



20T2

Comp9417

Review 2



Comp9417

Decision Tree

Top-Down Induction of Decision Trees

Main loop:

- $A \leftarrow$ the “best” decision attribute for next node to split examples
- Assign A as decision attribute for node
- For each value of A , create new descendant of node (child node)
- Split training examples to child nodes
- If training examples perfectly classified (pure subset), Then STOP, Else iterate over new child nodes

Discovered by two people independently:

- Ross Quinlan (ID3: 1986), (C4.5: 1993)
- Breiman et al (CaRT: 1984) from statistics



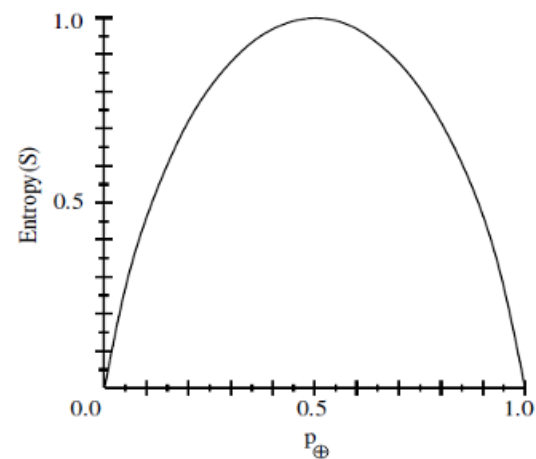
Comp9417

Entropy and Information Gain

$$\text{Entropy}(S) = H(S) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

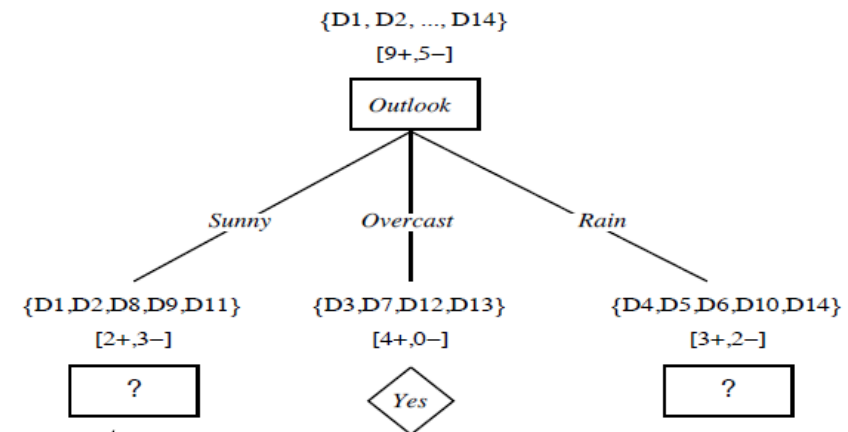
- v is the possible values of attribute A
- S is the set of examples we want to split
- S_v is the subset of examples where $X_A = v$



Comp9417

Example

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



Which attribute should be tested here?

$$S_{\text{sunny}} = \{D1, D2, D8, D9, D11\}$$

$$\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = .970 - (3/5) 0.0 - (2/5) 0.0 = .970$$

$$\text{Gain}(S_{\text{sunny}}, \text{Temperature}) = .970 - (2/5) 0.0 - (2/5) 1.0 - (1/5) 0.0 = .570$$

$$\text{Gain}(S_{\text{sunny}}, \text{Wind}) = .970 - (2/5) 1.0 - (3/5) .918 = .019$$



Comp9417

Gain Ratio

$$\text{SplitEntropy}(S, A) = - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

Where:

- A : candidate attribute
- v : possible values of A
- S : Set of examples $\{X\}$ at the node
- S_v : subset where $X_A = v$

$$\text{GainRatio}(S, A) = \frac{\text{Gain}(S, A)}{\text{SplitEntropy}(S, A)}$$



Comp9417

Overfitting

Consider error of hypothesis h over

- training data: $error_{train}(h)$
- entire distribution \mathcal{D} of data: $error_{\mathcal{D}}(h)$

Definition

Hypothesis $h \in H$ overfits training data if there is an alternative hypothesis $h' \in H$ such that

$$error_{train}(h) < error_{train}(h')$$

And

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$



Comp9417

Pre-Pruning

- Maximum number of leaf nodes (in *sklearn*, *max_leaf_nodes*)
- Minimum number of samples to split (in *sklearn* *min_samples_split*)
- Maximum depth (in *sklearn*, *max_depth*)

In *sklearn*, this parameter is *min_samples_leaf*



Comp9417

Post-pruning

- Builds full tree first and prunes it afterwards
 - Attribute interactions are visible in fully-grown tree
- Problem: identification of subtrees and nodes that are due to chance effects
- Subtree replacement
- Possible strategies: error estimation, significance testing, MDL principle. We examine
 - reduced-error Pruning
 - Minimum error
 - Smallest tree



Comp9417

Reduced-error Pruning

Split data into *training* and *validation* set

Do until further pruning is harmful:

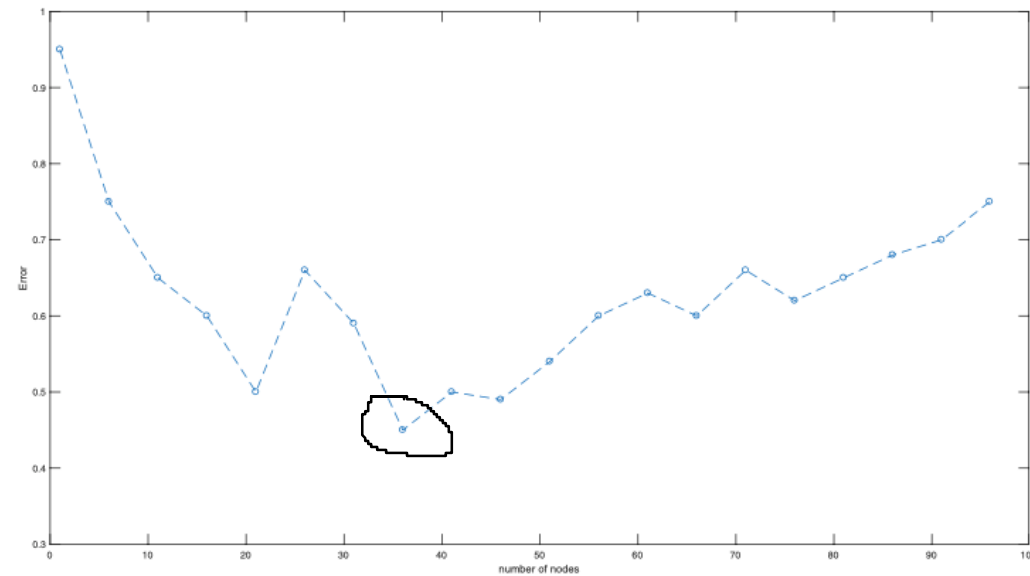
- Evaluate impact on *validation* set of pruning each possible node (plus those below it)
 - Greedily remove the one that most improves *validation* set accuracy
-
- **Good** produces smallest version of most accurate subtree
 - **Not so good** reduces effective size of training set



