CS B551 –Elements of Artificial Intelligence– Fall 2011

Homework #7
(Machine Learning and Agents)
Due: 11/17/11 (5:15 pm)

How to complete this HW: Either 1) type your answers in the empty spaces below each problem and print out this document, or 2) print this document and write your answers in the empty spaces on the printout. Return the homework in class, during office hours, or slip it under the door of Info E 257 before **5:15 on Thursday, 11/17/11**.

Your name: Enrique Areyan

Your email address:

Note on Honor Code: You must NOT look at previously published solutions of any of these problems in preparing your answers. You may discuss these problems with other students in the class (in fact, you are encouraged to do so) and/or look into other documents (books, web sites), with the exception of published solutions, without taking any written or electronic notes. If you have discussed any of the problems with other students, indicate their name(s) here:
N/A
Any intentional transgression of these rules will be considered an honor code violation.

General information: Justify your answers, but keep explanations short and to the point. Excessive verbosity will be penalized. If you have any doubt on how to interpret a question, tell us in advance, so that we can help you understand the question, or tell us how you understand it in your returned solution.

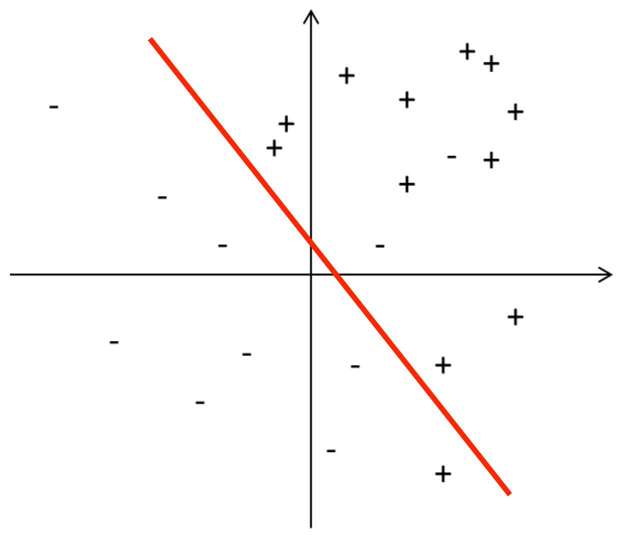
Grading:

Problem#	Max. grade	Your grade
I	20	
II	25	
III	25	
IV	30	
Total	100	

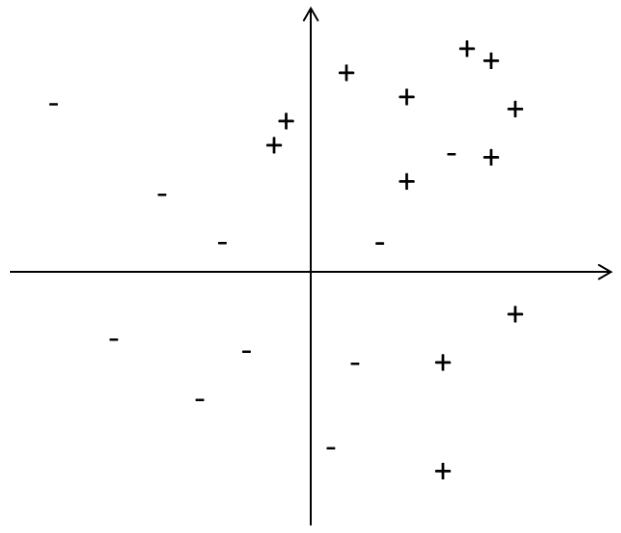
I. Nonparametric Learning (20 points)

1. Given the following dataset on two attributes (x1, x2), draw the decision boundary corresponding to:

   (a) A linear classifier, approximately as it would be fit using either a neural network or SVM learning technique.



