CSC 421 Assignment 2

Cameron Long
V00748439

November 9th, 2015

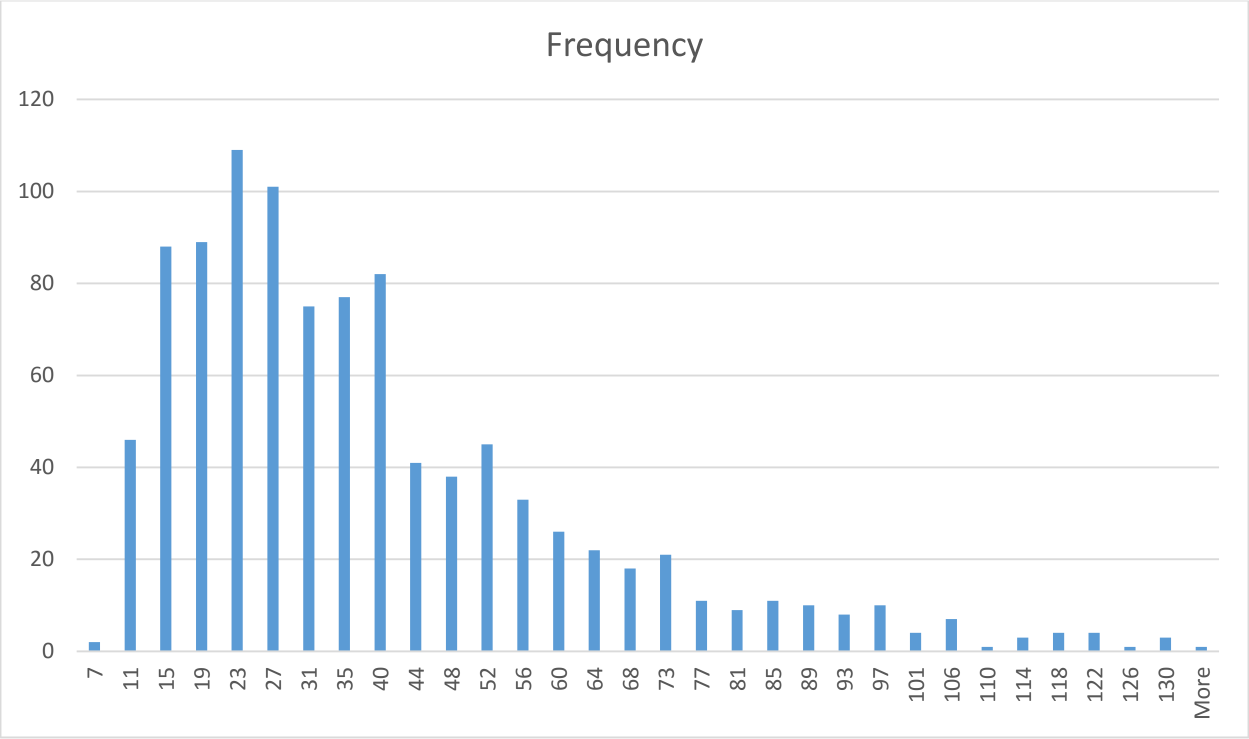
Part 1 – Probabilistic Simulation

To calculate a cumulative distribution you must first consider the probabilities of each event happening in an atomic fashion. For instance with our dice rolling, each number has a 1/6 (0.1666) chance of happening. So our cumulative distribution would look like this, after adding the individual probabilities

Dice Number	Cumulative Probability
1	0.166
2	0.333
3	0.500
4	0.666
5	0.833
6	1.000

So, we must map this distribution to a random number generator to take our discrete samples of this distribution. We can generate such a number fairly easily programmatically between 0 and 1, and map it to each bucket to represent each "dice roll" outcome.

When this principle is applied to a game such as chutes and ladders, we can simulate a number of "games" and record the amount of rolls it takes to complete the game. When the simulation as ran one thousand times, the following results were acquired.

