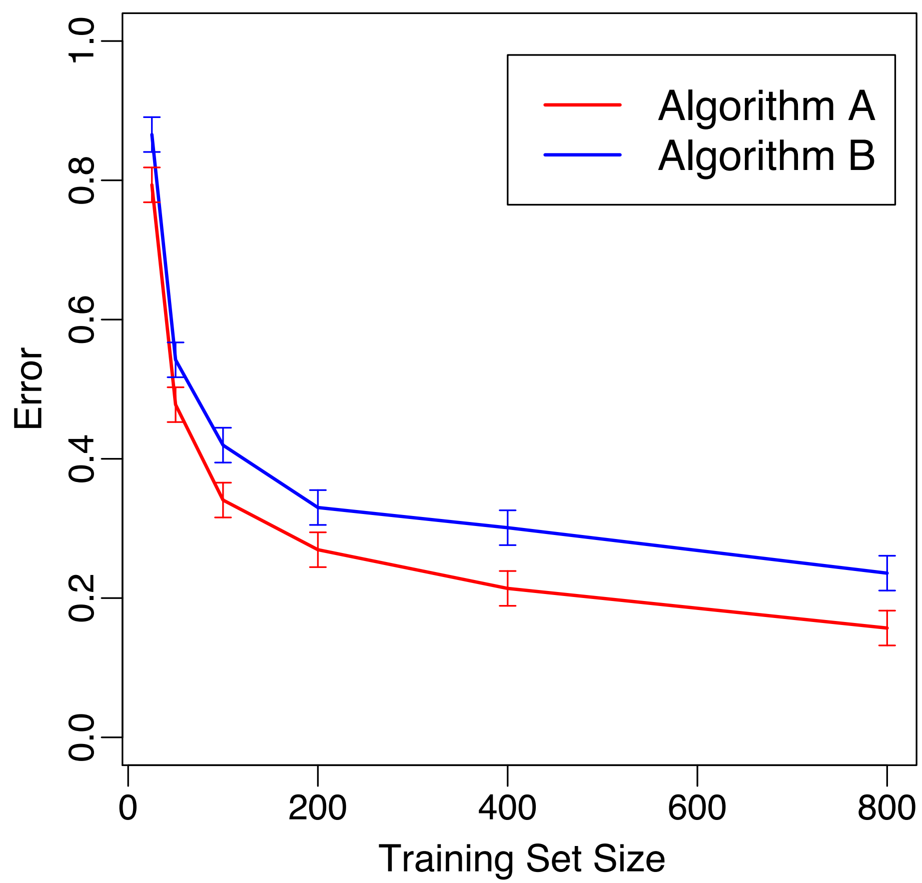
Evaluating classifiers (cont)

- Approach 2_A:
  - Partition D_0 into two disjoint subsets, learn model on one subset, measure error on the other subset
  - **Problem**: this is a point estimate of the error on one subset



Evaluating classifiers (cont)

- Approach 2_B:
  - Repeat 2_A k times (randomly partitioning each time)
  - Average error rates over the k trials
  - Plot average error and standard error bars
- *Any problem with this approach?*

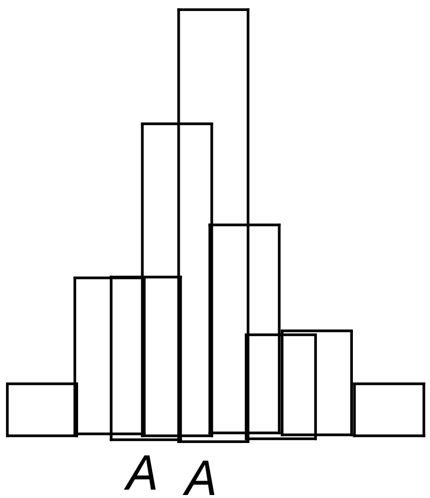


Overlapping test sets are dependent

- Repeated sampling of test sets leads to overlap (i.e., dependence) among test sets... this will results in underestimation of variance

