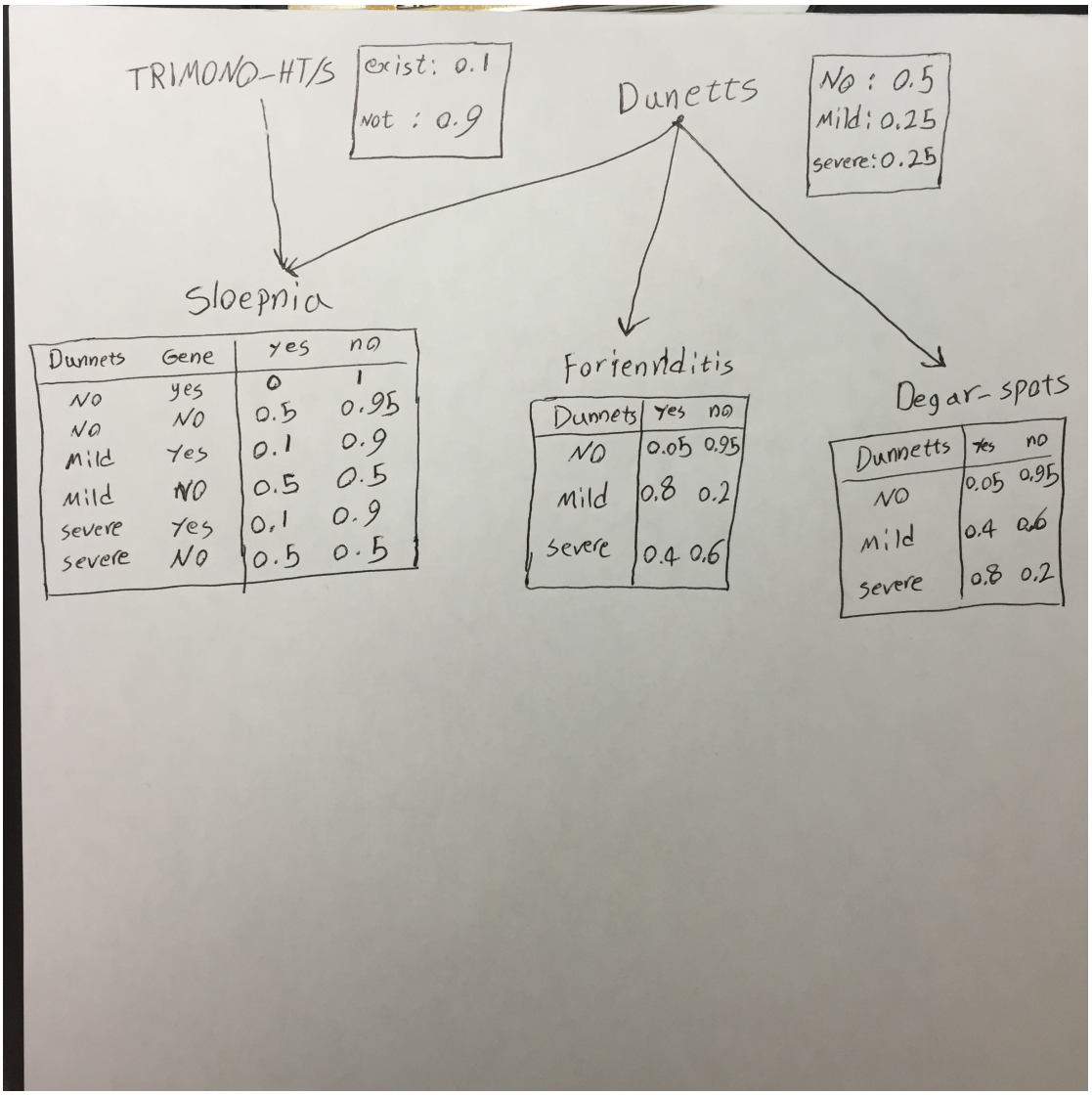
University of Waterloo
CS 486 - Fall 2015
Assignment 4

