CS486 Assignment 3

CHEN YIFEI

20754459

Part A:

Q1:The construction of factor and the function needed are in the file ‘factor.py’:

```
from math import log10,floor
class Factor(object):
    # width during formatting
    FORMAT_WIDTH = 10
    # round up to roundTo decimal places
    ROUNDTO = 3

    #  round up to n significant figures
    @staticmethod
    def significant(x):
        return round(x, -int(floor(log10(x))-(Factor.ROUNDTO-1)))

    def __init__(self, varList, varValues, mappedValues):
        self.factorTable = dict()
        self.mappedValues = mappedValues
        for i in xrange(0,len(varList)):
            self.factorTable[varList[i]] = varValues[i]

    # Restrict this factor on a variable with a particular value
    def restrict(self, variable, value):
        # search all the positions where value of indicated variable satisfy the parameter
'value'
        index = [i for i, val in enumerate(self.factorTable[variable]) if val == value]
```

```
        # firstly we take out the 'column' from the factorTable to continue restrict operation
        del self.factorTable[variable]

        # then we do the restriction operation
        variables = self.factorTable.keys()
        restrictVars = map(lambda var: [val for i, val in enumerate(self.factorTable[var]) if i in
index], variables)
        self.mappedValues = [val for i, val in enumerate(self.mappedValues) if i in index]
        for i in xrange(0, len(variables)):
            self.factorTable[variables[i]] = restrictVars[i]

    # compute the multiplication of two factors
    @staticmethod
    def multiply(factorA, factorB):
        # get the variables list of each factor
        factorA_Vars = factorA.factorTable.keys()
        factorB_Vars = factorB.factorTable.keys()
        # figure out the variables which are common for these two factors and assign them
into newVars firstly
        newVars = [var for var in factorA_Vars if var in factorB_Vars]

        # select the rows in factorB that has to be multiplied with the rows in factorA
        multiply_A = [[] for i in xrange(len(factorA.mappedValues))]
        numMappedValues = 0
        for i in xrange(len(multiply_A)):
            for j in xrange(len(factorB.mappedValues)):
                sameCommonValues = True
                # we only multiply the tuple with the same values of common variables
                for var in newVars:
                    if factorA.factorTable[var][i] != factorB.factorTable[var][j]:
                        sameCommonValues = False
                        break
                # now the Common Variables should have the same values
```

```
                if sameCommonValues:
                    multiply_A[i].append(j)
                    numMappedValues+= 1

        # now the newVars should contain the union of the two factors
        newVars.extend([var for var in factorA_Vars if var not in newVars])
        newVars.extend([var for var in factorB_Vars if var not in newVars])

        newValues = [[] for i in xrange(len(newVars))]
        multiplyValues = []

        tuples = tuple(enumerate(newVars))

        # for each row in factorA
        for row in xrange(len(multiply_A)):
            # for each rows in factorB to be multiplied with a row in in factorA
            for row_in_factorB in multiply_A[row]:
                for i, var in tuples:
                    # assign values of these variables row by row
                    newValues[i].append(factorA.factorTable[var][row]       if       var       in
factorA_Vars \
                                                    else factorB.factorTable[var][row_in_factorB])
                multiplyValues.append(Factor.significant(factorA.mappedValues[row]      *
factorB.mappedValues[row_in_factorB]))

        return Factor(newVars, newValues, multiplyValues)

    # sum a variable in a factor
    def sumout(self, variable):
        # notice that we should do nothing for a factor with only one variable
        if len(self.factorTable.keys()) == 1:
            return
```

```
# get the domain of the variable
domainOfVar = set(self.factorTable[variable])

# delete the column of this variable from table entirely
del self.factorTable[variable]

row_to_sum = []
allOfVars = self.factorTable.keys()
for i in xrange(len(self.mappedValues)):
    if i in row_to_sum:
        # if i already found should be sum, continue to next iteration
        continue

    limitOfMatch = len(self.mappedValues) / len(domainOfVar)

    for j in xrange(i+1, len(self.mappedValues)):
        # check if the remaining values of vars of two row are the same
        sameValues = True
        for var in allOfVars:
            if self.factorTable[var][i] != self.factorTable[var][j]:
                sameValues = False

        if sameValues and j not in row_to_sum:
            row_to_sum.append(j)
            limitOfMatch-= 1

        # if we already found pair of  matches up to limitOfMatch
        if limitOfMatch == 0:
            break

newValues = [[] for i in xrange(len(allOfVars))]
# create a new list of mapped values with length
sumoutValues = []
```

```
        lists = list(enumerate(allOfVars))
        sumRowIndex = 0
        for i in xrange(len(self.mappedValues)):

            if i in row_to_sum:
                continue

            for w, var in lists:
                newValues[w].append(self.factorTable[var][i])

            # add the rows as a sum of a kind of variable values
            sum = self.mappedValues[i]
            for j in xrange(0, len(domainOfVar)-1):
                sum+= self.mappedValues[row_to_sum[sumRowIndex]]
                sumRowIndex+= 1

            sumoutValues.append(sum)

        for j, var in lists:
            # clear former values
            del self.factorTable[var][:]
            self.factorTable[var].extend(newValues[j])

        # clear old mappedValues
        del self.mappedValues[:]
        self.mappedValues.extend(sumoutValues)



    def normalize(self):
        totalMappedValues = sum(self.mappedValues)
        self.mappedValues[:] = map(lambda x: Factor.significant(x/totalMappedValues),
self.mappedValues)
```

```
    @staticmethod
    def inference(factorList, queryVariables, orderedListOfHiddenVariables, evidenceList):
        # figure out the variables which are in the query and evidence from orderedListOfHiddenVariables
        step = 0
        if len(evidenceList) > 0:
            evidenceOfVars = zip(*evidenceList)[0]
        else:
            evidenceOfVars = []
        orderedListOfHiddenVariables = filter(lambda x: x not in queryVariables \
                                              and x not in evidenceOfVars, orderedListOfHiddenVariables)

        for evidence in evidenceList:
            for factor in factorList:
                # if the evidence variable is in this factor, we should restrict them
                if evidence[0] in factor.factorTable.keys():
                    factor.restrict(evidence[0], evidence[1])

        evidences = reduce(lambda x,y: '{!s}, {!s}'.format(x,y), evidenceList) if len(evidenceList) > 0 else 'nothing'
        print ("step ",step)
        step += 1
        Factor.textFormattedPrint('Restricting factors on evidence {!s}'.format(evidences), 50, factorList)

        for hiddenVariable in orderedListOfHiddenVariables:
            factorsWithHiddenVar = []
            for factor in factorList:
```

```
            if hiddenVariable in factor.factorTable.keys():
                factorsWithHiddenVar.append(factor)

        # take out all the factors with a hidden variable from the list
        factorList = filter(lambda x: x not in factorsWithHiddenVar, factorList)

        # compute product of hidden variables
        if len(factorsWithHiddenVar) > 0:
            print ("step ",step)
            step += 1
            Factor.textFormattedPrint('Summing over {!s}'.format(hiddenVariable), 30,
[])
            multiplyFactor = reduce(Factor.multiply, factorsWithHiddenVar)
        else:

            continue

        print ("step ",step)
        step += 1
        Factor.textFormattedPrint('Intermediate multiply factor on
{!s}'.format(hiddenVariable),30,[multiplyFactor])

        # do the sumout operation among the hidden variable on the product factor
        print ("step ",step)
        step += 1
        multiplyFactor.sumout(hiddenVariable)
        Factor.textFormattedPrint('Summing out variable
{!s}'.format(hiddenVariable),30,[multiplyFactor])

        # insert the result back to the list
        factorList.append(multiplyFactor)

    # now the only issue we have is about factors with only the query variable, we
```

