Notes and Solutions for: Artifical Intelligence: A Modern Approach: by Stuart Russell and Peter Norvig.

John L. Weatherwax[*]

Nov 10, 2001

[*]wax@alum.mit.edu

Chapter 3 (Solving Problems by Searching)

Problem Solutions

Formulating Various Search Problems

Part (d): In this case we could take our state to be how much water is in each jug and our successor function would simulate filling jugs with water, pouring water from one jug to another, and pouring water out of a jug onto the ground. I have implemented some code that solve problems of this type in the python code Pouring_Problems.py. When that code is run it will solve the given problem instance and gives the following output

```
state=  (0, 0, 0);  from sink pour  12 into   0 to get state= (12, 0, 0)
state= (12, 0, 0);  from    0 pour   8 into   1 to get state=  (4, 8, 0)
state=  (4, 8, 0);  from    0 pour   3 into   2 to get state=  (1, 8, 3)
```

In this final state our first jug has one gallon of water in it (and satisfies our goal).

The missionaries and the cannibals

Part (a): The state space for this problem consists of a description of each side of the river where the description consists of the number of missionaries, the number of cannibals, and whether or not there is a boat on that side of the river. To solve this using the programming language python, I'll represent the state as a simple dictionary of dictionaries like the following (shown for the initial state)

```
state = { "LSide": { "nMiss": 3, "nCann": 3, "nBoat": 1 },
          "RSide": { "nMiss": 0, "nCann": 0, "nBoat": 0 } }
```

The "successor" function will have to check if a given action would result in the number of cannibals outnumbering the number of missionaries. If an action were to produce a state where that is true we cannot take that action.

A general graph search algorithm for this problem was coded in the python code missionaries_N_cannibals.py. When that code is run it comes up with the following solution to this problem (in a simplified notation):

```
Initial state:  {L: {nC=3, nM=3, nB=1}, R: {nC=0, nM=0, nB=0}}
Action                                  Resulting State
2 Cannibals LSide to RSide              {L: {nC=1, nB=0, nM=3}, R: {nC=2, nB=1, nM=0}}
1 Cannibal RSide to LSide               {L: {nC=2, nB=1, nM=3}, R: {nC=1, nB=0, nM=0}}
```

```
2 Cannibals LSide to RSide                {L: {nC=0, nB=0, nM=3}, R: {nC=3, nB=1, nM=0}}
1 Cannibal RSide to LSide                 {L: {nC=1, nB=1, nM=3}, R: {nC=2, nB=0, nM=0}}
2 Missionaries LSide to RSide             {L: {nC=1, nB=0, nM=1}, R: {nC=2, nB=1, nM=2}}
1 Cannibal, 1 Missionary RSide to LSide  {L: {nC=2, nB=1, nM=2}, R: {nC=1, nB=0, nM=1}}
2 Missionaries LSide to RSide             {L: {nC=2, nB=0, nM=0}, R: {nC=1, nB=1, nM=3}}
1 Cannibal RSide to LSide                 {L: {nC=3, nB=1, nM=0}, R: {nC=0, nB=0, nM=3}}
2 Cannibals LSide to RSide                {L: {nC=1, nB=0, nM=0}, R: {nC=2, nB=1, nM=3}}
1 Missionary RSide to LSide               {L: {nC=1, nB=1, nM=1}, R: {nC=2, nB=0, nM=2}}
1 Cannibal, 1 Missionary LSide to RSide  {L: {nC=0, nB=0, nM=0}, R: {nC=3, nB=1, nM=3}}
```

Solving the 8 puzzle

Under the conditions where all search nodes are generated at the same time (rather than generated on an "as needed" basis) an implementation of iterative deepening depth-first search can be found in the `python` code `eight_puzzle.py`. To help facilitate testing of this routine (and to not try and search for a solution to an unsolvable problem) I generate a random starting board by performing some number of random steps from the goal state. An example of the solution this routine could produce (stating with a board generated from 30 random steps from the goal state) I'm getting

```
Initial state:  [0, 2, 4, 1, 8, 5, 3, 6, 7]
Action                                 Resulting State
right                      [2, 0, 4, 1, 8, 5, 3, 6, 7]
right                      [2, 4, 0, 1, 8, 5, 3, 6, 7]
down                       [2, 4, 5, 1, 8, 0, 3, 6, 7]
left                       [2, 4, 5, 1, 0, 8, 3, 6, 7]
up                         [2, 0, 5, 1, 4, 8, 3, 6, 7]
left                       [0, 2, 5, 1, 4, 8, 3, 6, 7]
down                       [1, 2, 5, 0, 4, 8, 3, 6, 7]
down                       [1, 2, 5, 3, 4, 8, 0, 6, 7]
right                      [1, 2, 5, 3, 4, 8, 6, 0, 7]
right                      [1, 2, 5, 3, 4, 8, 6, 7, 0]
up                         [1, 2, 5, 3, 4, 0, 6, 7, 8]
up                         [1, 2, 0, 3, 4, 5, 6, 7, 8]
left                       [1, 0, 2, 3, 4, 5, 6, 7, 8]
left                       [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

The array notation above is a short hand for more standard "board" notion. For example, the initial state above (where 0 represents the blank space) has the equivalence

