# CS 440 Practice Midterm Problems
# (with answers)
# Nov 2, 2015

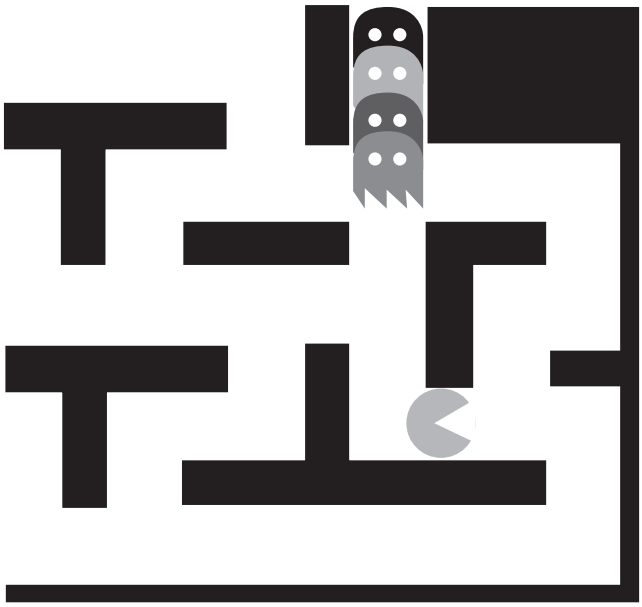
## Rules

- Write your answers directly on these pages.
- You may use a single sheet of paper with reference materials (two sides) that you have prepared.
- You may not discuss the test with anyone else or use any electronic devices (the problems involve minimal calculation).
- You may ask clarification questions during the exam. Raise your hand and we will come to you.
- The test has 7 problems. All count equally.
- Show your work for partial credit.

Your name: _____

*When you get the exam, you will see something like this. The problems that follow include more than you would see on an actual exam (at least twice as many), and have not been formatted to allow you to write the answers in place. These are real problems. They illustrate the style of the exam—questions that ask you to apply concepts and definitions from class to new problems and cases, and questions that ask you to work out the predictions of algorithms and models from class to specific new examples. This inventory of problems reflects the slightly different emphasis of the course in previous years, so you can expect fewer questions on stuff we didn't spend so much time on and maybe some new questions on stuff that we covered carefully that's not represented here.*

**1.** (Agents) Consider the enemy ghost agents in the video game pac man. The game is played on a grid. Some cells are occupied by walls, so that the board becomes a network of hallways and junctions. The ghosts move at a constant speed and change directions only at junctions. They implement a reactive architecture that chases the pac man player character.
The experience of the game depends on how the ghosts behave in situations like the one below:



Answer the following questions:

- Why is it important that the ghosts be able to separate themselves in this situation?

  If the ghosts all act the same way, then pac man can follow any path and the ghosts will just chase him together. This would eliminate any strategy aspect of the game.

- Explain how the flocking algorithm ensures that agents don't get too close together. Could we use something similar for pac man ghosts?

  In flocking, boids experience a separation force that pushes them continuously away from nearby boids. That doesn't work straightforwardly in the pac man world because it is played on a grid of tiles, so directional movement "away from" nearby ghosts would be undefined or unavailable.

- In fact, in pac man, the ghosts have different "personalities" which ensure that the ghosts have slightly different behavior when chasing pac man up close. One ghost aims for where pac man is, one aims a little bit ahead, one aims a little behind, and one moves off to loop around in a nearby corner. Explain why each of these constitutes a reactive strategy.

  > Each strategy just makes reference to the current state of the world, in terms of landmark positions on tiles in the board. There is no need to remember what has happened or predict what is going to happen.

- Explain why the combination of strategies makes interesting things happen in cases like the one shown above.

  > In this case, some of the ghosts will follow pac man, while others explore the corridors to the right and left. This makes it possible that they will surround pac man, and players must strategize whether they have time to eat all the pellets before their character is trapped, which is more challenging and perhaps even exciting.

**2.** (Agents) You're probably familiar with the children's game of rock–paper–scissors (many cultures have similar games). Two players play simultaneously. Each picks a shape: rock, paper, or scissors. If the players play the same shape, it is a tie. Otherwise, scissors beats paper, rock beats scissors, paper beats rock. One good strategy is to choose each shape randomly one third of the time. Then you can't lose to another good player. Unfortunately, you also can't defeat a weak player. On TV, Bart Simpson always plays rock, so Lisa plays paper and always wins. In fact, there are AI programs that defeat good human players much better than chance in fair play! These agents have *models* that predict what the opponent is likely to do.
How could an agent use one of these models to decide what it should play? Why would AI methodology be needed to build such a model? How might you do it? Be as specific as you can, and aim for something that would actually work.