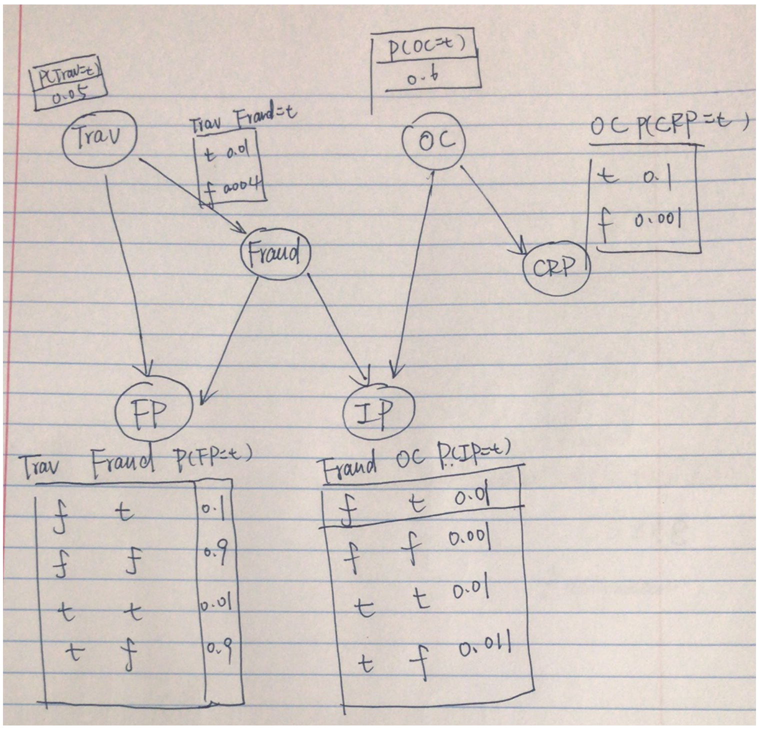
```
product them together
        inferredFactor = reduce(Factor.multiply, factorList)
        print ("step ",step)
        step += 1
        Factor.textFormattedPrint('multiplying factors with query variables: {!s}'.\
                                  format(queryVariables),45,[inferredFactor])

        # normalize stage
        inferredFactor.normalize()
        print ("step ",step)
        step += 1
        Factor.textFormattedPrint('Finally normalize the result',30,[inferredFactor])




    def copy(self):
        valuesOfKey = self.factorTable.items()
        return Factor(map(lambda x: x[0], valuesOfKey), map(lambda x: x[1], valuesOfKey),
self.mappedValues)

    # delete opertation which take out the 'column' in matrix
    def __del__(self):
        del self.mappedValues
        self.factorTable.clear()
        del self.factorTable

    # used to get the string form of this factor
    def __str__(self):
        formatting = ''
        variables = self.factorTable.keys()
        # create a format string to use
        for i in xrange(0, len(variables)+1):
```

```
            formatting+= '{!s:{width}}'
            if i == len(variables):
                formatting+= '\r\n'

        # create the formatted header
        returnValue = formatting.format('MappedVal', *variables, width=Factor.FORMAT_WIDTH)

        zippedValues = zip(self.mappedValues, *map(lambda x: self.factorTable[x], variables))

        # for each value create the formatted row
        for values in zippedValues:
            returnValue+= formatting.format(*values, width=Factor.FORMAT_WIDTH)

        return returnValue

    @staticmethod
    def textFormattedPrint(statement, Num, factors):
        print statement, '\r'
        print '+' * Num, '\r'
        for factor in factors:
            print factor, '\r'

        print '+' * Num, '\r'
```

