CS228: Probabilistic Graphical Models

Luke Jaffe

Due: 02/03/2017
Submitted: 02/05/2017

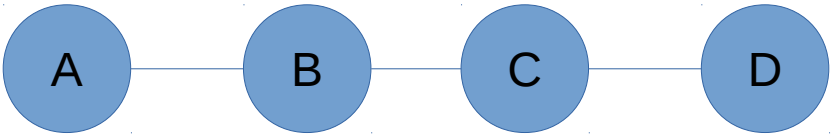
Problem 1: I-Maps and P-Maps

P satisfies:

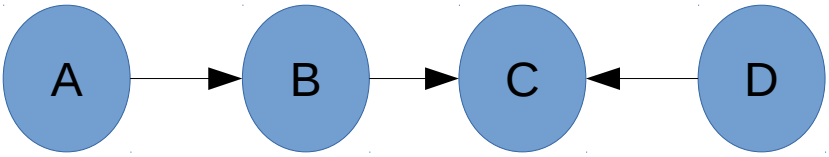
1. A ⊥C | B
2. A ⊥C | B, D
3. A ⊥D | B
4. A ⊥D | B, C
5. B ⊥D
6. B ⊥D | A
7. A ⊥D

a) Draw a Bayesian network that is a perfect map for P

Reasoning (referring to 7 independencies above by #):
- Because of #5,#7 there is no direct connection from B to D or A to D. Also, flow between B,D or A,D is blocked if C is unknown.
- There also must be no direct connection from A to C, since at least #1 would be false in this case.
- Since there is no independency info about {A,B}, {B,C}, {C,D}, we can assume an edge between these nodes.
- So the skeleton of the network is as follows:



- Since B ⊥D, but there is an edge from B to C, C to D, we know that BCD is v-structure
- To get the direction of A to B, we can look at #1, and #2, which are satisified by A → B
- So one P-map for these dependencies is:



-With a quick check, we can see that this satisfies all dependencies of the original list.
-But just to be sure, we should look at all possible indepencies, and make sure that this graph does not satisfy any which are not on the list.

All independencies:
[x] A ⊥B
[x] A ⊥B | C
[x] A ⊥B | D
[x] A ⊥B | C, D
[x] A ⊥C
[✓] A ⊥ C | B
[x] A ⊥C | D
[✓] A ⊥ C | B, D
[✓] A ⊥ D

[✓] A ⊥ D | B
[x] A ⊥D | C
[✓] A ⊥ D | B, C

[x] B ⊥C
[x] B ⊥C | A
[x] B ⊥C | D
[x] B ⊥C | A, D
[✓] B ⊥ D
[✓] B ⊥ D | A
[x] B ⊥D | C
[x] B ⊥D | A, C

[x] C ⊥D
[x] C ⊥D | A
[x] C ⊥D | B
[x] C ⊥D | A, B

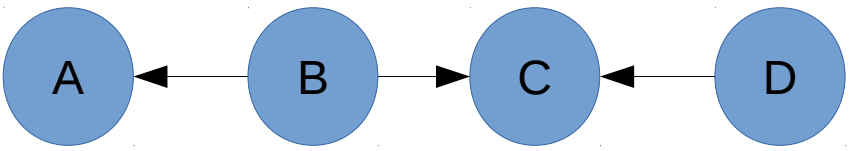
- Using the structure of the graph and going through the list, we can see that only dependencies in the original set are satisfied.

b) Does this perfect map have any I-equivalent graphs? If so, draw them. If not, explain why not.

First, we know that any I-equivalent graphs to this perfect map must have the same skeleton. So let's start by making a table with all possible arrow directions and analyze that.