Comp9417

Minimum Error

- Keep the Cross-Validation error during the growing
- Prune back to the point where the cross-validated error is a minimum.
 - In the following example, the first 36 nodes will be kept

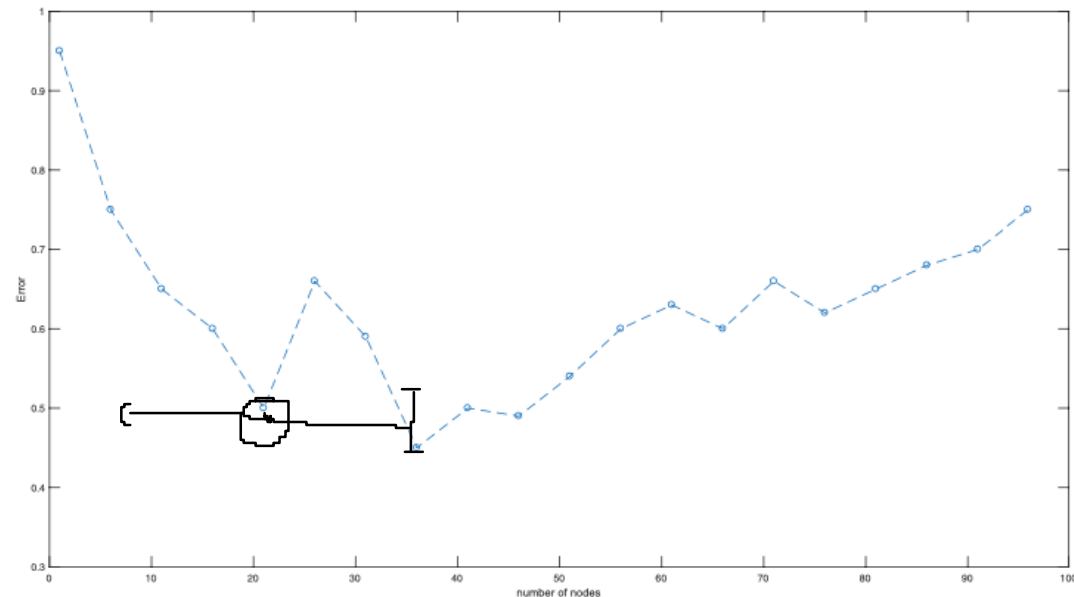




Comp9417

Smallest Tree

- Keep the Cross-Validation error during the growing
- Prune back the tree to the smallest tree within 1 standard error of the minimum error.
 - In the following example, the first 20 nodes will be kept



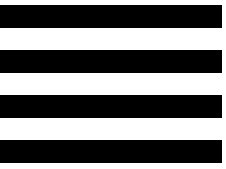
Comp9417

Continuous Valued Attr

- Example for $n - 1$ splits: more efficient number of splits if we know the class labels



- You can use the midway point as your boundary (e.g if $x > 12.4$ between points 3 and 4)
- Or, you can use the value in the training (e.g if $x > 11.8$ between points 3 and 4)



Comp9417

Inductive Bias

Inductive bias: set of assumptions needed in addition to training data to justify deductively learner's classifications

Restriction bias:

- The set of hypothesis that can be modelled by decision trees

Preference biases:

- Prefers trees with good splits near the top (splitting on features with the most information gain)
- Prefers shorter trees (comes naturally from good splits at the top and minimum description length)




Comp9417

Decision Tree

Pros:

- Interpretability ✓
- Easily handle irrelevant attributes (Gain = 0)
- Can handle both categorical and numerical data
- Can handle missing data
- Very compact (number of nodes \ll number of examples)
- Very fast at testing ✓

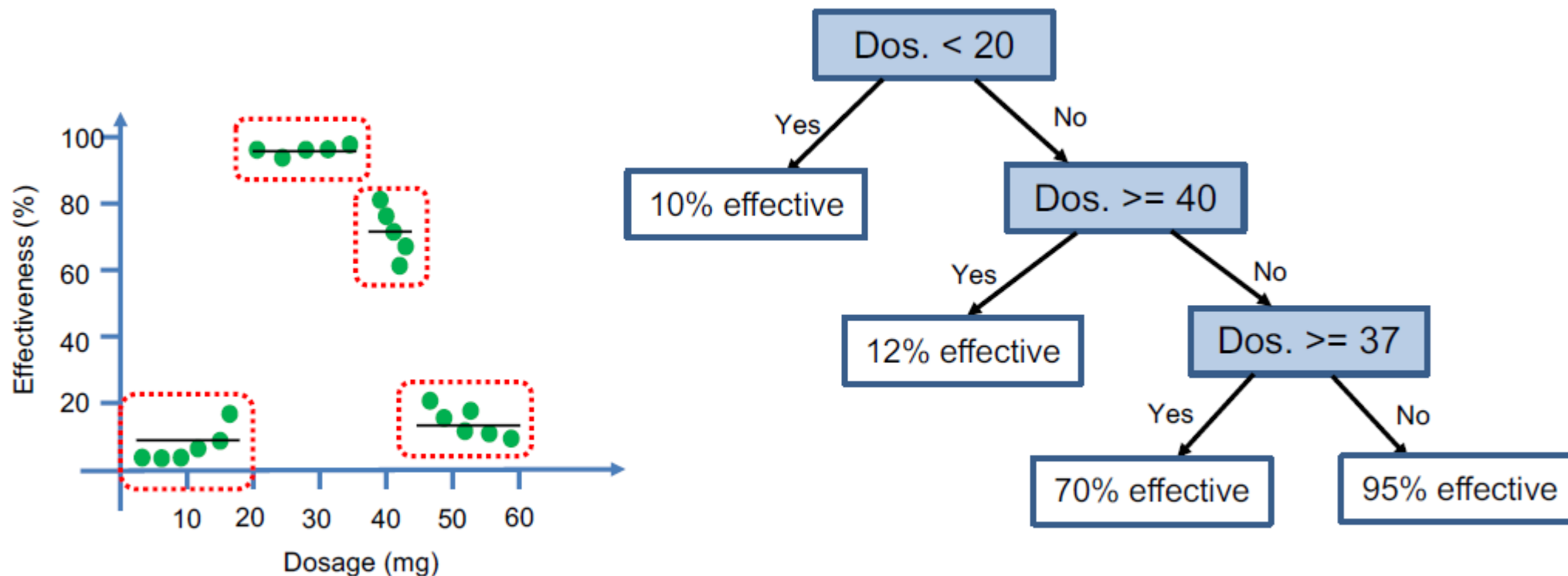
Cons:

