Problem Set 6

EECS 492: Winter 2017

Due: Monday, April 17, 11:59pm

Machine Learning

Note that while this assignment has six tasks, several of the tasks are smaller or more straightforward than previous tasks. Start working early and you shouldn't have any issues completing this assignment by the deadline.

Task 1: Decision Trees (20 points)

You are given a set of 8 training examples from a cupcake data set and are asked to construct a decision tree to classify them. The attributes are whether the cupcake has filling, is chocolate, has sprinkles, and is wheat. The label is a binary Yes or No as to whether Luke likes this cupcake. The following table shows each example along with its corresponding class and the values of the four abbreviated attributes.

example	Fill	Choc	Spr	Wheat	Likes
x1	True	False	True	True	Yes
x2	False	False	False	False	No
x3	False	True	False	True	Yes
x4	False	True	True	True	Yes
x5	True	False	True	False	No
x6	False	False	True	False	Yes
x7	True	False	False	True	No
x8	True	False	False	False	Yes

1. Using the decision tree learning algorithm determining splits based on information gain, build a new decision tree for this data. Show all information gain calculations and draw a final (learned) decision tree. Break ties in information gain according to the order of the attributes in the table, preferring earlier columns in the table.

2. Give a worst case bound on the number of nodes in the decision tree learned from the method from part (1) from an arbitrarily large training set in which each example has k features (including class), and b different values for each feature. Allow nodes to branch into b branches. [Assume nodes include both internal nodes and leaf nodes, where leaf nodes indicate the classification.].

3. Beyond the initial training examples, you are given a new set of examples. Consider the new set of examples for this problem.

example	Fill	Choc	Spr	Wheat	Likes
x9	False	True	False	False	Yes
x10	False	True	True	True	No

Does the decision tree you created correctly classify these examples?

4. (a) Does every decision tree that is consistent with a data set need to implement the same classification function? If so briefly state why; if not provide a counterexample.

   (b) Similarly, is it possible for two decision trees to implement the same function and yet have a different tree structure? If so, provide an example, if not briefly explain why.

   (c) Consider a set of T decision trees such that each tree t in the set achieves 100% classification accuracy on a set of training examples S. What fraction of all possible different examples needs to be in S to guarantee that all of the trees in T implement the same classification function

Task 2: Ensemble Learning (15 points)

1. Consider ensemble learning on the problem in Task 1, using the AdaBoost algorithm as given in R&N (page: 751). Assume, as in the book, that our learning algorithm is limited to only learning "decision stumps" - which are decision trees that split on only one attribute and each resulting leaf must be labeled with a class. Your ensemble learner should use the maximum information gain using the weighted examples, breaking ties by the order of the attributes in the table.

   Use training examples x1 through x8 in the Task 1 table and answer the following questions. (Note: use natural log for calculating your answers.)

   (a) What are the weights used for the training examples to do the first iteration of the algorithm?

   (b) Generate the initial decision stump (for k=1) given the AdaBoost algorithm. What attribute is chosen for splitting?

   (c) What is the error for this decision stump hypothesis? Enumerate the incorrectly classified examples.

   (d) What is the weight (z) of this hypothesis?

   (e) What classification would this 1-hypothesis ensemble give (using weighted-majority and random tie-breaking) to the test examples from Task 1(x9 and x10). What is the success rate?

   (f) What are the weights that should be assigned to each of the training examples for the next iteration (Note: NOT x9 and x10)?

   (g) Now generate the second decision stump (for k=2) given the AdaBoost algorithm. What attribute is chosen for splitting now?

(h) What is the error for this decision stump? Again, enumerate the incorrectly classified examples.

(i) What is the weight (z) of this second hypothesis?

(j) What classification would the resulting 2-hypothesis ensemble (using weighted-majority and random tie-breaking) give to the test cases x9 and x10?

Task 3: PAC-Learning (10 points)

Now, we will briefly investigate the principles of Probably Approximately Correct (PAC) learning for a Boolean classification problem. Let this problem have two binary attributes, one trinary attribute, and one 4-ary attribute. Assume a hypothesis is "bad" (i.e. not approximately correct) if it classifies more than two examples wrong. For the problems below, you may assume each example is equally likely to be seen.

1. What is the hypothesis space for this problem? How many hypotheses are in the initial hypothesis space (this is the hypothesis space before any examples are seen)? You may leave your answer in exponential form.

2. How many of the hypotheses in the initial hypothesis space are not bad (i.e. approximately correct)?

3. What is the probability that a randomly-chosen hypothesis from the initial hypothesis space will be approximately correct?

4. What value of ϵ corresponds to the definition of approximately correct for this problem, assuming examples are all equally likely to be encountered?

5. What is the upper bound on the probability a bad hypothesis will classify the first 12 training examples correctly for this problem?