Sina Motevalli 20455091

Problem 1

Bayesian Network



Dunnets	Gene	yes	no
No	yes	0	1
No	No	0.5	0.95
Mild	Yes	0.1	0.9
Mild	NO	0.5	0.5
severe	Yes	0.1	0.9
severe	NO	0.5	0.5

Dunnets	yes	no
NO	0.05	0.95
Mild	0.8	0.2
severe	0.4	0.6

Dunnetts	Yes	no
NO	0.05	0.95
Mild	0.4	0.6
severe	0.8	0.2

Code

```
#include <iostream>
#include <map>
#include <fstream>
#include <vector>
#include <math.h>
#include <time.h>
#include <cmath>
using namespace std;



int Train_Slop[2000];
int Train_Forienn[2000];
int Train_Degar[2000];
int Train_Genes[2000];
int Train_Dunnet[2000];

int Test_Slop[100];
int Test_Forienn[100];
int Test_Degar[100];
int Test_Genes[100];
int Test_Dunnet[100];

double Dunnet[3];
double Gene[2];
double Sloepnea[3][2][2];
double Forienndits[3][2];
double Degar_Spots[3][2];



double SUMS[2][2][2][2][3];


double mean(double x[], int size)
{
double sum=0;
double mean2;
for ( int i = 0; i <size; i++ )
{
sum += x[i];
}
mean2=sum/ (double) size;
return mean2;
}

double deviation (double x[], int size, double mean)
{
double deviation;
double sum2=0;

for ( int i = 0; i < size; i++ )
{
sum2 += pow((x[i]-mean),2);
}
deviation= sqrt(sum2/(size));
return deviation;
}


