Midterm for 6.034 Spring 2010
Name:

20	20	20	20

Good luck!

Question #1 — 20 points

Indicate whether the following statement is true. Provide a short explanation. You will not be awarded any points if your answer is not justified

(i) (2 points) Alpha-beta pruning can alter the computed minimax value of the root of a game search tree.

(F) $\alpha\beta$ pruning only discards subtrees that do not change the computed min-max value at the root. That is the whole point.

(ii) (2 points) When doing alpha-beta pruning on a game tree which is traversed from left to right, the leftmost branch will never be pruned.

(T) The leftmost leaf is evaluated first. You cannot prune a branch before evaluating any leaf.

(iii) (2 points) Iterative Deepening (ID) is preferred to DFS in CSP as it is always guaranteed to stop.

(F) CSP $\Rightarrow$ finite number of vars $\Rightarrow$ finite depth $\Rightarrow$ DFS stops too

(iv) (2 points) If $g(s)$ and $h(s)$ are two admissible A^* heuristics, then the following heuristic $\frac{1}{2}g(s) + \frac{1}{4}h(s)$ is also admissible.

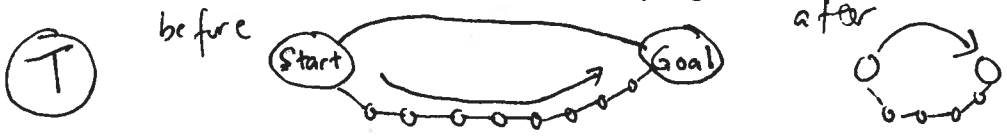
(T) $\frac{1}{2}g(s) + \frac{1}{4}h(s) \leq \underbrace{\frac{1}{2}g(s) + \frac{1}{2}h(s)}_{\text{admissible. why?}}$

$\frac{1}{2}g(s) \leq \frac{1}{2}$ True cost $\geq \frac{1}{2}h(s)$

$\Rightarrow \frac{1}{2}g(s) + \frac{1}{2}h(s) \leq$ True cost

(v) (2 points) For a search problem, the path returned by uniform cost search may change if we add a positive constant C to every edge cost.

(T)



(vi) (2 points) With every round of perceptron training, the number of incorrectly classified points always decreases.

(F) eg

i	y^i	$\bar{x}^i$
1	+1	1 1
2	-1	1 -1

for $\bar{w} = [-10^6 \ 1]$

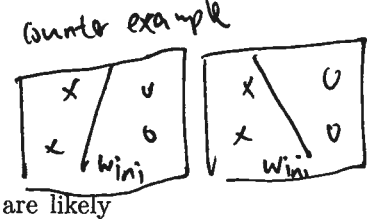
(vii) (2 points) If the data is linearly separable, the perceptron algorithm will always find the same separator, no matter how the weights are initialized.

(F) dual → primal

$$\bar{w} = \bar{w}_{ini} + \sum_i y^i \alpha^i \bar{x}^i$$

or

counter example



(viii) (2 points) Assuming significant noise in the training data, we are likely to increase prediction accuracy on the test set by increasing k in a k-NN classifier.

(T) Increasing k
⇒ less sensitive to local density
⇒ likely to be less sensitive to noise

(ix) (2 points) Assuming significant noise in the training data, we are likely to increase prediction accuracy on the test set by increasing the height of the tree (by doing less pruning).

(F) Less pruning ⇒ higher chance of overfitting to noise

(x) (2 points) Assuming significant noise in the training data, a learning algorithm with a low variance always has a lower cross-validation error than an algorithm with a high variance.

(F) Low variance learning algorithm
⇒ Low variance in cross-validation error
⇏ low cross-validation error
(eg. k-nn k=3 vs k = 1 million)

Question #2

20 points

1. (6 points) The following table shows a training set of several 2-dimensional training data points and their true labels. Let (x_1, x_2) denote each data point, y the corresponding true label, and i the corresponding position in the training set.

i	y	x_1	x_2
1	+1	0	−1
2	+1	1	−3
3	+1	1	−2
4	+1	2	−3
5	+1	3	−4
6	−1	3	−2
7	−1	3	1
8	−1	4	−1

In the table below, trace the perceptron parameter updates for the primal weight vector $\bar{\mathbf{w}}$ and the the dual variables $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_8)$ until convergence, where α_i is the dual variable for the i-th data vector.

