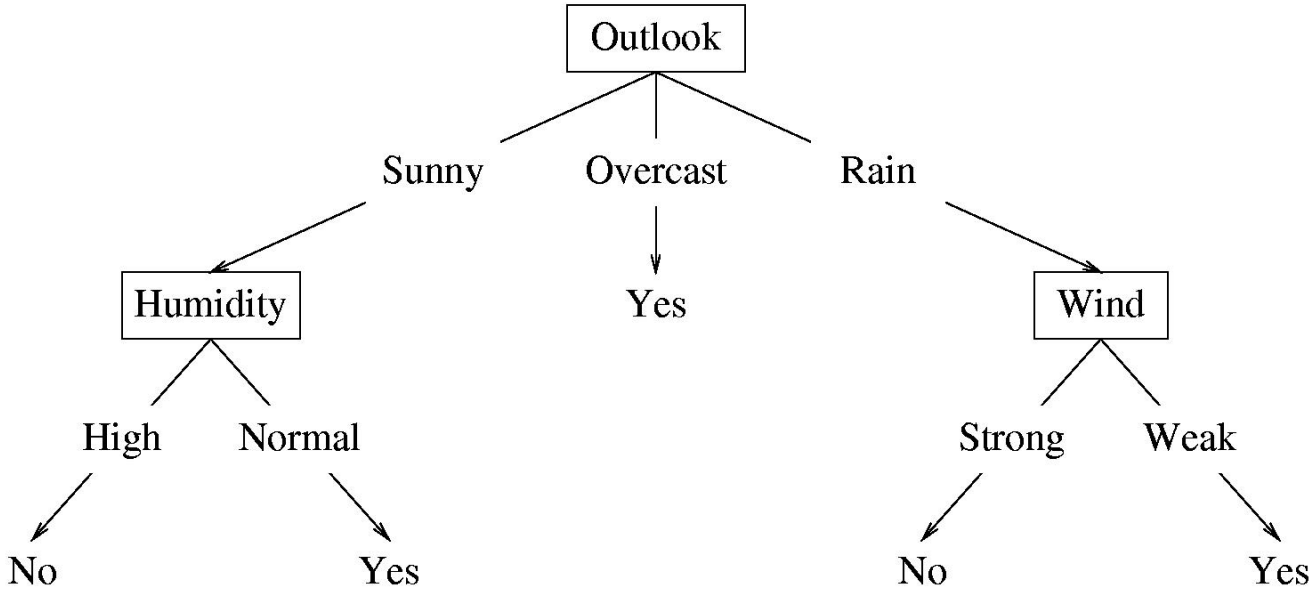
Decision Trees II & Overfitting

Machine Learning (AIM 5002-41)

Joon Hee Choi
Sungkyunkwan University

Summary of Decision Trees (So far)



- Decision tree induction → choose the best attribute
  - Choose split via information gain
  - Build tree greedily, recursing on children of split
  - Stop when we achieve homogeny
    - i.e., when all instance in a child have the same class

Summary of Decision Trees (So far)

- Information Gain: Mutual information of attribute A and the class variable of data set X

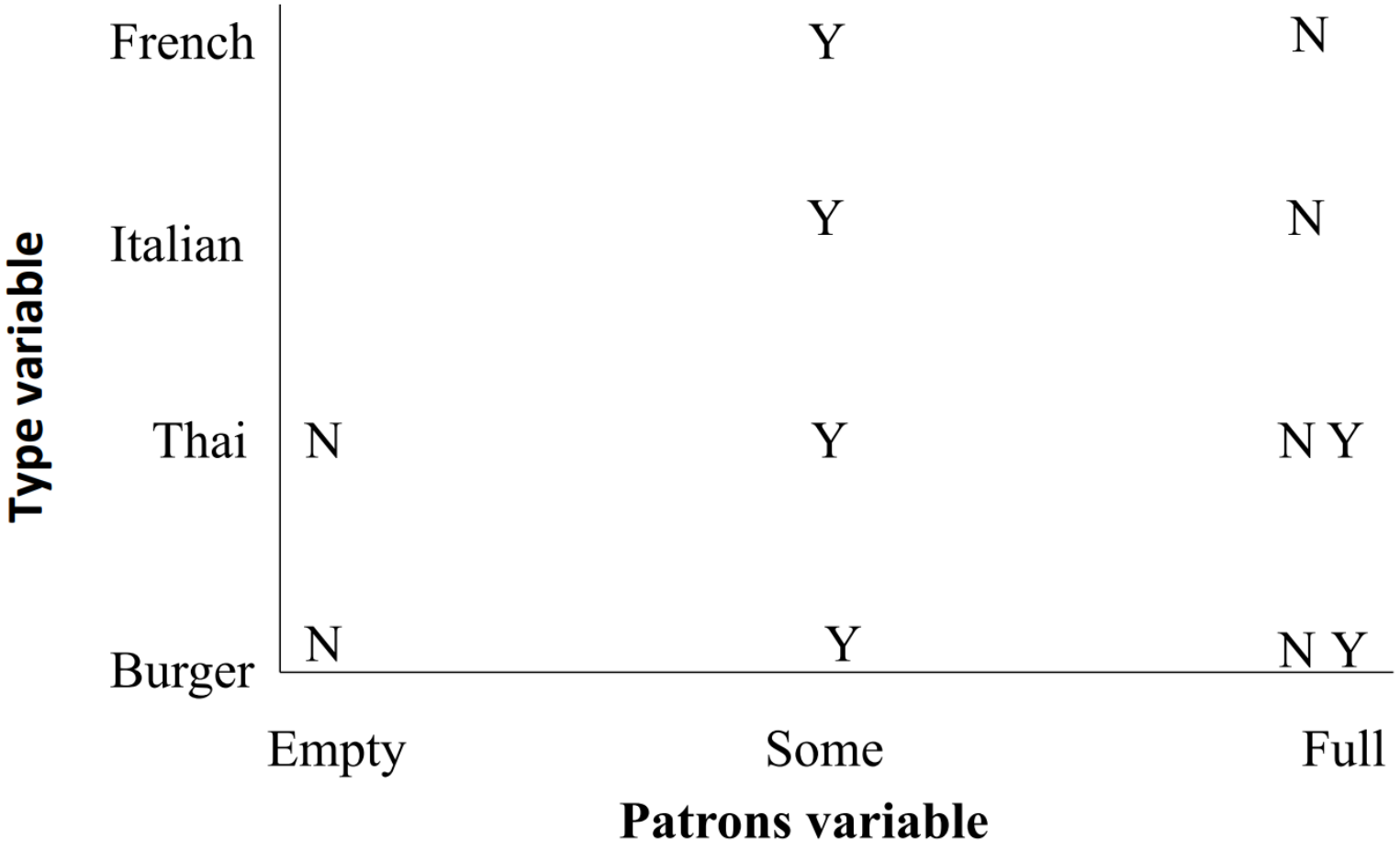
$$InfoGain(X, A) = H(X) - H(X \mid A)$$
$$= H(X) - \sum_{v \in values(A)} \underbrace{\frac{|\{x \in X \mid x_A = v\}|}{|X|}}_{\text{fraction of instances with value } v \text{ in attribute } A} \times \underbrace{H(\{x \in X \mid x_A = v\})}_{\text{entropy of those instances}}$$

- Entropy

$$H(X) = -\sum_{c \in Classes} \underbrace{\frac{|\{x \in X \mid class(x) = c\}|}{|X|}}_{\text{fraction of instances of class } c} \log_2 \frac{|\{x \in X \mid class(x) = c\}|}{|X|}$$

Restaurant Example

Random: Patrons or Wait-time; **Least-values**: Patrons; **Most-values**: Type; **Max-gain**: ???

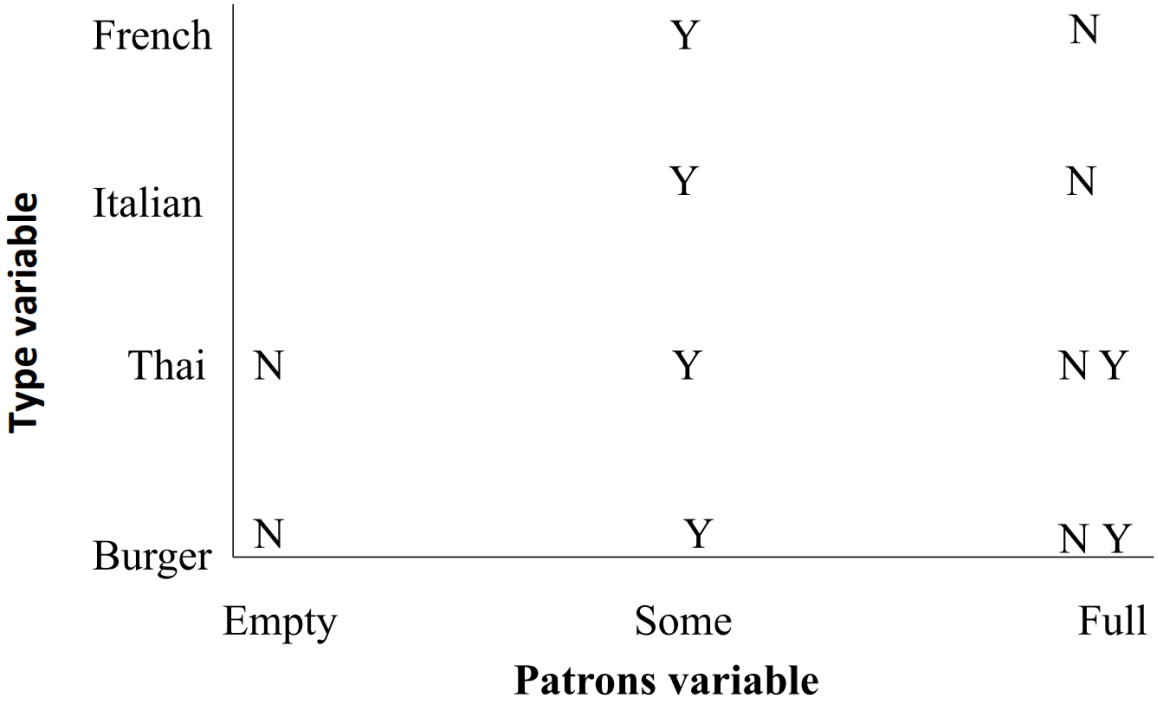


Computing information gain

$\mathbf{I(X)} = ?$

$\mathbf{I(Pat, X)} = ?$

$\mathbf{I(Type, X)} = ?$



$\text{Gain}(\text{Pat}, \text{X}) = ?$

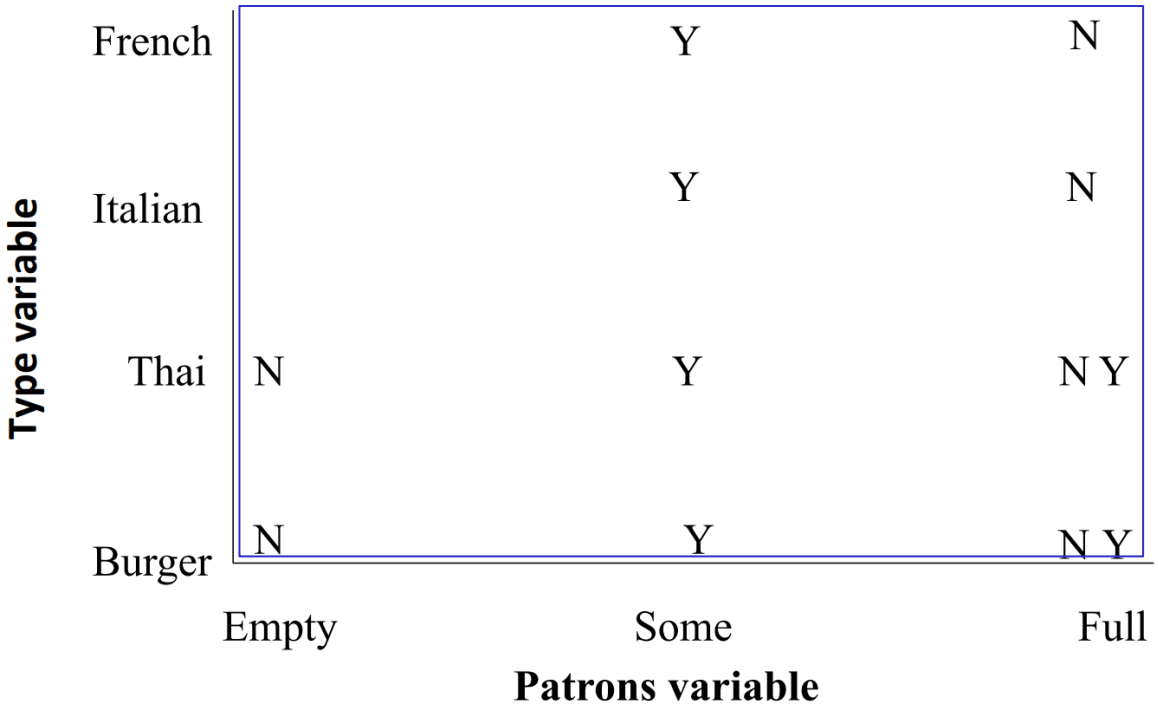
$\text{Gain}(\text{Type}, \text{X}) = ?$

