CS3244 Notes

[2020-01-23 Thu]

Contents

1 Concept Learning

1.1 Concept Learning

Concept learning: a form of supervised learning, infer a concept from training examples

Concept c: a boolean-valued function over a set of input instances

Input instances X: each instance $x \in X$ is represented by a conjunction of input attributes

1.1.1 Hypothesis

Hypothesis space H: each hypothesis $h \in H$ ($h : X \to \{0, 1\}$) is represented by a conjunction of *constraints* on input attributes

- Constraint can be a specific value, don't care, or no value allowed

Trade-off between *expressive power* and *smaller hypothesis space* (larger space $\to$ usually more data needed)

- $ax + b$ is less expressive, smaller hypothesis space ($a \times b$)
- $ax^2 + bx + c$ is more expressive, larger hypothesis space ($a \times b \times c$)

1.1.2 Consistency and Satisfiability

Consistency: a hypothesis h is *consistent* with a set of training examples $D \Leftrightarrow h(x) = c(x)\ \forall \langle x, c(x) \rangle \in D$

Satisfiability: an input instance $x \in X$ *satisfies* a hypothesis $h \in H \Leftrightarrow h(x) = 1$

Note: *consistent* $\neq$ *satisfy*! One is $h(x) = c(x)$, one is $h(x) = 1$

1.1.3 Example of `EnjoySport`

Example	Sky	AirTemp	...	EnjoySport
1	Sunny	Warm		1
2	Sunny	Cold		1
3	Rainy	Warm		0
4	Sunny	Warm		1

- Input instances: `Sky` (Sunny, Cloudy, Rainy), `AirTemp` (Warm, Cold), `Humidity` (Normal, High), `Wind` (Strong, Weak), `Water` (Warm, Cool), `Forecast` (Same, Change)
- Target concept: `EnjoySport` (boolean valued)
- Example of hypothesis: $\langle Sky = Sunny, AirTemp =?, Wind = Strong, ... \rangle$

1.1.4 Concept Learning

Goal: given these, search for a hypothesis $h \in H$ that is *consistent* with D. Concept learning is search!

- Target concept/function $c : X \to \{0, 1\}$ (unknown)
- Noise-free training examples D

Inductive Learning Assumption: a hypothesis that approximates target function well over sufficiently large set of *training* examples ALSO approximates target function well over *unobserved* examples

- More data → more confidence

Example: hypothesis space H for `EnjoySport`

- 5x4x4x4x4x4 synthetically distinct hypotheses
- 4x3x3x3x3x3+1 semantically distinct hypotheses (a hypothesis with 1+ ϕ symbols is semantically the same, classifies everything as a negative example)

1.1.5 Exploit Structure in Concept Learning

h_j is more general than/equal to h_k ($h_j \geq_g h_k$) $\Leftrightarrow$ any input instance x that satisfies h_k also satisfies h_j

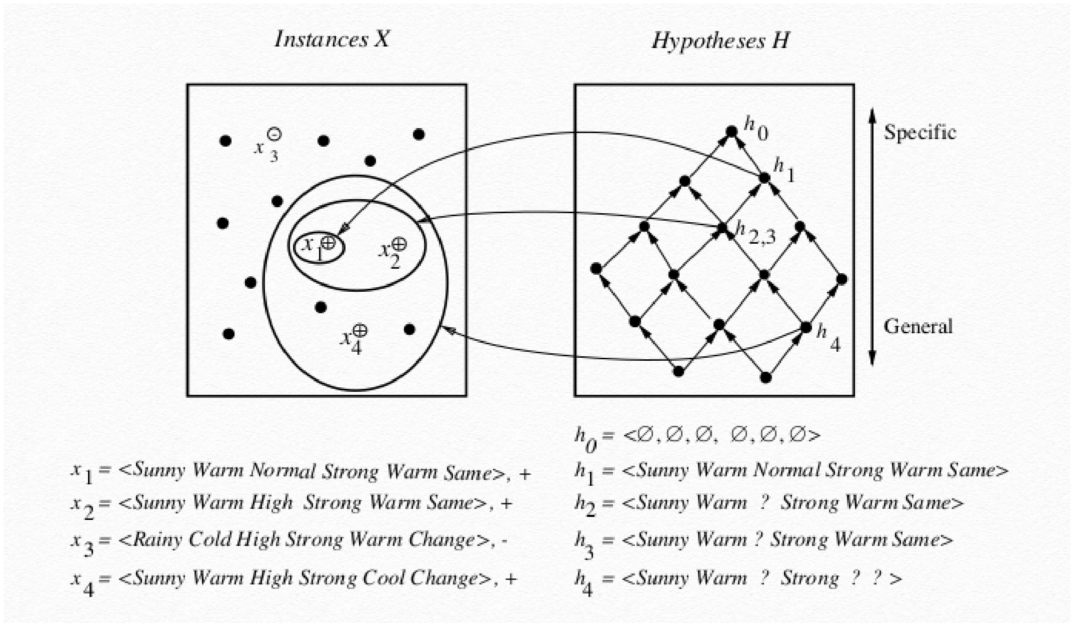
- $\forall x \in X \ (h_k(x) = 1) \rightarrow (h_j(x) = 1)$
- $\exists x \in X \ (h_k(x) = 1) \wedge (h_j(x) = 0)$ — the *negation*
- $\geq_g$ relation defines a partial order (reflexive, anti-symmetric, transitive) over H, but NOT a total order
- (Anti-symmetric: $x \geq y \wedge y \geq x \rightarrow x = y$)

1.2 FIND-S algorithm

Start with most specific hypothesis. Whenever it wrongly classifies +ve training example as –ve, "minimally" generalize to satisfy it.

1.2.1 Algorithm

1. Initialize h to most specific hypothesis in H
2. For each +ve training instance x:
   - For each attribute constraint a_i in h:
   - If x does not satisfiy constraint a_i, replace a_i in h by next more general constraint satisfied by x
3. Output hypothesis h



x_1 = <Sunny Warm Normal Strong Warm Same>, +
x_2 = <Sunny Warm High Strong Warm Same>, +
x_3 = <Rainy Cold High Strong Warm Change>, -
x_4 = <Sunny Warm High Strong Cool Change>, +

h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <Sunny Warm Normal Strong Warm Same>
h_2 = <Sunny Warm ? Strong Warm Same>
h_3 = <Sunny Warm ? Strong Warm Same>
h_4 = <Sunny Warm ? Strong ? ? >

Proposition 1: h is consistent with $D \Leftrightarrow$ every +ve training instance satisfies h, and every –ve training instance does not satisfy h.

1.2.2 Limitations

- Can't tell whether Find-S has learned target concept
- Can't tell when training examples are inconsistent (errors or noise)
- Picks a maximally specific h — but depending on H, there may be several (or none)!

1.3 Version Spaces

Version space $VS_{H,D}$ (wrt hypothesis space H and training examples D): subset of hypotheses from H that are consistent with D

- $VS_{H,D} = \{h \in H \mid h \text{ is consistent with } D\}$

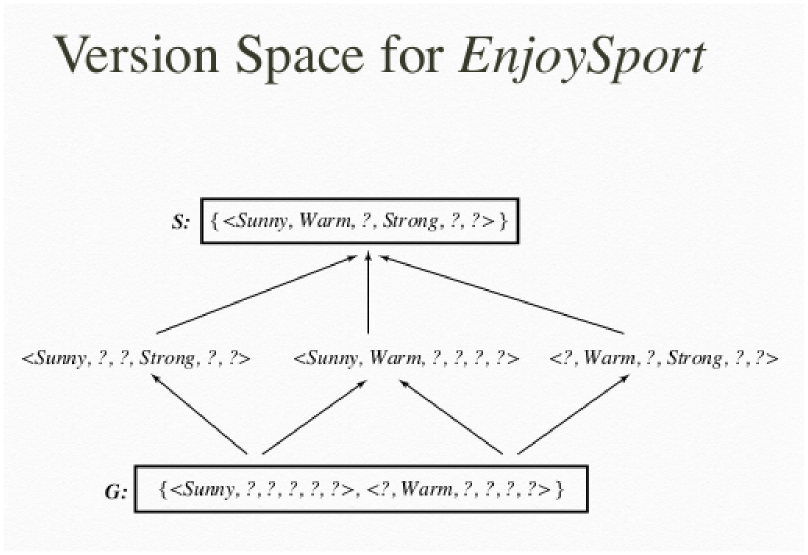
1.3.1 General and Specific Boundaries

General boundary G of $VS_{H,D}$: set of *maximally general* members of H consistent with D

- $G = \{g \in H \mid g \text{ consistent with } D \wedge (\nexists g' \in H \;\; g' >_g g \;\wedge\; g' \text{ consistent with } D\}$
- "Summary" of –ve training examples

Specific boundary S of $VS_{H,D}$: set of *maximally specific* members of H consistent with D

- $S = \{s \in H \mid s \text{ consistent with } D \wedge (\nexists s' \in H \;\; s >_g s' \;\wedge\; s' \text{ consistent with } D\}$
- "Summary" of +ve training examples


Version Space for *EnjoySport*
S: {<Sunny, Warm, ?, Strong, ?, ?>}
<Sunny, ?, ?, Strong, ?, ?>
<Sunny, Warm, ?, ?, ?, ?>
<?, Warm, ?, Strong, ?, ?>
G: {<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>}

Version Space Representation Theorem (VSRT): all hypotheses in the version space lie in some path from G to S

- $VS_{H,D} = \{h \in H \mid \exists s \in S, g \in G \;\; g \geq_g h \geq_g s\}$
- Proof omitted, see pg 21 of `2-ConceptLearning Part 1.pdf`

1.4 List-Then-Eliminate algorithm

Idea: List all possible hypotheses in H. Eliminate any hypothesis found inconsistent with any training example.