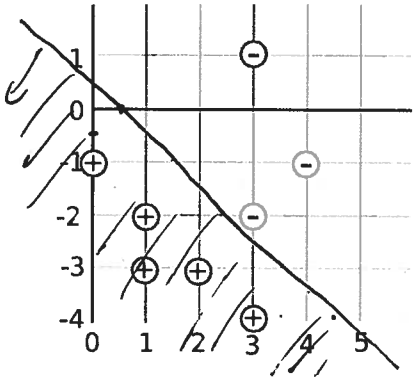
Remarks:

- Let learning rate $\eta = 1.0$
- The first coordinate of $\bar{\mathbf{w}}$ is the the bias b of the hyperplane
- A total of six (6) updates will be performed
- **Fill in a new row only when parameters are updated.**

Iteration	i	$\bar{\mathbf{w}}$	$\boldsymbol{\alpha}$
0	-	0 0 0	0 0 0 0 0 0 0 0
1	1	1 0 −1	1
1	6	0 −3 1	1 1
2	1	1 −3 0	2 1
2	2	2 −2 −3	2 1 1
2	6	1 −5 −1	2 1 2
3	2	2 −4 −4	2 2 0 0 0 2 0 0

check $2\begin{pmatrix}1\\0\\-1\end{pmatrix} + 2\begin{pmatrix}1\\1\\-3\end{pmatrix} - 2\begin{pmatrix}1\\3\\-2\end{pmatrix} = 2\begin{pmatrix}1\\-2\\-2\end{pmatrix}$

2. (2 points) Draw the final decision boundary and shade the region which labels points as +1.



$\bar{w} \cdot \begin{pmatrix} 1 \\ 0 \\ x_2 \end{pmatrix} = 0$

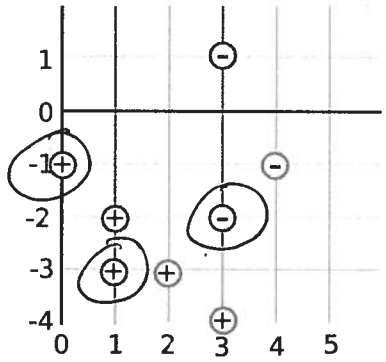
$\begin{pmatrix} 2 \\ -4 \\ -4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ x_2 \end{pmatrix} = 0$

$x_2 = -\frac{1}{2}$

$\begin{pmatrix} 2 \\ -4 \\ -4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \\ -4 \end{pmatrix} =$

$x_1 = \frac{18}{4} = \frac{9}{2} =$

3. (4 points) Consider that case where only one training data point is removed. In some cases, removing a single data point will change the decision boundary of the learned classifier. Circle data points that will cause such a change.



dual form

$\bar{w} = \sum \alpha^i y^i \bar{x}^i$

if $\alpha^i = 0$, not misclassfie during traing

⇒ presence no impact

So circle $\alpha^i \neq 0$ ($i = 1, 2, 6$)

4. (2 points) If the data points were re-ordered, would your answer in the previous question change? If yes, give an example. If no, explain why (in two or fewer sentences).

Yes. Swap $\bar{x}^1 \leftrightarrow \bar{x}^3$. $\bar{x}^3$ misclassified, $\alpha_3 \neq 0$

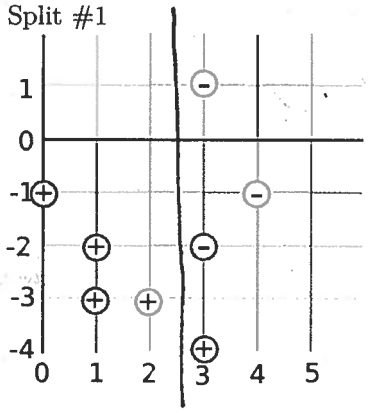
5. (2 points) Here, we consider using the decision tree learning algorithm on the same data set. Suppose each split can only be binary, i.e. it takes one of the following forms:

$$x_1 \leq t,$$
$$x_2 \leq t$$

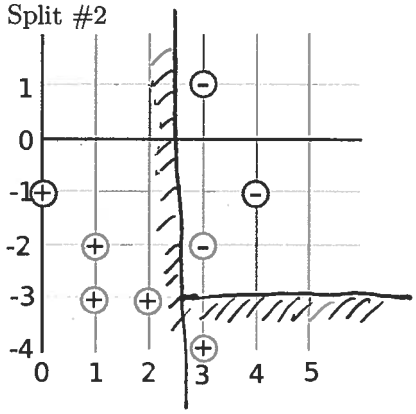
where t is a real number. Also, let the split point be the mid-point of the closest point on either side, i.e. t is a multiple of 0.5.