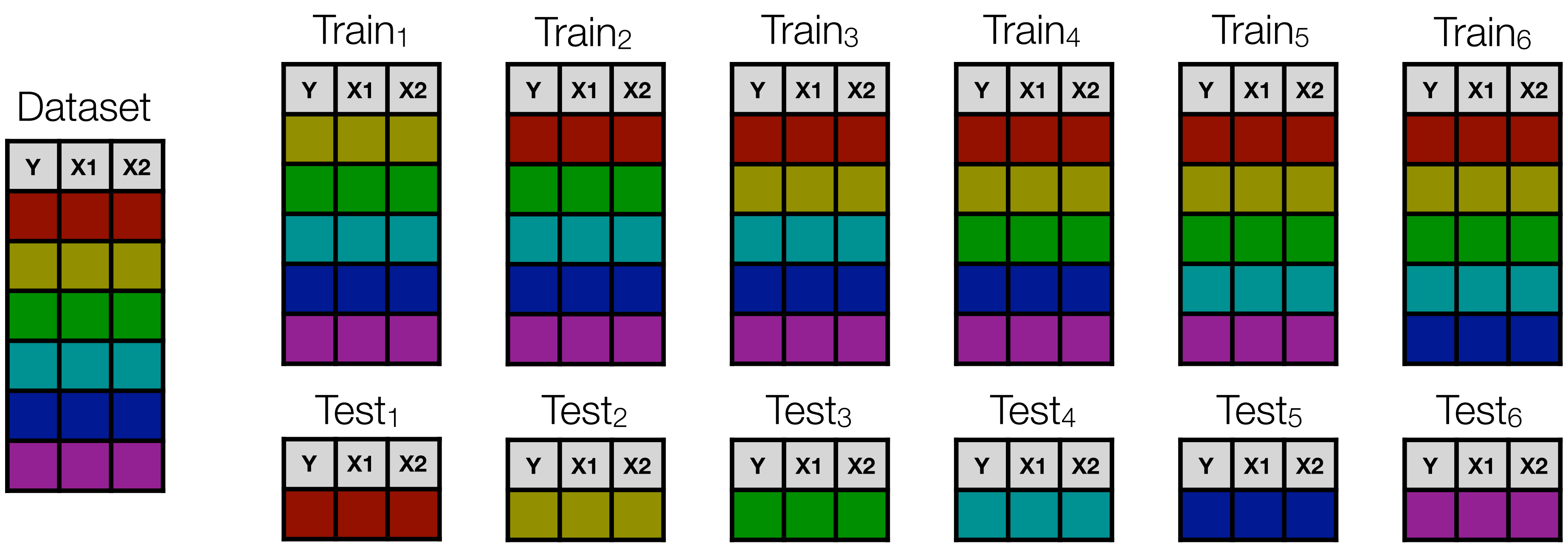
- Standard errors will be **biased** if performance is estimated from **overlapping** test sets *(Dietterich'98)*

- Recommendation:
  Use **cross-validation** to eliminate dependencies between test sets



K-fold cross validation

- Randomly **partition** training data into k folds
- For i=1 to k
  - Learn model on D - i^{th} fold; evaluate model on i^{th} fold
- Average results from all k trials

