Machine learning: Part 1

- Supervised learning
- Decision tree learning
- Statistical learning
- Learning from complete Data

*Slides based on those of Pascal Poupart

What is Machine Learning?

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. [Mitchell, 1997]

Common learning tasks

- Supervised learning: Given some example input – output pairs, learn a function that maps from input to output.
- Unsupervised learning: Final natural classes for examples
- Reinforcement learning: determine what to do based on a series of rewards or punishments

Examples

- Checker (reinforcement learning):
    - T: playing checker
    - P: percent of games won against an opponent
    - E: playing practice games against itself
- Handwriting recognition (supervised learning):
    - T: recognize handwritten words within images
    - P: percent of words correctly recognized
    - E: database of handwritten words with given classifications
- Customer profiling (分析) (unsupervised learning):
    - T: cluster customers based on transaction patterns
    - P: homogeneity (同种性) of clusters
    - E: database of customer transactions

Representation

- Representation of the learned information is important
  - Determines how the learning algorithm will work
- Common representations:
  - Linear weighted polynomials
  - Propositional logic
  - First order logic
  - Bayes nets
  - ...

Supervised learning

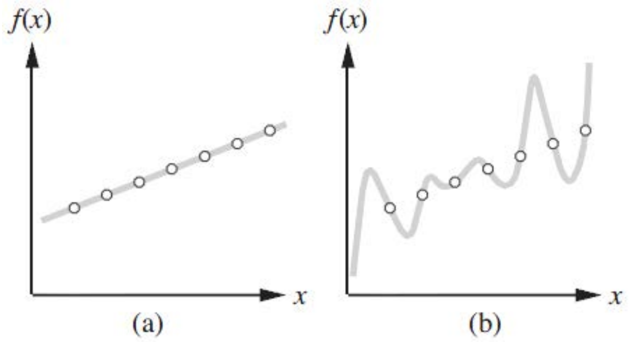
- Definition: Given a training set of N example input – output pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$, where each y_j was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f.
- The function h is a hypothesis.
- Learning is a search through the space of possible hypotheses for one that will perform well, even on new examples beyond the training set.

Classification and regression

- When the output y is one of a finite set of values, the learning problem is called classification (分类).
- Called Boolean or binary classification, if there are only two values.
- When y is a number, the learning problem is called regression (回归).

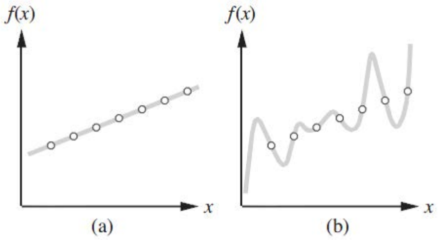
A regression example

- Fitting a function of a single variable to some data points
- A hypothesis is consistent if it agrees with all the data
- A linear hypothesis and a degree 7 polynomial hypothesis



Hypothesis space

- Hypothesis space: set of all hypotheses h under consideration
- *e.g.*, set of polynomials
- How to choose from among multiple consistent hypotheses?
- Prefer the simplest hypothesis consistent with the data.
- This principle is called Ockham's razor (奥坎姆剃刀), which is against all sorts of complications.
- *e.g.*, (a) should be preferred to (b).

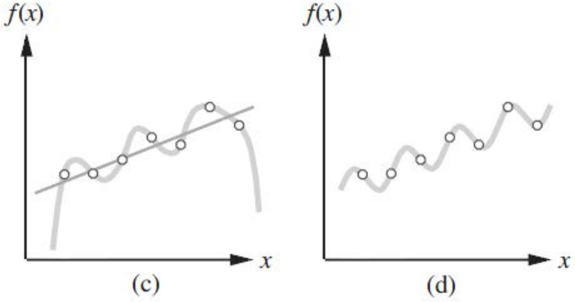


Generalization

- A good hypothesis will generalize (泛化) well, *i.e.*, predict unseen examples well
- In general, there is a tradeoff between complex hypotheses that fit the training data well and simpler hypotheses that may generalize better

An example

- No consistent straight line for this data set
- Require a degree-6 polynomial for an exact fit
- Can be fitted exactly by a simple function of the form $ax + b + c\sin(x)$



Realizability

- Finding a consistent hypothesis depends on the hypothesis space
- We say that a learning problem is realizable (可实现的) if the hypothesis space contains the true function.
- Unfortunately, we cannot always tell whether a given learning problem is realizable, because the true function is not known.