```

```
61  /*
62    This function computes P(Dunnet = dunnet , everything else)
63  */
64  double Prob_Dunnet(int Sloep, int Fori, int deg, int gene, int dunnet) {
65          double P = Dunnet[dunnet];
66          double P_F = Forienndit is[dunnet][Fori];
67          double P_D = Degar_Spots[dunnet][deg];
68          double P_G = Gene[gene];
69          double P_S_Given_G = Sloepnea[dunnet][gene][Sloep];
70          double result = P_F*P_D*P_G*P_S_Given_G*P;
71          return result;
72  }
73
74
75
76  double SumOfStuff() {
77          double sum = 0;
78          for(int i = 0; i < 2000; i++) {
79                  if(Train_Dunnet[i] == 1 ) {
80                          sum = sum + 1;
81                          SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][1] =
82                          SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][1] + 1;
83                  }
84                  else if( Train_Dunnet[i] == 2) {
85                          sum = sum + 1;
86                          SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][2] =
87                          SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][2] + 1;
88                  }
89                  else if( Train_Dunnet[i] == 0 ) {
90                          sum = sum + 1;
91                          SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][0] =
92                          SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][0] + 1;
93                  }
94                  else if(Train_Dunnet[i] == -1) {
95                          double no = Prob_Dunnet(Train_Slop[i], Train_Forienn[i], Train_Degar
                                [i],Train_Genes[i],0);
96                          double mild = Prob_Dunnet(Train_Slop[i], Train_Forienn[i],
                                Train_Degar[i],Train_Genes[i],1);
97                          double severe = Prob_Dunnet(Train_Slop[i], Train_Forienn[i],
                                Train_Degar[i],Train_Genes[i],2);
98                          sum = no+mild+severe;
99                          double s = no+mild+severe;
100                         double P_NO = no/s;
101                         double P_MILD = mild/s;
102                         double P_SEVERE = severe/s;
103                         SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][0] =
104                         SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][0] + P_NO;
105                         SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][1] =
106                         SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][1] + P_MILD;
107                         SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][2] =
108                         SUMS[Train_Slop[i]][Train_Forienn[i]][Train_Degar[i]][Train_Genes[i
                                ]][2] + P_SEVERE;
109             }
110         }
```

```
111         return sum;
112 }
113
114
115
116 double SumUp(int b1, int b2, int b3, int b4, int b5) {
117         double sum = 0;
118         for(int a1 = 0; a1 < 2; a1++) {
119                 for(int a2 = 0; a2 < 2; a2++) {
120                         for(int a3 = 0; a3 < 2; a3++) {
121                                 for(int a4 = 0; a4< 2; a4++) {
122                                         for(int a5 = 0; a5 < 3; a5++) {
123                                                 if( (a1 == b1 || b1 == -1) && (a2 == b2 ||
                                                       b2 == -1) && (a3 == b3 || b3 == -1)
                                                     &&
124                                                       (a4 == b4 || b4 == -1) && (a5 == b5
                                                           || b5 == -1)) {
125                                                       sum = sum + SUMS[a1][a2][a3][a4][a5
                                                           ];
126                                                 }
127                                         }
128                                 }
129                         }
130                 }
131         }
132         return sum;
133 }
134
135
136
137 void UpdateDunTable() {
138      double mild = SumUp(-1,-1,-1,-1,1);
139      double severe = SumUp(-1,-1,-1,-1,2);
140      mild = (double) mild/2000;
141      severe = (double) severe/2000;
142      Dunnet[0] = 1.0 - (mild+severe);
143      Dunnet[1] = mild;
144      Dunnet[2] = severe;
145 }
146
147
148 void UpdateGeneTable(){
149         double yes = SumUp(-1,-1,-1,1,-1);
150         yes = (double) yes/2000;
151         Gene[1] = yes;
152         Gene[0] = 1.0 - yes;
153 }
154
155
156 void UpdateSloepneaTable() {
157         Sloepnea[0][0][1] = SumUp(1,-1,-1,0,0)/SumUp(-1,-1,-1,0,0);
158         Sloepnea[1][0][1] = SumUp(1,-1,-1,0,1)/SumUp(-1,-1,-1,0,1);
159         Sloepnea[2][0][1] = SumUp(1,-1,-1,0,2)/SumUp(-1,-1,-1,0,2);
160         Sloepnea[0][1][1] = SumUp(1,-1,-1,1,0)/SumUp(-1,-1,-1,1,0);
161         Sloepnea[1][1][1] = SumUp(1,-1,-1,1,1)/SumUp(-1,-1,-1,1,1);
162         Sloepnea[2][1][1] = SumUp(1,-1,-1,1,2)/SumUp(-1,-1,-1,1,2);
163         for(int i = 0; i < 3; i++) {
164                 for(int j = 0; j < 2; j++) {
165                         Sloepnea[i][j][0] = 1.0 - Sloepnea[i][j][1];
166                 }
167         }
168 }
169
170
171 void UpdateForidennditisTable() {
```