```
[1, 4, 2, 3, 0, 5, 6, 7, 8] is the same as the "board" array([[1, 4, 2],
                                                                [3, 0, 5],
                                                                [6, 7, 8]])
```

Perhaps it is due to keeping all of these search nodes in memory but it seems that this algorithm is only able to solve (in a reasonable amount of time) relatively small 8 puzzle problems. I did not have time to implement the more memory efficient 8 puzzle solver and compare its performance.

Chapter 4 (Beyond Classical Search)

Notes on the Text

Notes on effective heuristic accuracy on performance

$$52 = 1 + b^* + b^{*2} + \cdots b^{*d-1} + b^{*d} = \frac{b^{*d+1} - 1}{b^* - 1} \,.$$

with $d = 5$ we can solve for b^* and get $b^* = 1.92$.

Chapter 14 (Probabilistic Reasoning)

Notes on the Text

Notes on efficient representation of conditional distributions

In this section we are given an example of a noisy-OR relationship between the variables $Cold$, Flu, and $Malaria$. We expect that each of these variables (by itself) will influence the presence of the symptom (here $fever$). Namely we could expect to somewhat easily find/compute numbers like

$$
\begin{aligned}
P(fever|cold, \neg flu, \neg malaria) &= 0.4 \\
P(fever|\neg cold, flu, \neg malaria) &= 0.8 \\
P(fever|\neg cold, \neg flu, malaria) &= 0.9\,.
\end{aligned}
$$

These give the probability that we have a fever given only one of the possible causes of a fever. Thus if we have malaria (and nothing else) it is very likely that we will have a fever, while if we have a cold (and nothing else) it is less likely. The book presents the probability complement of these relationships i.e. values for $P(\neg fever|cold, \neg flu, \neg malaria) = 0.6$, but the two are equivalent. In the noisy-OR model, rather than require a new parameter in the cases where more than one cause is true we compute the probabilities of these cases from the above inputs. For example, if rather than numbers we have taken parameters as

$$
\begin{aligned}
P(\neg fever|cold, \neg flu, \neg malaria) &= \theta_c \\
P(\neg fever|\neg cold, flu, \neg malaria) &= \theta_f \\
P(\neg fever|\neg cold, \neg flu, malaria) &= \theta_m\,,
\end{aligned}
$$

then using these as inputs we compute the probabilities of having a fever when we have multiple symptoms

$$
\begin{aligned}
P(\neg fever|\neg cold, flu, malaria) &= P(\neg fever|\neg cold, flu, \neg malaria) \\
&\quad \times P(\neg fever|\neg cold, \neg flu, malaria) = \theta_f\theta_m \\
P(\neg fever|cold, \neg flu, malaria) &= P(\neg fever|cold, \neg flu, \neg malaria) \\
&\quad \times P(\neg fever|\neg cold, \neg flu, malaria) = \theta_c\theta_m \\
P(\neg fever|cold, flu, \neg malaria) &= P(\neg fever|\neg cold, flu, \neg malaria) \\
&\quad \times P(\neg fever|cold, \neg flu, \neg malaria) = \theta_f\theta_c \quad \text{and finally} \\
P(\neg fever|cold, flu, malaria) &= P(\neg fever|cold, \neg flu, \neg malaria) \\
&\quad \times P(\neg fever|\neg cold, flu, \neg malaria) \\
&\quad \times P(\neg fever|\neg cold, \neg flu, malaria) = \theta_c\theta_f\theta_m\,.
\end{aligned}
$$

These are how the entries in the table from this section are constructed.

Note: Below here these notes have not been proofed yet.

Notes on completeness of the node ordering

Figure 14.3 (b) to specify $P(M)$ and $P(J)$ we need two numbers. To specify $P(E|M,J)$ requires 4 numbers. To specify $P(B|M,E,J)$ requires $2^3 = 8$ numbers. To specify $P(A|B,M,E,J)$ requires $2^4 = 16$ numbers this gives a total of 30 numbers. Why is this not equal to 31?

Notes on Figure 14.6

$$\begin{aligned} P(c|h) &= p(c|h,s)p(s|h) + p(c|h,\cancel{s})p(\cancel{s}|h) \\ &= p(c|h,s)p(s) + p(c|h,\cancel{s})p(\cancel{s})\,. \end{aligned}$$

Since by assumption we assume that subsidy is independent to harvest. If we take $p(s) = p(/\ s) = 0.5$.

Notes on the Variable Elimination Algorithm

We have

$$\begin{aligned} P(\text{John Call}|\text{Burglary} = \text{True}) &= \alpha P(\text{John Call}|\text{Burglary} = \text{True}) \\ &= \alpha \sum_e \sum_a \sum_m P(J,b,e,a,m) \\ &= \alpha \sum_e \sum_a \sum_m P(b)P(e)P(e|b,e)P(J|a)P(m|a) \\ &= \alpha P(j) \sum_e P(e) \sum_a P(a|b,e)P(J|a) \sum_m P(m|a)\,, \end{aligned}$$

Notes on Approximate Inference in Bayesian Networks

Sample from each variable cloudy, sprinkler, rain, wet grass with the variable specification of $[true, false, true, true]$ we have

$$\begin{aligned} S_{PS}(true,false,true,true) &= P(cloudy = true)P(sprinkle = false|cloudy = true) \\ &\times P(Rain = true|Cloudy = true) \\ &\times P(wetgrass = true|Sprinkle = false, Cloudy = true) \\ &= 0.5(0.9)(0.8)(0.9) = \end{aligned}$$

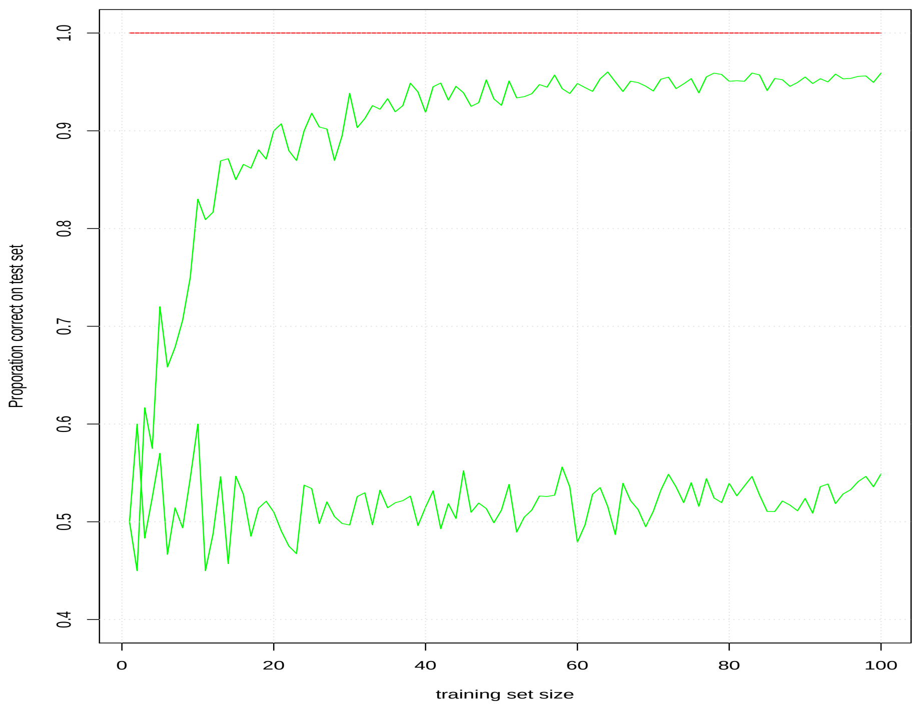


Figure 1: Duplication of the books figure 18.7 using the `R` code `dup_fig_18_7.R`. The red curve (a straight line at 1) is the classification accuracy of the build decision tree on the training data. The top most green curve is the classification accuracy of a decision tree on a new test set of data. The bottom curve is the classification accuracy on the test data using "table lookup" discussed in the problems.

Chapter 18 (Learning From Observations)

Notes on the Text

Notes on Section 18.3 (learning decision trees)

See the `R` code `restaurant_problem_gen_data.R` where we randomly generate input data for the restaurant problem and then use the tree given in the books figure 18.2 to decide what values to assign to the output *WillWait*. Next we implement the books function `DECISION-TREE-LEARNING` in the `R` code `decision_tree_learning.R`. Then in the `R` code `dup_fig_18_7.R` we generate training and testing data for various training set sizes and duplicate the learning curve given in the books figure 18.7. Running the `R` code gives rise to the Figure 1. This result matches the books result quite closely.

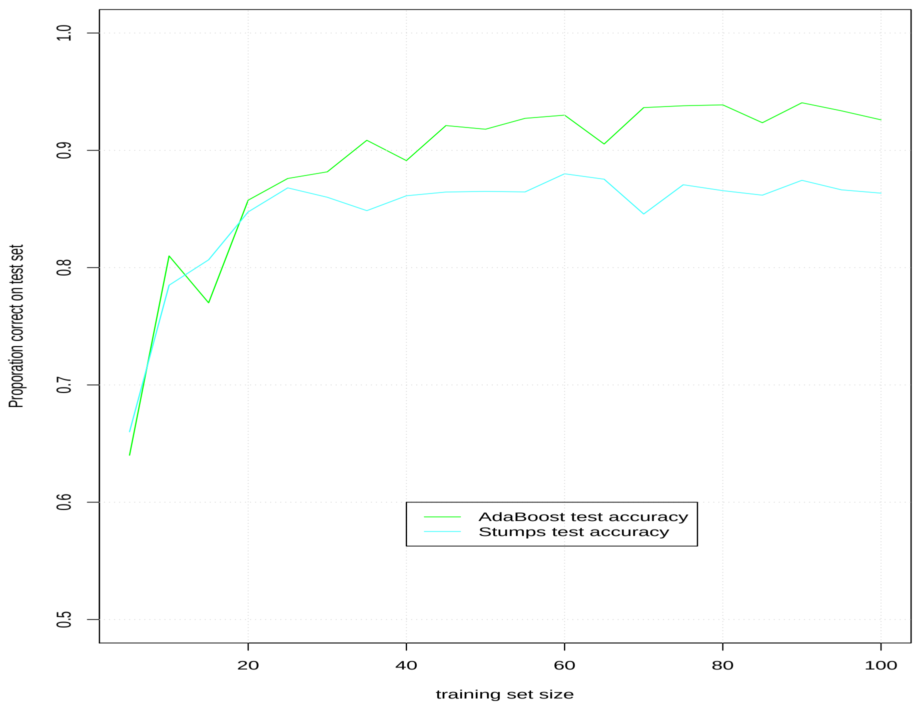


Figure 2: Duplication of the books figure 18.11 using the `R` code `dup_fig_18_11.R`.

Notes on Section 18.4 (ensemble learning)

See the `R` code `dup_fig_18_11.R` where we use the `R` code `adaboost_w_decision_trees.R` to duplicate the learning curve given in the books figure 18.11. Running the `R` code gives rise to the Figure 2. This result matches the books result quite closely.

Problem Solutions

Problem 18.4 (testing the same attribute twice)

For the decision trees presented in this chapter, we build them in a greedy manner looking for the attribute that when we split on it will result in the largest information gain. Once that attribute has been found we split the training set based on that attributes possible values and consider each resulting sub set of the data in a recursive manner, ignoring values of the attribute we split on. The reason we ignore the attribute we split on is that, once top level data is split using a specific attribute value then each resulting subgroup of data has the *same* attribute value and thus there is no variation in that attribute in which to facilitate further splitting. If a decision tree is built using continuous variables as input and the split points are restricted to be Boolean i.e. $X < 0$ then we may want to revisit this attribute at further points down the tree since not every sample will have the same value.

Problem 18.5 (returning the correct tree)

If the data generation process is without noise I think the answer is yes. In other words, if the data generation process produces data that is generated from a tree the learned tree will eventually match that from the data generation process (logically) as the number of samples goes to infinity. The learned tree may not be exactly the same tree that generated the data but will be logically equivalent (i.e. provide the same labeling to examples). If the process contains noise then again under the infinite data case the resulting tree should be logically equivalent to the data generation process. Practically decision trees are very sensitive to noise and thus the number of samples one needs to have a good match can will be larger in the "with noise" case vs. the "without noise" case.

Problem 18.6 (table look-up learning)

The suggested algorithm really just "memorizes" the training data and when presented with an input it has seen before returns that inputs response. If a provided input happens to have more than one output the classifier returns the most likely response. This algorithm is implemented in the `R` code `table_lookup_learning.R`. To compare this algorithm with others in the `R` code `dup_fig_18_7.R` we present learning curves of `table_lookup_learning.R` compared with that of `decision_tree_learning.R`.

Problem 18.7 (zero leave-one-out cross validation?)