- Only axis-aligned splits of the data
- Tend to overfit 
- Greedy (may not find the best tree)
 - Exponentially many possible trees

Comp9417

Regression Tree

One option would be to use regression trees



Each leaf corresponds to average drug effectiveness in a different cluster of examples, the tree does a better job than *Linear Regression*



Comp9417

Regression Tree

- Mean squared error after setting the threshold, for each subset with m examples is:

$$MSE(Y_i) = \frac{1}{m} \sum_{j=1}^m (y_j - \bar{y})^2$$

- The MSE here is equal to the variance of the examples in that subset
- Compute the weighted average of variance

$$\text{weighted average variance} = \sum_i^l \frac{|Y_i|}{|Y|} MSE(Y_i)$$

- We pick a threshold which minimizes the weighted average of mean squared error / variance



Comp9417

Regression Tree

Imagine you are a collector of vintage Hammond tonewheel organs. You have been monitoring an online auction site, from which you collected some data about interesting transactions:

#	Model	Condition	Leslie	Price
1.	B3	excellent	no	4513
2.	T202	fair	yes	625
3.	A100	good	no	1051
4.	T202	good	no	270
5.	M102	good	yes	870
6.	A100	excellent	no	1770
7.	T202	fair	no	99
8.	A100	good	yes	1900
9.	E112	fair	no	77



Comp9417

Regression Tree

From this data, you want to construct a regression tree that will help you determine a reasonable price for your next purchase.

There are three features, hence three possible splits:

Model = [A100, B3, E112, M102, T202]
[1051, 1770, 1900][4513][77][870][99, 270, 625]

Condition = [excellent, good, fair]
[1770, 4513][270, 870, 1051, 1900][77, 99, 625]

Leslie = [yes, no]
[625, 870, 1900][77, 99, 270, 1051, 1770, 4513]



Comp9417

Regression Tree

- The means of the first split are 1574, 4513, 77, 870 and 331, and the weighted average of mean squared errors is 3.21×10^5 .
- The means of the second split are 3142, 1023 and 267, with weighted average of mean squared errors 2.68×10^6 ;
- for the third split the means are 1132 and 1297, with weighted average of mean squared errors 1.55×10^6 .

We therefore branch on *Model* at the top level. This gives us three single-instance leaves, as well as three *A100*s and three *T202*s.



Comp9417

Regression Tree

For the A100s we obtain the following splits:

Condition = [*excellent, good, fair*]
[1770][1051,1900][]

Leslie = [*yes, no*] [1900]
[1051,1770]

Without going through the calculations we can see that the second split results in less variance (to handle the empty child, it is customary to set its variance equal to that of the parent). For the T202s the splits are as follows:

Condition = [*excellent, good, fair*]
[][270][99,625]

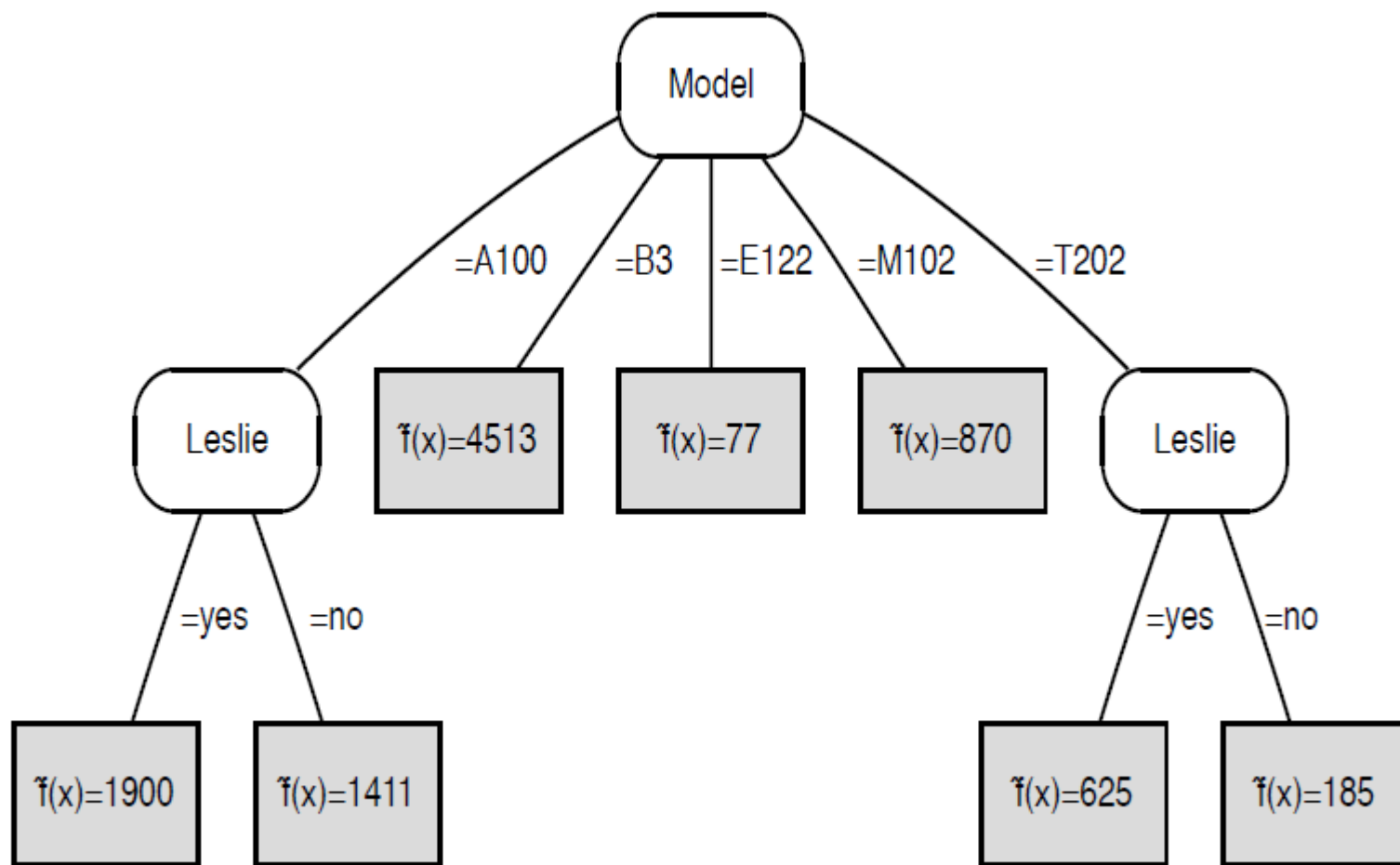
Leslie = [*yes, no*]
[625][99,270]

Again we see that splitting on Leslie gives tighter clusters of values. The learned regression tree is depicted on the next slide.