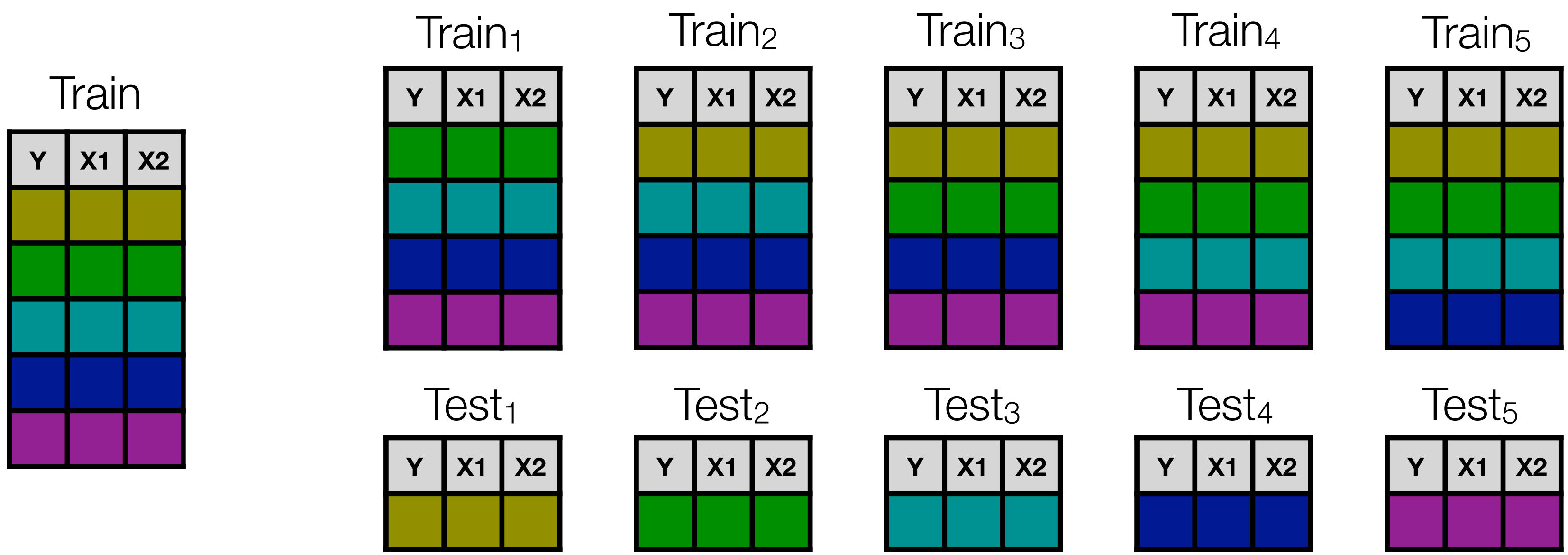


Places to use cross-validation

- Parameter setting
  - Decision tree example: Choose threshold for split function with cv
    - Repeatedly learn model with different thresholds
    - Pick threshold that shows best cross-validation performance
- Model evaluation
  - Estimate model performance across k-fold cv trials
  - Use performance measurement as empirical sampling distribution for model performance
  - Evaluate difference between algorithms with statistical test

Returning to CART decision tree pruning

Choosing a Gini threshold with CV



- For i in 1.. 5
  - For t in threshold set (e.g, [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8])
    - Learn decision tree on Train_i with Gini gain threshold t (i.e. stop growing is Gini gain is greater than t)
    - Evaluate learned tree on Test_i (e.g., with accuracy)
  - Pick t_{max} with the max score on the test set
- Learn tree on Train with avg(t_{max}) as Gini gain threshold

Data mining process

- Step 1: read in data, choose **data representation**
- Step 2: split into training and test sets (**data selection**)
- Step 3: create features (**data preprocessing**)
- Step 4: learn a model
    - choose naive Bayes model (**knowledge representation**)
    - **learning**: maximize likelihood with convex optimization (**search**); score with likelihood (**scoring function**)
- Step 5: apply model (**prediction**)
    - Note: zero-one loss evaluation uses a different score than learning
- Step 6: evaluate predictions (**evaluation**)