For the perfectly paired samples (25 in each of class A and B) when we leave one sample out (say from class A) we will have 24 samples remaining from class A and 25 samples from class B. The majority classifier in this case will return B which is incorrect since the sample held out was from A. Thus the classifier is wrong for every sample we perform this process to (giving zero classification accuracy).

Problem 18.8 (error criterion)

Let t_i be the "target" for the ith training sample have the value 0 if the output should be "false" and 1 if the output should be "true". Let $\hat{t}_i$ be the assigned label from the classifier.

Part (a): In this case our metric for performance is the absolute error or

$$E \equiv \sum_{i=1}^{n+p} |\hat{t}_i - t_i| \,,$$

where we have n negative and p positive examples. We assume that we will pick the same value $\hat{t}$ (i.e. it is independent of i) for all samples in this node and the above becomes

$$\sum_{\text{negative examples}} |\hat{t}| + \sum_{\text{positive examples}} |\hat{t} - 1| = n|\hat{t}| + p|\hat{t} - 1| \, .$$

The expression on the right-hand-side is the sum of two piecewise linear functions and is thus a piecewise linear function. When $\hat{t} \to \pm\infty$ its value goes to $+\infty$ and its smallest value will be at the "corners". This means when $\hat{t} = 0$ or when $\hat{t} = 1$. If $\hat{t} = 0$, classify everything as false, we get the value of p, the number of positive examples. When $\hat{t} = 1$, classify everything as true, we get n, the number of negative examples. Thus to make this expression as small as possible we take $\hat{t} = 0$ if $p < n$ and take $\hat{t} = 1$ if $p > n$. This is the same as picking the majority classification.

Part (b): In this case our metric for performance is the squared error or

$$E \equiv \sum_{i=1}^{n+p} (\hat{t}_i - t_i)^2 = n\hat{t}^2 + p(\hat{t} - 1)^2 \, .$$

It is now possible that the minimum value of the above happens to be between the values of $[0, 1]$. To find possible minimums we take the first derivative of the above expression with respect to $\hat{t}$, set the result equal to zero, and solve for $\hat{t}$ to get

$$2n\hat{t} + 2p(\hat{t} - 1) = 0 \quad \text{so} \quad \hat{t} = \frac{p}{n+p} \, .$$

The second derivative of the above is given by $2n + 2p > 0$ showing that the above expression is a minimum. Evaluating the above at $\hat{t} = \frac{p}{n+p}$ we get $\frac{np}{n+p}$. Since our minimum location is inside the domain $[0, 1]$ we don't need to check to see if the end points of the domain have a smaller objective value.

Problem 18.11 (implementing χ^2-pruning)

I'll assume in working this problem that we will apply χ^2-pruning as the tree is built. One can imagine running a pruning routine *after* a tree has been overfit and using it to remove unnecessary splits. This procedure works as follows when we are considering next splitting on the attribute A that has the largest information gain before we actually make the split we will perform a statistical hypothesis test to determine if indeed this split is significant. To do this we perform the following steps

- Compute for each of the v child nodes $i = 1, 2, \cdots, v-1, v$ the values

$$\hat{p}_i = (p_i + n_i)\left(\frac{p}{p+n}\right)$$
$$\hat{n}_i = (p_i + n_i)\left(\frac{n}{p+n}\right) \, .$$

  Here $\frac{p}{p+n}$ is the fraction of positive examples in the parent node and thus $\hat{p}_i$ as defined above is the expected number of positive examples in the ith node that has $p_i + n_i$ samples flowing there. As similar statement can be made for the expression $\hat{n}_i$.

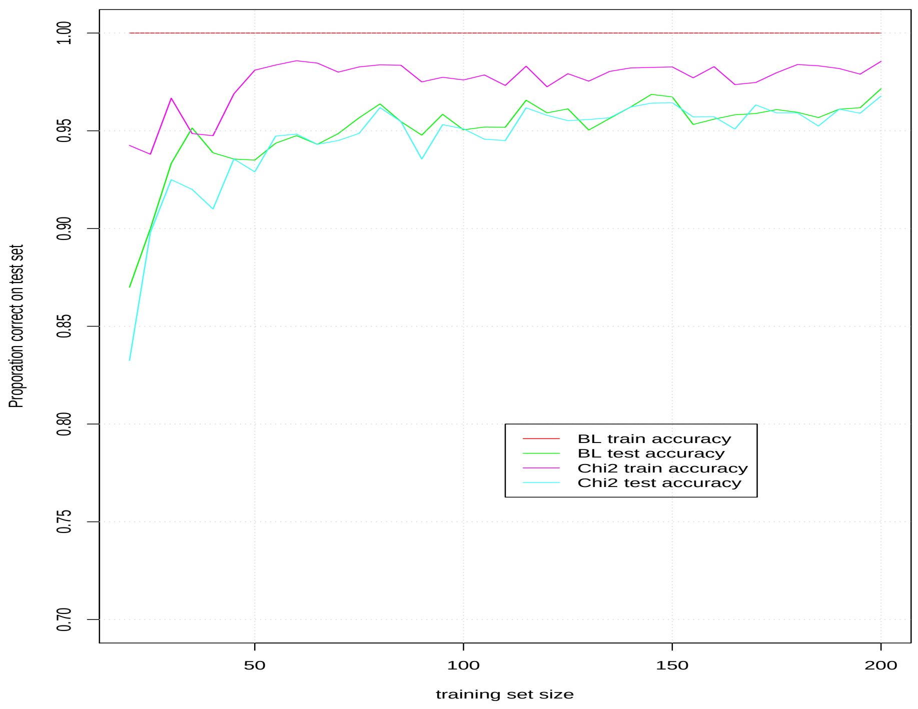


Figure 3: Learning curves for tree learning with χ^2 pruning.

- Next compute the value
$$D = \sum_{i=1}^{v} \left(\frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} \right) .$$
We expect D to be large if this split is significant since then $p_i \neq \hat{p}_i$.
- Compute the value of $c_{\alpha,v-1}$ or the $(1-\alpha)\%$ critical value of a χ^2 distribution with $v-1$ degrees of freedom. If $D < c_{\alpha,v-1}$ we conclude that we cannot reject the null hypothesis and *don't* perform the indicated splitting on this parent node.

This procedure is implemented in the `R` code `decision_tree_learning.R`. We test this code in the script `chap_18_prob_11.R`. When that code is run we get the plot given in Figure 3.

Problem 18.12 (learning a decision tree with missing attribute values)

The expression for the gain obtained when we split on an attribute A is given by

$$\begin{aligned}\text{Gain}(A) &= I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{Remainder}(A) \\ &= I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \sum_{i=1}^{v} \left(\frac{p_i + n_i}{p+n}\right) I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right) . \qquad (1)\end{aligned}$$

Recall that p_i and n_i are the positive and negative examples that result when we require the attribute A take on its ith value. We need a way to modify this calculation in the case where the training data might have missing values for this attribute. If we assume that there are m training samples with missing attributes for A then we would expect that on average if we had split these examples into positive and negative examples when they had the ith attribute values we would get

$$m\left(\frac{p_i}{p_i+n_i}\right),$$

additional positive examples and

$$m\left(\frac{n_i}{p_i+n_i}\right),$$

additional negative examples. Thus create given m training examples with missing values for attribute A we augment p_i and n_i to include the expected number of examples we would get for each of these m. This means that we can compute

$$\begin{aligned}\hat{p}_i &= p_i + m\left(\frac{p_i}{p_i+n_i}\right)\\ \hat{n}_i &= n_i + m\left(\frac{n_i}{p_i+n_i}\right),\end{aligned}$$

and run the standard tree building algorithm with $\hat{p}_i$ and $\hat{n}_i$ in place of p_i and n_i. This is implemented in the `R` code `decision_tree_learning.R`. To compare performance in the cases where there is missing attributes in the training data with the `R` code `chap_18_prob_12.R` we build trees with complete data and then trees with incomplete data (data that contains missing attribute values for a fixed number of the training samples). We then generate test data with the same properties and see how the testing errors compare. When that `R` code is run we get the plot in Figure 4. As we see when the number of training examples increases while the number of samples with an error stays constant the learned decision-tree will continue to improve the more data is observed.

Problem 18.13 (building a decision tree using the gain ratio)

Recall that the information gain when we split on attribute A is given by Equation 1 where the information content I is given by

$$I(p,q) = -p\log_2(p) - q\log_2(q)\,, \tag{2}$$

here p and q are probabilities so $p+q=1$. In general for a distribution that has v possible values v_i each with probabilities $P(v_i)$ for $i = 1, 2, \cdots, v-1, v$ we have the information content defined as

$$I(P(v_1), P(v_2), \cdots, P(v_{v-1}), P(v_v)) = -\sum_{i=1}^{v} P(v_i)\log_2(P(v_i))\,. \tag{3}$$

The **gain ratio** is then the ratio of the information gain (given by Equation 1) and the intrinsic information I given by Equation 3. In the `R` code `decision_tree_learning.R` there is an option that allows trees to be built using this criterion.

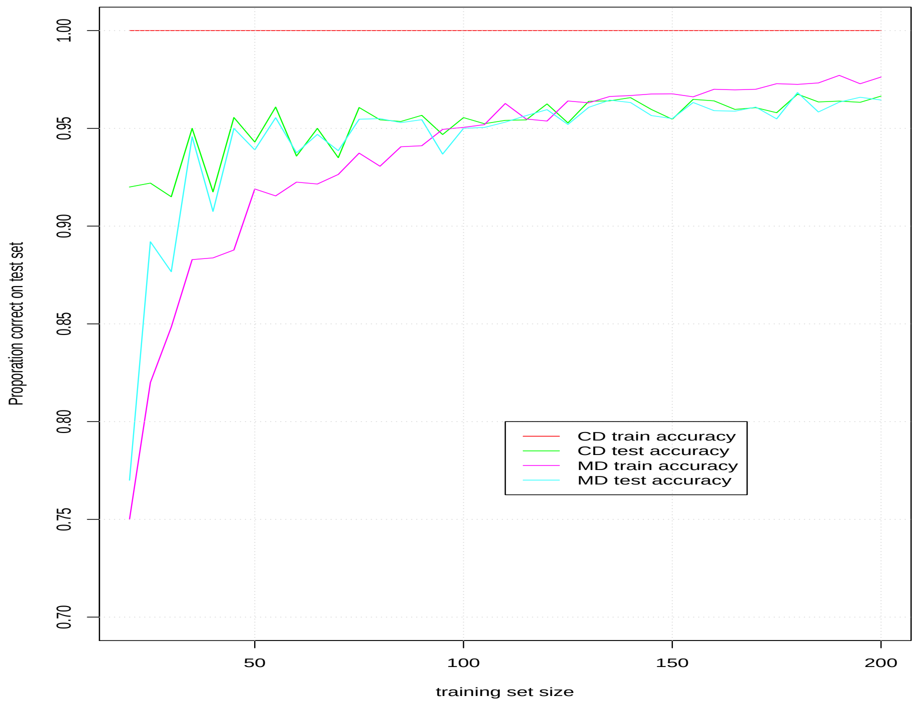


Figure 4: Learning rates for training with complete data (CD) and training with data that contains missing data (MD). Note that the training and testing error for the complete case (where there are no missing attributes represented as the R NA) is better in the long run than in the case where we have missing attributes. This is to be expected since we are loosing information from the training set with the introduction of missing attributes. For a fixed number of missing attributes the testing error learning rate in each case is asymptotic to the same value. This is to be expected because eventually the amount of good data overwhelms the amount of bad data. If we enforced a fixed fraction of data with missing attributes at every training set size we might see the error rates of the missing data case becomes a constant factor worse than in learning when we have complete data.

Problem 18.14 (an ensemble of learning algorithms)

We have that ϵ is the probability that a single classifier makes a mistake. We want to evaluate the probability that the ensemble of classifiers makes a mistake. This will happen if at least $\lceil \frac{M}{2} \rceil$ of the classifiers vote incorrectly (i.e. make a mistake). Assuming independence we will have exactly k classifiers make a mistake with probability

$$\binom{M}{k} \epsilon^k (1-\epsilon)^{N-k} .$$

we will thus have at least $\lceil \frac{M}{2} \rceil$ classifiers make a mistake with a probability of

$$\sum_{k=\lceil \frac{M}{2} \rceil}^{M} \binom{M}{k} \epsilon^k (1-\epsilon)^{N-k} .$$

For the given values of M and ϵ we find the above probability given by