> Suppose you have a model that predicts what your opponent is likely to play. You run the model to get a move $M$. In rock–paper–scissors, there is a unique move $B$ that beats $M$. So you play $B$. You need AI methodology to adapt the model to the ways people tend to play the game. You don't know this in advance as a programmer. One direction might be to play the game randomly for a while and get a data set of human play. You can then set up a supervised classification learning problem to predict the opponent's next move given play on recent rounds.

**3.** (Agent architecture) The next questions involve building a sophisticated *reactive agent* to play the classic 8-bit video game Frogger. You may recall that in this game you control a frog who must cross a highway and a river to reach a goal. Traffic moves on the highway in lanes; the frog dies if it gets run over. To cross the river, the frog must jump from support to support; for some reason the frog dies if it gets wet. The supports float in lanes too. They include logs as well as turtles that can sometimes decide to submerge.

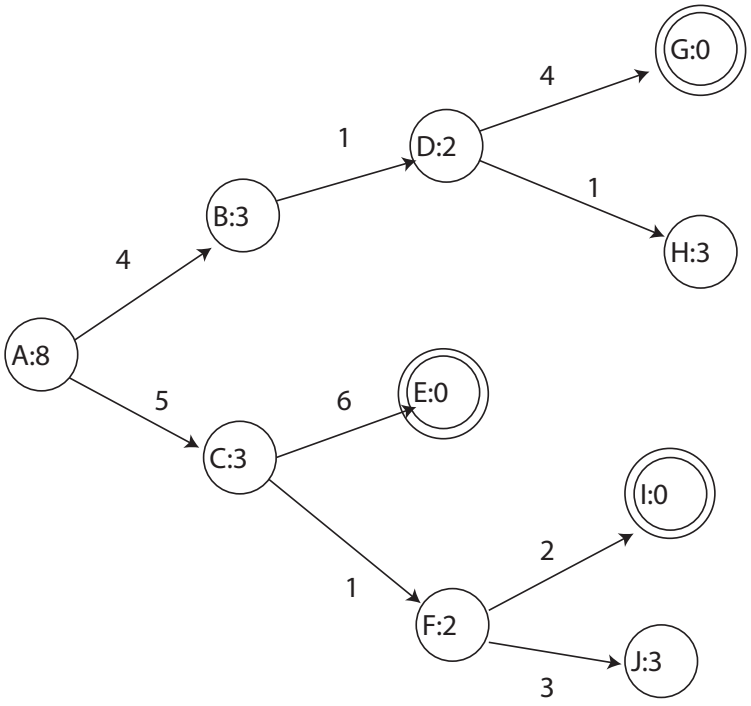
- At each step, the frog has the option to move up to the next lane (towards the goal), stay at its current lane, or retreat to the previous lane (away from the goal). The design calls for programming the decision using *affordances* that say what the frog can safely do in a lane. What are affordances in general and how would you implement them here?

  > Affordances are actions that it's possible for an agent to take with respect to objects in their environment. Modeling the current situation in terms of affordances means that the agent starts from a set of candidate actions that make sense in its situation. For frogger, you can define affordances as follows: a move in the highway is possible if it's clear of traffic; a move on the river is possible if it has a support.

- The design calls for you to implement two behaviors. One moves you gradually towards home—moving up if you can or waiting if you must. The other avoids imminent death—for example, getting off the turtles if they start to sink, however you can. You will use *the subsumption architecture* to combine these behaviors. Define the subsumption architecture and say how you would use it with these two behaviors.

  > The subsumption architecture consists of a set of ranked behaviors running in parallel. Each one suggests an action to perform now. The behaviors are ranked; the ranking determines what action to do when the behaviors conflict. In this case, the behaviors are to advance and to survive. Surviving can be implemented as a lower-level behavior that takes priority over advancing.