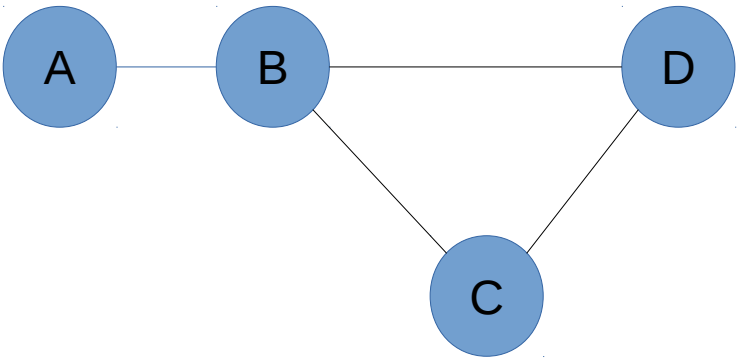
A,B	**B,C**	**C,D**	**Reasoning**
→	→	→	Violates #5,#7
→	→	←	Original
→	←	→	Violates #1
→	←	←	Violates #1
←	→	→	Violates #5
←	→	←	I-equivalent
←	←	→	Violates #5
←	←	←	Violates #5

This table shows us that there is one I-equivalent graph, which just has the edge A,B going the opposite way. We can confirm that this graph is truly I-equivalent by iterating through the long list above (it is).



c) Draw a Markov network that is a minimal I-map for P and explain why the Markov network is or is not also a perfect map.

First, we will draw a Markov network which is an I-map for P, which means that I(G) $\subseteq$I(P). To do this, we must moralize the graph, which in this case means marrying the parents of C by adding an edge from B to D:



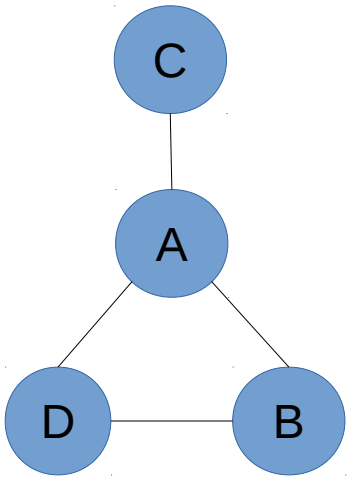
To Show that this I-map is minimal, we can remove each edge and see if any independences not in I(P) are created.
- Remove AB: A $\perp$B is created, not in I(P)
- Remove BC: B $\perp$C | D is created, not in I(P)
- Remove BD: B $\perp$D | C is created, not in I(P)
- Remove CD: C $\perp$D | B is created, not in I(P)

Since removing any edge in G violates I(G) $\subseteq$I(P), this I-map is minimal.

To check if it is perfect, we will look at the 7 independencies of I(P) to see if they are all covered. A quick check shows that 5,6,7 are not in I(G). For example B $\perp$D clearly is not since there is an edge BD.

Problem 2: I-Maps and P-Maps

a) Decide whether the following two independences are in I(P):



a	b	c	d	P(A=a, B=b, C=c, D=d)
0	0	0	0	1/8
1	1	0	0	1/8
1	1	1	0	1/4
0	1	0	1	1/4
1	0	1	1	1/4

To get probabilities for combinations of variables, we can marginalize out the others.

i. C ⊥D: Yes

P(C=0) = 1/8 + 1/8 + ¼ = ½
P(C=1) = ¼ + ¼ = ½

P(D=0) = 1/8 + 1/8 + ¼ = ½
P(D=1) = ¼ + ¼ = ½

P(C=0, D=0) = 1/8 + 1/8 = ¼
P(C=0, D=1) = ¼
P(C=1, D=0) = ¼
P(C=1, D=1) = ¼

P(C=0)*P(D=0) = ½ * ½ = ¼ = P(C=0, D=0)
P(C=0)*P(D=1) = ½ * ½ = ¼ = P(C=0, D=1)
P(C=1)*P(D=0) = ½ * ½ = ¼ = P(C=1, D=0)
P(C=1)*P(D=1) = ½ * ½ = ¼ = P(C=1, D=1)