```
[1] "    5     0.10   0.008560"
[1] "    5     0.20   0.057920"
[1] "    5     0.40   0.317440"
[1] "   10     0.10   0.000147"
[1] "   10     0.20   0.006369"
[1] "   10     0.40   0.166239"
[1] "   20     0.10   0.000001"
[1] "   20     0.20   0.000563"
[1] "   20     0.40   0.127521"
```

See the `R` script `chap_18_prob_14.R`.

Chapter 20 (Statistical Learning Methods)

Notes on the Text

Notes on Section 20.1 (the likelihood of various data sequences)

It may help to consider the general equations for posterior probability and the predicted probability in the case of observing 10 lime candies in a row. When $P(d_j = \text{lime}|h_3) = \frac{1}{2}$ if we see 10 limes in a row then we would have the probability of this data vector is

$$P(\mathbf{d} = \mathbf{lime}|h_3) = \left(\frac{1}{2}\right)^{10} .$$

Here $\mathbf{d}$ is the vector of all samples and **lime** is a vector of 10 lime readings. In general to determine the probability that a given hypothesis h_i for $1 \leq i \leq 5$ is true, given the data vector $\mathbf{d}$ of ten measurements we would need to evaluate

$$P(h_i|\mathbf{d}) = \alpha \left(\prod_{j=1}^{10} P(d_j|h_i) \right) P(h_i) . \tag{4}$$

In the above we need the individual likelihoods of each data i.e. $P(d_j|h_i)$. In this case these are computed from the information in the problem where we are told that

$$\begin{aligned}
P(d_j = \text{lime}|h_1) &= 0 \\
P(d_j = \text{lime}|h_2) &= \frac{1}{4} \\
P(d_j = \text{lime}|h_3) &= \frac{1}{2} \\
P(d_j = \text{lime}|h_4) &= \frac{3}{4} \\
P(d_j = \text{lime}|h_5) &= 1 .
\end{aligned}$$

Values for the likelihoods $P(d_j = \text{cherry}|h_i)$ can be computed using $P(d_j = \text{cherry}|h_i) = 1 - P(d_j = \text{lime}|h_i)$. Thus in the case where all 10 measurements are of lime candies the probability of each type of bag h_i using Equation 4 would be given by

$$P(h_i|\mathbf{d}) = \alpha P(\text{lime}|h_i)^{10} P(h_i) .$$

To compute the *predicted* next flavor given N measurements we would use

$$P(d_{N+1} = \text{lime}|d_1, \cdots, d_N) = \sum_{i=1}^{5} P(d_{N+1} = \text{lime}|h_i) P(h_i|d_1, \cdots, d_N) . \tag{5}$$

Here $P(h_i|d_1, \cdots, d_N)$ would be computed using Equation 4 but of course over N measurements and not just 10.

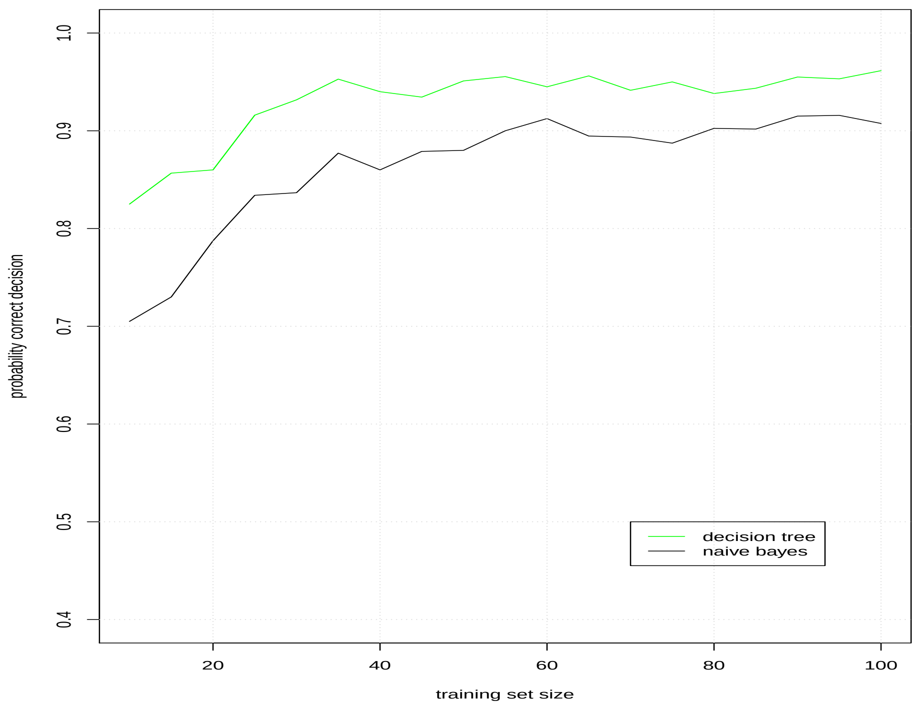


Figure 5: Bayesian learning when data is drawn from each of the five different bag types.

Notes on the Naive Bayes Model

In the R code `naive_bayes.R` we implement the Naive Bayes model. In the R code `dup_fig_20_3.R` we run this code and the code `decision_tree_learning.R` to learn models for the restaurant problem. The learning curves for these two methods are plotted in Figure 5. This plot agrees well with the one presented in the book.

Notes on Maximum-likelihood parameter learning in discrete models

With the joint density parametrization of the two variables (flavor, wrapper) given by

$$
\begin{aligned}
P(F = c, W = g) &= \theta(1-\theta_1) \\
P(F = c, W = r) &= \theta\theta_1 \\
P(F = l, W = g) &= (1-\theta)(1-\theta_2) \\
P(F = l, W = r) &= (1-\theta)\theta_2 \,.
\end{aligned}
$$

Here c is a cherry flavored candy, l is a lime flavored candy, g is a green wrapper, and r is a red wrapper. The parameters of the above model are θ, θ_1, and θ_2 (and thus the hypothesis h depends on them). We have that given a sequence of N candies in $\mathbf{d}$ where we observe both the flavor and the wrapper color the likelihood of this data sequence is given by

$$
P(\mathbf{d}|h_{\theta,\theta_1,\theta_2}) = \theta^c(1-\theta)^l\theta_1^{r_c}(1-\theta_1)^{g_c}\theta_2^{r_l}(1-\theta_2)^{g_l}\,.
$$

Thus the log-likelihood for the full data set looks like

$$\begin{aligned} L &= \log(P(\mathbf{d}|h_{\theta,\theta_1,\theta_2})) \\ &= [c\log(\theta) + l\log(1-\theta)] + [r_c\log(\theta_1) + g_c\log(1-\theta_1)] + [r_l\log(\theta_2) + g_l\log(1-\theta_2)] \,. \end{aligned}$$

which shows a decomposition into three parts each of which only depends on terms that depend only on one of the parameters θ, θ_1, and θ_2.

Notes on learning with hidden variables

The book discuss the number of parameters in the two Bayes networks given in Figure 20.7. The network on the left has a hidden variable $HeartDisease$. We assume that each variable has three states. In this section of these notes we document how the book counted the total number of variables. For the network on the left we need to specify 2 parameters to specify the distribution $P(Smoking)$ the third probability is constrained since the three numbers must sum to one. We have two more parameters to specify the distribution for each of $P(Diet)$ and $P(Exercise)$. This given a total of $2 + 2 + 2 = 6$ parameters to specify the first row. Then to specify the conditional probability distribution of $HeartDisease$ for each value of the parents, there are $3^3 = 27$ unique ways to specify the parents to this node and for each one we need 2 (the third one is determined such that everything sums to one). This means that we have $2 \times 27 = 54$ additional parameters. This gives at total of $6 + 54 = 60$ parameters thus far. Next to specify the conditional probability distribution of $P(Symptom_i|HeartDisease)$ for the three symptoms. We have to specify one of the three values for $HeartDisease$ and then for each value two parameters for 6 parameters for a symptom. For all three symptoms this gives an addition $3(6) = 18$ parameters. In total we thus have $60 + 18 = 78$ parameters.