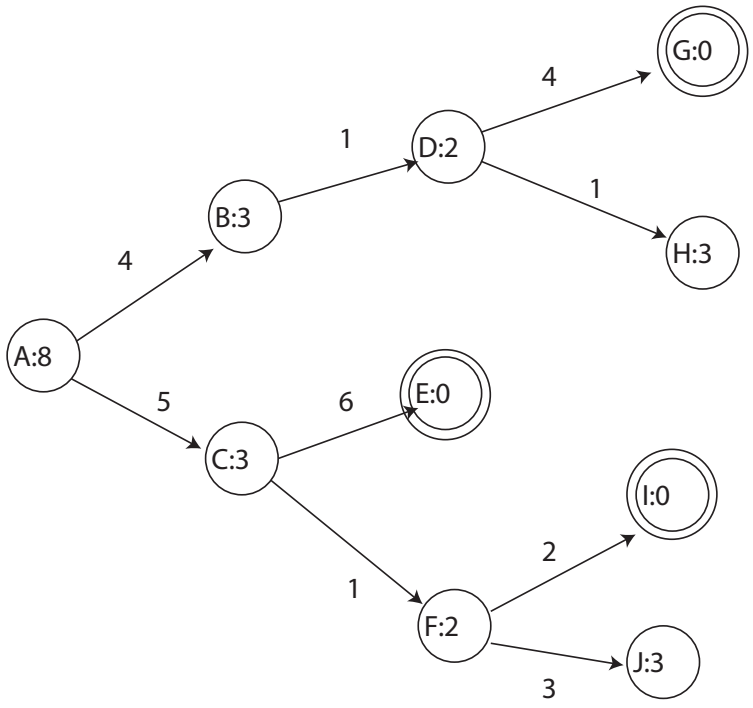
**4.** (Search) Consider the following search tree, where edges are labeled with path cost and states are labeled with heuristic value.




Start at state A. Double circle states are goals.
List the order in which states are explored by iterative deepening search (increasing the depth limit from 0 in increments of 1 at each iteration), up to and including the first goal state. Assume the algorithm breaks ties by considering states in alphabetical order.

| Iterative deepening search proceeds: A; A; B; C; A; B; D; C; E. |
|---|

**5.** (Search) Consider the same search tree repeated below:

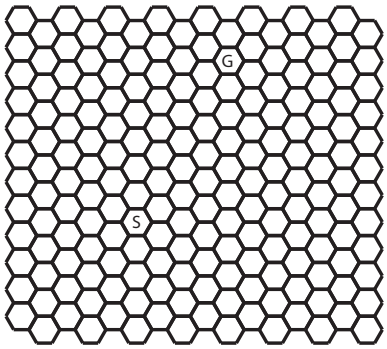



List the order in which states are explored by A* search, up to and including the first goal state. Assume the algorithm breaks ties by considering states in alphabetical order.

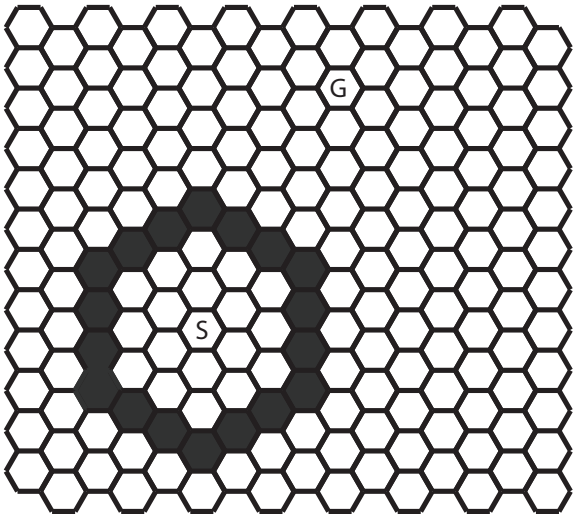
A* Search:

| Explored | Frontier |
|---|---|
| A | B:7, C:8 |
| B | D:7, C:8 |
| D | C:8, G:9, H:9 |
| C | F:8, G:9, H:9, E:11 |
| F | I:8, G:9, H:9, E:11, J:12 |
| I | G:9, H:9, E:11, J:12 |