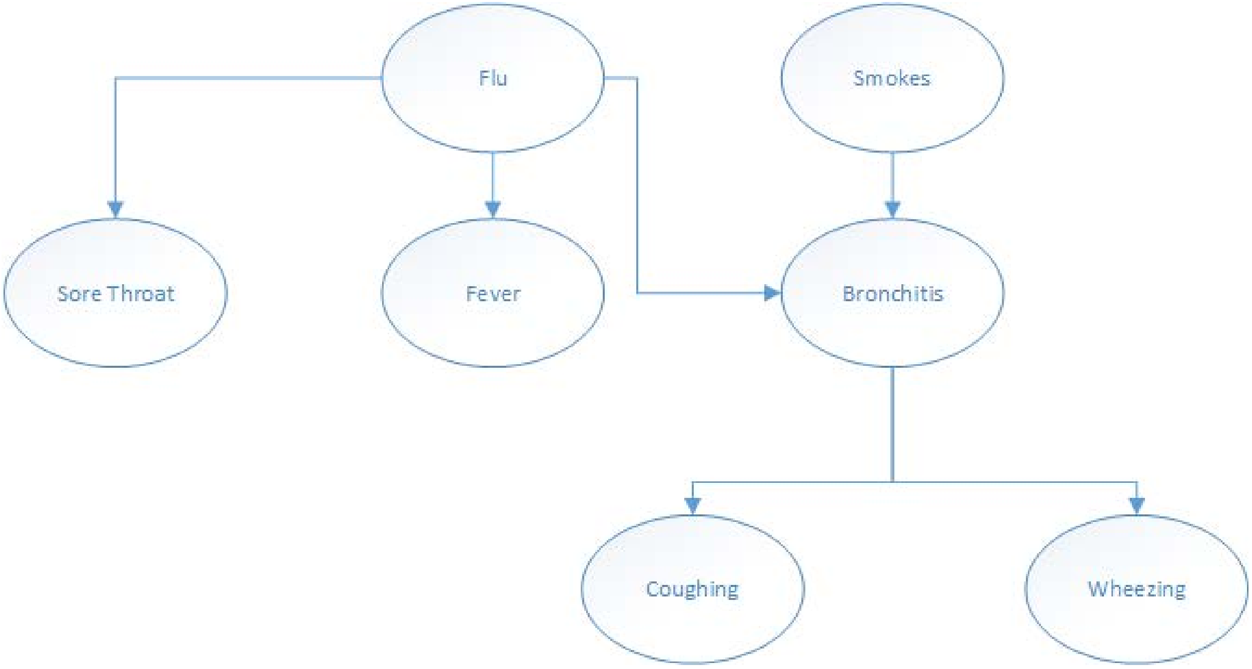


Average	Standard Deviation
37.415	23.574

This graph represents the number of rolls needed to complete the game over a thousand games. The minimum number of rolls was experimentally show to be 7 rolls, with one player needing over 130 rolls to complete the game. Different board configurations could also be used to change these values.

Part 2 – Bayesian Belief Networks

Consider the following Belief Network:



With Probability Tables:

	True	False
Flu	0.1	0.90
Smokes	0.2	0.80

	Sore Throat	
Flu	**True**	**False**
T	0.85	0.15
F	0.30	0.70
	Fever	
T	0.95	0.05
F	0.20	0.80

Bronchitis			
Flu	**Smokes**	**True**	**False**
F	**F**	0.005	0.995
F	**T**	0.50	0.5
T	**F**	0.30	0.7
T	**T**	0.90	0.1

Coughing		
Bronchitis	**True**	**False**
T	0.90	0.10
F	0.50	0.50
Wheezing		
T	0.95	0.05
F	0.30	0.70

To obtain an estimate of the probability of having a fever if you have the flu, you would have to take samples of people with the flu, and record whether or not they have a fever. Or, every time you get sick, record the same data. But that could take years to obtain enough samples to create reliable data.

$P(fl, st, fe, \neg br, \neg sm, co, \neg wh)$ can be calculated by taking numbers directly from these tables

Such as:

$$P(fl, st, fe, \neg br, \neg sm, co, \neg wh) = 0.10 * 0.80 * 0.95 * 0.85 * 0.70 * 0.50 * 0.70 = 0.015827$$