Q2:The construction of factor and the function needed are

in the file ‘factor.py’ :

(a)



(b) the code :

```
from factor import Factor

def main():
    hiddenVariables = ['Trav','FP','Fraud','IP','OC','CRP']
    factorTrav = Factor(['Trav'], [['t','f']], [0.05,0.95])
    factorFraudTrav = Factor(['Fraud','Trav'],[['t','t','f','f'],['t','f','t','f']],[0.01,0.004,0.99,0.996])
    factorFPFraudTrav = Factor(['FP','Fraud','Trav'],[['t','t','t','t','f','f','f','f'],
                                                      ['t','t','f','f','t','t','f','f'],
                                                      ['t','f','t','f','t','f','t','f']],
                                                      [0.9,0.1,0.9,0.01,0.1,0.9,0.1,0.99])
    factorIPFraudOC = Factor(['IP','Fraud','OC'],[['t','t','t','t','f','f','f','f'],
                                                  ['t','t','f','f','t','t','f','f'],
                                                  ['t','f','t','f','t','f','t','f']],
                                                  [0.02,0.011,0.01,0.001,0.98,0.989,0.99,0.999])
    factorCRPOC = Factor(['CRP','OC'],[['t','t','f','f'],['t','f','t','f']],[0.1,0.001,0.9,0.999])
    factorOC = Factor(['OC'],[['t','f']],[0.6,0.4])


    copyFraudTrav = factorFraudTrav.copy()
    copyIPFraudOC = factorIPFraudOC.copy()
    copyOC = factorOC.copy()
    copyTrav = factorTrav.copy()
    copyCRPOC = factorCRPOC.copy()
    copyFPFraudTrav = factorFPFraudTrav.copy()


    Factor.inference([factorFraudTrav,factorTrav],['Fraud'],hiddenVariables,[])
    Factor.inference([copyTrav, copyFraudTrav, copyFPFraudTrav, copyIPFraudOC, copyOC, copyCRPOC],
                     ['Fraud'],hiddenVariables,[['FP','t'],['CRP','t'],['IP','f']])

main()
```