Draw the decision boundary after each split. (There might be fewer than four splits.) In the final split, shade the region that labels points as +.

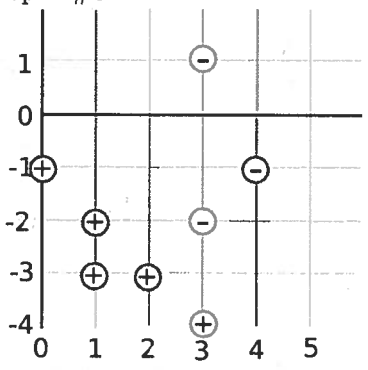
Split #1



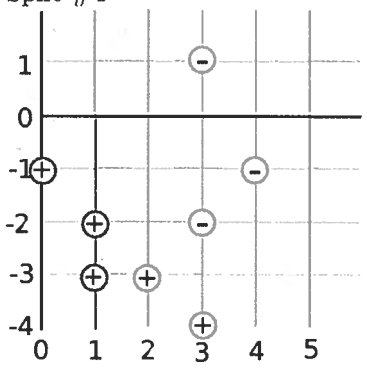
Split #2



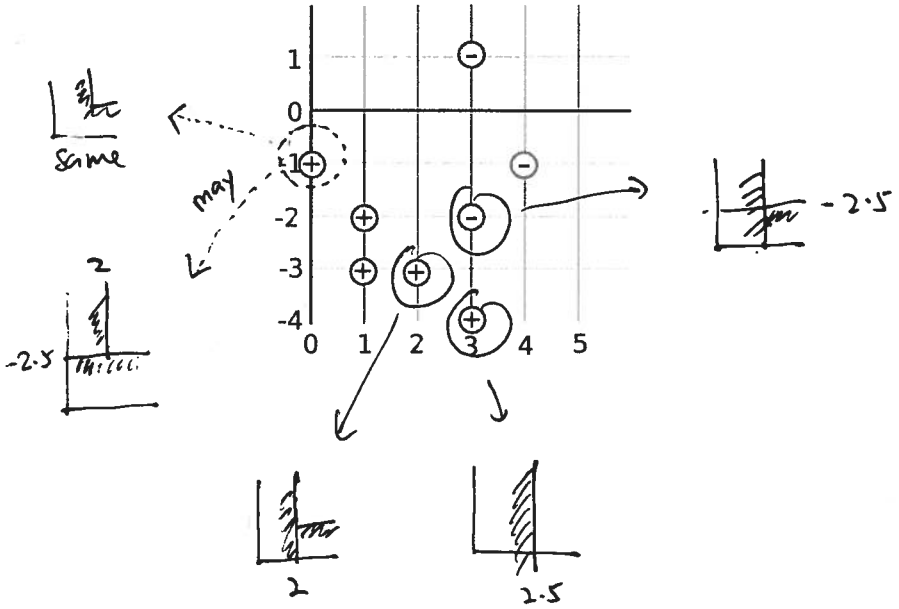
Split #3



Split #4



6. (4 points) Consider that case where only one training data point is removed. In some cases, removing a single data point will change the decision boundary of the learned classifier. Circle data points that will cause such a change.



Question #3 — 20 points

In this question, we consider several variants of the A^* **tree** search algorithm as follows:

(a) Standard A^*

(b) A^*, but we apply the goal test before enqueuing nodes rather than by selecting nodes from the queue Q

(c) A^*, but $PickNode(Q)$ based only on lowest $g(n)$ (ignoring $h(n)$)

(d) A^*, but $PickNode(Q)$ based only on lowest $h(n)$ (ignoring $g(n)$)

(e) A^*, but $PickNode(Q)$ based on lowest $g(n) + h(n')$

(f) A^*, but $PickNode(Q)$ based on lowest $g(n') + h(n)$

where

Q	a queue from which a node is picked
n	a node
n'	a parent of n
$g(n)$	the cumulative path cost from the start node to node n
$h(n)$	a lower bound on the shortest path from node n to a goal state
$PickNode(Q)$	a function which picks from Q according to some criterion

We also assume all costs are *non-negative*.