1.4.1 Algorithm

- *VersionSpace* $\leftarrow$ list of all hypotheses in H
- For each training example $\langle x, c(x) \rangle$:
  - Remove from *VersionSpace* any hypothesis h for which $h(x) \neq c(x)$ (inconsistent).

1.4.2 Limitation

Prohibitively expensive to exhaustively enumerate all hypotheses

1.5 Candidate-Elimination algorithm

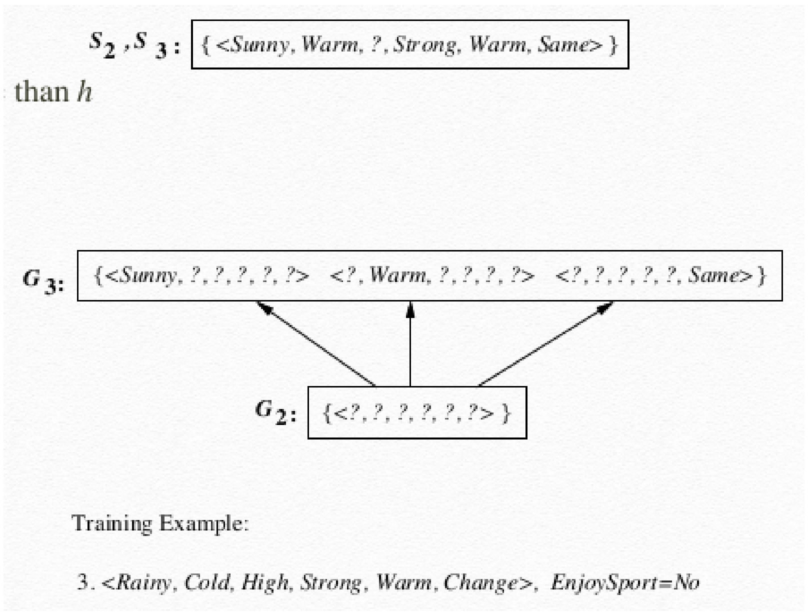
Idea: Start with most general and specific hypotheses. Each training example "minimally" generalizes S and specializes G to remove inconsistent hypotheses from version space.

- +ve training examples "bring down" S to be more general
- –ve training examples "raise up" G to be more specific

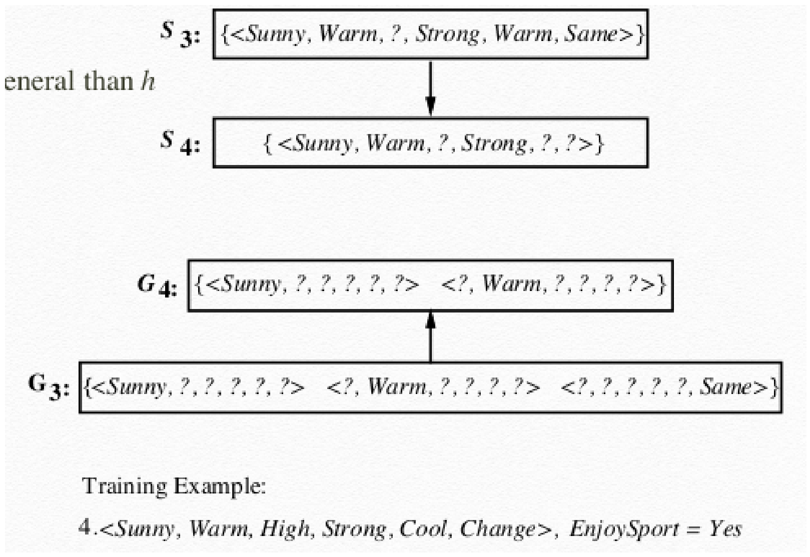
1.5.1 Algorithm

- $G \leftarrow$ maximally general hypotheses in H
- $S \leftarrow$ maximally specific hypotheses in H
- For each +ve training example d:
  - Remove from G any hypothesis inconsistent with d
  - For each $s \in S$ not consistent with d:
    - Remove s from S
    - Add all minimal generalizations h of s, such that h is consistent with d, and some member of G is more general than h
    - Finally, remove from S any hypothesis that is more general than another hypothesis in S (restore boundary property)
- For each –ve training example d:
  - Remove from S any hypothesis inconsistent with d
  - For each $g \in G$ not consistent with d:
    - Remove g from G
    - Add all minimal specializations h of s, such that h is consistent with d, and some member of S is more specific than h
    - Finally, remove from G any hypothesis that is more specific than another hypothesis in G (restore boundary property)

–ve example (specialises G)


S_2, S_3: {<Sunny, Warm, ?, Strong, Warm, Same>}
than h
G_3: {<Sunny, ?, ?, ?, ?, ?> <?, Warm, ?, ?, ?, ?> <?, ?, ?, ?, ?, Same>}
G_2: {<?, ?, ?, ?, ?, ?>}
Training Example:
3. <Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No

+ve example (generalises S, but also eliminates one from G)


S_3: {<Sunny, Warm, ?, Strong, Warm, Same>}
eneral than h
S_4: {<Sunny, Warm, ?, Strong, ?, ?>}
G_4: {<Sunny, ?, ?, ?, ?, ?> <?, Warm, ?, ?, ?, ?>}
G_3: {<Sunny, ?, ?, ?, ?, ?> <?, Warm, ?, ?, ?, ?> <?, ?, ?, ?, ?, Same>}
Training Example:
4.<Sunny, Warm, High, Strong, Cool, Change>, EnjoySport = Yes

1.5.2 Properties

- Cannot handle error/noise in training data
- Hypothesis space may be *biased*, if hypothesis representation is not expressive enough (such that target concept $c \notin H$), we can have no consistent hypotheses
- (For example: our current hypothesis representation cannot support multiple values—e.g. for *Sky* attribute, *{Sunny, Cloudy}* but excluding *Rainy*)

Active learner: if it can choose an input instance that satisfies exactly $\frac{1}{2}$ of the hypotheses in version space, then version space HALVES with each training example $\rightarrow \log_2$ training examples to find target concept c (fewer training examples needed!)

1.5.3 Classifying unobserved input instances

Can do so if we don't have an exact target concept, using the version space

Classifying with 100% confidence

- Proposition 3: an input instance x satisfies every member of $S \Leftrightarrow x$ satisfies every hypothesis in version space (surely +ve)
- Proposition 4: an input instance x satisfies none of the members of $G \Leftrightarrow x$ satisfies none of the hypotheses in version space (surely –ve)

Classifying with some confidence

- Take a majority vote of hypotheses in version space (assuming all are equally probable)

1.6 Unbiased Learner

Remember: our hypothesis space may be biased

Choose H that can express every teachable concept, i.e. $H = \mathcal{P}(X)$, so $|H| = 2^{|X|}$. Basically, consider every input instance individually.

Example: $\langle x_1, 1\rangle, \langle x_2, 1\rangle, \langle x_3, 1\rangle, \langle x_4, 0\rangle, \langle x_5, 0\rangle$

- $S \leftarrow \{(x_1 \vee x_2 \vee x_3)\}$
- $G \leftarrow \{\neg(x_4 \vee x_5)\}$

Limitation: Cannot classify new unobserved instances!

1.7 Inductive Bias

Inductive bias: inductive bias of a concept learning algorithm L is any minimal set of assertions B such that for any target concept c and corresponding noise-free training examples D_c,

$\forall x \in X \ (B \wedge D_c \wedge x) \models c(x) = L(x, D_c)$

Intuitively, it's the set of assumptions that allow you to generalise from training examples to the entire input instance space.

Inductive bias of Candidate-Elimination: $B = \{c \in H\}$

- Assumption: Candidate-Elimination outputs a classification $L(x, D_c)$ for input instance x if vote among hypotheses in VS_{H,D_c} is unanimously +ve or –ve, does not output a classification otherwise
- If $c \in H$, then $c \in VS_{H,D_c}$ since c is consistent with D_c (by definition of version space)
- ?

Inductive vs. deductive inference

Comparing learners with different inductive biases (weak $\rightarrow$ strong generalization power)

- Rote-Learner: store all examples, classify if it matches example
- Candidate-Elimination: $c \in H$
- Find-S: $c \in H$ and all instances are –ve unless training examples tell you otherwise

1.8 Summary

Concept learning is a search through our hypothesis space H. We can order H with a general-to-specific ordering, where boundaries G and S characterize the learner's uncertainty.

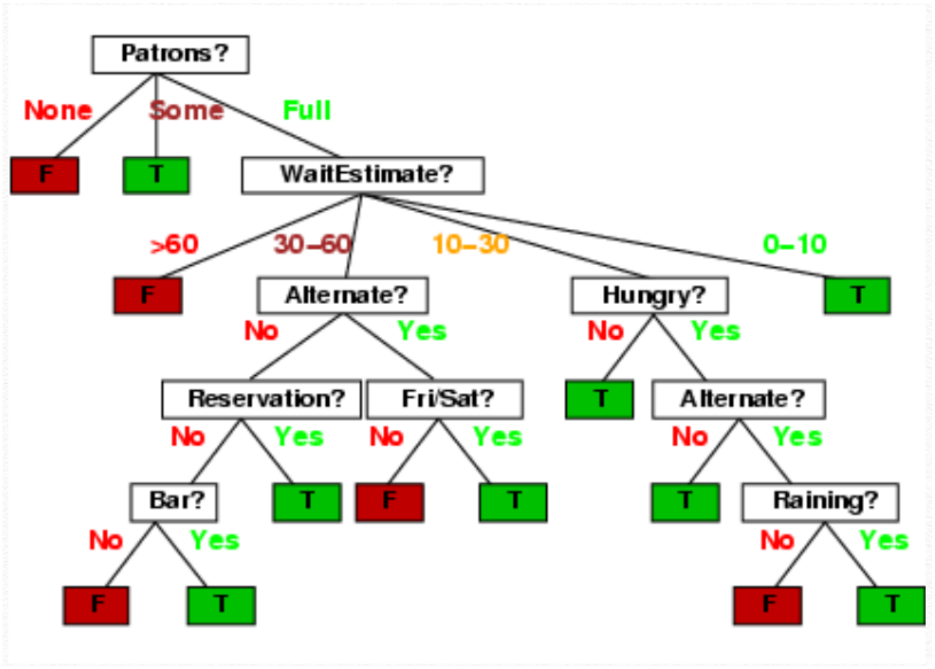
We've learned the Find-S and Candidate-Elimination algorithms.