now we can see the steps of two inference from the terminal:

```
[v1022-rn-76-14:a3partA chenyifei$ python q2b.py
('step ', 0)
Restricting factors on evidence nothing
++++++++++++++++++++++++++++++++++++++++++++++++++
MappedVal Fraud     Trav
0.01      t         t
0.004     t         f
0.99      f         t
0.996     f         f

MappedVal Trav
0.05      t
0.95      f

++++++++++++++++++++++++++++++++++++++++++++++++++
('step ', 1)
Summing over Trav
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++
('step ', 2)
Intermediate multiply factor on Trav
++++++++++++++++++++++++++++++
MappedVal Fraud     Trav
0.0005    t         t
0.0038    t         f
0.0495    f         t
0.946     f         f

++++++++++++++++++++++++++++++
('step ', 3)
Summing out variable Trav
++++++++++++++++++++++++++++++
MappedVal Fraud
0.0043    t
0.9955    f

++++++++++++++++++++++++++++++
('step ', 4)
multiplying factors with query variables: ['Fraud']
+++++++++++++++++++++++++++++++++++++++++++++
MappedVal Fraud
0.0043    t
0.9955    f

+++++++++++++++++++++++++++++++++++++++++++++
('step ', 5)
Finally normalize the result
++++++++++++++++++++++++++++++
MappedVal Fraud
0.0043    t
0.996     f
```

```
++++++++++++++++++++++++++++++
('step ', 0)
Restricting factors on evidence ['FP', 't'], ['CRP', 't'], ['IP', 'f']
++++++++++++++++++++++++++++++++++++++++++++++++++
MappedVal Trav
0.05      t
0.95      f

MappedVal Fraud     Trav
0.01      t         t
0.004     t         f
0.99      f         t
0.996     f         f

MappedVal Fraud     Trav
0.9       t         t
0.1       t         f
0.9       f         t
0.01      f         f

MappedVal Fraud     OC
0.98      t         t
0.989     t         f
0.99      f         t
0.999     f         f

MappedVal OC
0.6       t
0.4       f

MappedVal OC
0.1       t
0.001     f

++++++++++++++++++++++++++++++++++++++++++++++++++
('step ', 1)
Summing over Trav
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++
('step ', 2)
Intermediate multiply factor on Trav
++++++++++++++++++++++++++++++
MappedVal Fraud     Trav
0.00045   t         t
0.0446    f         t
0.00038   t         f
0.00946   f         f

++++++++++++++++++++++++++++++
('step ', 3)
Summing out variable Trav
++++++++++++++++++++++++++++++
MappedVal Fraud
0.00083   t
0.05406   f

++++++++++++++++++++++++++++++
('step ', 4)
Summing over OC
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++
('step ', 5)
Intermediate multiply factor on OC
++++++++++++++++++++++++++++++
MappedVal Fraud     OC
0.0588    t         t
0.000396  t         f
0.0594    f         t
0.0004    f         f
```

```
++++++++++++++++++++++++++++++
('step ', 6)
Summing out variable OC
++++++++++++++++++++++++++++++
MappedVal Fraud
0.059196  t
0.0598    f

++++++++++++++++++++++++++++++
('step ', 7)
multiplying factors with query variables: ['Fraud']
+++++++++++++++++++++++++++++++++++++++++++++
MappedVal Fraud
4.91e-05  t
0.00323   f

+++++++++++++++++++++++++++++++++++++++++++++
('step ', 8)
Finally normalize the result
++++++++++++++++++++++++++++++
MappedVal Fraud
0.015     t
0.985     f
```

(c) The code:

```python
from factor import Factor

def main():
    hiddenVariables = ['Trav','FP','Fraud','IP','OC','CRP']
    factorTrav = Factor(['Trav'], [['t','f']], [0.05,0.95])
    factorFraudTrav = Factor(['Fraud','Trav'],[['t','t','f','f'],['t','f','t','f']],[0.01,0.004,0.99,0.996])
    factorFPFraudTrav = Factor(['FP','Fraud','Trav'],[['t','t','t','t','f','f','f','f'],
                                                      ['t','t','f','f','t','t','f','f'],
                                                      ['t','f','t','f','t','f','t','f']],
                                                      [0.9,0.1,0.9,0.01,0.1,0.9,0.1,0.99])
    factorIPFraudOC = Factor(['IP','Fraud','OC'],[['t','t','t','t','f','f','f','f'],
                                                  ['t','t','f','f','t','t','f','f'],
                                                  ['t','f','t','f','t','f','t','f']],
                                                  [0.02,0.011,0.01,0.001,0.98,0.989,0.99,0.999])
    factorCRPOC = Factor(['CRP','OC'],[['t','t','f','f'],['t','f','t','f']],[0.1,0.001,0.9,0.999])
    factorOC = Factor(['OC'],[['t','f']],[0.6,0.4])

    Factor.inference([factorTrav,factorFraudTrav,factorFPFraudTrav,factorIPFraudOC,factorOC,factorCRPOC],
                    ['Fraud'],hiddenVariables,[['FP','t'],['CRP','t'],['IP','f'],['Trav','t']])

main()
```

