1

a

$$P(\text{rain,accident}) = P(\text{rain}) \times P(\text{accident}) = .005$$
$$P(\text{accident}|\text{rain}) = P(\text{accident}) = .05$$

b

$$\begin{aligned} P(a|r) &= P(r|a)P(a)/P(r) \\ &= \frac{.4(.05)}{.1} \\ &= .2 \end{aligned}$$

c

Because the probability of having an accident and it being a rainy day are not independent.

d

Noting:

1. Probability of having a motif given a long length is observed to be zero.
2. Probability of having a repeat given medium GC is observed to be zero.

Thus in the naive bayes classifier we have

$$P(\text{motif}) = ... \times P(\text{motif}|\text{length} = \text{Long}) \times ... = 0$$

and

$$P(\text{repeat}) = ... \times P(\text{repeat}|\text{GC} = \text{Medium}) \times ... = 0$$

so we conclude with 100% certainty that class = gene.

2

a

$E_k[L]$ is the probability of choosing label j when the correct label is k times the loss associated with this misclassification, L_{kj}. Choosing the labeling "cancer" gives:

$$E_{\text{not cancer}}[L] = L_{cn} \times (P(k_n|d)) \overset{\text{(bayes)}}{=} L_{cn} \times \left(\frac{P(d|k_n) \times P(k_n)}{P(d)} \right)$$

where k_n denotes the actual state "not cancer". Choosing the labeling "not cancer" for d gives:

$$E_{\text{cancer}}[L] = L_{nc} \times (P(k_c|d)) \overset{\text{(bayes)}}{=} L_{nc} \times \left(\frac{P(d|k_c) \times P(k_c)}{Pd} \right)$$

b

In this case the decision rule simply chooses the label with the highest likelihood.

c

This loss matrix corresponds to equal losses for any misclassification.

3

See included files dataX_out.pdf. Also, see gibbs.py.

4

a

The speedup in using a hash table comes from not having to search a list to find an element but being able to immediately enter new entries into (in this case) a keyed dictionary. It is theoretically possible to implement a hash table for network motifs but hashing a given subgraph will be nontrivial and the table could take up an exponentially increasing space in memory.

In the parts that follow, I will assume the existence of such a hash table with table entries added as they are required (as in the "sample intensive approach" from the first pset). Keys for each entry of the hash table will be defined by the adjacency matrix of the first encountered graph in its equivalence class.

Noting that we can check graph equivalence of G, H by testing equality of the adjancy matrices $A_{\mathcal{P}(G)}, A_H$ for each permuation $\mathcal{P}$ of the vertices of G, the hash function to locate a new graph, G in our table will simply try to hash each $A_{\mathcal{P}(G)}$. In the case where G does not yet exist in the hash table, we will create a new entry for it.

b

Given the hash table above, it is trivial to test for a graph isomorphism. We simply construct a table hashing all of the subgraphs of H and then hash G. If G hashes successfully it is a subgraph. This algorithm will indeed be inefficient.

c

Construct a hash table tallying the number of occurences of every 5 node subgraph within the network $\mathcal{N}$ and then do the same thing for a thousand random networks of the same size and with the same connectivity as $\mathcal{N}$.

Computing the hash key for any such subgraph will take time constant time (of order 5!) and so overall runtime will have scale in worst case as the number of 5 vertex subsets of $\mathcal{N}$: $\binom{|V|}{5}$ times the number of subgraphs possible for each 5 vertex subset (in worst case: $(|5|^2)!$). We have to do this for $\mathcal{N}$ and a thousand random subgraphs, adding a $1000\times$ (constant) scale factor to run time but leaving the asymptotic form of worst case running unchanged

$$\text{runtime} \propto \binom{|V|}{5} \tag{1}$$

d

In a sample-centric approach we construct the hashing dictionary as entries are required by the sample. In the motif centric approach we create the dictionary at the outset and are guaranteed that any motif will hash. In the hash table approach to network motif discovery described above, the two approaches are exactly analogous to the approaches we saw in pset 1. The algorithm in (c) is sample centric.

e

Sample centric approaches are useful when to hash every possible motif would be difficult and we have reason to believe that only a small number of possible motifs will be realized in our sample. Motif centric approaches make sense when either the space of potential motifs is small or we believe that our sample will realize most possible motifs.

If we were only interested in a small subgraphs with a very small number of nodes or a fixed connectivity and number of edges we could thus approach this problem in a motif-centric manner. Given finite memory size, the above approach would be impractical in either case.

f

The easiest way to improve performance would be to implement the hash function more intelligently. Instead of checking the equivalency of adjancency matrices in every permutation of subgraph vertices we could choose a preferred ordering for the vertices (e.g by connection number, by connection number of adjacent vertices, etc) and test the hash function in one shot. This speedup would apply to either a sequence or a motif centric approach.