**6.** (Search) Consider path planning in a world of hexagonal tiles, rather than square tiles, as shown below:



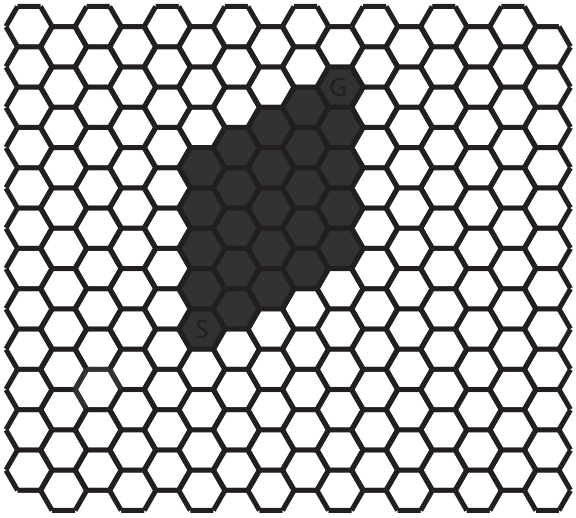

The agent can move from any hexagon to any adjacent hexagon in one step of time with one unit of effort. Suppose the agent is starting at position $S$ and needs to go to $G$. Shade below the frontier after three levels of breadth-first search:




In general, how many cells will be on the frontier after $d$ steps, for $d$ greater than 0?

| |
|---|
| $6d$ |

Shade below all the cells that lie on optimal paths from the start to the goal.



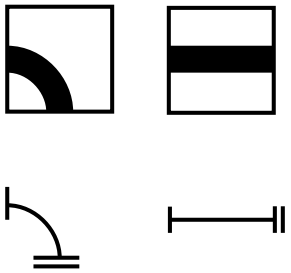

In the worst case, how many cells are there that lie on optimal paths between two points, assuming that path takes $d$ steps?

> Technically, using floor and ceiling functions, it's: $\left(\lceil \frac{d}{2} \rceil + 1\right)\left(\lfloor \frac{d}{2} \rfloor + 1\right)$. This is the area of the largest rectangle whose two sides add up to $d + 2$. But anything compatible with $O(d^2)$, such as roughly $\left(\frac{d}{2}\right)^2$, would count as correct.

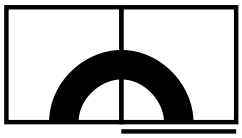
Suppose we have the choice of tiling a virtual world with squares or with hexagons. In either case, we will divide up the world so that there are $n$ cells total, so the area of each cell will be the same. Agents will be able to move from one cell across an edge to one of the four or six adjacent cells at each step (this is the motion we just considered and the one we usually discussed in class). We will use A* to plan paths. In which world will path planning be more efficient? Justify your answer in terms of the search space, including considerations such as the branching factor, the complexity of solution paths and the number of cells explored.

> The complexity of A* depends on the number of cells it needs to visit—if there's a small detour then A* has to visit a big fraction of the squares on optimal trajectories, worst case, to see that they all lead to dead ends. That's the same number, about $\frac{d}{2}^2$, for both rectangular grids and hex grids, where $d$ is the length of the path. But in hex worlds paths are shorter on average; the $\frac{d}{2}^2$ area in a hex world is one sixth of the available space, where it's just a quarter in grid worlds. So hex search has less to explore.