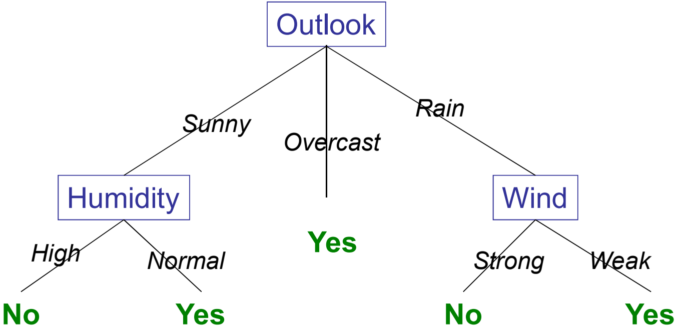
Realizability

- Why not let H be the class of all Java programs, or Turing machines, since every computable function can be represented by some Turing machine?
- There is a tradeoff between the expressiveness of a hypothesis space and the complexity of finding a good hypothesis within that space.
- *e.g.*, fitting a straight line to data is easy; fitting high-degree polynomials is harder; and fitting Turing machines is in general undecidable.

Decision trees

- Represent a function that takes as input a vector of attribute values and returns a “decision” —a single output value.
- Reach the decision by performing a sequence of tests.
- Nodes: labeled with attributes
- Edges: labeled with attribute values
- Leaves: labeled with output values

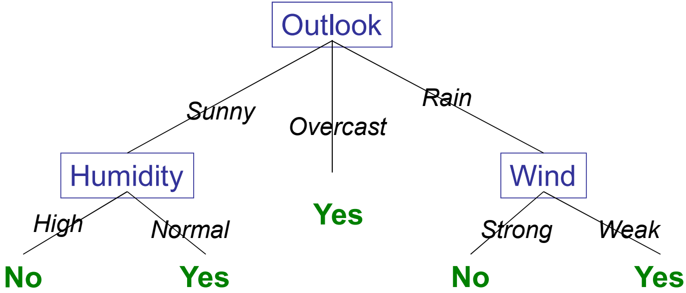
An example (playing tennis)



An instance
<Outlook=Sunny, Temp=Hot, Humidity=High, Wind=Strong>

Classification: No

Decision trees can represent disjunctions of conjunctions of constraints on attribute values



(Outlook=Sunny ∧ Humidity=Normal)
∨ (Outlook=Overcast)
∨ (Outlook=Rain ∧ Wind=Weak)

Decision tree representation

- Any Boolean function can be written as a decision tree
- By allowing each row in the truth table correspond to a path in the tree
- Can often use small trees
- Some functions require exponentially large trees
- *e.g.*, the majority function, which returns true iff more than half of the inputs are true,
- No representation efficient for all functions
- With n Boolean attributes, there are 2^{2^n} Boolean functions

Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: choose "most significant" attribute as root of (sub)tree

function DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*) **returns** a tree

```
if examples is empty then return PLURALITY-VALUE(parent_examples)
else if all examples have the same classification then return the classification
else if attributes is empty then return PLURALITY-VALUE(examples)
else
    A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
    tree ← a new decision tree with root test A
    for each value v_k of A do
        exs ← {e : e ∈ examples and e.A = v_k}
        subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
        add a branch to tree with label (A = v_k) and subtree subtree
    return tree
```

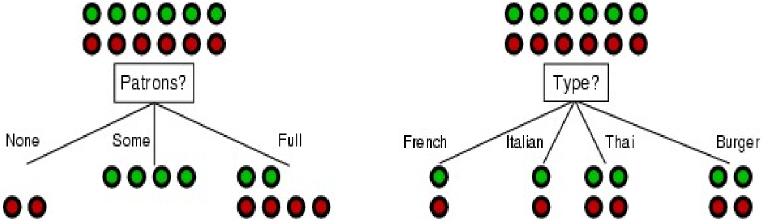
Plurality-value(examples) returns the majority classification of the examples

An example: restaurant

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



- *Patrons?* is a better choice

Using information theory

- We will use the notion of information gain (信息增益), which is defined in terms of entropy (熵), the fundamental quantity in information theory.
- Entropy is a measure of the uncertainty of a random variable; acquisition of information corresponds to a reduction in entropy.
- A random variable with only one value has no uncertainty and thus its entropy is defined as zero.
- A flip of a fair coin has "1 bit" of entropy.
- The roll of a fair four-sided die has 2 bits of entropy, because it takes 2 bits to describe one of 4 equally probable choices.