Comp9417

Regression Tree





Comp9417

sample

A	B	Class
1	0	+
0	1	-
1	1	-
1	0	+
1	1	-
1	1	-
0	0	+
1	1	+
0	0	+
0	0	-

- (a) What is the entropy of these examples with respect to the given classification?
- (b) What is the Information gain of attribute A on sample S above?
- (c) What is the information gain of attribute B on sample S above?
- (d) What would be chosen as the 'best' attribute by a decision tree learner using the information gain splitting criterion? Why?

Comp9417

sample

Question 2 Here is small dataset for a two-class prediction task. There are 4 attributes, and the class is in the rightmost column. Look at the examples. Can you guess which attribute(s) will be most predictive of the class ?

species	rebel	age	ability	homeworld
pearl	yes	6000	regeneration	no
bismuth	yes	8000	regeneration	no
pearl	no	6000	weapon-summoning	no
garnet	yes	5000	regeneration	no
amethyst	no	6000	shapeshifting	no
amethyst	yes	5000	shapeshifting	no
garnet	yes	6000	weapon-summoning	no
diamond	no	6000	regeneration	yes
diamond	no	8000	regeneration	yes
amethyst	no	5000	shapeshifting	yes
pearl	no	8000	shapeshifting	yes
jasper	no	6000	weapon-summoning	yes

You probably guessed that attributes 3 and 4 were not very predictive of the class, which is true. However, you might be surprised to learn that attribute “species” has higher information gain than attribute ”rebel”. Why is this ? Refer to slides 37-38 on “Attributes with Many Values” in the lecture notes.

Suppose you are told the following: for attribute “species” the Information Gain is 0.52 and *Split Information* is 2.46, whereas for attribute “rebel” the Information Gain is 0.48 and *Split Information* is 0.98.

Which attribute would the decision-tree learning algorithm select as the split when using the *Gain Ratio* criterion instead of Information Gain ? Is Gain Ratio a better criterion than Information Gain in this case ?



Comp9417

sample

Question 3 Assume we learn a decision tree to predict class Y given attributes A , B and C from the following training set, with no pruning.

A	B	C	Y
0	0	0	0
0	0	1	0
0	0	1	0
0	1	0	0
0	1	1	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	0	1
1	1	1	0
1	1	1	1

What would be the training set error for this dataset ? Express your answer as the number of examples out of twelve that would be misclassified.

Answer

2. There are two pairs of examples with the same values for attributes A , B and C but a different (contradictory) value for class Y . One example from each of these pairs will always be misclassified (noise).

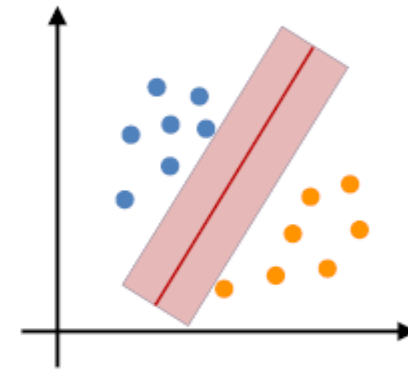
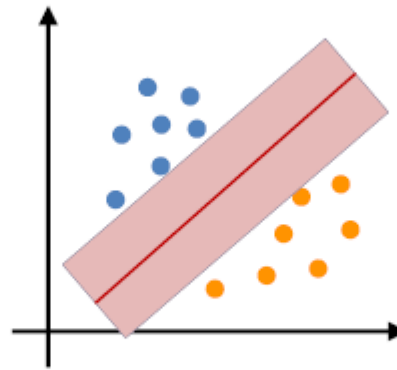
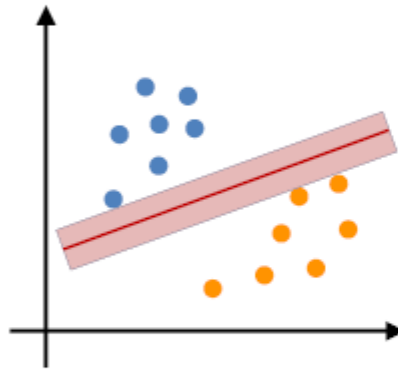


Comp9417

SVM

Which line is a better classifier?