steps:

```
v1022-rn-76-14:a3partA chenyifei$ python q2c.py
('step ', 0)
Restricting factors on evidence ['FP', 't'], ['CRP', 't'], ['IP', 'f'], ['Trav',
 't']
++++++++++++++++++++++++++++++++++++++++++++++++++
MappedVal
0.05

MappedVal Fraud
0.01      t
0.99      f

MappedVal Fraud
0.9       t
0.9       f

MappedVal Fraud     OC
0.98      t         t
0.989     t         f
0.99      f         t
0.999     f         f

MappedVal OC
0.6       t
0.4       f

MappedVal OC
0.1       t
0.001     f

++++++++++++++++++++++++++++++++++++++++++++++++++
('step ', 1)
Summing over OC
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++
('step ', 2)
Intermediate multiply factor on OC
++++++++++++++++++++++++++++++
MappedVal Fraud     OC
0.0588    t         t
0.000396  t         f
0.0594    f         t
0.0004    f         f

++++++++++++++++++++++++++++++
('step ', 3)
Summing out variable OC
++++++++++++++++++++++++++++++
MappedVal Fraud
0.059196  t
0.0598    f

++++++++++++++++++++++++++++++
('step ', 4)
multiplying factors with query variables: ['Fraud']
+++++++++++++++++++++++++++++++++++++++++++++
MappedVal Fraud
2.66e-05  t
0.00267   f

+++++++++++++++++++++++++++++++++++++++++++++
('step ', 5)
Finally normalize the result
++++++++++++++++++++++++++++++
MappedVal Fraud
0.00986   t
0.99      f

++++++++++++++++++++++++++++++
```

(d) code:

```python
from factor import Factor

def main():
    hiddenVariables = ['Trav','FP','Fraud','IP','OC','CRP']
    factorTrav = Factor(['Trav'], [['t','f']], [0.05,0.95])
    factorFraudTrav = Factor(['Fraud','Trav'],[['t','t','f','f'],['t','f','t','f']],[0.01,0.004,0.99,0.996])
    factorFPFraudTrav = Factor(['FP','Fraud','Trav'],[['t','t','t','t','f','f','f','f'],
                                                      ['t','t','f','f','t','t','f','f'],
                                                      ['t','f','t','f','t','f','t','f']],
                                                      [0.9,0.1,0.9,0.01,0.1,0.9,0.1,0.99])
    factorIPFraudOC = Factor(['IP','Fraud','OC'],[['t','t','t','t','f','f','f','f'],
                                                  ['t','t','f','f','t','t','f','f'],
                                                  ['t','f','t','f','t','f','t','f']],
                                                  [0.02,0.011,0.01,0.001,0.98,0.989,0.99,0.999])
    factorCRPOC = Factor(['CRP','OC'],[['t','t','f','f'],['t','f','t','f']],[0.1,0.001,0.9,0.999])
    factorOC = Factor(['OC'],[['t','f']],[0.6,0.4])


    copyIPFraudOC = factorIPFraudOC.copy()
    copyFraudTrav = factorFraudTrav.copy()
    copyCRPOC = factorCRPOC.copy()
    copyOC = factorOC.copy()
    copyTrav = factorTrav.copy()
    copyFPFraudTrav = factorFPFraudTrav.copy()

    Factor.inference([copyTrav,copyFraudTrav,copyFPFraudTrav,copyIPFraudOC,copyOC,copyCRPOC],
                     ['Fraud'],hiddenVariables,[['IP','t']])
    Factor.inference([factorTrav,factorFraudTrav,factorFPFraudTrav,factorIPFraudOC,factorOC,factorCRPOC],
                     ['Fraud'],hiddenVariables,[['IP','t'],['CRP','t']])


main()
```