Since P(C)*P(D) = P(C, D) for all c in C, d in D, C ⊥D is true.

ii. C ⊥B: No

P(C=0) = 1/8 + 1/8 + ¼ = ½
P(C=1) = ¼ + ¼ = ½

P(B=0) = ¼ + 1/8 = 3/8
P(B=1) = ¼ + ¼ + 1/8 = 5/8

P(C=0, B=0) = 1/8
P(C=0, B=1) = ¼ + 1/8 = 3/8
P(C=1, B=0) = ¼
P(C=1, B=1) = ¼

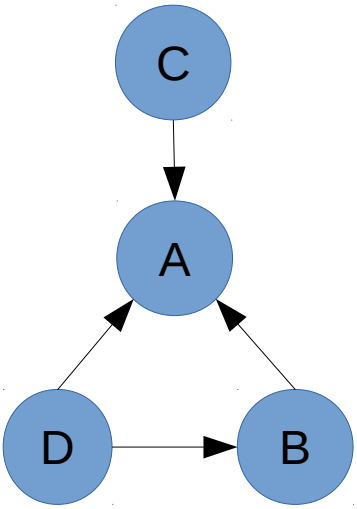
P(C=0)*P(B=0) = ½ * 3/8 = 3/16 ≠ P(C=0, B=0)
P(C=0)*P(B=1) = ½ * 5/8 = 5/16 ≠ P(C=0, B=1)
P(C=1)*P(B=0) = ½ * 3/8 = 3/16 ≠ P(C=1, B=0)
P(C=1)*P(B=1) = ½ * 5/8 = 5/16 ≠ P(C=1, B=1)

Since P(C)*P(B) ≠ P(C, B) for at least some c in C, b in B, C ⊥B is false.

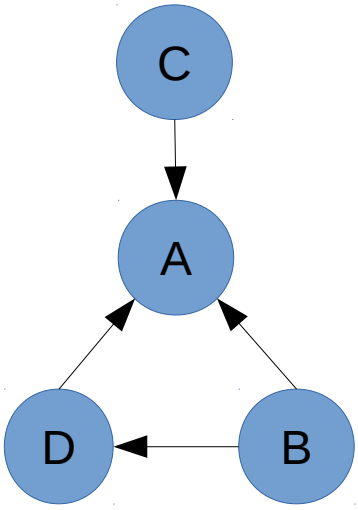
b) Give a direction to each individual edge in G, such that the resulting Bayesian network is a minimal I-map of P. Is your solution unique? Briefly state why.

Since, we know C ⊥D, we also know there are only 3 possible Bayesian networks following the given skeleton and meet that criterion. This is true because we know there must be a v-structure blocking both paths from C to D. The v-structure C → A ← D must be present for any possible network for it to be valid, or there would be an active path from C to D. There must also be a v-structure blocking the path from C to A to B to D. There are two valid networks in which C → A ← B is present, and one valid network in which A → B ← D is present.

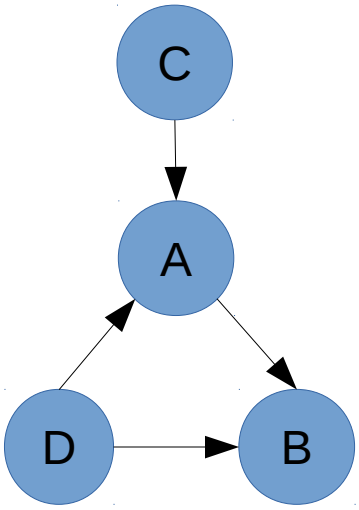
Network 1



Network 2



Network 3



However, we can see that two of these networks violate $C \perp B$ is false. Network 1 and 2 both give that $C \perp B$ is true. Therefore, only network 3 is valid.