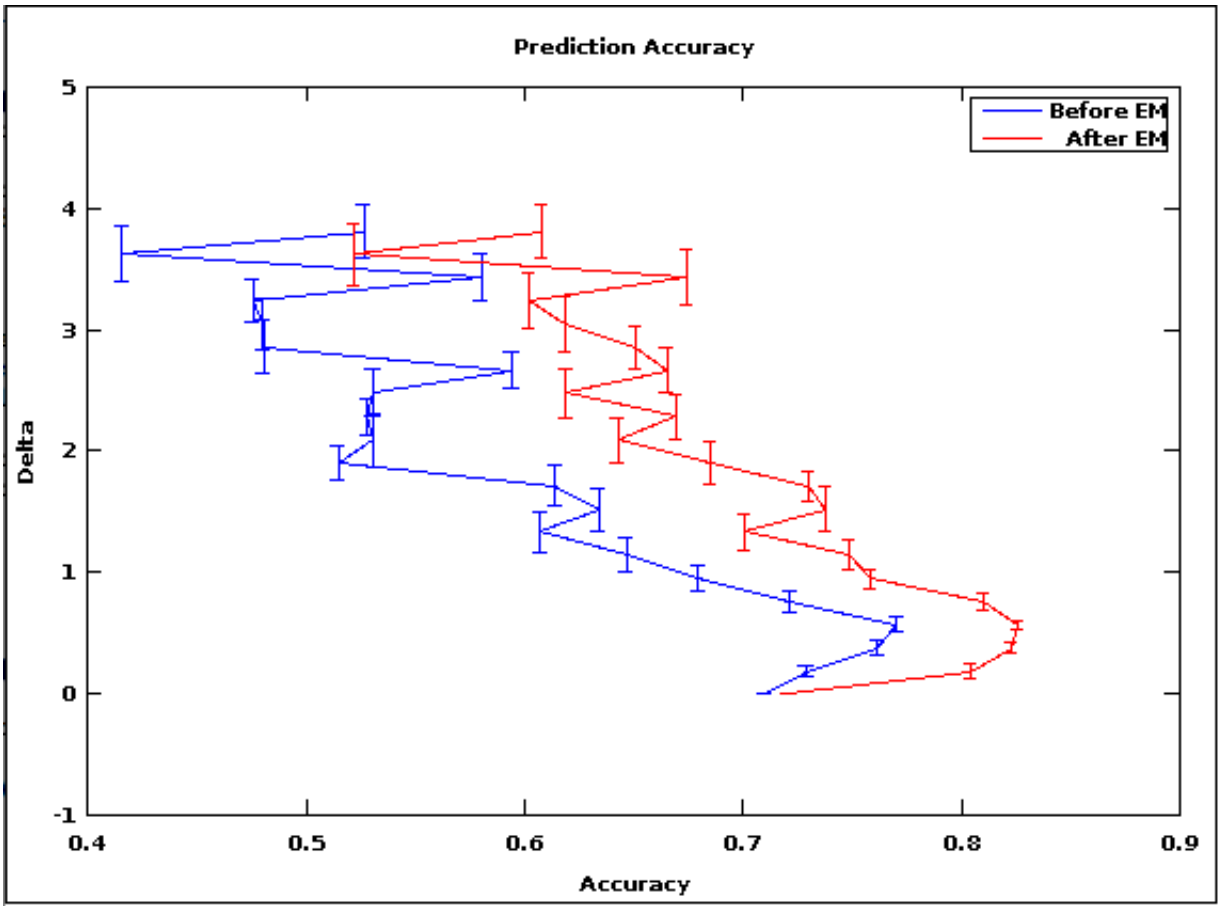
```
172         Forienndit is[0][1] = SumUp(-1,1,-1,-1,0)/SumUp(-1,-1,-1,-1,0);
173         Forienndit is[1][1] = SumUp(-1,1,-1,-1,1)/SumUp(-1,-1,-1,-1,1);
174         Forienndit is[2][1] = SumUp(-1,1,-1,-1,2)/SumUp(-1,-1,-1,-1,2);
175         for(int i = 0; i < 3; i++) {
176                 Forienndit is[i][0] = 1.0 - Forienndit is[i][1];
177         }
178 }
179
180
181 void UpdateDegarTable() {
182         Degar_Spots[0][1] = SumUp(-1,-1,1,-1,0)/SumUp(-1,-1,-1,-1,0);
183         Degar_Spots[1][1] = SumUp(-1,-1,1,-1,1)/SumUp(-1,-1,-1,-1,1);
184         Degar_Spots[2][1] = SumUp(-1,-1,1,-1,2)/SumUp(-1,-1,-1,-1,2);
185         for(int i = 0; i < 3; i++) {
186                 Degar_Spots[i][0] = 1.0 - Degar_Spots[i][1];
187         }
188 }
189
190
191 void MakeSumZero() {
192         for(int a1 = 0; a1 < 2; a1++) {
193                 for(int a2 = 0; a2 < 2; a2++) {
194                         for(int a3 = 0; a3 < 2; a3++) {
195                                 for(int a4 = 0; a4< 2; a4++) {
196                                         for(int a5 = 0; a5 < 3; a5++) {
197                                                 SUMS[a1][a2][a3][a4][a5] = 0;
198                                         }
199                                 }
200                         }
201                 }
202         }
203 }
204
205
206 double TestAccuracy() {
207         double correct = 0;
208         for(int i = 0; i < 100; i++) {
209                 double P0 = Prob_Dunnet(Test_Slop[i],Test_Forienn[i],Test_Degar[i],
                        Test_Genes[i],0);
210                 double P1 = Prob_Dunnet(Test_Slop[i],Test_Forienn[i],Test_Degar[i],
                        Test_Genes[i],1);
211                 double P2 = Prob_Dunnet(Test_Slop[i],Test_Forienn[i],Test_Degar[i],
                        Test_Genes[i],2);
212                 int label;
213                 if(P0 >= P1 && P0 >= P2) {
214                         label = 0;
215                 }
216                 else if(P1 >= P0 && P1 >= P2) {
217                         label = 1;
218                 }
219                 else {
220                         label = 2;
221                 }
222                 if( label == Test_Dunnet[i] ) {
223                         correct = correct + 1;
224                 }
225         }
226         return correct/(double)100;
227 }
228
229
230 double fRand(double fMax)
231 {
232     double f = (double)rand() / RAND_MAX;
233     return f * (fMax );
```

```
}


void setinitialvalue(double max) {
        double s;

        Dunnet[0] = 0.5 + fRand(max);
        Dunnet[1] = 0.25 + fRand(max);
        Dunnet[2] = 0.25 + fRand(max);
        s = (Dunnet[0]+Dunnet[1]+Dunnet[2]);
        Dunnet[0] = Dunnet[0]/s;
        Dunnet[1] = Dunnet[1]/s;
        Dunnet[2] = Dunnet[2]/s;



        Gene[0] = 0.9 + fRand(max);
        Gene[1] = 0.1 + fRand(max);
        s = (Gene[0]+Gene[1]);