Next for the complete network we again have to specify a total of 6 parameters to specify the prior distributions of $Smoking$, $Diet$, and $Exercise$ just as in the first network. To determine the number of parameters needed to describe the conditional probability table (CPT) for $Symptom_1$ we note that we have to specify the values of $Smoking$, $Diet$, and $Exercise$ which requires $3^3 = 27$ specifications and for each of them we have two parameters giving $2(27) = 54$ parameters. Next we determine the number of parameters needed to describe the conditional probability table for $Symptom_2$. We again have to specify the values of $Smoking$, $Diet$, and $Exercise$ but now we also have to specify the value of $Symptom_1$ which in total requires $3^4 = 81$ specifications for the conditioning variables of $Symptom_2$. Thus for this CPT we need $2(81) = 162$ parameters. Finally for the specification of the CPT for $Symptom_3$ (using the same logic) we have

$$3^5(2) = 486\,,$$

parameters. Thus in total we have $6 + 54 + 162 + 486 = 708$ parameters.

Notes on Learning Bayesian Networks with Hidden Variables

In the `python` code `dup_Bayes_nets_w_hidden_variables.py` we have some code to duplicate the numbers in using the EM algorithm to compute the parameter estimates in the Bayesian network for the mixture model for candy.

Notes on Single Layer Feed-Forward Neural Networks (Perceptrons)

Using code developed earlier for decision tree learning with the new codes in

- `majority_function_gen_data.R` to generate data for the majority function and
- `perceptron_learning.R` to learn a perceptron

we can run `dup_fig_20_22_majority_function.R` and `dup_fig_20_22_restaurant_problem.R` to duplicate the books Figure 20.22. The results of running these two scripts are given in Figure 6.

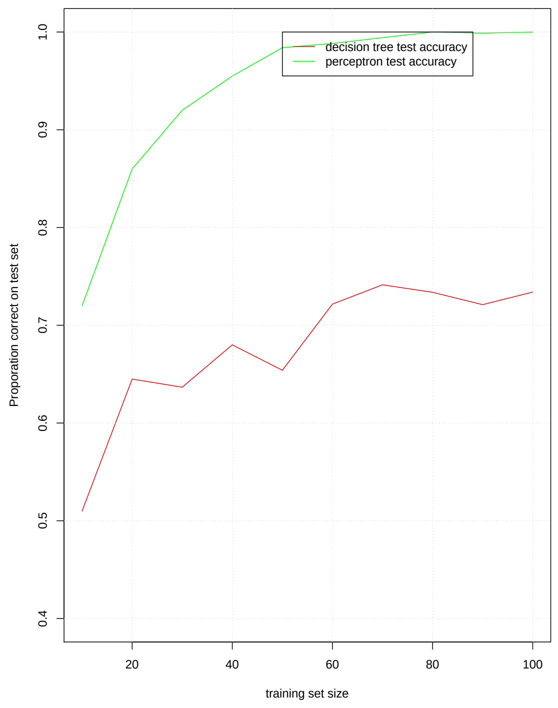
Problem Solutions

Problem 20.1-20.2 (Bayesian learning and prediction)

See the `R` code `chap_20_prob_1_N_2.R` where this problem is worked. When we run that `R` code we get the plots shown in Figure 7. Notice that in each case as enough data is drawn the type of bag the candies is coming from can be determined.

Problem 20.3 (when to trade/sell your candy)

Assume that bag starts with N candy pieces and we will draw and unwrap m candies. Then from this sequence of candies that we unwrap we get a "measurement" vector $\mathbf{d}$ with m measurements of either lime or cherry. Then we will have $N - m$ candies left. The fair value of the bag for Anne is given by its expected utility and the fair value of the bag for Bob is also given by his expected utility. The fair value is the price at which either Bob or Anne will pay for the given bag. If we compute the expected utility (over the different bag hypothesis



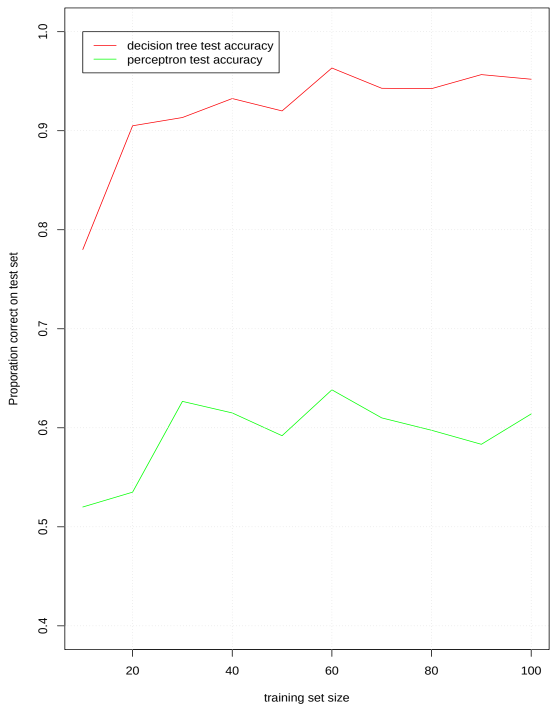
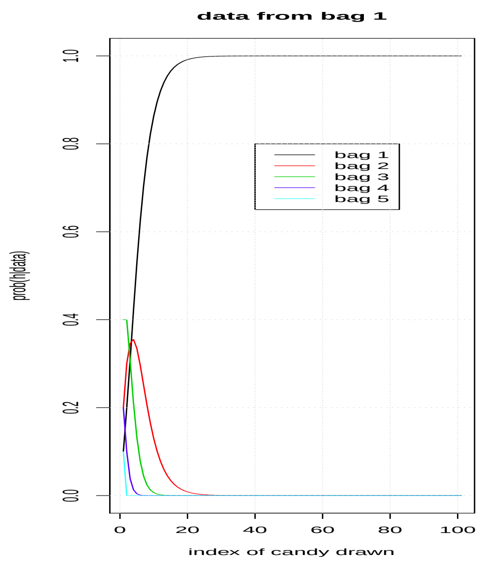
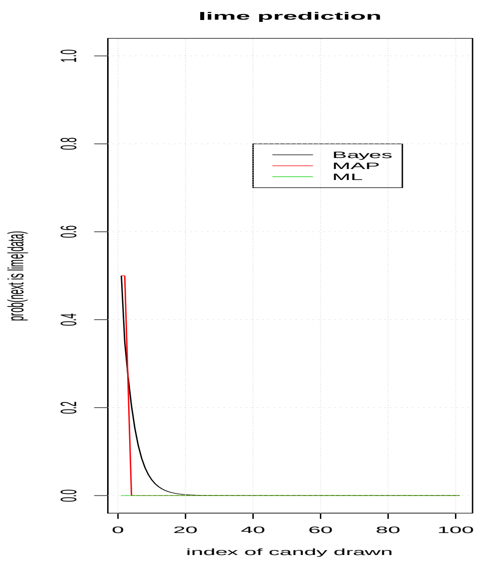
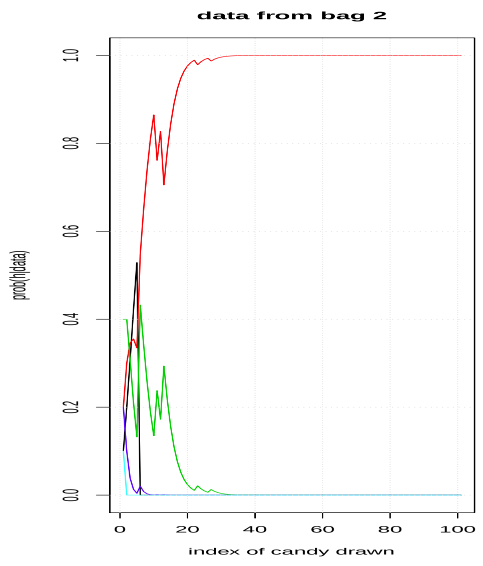
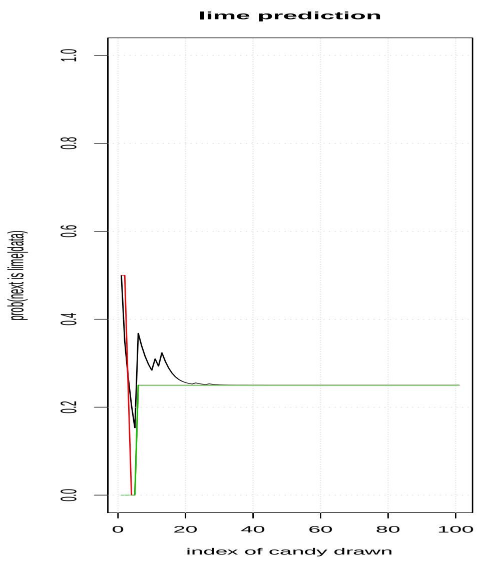


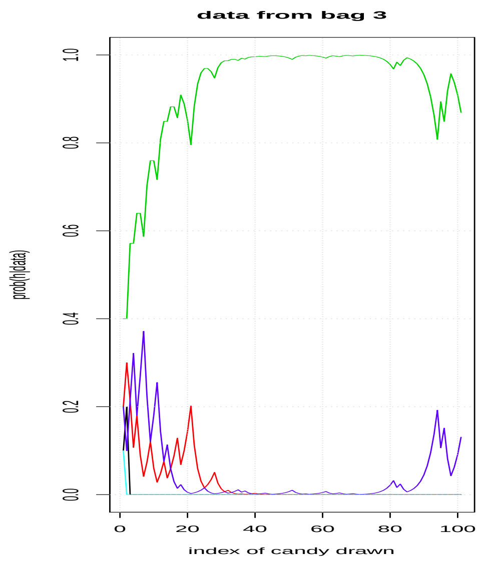
Figure 6: **Left:** Learning a decision tree (red curve) and a perceptron (green curve) on data from the majority function. Notice that on this learning task the perceptron is better able to learn this task. **Right:** Learning a decision tree (red curve) and a perceptron (green curve) on data from the restaurant problem. Notice that the decision tree vastly outperforms the perceptron in this learning task. Note in both of these learning tasks for given training set size we trained each algorithm 10 times and then report the average classification accuracy (the probability that we make a correct classification over the entire test dataset). This has the effect of smoothing each curve so that the true trend in learning can be more easily seen.

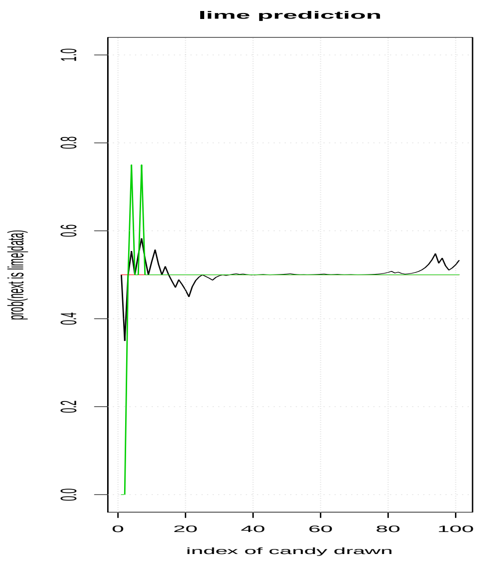

data from bag 1
bag 1
bag 2
bag 3
bag 4
bag 5
prob(h|data)
index of candy drawn

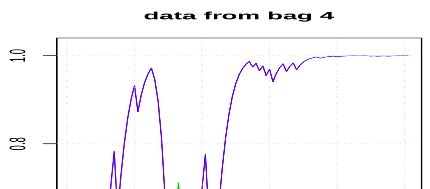

lime prediction
Bayes
MAP
ML
prob(next is lime|data)
index of candy drawn

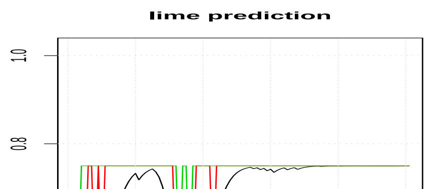

data from bag 2
prob(h|data)
index of candy drawn


lime prediction
prob(next is lime|data)
index of candy drawn


data from bag 3
prob(h|data)
index of candy drawn


lime prediction
prob(next is lime|data)
index of candy drawn


data from bag 4


lime prediction

h_i) for Anne after unwrapping the m pieces is given by

$$\begin{aligned}
E[U_A] &= c_A 1(N-m)P(h_1|\mathbf{d}) \\
&\quad + c_A 0.75(N-m)P(h_2|\mathbf{d}) + l_A 0.25(N-m)P(h_2|\mathbf{d}) \\
&\quad + c_A 0.5(N-m)P(h_3|\mathbf{d}) + l_A 0.5(N-m)P(h_3|\mathbf{d}) \\
&\quad + c_A 0.25(N-m)P(h_4|\mathbf{d}) + l_A 0.75(N-m)P(h_4|\mathbf{d}) \\
&\quad + l_A 1(N-m)P(h_5|\mathbf{d}) .
\end{aligned}$$

A row of the above can be reasoned as follows. For the second row, where we are considering h_2 to be true then we will have on average $0.75(N-m)$ cherry candies and $0.25(N-m)$ lime candies. For Anne each cherry candies carries a utility of c_A and each lime candy carries a utility of l_A. Thus when we weight by the probability of each hypothesis we get the above expression. The utility for Bob is computed in the same way and we get

$$\begin{aligned}
E[U_B] &= c_B 1(N-m)P(h_1|\mathbf{d}) \\
&\quad + c_B 0.75(N-m)P(h_2|\mathbf{d}) + l_B 0.25(N-m)P(h_2|\mathbf{d}) \\
&\quad + c_B 0.5(N-m)P(h_3|\mathbf{d}) + l_B 0.5(N-m)P(h_3|\mathbf{d}) \\
&\quad + c_B 0.25(N-m)P(h_4|\mathbf{d}) + l_B 0.75(N-m)P(h_4|\mathbf{d}) \\
&\quad + l_B 1(N-m)P(h_5|\mathbf{d}) .
\end{aligned}$$

Anne will want to sell her bag of candies when Bob's utility (the price he will pay) is greater than Anne's value of that bag or

$$E[U_B] > E[U_A] . \tag{6}$$

Notice that if we put the above two expressions for $E[U_A]$ and $E[U_B]$ into the above the expression $N-m$ cancel and the results are independent of the number of samples. To use this procedure one would unwrap candy until Equation 6 is true. At this point Anne should sell her bag of candy to Bob for an amount $E[U_B]$.

Problem 20.4 (two statisticians)