In the table below, we analyze the network 3 to find I(G). We find that the structure of the network yields only two independencies:
1. $C \perp D$
2. $B \perp C \mid A, D$

We have already shown that $C \perp D$ is in I(P). To show that G is an I-map of P, we just need to show the $B \perp C \mid A, D$ is in I(P):

If $B \perp C \mid A, D$ then $P(B \mid A, D) * P(C \mid A, D) = P(A, B \mid C, D)$ for all values of A,B,C,D.

a	**b**	**d**	**P(B=b | A=a, D=d)**
0	0	0	1/8 / 1/8 = 1
0	0	1	0
0	1	0	0
0	1	1	¼ / ¼ = 1
1	0	0	0
1	0	1	¼ / ¼ = 1
1	1	0	3/8 / 3/8 = 1
1	1	1	0

a	**c**	**d**	**P(C=c | A=a, D=d)**
0	0	0	1/8 / 1/8 = 1
0	0	1	¼ / ¼ = 1
0	1	0	0
0	1	1	0
1	0	0	1/8 / 3/8 = 1/3
1	0	1	0
1	1	0	¼ / 3/8 = 2/3

1	1	1	¼ / ¼ = 1

a	**b**	**c**	**d**	**P(B=b | A=a, D=d) *P(C=c | A=a, D=d)**	**P(B=b, C=c | A=a, D=d)**
0	0	0	0	1 * 1	1/8 / ¼ = 1/2
0	0	0	1	0	0
0	0	1	0	1 *	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	1 *	¼ / ¼ = 1
0	1	1	0	0	0
0	1	1	1	1 *	0
1	0	0	0	0	0
1	0	0	1	1 *	0
1	0	1	0	0	0
1	0	1	1	1 *	¼ / ¼ = 1
1	1	0	0	1 *	1/8 / ¼ = 1/2
1	1	0	1	0	0
1	1	1	0	1 *	¼ / ¼ = 1
1	1	1	1	0	0

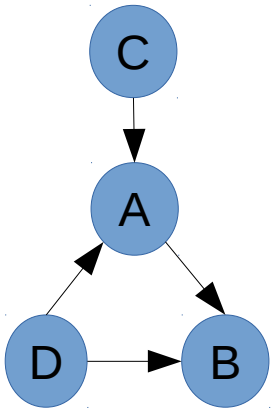
It is time-consuming to do these calculations by hand, so I wrote a python program to fill in the table:

Independency	**Network 3**	**I(P)**
A ⊥B	x	x
A ⊥B | C	x	x
A ⊥B | D	x	x
A ⊥B | C, D	x	x
A ⊥C	x	x
A ⊥C | B	x	x
A ⊥C | D	x	x
A ⊥C | B, D	x	✓
A ⊥D	x	x
A ⊥D | B	x	x
A ⊥D | C	x	x
A ⊥D | B, C	x	x

B ⊥C	x	x
B ⊥C | A	x	x
B ⊥C | D	x	x
B ⊥C | A, D	✓	✓
B ⊥D	x	x
B ⊥D | A	x	x
B ⊥D | C	x	x
B ⊥D | A, C	x	x
C ⊥D	✓	✓
C ⊥D | A	x	x
C ⊥D | B	x	x
C ⊥D | A, B	x	✓

As we can see, I(G) ⊆I(P), so this is indeed an I-map of P. To show that it is minimal, we will remove each edge one at a time and observe if any independences not in I(P) are added:
Remove AB: A ⊥B | D not in I(P)
Remove AC: A ⊥C not in I(P)
Remove AD: A ⊥D not in I(P)
Remove BD: B ⊥D | A not in I(P)

Since removing any edge will add an independency not in I(P), this is indeed a minimal I-map:



As for whether it is unique, we must check all 16 possible networks:

AB	**AC**	**AD**	**BD**	**Valid I-map**	**Reason**
→	→	→	→	X	B ⊥C | A
→	→	→	←	X	C ⊥D | A
→	→	←	→	X	Cycle
→	→	←	←	X	C ⊥D | A