Active learners can generative informative queries. (What should I learn next?)

Inductive biases, when stronger, can classify a greater proportion of unobserved input instances.

2 Decision Tree Learning

2.1 Decision Trees



2.1.1 Decision Tree Learning vs. Concept Learning

	Concept Learning	Decision Tree Learning
Target concept	Binary outputs	Discrete outputs
Training data	Noise-free	Robust to noise
Hypothesis space	Restricted (hard bias)	Complete and expressive
Search strategy	Complete: version space	Incomplete: prefers shorter tree (soft bias)
	Refine search per example	Refine search using all examples. No backtracking
Exploit structure	General-to-specific ordering	Simple-to-complex ordering

2.1.2 Expressive Power

Expressive power: decision tree learning can express *any* function of the input attributes!

- But we also want to find a *compact* decision tree
- To express a boolean decision tree, convert to disjunctive normal form, i.e. $C = Path_1 \vee Path_2 \vee \ldots$, where each path is a conjunction of attributes e.g. $(\neg A \wedge B \wedge \neg C \wedge ...)$

2.1.3 Hypothesis/Search Space

Number of distinct binary DTs with m boolean attributes

= number of boolean-valued functions

= number of distinct truth tables with 2^m rows

$= 2^{2^m}$

2.2 Decision-Tree-Learning algorithm

Goal: find a small tree consistent with training examples

Idea: greedily choose "most important" attribute as root of subtree

```
procedure DECISION-TREE-LEARNING($examples, attributes, parent\_examples$)
    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
        $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE($a, examples$)
        $tree \leftarrow$ a new decision tree with root test $A$
        for each value $v_k$ $of A$ do
            $exs \leftarrow \{e : e \in examples$ and $e.A = v_k\}$
            $subtree \leftarrow$ DECISION-TREE-LEARNING($exs, attributes - A, examples$)
            add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
    return $tree$
```

2.2.1 Plurality-Value

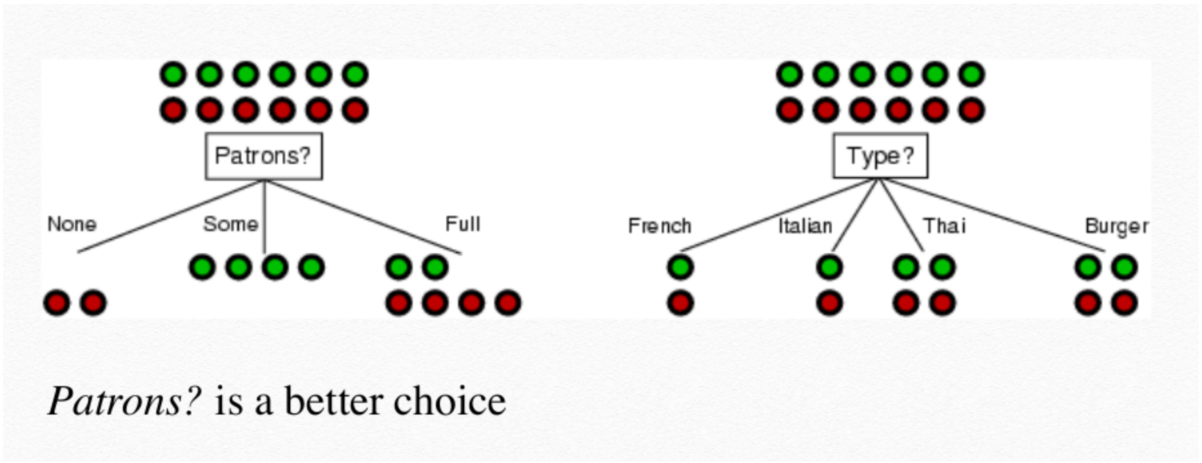
Plurality-value is just a majority vote. Allows you to still classify, even when:

- Data is noisy or wrong
- Domain is non-deterministic
- Not all attributes are accounted for

2.2.2 Importance: how to choose the most important attribute?

Idea: attribute should split the examples into subsets that are ideally all +ve, or all –ve

- Select attribute that *maximises information gain*, by reducing the most entropy (uncertainty)



- $Gain(Wait, Patrons) = B(\frac{6}{12}) - [\frac{2}{12}B(\frac{0}{2}) + \frac{4}{12}B(\frac{4}{4}) + \frac{6}{12}B(\frac{2}{6})] = 0.459 = 0.0541$ bits (??)
- $Gain(Wait, Type) = B(\frac{6}{12}) - [\frac{2}{12}B(\frac{1}{2}) + \frac{2}{12}B(\frac{1}{2}) + \frac{4}{12}B(\frac{1}{2}) + \frac{4}{12}B(\frac{1}{2}) = 0$ bits

Information gain: $Gain(C, A)$ of target concept C from attribute test on A is the expected reduction in entropy

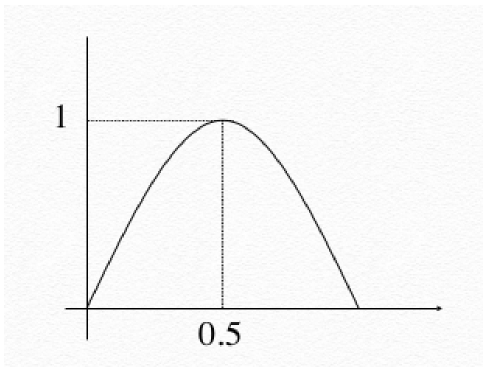
- Chosen attribute A divides training set E into subsets $E_1 \ldots E_d$ corresponding to the d distinct values of A. Each subset E_i has p_i +ve and n_i −ve examples

$$\begin{aligned} Gain(C, A) &= H(C) - H(C|A) \\ &= B(\frac{p}{p+n}) + \sum_{i=1}^{d} \frac{p_i + n_i}{p+n} B(\frac{p_i}{p_i + n_i}) \text{ (i.e. weighted average of the entropies of child nodes)} \end{aligned}$$

Entropy: measures the *uncertainty of classification*

- $H(C) = -\sum_{i=1}^{k} P(c_i) \log_2 P(C_i) = B(\frac{p}{p+n})$ where p is #+ve examples, n is #−ve examples
- $B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$ is entropy of boolean r.v. true with probability q

Entropy curve: idea is that $B(\frac{p}{p+n})$ is 0 when $\frac{p}{p+n} = 0/1$, is 1 when $\frac{p}{p+n} = 0.5$



2.2.3 Hypothesis Space Search

Decision-Tree-Learning is guided by the *information gain* heuristic (from the Importance function)

- It's a heuristic, so no guarantee that tree will actually be shortest

2.2.4 Inductive Bias

Inductive bias of Decision-Tree-Learning

1. Shorter trees are preferred
2. Trees that place *high information gain attributes close to the root* are preferred

If we only consider (1), then it is the *exact* inductive bias of BFS for the shortest consistent decision tree—prohibitively expensive

Note: bias is a *preference* for some hypotheses, not a *restriction* of the hypothesis space.

2.2.5 Occam's Razor

Occam's razor: prefer the shortest/simplest hypothesis that fits the data

- Simple hypothesis that fits data is unlikely to be a coincidence
- Complex hypothesis that fits data could be coincidence—could be overfitting
- (BUT) it could be wrong

2.3 Overfitting

Overfitting: hypothesis h *overfits* the set of training examples $D \Leftrightarrow \exists h' \in H \backslash \{h\}\ (error_D(h) < error_D(h')) \wedge (error_{D_X}(h) > error_{D_X}(h'))$

- $error_D(h)$ and $error_{D_X}(h)$ — errors of h over D and set D_X of examples corresponding to instance space X respectively
- i.e. overfitting occurs when there's another hypothesis that's worse on the training examples, but better on the instance space in general

Why does overfitting occur?

- Training examples are noisy or wrong
- Data is limited, but the target concept is complex

2.3.1 Avoiding overfitting

Approaches

- Stop growing DT when expanding a node is not statistically significant in improving performance over entire instance space—use hypothesis testing/chi-squared tests
- ($\star$) Allow DT to grow and overfit the data, then post-prune it

Partition training data into *training* dataset, and *validation* dataset ($\sim \frac{2}{3}/\frac{1}{3}$) — measure performance over both

Minimum description length, MDL: minimize $size(tree)$ and $size(misclassifications(tree))$

2.3.2 Reduced-Error Pruning

Idea: keep pruning (convert entire subtree under it to a leaf node), until further pruning is harmful

- Greedily prune the node that most improves *validation* set accuracy

2.3.3 Rule Post-Pruning

Convert DT into an equivalent set of *rules*, by creating one rule for each path from root $\rightarrow$ leaf

- e.g. IF $(Patrons = Full) \wedge (Hungry = Yes) \wedge (Type = Thai)$ THEN $(Wait = No)$

Prune (generalize) each rule by removing any precondition/conjunct that improves its estimated accuracy

Sort pruned rules by estimated accuracy into desired sequence used to classify unobserved input instances

2.4 Different Attribute Types

2.4.1 Continuous-Valued Attributes

Solution: make it *discrete*! Define a *discrete*-valued input attribute to *partition* the values into set of intervals for testing

- e.g. $WaitEstimate$: $0 - 10, 10 - 30, 30 - 60, > 60$

2.4.2 Attributes with Many Values

Problem: *Gain* has a preference for attributes with many values, e.g. *Date* has 365 values $\rightarrow$ each value might have only 1 training example $\rightarrow$ not helpful in classifying

Solution: $Gain \rightarrow GainRatio(C, A) = \frac{Gain(C,A)}{SplitInformation(C,A)}$, i.e. normalize gain by split information

- $SplitInformation(C, A) = -\sum_{i=1}^{d} \frac{|E_i|}{|E|} \log_2 \frac{|E_i|}{|E|}$
- i.e. split information measures how distributed the training set subsets E_i are

2.4.3 Attributes with Differing Costs

Problem: suppose each attribute test comes with a given cost—how to learn a DT that has low expected cost?