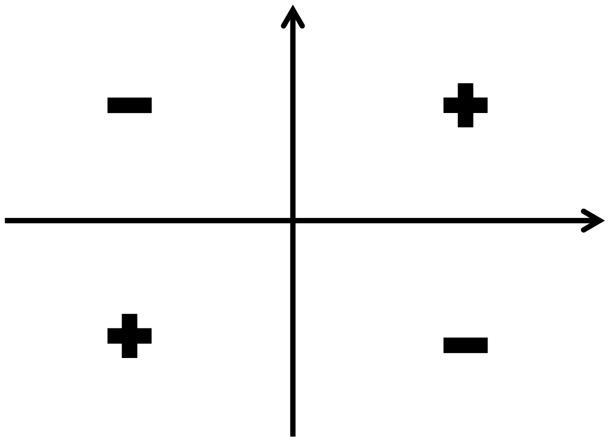
   (b) A nearest-neighbors classifier. (Your drawing need not be exact, but should capture most of the relevant details.)



2. How many errors does a nearest neighbor classifier make on a training set, assuming no two training examples have exactly the same attributes?

The training example depends on the choice of *k* (number of neighbors) and the data distribution. However, we can be certain that if *k*=1 and the function *d* used for the classification task is a proper distance metric, then the training error will always be zero. This is because when querying a training example *t* the result will always be its own label (*d*(*t*,*t*)=0, and no two training examples have the same attributes).

Assume k=3. Draw an example training set on which a k-nearest neighbor classifier achieves 100% training error.
In the next figure, all examples are equidistant from all other examples except the example in the opposite diagonal. These are the only example on the training set.



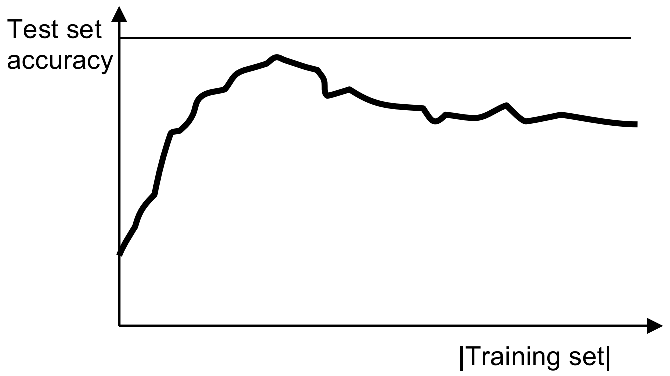
3. Describe the difference between nonparametric learning and parametric learning (e.g., function learning). Give at least two strengths of nonparametric learning relative to parametric learning, and vice versa.

On the one hand, in parametric learning, given a training data **D** and a function class **C**, we are to find the set of parameters **P** such that a function $f_{\mathbf{P}}(\mathbf{D})$ fits the data well (or we can also think of the function as doing a good job explaining the data). This means that we need to choose the space of functions **C** that we are going to search in advance, and it remains fix throughout the learning task. No matter how much data one throws at the classifier, the structure of the model remains the same (either you have a linear classifier, quadratic, etc). An advantage of this model with respect to nonparametric learning is that a good model discovered with this method will give us a better intuition of what is actually happening in the real world. Also, once we are done with the training data, we can discarded and just save the parameters **P**.

On the other hand, in nonparametric learning, we are also given a set of data **D**, but we are not constrained to search within a fix space of functions. In other words, the number of parameters **P** can change with the amount of data given to the classifier. The naïve example will be that of a lookup table. The advantage of this model with respect to the later is that we do not need to have any kind of knowledge about what model could best fit the data. Also, the model can grow with the data set and it may be more scalable.