steps:

```
v1022-rn-76-14:a3partA chenyifei$ python q2d.py
('step ', 0)
Restricting factors on evidence ['IP', 't']
++++++++++++++++++++++++++++++++++++++++++++++++++++
MappedVal Trav
0.05      t
0.95      f

MappedVal Fraud     Trav
0.01      t         t
0.004     t         f
0.99      f         t
0.996     f         f

MappedVal FP        Fraud     Trav
0.9       t         t         t
0.1       t         t         f
0.9       t         f         t
0.01      t         f         f
0.1       f         t         t
0.9       f         t         f
0.1       f         f         t
0.99      f         f         f

MappedVal Fraud     OC
0.02      t         t
0.011     t         f
0.01      f         t
0.001     f         f

MappedVal OC
0.6       t
0.4       f

MappedVal CRP       OC
0.1       t         t
0.001     t         f
0.9       f         t
0.999     f         f

++++++++++++++++++++++++++++++++++++++++++++++++++++
('step ', 1)
Summing over Trav
+++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++
('step ', 2)
Intermediate multiply factor on Trav
+++++++++++++++++++++++++++++++
MappedVal FP        Fraud     Trav
0.00045   t         t         t
5e-05     f         t         t
0.0446    t         f         t
0.00495   f         f         t
0.00038   t         t         f
0.00342   f         t         f
0.00946   t         f         f
0.937     f         f         f

+++++++++++++++++++++++++++++++
('step ', 3)
Summing out variable Trav
+++++++++++++++++++++++++++++++
MappedVal FP        Fraud
0.00083   t         t
0.00347   f         t
0.05406   t         f
0.94195   f         f

+++++++++++++++++++++++++++++++
('step ', 4)
Summing over FP
+++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++
('step ', 5)
Intermediate multiply factor on FP
+++++++++++++++++++++++++++++++
MappedVal FP        Fraud
0.00083   t         t
0.00347   f         t
0.05406   t         f
0.94195   f         f

+++++++++++++++++++++++++++++++
('step ', 6)
Summing out variable FP
+++++++++++++++++++++++++++++++
MappedVal Fraud
0.0043    t
0.99601   f

+++++++++++++++++++++++++++++++
('step ', 7)
Summing over OC
+++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++
('step ', 8)
Intermediate multiply factor on OC
+++++++++++++++++++++++++++++++
MappedVal CRP       Fraud     OC
0.0012    t         t         t
0.0108    f         t         t
4.4e-06   t         t         f
0.0044    f         t         f
0.0006    t         f         t
0.0054    f         f         t
4e-07     t         f         f
0.0004    f         f         f

+++++++++++++++++++++++++++++++
('step ', 9)
Summing out variable OC
+++++++++++++++++++++++++++++++
MappedVal CRP       Fraud
0.0012044 t         t
0.0152    f         t
0.0006004 t         f
0.0058    f         f

+++++++++++++++++++++++++++++++
('step ', 10)
Summing over CRP
+++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++
('step ', 11)
Intermediate multiply factor on CRP
+++++++++++++++++++++++++++++++
MappedVal CRP       Fraud
0.0012044 t         t
0.0152    f         t
0.0006004 t         f
0.0058    f         f

+++++++++++++++++++++++++++++++
('step ', 12)
Summing out variable CRP
+++++++++++++++++++++++++++++++
MappedVal Fraud
0.0164044 t
0.0064004 f

+++++++++++++++++++++++++++++++++++++++++++++
('step ', 13)
multiplying factors with query variables: ['Fraud']
+++++++++++++++++++++++++++++++++++++++++++++
MappedVal Fraud
7.05e-05  t
0.00637   f

+++++++++++++++++++++++++++++++++++++++++++++
('step ', 14)
Finally normalize the result
+++++++++++++++++++++++++++++++
MappedVal Fraud
0.0109    t
0.989     f
```

PartB Code：

preprocess.py

preprocess.py 〉 No Selection

```python
from sets import Set

# the function to produce the matrix from the content of the files
# which has the same functionality with load.m script in matlab

def setMatrix(data, label, numFeatures):
    Matrix = []
    #construct the matrix with the size [numArticle][numFeatures + 1 ]
    #and assign the label value of each article is in the last column
    with open(label, 'r') as f:
        docId = 0
        for line in f:
            Matrix.append([0] * (numFeatures + 1))
            Matrix[docId][-1] = int(line)
            docId+= 1

    #read the data file and record the words existence of each article in the matrix
    with open(data, 'r') as f:
        for line in f:
            lineSplit = line.split()
            fileID = int(lineSplit[0])
            featureNum = int(lineSplit[1])
            Matrix[fileID-1][featureNum-1] = 1

    return Matrix

# obtain the arrays of numfeature and exactly words in correspond order
def readWords(haveDone):
    numFeatures = 0
    features = []
    words = []
    if haveDone:
        featuresSet = Set(featuresSet)

    with open('words.txt', 'r') as f:
        for line in f:
            if haveDone:
                features.add(numFeatures)
            else:
                features.append(numFeatures)

            numFeatures = numFeatures + 1
            wordsSet.append(line.rstrip())

    return (numFeatures, features, words)
```