        Gene[0] = Gene[0] / s;
        Gene[1] = Gene[1] / s;


        Sloepnea[0][0][0] = 0.95 +fRand(max);
        Sloepnea[0][0][1] = 0.05 +fRand(max);
        s = (Sloepnea[0][0][1] + Sloepnea[0][0][0]);
        Sloepnea[0][0][0] = Sloepnea[0][0][0] / s;
        Sloepnea[0][0][1] = Sloepnea[0][0][1] / s;

        Sloepnea[0][1][0] = 1.0 +fRand(max);
        Sloepnea[0][1][1] = 0.0 +fRand(max);
        s = (Sloepnea[0][1][0] + Sloepnea[0][1][1]);
        Sloepnea[0][1][0] = Sloepnea[0][1][0] / s;
        Sloepnea[0][1][1] = Sloepnea[0][1][1] / s;


        Sloepnea[1][0][0] = 0.5 +fRand(max);
        Sloepnea[1][0][1] = 0.5 +fRand(max);
        s = Sloepnea[1][0][0]+ Sloepnea[1][0][1];
        Sloepnea[1][0][0] = Sloepnea[1][0][0]/s;
        Sloepnea[1][0][1] = Sloepnea[1][0][1]/s;

        Sloepnea[2][0][0] = 0.5 +fRand(max);
        Sloepnea[2][0][1] = 0.5 +fRand(max);
        s = Sloepnea[2][0][0]+Sloepnea[2][0][1];
        Sloepnea[2][0][0] = Sloepnea[2][0][0]/s;
        Sloepnea[2][0][1] = Sloepnea[2][0][1]/s;



        Sloepnea[1][1][0] = 0.95 +fRand(max);
        Sloepnea[1][1][1] = 0.05 +fRand(max);
        s = Sloepnea[1][1][0]+ Sloepnea[1][1][1];
        Sloepnea[1][1][0] = Sloepnea[1][1][0]/s;
        Sloepnea[1][1][1] = Sloepnea[1][1][1]/s;

        Sloepnea[2][1][0] = 0.95 +fRand(max);
        Sloepnea[2][1][1] = 0.05 +fRand(max);
        s = Sloepnea[2][1][0]+Sloepnea[2][1][1];
        Sloepnea[2][1][0] = Sloepnea[2][1][0]/s;
        Sloepnea[2][1][1] = Sloepnea[2][1][1]/s;



        Forienndit is[0][0] = 0.95 +fRand(max);
```

```
        Forienndi tis[0][1] = 0.05 +fRand(max);
        s = Forienndi tis[0][0]+Forienndi tis[0][1];
        Forienndi tis[0][0] = Forienndi tis[0][0]/s;
        Forienndi tis[0][1] = Forienndi tis[0][1]/s;


        Forienndi tis[1][0] = 0.2 +fRand(max);
        Forienndi tis[1][1] = 0.8 +fRand(max);
        s = Forienndi tis[1][0]+ Forienndi tis[1][1];
        Forienndi tis[1][0] = Forienndi tis[1][0]/s;
        Forienndi tis[1][1] = Forienndi tis[1][1]/s;

        Forienndi tis[2][0] = 0.6 +fRand(max);
        Forienndi tis[2][1] = 0.4 +fRand(max);
        s = Forienndi tis[2][0]+Forienndi tis[2][1];
        Forienndi tis[2][0] = Forienndi tis[2][0]/s;
        Forienndi tis[2][1] = Forienndi tis[2][1]/s;




        Degar_Spots[0][0] = 0.95 +fRand(max);
        Degar_Spots[0][1] = 0.05+ fRand(max);
        s = Degar_Spots[0][0]+Degar_Spots[0][1];
        Degar_Spots[0][0] = Degar_Spots[0][0]/s;
        Degar_Spots[0][1] = Degar_Spots[0][1]/s;


        Degar_Spots[1][0] = 0.6 +fRand(max);
        Degar_Spots[1][1] = 0.4 +fRand(max);
        s = Degar_Spots[1][0]+Degar_Spots[1][1];
        Degar_Spots[1][0] = Degar_Spots[1][0]/s;
        Degar_Spots[1][1] = Degar_Spots[1][1]/s;

        Degar_Spots[2][0] = 0.2 +fRand(max);
        Degar_Spots[2][1] = 0.8 +fRand(max);
        s = Degar_Spots[2][0]+Degar_Spots[2][1];
        Degar_Spots[2][0] = Degar_Spots[2][0]/s;