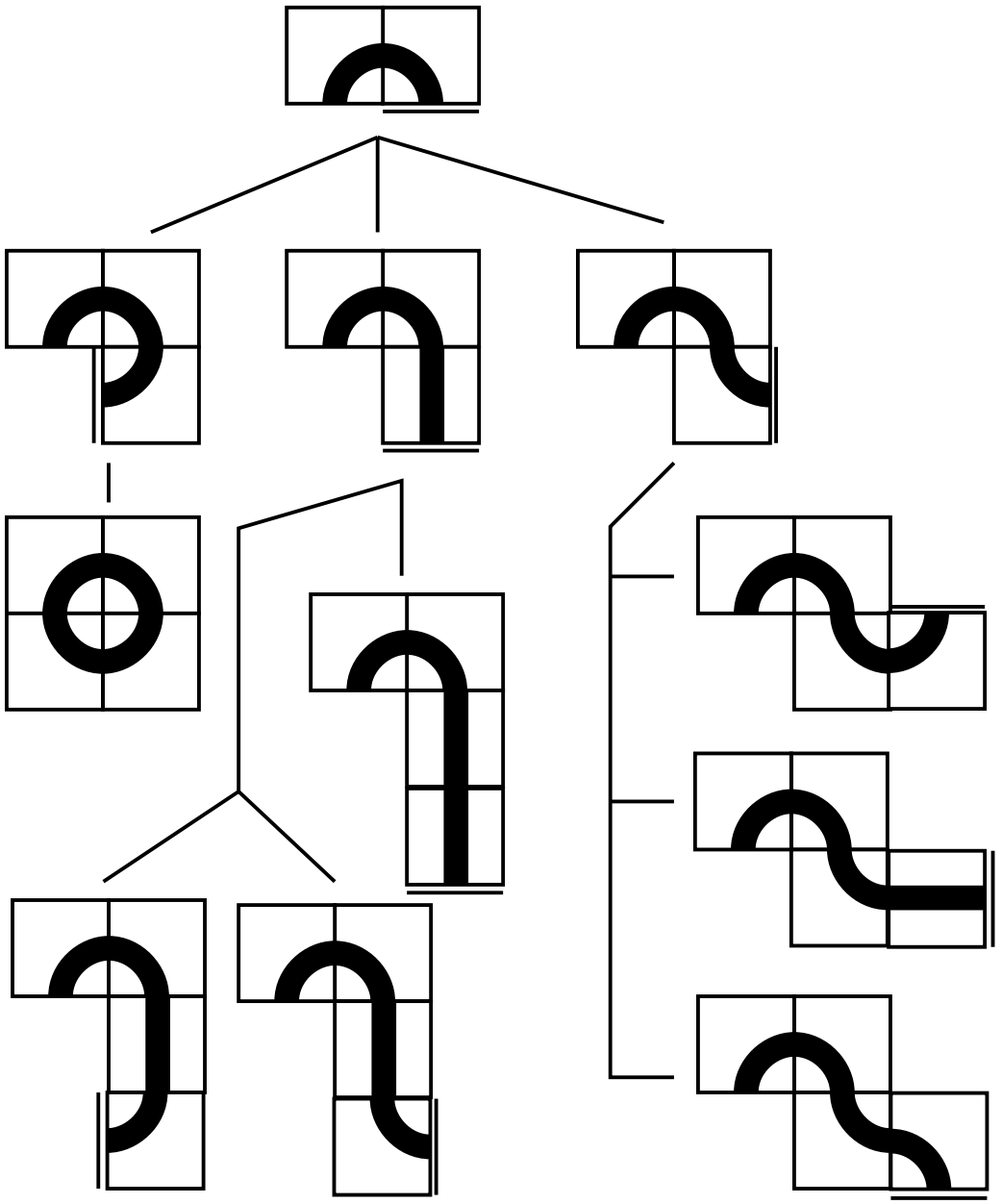
**7.** (Search) Imagine assembling a model railroad out of segments. Segments can be straight or they can curve 90 degrees (left or right). Each takes up a square, as below. You have an unlimited number of all kinds. Think of the segments as being solid elements but you can draw them more compactly using lines and hatches to abbreviate the pieces as illustrated.



Assume that you always have a single active edge where you need to continue assembly. This can be marked by an extra line in the figures as shown. When you connect a piece, it must line up at both ends—the track must continue into any square occupied by an adjacent segment; the track can never dead-end into an occupied square.
Suppose you start with two curved blocks as shown:



On a blank page, draw the complete search space for the next two pieces you can add.
*(In an actual exam the following page would be left blank.*



Is the search space a tree or a general graph?

The search space is a tree. Because you can only add along a single edge there is only one way to compose any railroad.

How many neighbors does each state have at most?

Each state has at most three neighbors.

**8.** (Search) Verbal arithmetic is defined as follows in Wikipedia:

> Verbal arithmetic, also known as alphametics, cryptarithmetic, crypt-arithmetic, cryptarithm or word addition, is a type of mathematical game consisting of a mathematical equation among unknown numbers, whose digits are represented by letters. The goal is to identify the value of each letter. The name can be extended to puzzles that use non-alphabetic symbols instead of letters.
>
> The equation is typically a basic operation of arithmetic, such as addition, multiplication, or division. The classic example, published in the July 1924 issue of Strand Magazine by Henry Dudeney, is:

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

> The solution to this puzzle is O = 0, M = 1, Y = 2, E = 5, N = 6, D = 7, R = 8, and S = 9.
>
> Traditionally, each letter should represent a different digit, and (as in ordinary arithmetic notation) the leading digit of a multi-digit number must not be zero. A good puzzle should have a unique solution, and the letters should make up a phrase (as in the example above).