II. Machine Learning (25 points)

1. John has devised a new classification learning algorithm, JohnClassify. When evaluating the performance of JohnClassify on a real-world training set and testing set, he observes the following learning curves:



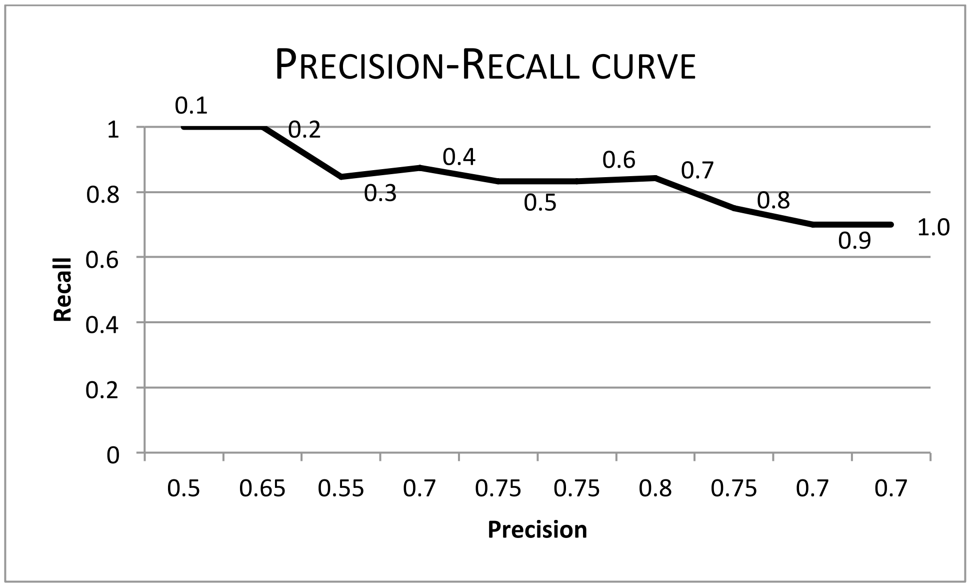
John observes that the accuracy of the classifier on the testing set decreases as the training set grows larger. What is the term that describes this problem? **Data Overfit**

2. To predict the classification of a new data point x, JohnClassify produces a real valued number p(x) that estimates the probability that CONCEPT(x) is true. On John's testing set, he observes the following predictions:

CONCEPT(x)	p(x)
True	0.9
True	0.8
True	0.8
True	0.5
True	0.3
True	0.3
False	0.8
False	0.7
False	0.5
False	0.5
False	0.4
False	0.4
False	0.4
False	0.2
False	0.2
False	0.2
False	0.1
False	0.1
False	0.1
False	0.1

Plot and label the precision/recall curve of the classifier in the space below. What prediction threshold would you use if you prefer balanced accuracy on positive and negative examples? If false negatives are considered twice as bad as false positives? If false positives are twice as bad as false negatives?

I did the calculations on an excel spreadsheet, using ten thresholds in increment of 0.1, i.e., 0.1, 0.2, 0.3, …, 1. Here is the graph from excel. In the graph, I have labeled the threshold value at each point on the curve.



If I prefer balanced accuracy on positive and negative examples, I would pick threshold 0.7.
If false negatives were considered twice as bad as false positives, I would pick threshold 0.2.
If false positives were twice as bad as false negatives, I would pick threshold 0.9.

3. Assume JohnClassify has a complexity parameter C that governs the complexity of the hypothesis class considered during learning (larger C implies higher complexity). To answer the prior questions, John set C to a fixed default parameter C_1. Now John is considering a new complexity parameter C_2. In his tests, he finds that when he uses C_2, he achieves a lower test error than when he uses C_1. What is the potential danger in concluding that C_2 is a superior parameter value? What steps might John take to reduce these risks?

The potential danger in concluding that C_2 is a superior parameter value, is that C_2 may be overfitting the particular data being used and thus, may not scale well to larger and/or different data sets. John might want to test the classifier, both with C_1 and C_2, on different data sets and analyze its performance. He can also plot the accuracy of both parameters with different data sets and try to understand the behavior. If it is feasible to come up with a third parameter C_3 with complexity between C_1 and C_2, he can use this other parameter in the analysis to smooth the transition between parameters. If not, C_3 can be constructed by using a subset of C_2. The general idea is to try to detect if the parameters are overfitting the data and if so, lower their complexity.