Computing information gain

$\mathbf{I(X)}$
$= -(.5 \log .5 + .5 \log .5)$
$= .5 + .5 = 1$

$\mathbf{I(Pat, X)} = ?$

$\mathbf{I(Type, X)} = ?$



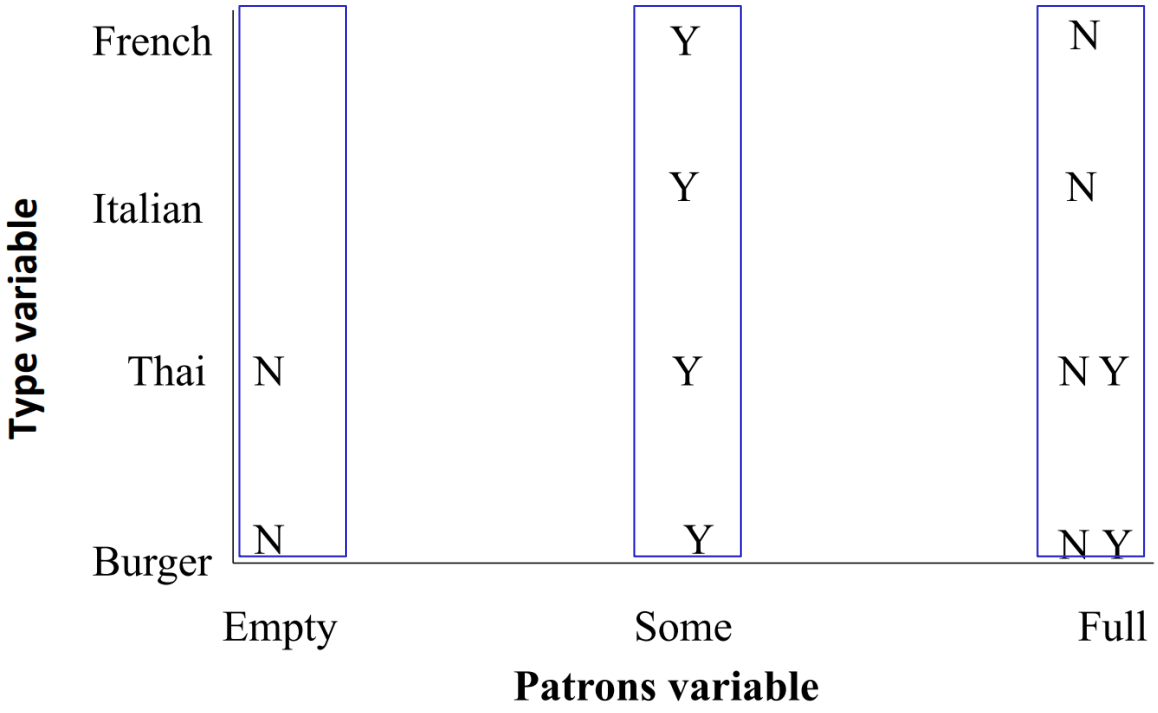
$\text{Gain}(\text{Pat}, X) = ?$
$\text{Gain}(\text{Type}, X) = ?$

Computing information gain

$$\mathbf{I(X)} = -(.5\log.5 + .5\log.5) = .5 + .5 = 1$$

$$\mathbf{I(Pat, X)} = \frac{2}{12}(0) + \frac{4}{12}(0) + \frac{6}{12}\left(-\left(\frac{4}{6}\log\frac{4}{6} + \frac{2}{6}\log\frac{2}{6}\right)\right) = \frac{1}{2}\left(\frac{2}{3} * .6 + \frac{1}{3} * 1.6\right) = .47$$

$\mathbf{I(Type, X)} = ?$



$\text{Gain}(\text{Pat}, X) = ?$

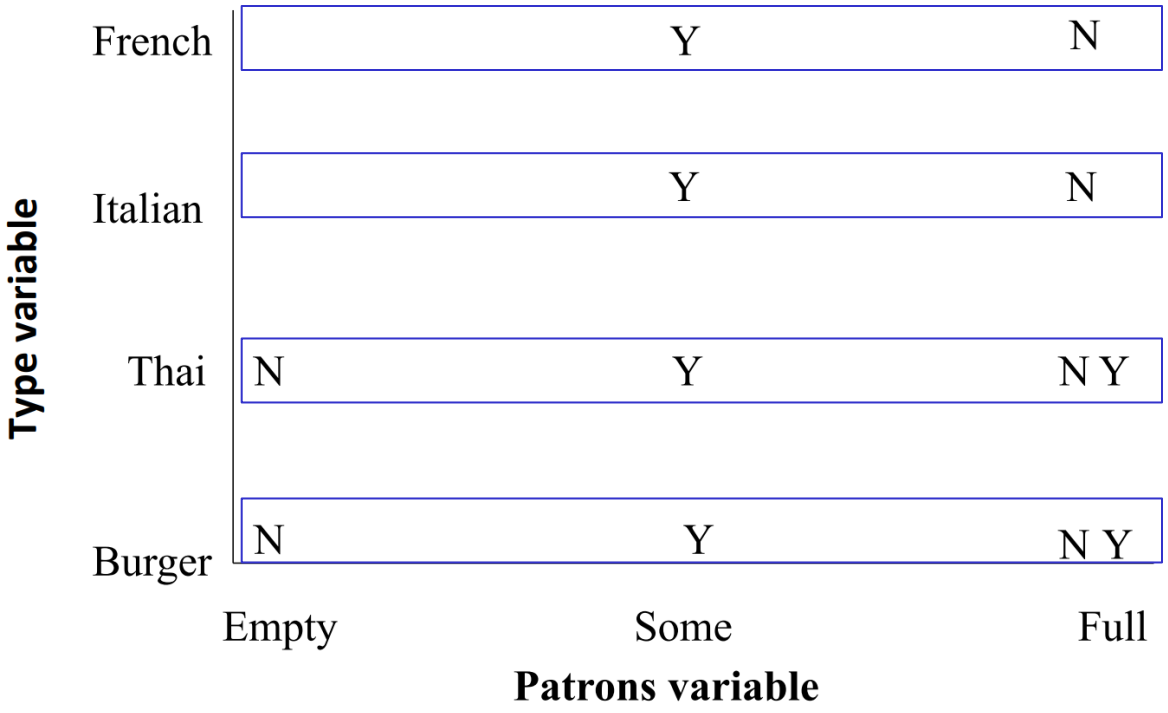
$\text{Gain}(\text{Type}, X) = ?$

Computing information gain

$$\mathbf{I(X)}$$
$$= -(.5 \log .5 + .5 \log .5)$$
$$= .5 + .5 = 1$$

$$\mathbf{I(Pat, X)}$$
$$= \frac{2}{12}(0) + \frac{4}{12}(0) + \frac{6}{12}\left(-\left(\frac{4}{6}\log\frac{4}{6} + \frac{2}{6}\log\frac{2}{6}\right)\right)$$
$$= \frac{1}{2}\left(\frac{2}{3} * .6 + \frac{1}{3} * 1.6\right) = .47$$

$$\mathbf{I(Type, X)}$$
$$= \frac{2}{12}(1) + \frac{2}{12}(1) + \frac{4}{12}(1) + \frac{4}{12}(1) = 1$$



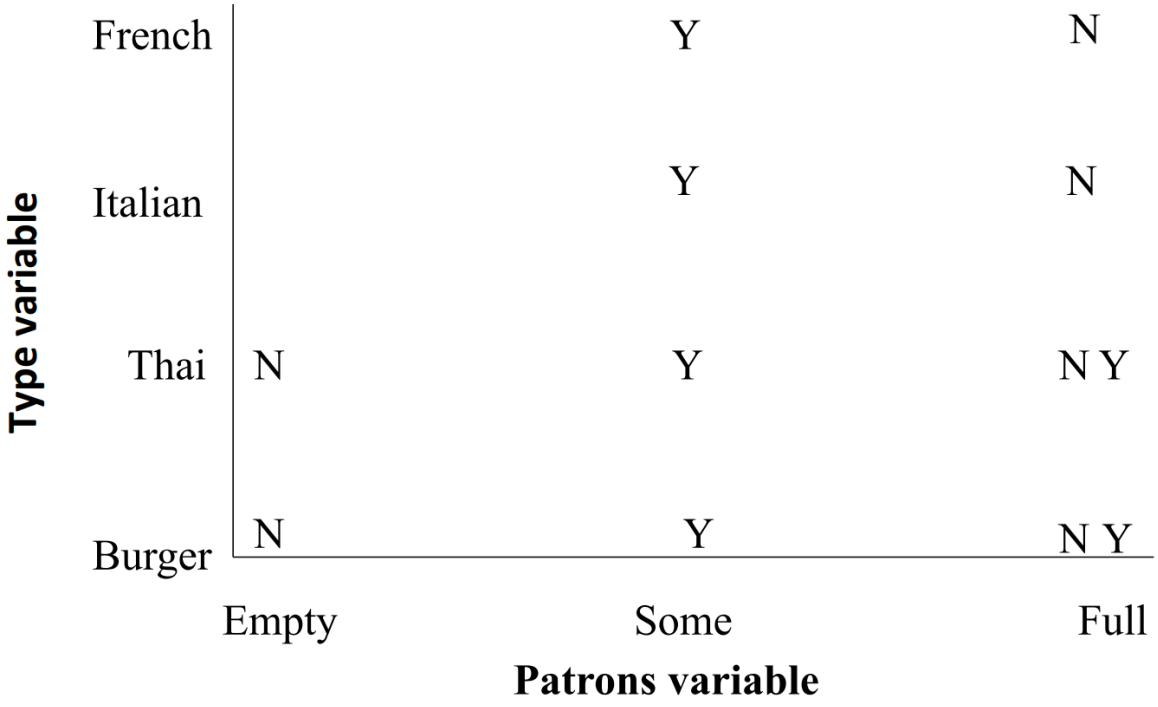
Gain(Pat, X) =?
Gain(Type, X) =?

Computing information gain

$$\mathbf{I(X)} = -(.5\log.5 + .5\log.5) = .5 + .5 = 1$$

$$\mathbf{I(Pat, X)} = \frac{2}{12}(0) + \frac{4}{12}(0) + \frac{6}{12}\left(-\left(\frac{4}{6}\log\frac{4}{6} + \frac{2}{6}\log\frac{2}{6}\right)\right) = \frac{1}{2}\left(\frac{2}{3} * .6 + \frac{1}{3} * 1.6\right) = .47$$

$$\mathbf{I(Type, X)} = \frac{2}{12}(1) + \frac{2}{12}(1) + \frac{4}{12}(1) + \frac{4}{12}(1) = 1$$



$$\text{Gain}(\text{Pat}, X) = 1 - .47 = .53$$

$$\text{Gain}(\text{Type}, X) = 1 - 1 = 0$$

Attributes with Many Values

- Problem
  - If attribute has many values, InfoGain() will select it.
  - e.g., imagine using $\mathbf{date = Jan\_28\_2011}$ as an attribute

- Alternative approach: use GainRatio() instead

$$GainRatio(X, A) = \frac{InfoGain(X, A)}{SplitInformation(X, A)}$$

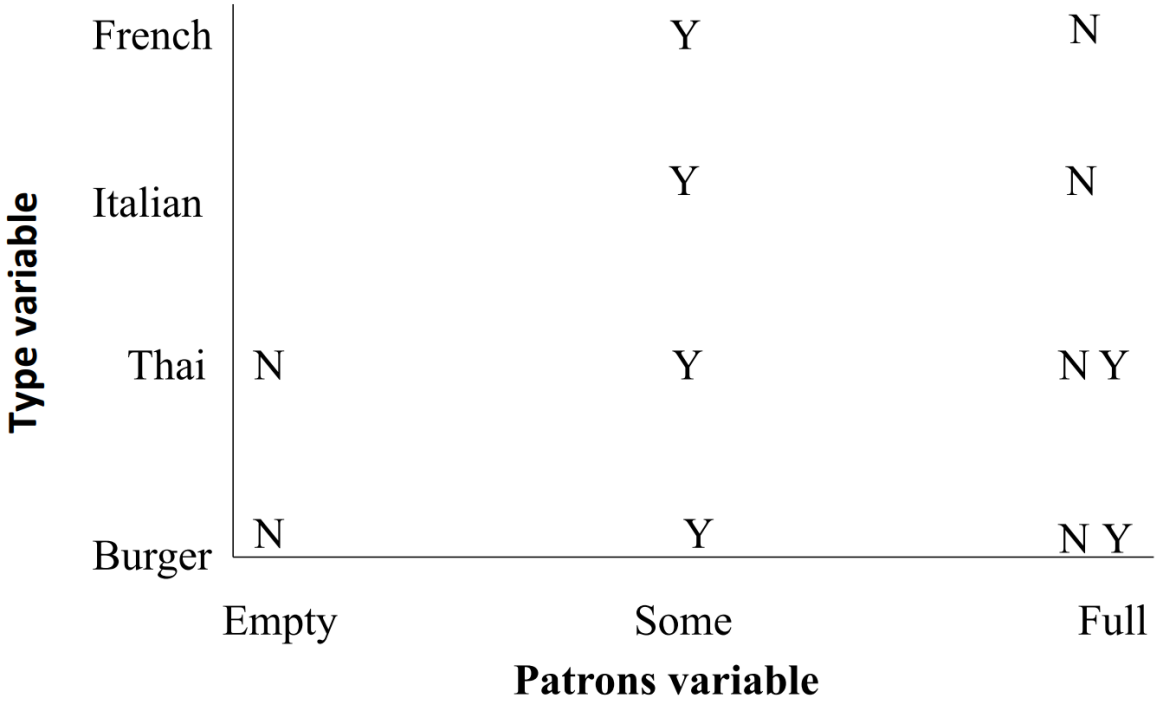
$$SplitInformation(X, A) = -\sum_{v \in values(A)} \frac{|X_v|}{|X|} \log_2 \frac{|X_v|}{|X|}$$

where X_v is a subset of X for which A has value v

Computing Gain Ratio

Already computed:

- $\mathrm{I}(\mathrm{X}) = 1$
- $\mathrm{I}(\mathrm{Pat}, \mathrm{X}) = 0.47$
- $\mathrm{I}(\mathrm{Type}, \mathrm{X}) = 1$
- $\mathrm{Gain}(\mathrm{Pat}, \mathrm{X}) = 0.53$
- $\mathrm{Gain}(\mathrm{Type}, \mathrm{X}) = 0$