        Degar_Spots[2][1] = Degar_Spots[2][1]/s;


}



void Start() {
        double before = 1;
        double after = 2;
        while(after-before > 0.1 || after - before < -0.1) {
                MakeSumZero();
                before = SumOfStuff();
                UpdateDunTable();
                UpdateGeneTable();
                UpdateSloepneaTable();
                UpdateForidennditisTable();
                UpdateDegarTable();
                after = SumOfStuff();
        }
}


int main() {
        ifstream my;
        my.open("traindata.txt");
```

```
364         for(int i = 0; i < 2000; i++) {
365                 int sloe, fori, degar, gen, dun;
366                 my >> sloe >> fori >> degar >> gen >> dun;
367                 Train_Slop[i] = sloe;
368                 Train_Forienn[i] = fori;
369                 Train_Degar[i] = degar;
370                 Train_Genes[i] = gen;
371                 Train_Dunnet[i] = dun;
372         }
373         my.close();
374         my.open("testdata.txt");
375         for(int i = 0; i < 100; i++) {
376                 int sloe, fori, degar, gen, dun;
377                 my >> sloe >> fori >> degar >> gen >> dun;
378                 Test_Slop[i] = sloe;
379                 Test_Forienn[i] = fori;
380                 Test_Degar[i] = degar;
381                 Test_Genes[i] = gen;
382                 Test_Dunnet[i] = dun;
383         }
384         my.close();
385         ////////////////////
386         time_t seconds;
387     time(&seconds);
388     srand((unsigned int) seconds);
389         ofstream deltas,meansbfore,meansafter,sdbefore,sdafter;
390         deltas.open("delta");
391         meansbfore.open("menbefore");
392         meansafter.open("meansafter");
393         sdbefore.open("sdbefore");
394         sdafter.open("sdafter");
395
396
397         double add = (double) 4.0/21.0;
398         double i = 0;
399         while(i < 4) {
400                 double x[20];
401                 double y[20];
402                 for(int j = 0; j < 20; j++) {
403                         setinitialvalue(i);
404                         y[j] = TestAccuracy();
405                         Start();
406                         x[j] = TestAccuracy();
407
408                 }
409                 double m = mean(x,20);
410                 double sd = deviation(x,20,m);
411                 double mbefore = mean(y,20);
412                 double SDbefore = deviation(y,20,mbefore);
413                 deltas << i << endl;
414                 meansbfore  << mbefore << endl;
415                 meansafter <<  m << endl;
416                 sdbefore << SDbefore << endl;
417                 sdafter << sd << endl;
418                 i = i + add;
419         }
420         deltas.close();
421         deltas.close();
422         meansbfore.close();
423         meansafter.close();
424         sdbefore.close();
425         sdafter.close();
426 }
```