III. Codebreaking Agent (25 points)

Consider the "code-breaker" game illustrated below, where the objective is to guess a 4-color secret code (which is unknown to the player) within N guesses. There are C possible colors. After the player performs a guess, the opponent replies with A) the number of colors correct and in the correct spot (illustrated by a black dot), and B) the number of colors correct but *not* in the correct spot (illustrated by a white dot).

1. Formulate the player in an agent-based framework. Give:
   a) The type of task environment (observable/non-observable, stochastic/deterministic, episodic/non-episodic, static/dynamic, single/multi agent).
   Let us consider the agent to be the one trying to discover the colors, and the opponent just as part of the environment. Then the agent-based formulation is:
      i) **Partially observable**: the agent can perceive the arrangement of the color guesses and the reply from the environment. However, the agent cannot directly see the correct color combination.
      ii) **Nondeterministic**: although the player can be certain of any new arrangement of color, it is not able to determine a priori whether or not the arrangement will matched that of the secret code.
      iii) **Non-episodic**: steps taken at each time depend on previous ones if the code is to be discovered in fewer steps than by brute-force enumeration.
      iv) **Static**: the configuration of the secret code is fixed at the beginning of the game and it does not change throughout the entire duration of the play.
      v) **Single agent**: as mentioned before, one can think of this game as having only one agent trying to decipher the code. The responses to each color change made by the agent are part of the environment. One can think of the agent as trying to solve a puzzle by itself, using a mechanism to check the correct configuration.

   b) The set of percepts, and the number of possible percepts.
      i) $P = \{(c_1, c_2, c_3, c_4)\} \cup \{(r_1, r_2, r_3, r_4)\}$, where c_i is any of the n possible colors and r_j is either a black or with dot or a special character, lets say *, to denote that a response is empty, e.g., (b,w,w,*) denote a response in which one guess is in the correct place, two of them are the right color but not in place, and the remaining one is empty, meaning is not the right color nor in the right place.
      ii) The number of possible percepts is $|P| = n^4 + 3^4$. On the one hand, there are four possible choices of colors. On the other, there are 3^4 responses.

   c) The set of actions, and the number of possible actions.
      i) The possible actions are to change one or more of the four colors.
      ii) The number of possible actions is the combination of: a) the number of possible changes to one color, b) the number of possible changes to two colors, c) the number of possible changes to three colors, and d) the number of possible changes to four colors. Thus,

$$\binom{4}{1}(n-1)^1 + \binom{4}{2}(n-1)^2 + \binom{4}{3}(n-1)^3 + \binom{4}{4}(n-1)^4 = \sum_{i=1}^{4}\binom{4}{i}(c-1)^i$$

   d) The performance criterion.

    i) Maximize the number of black dots in the response. As a second criterion, the agent can maximize the white dots, but never in detriment of the black dots. As a third criterion, it can minimize the number of empty responses.

  e) The number of possible simple reflex agents.

    i) Simple reflex agents select actions on the basis of the current percept, ignoring the rest of the percept history. Therefore, all a simple reflex agent could do is change the color of one, two, three or four parts of the code and check the result configuration. There is no history, so it cannot discard bad moves by comparing with the result information at each step. With this configuration, we would have 4 agents that would stop once the results (b,b,b,b) is found. However, if the agents are allowed to randomize its actions, we could have agents switching from one of these four possible agents to another, e.g., flip a coin, if it is less than a certain threshold then change one color, if it is not then change two colors, etc. Thus, with random actions, we could have any combination of these simple 4 agents.