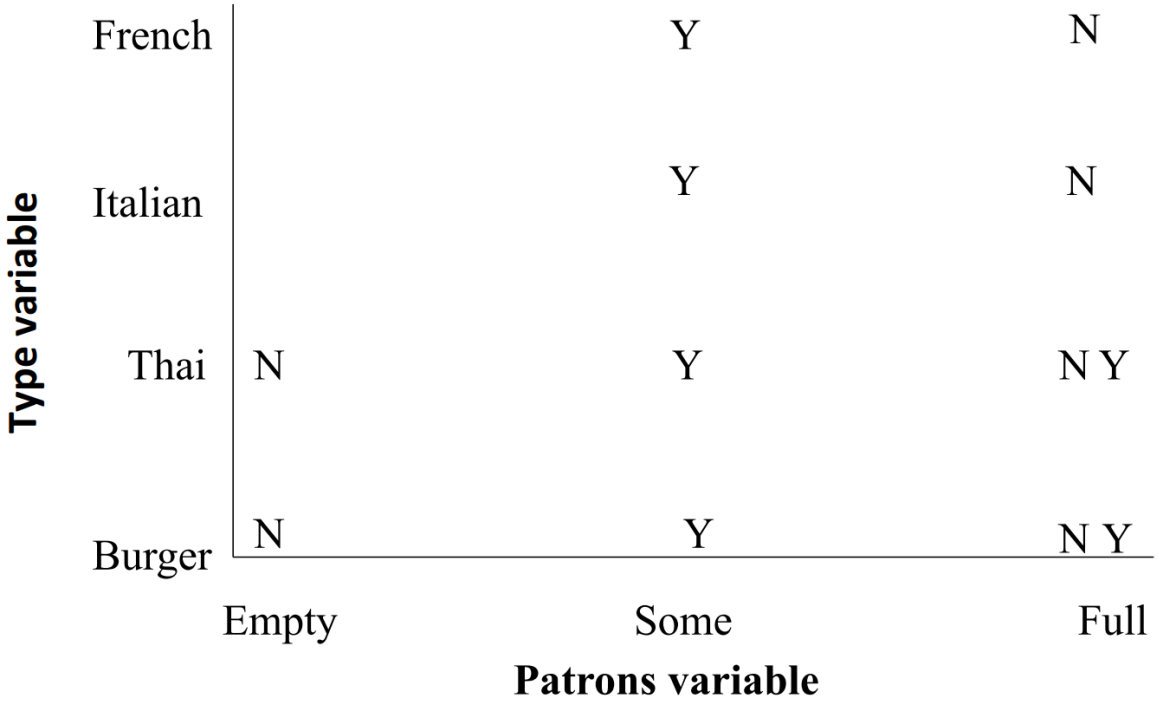
$$\mathrm{SplitInfo}(\mathrm{Pat}, \mathrm{X}) = -\left(\frac{1}{6}\log\frac{1}{6} + \frac{1}{3}\log\frac{1}{3} + \frac{1}{2}\log\frac{1}{2}\right) = 1.47$$

$$\mathrm{SplitInfo}(\mathrm{Type}, \mathrm{X}) = -\left(\frac{1}{6}\log\frac{1}{6} + \frac{1}{6}\log\frac{1}{6} + \frac{1}{3}\log\frac{1}{3} + \frac{1}{3}\log\frac{1}{3}\right) = 1.93$$

Computing Gain Ratio

Already computed:

- $\mathrm{I}(\mathrm{X}) = 1$
- $\mathrm{I}(\mathrm{Pat}, \mathrm{X}) = 0.47$
- $\mathrm{I}(\mathrm{Type}, \mathrm{X}) = 1$
- $\mathrm{Gain}(\mathrm{Pat}, \mathrm{X}) = 0.53$
- $\mathrm{Gain}(\mathrm{Type}, \mathrm{X}) = 0$



$$\mathrm{SplitInfo}(\mathrm{Pat}, \mathrm{X}) = -\left(\frac{1}{6}\log\frac{1}{6} + \frac{1}{3}\log\frac{1}{3} + \frac{1}{2}\log\frac{1}{2}\right) = 1.47$$

$$\mathrm{SplitInfo}(\mathrm{Type}, \mathrm{X}) = -\left(\frac{1}{6}\log\frac{1}{6} + \frac{1}{6}\log\frac{1}{6} + \frac{1}{3}\log\frac{1}{3} + \frac{1}{3}\log\frac{1}{3}\right) = 1.93$$

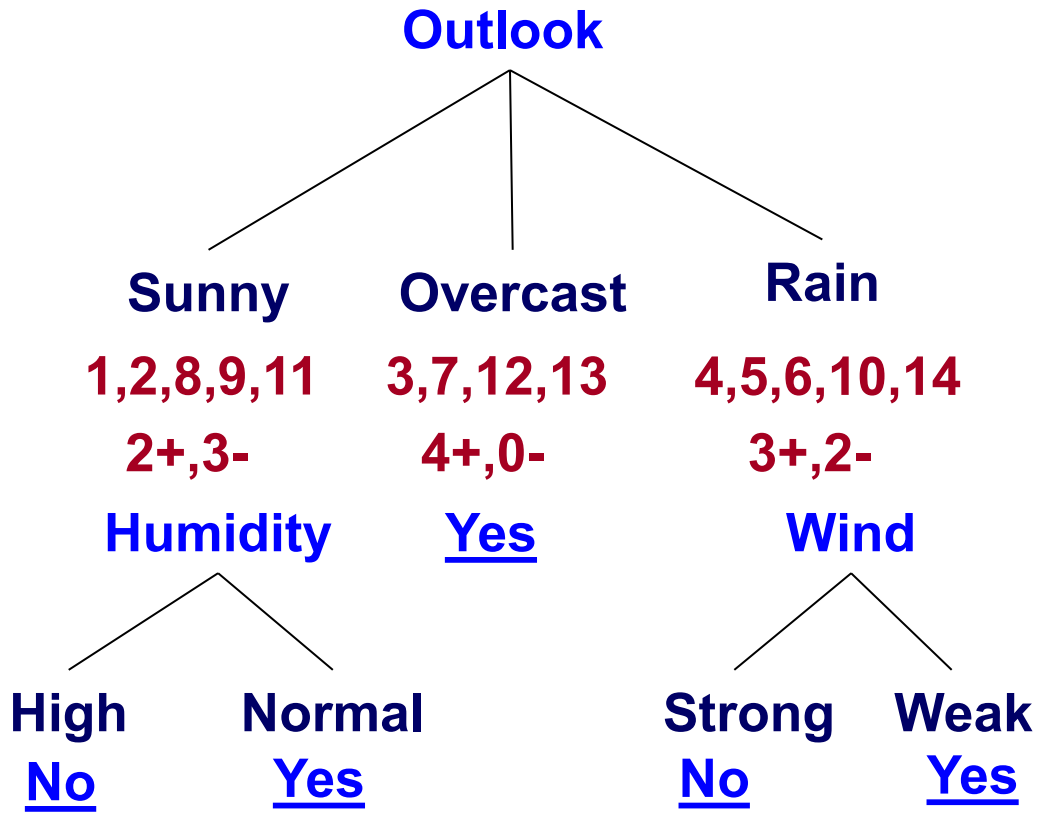
$$\mathbf{GainRatio(Pat, X)} = \frac{\mathbf{Gain\ (Pat,}\boldsymbol{X}\mathbf{)}}{\mathbf{SplitInfo(Pat,}\boldsymbol{X}\mathbf{)}} = \frac{\mathbf{0.53}}{\mathbf{1.47}} = \mathbf{0.36}$$

$$\mathrm{GainRatio}(\mathrm{Type}, \mathrm{X}) = \frac{\mathrm{Gain}(\mathrm{Type,X})}{\mathrm{SplitInfo}(\mathrm{Type,X})} = \frac{0}{1.93} = 0$$