The first statistician (the Bayesian) would weight his drugs in proportion to how likely each disease is. Thus he would request a drug cocktail consisting of a mixture of 40% anti-A and 60% anti-B. The second statistician would request the maximum-likelihood solution which in this case would be to assume that the disease is B and then request to take all a cocktail of only anti-B.

When the type B disease now comes in two types dextro-B and levo-B since they are equally likely the possible disease sources are

- disease A 40%
- disease dextro-B 30%
- disease levo-B 30%

The Bayesian would now request a cocktail consisting of 40% anti-A, 30% anti-B, 30% anti-B which is really the same cocktail requested before. The second statistician would now select the maximum likelihood solution (which is disease A now) and thus request a cocktail of 100% anti-A. Thus the second statistician has now selected an entirely different cocktail while the Bayesian cocktail has not changed.

Problem 20.5 (naive Bayes learning with boosting)

In this problem, we compare the learning performance of decision trees, naive Bayes, and boosted naive Bayes on data for the restaurant problem. The naive Bayes algorithm is implemented in the routine `naive_bayes.R`, decision trees in the code `decision_tree_learning.R` and boosted naive bayes is implemented in the code `adaboost_w_naive_bayes.R`.

These three routines are then called from the script `chap_20_prob_5.R`. We then plot the learning curves for each algorithm in Figure 8. These are curves where we observe the test set accuracy as the number of samples in the dataset increases. In that figure we see that the best performer is the decision tree algorithm. This might be due to the fact that the actual data was generated from a function that is effectively a decision tree. Thus the decision tree learning is only having to learn the *parameters* (the attributes to split on at each stage) of the model and not the functional form. The other learners must learn the functional form (a decision tree) as well as the parameters of the tree. This means that we expect them to have a harder task to perform. We see that the boosted naive bayes is able to learn from the data better than direct naive bayes but only if enough data is present. The fact that more data needs to be supplied for boosted naive bayes to outperform direct naive bayes is an argument for using direct naive bayes (where in the real world data is always limited).

Problem 20.6 (maximum-likelihood parameter learning)

Since the distribution of y given x is assumed to be Gaussian it takes the functional form

$$p(y|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(y-(\theta_1 x+\theta_2))^2} .$$

The likelihood of the data $\mathbf{d}$ given the parameters of the model or $h_{\theta_1,\theta_2,\sigma}$ is

$$p(\mathbf{d}|h_{\theta_1,\theta_2,\sigma}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(y_i-(\theta_1 x_i+\theta_2))^2} .$$

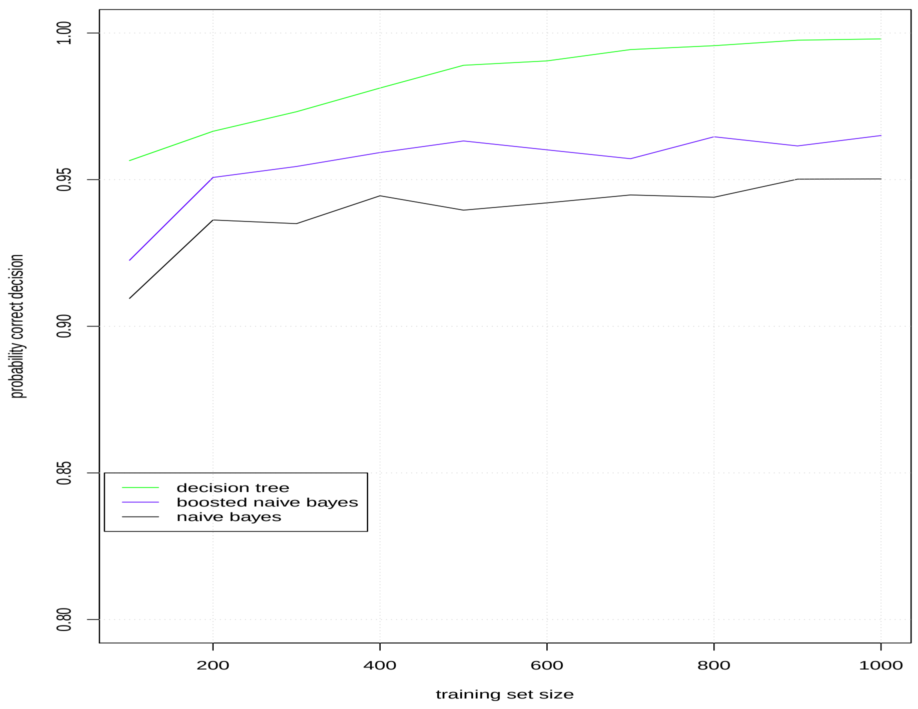


Figure 8: Comparing boosted naive Bayes learning with naive Bayes (by itself) and decision tree learning.

Thus the log-likelihood of this expression is given by

$$L = \log(P(\mathbf{d}|h_{\theta_1,\theta_2,\sigma})) = \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(y_i-(\theta_1 x_i+\theta_2))^2}\right)$$

$$= -N\log(\sqrt{2\pi}\sigma) - \sum_{i=1}^{N} \frac{1}{2\sigma^2}(y_i - (\theta_1 x_i + \theta_2))^2$$

$$= -N\log(\sqrt{2\pi}) - \frac{N}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - (\theta_1 x_i + \theta_2))^2 \,.$$

From the above, to find the maximum of L we need to take the derivatives with respect to the parameters (in this case θ_1, θ_2, and σ^2), set the resulting equations equal to zero, and solve for the parameters. We get

$$\frac{\partial L}{\partial \theta_1} = -\frac{1}{\sigma^2}\sum_{i=1}^{N}(y_i - (\theta_1 x_i + \theta_2))(-x_i) = 0\,, \tag{7}$$

$$\frac{\partial L}{\partial \theta_2} = -\frac{1}{\sigma^2}\sum_{i=1}^{N}(y_i - (\theta_1 x_i + \theta_2))(-1) = 0\,, \tag{8}$$

$$\frac{\partial L}{\partial \sigma^2} = -\frac{N}{2}\frac{1}{\sigma^2} + \frac{1}{2(\sigma^2)^2}\sum_{i=1}^{N}(y_i - (\theta_1 x_i + \theta_2))^2 = 0\,. \tag{9}$$

Note that in the above we are solving for σ^2 that maximizes the log-likelihood of the data (rather than σ) since that is computationally easier and equivalent. Distributing the sum in Equation 8 gives

$$\sum_{i=1}^{N} y_i - \theta_1 \sum_{i=1}^{N} x_i - \theta_2 N = 0\,.$$

If we divide this equation by N we get

$$\frac{1}{N}\sum_{i=1}^{N} y_i - \theta_1 \left(\frac{1}{N}\sum_{i=1}^{N} x_i\right) - \theta_2 = 0\,.$$

Equation 7 gives

$$\frac{1}{N}\sum_{i=1}^{N} x_i y_i - \theta_1 \left(\frac{1}{N}\sum_{i=1}^{N} x_i^2\right) - \theta_2 \left(\frac{1}{N}\sum_{i=1}^{N} x_i\right) = 0\,.$$

These give two equations and two unknowns for the variables θ_1 and θ_2. As a matrix system (and using somewhat standard notation for averages) this looks like

$$\left[\begin{array}{cc} \bar{x} & 1 \\ \overline{x^2} & \bar{x} \end{array}\right] \left[\begin{array}{c} \theta_1 \\ \theta_2 \end{array}\right] = \left[\begin{array}{c} \bar{y} \\ \overline{xy} \end{array}\right]\,.$$

This can be solved with Cramer's rule and we find

$$\theta_1 = \frac{\left|\begin{array}{cc} \bar{y} & 1 \\ \overline{xy} & \bar{x} \end{array}\right|}{\left|\begin{array}{cc} \bar{x} & 1 \\ \overline{x^2} & \bar{x} \end{array}\right|} = \frac{\bar{x}\bar{y} - \overline{xy}}{\bar{x}^2 - \overline{x^2}} \tag{10}$$

$$\theta_2 = \frac{\left|\begin{array}{cc} \bar{x} & \bar{y} \\ \overline{x^2} & \overline{xy} \end{array}\right|}{\left|\begin{array}{cc} \bar{x} & 1 \\ \overline{x^2} & \bar{x} \end{array}\right|} = \frac{\bar{x}\overline{xy} - \overline{x^2}\bar{y}}{\bar{x}^2 - \overline{x^2}}\,. \tag{11}$$

Once we have θ_1 and θ_2 solved for we can use Equation 9 to solve for σ^2. We find