(a) Formulate solving a verbal arithmetic problem as a search problem. Briefly, what are the states, what are the actions, what are the transitions, what is the initial state and what is a goal? (Your answer can be short English sentences; mathematical definitions are not required.)

A state is a partial assignment of unique digit values to a subset of the letters in the problem. The action is to assign a digit value to the next letter. Transitions must make sure that the digit values are consistent with the equation and can fill in any assignments that are required mathematically given the constraints so far. In the initial state, no letters are assigned values. A goal is a consistent assignment of digits to all of the letters in the problem.

(b) Choose a search algorithm to use on this search problem and explain your choice. Things to consider include the size of paths, whether the search space is a tree or a general graph and whether you can define a good heuristic function.

This is a good place for depth first search; all the solutions are the same size and you can pick a unique variable to choose next in order to make the search space a tree. Much as in satisfiability, you can improve the search by doing inference to propagate the constraints that are entailed by the other chocies made so far.

**9.** (Search) Consider the famous one dimensional bin-packing problem.

> Given a finite collection of $n$ weights $w_1$, $w_2$, $w_3$, ... , $w_n$, and a collection of identical bins with capacity $C$ (which exceeds the largest of the weights), what is the minimum number $k$ of bins into which the weights can be placed without exceeding the bin capacity $C$?

See more about this problem at ams.
One way to solve it is by using search.

(a) Formulate solving a bin packing problem as a search problem. Briefly, what are the states, what are the actions, what are the transitions, what is the initial state and what is a goal? What is the cost function? (Your answer can be short English sentences; mathematical definitions are not required.)

A state involves a list of bins used so far, together with the weight assigned to them, plus a list of weights that still need to be placed. The actions involve picking a weight to place, and placing it in either one of the existing bins or into a new bin; the corresponding transition updates bins and weights accordingly. The initial state has no bins and all the weights unplaced. A goal state has all the weights in bins. The cost function is the number of bins (times the capacity $C$).

(b) Choose a search algorithm to use on this search problem and explain your choice. Things to consider include the size of paths, whether the search space is a tree or a general graph and whether you can define a good heuristic function.

This is a good problem to solve using $A^*$. You can define a number of good heuristic functions to estimate the work to be done. A simple one is to use the weight of the unplaced weights minus the remaining capacity of the partially-filled bins. You can improve this by distinguishing placeable weights from unplaceable weights. Unplaceable weights don't fit in any of the available bins. A better heuristic is then the total of the unplaceable weights, plus the weight of the placeable weights minus the remaining capacity of the partially-filled bins. All the solutions have the same length, however, and if you put an order on the weights you get a tree search space, so depth-first search is also reasonable; you would probably want to use iterative deepening search and gradually increase the number of allowable bins.

**10.** (Learning) A robot vacuum has a naive Bayes model for inferring whether animals in its environment are dogs ($D$) or cats ($C$). (Those are the only options, so $C = \neg D$.) The robot has access to three perceptual features: if the animal looks bigger than 18 inches tall ($B$, for big), if the animal is making noise ($N$), and if the animal is running away ($R$). The needed parameters of the model are exhibited below:

$$P(D)$$
$$P(B|D) \;\; P(N|D) \;\; P(R|D)$$
$$P(B|\neg D) \;\; P(N|\neg D) \;\; P(R|\neg D)$$

**(a).** Give a general expression for how the model makes its prediction about what a big animal, making noise and running away is ($BNR$).

The model calculates $P(BNRD) = P(D)P(B|D)P(N|D)P(R|D)$ and $P(BNR\neg D) = P(\neg D)P(B|\neg D)P(N|\neg D)P(R|\neg D)$. It predicts $D$ if $P(BNRD)$ is bigger and predicts $\neg D$ otherwise.

**(b).** Suppose the actual model parameters are these:

$$P(D) = \tfrac{1}{3}$$
$$P(B|D) = \tfrac{2}{3} \;\; P(N|D) = \tfrac{1}{2} \;\; P(R|D) = \tfrac{1}{10}$$
$$P(B|\neg D) = \tfrac{1}{10} \;\; P(N|\neg D) = \tfrac{1}{8} \;\; P(R|\neg D) = \tfrac{2}{3}$$

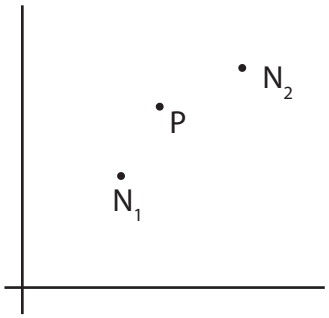
What decision does the model actually make for a $BNR$ animal?