Entropy

- The entropy of a random variable V with values v_k, each with probability $P(v_k)$:

$$H(V) = -\sum_k P(v_k) \log_2 P(V_k)$$

- The entropy of a Boolean random variable that is true with probability q:

$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$$

- If a training set contains p positive examples and n negative examples, then the entropy of the goal attribute on the whole set is

$$H(Goal) = B(\frac{p}{p+n})$$

Information gain

- An attribute A with d distinct values divides the training set E into subsets $E_1, \ldots, E_d$.
- Each subset E_k has p_k positive examples and n_k negative examples,
- So the expected entropy remaining after testing attribute A is

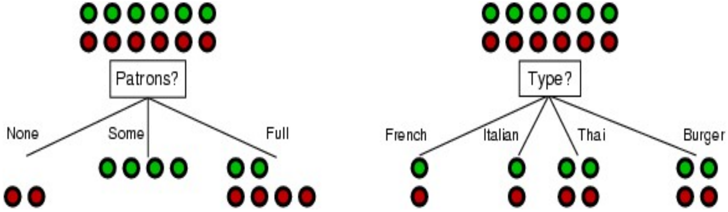
$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p + n} B(\frac{p_k}{p_k + n_k}).$$

- The information gain (IG) from the attribute test on A is the expected reduction in entropy:

$$Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$$

- Choose the attribute with the largest IG

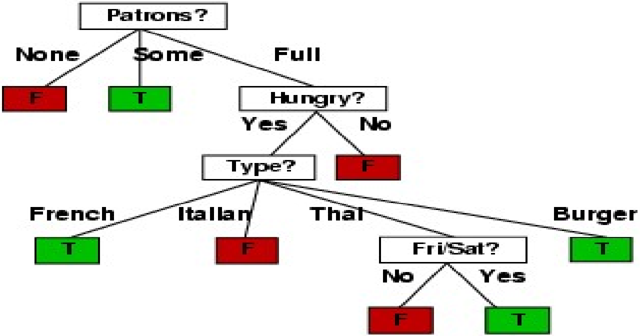
An example



- For the training set, $p = n = 6$, $B(6/12) = 1$
- $Gain(Pat) = 1 - [\frac{2}{12}B(\frac{0}{2}) + \frac{4}{12}B(\frac{4}{4}) + \frac{6}{12}B(\frac{2}{6})] \approx 0.541$
- $Gain(Type) = 1 - [\frac{2}{12}B(\frac{1}{2}) + \frac{2}{12}B(\frac{1}{2}) + \frac{4}{12}B(\frac{2}{4}) + \frac{4}{12}B(\frac{2}{4})] = 0$
- So Patrons is a better attribute to split on.
- In fact, Patrons has the maximum gain of any of the attributes and would be chosen by the DTL algorithm as the root.

An example

Decision tree learned from the 12 examples:



Performance of a learning algorithm

- A learning algorithm is good if it produces a hypothesis that does a good job of predicting classifications of unseen examples
- Verify performance with a test set
  - Collect a large set of examples
  - Divide into 2 disjoint sets: training set and test set
  - Learn hypothesis h with training set
  - Measure percentage of correctly classified examples by h in the test set
  - Repeat 2-4 for different randomly selected training sets of varying sizes

Learning curves

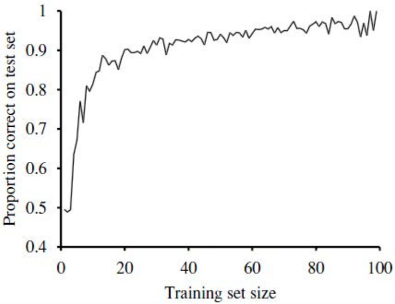


Figure 18.7 A learning curve for the decision tree learning algorithm on 100 randomly generated examples in the restaurant domain. Each data point is the average of 20 trials.

Overfitting

- Decision-tree grows until all training examples are perfectly classified
- But what if
  - Data is noisy
  - Training set is too small to give a representative sample of the target function
- May lead to Overfitting!
  - Common problem with most learning algorithms

Overfitting (过度拟合)

- Definition: Given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$ such that h has smaller error than h' over the training examples but h' has smaller error than h over the entire distribution of instances
- Avoiding overfitting for DTL: Decision tree pruning: Eliminating nodes that are not clearly relevant.