- The line with bigger margin

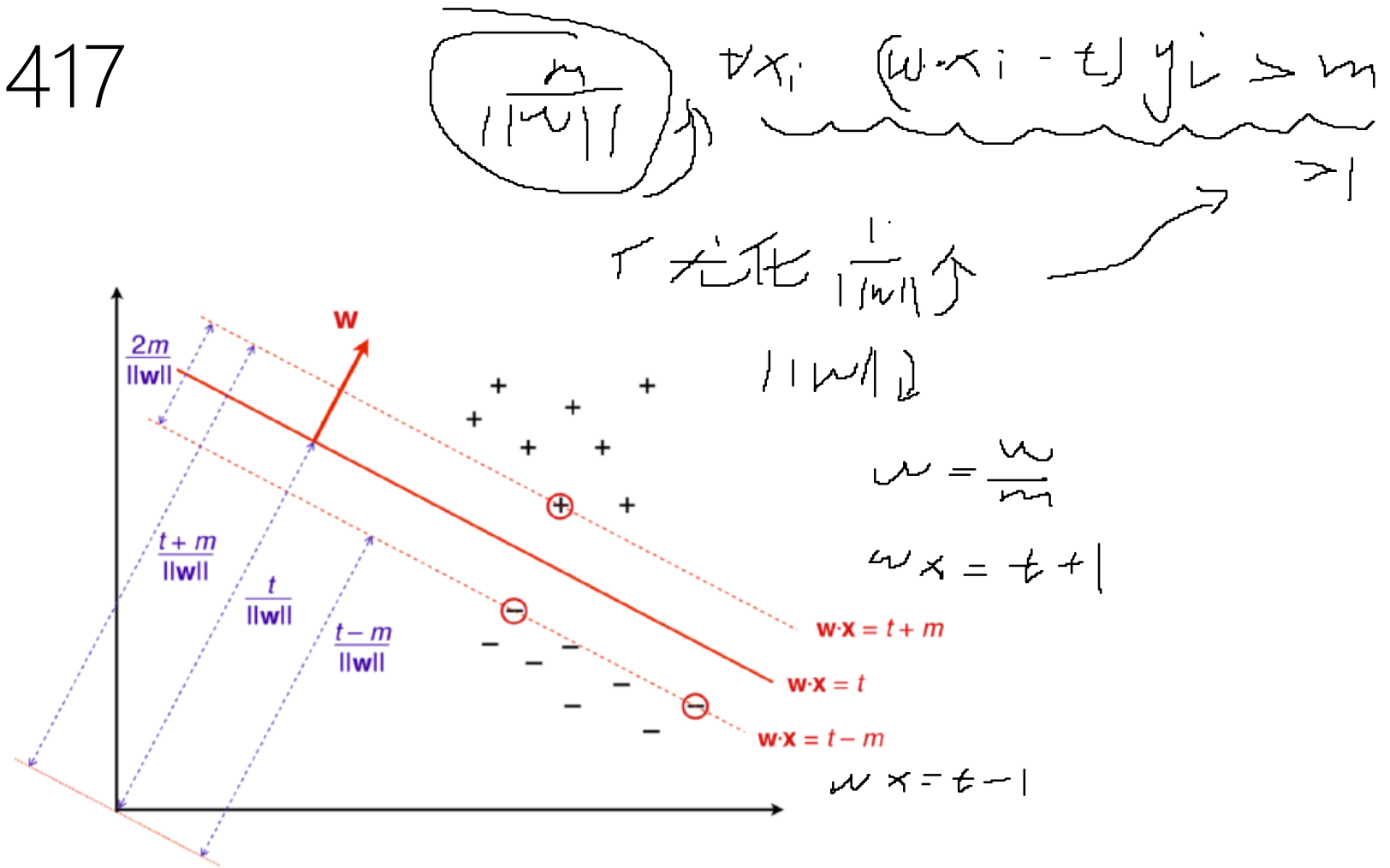


- Why bigger margin is better?
- Can I find a w that maximizes the margin?



Comp9417

SVM



The geometry of a support vector classifier. The circled data points are the support vectors, which are the training examples nearest to the decision boundary. The support vector machine finds the decision boundary that maximizes the margin $m/||w||$.



Comp9417

SVM

Our focus is on the points which the line (hyperplane) can predict them correctly. So we can have:

$$|w \cdot x_i - t| = y_i(w \cdot x_i - t)$$

And we can change the “min” in constraint as follow:

$$y_i(w \cdot x_i - t) \geq 1 \quad \text{for } i = 1, \dots, m$$

We can also transform the maximization problem into the following minimization problem:

$$\min \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i - t) \geq 1 \quad \text{for } i = 1, \dots, m$$

$w \in \mathbb{R}^n, t \in \mathbb{R}$

How to solve this? Largangian multipliers



Comp9417

SVM

- Constrained optimization problems are generally expressed as:

$$\min_{x_1, \dots, x_n} f(x_1, \dots, x_n)$$

Subject to:

$$g_1(x_1, \dots, x_n) \leq 0, g_2(x_1, \dots, x_n) \leq 0, \dots, g_k(x_1, \dots, x_n) \leq 0$$

- Lagrange multiplier methods involve the modification of the objective function through the addition of terms that describe the constraints. The objective function $f(x)$ is augmented by the constraint equations through a set of non-negative multiplicative Lagrange multipliers, $\alpha_j \geq 0$ and it is called the dual Lagrangian:

$$\mathcal{L}(x_1, \dots, x_n, \alpha_1, \dots, \alpha_m) = f(x_1, \dots, x_n) + \sum_{j=1}^k \alpha_j g_j(x_1, \dots, x_n)$$



Comp9417

SVM

Adding the constraints with multipliers α_i for each training example gives the Lagrange function:

$$\begin{aligned}\mathcal{L}(w, t, \alpha_1, \dots, \alpha_m) &= \frac{1}{2} ||w||^2 - \sum_{i=1}^m \alpha_i (y_i (w \cdot x_i - t) - 1) \\ &= \frac{1}{2} ||w||^2 - \sum_{i=1}^m \alpha_i y_i (w \cdot x_i) + \sum_{i=1}^m \alpha_i y_i t + \sum_{i=1}^m \alpha_i \\ &= \frac{1}{2} w \cdot w - w \cdot \left(\sum_{i=1}^m \alpha_i y_i x_i \right) + t \left(\sum_{i=1}^m \alpha_i y_i \right) + \sum_{i=1}^m \alpha_i\end{aligned}$$

Comp9417

SVM

- First we have to minimize \mathcal{L} with respect to w and t
- By taking the partial derivative of the Lagrange function with respect to t and setting it to 0 we find:

$$\sum_{i=1}^m \alpha_i y_i = 0$$

- Similarly, by taking the partial derivative of the Lagrange function with respect to w and setting to 0 we obtain:

$$w = \sum_{i=1}^m \alpha_i y_i x_i$$

– the same expression as we derived for the perceptron.

- The dual optimization problem for support vector machines is to maximize the dual Lagrangian under positivity constraints and one equality constraint:

$$\alpha_1^*, \dots, \alpha_m^* = \operatorname{argmax}_{\alpha_1, \dots, \alpha_m} -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i \cdot x_j + \sum_{i=1}^m \alpha_i$$

subject to $\alpha_i > 0, 1 \leq i \leq m, \sum_{i=1}^m \alpha_i y_i = 0$



Comp9417

SVM

$$X = \begin{pmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \end{pmatrix} \quad y = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} \quad X' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \end{pmatrix}$$

The matrix X' on the right incorporates the class labels; i.e., the rows are $y_i x_i$. The Gram matrix is (without and with class labels):

$$XX^T = \begin{pmatrix} 5 & 3 & -5 \\ 3 & 5 & -3 \\ -5 & -3 & 5 \end{pmatrix} \quad X'X'^T = \begin{pmatrix} 5 & 3 & 5 \\ 3 & 5 & 3 \\ 5 & 3 & 5 \end{pmatrix}$$

The dual optimization problem is thus

$$\operatorname{argmax}_{\alpha_1, \alpha_2, \alpha_3} -\frac{1}{2}(5\alpha_1^2 + 3\alpha_1\alpha_2 + 5\alpha_1\alpha_3 + 3\alpha_2\alpha_1 + 5\alpha_2^2 + 3\alpha_2\alpha_3 + 5\alpha_3\alpha_1 + 3\alpha_3\alpha_2 + 5\alpha_3^2) + \alpha_1 + \alpha_2 + \alpha_3$$

$$= \operatorname{argmax}_{\alpha_1, \alpha_2, \alpha_3} -\frac{1}{2}(5\alpha_1^2 + 6\alpha_1\alpha_2 + 10\alpha_1\alpha_3 + 5\alpha_2^2 + 36\alpha_3 + 5\alpha_3^2) + \alpha_1 + \alpha_2 + \alpha_3$$

subject to $\alpha_1 \geq 0, \alpha_2 \geq 0, \alpha_3 \geq 0$ and $-\alpha_1 - \alpha_2 + \alpha_3 = 0$.