→	←	→	→	X	B ⊥C | A
→	←	→	←	X	C ⊥D | A
→	←	←	→	X	Cycle
→	←	←	←	✓	Original
←	→	→	→	X	B ⊥C | A
←	→	→	←	X	Cycle
←	→	←	→	X	B ⊥C | A
←	→	←	←	X	B ⊥C | A
←	←	→	→	X	B ⊥C
←	←	→	←	X	Cycle
←	←	←	→	X	B ⊥C
←	←	←	←	X	B ⊥C

Since there is only one network with I(G) ⊆I(P), the solution is unique.
c. Give the CPDs for each node in your Bayesian network specified in (b).

These were also calculated using the python script below.

P(A, B, C, D) = P(C)*P(D)*P(A|C,D)*P(B|A,D)

P(C=0) = 0.5
P(C=1) = 0.5

P(D=0) = 0.5
P(D=1) = 0.5

P(A=0|C=0,D=0) = 0.5
P(A=0|C=0,D=1) = 1.0
P(A=0|C=1,D=0) = 0.0
P(A=0|C=1,D=1) = 0.0
P(A=1|C=0,D=0) = 0.5
P(A=1|C=0,D=1) = 0.0
P(A=1|C=1,D=0) = 1.0
P(A=1|C=1,D=1) = 1.0

P(B=0|A=0,D=0) = 1.0
P(B=0|A=0,D=1) = 0.0
P(B=0|A=1,D=0) = 0.0
P(B=0|A=1,D=1) = 1.0
P(B=1|A=0,D=0) = 0.0
P(B=1|A=0,D=1) = 1.0
P(B=1|A=1,D=0) = 1.0
P(B=1|A=1,D=1) = 0.0