K-fold Cross-validation (交叉验证)

- Split data in two parts, one for training, one for testing the accuracy of a hypothesis
- Run k experiments, each time putting aside $1/k$ of the data to test on
- Take the average test set score of the k rounds
- Popular values for k are 5 and 10

Exercise

Perform DTL on the following dataset, where D is the output

A	B	C	D
0	0	1	0
0	1	0	0
1	1	0	0
0	0	1	1
1	1	1	1
1	0	0	1
1	1	0	1

Candy example

- Favorite candy sold in two flavors: Cherry (yum), Lime (ugh)
- Same wrapper for both flavors
- Sold in bags with different ratios:
  - 100% cherry
  - 75% cherry + 25% lime
  - 50% cherry + 50% lime
  - 25% cherry + 75% lime
  - 100% lime
- You bought a bag of candy but don't know its flavor ratio
- After eating k candies:
  - What's the flavor ratio of the bag?
  - What will be the flavor of the next candy?

Candy example

- Hypothesis H: probabilistic theory of the world
  - h_1: 100% cherry
  - h_2: 75% cherry + 25% lime
  - h_3: 50% cherry + 50% lime
  - h_4: 25% cherry + 75% lime
  - h_5: 100% lime
- Data D: evidence about the world
  - d_1: 1st candy is cherry
  - d_2: 2nd candy is lime
  - d_3: 3rd candy is lime
  - ...

Bayesian Learning

- Prior: $Pr(H)$
- Likelihood: $Pr(d|H)$
- Evidence: $d = \langle d_1, d_2, \ldots, d_n \rangle$
- Computing the posterior using Bayes'Theorem:

$$Pr(H|d) = \alpha Pr(d|H) Pr(H)$$

Bayesian Prediction

- Suppose we want to make a prediction about an unknown quantity X (*i.e.*, the flavor of the next candy)

$$P(X|d) = \sum_i P(X|d, h_i)P(h_i|d) = \sum_i P(X|h_i)P(h_i|d)$$

- Predictions are weighted averages of the predictions of the individual hypotheses
- Hypotheses serve as "intermediaries" between raw data and prediction

Candy Example

- Hypothesis H:
  - h_1: 100% cherry
  - h_2: 75% cherry + 25% lime
  - h_3: 50% cherry + 50% lime
  - h_4: 25% cherry + 75% lime
  - h_5: 100% lime
- Assume prior $P(H) = \langle 0.1, 0.2, 0.4, 0.2, 0.1 \rangle$
- Assume candies are i.i.d. (identically and independently distributed), *i.e.*, $P(d|h) = \Pi_j P(d_j|h)$
- Suppose first 10 candies all taste lime:
  - $P(d|h_5) = 1^{10} = 1$,
  - $P(d|h_3) = 0.5^{10} = 0.00097$
  - $P(d|h_1) = 0^{10} = 0$

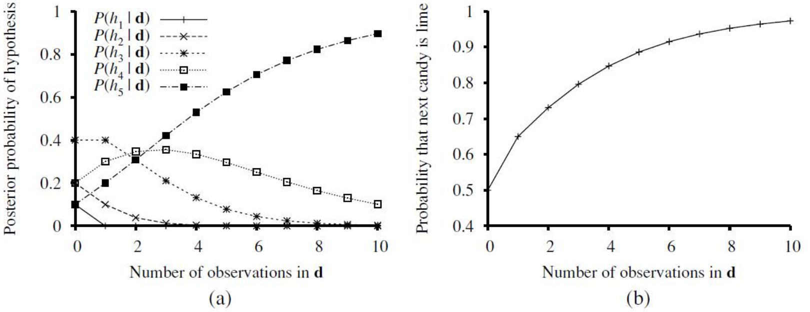


Figure 20.1 (a) Posterior probabilities $P(h_i \mid d_1, \ldots, d_N)$ from Equation (20.1). The number of observations N ranges from 1 to 10, and each observation is of a lime candy. (b) Bayesian prediction $P(d_{N+1} = lime \mid d_1, \ldots, d_N)$ from Equation (20.2).