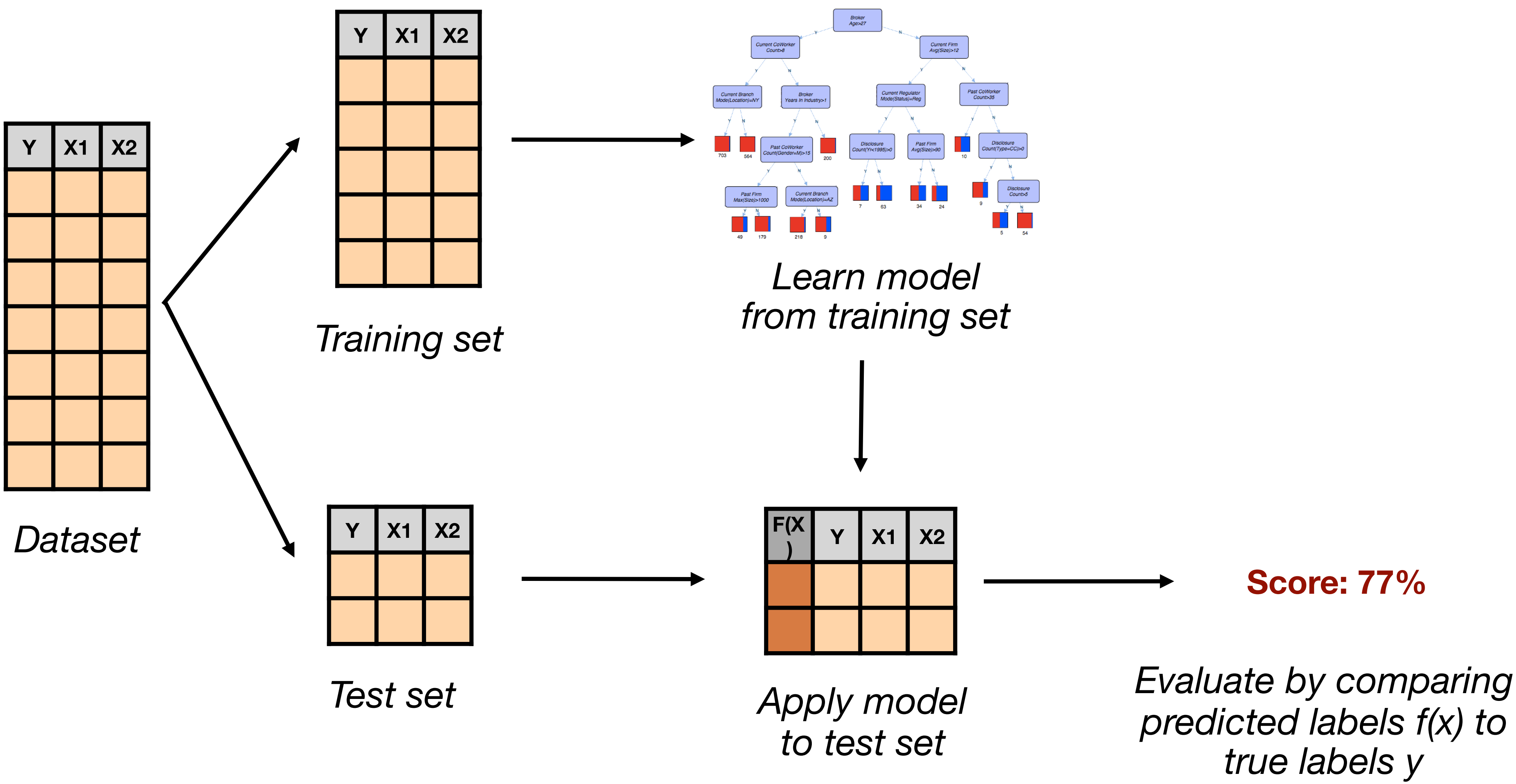
Putting it all together: Classification

Inputs and choices

- Input:
    - Dataset
    - Task
- Choices
    - Knowledge representation
    - Scoring function
    - Evaluation

- *Example:*
    - *Yelp data*
    - *Classification:*
      predict goodForGroups (Y)
      *using discrete attributes (**X**)*
    - *Naive Bayes*
    - *MLE w/smoothing*
    - *Zero-one loss,*
      square-loss

Illustration



Step 1

- Read in data
- Choose a data representation, e.g.,
  - In python the data can be represented as a list of lists (of strings):
    [['3', '?', 'alfa-romero', 'gas', 'std', 'two', 'convertible', 'rwd', 'front', '88.60', '168.80', '64.10', '48.80', '2548', 'dohc', 'four', '130', 'mpfi', '3.47', '2.68', '9.00', '111', '5000', '21', '27', '13495'], ['3', '?', 'alfa-romero', 'gas', 'std', 'two', 'convertible', 'rwd', 'front', '88.60', '168.80', '64.10', '48.80', '2548', 'dohc', 'four', '130', 'mpfi', '3.47', '2.68', '9.00', '111', '5000', '21', '27', '16500'], ...]
  - Or you can use separate structures to store the attributes and class labels (e.g., by assigning a unique id to each instance and using maps with the id as key)