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\theta_1 x_i + \theta_2))^2\,.$$

Note we would use the values of θ_1 and θ_2 found in Equations 10 and 11 to get the estimate of σ^2 using the above expression.

Problem 20.7 (maximum-likelihood learning of the noisy-OR model)

See Page 6 for some discussion on the noisy-OR model. If we assume that we have complete data meaning that we have true/false values for each of the variables ($fever, cold, flu, malaria$).

Then with N data points the total likelihood for this data is given by

$$\begin{aligned}
\prod_{i=1}^{N} P(fever_i, cold_i, flu_i, malaria_i) &= \prod_{i=1}^{N} P(fever_i|cold_i, flu_i, malaria_i) P(cold_i, flu_i, malaria_i) \\
&= P(cold_i)^N P(flu_i)^N P(malaria_i)^N \\
&\times \prod_{i=1}^{N} P(fever_i|cold_i, flu_i, malaria_i) \,.
\end{aligned}$$

Here we have assumed that the causes *cold*, flu, or *malaria* are all independent. Then to maximize the above likelihood is equivalent to maximizing the above conditional likelihood. To study how to maximize our likelihood lets generalize the notation a bit. We start with a Boolean (0 for false and 1 for true) symptom Y (like $fever$) and some number c of Boolean possible *causes* (like *cold*, flu, or *malaria*). The noisy-OR model is useful when its easier to specify the probability of the symptom given a single cause is true i.e.

$$P(Y=0|X_j=1, X_{-j}=0) = \theta_j \quad \text{or equivalently} \quad P(Y=1|X_j=1, X_{-j}=0) = 1-\theta_j \,.$$

Here the notation $X_{-j}=0$ means that we know that all the causes (but the jth which we don't necessarily know about) are false. Given the above parameters we can express the value of any set of conditioning variables on the event $Y=0$ as follows

$$\begin{aligned}
P(Y=0|X_1=x_1, X_2=x_2, \cdots, X_{c-1}=x_{c-1}, X_c=x_c) &= \prod_{j=1}^{c} P(Y=0|X_j=1, X_{-j}=0)^{x_j} \\
&= \prod_{j=1}^{c} \theta_j^{x_j} \,.
\end{aligned}$$

stating mathematically the fact that the probability we have our Boolean symptom $Y=0$ is the product of the probability we have $Y=0$ for each of the cause variables that are true $x_j=1$. Using this we can compute

$$P(Y=1|X_1=x_1, X_2=x_2, \cdots, X_{c-1}=x_{c-1}, X_c=x_c) = 1 - \prod_{j=1}^{c} \theta_j^{x_j} \,.$$

We can now express the total data likelihood in terms of the parameters θ_j. We assume that the realization of the symptom variable for each of our N cases is denoted y_i (for $i=1,\ldots,N$) and the realization of our possible causes for each of the N cases is denoted x_{ij} (for $i=1,\ldots,N$ and $j=1,\ldots,c$). Using this our conditional likelihood L is given by

$$\begin{aligned}
L &= \prod_{i=1}^{N} P(Y=y_i|x_{i1}, x_{i2}, \cdots, x_{i,c-1}, x_{i,c}) \\
&= \prod_{i=1}^{N} P(Y=1|x_{i1}, x_{i2}, \cdots, x_{i,c-1}, x_{i,c})^{y_i} P(Y=0|x_{i1}, x_{i2}, \cdots, x_{i,c-1}, x_{i,c})^{1-y_i} \\
&= \prod_{i=1}^{N} \left(1 - \prod_{j=1}^{c} \theta_j^{x_{ij}}\right)^{y_i} \left(\prod_{j=1}^{c} \theta_j^{x_{ij}}\right)^{1-y_i} \,.
\end{aligned}$$

The logarithm of this expression is given by

$$\begin{aligned}\log(L) &= \sum_{i=1}^{N}\left(y_i \log\left(1-\prod_{j=1}^{c}\theta_j^{x_{ij}}\right) + (1-y_i)\log\left(\prod_{j=1}^{c}\theta_j^{x_{ij}}\right)\right) \\ &= \sum_{i=1}^{N}\left(y_i \log\left(1-\prod_{j=1}^{c}\theta_j^{x_{ij}}\right) + (1-y_i)\sum_{j=1}^{c} x_{ij}\log\theta_j\right).\end{aligned}$$

To maximize the log-likelihood we need to take the derivative of the above with respect to the parameters $\theta_{j'}$, set the resulting expressions equal to zero, and solve for $\theta_{j'}$ for $j' = 1, \cdots, c$. The steps to compute the derivative of $\log(L)$ with respect to $\theta_{j'}$ are given by

$$\begin{aligned}\frac{\partial \log(L)}{\partial \theta_{j'}} &= \sum_{i=1}^{N}\left\{\left(\frac{-y_i}{1-\prod_{j=1}^{c}\theta_j^{x_{ij}}}\right)\frac{\partial}{\partial\theta_{j'}}\left(\prod_{j=1}^{c}\theta_j^{x_{ij}}\right) + (1-y_i)\frac{x_{ij'}}{\theta_{j'}}\right\} \\ &= \sum_{i=1}^{N}\left\{\left(\frac{-y_i}{1-\prod_{j=1}^{c}\theta_j^{x_{ij}}}\right)\left(\prod_{j=1;j\neq j'}^{c}\theta_j^{x_{ij}}\right)\left(x_{ij'}\theta_{j'}^{x_{ij'}-1}\right) + (1-y_i)\frac{x_{ij'}}{\theta_{j'}}\right\} \\ &= \sum_{i=1}^{N}\left\{\frac{-y_i x_{ij'}}{1-\prod_{j=1}^{c}\theta_j^{x_{ij}}}\left(\frac{\prod_{j=1}^{c}\theta_j^{x_{ij}}}{\theta_{j'}}\right) + (1-y_i)\frac{x_{ij'}}{\theta_{j'}}\right\} \\ &= \sum_{i=1}^{N}\left\{\frac{x_{ij'}}{\theta_{j'}}\left(\frac{1-y_i-\prod_{j=1}^{c}\theta_j^{x_{ij}}}{1-\prod_{j=1}^{c}\theta_j^{x_{ij}}}\right)\right\},\end{aligned}$$

when we simplify a bit. Setting these expression equal to zero would result in a nonlinear system of equations that would need to be solved for $\theta_{j'}$. It would probably be easier in practice to use the above derivatives in a gradient assent type algorithm to maximize $\log(L)$ numerically rather than analytically.

Problem 20.8 (the beta distribution)

Part (a): The books equation 20.6 is the definition of the beta distribution given by

$$\text{beta}[a,b](\theta) = \alpha\theta^{a-1}(1-\theta)^{b-1}. \tag{12}$$

From this expression in order for this to be a density we must have the integral with respect to θ over its domain (which is $[0,1]$) equal 1 or

$$\alpha\int_0^1 \theta^{a-1}(1-\theta)^{b-1}d\theta = 1.$$

The integral above is the definition of the `beta` function defined as

$$B(a,b) \equiv \int_0^1 \theta^{a-1}(1-\theta)^{b-1}d\theta.$$

It can be shown that the above integral expression is equivalent to

$$\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)},$$

see for example Rudin [1] and thus our normalization constant α is given by $\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$.

Part (b): The mean of the beta distribution is given by