Accuracy Graph



Problem 2

Standard Gamble

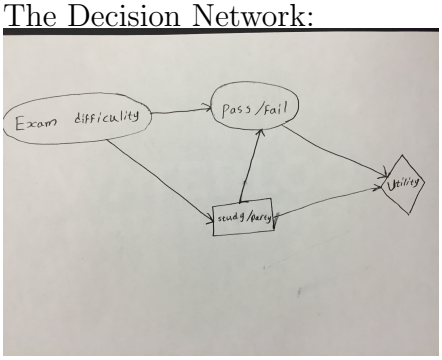
My utility Table:

party/study	pass/fail	utility
party	pass	1
study	pass	0.8
party	fail	0.2
study	fail	0

Here is how I chose the utilities. I considered (party,pass) with utility 1 (the highest utility) and (study,fail) with utility 0, the lowest utility. Then I set up the lottery as follows:
For $p \in [0, 1]$, I get (party, study) with probability p and I get (study,fail) with probability $1 - p$.
Then given some other $o \in \{party, study\} \times \{pass, fail\}$, I considered with which values of p am I going to choose o and with which values am I going to gamble. The breaking point of these values became the utilities I chose in the above chart.

Decision Network and CPTs

The Decision Network:



The CPTs:

Hard/Easy	Study/Party	Pass	Fail
Easy	Study	0.9	0.1
Hard	Study	0.6	0.4
Easy	Party	0.6	0.4
Hard	Party	0.35	0.65

party/study	pass/fail	utility
party	pass	1
study	pass	0.8
party	fail	0.2
study	fail	0

Value of partying and studying

We have the following 2 cases to consider:

Easy course

$$\begin{aligned} E(u|D = party, easy) &= P(fail\ |party, easy)u(fail, party) + P(pass\ |party, easy)u(pass, party) \\ &= (0.4)(0.2) + (0.6)(1) \\ &= 0.68 \end{aligned}$$

$$\begin{aligned} E(u|D = study, easy) &= P(fail\ |study, easy)u(fail, study) + P(pass\ |study, easy)u(pass, study) \\ &= (0.1)(0) + (0.9)(0.8) \\ &= 0.72 \end{aligned}$$