Solution: $Gain \rightarrow \frac{Gain^2(C,A)}{Cost(A)}$ or $\frac{2^{Gain(C,A)}-1}{(Cost(A)+1)^w}$, where $w \in [0, 1]$ determines importance of cost

2.4.4 Missing Attribute Values

Problem: what if some training examples are missing values for A?

Solution: use training example anyway and sort through DT

- If node n tests A, assign the *most common value* of A among other examples sorted to node n
- Assign most common value of A among other examples sorted to node n with *same value of output/target concept*
- Assign *probability* p_i to each possible value of A, then assign *fraction* p_i of example to each descendant in DT

Classify new unobserved input instances with missing attribute values in same manner

2.5 Summary

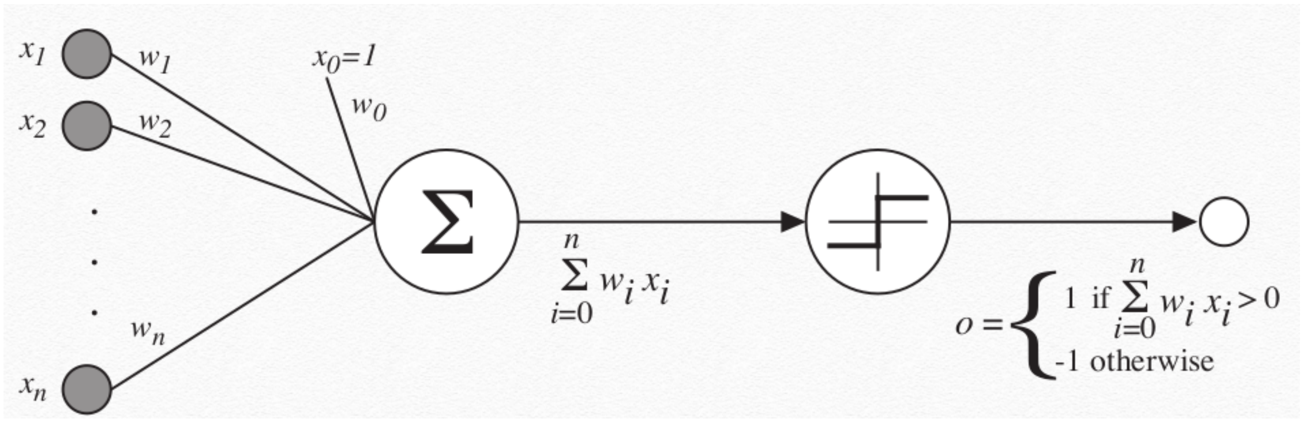
- Decision tree learning uses *information gain*
- Overfitting comes from having noisy/limited training data, can be avoided with post-pruning
- Extensions to Decision-Tree-Learning: attributes with continuous/missing/many values, and differing costs

3 Neural Networks

3.1 Neural Nets

	Decision Tree Learning	**Neural Nets**
Target function/output	Discrete	Discrete/real vector
Input instance	Discrete	Discrete/real, high-dimensional
Training data	Robust to noise	Robust to noise
Hypothesis space	Complete, expressive	Restricted (#hidden units–hard bias), expressive
Search strategy	Incomplete: prefer shorter tree (soft bias)	Incomplete: prefer smaller weights (soft bias)
	Refine search using all examples	Gradient descent/ascent
	No backtracking	Batch mode/stochastic
Training time	Short	Long
Prediction time	Fast	Fast
Interpretability	White-box	Black-box

3.2 Perceptron


x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$

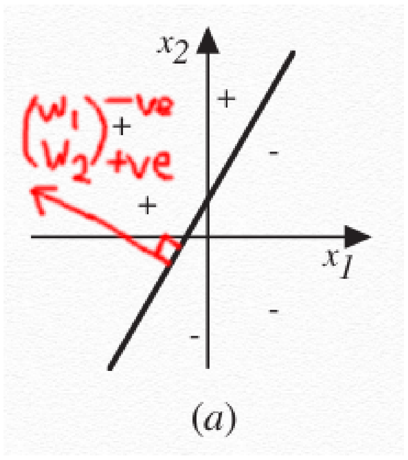
$$o(\mathbf{x}) = sgn(\mathbf{w} \cdot \mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron calculates a linear combination of a vector of real-valued inputs, and runs through step function

- Inputs: $\mathbf{x} = (x_0, \ldots, x_n)^T$ (x_0 is bias input)
- Weights: $\mathbf{w} = (w_0, \ldots, w_n)^T$ (w_0 is bias weight)
- Linear sum: $\mathbf{w} \cdot \mathbf{x} = \sum_{i=0}^{n} w_i x_i$
- Activation function: step function (1 if +ve, –1 if –ve)
- Hypothesis space: $H = \{\mathbf{w} \mid \mathbf{w} \in \mathbb{R}^{n+1}\}$

3.2.1 Decision Surface of Perceptron

- Input vector: $\mathbf{x} = (1, x_1, x_2)^T$
- Weight vector: $\mathbf{w} = (w_0, w_1, w_2)^T$
- Decision surface/line: hyperplane represented by $\mathbf{w} \cdot \mathbf{x} = 0 \Rightarrow x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$



<u>Orthogonal vector</u>

- ($\star$) Always points to the +ve examples
- Sign of w_n: if it points in same direction as x_n axis, it's +ve, else it's –ve
- Sign of w_0: depends on slope of line, and whether it's above/below 0 (check from equation)

<u>What if function is not linearly separable?</u>

- E.g. $XOR(x_1, x_2)$
- Note: every boolean function can be represented with neural network with 2 layers (only 1 hidden layer)

3.3 Perceptron Training Rule

<u>Idea</u>: initialize *randomly*. Iterate through every training example, and apply the training rule to each training example until $\mathbf{w}$ is consistent:

$$w_i \leftarrow w_i + \Delta w_i, \quad \Delta w_i = \eta(t-o)x_i$$

- $t = c(\mathbf{x})$: target output for training example $\langle \mathbf{x}, c(\mathbf{x}) \rangle$
- $o = o(\mathbf{x})$: perceptron output
- η is *learning rate*: small +ve constant (usually decays over time)
- Intuitively, $(t-o)$ measures the misclassification

3.3.1 Why does the training rule work?

Assume all x_i are +ve.

- If t is +ve and o is –ve, then $t-o$ is +ve $\Rightarrow \Delta w_i$ is +ve $\Rightarrow w_i \uparrow \Rightarrow \mathbf{w}\cdot\mathbf{x} \uparrow \Rightarrow$ makes o more +ve
- If t is –ve and o is +ve, then $t-o$ is –ve $\Rightarrow \Delta w_i$ is –ve $\Rightarrow w_i \downarrow \Rightarrow \mathbf{w}\cdot\mathbf{x} \downarrow \Rightarrow$ makes o more –ve

($\star$) Guaranteed to converge if training examples are *linearly separable* and η is *sufficiently small*!

<u>Problem</u>: if training examples are not linearly separable, can fail to converge!

3.4 Linear Unit Training Rule

Idea: Search H to find weight vector that converges to best-fit approximation for the training examples, even if they're linearly non-separable: we learn $\mathbf{w}$ that minimizes the loss function L_D

Linear unit: $o = \mathbf{w} \cdot \mathbf{x}$ (let's consider an *unthresholded* perceptron here)

Loss function (s.s.e): $L_D(\mathbf{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$

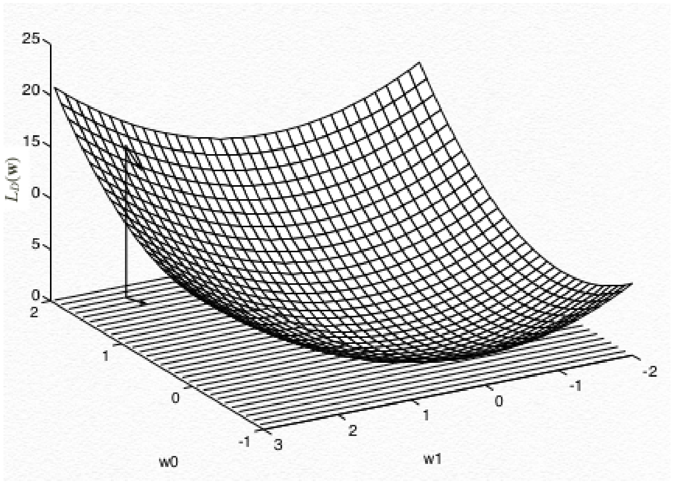
- D: set of training examples
- t_d and o_d: target output and output of linear unit for training example d

3.4.1 Gradient Descent

Idea: Find $\mathbf{w}$ that minimizes L by repeatedly updating it in the direction of steepest descent

Gradient: $\nabla L_D(\mathbf{w}) = [\frac{\partial L_D}{\partial w_0}, \frac{\partial L_D}{\partial w_1}, \ldots, \frac{\partial L_D}{\partial w_n}]$