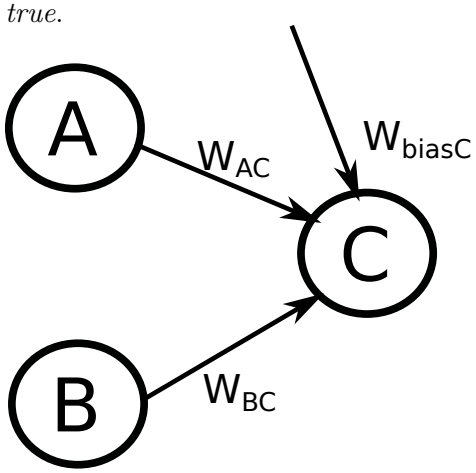
6. After 40 (distinct) examples are observed, how large is the remaining hypothesis space?

7. What is the probability that a randomly-chosen hypothesis that is consistent with the 40 (distinct) examples seen so far will be approximately correct? To better appreciate the benefit of having seen these examples also answer the following question: How many times larger is this probability than the probability in part c of choosing an approximately correct hypothesis without having seen any examples?

Task 4: Artificial Neural Networks (15 points)

This problem asks you to design artificial neural networks that will result in desired logical functions. Assume that all of the units are **perceptrons**, and that the **bias** input to all units is -1. When specifying weights, you should use integers for all weights **except** for the bias weights which should be integers + 0.5 (e.g. ... -1.5, -0.5, 0.5, 1.5, ...). All of the units use a threshold function as the activation function.
The possible values for each input are 0 and 1, where 0 corresponds to *false* and 1 corresponds to

true.



1. For the above network, find weights ($W_{A,C}$, $W_{B,C}$, $W_{bias,C}$) that implement the function A NOR B. That is, C will be activated if and only if the expression A NOR B is true for the values of boolean inputs A and B.

2. Now consider all Boolean functions of 2 variables. Which of these functions cannot be implemented by a single-layer perceptron network? Briefly justify your answer. (Hint: This shouldn't take a lot of work!)

3. Choose one from the functions you identified in part (b.) that couldn't be implemented by a single-layer perceptron, AND was not identified in class and use a multi-layer perceptron with one hidden layer of two nodes to implement it (using the weight conventions mentioned above). Please indicate which function you are implementing!

4. Consider a perceptron learning rule that contains a learning-rate parameter (see equation 18.7 in RN). Give only one or two sentences for each of these questions.

   (a) What is an advantage of a large learning rate?

   (b) What is an advantage of a small learning rate?

Task 5: Naïve Bayes (20 points)

Suppose that you are in charge of sorting desserts before they are given to Luke. Luke is terrible at identifying pastries, so you need to make an automatic identification method. Because we need to implement it quickly, the agent will determine which type of dessert it is based on a Naïve Bayes Classification.

1. We need to determine the prior probability of each classification, $p(C_k)$. The baker tells us that there are these, and only these, classifications of pastries (and their classification numbers) in the order: 20 Danishes(k=1), 27 Macaroons(k=2), 31 Strudels(k=3), and 22 Cannolis(k=4). From this information, what are the prior probabilities of each of the 4 classifications?

2. Suppose that Luke bursts through the door, and demands his classified desserts immediately, so you can use only the most simple model, consisting of just the "Class" node that can take on

any of the 4 classification values (1 through 4). For each of the test examples below, how would the agent using this very simple (one node!) Bayesian Network to classify the example? (If any ties occur, break in class order, $1 > 2 > 3 > 4$.) What is the accuracy of the classification (what fraction of the test cases are correctly classified)?

Sample	**Sweetness**	**Filled**	**Baker**	**Class**
x1	Very	Yes	Ryen	1
x2	Moderately	Yes	Ryen	4
x3	Extremely	Yes	Mark	3
x4	Extremely	No	Spruce	1
x5	Extremely	Yes	Spruce	3
x6	Very	Yes	Spruce	3
x7	Moderately	No	Mark	2

Luke wasn't particularly happy with your predictions from the last part, but he decided to give you one more chance for he is a just and merciful TA. Now assume we have time to find better estimates for the likelihoods by creating the complete Naïve Bayes model including CPTs. Using earlier observations, a friendly math loving baker has already identified the CPT entries for the attributes (Sweetness(S), Baker(B), Filled(F)) when the classification is Danishes or Macaroons, and these are given below. If a entry is not given, assume it has zero probability.

p(S=Extremely|C_1)=0.4, p(S=Very|C_1)=0.4, p(S=Moderately|C_1)=0.2
p(F=Yes|C_1)=0.4, p(F=No|C_1)=0.6
p(B=Ryen|C_1)=0.3, p(B=Spruce|C_1)=0.5 p(B=Mark|C_1)=0.2

p(S=Extremely|C_2)=0.2, p(S=Very|C_2)=0.2 p(S=Moderately|C_2)=0.6
p(F=Yes|C_2)=0.2, p(F=No|C_2)=0.8
p(B=Ryen|C_2)=0.2, p(B=Spruce|C_2)=0.2 p(B=Mark|C_2)=0.6

3. Now use the following classified examples to determine the likelihoods for Strudels and Cannolis in the same form as in the previous part.