Overfitting

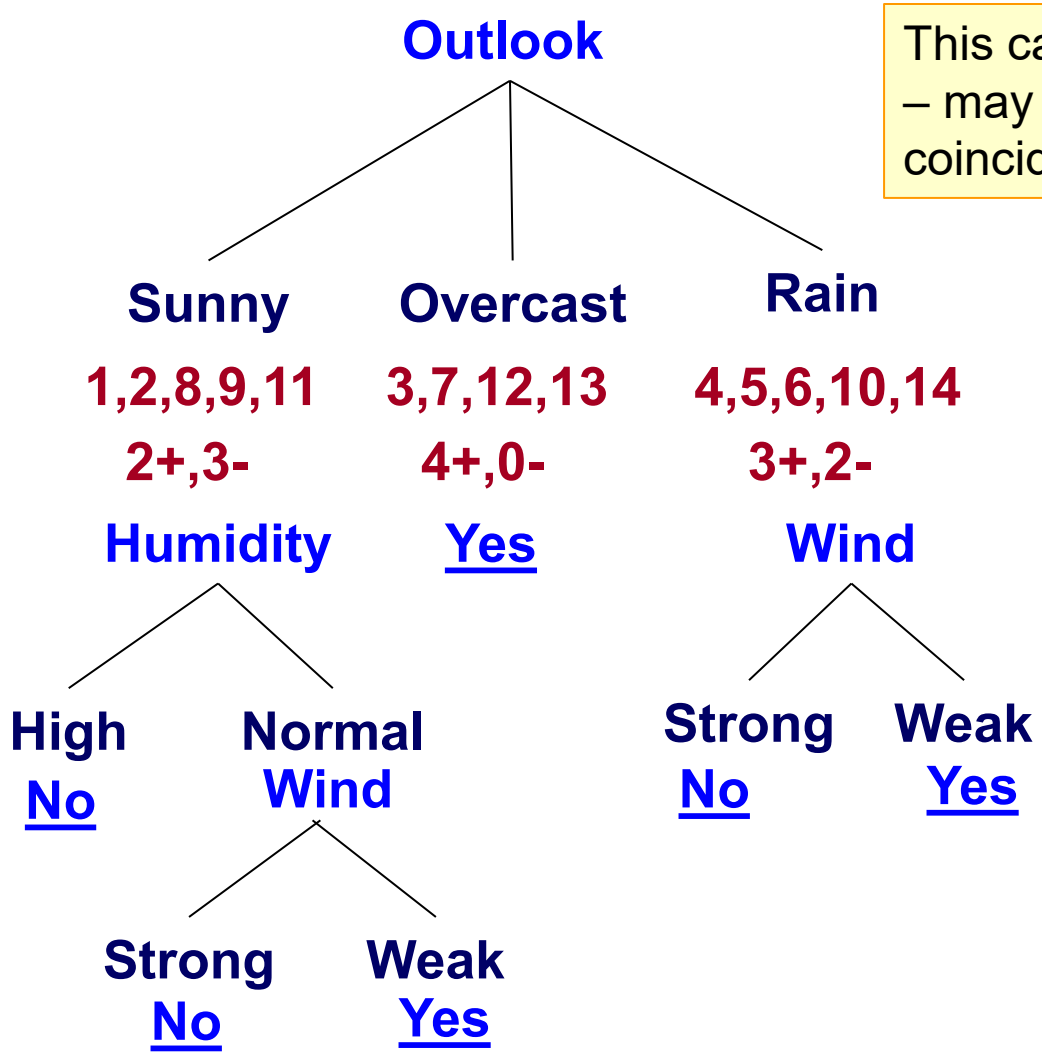
Example

- Outlook = Sunny, Temp = Hot, Humidity = Normal, Wind = Strong, NO



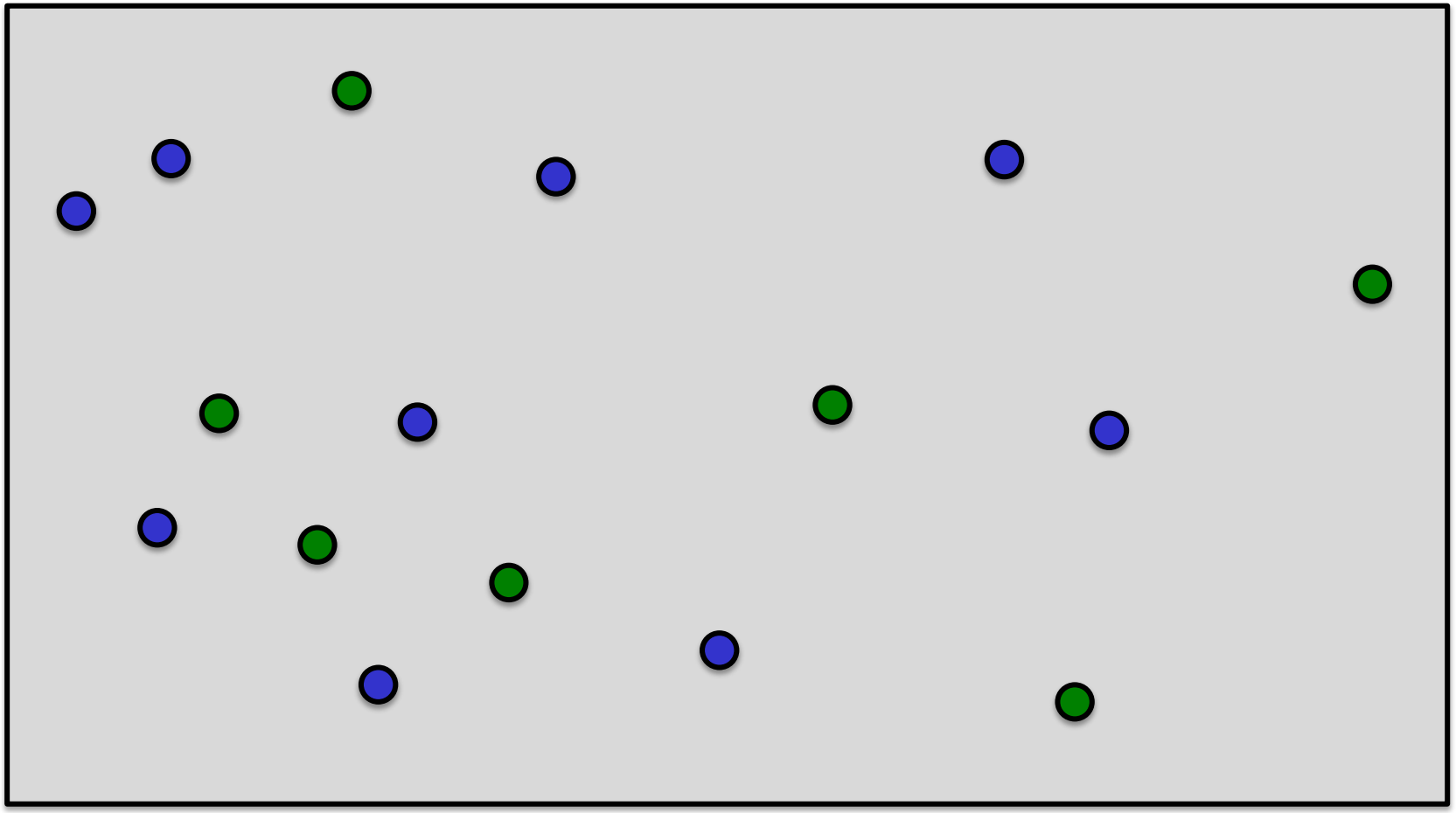
Overfitting - Example

- Outlook = Sunny, Temp = Hot, Humidity = Normal, Wind = Strong, NO

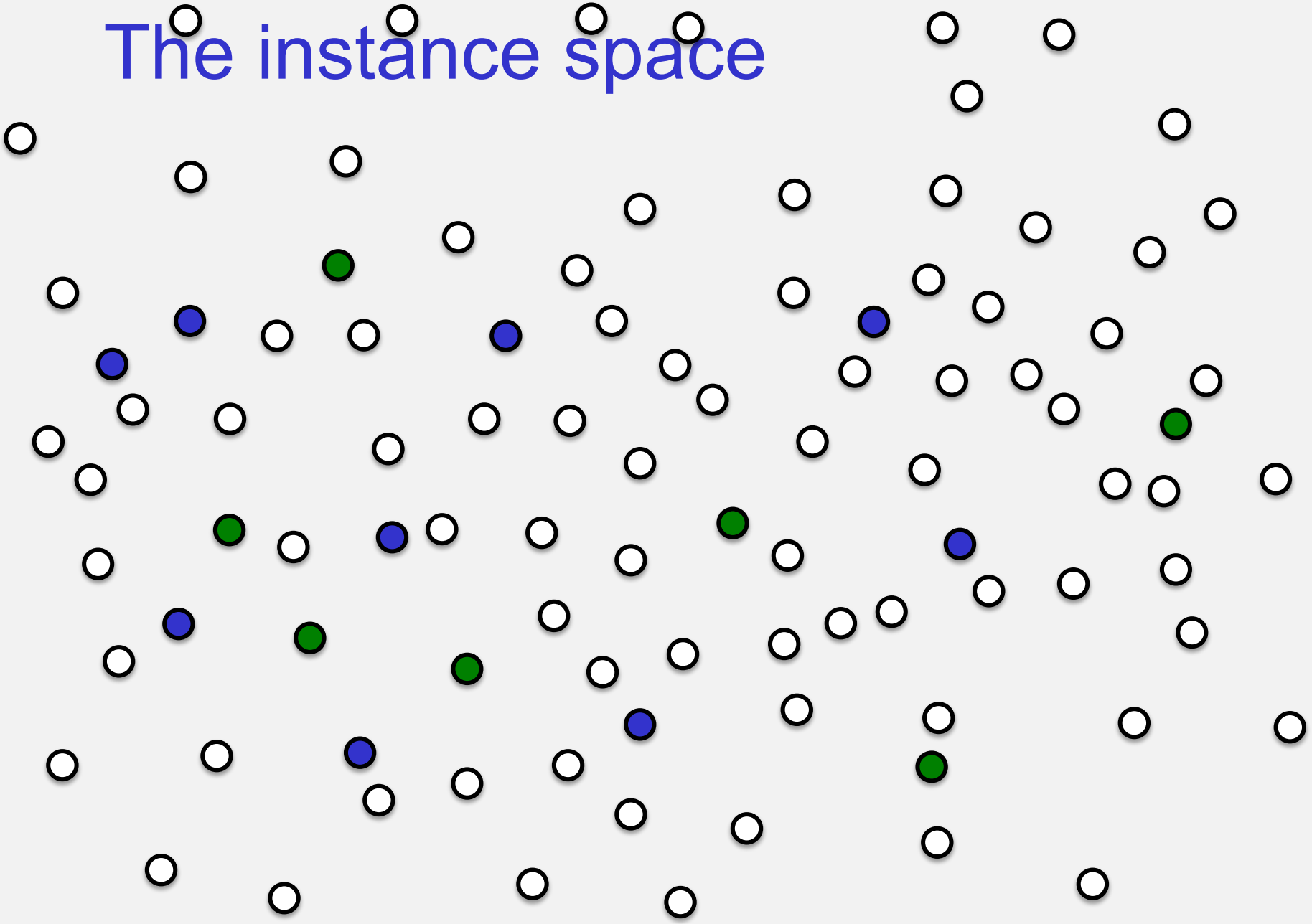


This can always be done – may fit noise or other coincidental regularities

Our training data

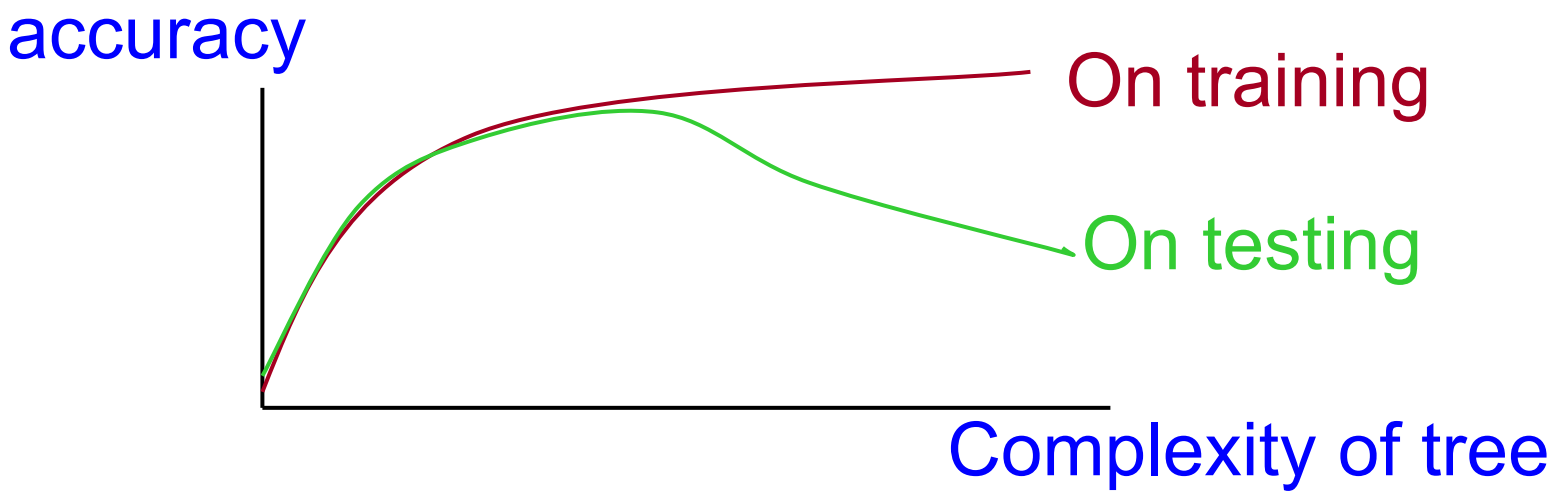


The instance space



Overfitting the Data

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization performance.
  - There may be noise in the training data the tree is fitting
  - The algorithm might be making decisions based on very little data
- A hypothesis h is said to overfit the training data if there is another hypothesis h', such that h has a smaller error than h' on the **training data** but h has larger error on the **test data** than h'.



Reasons for overfitting

- Too much variance in the training data
  - Training data is not a representative sample of the instance space
  - We split on features that are actually irrelevant
- Too much noise in the training data
  - Noise = some feature values or class labels are incorrect
  - We learn to predict the noise
- In both cases, it is a result of our will to minimize the empirical error when we learn, and the ability to do it (with DTs)

Pruning a decision tree

- Prune = remove leaves and assign majority label of the parent to all items

- Prune the children of S if:
  - all children are leaves, and
  - the accuracy on the validation set does not decrease if we assign the most frequent class label to all items at S.

Avoiding Overfitting

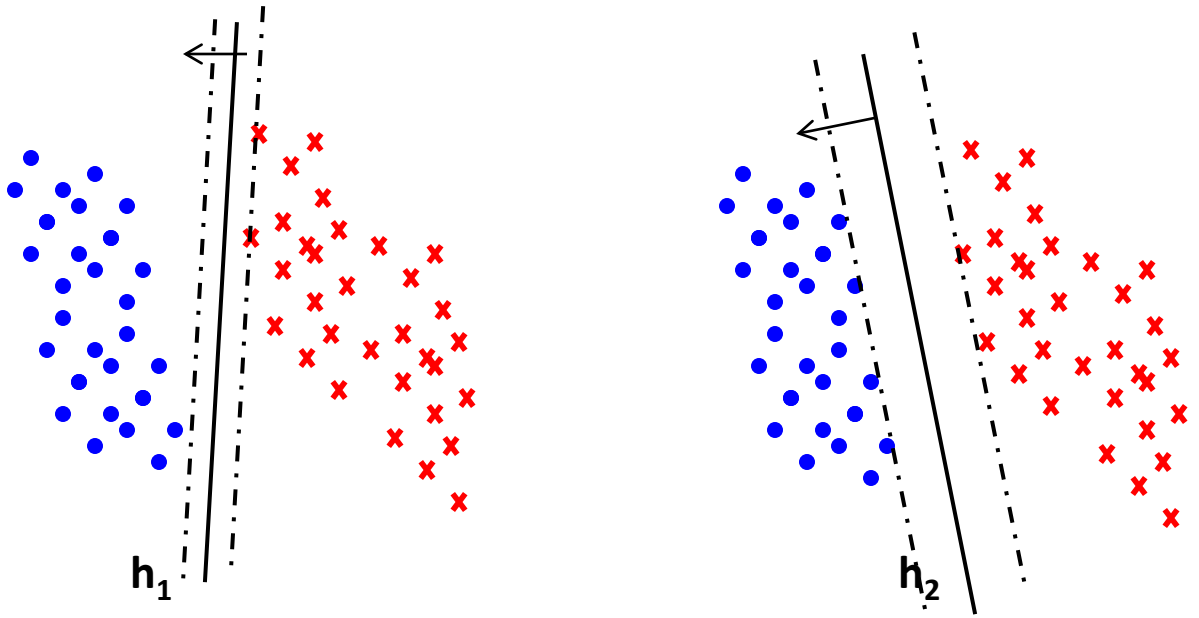
How can this be avoided with linear classifiers?

- Two basic approaches
  - **Pre-pruning**: Stop growing the tree at some point during construction when it is determined that there is not enough data to make reliable choices.
  - **Post-pruning**: Grow the full tree and then remove nodes that seem not to have sufficient evidence.
- Methods for evaluating subtrees to prune
  - **Cross-validation**: Reserve hold-out set to evaluate utility
  - **Statistical testing**: Test if the observed regularity can be dismissed as likely to occur by chance
  - **Minimum Description Length**: Is the additional complexity of the hypothesis smaller than remembering the exceptions?
- This is related to the notion of regularization that we will see in other contexts – keep the hypothesis simple.

Hand waving, for now.

Next: a brief detour into explaining generalization and overfitting

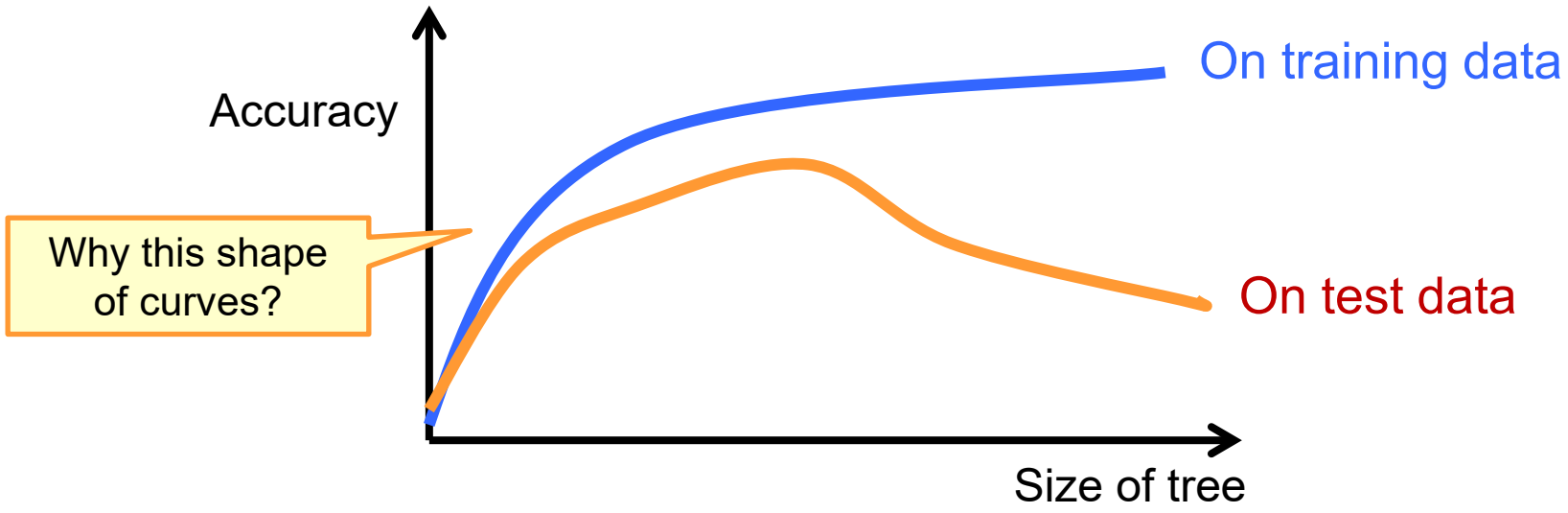
Preventing Overfitting



The i.i.d. assumption

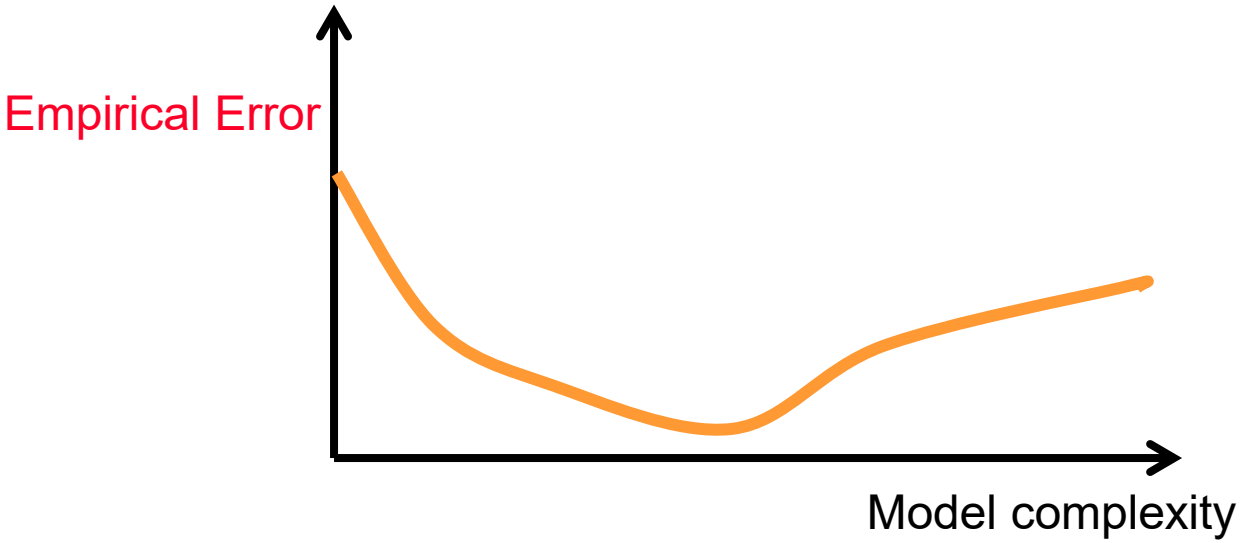
- Training and test items are independently and identically distributed (i.i.d.):
  - There is a distribution $P(\mathbf{X}, Y)$ from which the data $\mathcal{D} = \{(\mathbf{x}, y)\}$ is generated.
    - Sometimes it's useful to rewrite $P(\mathbf{X}, Y)$ as $P(\mathbf{X})P(Y|\mathbf{X})$
      Usually $P(\mathbf{X}, Y)$ is unknown to us (we just know it exists)
  - Training and test data are samples drawn from the *same* $P(\mathbf{X}, Y)$: they are identically distributed
  - Each $(\mathbf{x}, y)$ is drawn independently from $P(\mathbf{X}, Y)$