Training rule: $w_i \leftarrow w_i + \Delta w_i, \quad \Delta w_i = -\eta \frac{\partial L_D}{\partial w_i} \quad \equiv \quad \mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}, \quad \Delta \mathbf{w} = -\eta \nabla L_D(\mathbf{w})$



$$\frac{\partial L_D}{\partial w_i} = -\sum_{d \in D}(t_d - o_d)x_{id} \quad \equiv \quad \nabla L_D(\mathbf{w}) = -\sum_{d \in D}(t_d - o_d)\mathbf{x}_d$$

$$\Delta w_i = -\eta \frac{\partial L_D}{\partial w_i} = \eta \sum_{d \in D}(t_d - o_d)x_{id} \quad \equiv \quad \Delta \mathbf{w} = -\eta \nabla L_D(\mathbf{w}) = \eta \sum_{d \in D}(t_d - o_d)\mathbf{x}_d$$

Derivation

$$\begin{aligned}
\frac{\partial L_D}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d \in D}(t_d - o_d)^2 \\
&= \sum_{d \in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \mathbf{w} \cdot \mathbf{x}_d) \\
&= -\sum_{d \in D}(t_d - o_d)x_{id} \\
\nabla L_D(\mathbf{w}) &= -\sum_{d \in D}(t_d - o_d)\mathbf{x}_d
\end{aligned}$$

3.4.2 Gradient Descent Algorithm

```
GRADIENT-DESCENT($D, \eta$)
  • Initialise $\mathbf{w}$ randomly
  • Repeatedly apply linear unit training rule until satisfied:
      – Initialise $\Delta\mathbf{w} \leftarrow 0$
      – For each $d \in D$:
          1. Input instance $\mathbf{x}_d$ to linear unit and compute output $o$
          2. Compute $\Delta\mathbf{w} \leftarrow \Delta\mathbf{w} + \eta(t - o)\mathbf{x}_d$
      – Compute $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
```

3.4.3 Stochastic Gradient Descent

Batch gradient descent: loss defined over ALL training examples. Do until satisfied:

- Compute gradient $\nabla L_D(\mathbf{w}) = -\sum_{d \in D}(t_d - o_d)\mathbf{x}_d$
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L_D(\mathbf{w})$ where $L_D(\mathbf{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$

Stochastic gradient descent: loss defined over EACH training example. Do until satisfied, for each d:

- Compute gradient $\nabla L_d(\mathbf{w}) = -(t_d - o_d)\mathbf{x}_d$
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L_d(\mathbf{w})$ where $L_d(\mathbf{w}) = \frac{1}{2}(t_d - o_d)^2$

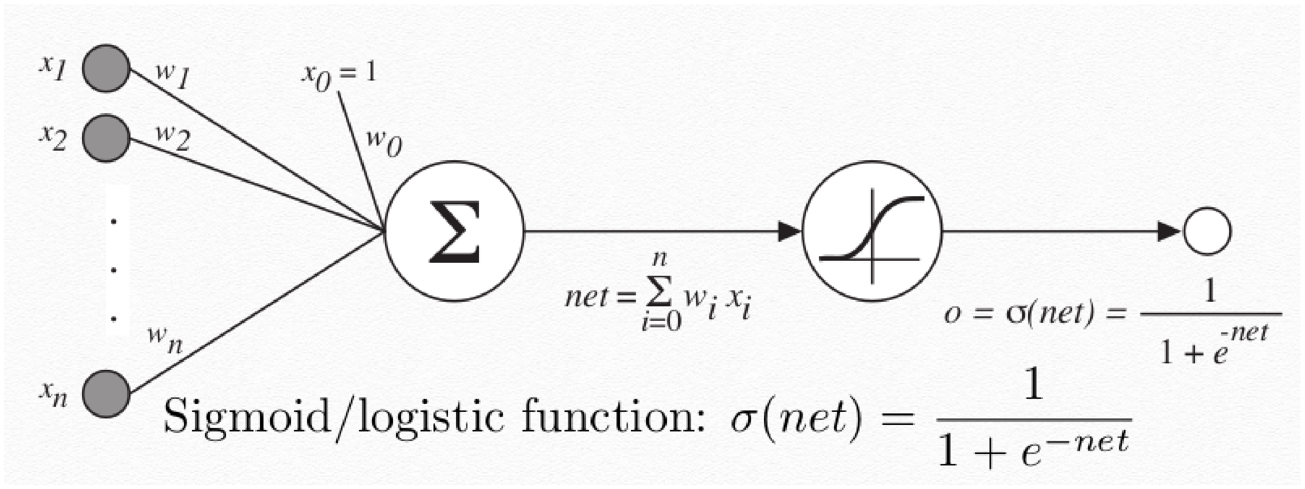
SGD will approximate batch GD arbitrarily closely if η is sufficiently small!

- Stochastic gradient is an unbiased estimator of the true gradient: $E[\nabla L_d(\mathbf{w})] = \nabla L_D(\mathbf{w})$

Why is SGD useful?

- Lower computational cost: does not use ALL training examples at once
- "Anytime" performance: can stop the computation at any time and still get some performance
- Economic cost: buying data in small batches
- Helps to escape *local minima*

3.5 Sigmoid Unit



x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$
Sigmoid/logistic function: $\sigma(net) = \frac{1}{1+e^{-net}}$

- Linear sum: $net = \sum_{i=0}^{n} w_i \cdot x_i = \mathbf{w} \cdot \mathbf{x}$
- Sigmoid function: $o = \sigma(net) = \frac{1}{1+e^{-net}}$
- $\frac{\mathrm{d}\sigma(net)}{\mathrm{d}net} = \sigma(net)(1-\sigma(net)) = o(1-o)$

3.5.1 Gradient Descent Rules

$$\frac{\partial L_D}{\partial w_i} = -\sum_{d\in D}(t_d - o_d)o_d(1-o_d)x_{id} \quad \equiv \quad \nabla L_D(\mathbf{w}) = -\sum_{d\in D}(t_d - o_d)o_d(1-o_d)\mathbf{x}_d$$

$$\Delta w_i = -\eta\frac{\partial L_D}{\partial w_i} \quad \equiv \quad \Delta\mathbf{w} = -\eta\nabla L_D(\mathbf{w})$$

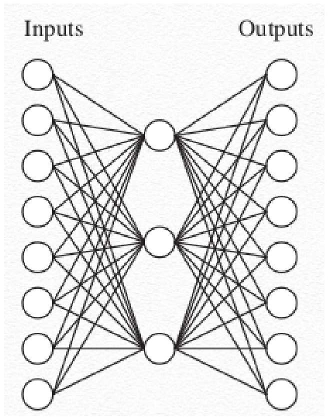
Derivation

$$\begin{aligned}
\frac{\partial L_D}{\partial w_i} &= \frac{\partial}{\partial w_i}\left(\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2\right) \\
&= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d\in D}(t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right) \\
&= -\sum_{d\in D}(t_d - o_d)\frac{\partial o_d}{\partial net_d}\frac{\partial net_d}{\partial w_i} \\
&= -\sum_{d\in D}(t_d - o_d)o_d(1-o_d)x_{id}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial o_d}{\partial net_d} &= \frac{\partial\sigma(net_d)}{\partial net_d} = o_d(1-o_d) \\
\frac{\partial net_d}{\partial w_i} &= \frac{\partial(\mathbf{w}\cdot\mathbf{x}_d)}{\partial w_i} = x_{id}
\end{aligned}$$

3.6 Multilayer Networks

2 layers with multiple outputs: Input layer (NO neurons) $\rightarrow$ Hidden layer (sigmoid) $\rightarrow$ Output layer (sigmoid)



$$L_D(\mathbf{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in K}(t_{kd} - o_{kd})^2 \text{ where } K \text{ is the set of output units}$$

3.6.1 Backpropagation Algorithm

```
BACKPROPAGATION($D, \eta$)
  • Initialise $\mathbf{w}$ randomly
  • Repeatedly do until satisfied:
    – For each $d \in D$:
      1. Input instance $\mathbf{w}_d$ into the network and compute every hidden output $o_h$ and output $o_k$
      2. For each output unit $k$, compute error $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
      3. For each hidden unit $h$, compute error $\delta_h \leftarrow o_h(1 - o_h)\sum_{k\in K} w_{hk}\delta_k$
      4. Update each weight $w_{hk} \leftarrow w_{hk} + \Delta w_{hk}$ where $\Delta w_{hk} = \eta\delta_k o_h$
      5. Update each weight $w_{ih} \leftarrow w_{ih} + \Delta w_{ih}$ where $\Delta w_{ih} = \eta\delta_k x_i$
```

3.6.2 Derivation of Backpropagation

We want to find Δw_{hk} and Δw_{ih}.

$$\begin{aligned}
\frac{\partial L_d}{\partial w_{hk}} &= \frac{\partial L_d}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial w_{hk}} \quad \text{where } net_k = \sum_{h'\in H} w_{h'k}o_{h'} \\
\frac{\partial L_d}{\partial o_k} &= \frac{\partial}{\partial o_k}\frac{1}{2}\sum_{k'\in K}(t_{k'} - o_{k'})^2 = -(t_k - o_k) \\
\frac{\partial o_k}{\partial net_k} &= \frac{\partial \sigma(net_k)}{\partial net_k} = o_k(1 - o_k) \\
\frac{\partial net_k}{\partial w_{hk}} &= o_k \\
\therefore \Delta w_{hk} &= -\eta\frac{\partial L_d}{\partial w_{hk}} = \eta\delta_k o_h \quad \text{where } \delta_k = (t_k - o_k)o_k(1 - o_k)
\end{aligned}$$