Sample	**Sweetness**	**Filled**	**Baker**	**Class**
x10	Moderately	No	Spruce	3
x11	Extremely	Yes	Mark	3
x12	Very	Yes	Spruce	3
x13	Very	Yes	Spruce	3
x14	Extremely	Yes	Spruce	3
x15	Very	Yes	Mark	3
x16	Extremely	No	Ryen	3
x17	Extremely	Yes	Spruce	3
x18	Extremely	Yes	Mark	3
x19	Moderately	No	Spruce	3

Sample	Sweetness	Filled	Baker	Class
x20	Very	Yes	Ryen	4
x21	Moderately	Yes	Spruce	4
x22	Very	No	Ryen	4
x23	Moderately	Yes	Ryen	4
x24	Moderately	Yes	Mark	4
x25	Very	Yes	Spruce	4
x26	Moderately	Yes	Ryen	4
x27	Very	Yes	Mark	4

4. Now we have determined estimates for the priors and likelihoods, and are ready to make our Naive Bayes predictions. Draw the Naive Bayes network and use your model to classify each of the pastries from part 2. Will Luke be happier with the classifications this time around?

Task 6: Q Learning (20 points)

Our reinforcement learning agent will use exploration/exploitation, using a simple parameter to select which to do. When exploration is chosen, a random action is taken no matter what is known, allowing the agent to learn. When exploitation is chosen, the agent makes the best decision based on what it has learned, taking the action with the highest Q value in that state, enabling the agent to use the information it has learned to maximize reward at the cost of potentially gaining more information.

You will implement an agent that learns the Q values by interacting with an environment from the previous homework assignment. One small difference is set $P_R = 0.8$, so that the rover has only a 20 percent chance of breaking on rocks. Our Q-Learning agent however knows substantially less than our agent from the last homework. It has no idea the layout of the environment, or any other information other than the reward it just received, the current state, and what it has learned. We also will let it know the total number of states (17), numbered 0-16, although this is not necessarily required in order for the agent to learn the environment. Note that state 16 is the BROKEN state.

There are two key components of the program you will be writing. Firstly a simulator that essentially takes in an action and current state and outputs the resulting state (in order to account for the randomness of breaking). You will also have an agent which implements the following pseudocode, but needs to ask the simulator function what its actions actually result in. Using a programming language of your choice, implement the Q-Learning algorithm according to the following pseudocode:

```
Initialize learning rate alpha = 0.05
Initialize discount parameter gamma = 0.95
Initialize explorationchance = 1.
Initialize Q values to 0.  This should include a Q value for each action-state pair.
Remember that all 5 actions are available in each state.
Initialize trials = 0 and initialize iterations = 0
Initialize action = "NONE"
Initialize learning = True
```

```
While learning:
     Read newstate and reward (which can be garbage if first round)
     if action in ["N","S","E","W","R"]:
          Find the Q value for state and action, called q here.
          Find the Q value for newstate with its action with the maximal Q value, q'
          Update q according to:  q = q + alpha * (reward + (gamma * q') - q)
     set state = newstate
     if there have been 1000 trials:
          set learning = False
     choose exploration or exploitation according to explorationchance
     if exploitation:
          set action as action with the highest Q value for newstate
     if exploration:
          set action to one of "N", "S", "E", "W", or "P" randomly.
     if not learning:
          set action = "DONE_SIGNAL"
     elif newstate received was Terminal (simulator sent "16"):
          set action = "RESTART_SIGNAL"
          set action = "NONE"
          set iterations = 0
          increment trials
          explorationchance = trials^-0.1 FOR PART 1
     elif iterations >= 15 (the time horizon for each trial)
          set action = "RESTART_SIGNAL"
          set action = "NONE"
          set iterations = 0
          increment trials
          explorationchance = trials^-0.1 FOR PART 1
     increment iterations
     commit to the action, compute information on result and reward
output Q values learned.  You may also automatically find policies, etc.
```

1. After implementing the above algorithm report the learned policy action for each non-terminal state. Calculate the state number of a state (x,y) by x+4y (e.g. the robot in position (1,1) would be in state number 5, using the notation from hw5). Also report the reward obtained by your agent in the last 10 trials it runs, and the average over all trials. Pick a random non-terminal starting location for each trial. A new trial also starts whenever `"RESTART_SIGNAL"` is sent.

2. Replace both `explorationchance` updates (marked with FOR PART 1 in the pseudocode) with a new update rule (comment out the part 1 rule for now, you'll want it later) which causes your agent to explore too often. Your update should be in the form of `explorationchance` $= trials^{-x.}$ Report the x you used. Report the learned policy action and its Q-value for each non-terminated state, and the rewards obtained in each of the last 10 trials and average reward. Consider these results and the value iteration policy below and justify why you believe it is using too much exploration and too little exploitation.

3. Now replace both `explorationchance` updates (comment out the part 1 and 2 rules so we can still see it in your submission) which causes your agent to exploit too often in the same format as part 2. Report the x you used. Report the learned policy action and its Q-value for each non-terminated state, and the rewards obtained in each of the last 10 trials and average reward. Consider these results and the value iteration policy below and justify why you believe it is using too much exploitation and too little exploration.