Comp9417

SVM

- Using the equality constraint we can eliminate one of the variables, say α_3 , and simplify the objective function to

$$\operatorname{argmax}_{\alpha_1, \alpha_2} -\frac{1}{2}(20\alpha_1^2 + 32\alpha_1\alpha_2 + 16\alpha_2^2) + 2\alpha_1 + 2\alpha_2$$

- Setting partial derivatives to 0 we obtain $-20\alpha_1 - 16\alpha_2 + 2 = 0$ and $-16\alpha_1 - 16\alpha_2 + 2 = 0$ (notice that, because the objective function is quadratic, these equations are guaranteed to be linear).
- We therefore obtain the solution $\alpha_1 = 0$ and $\alpha_2 = \alpha_3 = 1/8$. We then have $w = 1/8(x_3 - x_2) = \begin{pmatrix} 0 \\ -1/2 \end{pmatrix}$, resulting in a margin of $1/\|w\| = 2$.
- Finally, t can be obtained from any support vector, say x_2 , since $y_2(w \cdot x_2 - t) = 1$; this gives $-1 \cdot (-1 - t) = 1$, hence $t = 0$.



Comp9417

Kernel Trick

Let $x = (x_1, x_2)$ and $x' = (x_1', x_2')$ be two data points, and consider the following mapping to a three-dimensional feature space:

$$(x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

(original feature space) $\mathcal{X} \rightarrow \mathcal{Z}$ (new feature space)

The points in feature space corresponding to x and x' are

$$z = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \text{ and } z' = (x_1'^2, x_2'^2, \sqrt{2}x_1'x_2')$$

The dot product of these two feature vectors is

$$z \cdot z' = x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1x_1'x_2x_2' = (x_1x_1' + x_2x_2')^2 = (x_1 \cdot x_2)^2$$



Comp9417

Kernel SVM

If $z = \varphi(x)$, Lagrangian becomes:

$$\mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \varphi(x_i) \cdot \varphi(x_j) + \sum_{i=1}^m \alpha_i$$

If we can find a kernel function, such that:

$$K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$$

(Kernel corresponds to a map into a new feature space)

Then:

$$\mathcal{L}(\alpha) = -\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j K(x_i, x_j) + \sum_{i=1}^m \alpha_i$$



Comp9417

Kernel Trick

- Polynomial kernel is defined as:

$$K(x, x') = (x \cdot x' + c)^q$$

- RBF kernel is defined as:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$



Comp9417

Sample

Here is a toy data set of three examples shown as the matrix \mathbf{X} , of which the first two are classified as positive and the third as negative, shown as the vector \mathbf{y} . Start by constructing the *Gram matrix* for this data, incorporating the class labels, i.e., form the matrix $\mathbf{X}'\mathbf{X}'^T$. Then solve to find the support vectors, their Lagrange multipliers α , then determine the weight vector \mathbf{w} , threshold t and the margin m .

$$\mathbf{X} = \begin{pmatrix} 1 & 3 \\ 2 & 1 \\ 0 & 1 \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} +1 \\ +1 \\ -1 \end{pmatrix} \quad \mathbf{X}' = \begin{pmatrix} 1 & 3 \\ 2 & 1 \\ 0 & -1 \end{pmatrix}$$

Background To find a maximum margin classifier requires finding a solution for \mathbf{w} , t and margin m . For this we can use the following steps (refer to slides 50-56 from the “Kernel Methods” lecture):

1. set up Gram matrix for labelled data
2. set up expression to be minimised
3. take partial derivatives
4. set to zero and solve for each multiplier
5. solve for \mathbf{w}
6. solve for t
7. solve for m

Comp9417

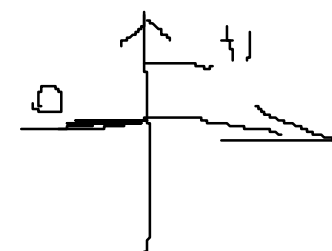
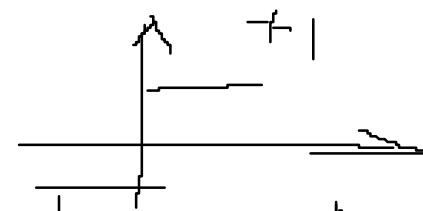
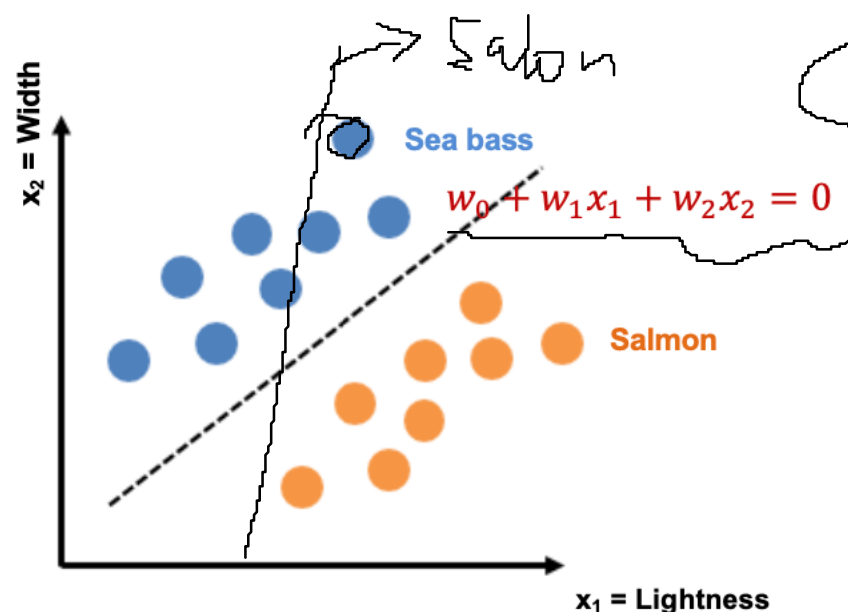
Perceptron Learning

Perceptron: is an algorithm for binary classification that uses a linear prediction function

If we have two attributes/features of x_1 and x_2 then we can predict the target function $f(x)$ with:

$$f(x) = \begin{cases} +1 & \text{if } w \cdot x \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$\hat{y} = f(x) = \text{sgn}(w \cdot x)$$



$$y_i = 1$$
$$\sum w_i x_i < 0$$

$$\sum w_i' x_i > 0$$
$$\sum w_i' x_i > \sum w_i x_i$$
$$\sum (w_i' - w_i) x_i$$

$$w_i' - w_i$$

$$w_i' - w_i$$

$$w = w + \eta y_i x_i$$

Comp9417

Perceptron Learning

Algorithm Perceptron(D, η) // perceptron training for linear classification

Input: labelled training data D in homogeneous coordinates; learning rate η .