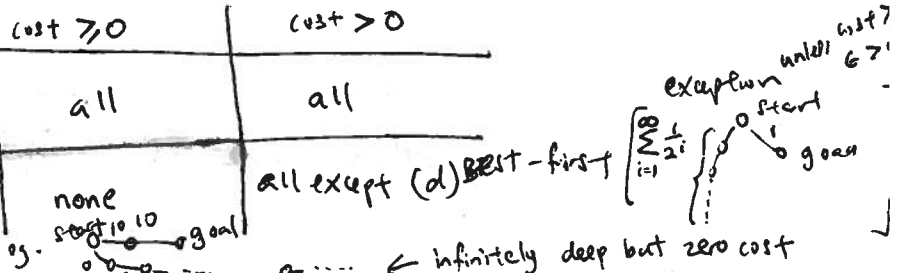
The search algorithm is reproduced below for your convenience:

```
Initialize Q with start node S
Repeat
  If Q is empty
     Fail
  else
     n = PickNode(Q)
  If state(n) is goal
    return n
  Dequeue n from Q
  Enqueue children of n to Q
```

1. (3 points) Which of the above variants are complete (i.e., guaranteed to find a solution when there is one), assuming all heuristics are admissable?

	cost ≥ 0	cost > 0
finite depth	all	all
infinite depth	none	all except (d) best-first

e.g. start 10 10 goal



← infinitely deep but zero cost

2. (3 points) Which of the above variants are optimal (i.e., guaranteed to find the best cost solution), assuming all heuristics are admissable?

A* (a) and (c) uniform cost

(b) — may miss better goals to be seen later (eg. goal with long initial path but short last step)

(d) — best-first

(e) — inadmissible $h(n)$ may be $> h^*(n)$ (f) (b) start → n' ····· n (goal)

Upper Bounds A^* exploits lower bounds h on the true completion cost h^*. Suppose now that we also have an upper bound $k(n)$ on the best completion cost (i.e., $\forall n, k(n) \geq h^*(n)$). Consider the point at which you are inserting a node n into the queue.

3. (3 points) Assume you are required to preserve optimality. In response to n's insertion, can you ever delete any nodes m currently on the queue? If yes, state a general condition under which nodes m can be discarded, if not, state why not. Your answer should involve various path quantities (g, h, k) for both the newly inserted node n and other nodes m on the queue.

Delete nodes m such that

$$g(m) + h(m) > g(n) + k(n)$$

$g(m) + h(m)$: best case for m

$g(n) + k(n)$: worse-case for n

4. (3 points) Assume that your goal now is to find some solution of cost less than some threshold t (if one exists). Your solution does not have to be optimal. In response to n's insertion, can you ever delete any nodes m currently on the queue? If yes, state a general condition, if not, state why not. Your answer should involve various path quantities (g, h, k) for both the newly inserted node n and other nodes m on the queue.

Delete node m if $g(m) + k(m) \geq t$

ϵ-Admissible Heuristics: Suppose that we have a heuristic function which is not admissible, but *ϵ-admissible*. That is, for some known $\epsilon > 0$ and for all nodes n:

$$h(n) \leq h^*(n) + \epsilon$$

where $h^*(n)$ is the optimal completion cost. In other words, $h(n)$ is never more than ϵ from being optimal.

heuristic function

5. (1 point) Is using A^* with an *ϵ-admissible* complete? Briefly justify.

	cost ≥ 0	cost > 0	
finite	✓	✓	not interesting
Infinite	X	Yes	node n on infinitely long path has infinitely large g(n)

6. (2 points) Assuming we utilize an *ϵ-admissible* heuristic in standard A^* search, how much worse than the optimal solution c^* might our solution be?

Suppose node n is found to be a goal.
Its ~~cost~~ score is $g(n) + h(n) = c^* + h(n)$
$\leq c^* + \epsilon$

7. (5 points) Suggest a modification to the A^* algorithm which is guaranteed to yield an optimal solutions using an *ϵ-admissible* heuristic with fixed, known ϵ. Justify your answer.

Solution A

replace h(n) by $\max(h(n) - \epsilon, 0)$ which is admissible

Solution B

After seeing first goal n, keep searching until we see ~~another~~ goal with score $> g(n) + h(n) + \epsilon$

Question #4 — 20 points

1. (6 points) In this question you will perform alpha-beta pruning on a standard game tree from left to right. The game tree is shown as a table as in the problem set — Each cell in the table corresponds to a node in the tree and the cells directly below a cell represent the children of that node.