Overfitting



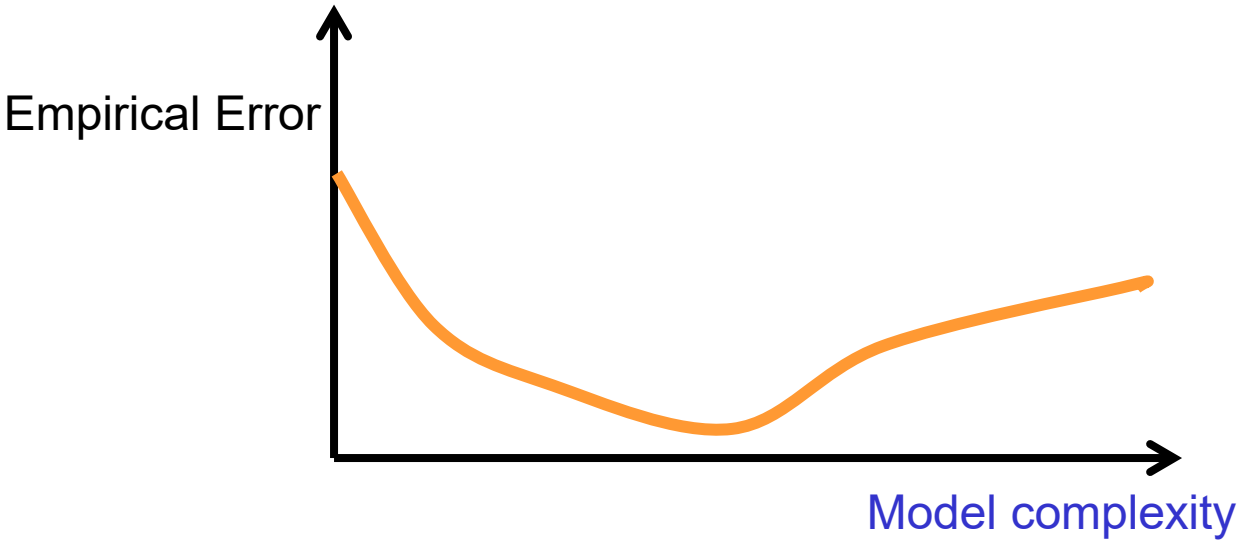
- A decision tree overfits the training data when its accuracy on the training data goes up but its accuracy on unseen data goes down

Overfitting



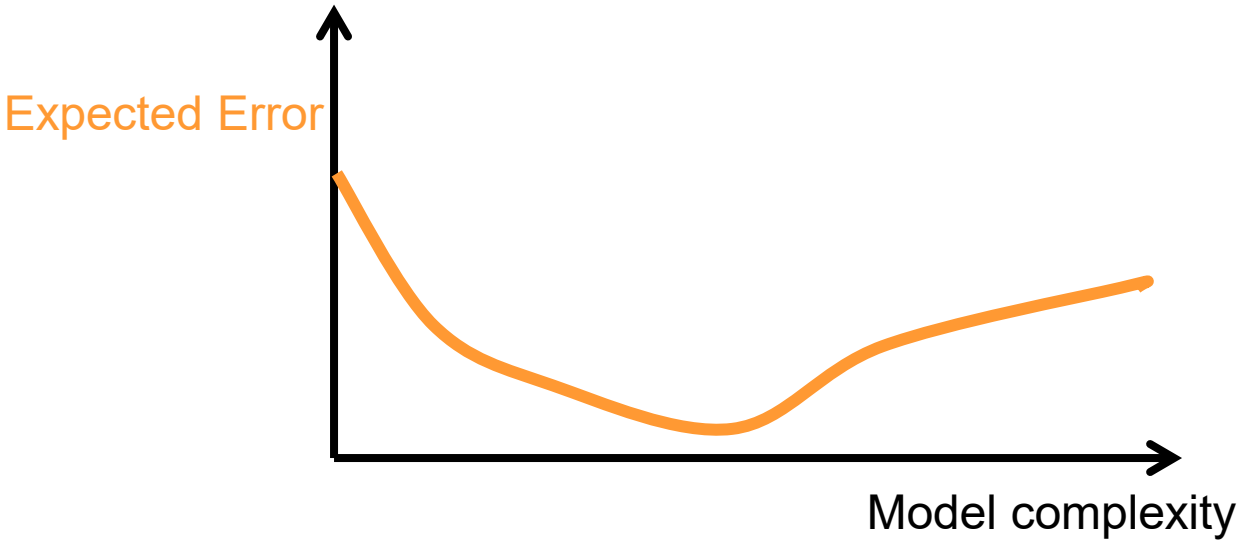
- Empirical error (= on a given data set):
  The percentage of items in this data set are misclassified by the classifier *f.*

Overfitting



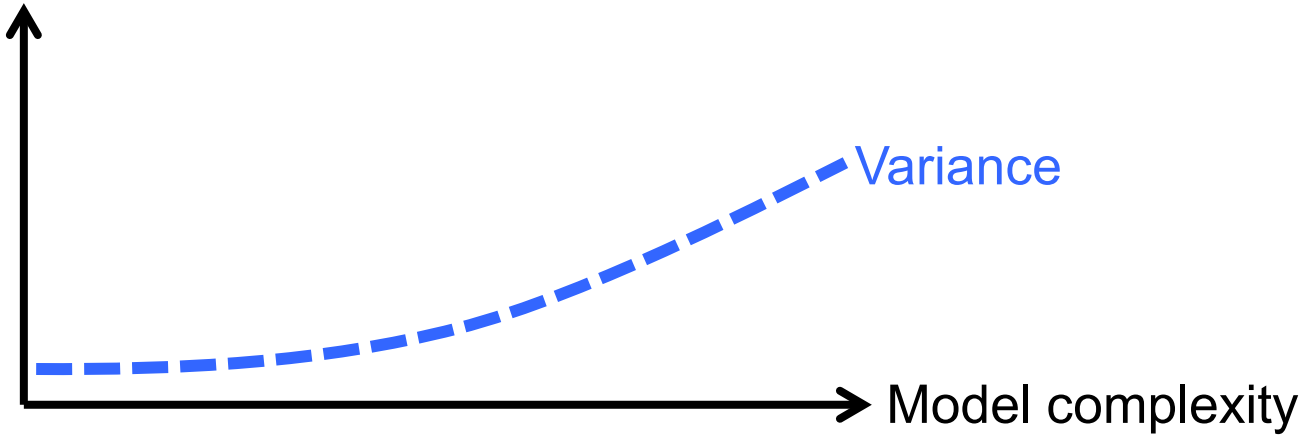
- Model complexity (informally):
  How many parameters do we have to learn?
  - Decision trees: complexity = #nodes

Overfitting



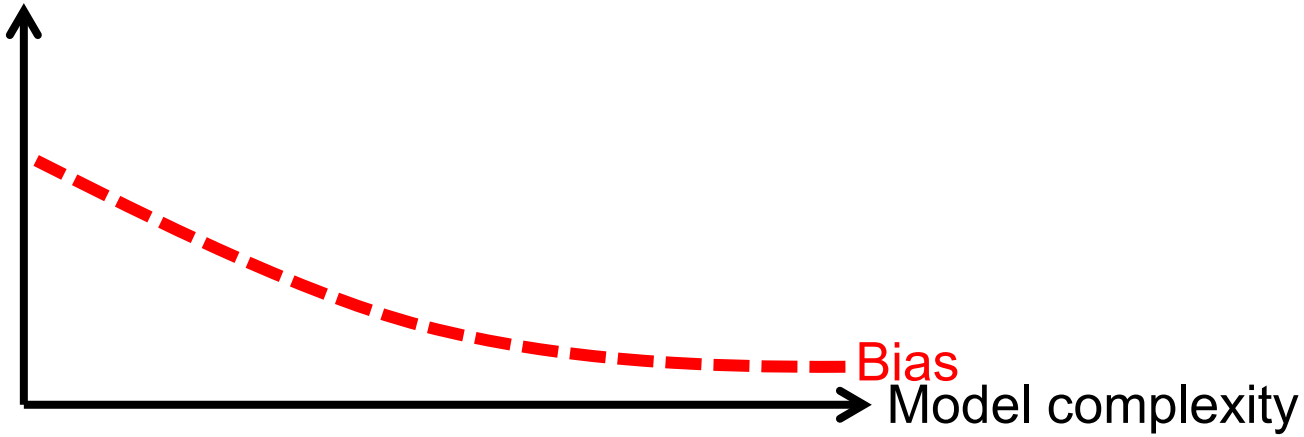
- Expected error:
  What percentage of items drawn from P(**x**,y) do we expect to be misclassified by *f*?
- (That's what we really care about – generalization)

Variance of a learner (informally)



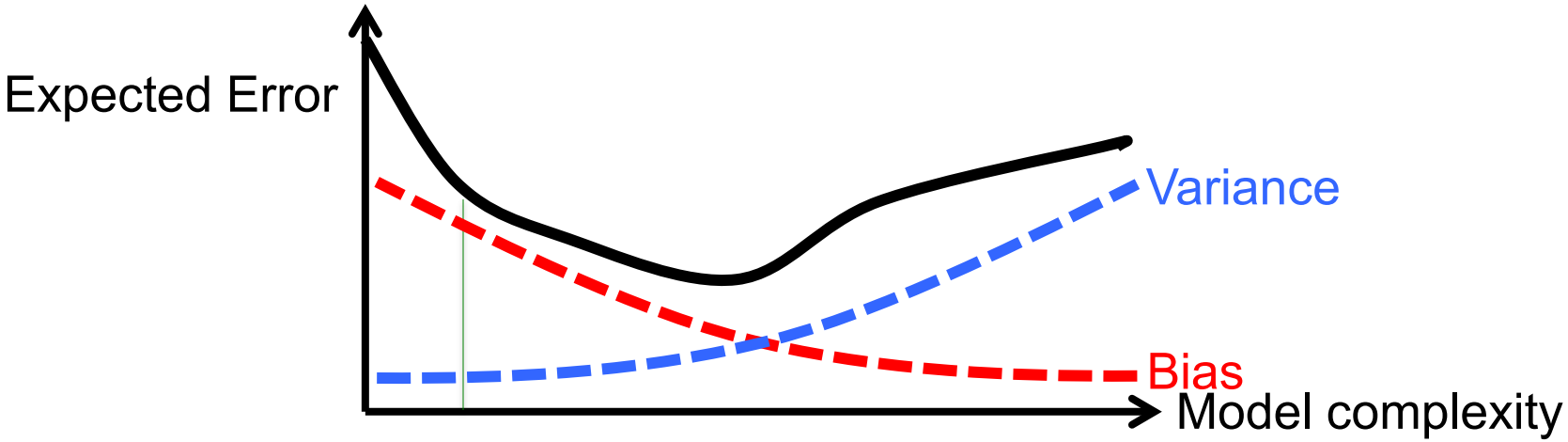
- How susceptible is the learner to minor changes in the training data?
  - (i.e. to different samples from P(X, Y))
- Variance increases with model complexity
  - Think about extreme cases: a hypothesis space with one function vs. all functions.
  - Or, adding the "wind" feature in the DT earlier.
  - The larger the hypothesis space is, the more flexible the selection of the chosen hypothesis is as a function of the data.
  - More accurately: for each data set D, you will learn a different hypothesis h(D), that will have a different true error e(h); we are looking here at the variance of this random variable.

Bias of a learner (informally)



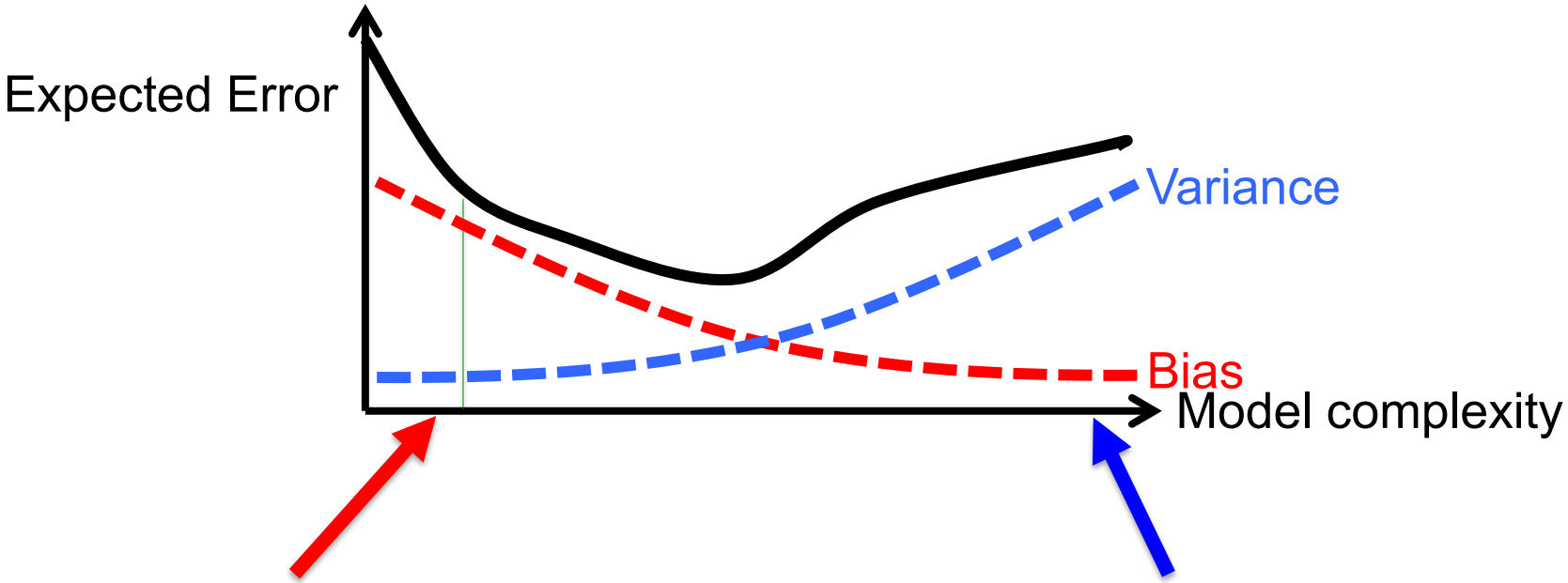
- How likely is the learner to identify the **target** hypothesis?
- Bias is low when the model is expressive (low empirical error)
- Bias is high when the model is (too) simple
  - The larger the hypothesis space is, the easiest it is to be close to the true hypothesis.
  - More accurately: for each data set D, you learn a different hypothesis h(D), that has a different true error e(h); we are looking here at the difference of the mean of this random variable from the true error.