naiveBayes.py

naiveBayes.py 〉 __init__

```python
from math import log, fabs
from __future__ import division
import heapq

class NaiveBayes(object):
    #  labbel
    ALTATHEISM = 1
    COMPGRAPHICS = 2

    def __init__(self, featuresSet, matrix):
        # get groups of Atheism and graphics separately
        Atheism = filter(lambda doc: doc[-1] == NaiveBayesLearner.ALTATHEISM, matrix)
        Graphics = filter(lambda doc: doc[-1] == NaiveBayesLearner.COMPGRAPHICS, matrix)

        # compute the prior probability
        self.priorAtheism = len(Atheism)/len(matrix)
        self.priorGraphics = len(Graphics)/len(matrix)

        # for each feature calculate the likelihood

        self.likelihoods = [((reduce(lambda x,y: (1 if y[feature] else 0) + x, Atheism, 0) + 1)/(len(Atheism) + 2),\
                            (reduce(lambda x,y: (1 if y[feature] else 0) + x, Graphics, 0) + 1)/(len(Graphics) + 2))\
                                for feature in featuresSet]

    def predict(self, doc):
        posteriorAtheism = log(self.priorAtheism)

        posteriorGraphics = log(self.priorGraphics)


        for i in xrange(0,len(self.likelihoods)):
            if doc[i]:
                posteriorAtheism += log(self.likelihoods[i][0])
            else:
                posteriorAtheism -= log(1-self.likelihoods[i][0])

            if doc[i]:
                posteriorGraphics += log(self.likelihoods[i][1])
            else:
                posteriorGraphics -= log(1-self.likelihoods[i][1])

        # now we return the result with higher ML
        return NaiveBayesLearner.ALTATHEISM if posteriorAtheism >= posteriorGraphics else NaiveBayesLearner.COMPGRAPHICS

    # for question 2
    def DiscriminativeWords(self, wordsSet):
        array = []
        # compute the absolute value of difference of likelihoods
        for i in xrange(0,len(self.likelihoods)):
            difference = fabs(log(self.likelihoods[i][0], 2) - log(self.likelihoods[i][1], 2))
            array.append((i,difference))
        # sort the result
        array.sort(lambda x,y: cmp(x[1],y[1]), None, True)
        for i in xrange(0, 10):
            print ("word: %s  discrimination: %d" %(wordsSet[res[i][0]], res[i][1]))
```

main.py

main.py 〉 No Selection

```python
from __future__ import division
from naiveBayes import NaiveBayes
from preprocess import setMatrix, readWords

# implement the prediction
# and check the correction rate
def getAccuracy(learner, matrix):
    numCorrectPredictions = 0
    for rows in matrix:

        if learner.predict(rows) == rows[-1]:
            numCorrectPredictions+= 1

    return numCorrectPredictions/len(matrix)

def main():
    numFeatures, featuresSet, wordsSet = readWords(False)
    trainMatrix = setMatrix('trainData.txt', 'trainLabel.txt', numFeatures)
    testMatrix = setMatrix('testData.txt', 'testLabel.txt', numFeatures)


    naiveLearner = NaiveBayes (featuresSet, trainMatrix)
    # get the accuracy for train and text file
    trainAccuracy = getAccuracy(naiveLearner, trainMatrix)
    testAccuracy = getAccuracy(naiveLearner, testMatrix)

    print ("trainAccuracy: %s  testAccuracy: %d" %(trainAccuracy,testAccuracy))

    naiveLearner.DiscriminativeWords(wordsSet)

main()
```

print out of code :

preprocess:

```python
from sets import Set

# the function to produce the matrix from the content of the files

# which has the same functionality with load.m script in matlab



def setMatrix(data, label, numFeatures):

    Matrix = []

    #construct the matrix with the size [numArticle][numFeatures + 1 ]

    #and assign the label value of each article is in the last column

    with open(label, 'r') as f:

        docId = 0

        for line in f:

            Matrix.append([0] * (numFeatures + 1))

            Matrix[docId][-1] = int(line)

            docId+= 1



    #read the data file and record the words existence of each article in the matrix

    with open(data, 'r') as f:
```

```
        for line in f:
            lineSplit = line.split()
            fileID = int(lineSplit[0])
            featureNum = int(lineSplit[1])
            Matrix[fileID-1][featureNum-1] = 1

    return Matrix

# obtain the arrays of numfeature and exactly words in correspond order
def readWords(haveDone):
    numFeatures = 0
    features = []
    words = []
    if haveDone:
        featuresSet = Set(featuresSet)

    with open('words.txt', 'r') as f:
        for line in f:
            if haveDone:
                features.add(numFeatures)
            else:
                features.append(numFeatures)

            numFeatures = numFeatures + 1
            wordsSet.append(line.rstrip())

    return (numFeatures, features, words)
```