Output: weight vector \mathbf{w} defining classifier $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$.

```
1  $\mathbf{w} \leftarrow \mathbf{0}$  // Other initialisations of the weight vector are possible
2  $\text{converged} \leftarrow \text{false}$ 
3 while  $\text{converged} = \text{false}$  do
4    $\text{converged} \leftarrow \text{true}$ 
5   for  $i = 1$  to  $|D|$  do
6     if  $\hat{y}_i \mathbf{w} \cdot \mathbf{x}_i \leq 0$  then // i.e.,  $\hat{y}_i \neq y_i$ 
7        $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$ 
8        $\text{converged} \leftarrow \text{false}$  // We changed  $\mathbf{w}$  so haven't converged yet
9     end
10  end
11 end
```



Comp9417

Recall

Gradient

$$\nabla E[\mathbf{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Gradient vector gives direction of *steepest increase* in error E

Negative of the gradient, i.e., *steepest decrease*, is what we want

Training rule:

$$\Delta \mathbf{w} = -\eta \nabla E[\mathbf{w}]$$

i.e.,

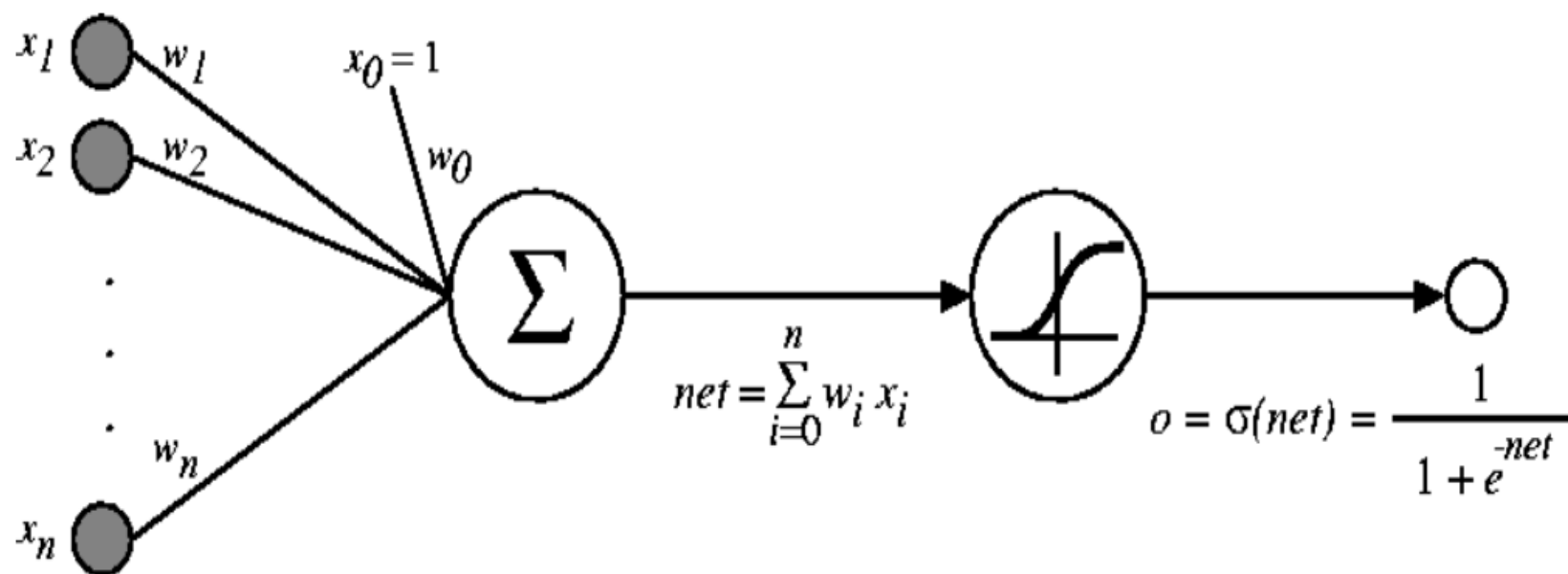
$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \mathbf{w} \cdot \mathbf{x}_d) \\ \frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d) (-x_{i,d}) \end{aligned}$$



Comp9417

MLP





Comp9417

MLP

Why use the sigmoid function $\sigma(x)$?

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient descent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units \rightarrow Backpropagation

Note: in practice, particularly for deep networks, sigmoid functions are less common than other non-linear activation functions that are easier to train, but sigmoids are mathematically convenient.



Comp9417

MLP

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

We know:

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

So:

$$\frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$



Comp9417

MLP for Classification

Sigmoid unit computes output $o(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x})$

Output ranges from 0 to 1

Example: binary classification

$$o(\mathbf{x}) = \begin{cases} \text{predict class 1} & \text{if } o(\mathbf{x}) \geq 0.5 \\ \text{predict class 0} & \text{otherwise.} \end{cases}$$

Questions:

- what error (loss) function should be used ?
- how can we train such a classifier ?



Comp9417

MLP for Classification

Minimizing square error (as before) does not work so well for classification

If we take the output $o(x)$ as the *probability* of the class of x being 1, the preferred loss function is the *cross-entropy*

$$-\sum_{d \in D} t_d \log o_d + (1 - t_d) \log (1 - o_d)$$

where:

$t_d \in \{0, 1\}$ is the class label for training example d , and o_d is the output of the sigmoid unit, interpreted as the probability of the class of training example d being 1.

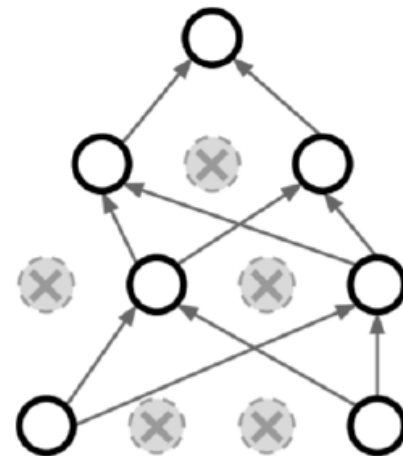
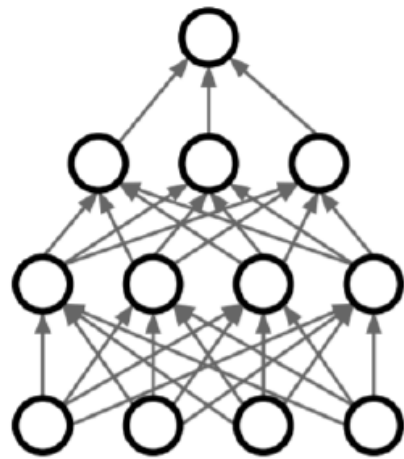
To train sigmoid units for classification using this setup, can use *gradient ascent* with a similar weight update rule as that used to train neural networks by gradient descent – this will yield the *maximum likelihood* solution.