Impact of bias and variance



- Expected error ≈ bias + variance

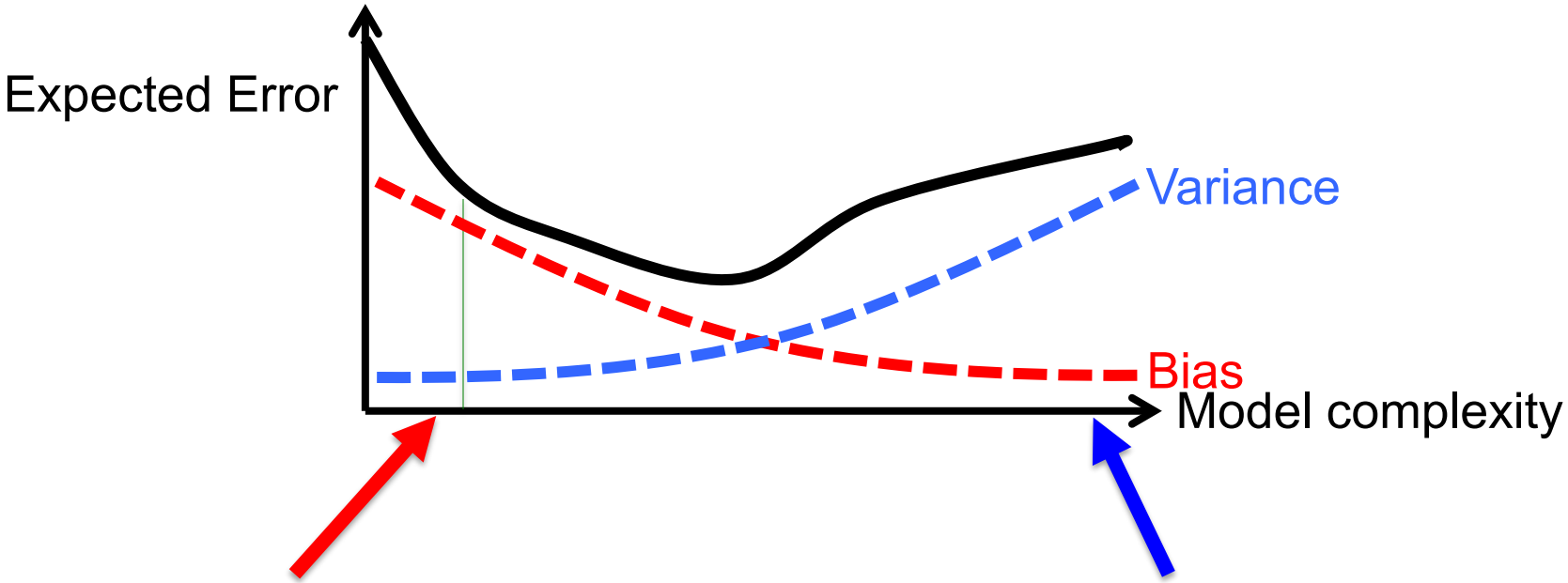
Model complexity



Simple models:
High bias and low variance

Complex models:
High variance and low bias

Underfitting and Overfitting



Simple models:
High bias and low variance

Complex models:
High variance and low bias

- This can be made more accurate for some loss functions.
- We will discuss a more precise and general theory that trades expressivity of models with empirical error

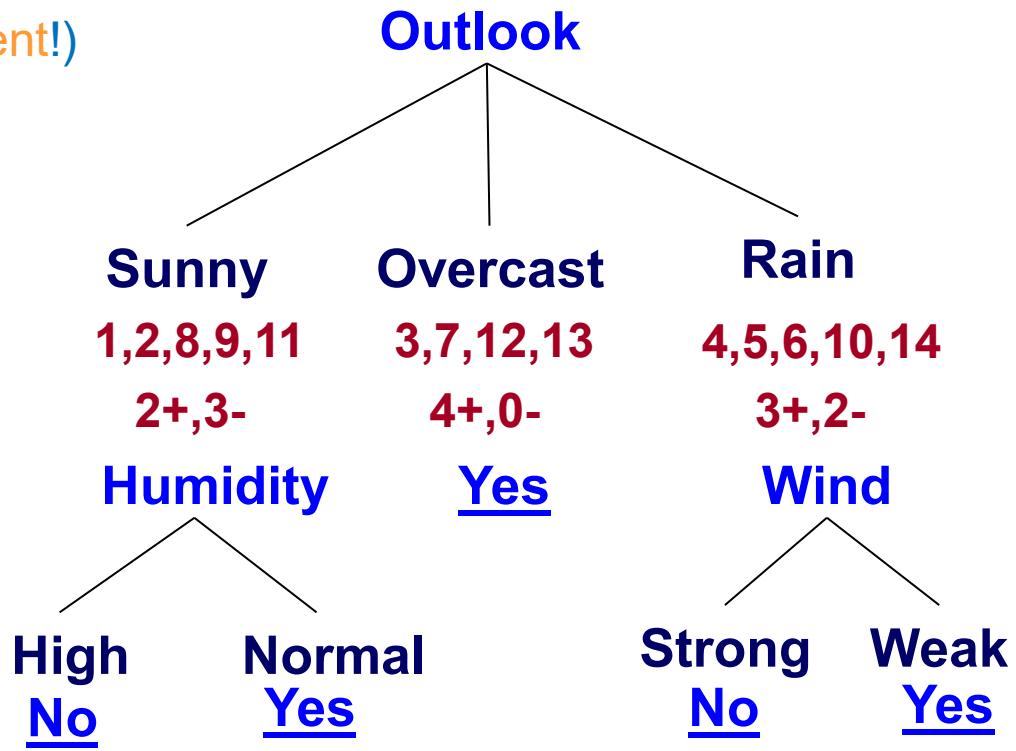
Avoiding Overfitting

How can this be avoided with linear classifiers?

- Two basic approaches
  - **Pre-pruning**: Stop growing the tree at some point during construction when it is determined that there is not enough data to make reliable choices.
  - **Post-pruning**: Grow the full tree and then remove nodes that seem not to have sufficient evidence.
- Methods for evaluating subtrees to prune
  - **Cross-validation**: Reserve hold-out set to evaluate utility
  - **Statistical testing**: Test if the observed regularity can be dismissed as likely to occur by chance
  - **Minimum Description Length**: Is the additional complexity of the hypothesis smaller than remembering the exceptions?
- This is related to the notion of regularization that we will see in other contexts – keep the hypothesis simple.

Trees and Rules

- Decision Trees can be represented as Rules
  - (outlook = sunny) and (humidity = normal) then YES
  - (outlook = rain) and (wind = strong) then NO
- Sometimes Pruning can be done at the rules level
  - Rules are generalized by
    erasing a condition (different!)



DT Extensions:
continuous attributes and missing values

Continuous Attributes

- Real-valued attributes can, in advance, be discretized into ranges, such as *big, medium, small*
- Alternatively, one can develop splitting nodes based on thresholds of the form A<c that partition the data into examples that satisfy A<c and A>=c. The information gain for these splits is calculated in the same way and compared to the information gain of discrete splits.
- How to find the split with the highest gain?
  - For each continuous feature A:
    - Sort examples according to the value of A
    - For each ordered pair (x,y) with different labels
      - Check the mid-point as a possible threshold, i.e.

$$S_{a \le x}, S_{a \ge y}$$

Continuous Attributes

- Example:
  - Length (L): 10 15 21 28 32 40 50
  - Class: - + + - + + -
  - Check thresholds: L<12.5; L<24.5; L<45
  - Subset of Examples= {…}, Split= k+, j-

- How to find the split with the highest gain?
  - For each continuous feature A:
    - Sort examples according to the value of A
    - For each ordered pair (x,y) with different labels
      - Check the mid-point as a possible threshold, i.e.

$$S_{a \le x}, S_{a \ge y}$$

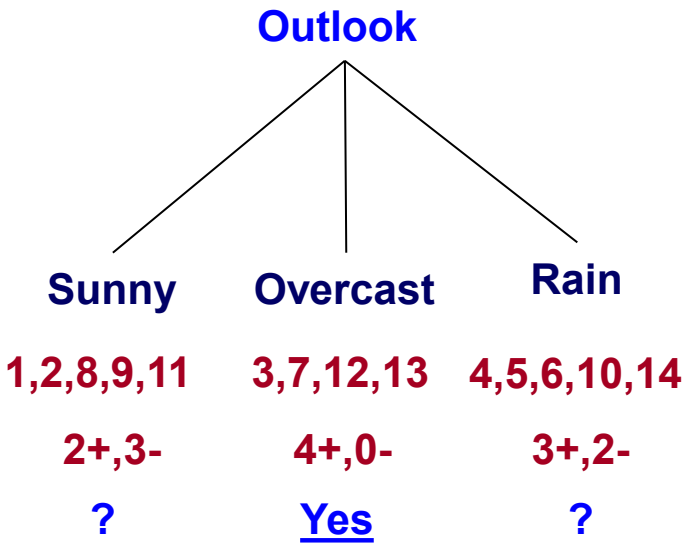
Missing Values

- Diagnosis = < fever, blood_pressure, …, blood_test=?, …>
- Many times values are not available for all attributes during training or testing (e.g., medical diagnosis)
- Training: evaluate Gain(S,a) where in some of the examples a value for a is not given

Missing Values

$$Gain(S,a) = Entropy(S) - \sum \frac{|S_v|}{|S|} Entropy(S_v)$$

Other suggestions?



$Gain(S_{sunny}, Temp) =$ **.97 – (3/5) 0 – (2/5) 1 = .57**

$Gain(S_{sunny}, Humidity) =$

- Fill in: assign the most likely value of X_i to **s**:
  argmax $_k$ $P(X_i = k)$: High
  - **.97- (3/5) Ent[+0,-3] - (2/5) Ent[+2,-0] = .97**
- Assign fractional counts $P(X_i = k)$ for each value of X_i to **s**
  - **.97- (2.5/5) Ent[+0,-2.5] - (2.5/5) Ent[+2,-.5] < .97**

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	???	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Missing Values

- Diagnosis = < fever, blood_pressure, …, blood_test=?, …>

- Many times values are not available for all attributes during training or testing (e.g., medical diagnosis)

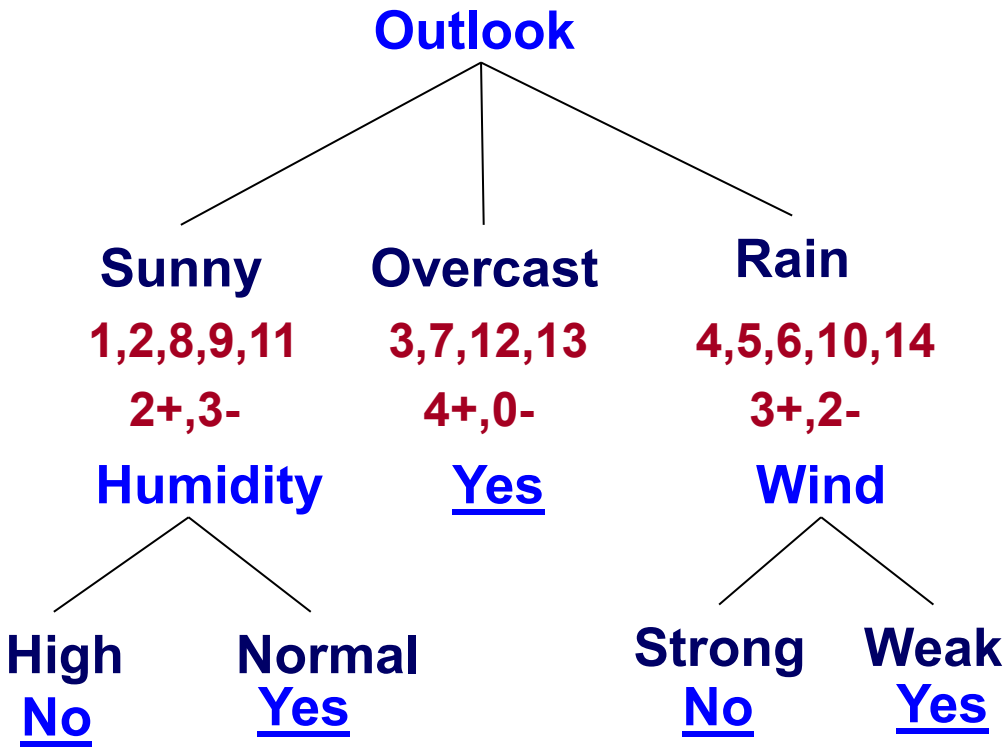
- Training: evaluate Gain(S,a) where in some of the examples a value for a is not given
- Testing: classify an example without knowing the value of a

Missing Values

Outlook = Sunny, Temp = Hot, Humidity = ???, Wind = Strong, label = ?? Normal/High

Outlook = ???, Temp = Hot, Humidity = Normal, Wind = Strong, label = ??