Bayesian learning properties

- Optimal (*i.e.*, given prior, no other prediction is correct more often than the Bayesian one)
- No overfitting (all hypotheses weighted and considered)
- There is a price to pay:
  - When hypothesis space is large, Bayesian learning may be intractable
  - *i.e.*, sum (or integral) over hypothesis often intractable
- Solution: approximate Bayesian learning

Maximum a posteriori (极大后验,MAP)

- Idea: make prediction based on most probable hypothesis
  - $h_{\text{MAP}} = \text{argmax}_{h_i} P(h_i|d)$
  - $P(X|d) \approx P(X|h_{\text{MAP}})$

- In contrast, Bayesian learning makes prediction based on all hypotheses weighted by their probability

Candy Example (MAP)

- Prediction after
  - 1 lime: $h_{\text{MAP}} = h_3$, $Pr(lime|h_{\text{MAP}}) = 0.5$
  - 2 limes: $h_{\text{MAP}} = h_4$, $Pr(lime|h_{\text{MAP}}) = 0.75$
  - 3 limes: $h_{\text{MAP}} = h_5$, $Pr(lime|h_{\text{MAP}}) = 1$
  - 4 limes: $h_{\text{MAP}} = h_5$, $Pr(lime|h_{\text{MAP}}) = 1$
  - ...
- After only 3 limes, it correctly selects h_5
- But what if correct hypothesis is h_4?
- After 3 limes, MAP incorrectly predicts h_5
  - MAP yields $P(lime|h_{MAP}) = 1$
  - Bayesian learning yields $P(lime|d) = 0.8$

MAP properties

- MAP prediction less accurate than Bayesian prediction since it relies only on one hypothesis h_{MAP}
- But MAP and Bayesian predictions converge as data increases
- Controlled overfitting (prior can be used to penalize complex hypotheses)
- Finding h_{MAP} may be intractable:
  - $h_{MAP} = \text{argmax}_h P(h|d)$
  - Optimization may be difficult

MAP computation

- Optimization:
  - $h_{\text{MAP}} = \text{argmax}_h P(h|d) = \text{argmax}_h P(h)P(d|h) = \text{argmax}_h P(h)\Pi_i P(d_i|h)$
- Product induces non-linear optimization
- Take the log to linearize optimization
  - $h_{\text{MAP}} = \text{argmax}_h \log P(h) + \sum_i \log P(d_i|h)$

Maximum Likelihood (极大似然,ML)

- Idea: simplify MAP by assuming uniform prior (*i.e.*, $P(h_i) = P(h_j)$ for all i, j)
  - $h_{\text{MAP}} = \text{argmax}_h P(h)P(d|h)$
  - $h_{\text{ML}} = \text{argmax}_h P(d|h)$
- Make prediction based on h_{ML} only:
  - $P(X|d) \approx P(X|h_{\text{ML}})$

ML properties

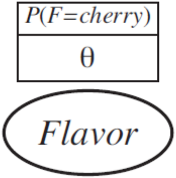
- ML prediction less accurate than Bayesian and MAP predictions since it ignores prior info and relies only on one hypothesis h_{ML}
- But ML, MAP and Bayesian predictions converge as data increases
- Subject to overfitting (no prior to penalize complex hypothesis that could exploit statistically insignificant data patterns)
- Finding h_{ML} is often easier than h_{MAP}
  - $h_{\text{ML}} = \text{argmax}_h \sum_i \log P(d_i|h)$

Statistical Learning

- Use Bayesian Learning, MAP or ML
- Complete data:
  - When data has multiple attributes, all attributes are known
  - Easy
- Incomplete data:
  - When data has multiple attributes, some attributes are unknown
  - Harder

Simple ML example

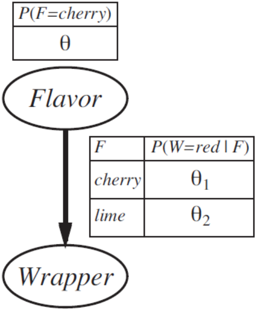
- Hypothesis h_θ
  - $P(cherry) = \theta$ and $P(lime) = 1 - \theta$
- Data d:
  - c cherries and l limes



- $P(d|h_\theta) = \theta^c (1-\theta)^l$
- $\log P(d|h_\theta) = c \log \theta + l \log(1-\theta)$
- $d(logP(d|h_\theta))/d\theta = c/\theta - l/(1-\theta)$
- $c/\theta - l/(1-\theta) = 0 \Rightarrow \theta = c/(c+l)$

More complicated ML example

- Hypothesis $h_{\theta,\theta_1,\theta_2}$
- Data d:
  - c cherries: g_c green and r_c red
  - l limes: g_l green and r_l red