Comp9417

Dropout

- Problem with overfitting – model performs well on training data but generalises poorly to testing data
- Dropout is a simple and effective method to reduce overfitting
- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyperparameter, such as 0.5

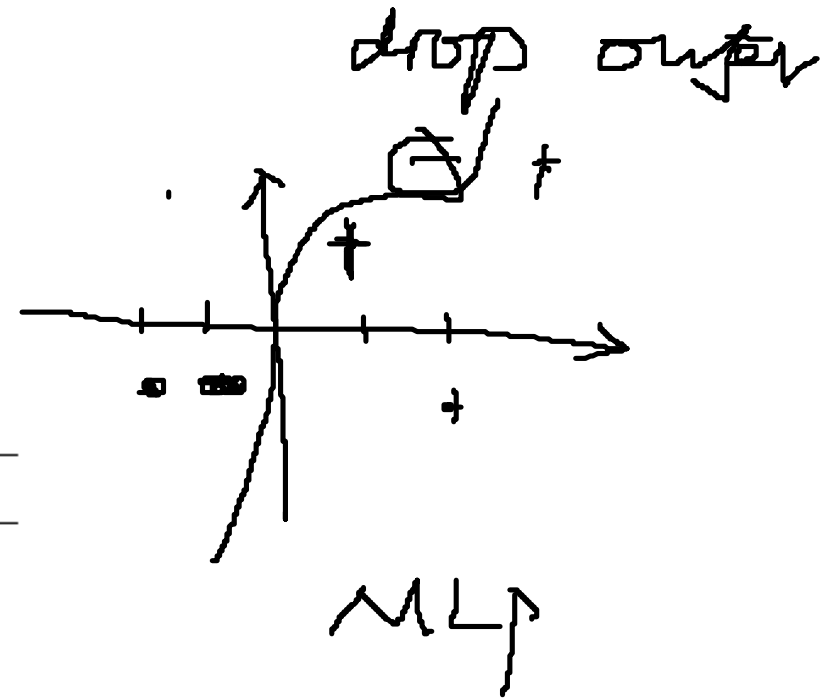




Comp9417

Sample

x_1	x_2	y
-2	-1	-1
2	-1	1
1	1	1
-1	-1	-1
3	2	1



(a) Apply the Perceptron Learning Algorithm with starting values $w_0 = 5$, $w_1 = 1$ and $w_2 = 1$, and a learning rate $\eta = 0.4$. Be sure to cycle through the training data in the same order that they are presented in the table.

$$w = \begin{bmatrix} 5 & 1 & 1 \end{bmatrix} \quad b = -2$$



Comp9417

Sample

$$5 - 2 - 1 = 2$$

Iteration	$w^T x$	$yw^T x$	w
1	2.00	-2.00	$[4.6, 1.8, 1.4]^T$
2	6.80	6.80	$[4.6, 1.8, 1.4]^T$
3	7.80	7.80	$[4.6, 1.8, 1.4]^T$
4	1.40	-1.40	$[4.2, 2.2, 1.8]^T$
5	14.40	14.40	$[4.2, 2.2, 1.8]^T$
6	-2.00	2.00	$[4.2, 2.2, 1.8]^T$
7	6.80	6.80	$[4.2, 2.2, 1.8]^T$
8	8.20	8.20	$[4.2, 2.2, 1.8]^T$
9	0.20	-0.20	$[3.8, 2.6, 2.2]^T$
10	16.00	16.00	$[3.8, 2.6, 2.2]^T$
11	-3.60	3.60	$[3.8, 2.6, 2.2]^T$
12	6.80	6.80	$[3.8, 2.6, 2.2]^T$
13	8.60	8.60	$[3.8, 2.6, 2.2]^T$

14

-1.0

1.0



Comp9417

Sample

(b) Consider a new point, $x_{\star} = (-5, 3)$. What is the predicted value and predicted class based on your learned perceptron for this point?

value: -2.6 , class: $y_{\star} = -1$

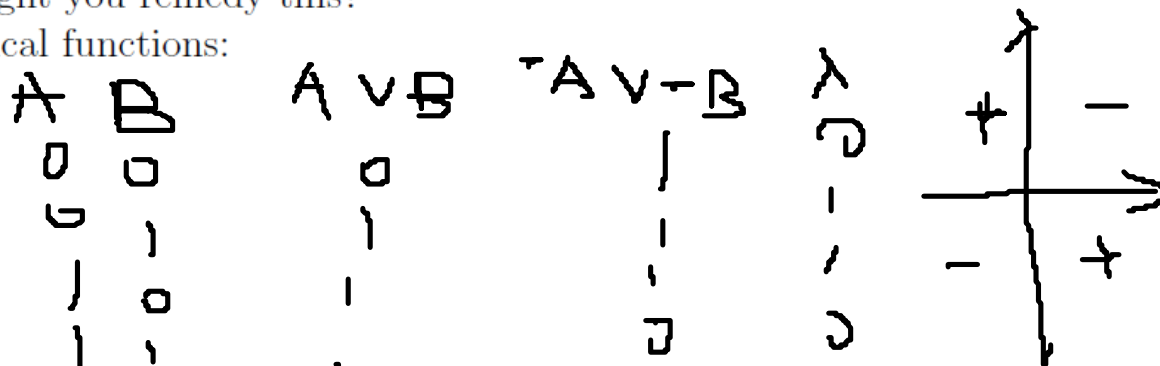
(c) Consider adding a new point to the data set, $x_{\star} = (2, 2)$ and $y_{\star} = -1$. Will your perceptron converge on the new dataset? How might you remedy this?

(d) Consider the following three logical functions:

1. $A \wedge \neg B$ ✓

2. $\neg A \vee B$ ✓

3. $(A \vee B) \wedge (\neg A \vee \neg B)$



Which of these functions can a perceptron learn? Explain. What are two ways that you can extend a perceptron to learn all three functions?

A Perceptron can only learn f_1 and f_2 . we can extend via multilayer perceptrons, or by using a smart transformation (i.e. kernel perceptron).