The model calculates $P(BNRD) = P(D)P(B|D)P(N|D)P(R|D) = \frac{1}{90}$ and $P(BNR\neg D) = P(\neg D)P(B|\neg D)P(N|\neg D)P(R|\neg D) = \frac{1}{240}$. So it thinks this is a dog.

**(c).** Suppose this model has been learned from a collection of training data. Given the parameters we learned, is it possible that the training data contained no examples of $BNR$ cats? Is it possible it contained no $BNR$ dogs? Might it have had no $BNR$ animals at all? Explain your answer.

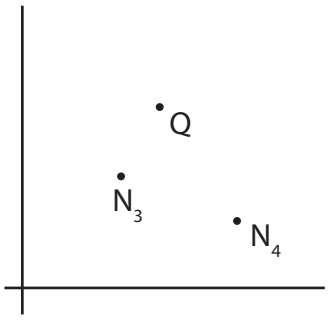
The model generalizes across cases. It's quite likely that the training data included $B$ dogs, $N$ dogs and $R$ dogs; and $B$ cats, $N$ cats and $R$ cats. This is a certainty if the model was trained using maximum likelihood estimation. However, the features do not have to have occurred in this combination. There could have been no $BNR$ animals at all (especially if the indepedence assumptions of the model are not met in the training data).

**11.** (Learning, 12 points) You are doing regression—predicting $y$ as a function of $x$—using a nearest neighbor technique.
**(a).** Graph a test point $P$ and two neighbors $N_1$ and $N_2$ in such a way that $P$ is predicted *better* using both $N_1$ and $N_2$ than just using its closest neighbor.




A prediction from $N_1$ and $N_2$ is between $N_1$ and $N_2$ and therefore closer to $P$ than either.

**(b).** Graph a test point $Q$ and two neighbors $N_3$ and $N_4$ in such a way that $P$ is predicted *worse* using both $N_3$ and $N_4$ than just using its closest neighbor.

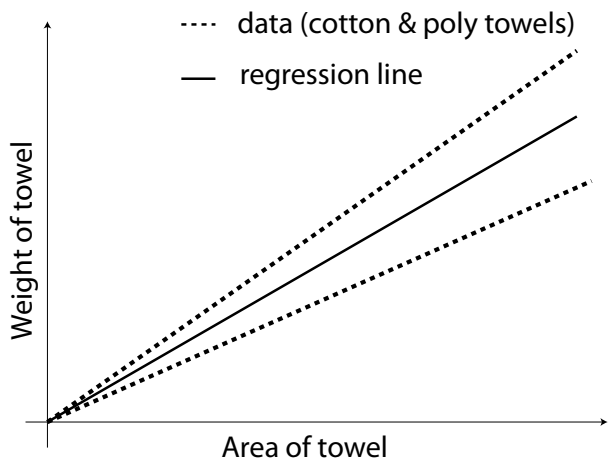



The prediction of $Q$ becomes worse when we include a contribution from $N_4$ as well as the contribution from $N_3$.

**12.** (Learning) A laundry-folding robot is trying to predict the weight of a towel from the area of the towel, which the robot can estimate visually. Suppose there is a linear relationship between the two variables. However, unbeknownst to the robot, there are two kinds of fabric used in the towels, cotton and polyester. Half the towels are cotton and half the towels are polyester. The fabric used is independent of the size of the towel. Cotton towels weigh 50% more than polyester. The robot cannot see the difference between the fabrics.

- Suppose the robot collects a large amount of data measuring weight as a function of area. Draw a schematic graph which shows how this data will be distributed.
- Suppose the robot learns to predict weight from area by fitting a linear function. Graph the prediction function that it will get.

The answers for the two problems are both to be found on the graph below, which shows both the data (dotted lines) and the prediction function (solid line).




- Suppose the robot predicts weight from area by a nearest-neighbor rule—the predicted weight of a sample towel will be the weight of the towel in the training data with the closest area to the measured area of the sample. What will this rule do in the limit of a large set of training data?