1/3 Yes + 1/3 Yes +1/3 No = Yes



Summary: Decision Trees

- Presented the hypothesis class of Decision Trees
  - Very expressive, flexible, class of functions
- Presented a learning algorithm for Decision Tress
  - Recursive algorithm
  - Key step is based on the notion of Entropy
- Discussed the notion of overfitting and ways to address it within DTs
  - In your problem set – look at the performance on the training vs. test
- Briefly discussed some extensions
  - Real valued attributes
  - Missing attributes
- Evaluation in machine learning
  - Cross validation
  - Statistical significance

Metrics
Methodologies

Metrics

- We train on our training data Train = $\{x_i, y_i\}_{1,m}$
- We test on Test data.
- We often set aside part of the training data as a development set, especially when the algorithms require tuning.
  - In the HW we asked you to present results also on the Training; why?
- When we deal with binary classification we often measure performance simply using Accuracy:

$$\text{accuracy} = \frac{\#\text{ correct predictions}}{\#\text{ test instances}}$$

$$\text{error} = 1 - \text{accuracy} = \frac{\#\text{ incorrect predictions}}{\#\text{ test instances}}$$

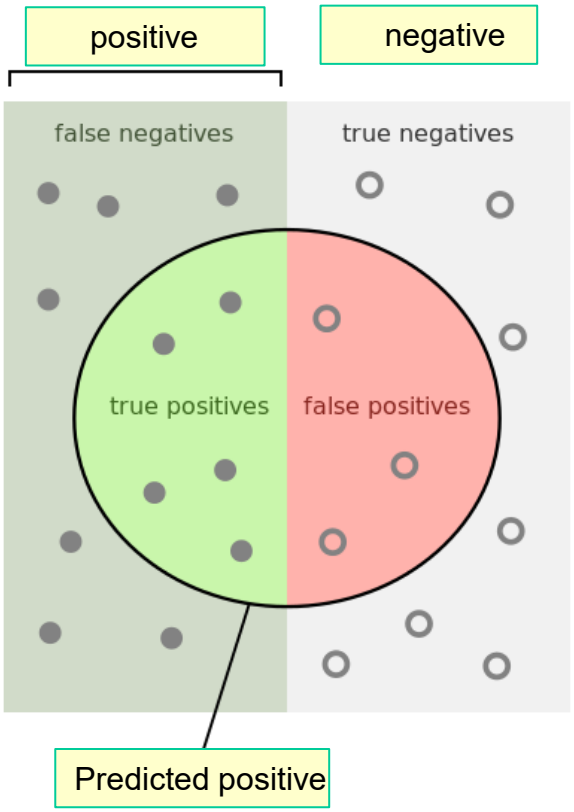
- Any possible problems with it?

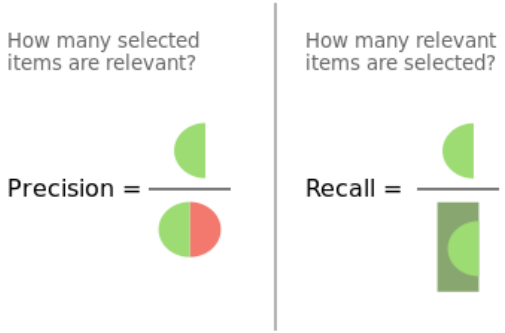
Alternative Metrics

- If the Binary classification problem is biased
  - In many problems most examples are negative
- Or, in multiclass classification
  - The distribution over labels is often non-uniform
- Simple accuracy is not a useful metric.
  - Often we resort to task specific metrics
- However one important example that is being used often involves Recall and Precision

- **Recall: # (positive identified = true positives) / # (all positive)**

- **Precision: # (positive identified = true positives) / # (predicted positive)**

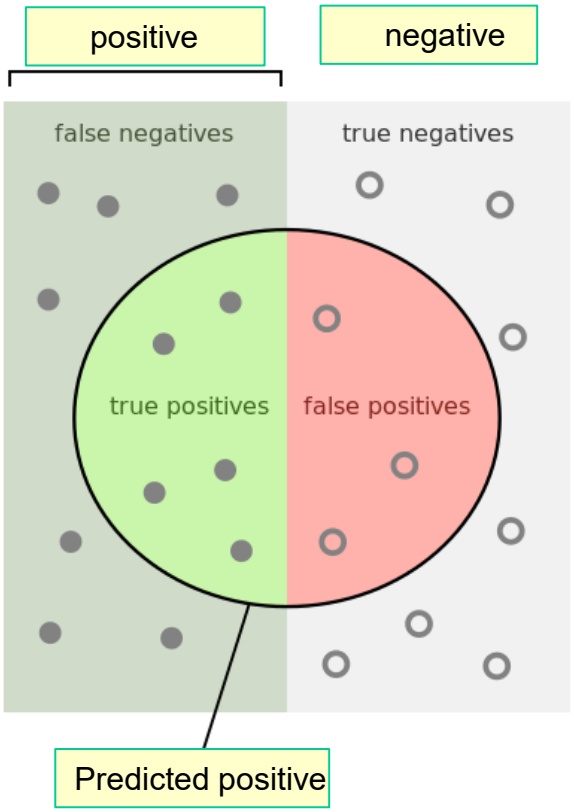


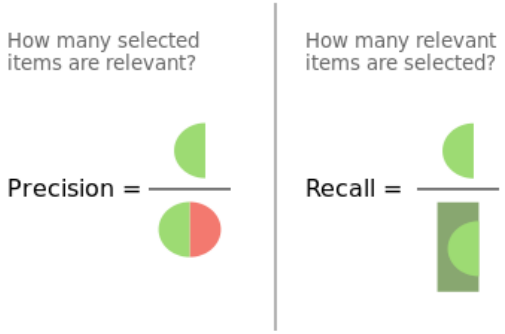


Example

- 100 examples, 5% are positive.
- Just say NO: your accuracy is 95%
  - Recall = precision = 0
- Predict 4+, 96-; 2 of the +s are indeed positive
  - Recall:2/5; Precision: 2/4

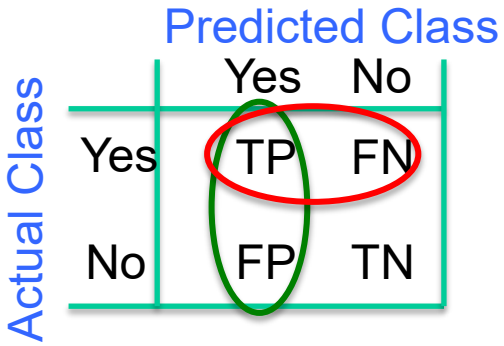
- **Recall: # (positive identified = true positives) / # (all positive)**
- **Precision: # (positive identified = true positives) / # (predicted positive)**





Confusion Matrix

- Given a dataset of P positive instances and N negative instances:



$$\text{accuracy} = \frac{TP + TN}{P + N}$$

- Imagine using classifier to identify positive cases (i.e., for information retrieval)

$$\text{precision} = \frac{TP}{TP + FP}$$

Probability that a randomly selected positive prediction is indeed positive

$$\text{recall} = \frac{TP}{TP + FN}$$

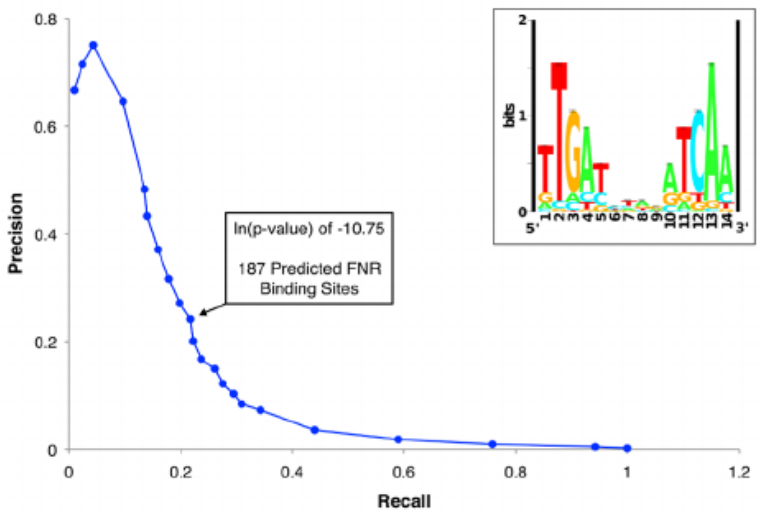
Probability that a randomly selected positive is identified

Relevant Metrics

- It makes sense to consider Recall and Precision together, or combine them into a single metric.
- Recall-Precision Curve:
- F-Measure:
  - A measure that combines precision and recall is the harmonic mean of precision and recall.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

  - F1 is the most commonly used metric.



Comparing Classifiers

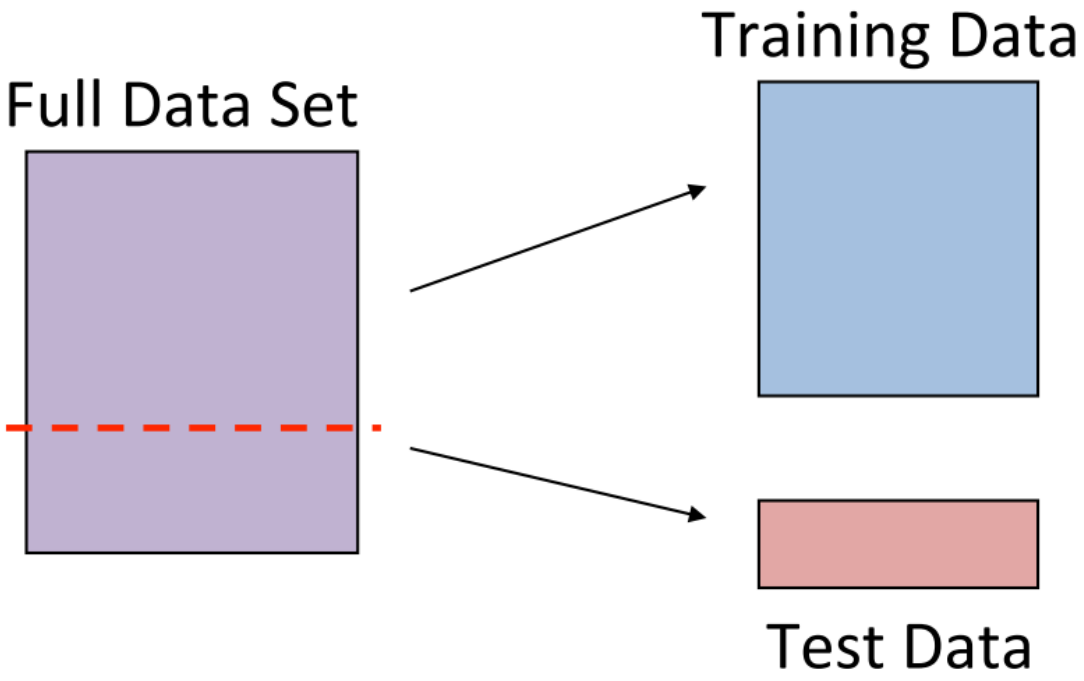
Say we have two classifiers, *C1* and *C2*, and want to choose the best one to use for future predictions

Can we use training accuracy to choose between them?

- No!

Instead, choose based on test accuracy...

Training and Test Data



Idea:

Train each model on the "training data" …

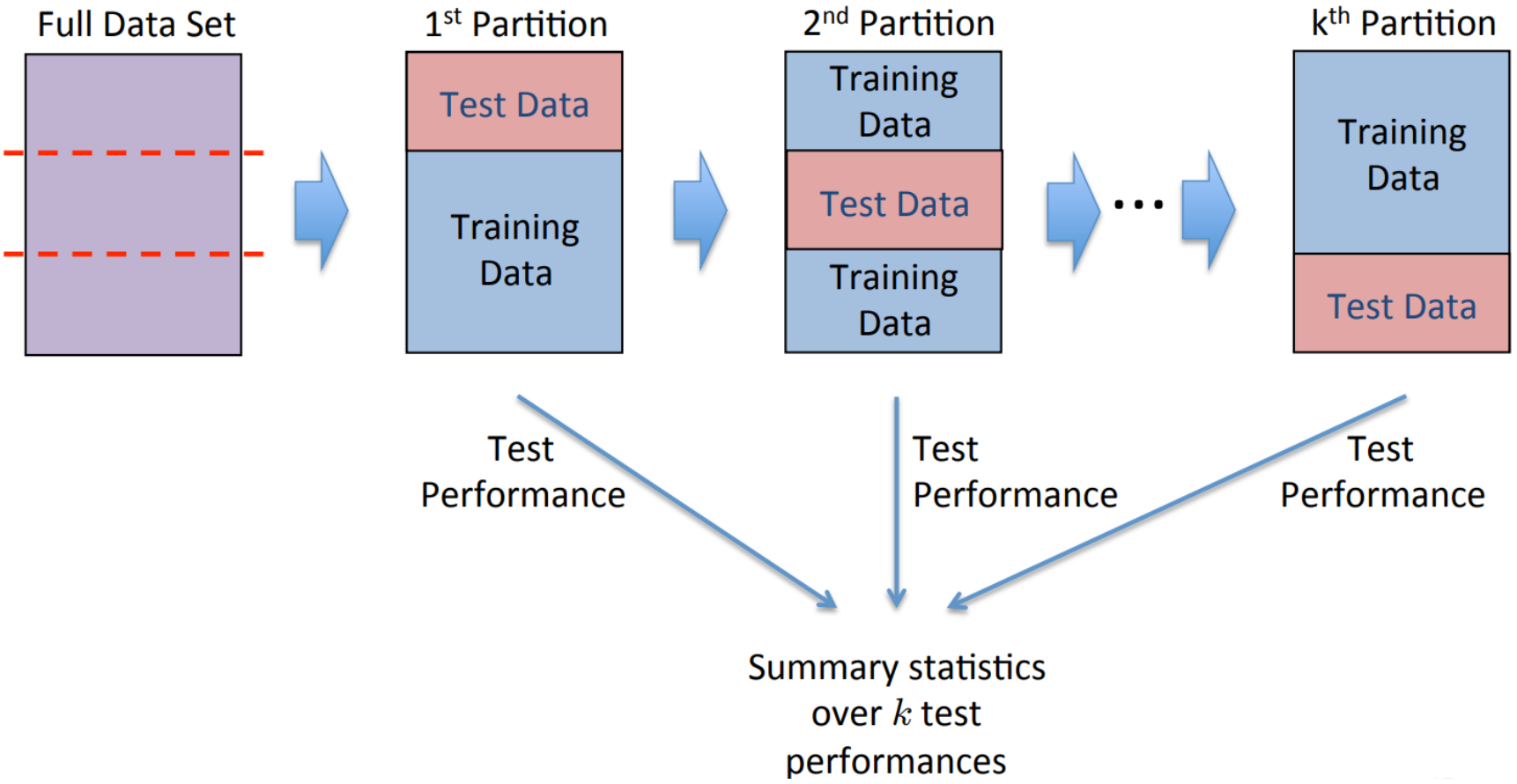
… and then test each model's accuracy on the test data

k-fold cross validation

- Why just choose one particular "split" of the data?
  - In principle, we should do this multiple times since performance may be different for each split

- *k*-Fold Cross-Validation (e.g., *k*=10)
  - Randomly partition full data set of *n* instances into *k* disjoint subsets (each roughly of size *n/k*)
  - Choose each fold in turn as the test set; train model on the other folds and evaluate
  - Compute statistics over *k* test performances, or choose best of the *k* models
  - Can also do "leave-one-out CV" where *k* = *n*