Naïvebayes

```
from math import log, fabs

from __future__ import division

import heapq


class NaiveBayes(object):

    #   labbel

    ALTATHEISM = 1

    COMPGRAPHICS = 2

    def __init__(self, featuresSet, matrix):

        # get groups of Atheism and graphics separately

        Atheism = filter(lambda doc: doc[-1] == NaiveBayesLearner.ALTATHEISM, matrix)

        Graphics = filter(lambda doc: doc[-1] == NaiveBayesLearner.COMPGRAPHICS, matrix)


        # compute the prior probability

        self.priorAtheism = len(Atheism)/len(matrix)

        self.priorGraphics = len(Graphics)/len(matrix)

        # for each feature calculate the likelihood


        self.likelihoods = [((reduce(lambda x,y: (1 if y[feature] else 0) + x, Atheism, 0) + 1)/(len(Atheism) + 2),\

                             (reduce(lambda x,y: (1 if y[feature] else 0) + x, Graphics, 0) + 1)/(len(Graphics) + 2))\

                                 for feature in featuresSet]

    def predict(self, doc):

        posteriorAtheism = log(self.priorAtheism)

        posteriorGraphics = log(self.priorGraphics)


        for i in xrange(0,len(self.likelihoods)):

            if doc[i]:

                posteriorAtheism += log(self.likelihoods[i][0])

            else:

                posteriorAtheism -= log(1-self.likelihoods[i][0])

            if doc[i]:

                posteriorGraphics += log(self.likelihoods[i][1])
```

```
            else:
                posteriorGraphics -= log(1-self.likelihoods[i][1])
        # now we return the result with higher ML
        return NaiveBayesLearner.ALTATHEISM if posteriorAtheism >= posteriorGraphics else
NaiveBayesLearner.COMPGRAPHICS
    # for question 2
    def DiscriminativeWords(self, wordsSet):
        array = []
        # compute the absolute value of difference of likelihoods
        for i in xrange(0,len(self.likelihoods)):
            difference = fabs(log(self.likelihoods[i][0], 2) - log(self.likelihoods[i][1], 2))
            array.append((i,difference))
        # sort the result
        array.sort(lambda x,y: cmp(x[1],y[1]), None, True)
        for i in xrange(0, 10):
            print ("word: %s    discrimination: %d" %(wordsSet[res[i][0]], res[i][1]))
```

main:

```
from __future__ import division
from naiveBayes import NaiveBayes
from preprocess import setMatrix, readWords


# implement the prediction
# and check the correction rate
def getAccuracy(learner, matrix):
    numCorrectPredictions = 0
    for rows in matrix:


        if learner.predict(rows) == rows[-1]:
            numCorrectPredictions+= 1


    return numCorrectPredictions/len(matrix)
```

```
def main():

    numFeatures, featuresSet, wordsSet = readWords(False)

    trainMatrix = setMatrix('trainData.txt', 'trainLabel.txt', numFeatures)

    testMatrix = setMatrix('testData.txt', 'testLabel.txt', numFeatures)

    naiveLearner = NaiveBayes (featuresSet, trainMatrix)

    # get the accuracy for train and text file

    trainAccuracy = getAccuracy(naiveLearner, trainMatrix)

    testAccuracy = getAccuracy(naiveLearner, testMatrix)


    print ("trainAccuracy: %s    testAccuracy: %d" %(trainAccuracy,testAccuracy))

    naiveLearner.DiscriminativeWords(wordsSet)

main
```

Question b

```
trainAccuracy: 0.92837  testAccuracy: 0.88967
word: graphics  discrimination: 6.38374874578
word: atheism  discrimination: 5.73389435561
word: religion  discrimination: 5.66678015975
word: moral  discrimination: 5.55986495583
word: keith  discrimination: 5.55986495583
word: evidence  discrimination: 5.55986495583
word: atheists  discrimination: 5.52239025041
word: god  discrimination: 5.45768456103
word: bible  discrimination: 5.40374575392
word: christian  discrimination: 5.36192557822
```

Question c

This assumption might be reasonable in some situations where the variables are not correlated firmly, and calculating the accurate result can be hard, therefore the choice of naïve Bayesian is reasonable.

However, we do production directly for conditional probabilities, which is not accurate. we saw that in this part of assignment the algorithm is not that good for estimation when faced with test data.

Question d

Suppose we have a dataset $\mathbf{e}$, where the i^{th} data assign a values $\mathbf{x}_i$ to observed variables $\mathbf{X}$, leaving $\mathbf{Z}$ latent variables (with values $\mathbf{z}_j$) unassigned, then

Recall: Bayes Net Maximum Likelihood (Complete data - $Z = \{\}$)

$$\theta_{V=true,pa(V)=\mathbf{v}} = \frac{\text{number in } \mathbf{e} \text{ with } (V = true \wedge pa(V) = \mathbf{v})}{\text{number in } \mathbf{e} \text{ with } pa(V) = \mathbf{v}}$$

Now: Bayes Net Expectation Maximization (incomplete data)

Start with some guess for θ,

E Step: Compute weights for each data $\mathbf{x}_i$ and latent variable(s) value(s) $\mathbf{z}_j$ (using e.g. variable elimination)

$$w_{ij} = P(\mathbf{z}_j|\theta, \mathbf{x}_i)$$

M Step: Update parameters:

$$\theta_{V=true,pa(V)=\mathbf{v}} = \frac{\sum_{ij} w_{ij}|V = true \wedge pa(V) = \mathbf{v} \;\; \text{in} \;\; \{\mathbf{x}_i, \mathbf{z}_j\}}{\sum_{ij} w_{ij}|pa(V) = \mathbf{v} \;\; \text{in} \;\; \{\mathbf{x}_i, \mathbf{z}_j\}}$$