   In each box, you have to fill in two numbers as in the problem set. The numbers are different depending if the box is the top or the bottom row of the MAX or MIN function:

   - In the top box: the values of alpha and beta passed into each call to MAX-VALUE and MIN-VALUE (not values which are returned from MAX-VALUE and MIN-VALUE)
   - In the bottom box: the value returned by MAX-VALUE or MIN-VALUE, and followed by the number of static evaluations that were done in the subtree below the box (not the actual MIN-MAX value of each subtree)

   arks:

   - If a node is pruned, so that no alpha or beta values are computed for it, enter "X" in the boxes.
   - Do not leave any box empty.

MAX	$-\infty$ ∞													
	10 11													
MIN	$-\infty$ ∞				5 ∞						5 ∞			
	5 4				5 4						10 3			
MAX	$-\infty$ ∞		$-\infty$ 5		5 ∞		5 7		X		5 ∞		5 10	
	5 2		10 2		7 2		5 2		X		10 2		11 1	
STATIC	5	−1	−1	10	2	7	−8	1	10	4	7	10	11	−4

↑ pruned

2. The standard Minimax algorithm calculates worst-case values in a *zero-sum* two player game, i.e., a game in which for all terminal states s, the utilities for players A (MAX) and B (MIN) obey $U_A(s) + U_B(s) = 0$. In the zero sum case, we know that $U_A(s) = -U_B(s)$ and so we can think of player B maximizing $U_B(s)$ as simply minimizing $U_A(s)$.

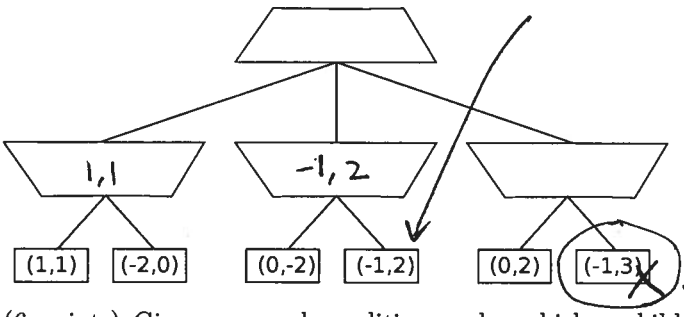
   In this question, we will consider the *nearly zero sum* case, in which $|U_A(s) + U_B(s)| \leq \epsilon$ at all terminal nodes s for some ϵ which is known in advance. For example, the game tree in the following figure is nearly zero sum for $\epsilon = 2$. In this game, each player still maximizes his or her own utility, but this no longer always minimizes the other player's utility.

   (a) (8 points) Draw a "X" in each node in this game tree which could be pruned with the appropriate generalization of alpha-beta pruning. Assume that the exploration is being done in the standard left to right depth-first order and the value of ϵ is known to be 2. Make sure you make use of ϵ in your reasoning.

$U_B(s) \leq U_A(s) \leq -U_B(s) + \epsilon$
$- \epsilon$

upon seeing (0,2)
A knows $U_A(s)$
can only get worse or equal to than zero
$\therefore U_B(0,2) = 2$,
$\Rightarrow U_A$ for all nodes in subtree
$\leq -2 + \epsilon = 0$
Best option for A so far gives $U_A(s) = 1$ from (1,1), so prune (-1,3)

can't prune this. why? what if it turns out to be (1.5, 1.5)?



← node b
← node n
← pruned node (If player B picks this, $U_A^{(n)}$ never $\geq$ best $U_A = \alpha = 1$)

   (b) (6 points) Give a general condition under which a child n of a B node (MIN node) b can be pruned. Your condition should generalize α-pruning and should be stated in terms of quantities such as the utilities $U_A(s)$ and/or $U_B(s)$ of relevant nodes s in the game tree, the bound ϵ, and so on. Do not worry about ties.

Prune node n if $\exists$ node s which is a sibling left of n
such that $-U_B(s) + \epsilon \leq \alpha$
where α is initialized to $-\infty$, and
and player A updates $\alpha = \max(\alpha, U_A(b'))$
↑ returned by player B

If we generalize to both α, β (initialized to $(-\infty, -\infty)$)
updated by $\alpha = \max(\alpha, U_A(b))$ $\beta = \max(\beta, U_B(a'))$
then prune with $-\beta + \epsilon \leq \alpha \equiv \alpha + \beta \geq \epsilon$