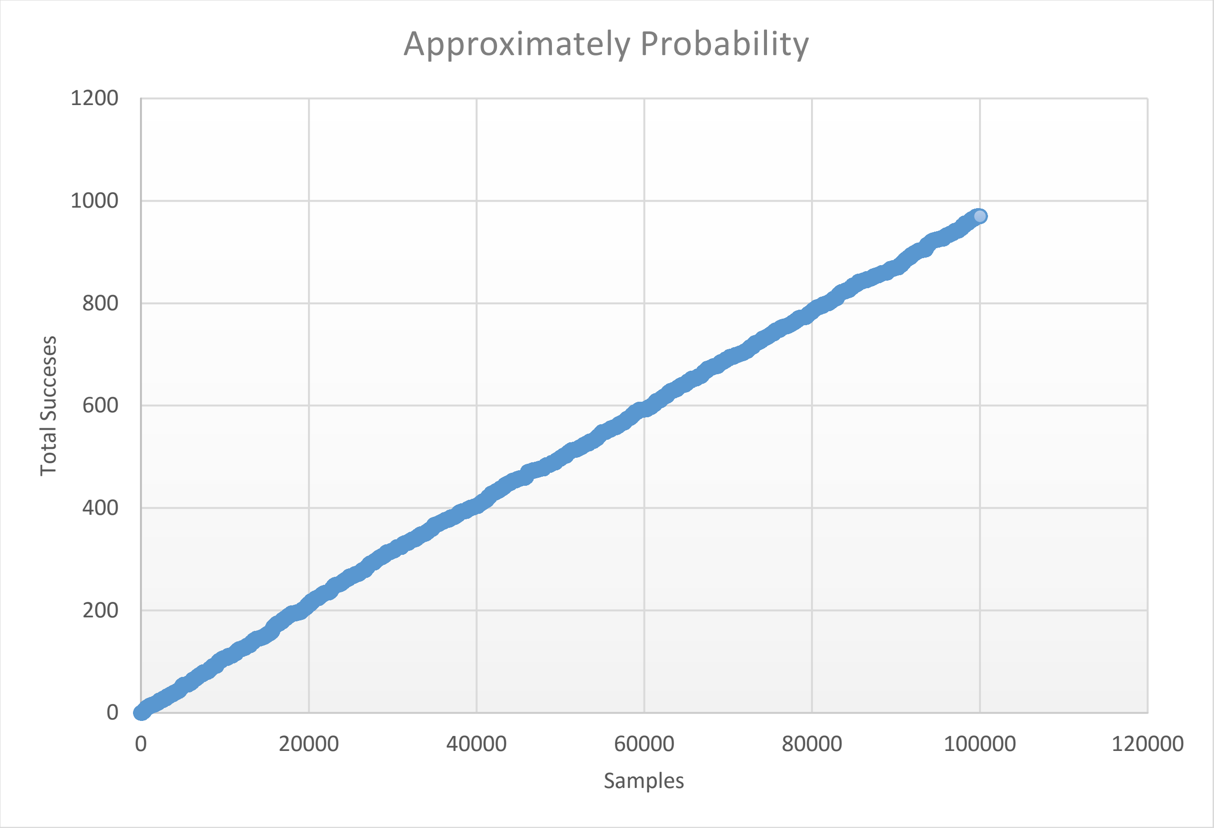
Similarly, $P(fl|co, \neg wh)$ can be calculated with some formula manipulation.

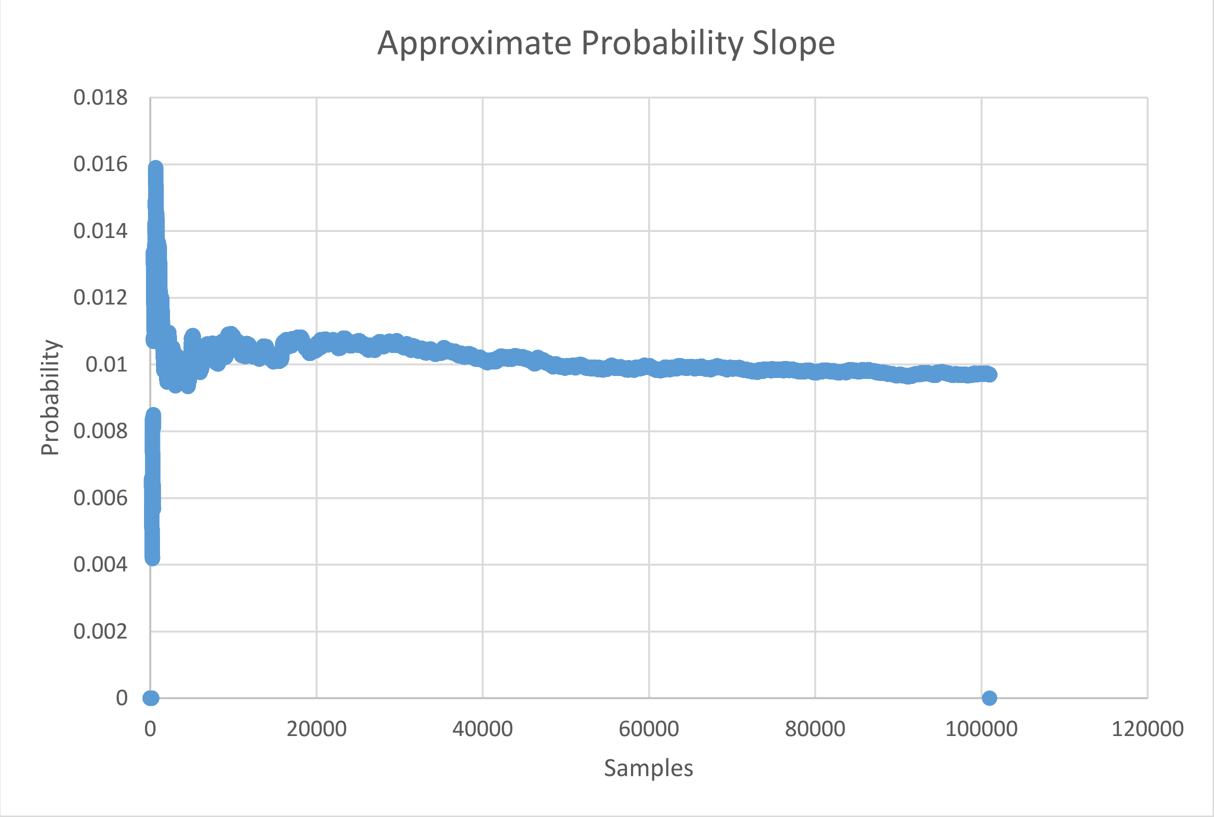
$$\frac{P(co, \neg wh\ |fl)P(fl)}{P(\neg wh, co)}$$