Nearest neighbor predicts at random that the new towel either has the weight of the cotton towel of that size or the weight of the polyester towel.

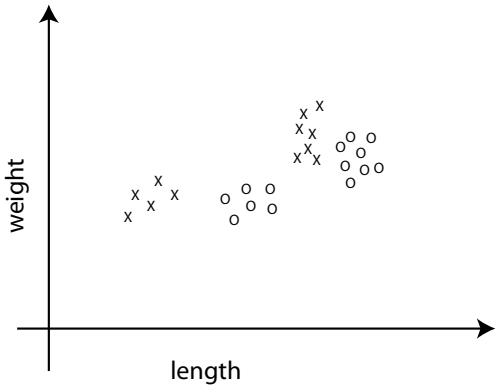
- Compare the error of the nearest-neighbor and regression methods. How will the average error of the two methods compare? How will the root mean square error of the two methods compare?

The nearest neighbor method is exactly right half the time and off by the difference between cotton and poly—call it $d$—the rest of the time. The regression answer is off by half of that difference—call it $d/2$—all the time. That means NN and regression have the same average error (related to $d/2$: NN gets 0 error half the time, and $d$ error half the time, for an average of $d/2$; regression just gets $d/2$ all the time.) However the average squared error of NN is a function of $d^2/2$ (all the error half the time) while the average squared error of regression is a function of $d^2/4$ (half the error all the time). So RMS error of NN is a factor of $\sqrt{2}$ times the RMS error of regression.

**13.** (Learning) You are doing regression—predicting $y$ as a function of $x$—using a nearest neighbor technique. You build an initial version of the system and you find many examples in test data where, like $Q$, performance is worse considering two neighbors than considering just the closest one. A coworker suggests that if you get enough data, you will eliminate these cases and predicting from two neighbors will work better on all points. Is this necessarily true? Explain.

Your coworker is wrong. It's possible that your data includes two different classes of items, and you cannot measure the difference between them. In one case $y$ is much bigger as a function of $x$ than it is in the other. Then no matter how much data you get, you always have the possibility that the nearest two neighbors are the same category as the test point, one is the same category, or neither is. If the nearest neighbor is the right category but the next nearest neighbor is the wrong one, then nearest neighbor will give the right answer for the test point and the two nearest neighbors will make a worse prediction for it, no matter how much data you have. You can't expect predicting from two neighbors to work better on all test points.

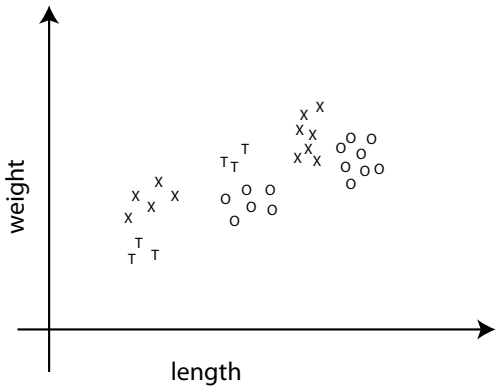
**14.** (Learning) Two naturalists, $A$ and $B$, build underwater robots to classify whales from two species, $X$ and $O$. $A$ and $B$ send their robots into the North Atlantic in October and get the training data plotted as the $X$'s and $O$'s on the graph below.




(a) $A$ uses nearest neighbor to build a classifier based on the training data. $B$ uses a model that gives a linear decision boundary between measurements classified as X's and those classified as O's.
Qualitatively speaking, how well do the two models work to describe *the training data*?

> To assess this, imagine training the classifier with all but one of the points from the training data, and testing the remaining point. Both classifiers will perform the task with high accuracy; nearest neighbor in fact seems likely to get them all right, while the linear decision boundary may misclassify a handful of outliers.

(b) It turns out that species $X$ and species $O$ breed at different times of year. The robots go back to the ocean in July and collect test data. The old baby whales from $X$ are a little grown up, and there are new baby $O$ whales. The new data is shown as $T$'s on the graph below, superimposed on the training data.




With this information, if you read the graph you can recognize for yourself the true class of each of the test items illustrated by $T$'s on the graph.
How do $A$ and $B$'s classifiers handle this test data?

| |
|---|
| The linear classifier classifies the new test data correctly, since the test data has the same linear relationship as the training data. The nearest neighbor classifier now classifies the adolescent $X$'s from the new data with the adolescent $O$'s from the training data. |