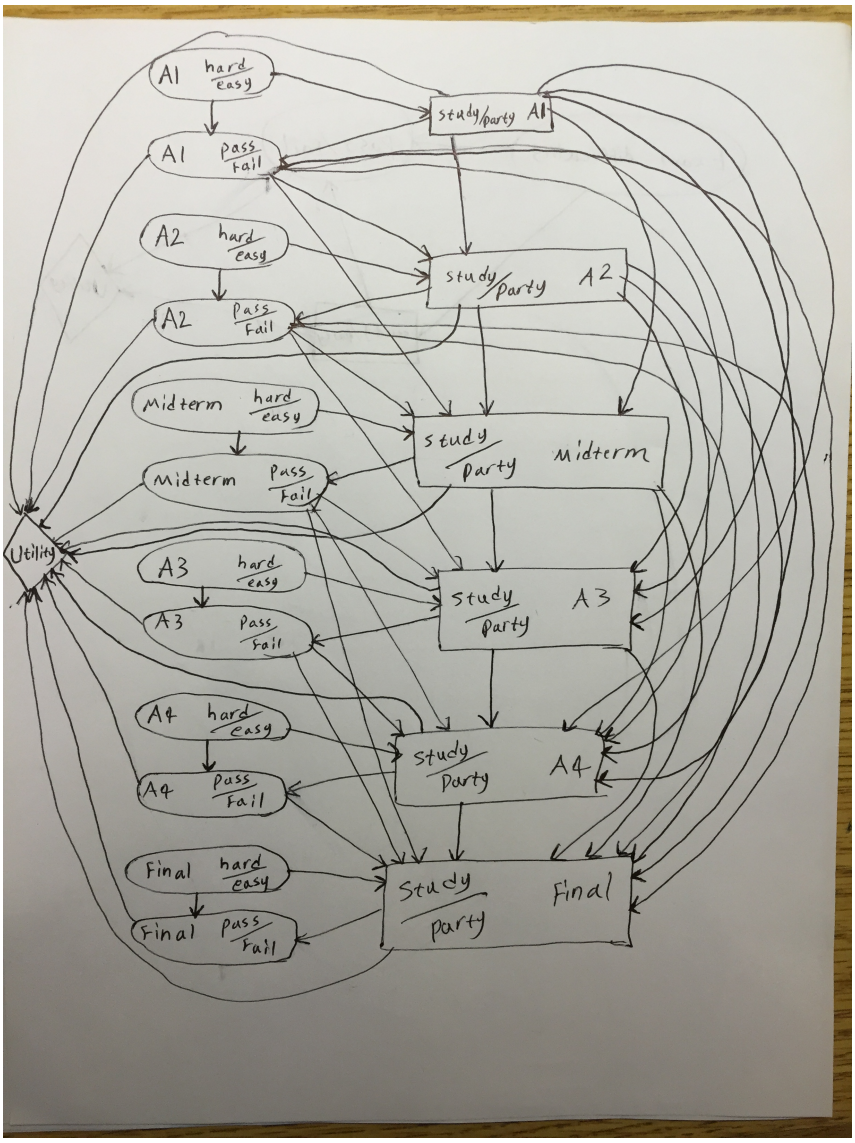
Hard course

$$\begin{aligned} E(u|D = party, hard) &= P(fail\ |party, hard)u(fail, party) + P(pass\ |party, hard)u(pass, party) \\ &= (0.65)(0.2) + (0.35)(1) \\ &= 0.48 \end{aligned}$$

$$\begin{aligned} E(u|D = study, hard) &= P(fail\ |study, hard)u(fail, study) + P(pass\ |study, hard)u(pass, study) \\ &= (0.4)(0) + (0.6)(0.8) \\ &= 0.48 \end{aligned}$$

So for easy courses, the best policy is to study. For hard courses, it does not matter, in both cases the expected utility is the same.

Decision Network for the sequential version



To make the task of reading the above decision network easier, the following are descriptions of certain aspects of the network:

1. All the assignment, the midterm, the final, and all the decisions are connected to the utility node.

2. For a decision node to study or party before an exam, we have the results of all the previous exams and all the past decisions and the difficulty of that exam connecting to the decision node.