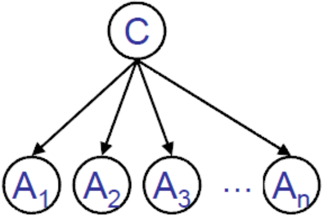
- $P(d|h_{\theta,\theta_1,\theta_2}) = \theta^c(1-\theta)^l\theta_1^{r_c}(1-\theta_1)^{g_c}\theta_2^{r_l}(1-\theta_2)^{g_l}$
- $c/\theta - l/(1-\theta) = 0 \Rightarrow \theta = c/(c+l)$
- $r_c/\theta_1 - g_c/(1-\theta_1) = 0 \Rightarrow \theta_1 = r_c/(r_c+g_c)$
- $r_l/\theta_2 - g_l/(1-\theta_2) = 0 \Rightarrow \theta_2 = r_l/(r_l+g_l)$

Laplace Smoothing

- An important case of overfitting happens when there is no sample for a certain outcome
  - *e.g.*, no cherries eaten so far
  - $P(cherry) = \theta = c/(c + l) = 0$
  - Zero prob. are dangerous: they rule out outcomes
- Solution: Laplace (add-one) smoothing
  - Add 1 to all counts
  - $P(cherry) = \theta = (c + 1)/(c + l + 2) > 0$
  - Much better results in practice

Naive Bayes models

- Want to predict a class C based on attributes $A_1, \ldots, A_n$
- Parameters:
  - $\theta = P(C = true)$
  - $\theta_{i1} = P(A_i = true | C = true)$
  - $\theta_{i2} = P(A_i = true | C = false)$
- Assumption: A_i's are independent given C



An example: restaurant

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Naive Bayes learning

- Notation: $p = \#(c), n = \#(-c), p_i^+ = \#(c, a_i)$,
  $n_i^+ = \#(c, -a_i), p_i^- = \#(-c, a_i), n_i^- = \#(-c, -a_i)$
- $P(d|h) = \theta^p(1-\theta)^n \Pi_i \theta_{i1}^{p_i^+} \theta_{i2}^{p_i^-} (1-\theta_{i1})^{n_i^+} (1-\theta_{i2})^{n_i^-}$
- $\theta = p/(p+n)$, $\theta_{i1} = p_i^+/(p_i^+ + n_i^+)$, $\theta_{i2} = p_i^-/(p_i^- + n_i^-)$,
- $P(C|a_1, \ldots, a_n) = \alpha P(C) \Pi_i P(a_i|C)$
- Choose the most likely class

Naive Bayes vs decision trees

Less accurate since the true hypothesis, which is a decision tree, is not representable exactly using a naive Bayes model.

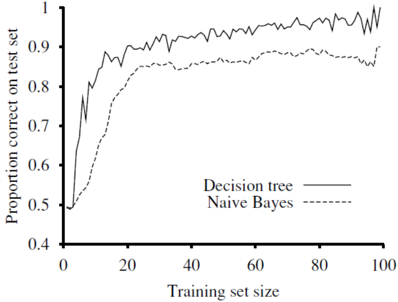


Figure 20.3 The learning curve for naive Bayes learning applied to the restaurant problem from Chapter 18; the learning curve for decision-tree learning is shown for comparison.

Bayesian network parameter learning (ML)

- Parameters $\theta_{V,pa(V)=\mathbf{v}}$:
  - CPTs: $\theta_{V,pa(V)=\mathbf{v}} = P(V|pa(V)=\mathbf{v})$
- Data **d**:
  - d_1 : $\langle V_1=v_{1,1}, V_2=v_{2,1}, \ldots, V_n = v_{n,1}\rangle$
  - d_2 : $\langle V_1=v_{1,2}, V_2=v_{2,2}, \ldots, V_n = v_{n,2}\rangle$
  - ...
- Maximum likelihood:
  - Set $\theta_{V,pa(V)=\mathbf{v}}$ to the relative frequencies of the values of V given the values **v** of the parents of V

$$\theta_{V,pa(V)=\mathbf{v}} = \#(V,pa(V)=\mathbf{v}) \,/\, \#(pa(V)=\mathbf{v})$$

Exercise

对一个新的输入$A = 0, B = 0, C = 1$，朴素贝叶斯分类器将会怎样预测D?

A	B	C	D
0	0	1	0
0	1	0	0
1	1	0	0
0	0	1	1
1	1	1	1
1	0	0	1
1	1	0	1

Exercise: Candy example

- Prior $P(H) = \langle 0.1, 0.2, 0.4, 0.2, 0.1 \rangle$
- Evidence $d = \langle lime, cherry, lime \rangle$
- Make predictions using Bayesian, MAP and ML learning