Python script used for problem 2:

```
#!/usr/bin/env python

import itertools

enum = {
    0 : 'A',
    1 : 'B',
    2 : 'C',
    3 : 'D'
}

table = [
    ([0, 0, 0, 0], 1./8.),
    ([1, 1, 0, 0], 1./8.),
    ([1, 1, 1, 0], 1./4.),
    ([0, 1, 0, 1], 1./4.),
    ([1, 0, 1, 1], 1./4.)
]

vals = [0, 1]

def marg1(i, i_v):
    mp = 0.0
    for entry in table:
        if entry[0][i] == i_v:
            mp += entry[1]
    return mp

def marg2(i, i_v, j, j_v):
    mp = 0.0
    for entry in table:
        if entry[0][i] == i_v and entry[0][j] == j_v:
            mp += entry[1]
    return mp

def cond_marg1(i, i_v, k, k_v):
    cp = 0.0
    mp = 0.0
    for entry in table:
        if entry[0][k] == k_v:
            mp += entry[1]
            if entry[0][i] == i_v:
                cp += entry[1]
    if mp == 0.0:
        return 0.0
    else:
```

```
        return cp/mp

def cond_marg2(i, i_v, k, k_v, l, l_v):
    cp = 0.0
    mp = 0.0
    for entry in table:
        if entry[0][k] == k_v and entry[0][l] == l_v:
            mp += entry[1]
            if entry[0][i] == i_v:
                cp += entry[1]
    if mp == 0.0:
        return 0.0
    else:
        return cp/mp

def cond2_marg1(i, i_v, j, j_v, k, k_v):
    cp = 0.0
    mp = 0.0
    for entry in table:
        if entry[0][k] == k_v:
            mp += entry[1]
            if entry[0][i] == i_v and entry[0][j] == j_v:
                cp += entry[1]
    if mp == 0.0:
        return 0.0
    else:
        return cp/mp

def cond2_marg2(i, i_v, j, j_v, k, k_v, l, l_v):
    cp = 0.0
    mp = 0.0
    for entry in table:
        if entry[0][k] == k_v and entry[0][l] == l_v:
            mp += entry[1]
            if entry[0][i] == i_v and entry[0][j] == j_v:
                cp += entry[1]
    if mp == 0.0:
        return 0.0
    else:
        return cp/mp

def check_ind(i, j):
    for i_v in vals:
        mi = marg1(i, i_v)
        for j_v in vals:
            mj = marg1(j, j_v)
            mij = marg2(i, i_v, j, j_v)
            if mi*mj != mij:
                return False
```

```
    return True

def check_cond_ind1(i, j, k):
    for k_v in vals:
        for i_v in vals:
            mi = cond_marg1(i, i_v, k, k_v)
            for j_v in vals:
                mj = cond_marg1(j, j_v, k, k_v)
                mij = cond2_marg1(i, i_v, j, j_v, k, k_v)
                if mi*mj != mij:
                    return False
    return True

def check_cond_ind2(i, j, k, l):
    for l_v in vals:
        for k_v in vals:
            for i_v in vals:
                mi = cond_marg2(i, i_v, k, k_v, l, l_v)
                for j_v in vals:
                    mj = cond_marg2(j, j_v, k, k_v, l, l_v)
                    mij = cond2_marg2(i, i_v, j, j_v, k, k_v, l, l_v)
                    if mi*mj != mij:
                        return False
    return True

for v in vals:
    print "P(C={}) = {}".format(v, marg1(2, v))
print

for v in vals:
    print "P(D={}) = {}".format(v, marg1(3, v))
print

for v_i in vals:
    for v_j in vals:
        for v_k in vals:
            print "P(A={}|C={},D={}) = {}".format(v_i, v_j, v_k, cond_marg2(0, v_i, 2, v_j, 3, v_k))
print

for v_i in vals:
    for v_j in vals:
        for v_k in vals:
            print "P(B={}|A={},D={}) = {}".format(v_i, v_j, v_k, cond_marg2(1, v_i, 0, v_j, 3, v_k))

for i,j in itertools.combinations(range(4), 2):
    ind_ij = check_ind(i, j)
    print "{} i {}: {}".format(enum[i], enum[j], ind_ij)
    k,l = [e for e in range(4) if e != i and e != j]
    cond_ind_ij_k = check_cond_ind1(i, j, k)
```

```
print "{} i {} | {}: {}".format(enum[i], enum[j], enum[k], cond_ind_ij_k)
cond_ind_ij_l = check_cond_ind1(i, j, l)
print "{} i {} | {}: {}".format(enum[i], enum[j], enum[l], cond_ind_ij_l)
cond_ind_ij_kl = check_cond_ind2(i, j, k, l)
print "{} i {} | {}, {}: {}".format(enum[i], enum[j], enum[k], enum[l], cond_ind_ij_kl)
```

3. The Restricted Boltzmann Machine

a. $$P(h_i = 1 \mid v) = \frac{1}{\sum_{h_i \in \{0,1\}} \exp(-B_i h_i - v^T W_{:i})}$$

$$= \frac{1}{1 + \exp(-B_i - v^T W_{:i})}, \quad P(h_i = 0 \mid v) = 1 - P(h_i = 1 \mid v)$$

b. Since the graph is bipartite, the hidden units are independent from each other given the visible units (and vice versa). This means $p(h \mid v)$ can be factored as:

$$P(h \mid v) = \prod_{i=1}^{n} P(h_i \mid v)$$

c.