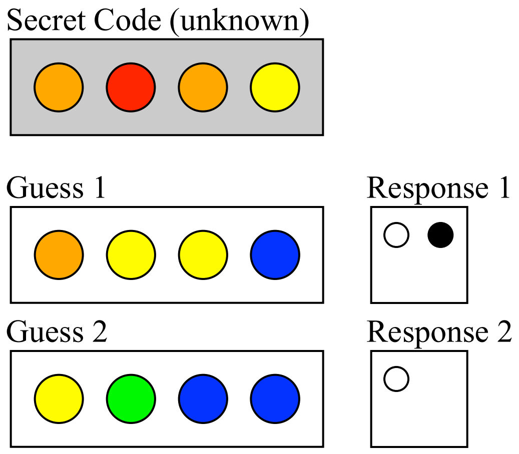
(Explain your answers.)

2. Discuss the advantages/disadvantages of the following types of agent on this task.

  a) Simple reflex agent.
    i) Advantages: easy to implement.
    ii) Disadvantages: without considering the percept history, these kinds of agents may fail to converge in less number of iterations than brute-force search (at the worst case they will perform brute-force search).
  b) Model-based reflex agent:
    i) Advantages: it can construct a model of the game and keep track of previous states.
    ii) Disadvantages: without a measure of what the goal is, the model does not really help it decide what actions to perform next.
  c) Rational, goal-based agent.
    i) Advantages: together with a model of the world, this agent can choose a minimal set of steps to reach its goal, i.e., all results' dots in black.
    ii) Disadvantages: for this simple task there is no relevant disadvantage on using a rational, goal-based agent (assuming it also has a model of the world)
  d) Learning-based agent.
    i) Advantages: the only use a learning-based agent would have for this task is if the secret code follows a certain pattern over different runs of the game. If so, the agent could learn to arrive at the goal faster by remembering previous secret code configurations.
    ii) Disadvantages: however, if the secret code is chosen at random (with uniform probability), then it may not be worthwhile to implement a learning agent. A rational, goal-based agent could encode all the necessary information to "break" the code.

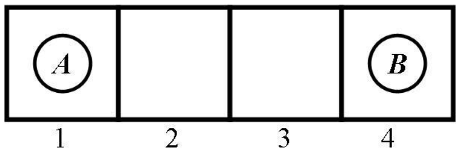
3. Briefly describe an agent program that can break *any* code in fewer guesses than the C^4 guesses needed by brute-force enumeration.

```
function Model-Goal-Based-CodeBreaker-Agent() returns an action
    persistent history of guesses, history of results;
     //Decide a move
    if last-results is null and history length is zero //There is not a single color or position correctly guessed and we are beginning the game
        return change-all-colors
    else //There is at least one position correctly guessed and we have some history. Return an action using history.
        if two moves ago we had a black and one move ago we don't, rollback the changes and try to figure out which part of the code is correct.
            Return changes;
      ... //program more of these rules.
      if previous result is all black
         return null; //code is already known, do nothing
```



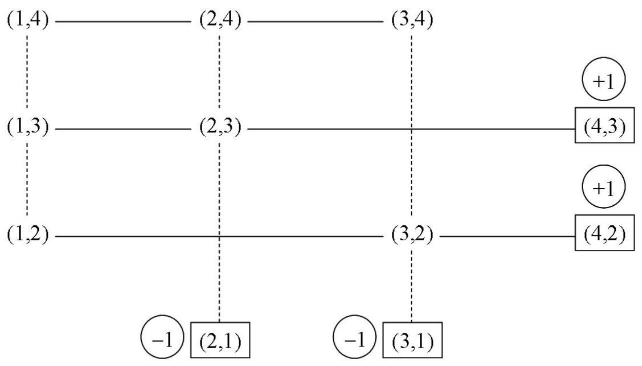
IV. Adversarial Markov Decision Problems (30 points)

Consider the following two-player game on the board depicted below, with 4 squares arranged in a line and numbered 1 through 4.