Step 2

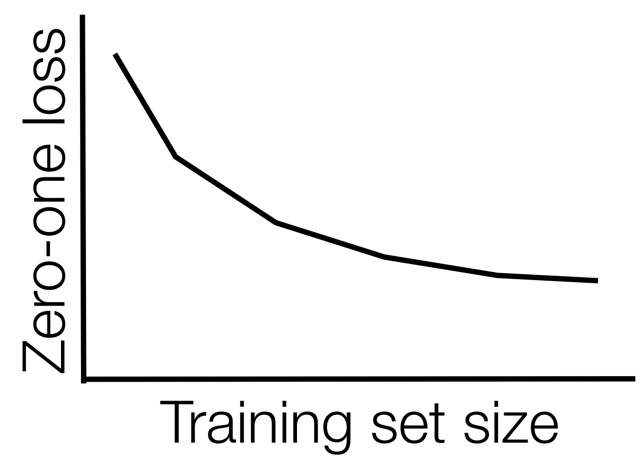
- Split into training and test sets
- There are many ways to split the data into training and test sets...
- The primary goal is to ensure that training and test examples are **disjoint**. This prevents the evaluation from being biased.
- Simple example:

```
partition1 = []
partition2 = []
partition3 = []
i = 0
for item in trainDS.getItems():
    if i<65: partition1.append(item)
    elif i<130: partition2.append(item)
    elif i<195: partition3.append(item)
    i += 1
partitions = [partition1,partition2,partition3]
```

Step 2b

- Consider repeated subsamples of the datato plot learning curves
- For each < $train_i$, $test_i$ >:
    - Learn model with $train_i$
    - Apply model to $test_i$

- You will average results over the 10 trials for each TSS to plot learning curve



Step 3

- From training data, create features (**X’**)
  - *Note: for your assignment you do not need to create features, just drop the continuous attributes, and use the discrete features as is*
- Example:
  - Let **X** be the set of 10 nominal attributes
  - For each attribute X_i with *k* possible values, construct k binary features to to add to **X’**, e.g.,
    - for X_i={red, green, blue}
      let F_1={red, ¬red}, F_2={green, ¬green}, F_3={blue, ¬blue}
      then **X’** = **X’** + {F_1, F_2, F_3}

Step 4

- Given training data, learn a model to predict Y given **X**
- Learn NBC model
  - Estimate class prior P(Y)
  - For each attribute estimate CPD $P(X_i \mid Y)$
  - Use smoothing for probability estimates

Step 5

- Given test data, apply model M to predict Y given **X**
- For each example, calculate:

$$P'(Y=1|\mathbf{X}) = \prod_i P(X_i = x_i|Y=1)P(Y=1)$$

$$P'(Y=0|\mathbf{X}) = \prod_i P(X_i = x_i|Y=0)P(Y=0)$$

$$P(Y=1|\mathbf{X}) = \frac{P'(Y=1|\mathbf{X})}{P'(Y=1|\mathbf{X}) + P'(Y=0|\mathbf{X})}$$

$$P(Y=0|\mathbf{X}) = 1 - P(Y=1|\mathbf{X})$$

- Predict class with max probability, i.e., if P(Y=1|X) > P(Y=0|X) then predict Y=1

Step 6

- Given a set of predictions for test data, evaluate the model by comparing the predicted values to the true values
  - Zero-one loss measures the mismatches between predicted and true class label:

$$Loss_{0/1}(T) = \frac{1}{n} \sum_{i \in n} \left\{ \begin{array}{ll} 0 & \text{if } y(i) = \hat{y}(i) \\ 1 & \text{otherwise} \end{array} \right\}$$

  - Squared loss measures the quality of the probability estimates. Let p_i refer to the probability that the NBC assigns to example *i*'s true class value, then:

$$Loss_{sq}(T) = \frac{1}{n} \sum_{i \in n} (1 - p_i)^2$$