$$\sum_h \exp(\phi(v,h)) = \sum_h \exp(-a^T v - B^T h - v^T W h)$$

$$= \sum_{h_1,\ldots,h_n} \exp\left(-a^T v + \sum_{i=1}^{n} -B_i h_i - v^T W_{\cdot i} h_i\right)$$

$$= -na^T v \cdot \sum_{h_1,\ldots,h_n} \prod_{i=1}^{n} \exp(-B_i h_i - v^T W_{\cdot i} h_i)$$

$$= -na^T v \cdot \prod_{i=1}^{n} \sum_{h_i \in \{0,1\}} \exp(-B_i h_i - v^T W_{\cdot i} h_i)$$

$$= -na^T v \prod_{i=1}^{n} (1 + \exp(-B_i - v^T W_{\cdot i}))$$

So yes, this can be computed in time not exponential in n.
Time complexity $= O(nm)$

d. This should be symmetrical to part c, with final formula as:

$$\sum_v \exp(\phi(v,h)) = -mB^T h \cdot \prod_{i=1}^{m} (1 + \exp(-a_i - W_{i\cdot} h))$$

Also yes, runtime $= O(mn)$

e. Combining these 2 formulas, we get:

$$Z = \sum_h \sum_v \exp(\phi(v,h)) = nmB^T h a^T v \prod_{i=1}^{n} \prod_{j=1}^{m} (1 + \exp(-W_{ij}))$$

Runtime $= O(nm)$, so yes, not exponential

4. Maximum Likelihood Learning of Bayes Nets

$$l_G(\theta; D) = \sum_{i=1}^{n} \sum_{u \in Val(Pa(X_i))} \sum_{x_i} M[x_i, u] \log \theta_{x_i|u}$$

$$\theta^{ML}_{x_i|u} = \frac{M[x_i, u]}{M[u]}$$

Show that $\max_{\theta} l_{G'}(\theta'; D) \geq \max_{\theta} l_G(\theta; D)$

$$M[u_i] = M[U_i, X_i]$$

$$\sum_{x_i} M[u_i] = \sum_{x_i} M[U_i, X_i] = M[U_i]$$

Now use Jensen's inequality in the form

$$f\left(\frac{\sum a_i x_i}{\sum a_i}\right) \leq \left(\frac{\sum a_i f(x_i)}{\sum a_i}\right)$$

$$-\log\left(\frac{M[u_n, x_n]}{M[x_n, u_n, x_n]}\right) \leq \frac{M[x_n, u_n]}{M[x_n, u_n, x_n]}\left(-\log\left(\frac{M[u_n]}{M[x_n, u_n]}\right)\right)$$

Therefore, the fit of G' to the data is strictly improved from G, since the inequality is satisfied.

Problem 5: Programming Assignment

a) Implement a NB classifier.

Naive Bayes
  10-fold cross validation total test accuracy 0.9181 on 232 examples

b) Implement a TANB classifier.

TANB Classifier
  10-fold cross validation total test accuracy 0.9526 on 232 examples

c)

i. What is the marginal probability this Congressman is a Democrat (C=1)?

To handle missing fields in the entry, I marginalized over all possible values for those missing fields. To do this, I made a function called gen_missing, which recursively generates all 2^n entries, where n is the number of missing fields. To marginalize, I sum the conditional probability over all these entries.

Naive Bayes Classifier on missing data
  P(C=1|A_observed) = 0.9827
TANB Classifier on missing data
  P(C=1|A_observed) = 0.9899

ii. Can you predict how this Congressman voted on education spending (A_{12})?

To predict a missing field, I marginalize over A_i given all other variables (C, and other A_i).

Naive Bayes A12 for missing congressman
  P(C=0|A_observed) = 0.8688
TANB A12 for missing congressman
  P(C=0|A_observed) = 0.8977

d)

i. What is the test error when you train both classifiers on a smaller subset of the training data?

Naive Bayes
  10-fold cross validation total test accuracy 0.9009 on subset
TANB Classifier
  10-fold cross validation total test accuracy 0.8922 on subset

ii. Explain why the test error for the TANB may not strictly be better than NB.
The test error for TANB may not be strictly better than NB because there may not be enough data to compute a sufficient model with TANB. For all cases in which there are no values for a given conditional probability estimate, and Laplace smoothing is used, the model is biased towards the uniform for those conditionals and its accuracy is reduced.