3.6.3 Remarks on Backpropagation

- Multiple local minima for L_D, so GD does not necessarily converge to global minimum. In practice, GD often performs well, especially after multiple random initialisations of $\mathbf{w}$
- Weight momentum $\alpha \in [0,1]$ often included: $\Delta w_{hk} \leftarrow \eta\delta_k o_h + \alpha\Delta w_{hk}, \quad \Delta w_{ih} \leftarrow \eta\delta_h x_i + \alpha\Delta w_{ih}$
- Generalisable to networks of arbitrary depth
  - Step 3: Let K be all units in *next* layer, whose inputs include output of h
  - Step 5: Let x_i be output of unit i in *previous* layer, that is an input to h
- Expressive hypothesis space
  - Every *boolean* function can be represented by a network with 1 hidden layer! (But could take exponential number of hidden units, in number of inputs)
  - Every *bounded continuous* function can be approximated by network with 1 hidden layer
  - *Any* function can be approximated by network with 2 hidden layers
- Approximate inductive bias: smooth interpolation between data points

3.7 Alternative Loss/Error Functions

- Penalize large weights:

$$L_D(\mathbf{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in K}(t_{kd} - o_{kd})^2 + \gamma\sum_{j,l} w_{jl}^2$$

- Train on target values as well as slopes:

$$L_D(\mathbf{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in K}\left[(t_{kd} - o_{kd})^2 + \mu\sum_{i=1}^{n}(\frac{\partial t_{kd}}{\partial x_{id}} - \frac{\partial o_{kd}}{\partial x_{id}})^2\right]$$

- Tie together weights

4 Bayesian Inference

4.1 Why Bayesian Inference?

4.1.1 Bayesian Inference

- Allows *prior* knowledge to be combined with *observed data* to give a *probabilistic prediction*
- Allows new input instance to be classified by *combining predictions of multiple hypotheses* weighted by their beliefs
- *Incrementally updates belief* of hypothesis with each training example
- Useful *conceptual framework*: provides "gold standard" to evaluate other learning algorithms

4.1.2 Importance of Bayesian Learning Algorithms

- They calculate explicit probabilities for hypotheses (e.g. naive Bayes classifier), and are practical and effective for some problems
- They provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities
  - Analyse the conditions under which FIND-S and CANDIDATE-ELIMINATION output the most probable hypothesis given training data
  - Neural networks: choosing to minimise SSE when searching space of neural networks, but choosing cross-entropy when learning target functions that predict probabilities
  - Decision trees: analyse inductive bias that favour short trees

4.2 Bayes' Theorem

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Idea: update *prior* belief to *posterior* belief, given data D

- $P(h)$: *prior* belief of hypothesis h
- $P(D|h)$: likelihood of data D given h
- $P(D) = \sum_{h \in H} P(D|h)P(h)$: marginal likelihood/evidence of D
- $P(h|D)$: *posterior* belief of h given D

4.2.1 Limitations of Bayes' Theorem

Requires specifying probabilities and underlying distributions

- Some priors never occur (e.g. uniform prior, where all hypotheses have equal probability—could be incorrect)
- Likelihood function could be chosen wrongly. Conjugate priors
- Not enough data to substantiate a prior—how do we know which prior to assume?

Often prohibitively expensive to compute evidence

- How to get $P(D)$ easily, especially when hypothesis space is large?
- Solutions: approximate inference, variational inference

4.2.2 Maximum *a posteriori* (MAP) hypothesis

MAP: the most probable hypothesis given the training data, i.e. gives the highest unnormalized posterior

$$h_{MAP} = \underset{h \in H}{\arg\max}\, P(h|D) = \underset{h \in H}{\arg\max}\, P(D|h)P(h)$$

- Unnormalized posterior: $P(D|h)P(h)$

4.2.3 Maximum likelihood (ML) hypothesis

ML: similar to MAP, but assume all hypotheses have equal probability

$$h_{ML} = \underset{h \in H}{\arg\max}\, P(D|h)$$

- Likelihood (of the data given h): $P(D|h)$

4.3 Example: Medical Diagnosis

Medical test for cancer

- $P(+|cancer) = 0.98$
- $P(-|\neg cancer) = 0.97$
- $P(cancer) = 0.008$

Unnormalised posteriors

- $P(+|cancer)P(cancer) = 0.98 \times 0.008 = 0.00784$
- $P(+|\neg cancer)P(\neg cancer) = 0.03 \times 0.992 = 0.02976$
- $\therefore h_{MAP} = \neg cancer$
- $P(cancer|+) = \frac{0.00784}{0.00784+0.02976} = 0.20851$

4.4 Basic Probability Formulas

4.4.1 Chain Rule for Probability

Joint probability of a conjunction of events A_1 to A_i is the product:

$$P(A_1, \ldots, A_n) = \prod_{i=1}^{n} P(A_i|A_1, \ldots, A_{i-1})$$

4.4.2 Inclusion-Exclusion Principle

Probability of a union of events can be expressed as sums of joint probabilities

$$P(\cup_{i=1}^{n} A_i) = \sum_{1 \le i \le n} P(A_i) - \sum_{1 \le i < j \le n} P(A_i, A_j) + ... + (-1)^{n-1} P(A_1, \dots, A_n)$$

4.4.3 Marginalization

If $A_1, \dots, A_n$ are mutually exclusive and form a partition, then $P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$

4.5 Brute-Force MAP Hypothesis Learning Algorithm

1. For each hypothesis $h \in H$, calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with highest posterior probability

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

4.5.1 Requirements

We need to choose:

- $P(D|h)$
- $P(h)$

...and that will be sufficient to determine $P(D)$.

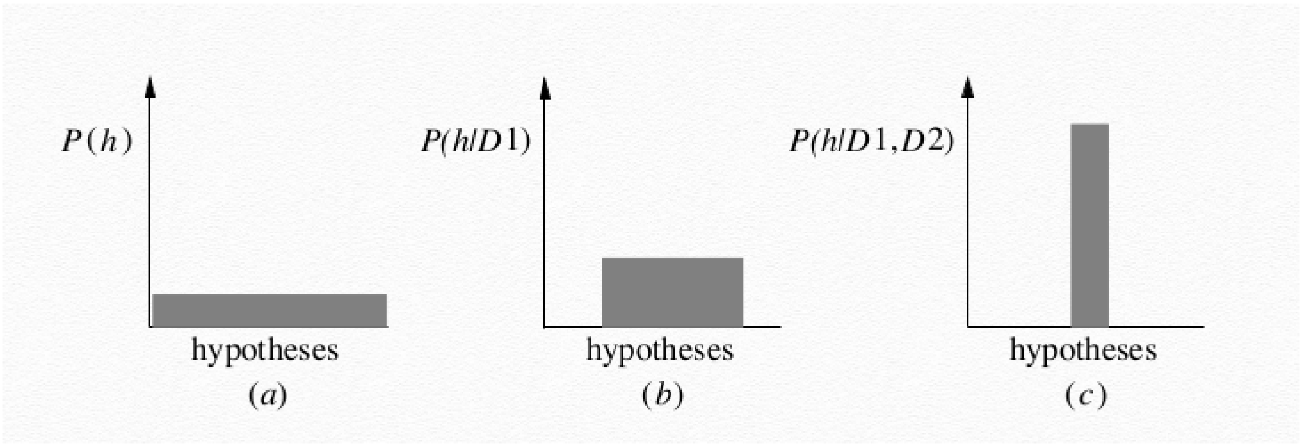
4.6 Bayesian Inference: Concept Learning

Let's now relate the the brute-force MAP learning algorithm to concept learning. We choose:

- $P(D|h)$: deterministic likelihood (0 or 1), depending on whether h is consistent with D
- $P(h)$: uniform distribution

$$\begin{aligned} P(D|h) &= \begin{cases} 1 & \text{if } h \text{ is consistent with D, i.e. } c(\mathbf{x}_d) = h(\mathbf{x}_d)\ \forall d \in D \\ 0 & \text{otherwise} \end{cases} \\ P(h) &= \frac{1}{|H|} \\ \therefore P(h|D) &= \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with D} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

($\star$) Every consistent hypothesis is a MAP hypothesis! Since all consistent hypotheses are equally likely.



4.7 Bayesian Inference: Learning a Continuous-Valued Function

($\star$) Under certain assumptions, a learning algorithm trying to learn a continuous-valued target function that *minimizes SSE* (between hypothesis predictions and training data) will give the ML hypothesis.

4.7.1 Setup

- Target function f is continuous
- Training examples $D = \{\langle \mathbf{x}_d, t_d \rangle\}$ are fixed
- Output t_d is noisy, where errors are normally distributed

$$\text{Let } t_d = f(\mathbf{x}_d) + \epsilon_d, \text{ where } \epsilon_d \sim N(0, \sigma^2)$$
$$\text{Then } h_{ML} = \arg\min_{h \in H} \frac{1}{2} \sum_{h \in D} (t_d - h(\mathbf{x}_d))^2$$

4.7.2 Derivation

$$\begin{aligned}
h_{ML} &= \arg\max_{h \in H} p(D|h) \\
&= \arg\max_{h \in H} \prod_{d \in D} p(t_d | h, \mathbf{x}_d) \text{ (assuming training examples are mutually independent)} \\
&= \arg\max_{h \in H} \prod_{d \in D} \frac{1}{\sqrt{2\pi}\sigma} \; exp(-\frac{(t_d - h(\mathbf{x}_d))^2}{2\sigma^2}) \\
&= \arg\max_{h \in H} \sum_{d \in D} \ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{(t_d - h(\mathbf{x}_d))^2}{2\sigma^2} \\
&= \arg\max_{h \in H} \sum_{d \in D} -\frac{(t_d - h(\mathbf{x}_d))^2}{2\sigma^2} \\
&= \arg\max_{h \in H} \frac{1}{2} \sum_{d \in D} -(t_d - h(\mathbf{x}_d))^2 \\
&= \arg\min_{h \in H} \frac{1}{2} \sum_{d \in D} (t_d - h(\mathbf{x}_d))^2
\end{aligned}$$

4.8 Bayesian Inference: Learning to Predict Probabilities

($\star$) Under certain assumptions, a learning algorithm trying to predict probabilities that *minimizes cross-entropy* will give the ML hypothesis.