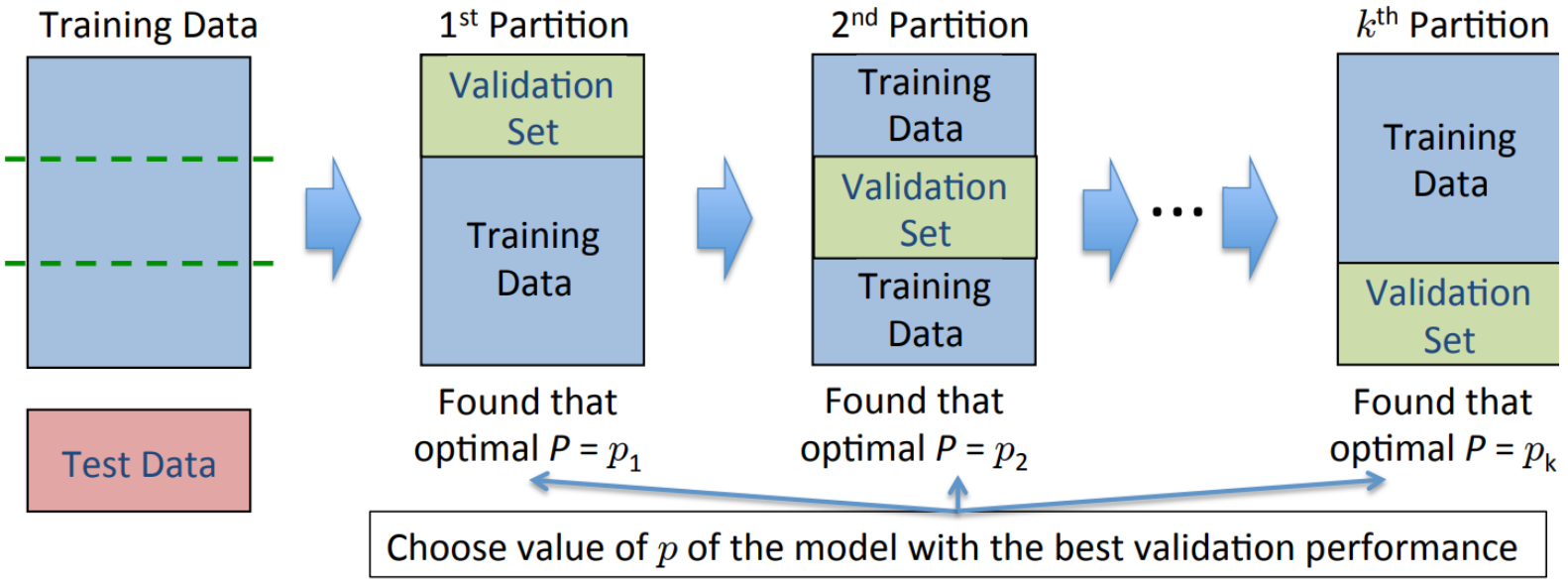
Example 3-Fold CV



Optimizing Model Parameters

Can also use CV to choose value of model parameter P

- Search over space of parameter values
  - Evaluate model with *P* = *p* on validation set
- Choose value p' with highest validation performance
- Learn model on full training set with P = p'

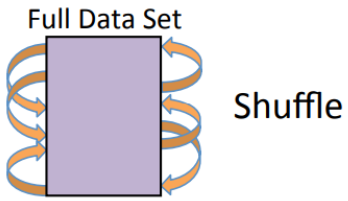


More on Cross-Validation

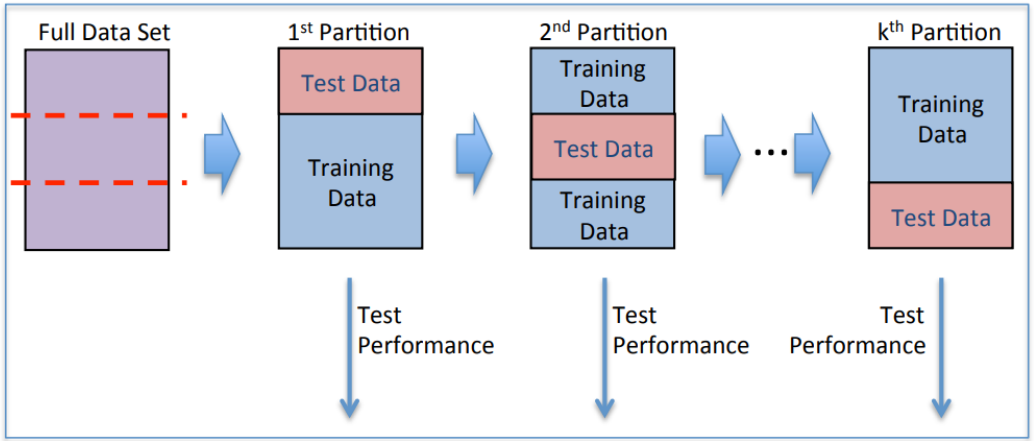
- Cross-validation generates an approximate estimate of how well the classifier will do on "unseen" data
  - As $k \rightarrow n$, the model becomes more accurate (more training data)
  - … but, CV becomes more computationally expensive
  - Choosing $k < n$ is a compromise
- Averaging over different partitions is more robust than just a single train/validate partition of the data
- It is an even better idea to do CV repeatedly!

Multiple Trials of k-Fold CV

1. Loop for *t* trials:
   1) Randomize Data Set



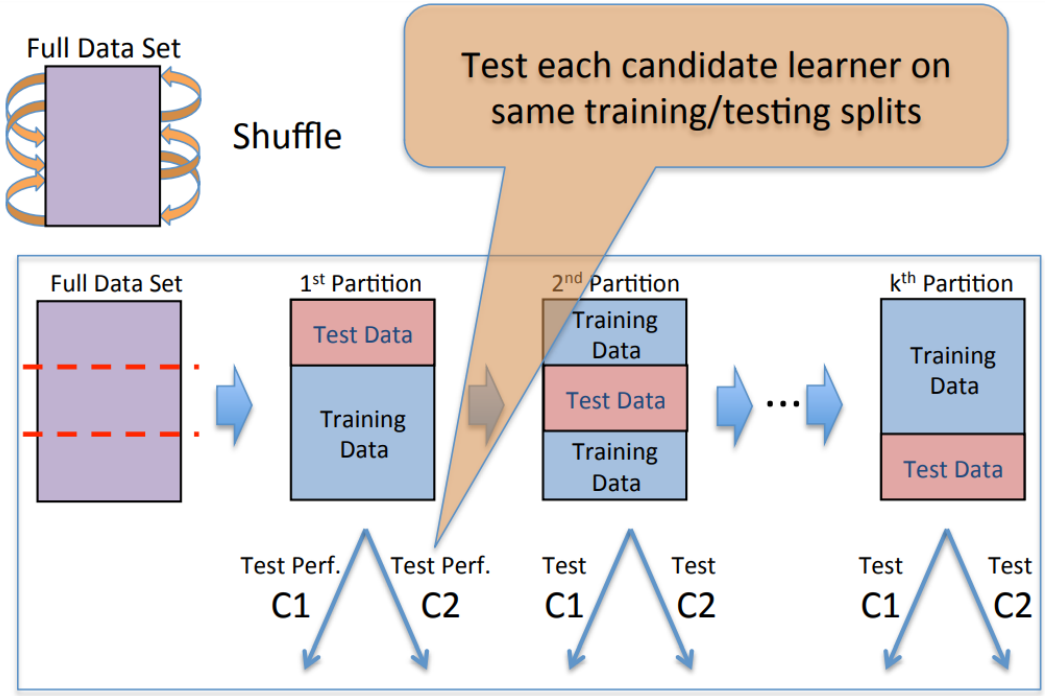
   2) Perform *k*-fold CV



2. Compute statistics over *t* x *k* test performances

Multiple Trials of k-Fold CV

1. Loop for *t* trials:
   1) Randomize Data Set
   2) Perform *k*-fold CV

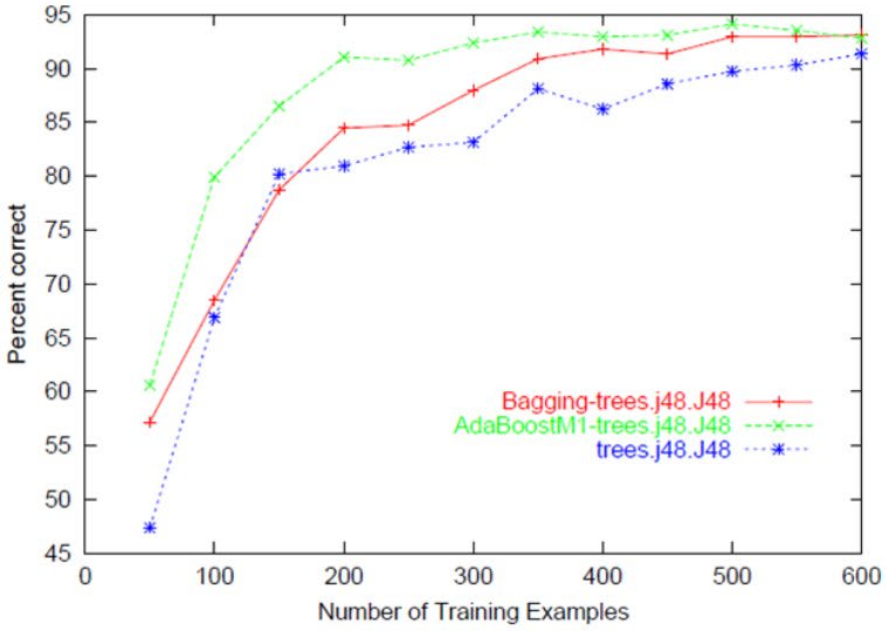


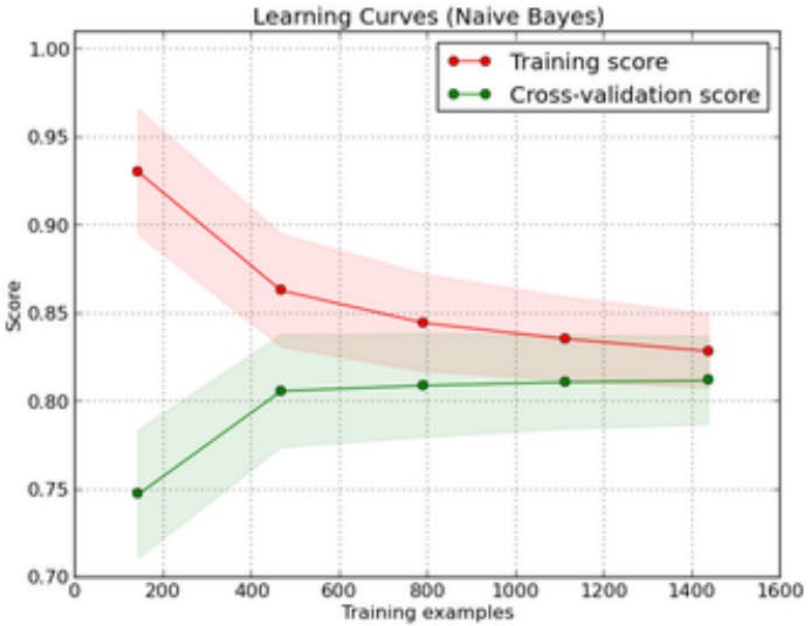
2. Compute statistics over *t* x *k* test performances

Allows us to do paired summary statistics (e.g., paired t-test)

Learning Curve

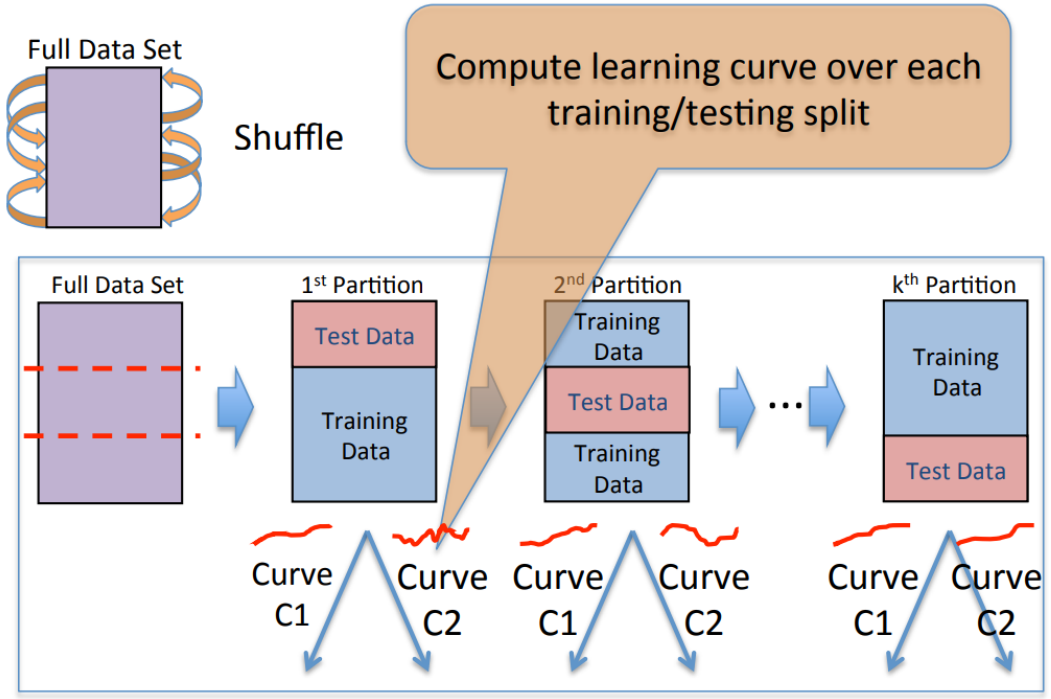
- Shows performance versus the # training examples
  - Compute over a single training/testing split
  - Then, average across multiple trials of CV





Multiple Trials of k-Fold CV

1. Loop for *t* trials:
   1) Randomize Data Set
   2) Perform *k*-fold CV



2. Compute statistics over *t* x *k* test performances

Decision Trees - Summary

- Hypothesis Space:
  - Variable size (contains all functions)
  - Deterministic; Discrete and Continuous attributes
- Search Algorithm
  - ID3 - batch
  - Extensions: missing values
- Issues:
  - What is the goal?
  - When to stop? How to guarantee good generalization?
- Did not address:
  - How are we doing? (Correctness-wise, Complexity-wise)

Reference

- https://www.seas.upenn.edu/~cis519