Each player has a single token. Player *A* starts with his token at location 1 and player *B* with his token at location 4. Player *A* moves first. The two players take turns. When it is *X*'s turn to play (where *X* = *A* or *B*), he must move his token to an open adjacent location in either direction. If the opponent's token is at an adjacent location, then *X* may jump over his opponent's token to the next location, if any. For example, if *A* is at 3 and *B* at 2, then *A* may move to 1. The game ends when either *A*'s token reaches location 4 (then *A* wins) or when *B*'s token reaches location 1 (then *B* wins). The value of the game for a player is +1 when he wins and –1 when he loses.

The state-space graph for the game is shown below with solid lines for *A*'s possible and dashed lines for *B*'s possible moves:



There are four terminal states shown with square boxes. Actions have no cost. **All rewards and utilities are expressed from *A*'s point of view.** The rewards in the terminal states are shown within circles near the corresponding states. The rewards in all other states are 0. Let $U_A(s)$ denote the utility of a state s when it is *A*'s turn to move and $U_B(s)$ its utility when it is *B*'s turn to move.

1. The Bellman equation defining $U_A(s)$ is:

$$U_A(s) = R(s) + \max_{a\in \mathrm{Appl}(A,s)} \Sigma_{s'\in \mathrm{Succ}(s,a)} \mathrm{P}(s'|s,a)\times U_B(s')$$

   where $R(s)$ denotes the reward collected in state *s*, Appl(*A*,*s*) is the set of all possible moves of *A* in *s*, Succ(*s*,*a*) is the set of all states that can be reached by performing move *a* in *s*, and P(*s*'|*s*,*a*) is the probability of reaching state *s*' from *s* by performing move *a*. [Here, since there is no uncertainty in executing an action, Succ(*s*,*a*) contains only one state *s*' and P(*s*'|*s*,*a*) = 1. However, we keep the equation in its general form for Question 4 below.]

   Write down the equation defining $U_B(s)$.

$$U_B(S) = R(S) + min_{\ a\in Appl(B,s)} \Sigma_{s' \epsilon Succ(s,a)} P(s'|s,a)\times U_A(s')$$

2. Briefly explain how to perform value iteration with the above two equations and fill the following table using value iteration. The utilities at non-terminal states in the first row of the table have been initialized to 0. The utilities at terminal states in the first row are set to the rewards collected in these states. Fill the next two rows.

	(1,4)	(2,4)	(3,4)	(1,3)	(2,3)	**(4,3)**	(1,2)	(3,2)	**(4,2)**	**(2,1)**	**(3,1)**
U_A	0	0	0	0	0	+1	0	0	+1	−1	−1
U_B	0	0	0	0	-1	+1	0	-1	+1	-1	-1
U_A	0	0	0	-1	+1	+1	-1	+1	+1	-1	-1

3. Define a suitable termination condition for value iteration in this example.

When the change between the last rows of the same player is bellow a certain threshold ε. As a metric to measure the change, we could use Manhattan distance. If we want the iteration to converge, simply set $\varepsilon = 0$. If you can't wait very long, set $\varepsilon > 0$, a small number.

4. Let us now assume that each player tosses a coin before choosing a move. If the coin lands showing heads, then the player must move, otherwise no move is allowed and the other player takes turn. (Therefore, a player will move his token with probability 0.5 and will not move it with probability 0.5.) Using value iteration, fill the three empty rows in the following table.

	(1,4)	(2,4)	(3,4)	(1,3)	(2,3)	**(4,3)**	(1,2)	(3,2)	**(4,2)**	**(2,1)**	**(3,1)**
U_A	0	0	0	0	0	+1	0	0	+1	−1	−1
U_B	0	0	0	0	-1/2	+1	0	-1/2	+1	-1	-1
U_A	0	0	0	-1/4	1/4	+1	-1/4	1/4	+1	-1	-1
U_B	-1/8	1/8	1/8	-1/4	-3/8	+1	-1/4	-3/8	+1	-1	-1