4.8.1 Setup

- Non-deterministic concept c, giving 0 or 1 with some probability
- Target function f is the *probability* that $c(x) = 1$, i.e. $f(x) = P(c(x) = 1)$
- Training examples $D = \{\langle \mathbf{x}_d, t_d \rangle\}$ where $t_d = c(\mathbf{x}_d)$

Example 1: $\mathbf{x}$ refers to patient's symptoms; $c(\mathbf{x}_d) = 1$ if patient survives, 0 if patient dies

Example 2: $\mathbf{x}$ refers to loan applicant's history; $c(\mathbf{x}_d) = 1$ if loan repaid, 0 if loan not repaid

$$\text{Let } t_d = c(\mathbf{x}_d), \text{ where } c(\mathbf{x}_d) = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1-p \end{cases} \text{ and } f(\mathbf{x}_d) = P(c(\mathbf{x}_d) = 1) \text{ i.e. } P(t_d = 1 | \mathbf{x}_d)$$

$$\text{Then } h_{ML} = \arg\max_{h \in H} \sum_{d \in D} t_d \ln h(\mathbf{x}_d) + (1 - t_d) \ln(1 - h(\mathbf{x}_d))$$

4.8.2 Derivation

(Note: $\mathbf{x}_d$ is no longer fixed, it is unknown and now a random variable — we assume it's independent of h. We could make the simplifying assumption that training examples are fixed, but the final outcome is still the same.)

$$\begin{aligned}
P(D|h) &= \prod_{d \in D} P(\mathbf{x}_d, t_d | h) \\
&= \prod_{d \in D} P(t_d | h, \mathbf{x}_d) P(\mathbf{x}_d) \\
P(t_d | h, \mathbf{x}_d) &= \begin{cases} h(\mathbf{x}_d) & \text{if } t_d = 1 \\ 1 - h(\mathbf{x}_d) & \text{if } t_d = 0 \end{cases} \\
&= h(\mathbf{x}_d)^{t_d} (1 - h(\mathbf{x}_d))^{1 - t_d} \\
h_{ML} &= \arg\max_{h \in H} P(D|h) \\
&= \arg\max_{h \in H} \prod_{d \in D} h(\mathbf{x}_d)^{t_d} (1 - h(\mathbf{x}_d))^{1 - t_d} P(\mathbf{x}_d) \\
&= \arg\max_{h \in H} \prod_{d \in D} h(\mathbf{x}_d)^{t_d} (1 - h(\mathbf{x}_d))^{1 - t_d} \\
&= \arg\max_{h \in H} \sum_{d \in D} t_d \ln h(\mathbf{x}_d) + (1 - t_d) \ln(1 - h(\mathbf{x}_d))
\end{aligned}$$

4.8.3 Gradient *Ascent* to Maximize Likelihood in a Sigmoid Unit

- $U_D(h)$ refers to that thing previously for a given hypothesis h.
- Gradient ascent because we want to maximize, not minimize

$$\begin{aligned}
U_D &= \sum_{d\in D} t_d \ln h(\mathbf{x}_d) + (1-t_d)\ln(1-h(\mathbf{x}_d)) \text{ (cross-entropy, but not negative)}\\
\frac{\partial U_D}{\partial w_i} &= \sum_{d\in D} \frac{\partial U_D}{\partial h(\mathbf{x}_d)} \frac{\partial h(\mathbf{x}_d)}{\partial w_i}\\
&= \sum_{d\in D} \frac{\partial t_d \ln h(\mathbf{x}_d) + (1-t_d)\ln(1-h(\mathbf{x}_d))}{\partial h(\mathbf{x}_d)} \frac{\partial h(\mathbf{x}_d)}{\partial w_i}\\
&= \sum_{d\in D} \frac{t_d - h(\mathbf{x}_d)}{h(\mathbf{x}_d)(1-h(\mathbf{x}_d))} h(\mathbf{x}_d)(1-h(\mathbf{x}_d))x_{id}\\
&= \sum_{d\in D} (t_d - h(\mathbf{x}_d))x_{id}\\
w_i &\leftarrow w_i + \Delta w_i \text{ where } \Delta w_i = \eta \frac{\partial U_D}{\partial w_i}
\end{aligned}$$

(Note: even if the model outputs a high probability, it doesn't mean that the model is confident in its prediction!)

4.9 Minimum Description Length (MDL) Principle

Occam's Razor: choose the shortest explanation for the observed data

$$\begin{aligned}
h_{MAP} &= \arg\max_{h\in H} P(D|h)P(h)\\
&= \arg\max_{h\in H} \log_2 P(D|h) + \log_2 P(h)\\
&= \arg\min_{h\in H} -\log_2 P(D|h) - \log_2 P(h)
\end{aligned}$$

Information theory: optimal (shortest expected description length) code for message with probability p is $-\log_2 p$ bits

- $-\log_2 P(h)$: description length of h under optimal code
- $-\log_2 P(D|h)$: description length of D given h under optimal code

4.9.1 Minimum Description Length (MDL)

$$h_{MDL} = \arg\min_{h\in H} L_{C_1}(h) + L_{C_2}(D|h)$$

where $L_C(x)$ is description length of x under encoding scheme C

Example: H = decision trees

- $L_{C_1}(h)$ is #bits to describe tree h
- $L_{C_2}(D|h)$ is #bits to describe D given h
  - $L_{C_2}(D|h) = 0$ if examples classified perfectly by h

  - Otherwise, only misclassifications need to be described

- Idea: minimize *length(tree)* and *length(misclassifications(tree))*
  - h_{MDL} trades off tree size for training errors, to mitigate overfitting

4.10 Most Probable Classification of New Instances

Given new instance $\mathbf{x}$, what is its *most probable classification* given the new training data D?

- h_{MAP} is the most probable hypothesis, but not the most probable classification!
- We need to find t that maximises $P(t|D)$, not h that maximises $P(h|D)$!

Example

- Consider 3 possible hypotheses: h_1, h_2, h_3; let classifications $T = \{+, -\}$
- $P(h_1|D) = 0.4$, $P(h_2|D) = 0.3$, $P(h_3|D) = 0.3$
- $h_1(\mathbf{x}) = +$, $h_2(\mathbf{x}) = -$, $h_3(\mathbf{x}) = -$
- Most probable hypothesis $h_{MAP} = h_1$, but most probable classification is $-$:
  $P(-|D) = 0.6 > P(+|D) = 0.4$!

4.10.1 Bayes-Optimal Classifier

$$\arg\max_{t \in T} P(t|D) = \arg\max_{t \in T} \sum_{h \in H} P(t|h)P(h|D)$$

Limitations

- *Computationally costly* if H is large

4.10.2 Gibbs Classifier

Sample a hypothesis h from the posterior belief $P(h|D)$, then use h to classify $\mathbf{x}$

- Expected misclassification error is $\leq 2\times$ that of Bayes-optimal classifier!

4.10.3 Naive Bayes Classifier

Let target concept $c : X \to T$, where each instance $\mathbf{x} \in X$ is represented by input attributes $\mathbf{x} = (x_1, \dots, x_n)^T$.

Naive Bayes assumption: $P(x_1, \dots, x_n|t) = \prod_{i=1}^{n} P(x_i|t)$

- $(\star)$ Assumption: input attributes are *conditionally independent* given classification

Most probable classification of new instance $\mathbf{x}$:

$$
\begin{aligned}
t_{MAP} &= \arg\max_{t \in T} P(t|x_1, \ldots, x_n) \\
&= \arg\max_{t \in T} \frac{P(x_1, \ldots, x_n|t)P(t)}{P(x_1, \ldots, x_n)} \\
&= \arg\max_{t \in T} P(x_1, \ldots, x_n|t)P(t) \\
t_{NB} &= \arg\max_{t \in T} P(t) \prod_{i=1}^{n} P(x_i|t)
\end{aligned}
$$

Space analysis: $2n$ in number of input attributes, instead of $2 \times (2^n - 1)$!

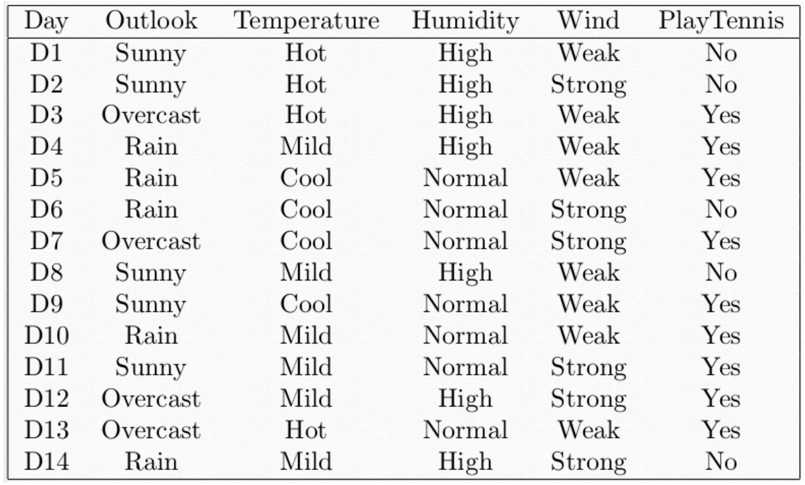
Data analysis: frequency counting $2n \times 2$, instead of 2×2^n!