$$\int_0^1 \theta \left(\alpha\theta^{a-1}(1-\theta)^{b-1}\right) d\theta = \alpha \int_0^1 \theta^a (1-\theta)^{b-1} d\theta = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \left(\frac{\Gamma(a+1)\Gamma(b)}{\Gamma(a+b+1)}\right) .$$

From the properties of the Gamma function we have that

$$\frac{\Gamma(a+b)}{\Gamma(a+b+1)} = \frac{1}{a+b} \quad \text{and}$$
$$\frac{\Gamma(a+1)}{\Gamma(a)} = a \, ,$$

so our expression for the mean becomes

$$E[\theta] = \frac{a}{a+b} .$$

Part (c): To find the mode we want to find the maximum of the beta distribution. To do this we take the derivative of the beta distribution with respect to θ, set the result equal to zero, and solve for θ. We find the first derivative given by

$$\alpha \left[(a-1)\theta^{a-2}(1-\theta)^{b-1} + \theta^{a-1}(b-1)(1-\theta)^{b-2}(-1)\right] .$$

If we set this equal to zero and then divide by $\alpha\theta^{a-2}(1-\theta)^{b-2}$ we get the equation

$$(a-1)(1-\theta) - (b-1)\theta = 0 \quad \text{or} \quad \theta = \frac{a-1}{a+b-2} .$$

Part (d): In the `R` code `chap_20_prob_8.R` we plot the beta distribution with $\epsilon = 10^{-3}$. We see that there is only nonzero probability at the two extremes $\theta = 0$ and $\theta = 1$. Thus the beta$[\epsilon, \epsilon](\theta)$ distribution gives a "two choice" distribution. When we update, say with a positive example we will obtain the beta$[\epsilon + 1, \epsilon](\theta)$ distribution. This is a distribution with almost all of the probability mass centered on $\theta = 1$. More positive samples make this distribution more peaked on $\theta = 1$. If we get an equal number of positive and negative examples the distribution gets more and more peaked around $\theta = 0.5$.

Problem 20.9 (adding a new link to a network cannot decrease the likelihood)

Imagine running maximum likelihood on the original network to get a set of parameters P_0 for $P(y|x_1, \cdots, x_k)$, the conditional probabilities of the child y given the parents $x_1, \cdots, x_k$ and a final likelihood of L_0. If we introduce a new parent (say x_{k+1}) to the variable y so that now we have to specify the conditional probabilities $P(y|x_1, \cdots, x_k, x_{k+1})$ if we use the old parameters P_0 from before for these probabilities and ignoring what the value of x_{k+1} is, then the new network will have the same likelihood L_0 as the first network. This is because when we compute the product of probabilities (or sum of log-probabilities) no mater what the value of x_{k+1} is we will have

$$P(y|x_1, \cdots, x_k, x_{k+1}) = P(y|x_1, \cdots, x_k) \, ,$$

and our likelihood in the new net will not be different from the likelihood in the old net. If we apply a maximum likelihood learning procedure in the new network we must end with a likelihood that is larger than the one we started with which is L_0. Thus the new network must have a larger or equal likelihood to the original net.

Problem 20.11 (The XOR network)

Since the XOR function is not linearly separable we will need a hidden unit to represent this function. Recall that the XOR function is given by the Table 1. If we plot each of the input

X	Y	XOR(X,Y)
0	0	0
1	0	1
0	1	1
1	1	0

Table 1: The output from the XOR function given its two inputs.

points in the (X, Y) plane we can see that we can separate the points that have a response of 1 with those that have a response of 0 by thinking about putting two lines in the (X, Y) plane and then requiring that the points with a response of 1 be between the two lines. The points not between these two lines should have a response of 0. Now note that the training points with $XOR(X, Y) = 1$ are between the two lines

$$x + y = \frac{1}{2} \, .$$

and

$$x + y = \frac{3}{2} \, .$$

We can construct these two lines in the first layer and then perform the AND mapping in the second layer. In Figure 9 we present the network that performs these two operations.

Problem 20.15 (Running the Perceptron)

WWX: This problem is not yet finished.

See the `R` code `chap_20_prob_15.R` where this problem is worked.

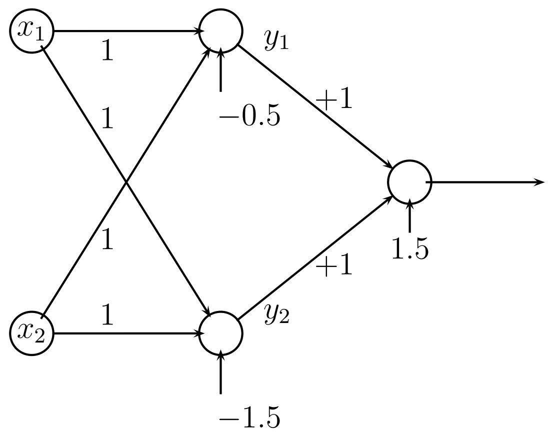


Figure 9: The XOR network for problem 11.

Problem 20.16

The books equation 20.13 is

$$L = T\log(p) + (1-T)\log(1-p)\,.$$

Here the output p as a function of the inputs given by

$$p = g\left(\sum_{j=0}^{n} W_j a_j\right)\,.$$

The later expression is because we are assuming a single layer percepton with a single output unit. The derivative of L with respect to W_j is given by

$$\frac{\partial L}{\partial W_j} = \frac{T}{p}\frac{\partial p}{\partial W_j} - \frac{(1-T)}{1-p}\frac{\partial p}{\partial W_j}\,.$$

For the sigmoidal activation unit we have $p(x) = \frac{1}{1+e^{-x}}$ and one can show that $p'(x) = p(1-p)$. Thus using this for $\frac{\partial L}{\partial W_j}$ we get

$$\frac{\partial L}{\partial W_j} = T(1-p)a_j - (1-T)pa_j = (T-p)a_j\,.$$

Note that $T-p$ is the error in the output unit and thus we have shown

$$\frac{\partial L}{\partial W_j} = Err \times a_j\,,$$

as we were to show.

Chapter 21 (Reinforcement Learning)

Notes on the Text

Notes on passive reinforcement learning

The first trial specified has 7 states each with the constant reward -0.04 and one state with the reward $+1$. The total reward from $(1,1)$ with a discount $\gamma = 1$ is then given by

$$7(-0.04) + 1 = -0.28 + 1 = 0.72\,,$$

for starting with the state $(1,1)$. For the state $(1,2)$ the first time we visit that state we find that the total reward then observed as we move towards the goal is given by

$$6(-0.04) + 1 = -0.24 + 1 = 0.76\,,$$

and the second time we visit $(1,2)$ we get a total reward of

$$4(-0.04) + 1 = -0.16 + 1 = 0.84\,.$$

These numbers agree with what the book presents.

References

[1] W. Rudin. *Principles of mathematical analysis*. McGraw-Hill Book Co., New York, third edition, 1976. International Series in Pure and Applied Mathematics.