Limitation: needs moderate to large amounts of training data

4.10.4 Naive Bayes Algorithm

```
NAIVE-BAYES-LEARN($D$)
  • For each value of target output $t \in T$:
      – $\widehat{P}(t) \leftarrow$ estimate $P(t)$ using $D$
      – For each value of attribute $x_i$:
          ∗ $\widehat{P}(x_i|t) \leftarrow$ estimate $P(x_i|t)$ using $D$
CLASSIFY-NEW-INSTANCE($\mathbf{x}$)
  • $t_{NB} = \arg\max_{t \in T} \widehat{P}(t) \prod_{i=1}^{n} \widehat{P}(x_i|t)$
```

4.10.5 Example of Naive Bayes



Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Predict the target concept *PlayTennis* for new instance
$\langle Outlook = Sunny, Temperature = Cool, Humidity = High, Wind = Strong \rangle$

- $P(Yes)P(Sunny|Yes)P(Cool|Yes)P(High|Yes)P(Strong|Yes) = \frac{9}{14} \cdot \frac{2}{9} \cdot \ldots = 0.005291$
- $P(No)P(Sunny|No)P(Cool|No)P(High|No)P(Strong|No) = \frac{5}{14} \cdot \frac{3}{5} \cdot \ldots = 0.02057$
- $P(No|Sunny, Cool, High, Strong) = \frac{0.02057}{0.02057+0.005291} = 0.7954$

4.10.6 Properties of Naive Bayes

Problem: What if for some attribute value x_i, none of the training instances have target output t?

$$\widehat{P}(x_i|t) = 0 \;\Rightarrow\; \widehat{P}(t)\prod_{i=1}^{n}\widehat{P}(x_i|t) = 0$$

- Biased underestimate of true likelihood
- As long as one probability is 0, the entire probability is 0
- Possible if we have little data!

Solution: Use Bayesian estimate:

$$\widehat{P}(x_i|t) = \frac{|D_{tx_i}| + mp}{|D_t| + m}$$

- $|D_t|$: #training examples with target output value t across all attribute values
- $|D_{tx_i}|$: #training examples with target output value t and attribute value x_i
- p: prior estimate for $\widehat{P}(x_i|t)$ (if don't know, use uniform prior)
- m: weight given to prior p (number of "virtual" examples)

4.11 Expectation Maximisation (EM)

Helps us to find maximum likelihood parameters of a model, involving variables that *can't be observed directly*

When to use EM?

- Data is only partially observable (hidden/latent variables)
- Unsupervised learning (target output unobservable)
- Supervised learning (some input attributes unobservable)

Applications

- Training Bayesian belief networks
- Unsupervised clustering
- Learning hidden Markov models
- Inverse reinforcement learning

4.11.1 EM for Estimating M means

Generating data from mixture of M Gaussians

- Instances $\mathbf{x}_d$ from X, each generated by a Gaussian distribution selected with equal probability from a mixture of M Gaussians, all with the same known variance σ^2
- Unknown means $\langle \mu_1, \dots, \mu_M \rangle$
- Don't know which instance is generated by which Gaussian

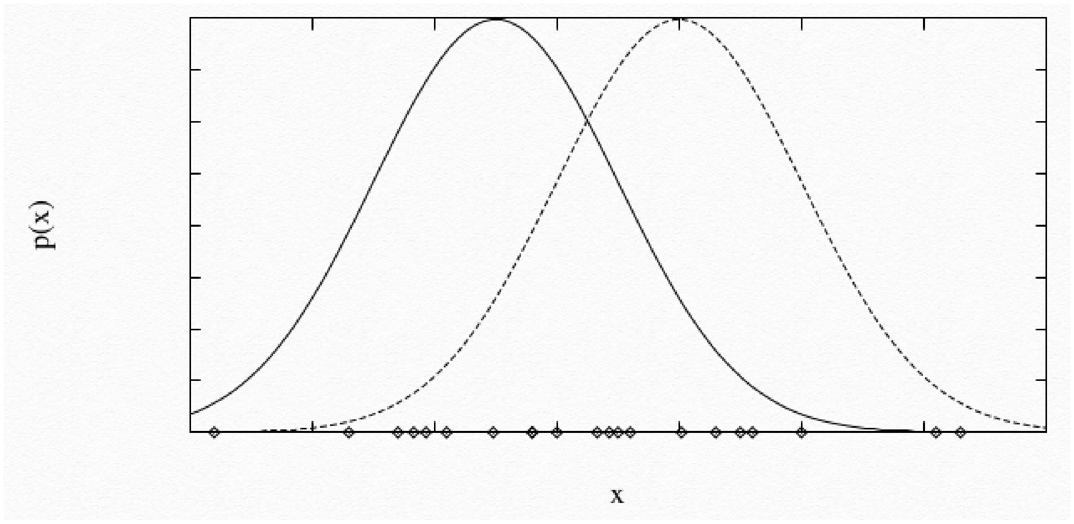
Determine *maximum likelihood* (ML) estimates of $\langle \mu_1, \dots, \mu_M \rangle$

Consider full description of each instance as $d = \langle x_d, z_{d1}, \dots, z_{dm} \rangle$

- x_d is observable

- z_{dm} is unobservable indicator variable: 1 if it's generated from m^{th} Gaussian, 0 otherwise
- ($\star$) Realise that $\mathbb{E}[z_{dm}]$ is the probability that m^{th} Gaussian is selected given that x_d is generated

Example: 2 Gaussians



4.11.2 EM Algorithm for 2 Gaussians

Pick random initial $h = \langle \mu_1, \mu_2 \rangle$

- E step: Calculate the expected value $\mathbb{E}[z_{dm}]$ of each hidden variable z_{dm}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

$$\mathbb{E}[z_{dm}] = \frac{p(x_d|\mu_m)}{\sum_l p(x_d|\mu_l)} = \frac{\exp(-\frac{1}{2\sigma^2}(x_d - \mu_m)^2)}{\sum_l \exp -\frac{1}{2\sigma^2}(x_d - \mu_l)^2}$$

- M step: Calculate a new ML hypothesis

$$h' = \langle \mu_1', \mu_2' \rangle$$

, assuming the value taken on by each z_{dm} is its expected value $\mathbb{E}[z_{dm}]$ computed above. Replace h by h'.

$$\mu_m' \leftarrow \frac{\sum_{d \in D} \mathbb{E}[z_{dm}] x_d}{\sum_{d \in D} \mathbb{E}[z_{dm}]}$$

(Intuitively, input instance is weighted by the probability that it's generated by the $m^t h$ Gaussian)

4.11.3 EM Algorithm

Converges to *local* ML hypothesis h' and provides estimates of hidden/latent variables z_{dm}

Local maximum in $\mathbb{E}[\ln p(D|h')]$: expected log likelihood

- D is complete data (observable x_d plus unobservable z_{dm} variables)
- Expectation is w.r.t unobserved variables in D

4.11.4 General EM Problem

Given

- Observed data $\{\mathbf{x}_d\}$
- Unobserved data $\{\mathbf{z}_d\}$ where $\mathbf{z}_d = \langle z_{d1}, \dots, z_{dM} \rangle$
- Parameterized probability distribution $p(D|h)$ where
  - $D = \{d\}$ is the complete data where $d = \langle \mathbf{x}_d, \mathbf{z}_d \rangle$
  - h comprises the parameters

Determine ML hypothesis h' that (locally) maximizes $\mathbb{E}[\ln p(D|h')]$

4.11.5 General EM Algorithm

Define $Q(h|h') = \mathbb{E}[\ln p(D|h')|h, \{\mathbf{x}_d\}_{d \in D}]$ given current hypothesis h and observed data $\{\mathbf{x}_d\}_{d \in D}$ to estimate the latent variables $\{\mathbf{z}\}_{d \in D}$

EM Algorithm: Pick a random initial h. Then iterate:

- E step: Calculate $Q(h|h') = \mathbb{E}[\ln p(D|h')|h, \{\mathbf{x}_d\}_{d \in D}$ using h and observed data $\{\mathbf{x}_d\}_{d \in D}$ to estimate latent variables $\{\mathbf{z}_d\}_{d \in D}$
- M step: Replace h by h' that maximises this Q function: $h \leftarrow \arg\max_{h'} Q(h'|h)$

4.11.6 Applying EM to Estimate M Means

E step

$$\begin{aligned}
p(d|h') &= p(x_d, z_{d1}, \dots, z_{dM}|h') \text{ (where } h' = \langle \mu'_1, \dots, \mu'_M \rangle) \\
&= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2} \sum_{m=1}^{M} z_{dm}(x_d - \mu'_m)^2 \right) \text{ (take only the selected Gaussian)} \\
\ln p(D|h') &= \ln \prod_{d \in D} p(d|h') = \sum_{d \in D} \ln p(d|h') \\
&= \sum_{d \in D} \left(\ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{m=1}^{M} z_{dm}(x_d - \mu'_m)^2 \right) \\
Q(h'|h) &= \mathbb{E}[\ln p(D|h')] \\
&= \mathbb{E}\left[\sum_{d \in D} \left(\ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{m=1}^{M} z_{dm}(x_d - \mu'_m)^2 \right) \right] \\
&= \sum_{d \in D} \left(\ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{m=1}^{M} \mathbb{E}[z_{dm}](x_d - \mu'_m)^2 \right) \\
\text{where } \mathbb{E}[z_{dm}] &= \frac{\exp(-\frac{1}{2\sigma^2}(x_d - \mu_m)^2)}{\sum_{l=1}^{M} \exp(-\frac{1}{2\sigma^2}(x_d - \mu_l)^2)}
\end{aligned}$$

M step

$$\begin{aligned}
&\arg\max_{h'} Q(h|h') \\
&= \arg\max_{h'} \sum_{d \in D} \left(\ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{m=1}^{M} \mathbb{E}[z_{dm}](x_d - \mu'_m)^2 \right) \\
&= \arg\max_{h'} \sum_{d \in D} - \sum_{m=1}^{M} \mathbb{E}[z_{dm}](x_d - \mu'_m)^2 \\
&= \arg\min_{h'} \sum_{d \in D} \sum_{m=1}^{M} \mathbb{E}[z_{dm}](x_d - \mu'_m)^2
\end{aligned}$$

$$\mu'_m \leftarrow \frac{\sum_{d \in D} \mathbb{E}[z_{dm}] x_d}{\sum_{d \in D} \mathbb{E}[z_{dm}]}$$