$$\frac{P(co, \neg wh\ |br)P(fl)}{P(\neg wh, co)}$$

$$\frac{0.05*0.90*0.10}{0.05*0.90} = 0.10$$

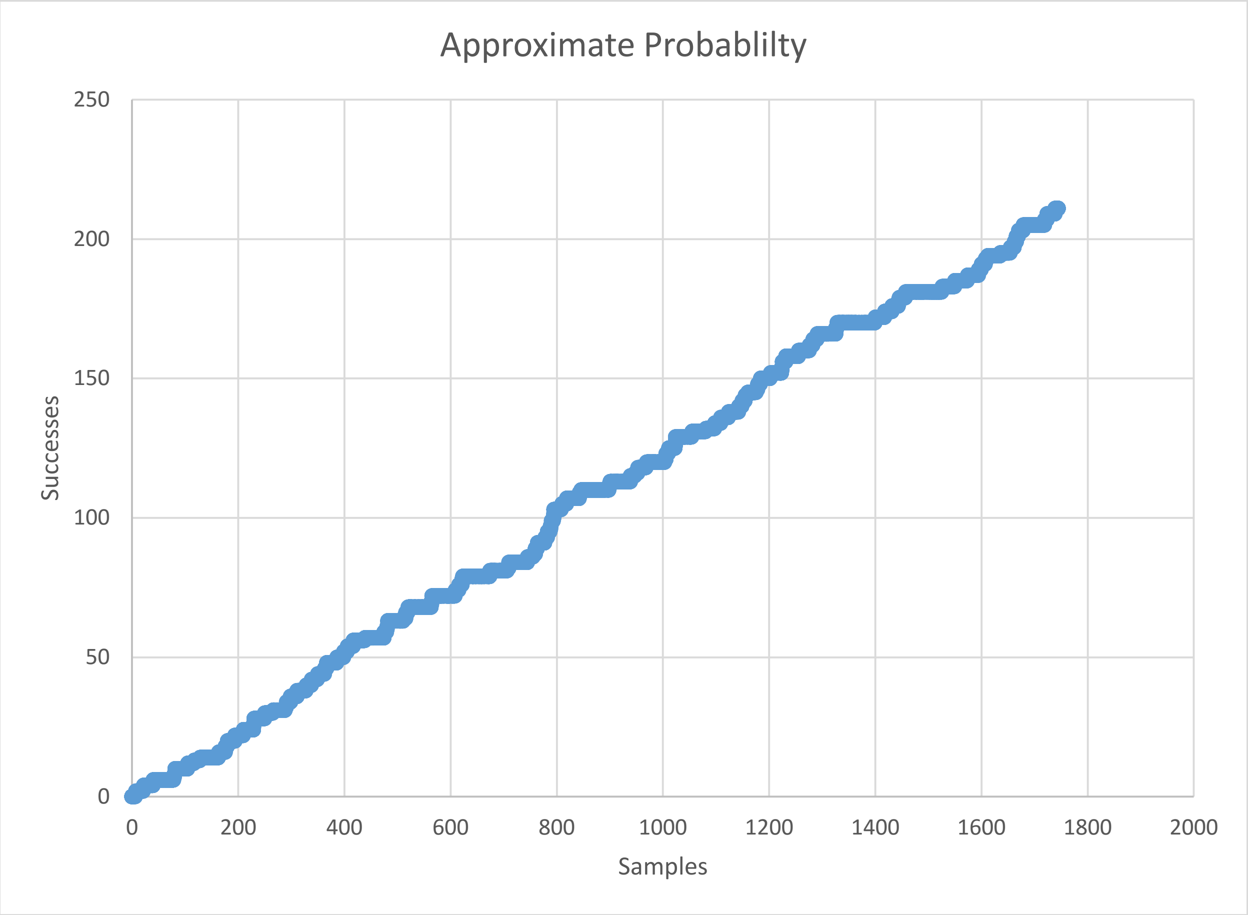
We can also simulate these results programmatically. This chart is a representation of 100000 samples of the Bayesian network for $P(fl, st, fe, \neg br, \neg sm, co, \neg wh)$

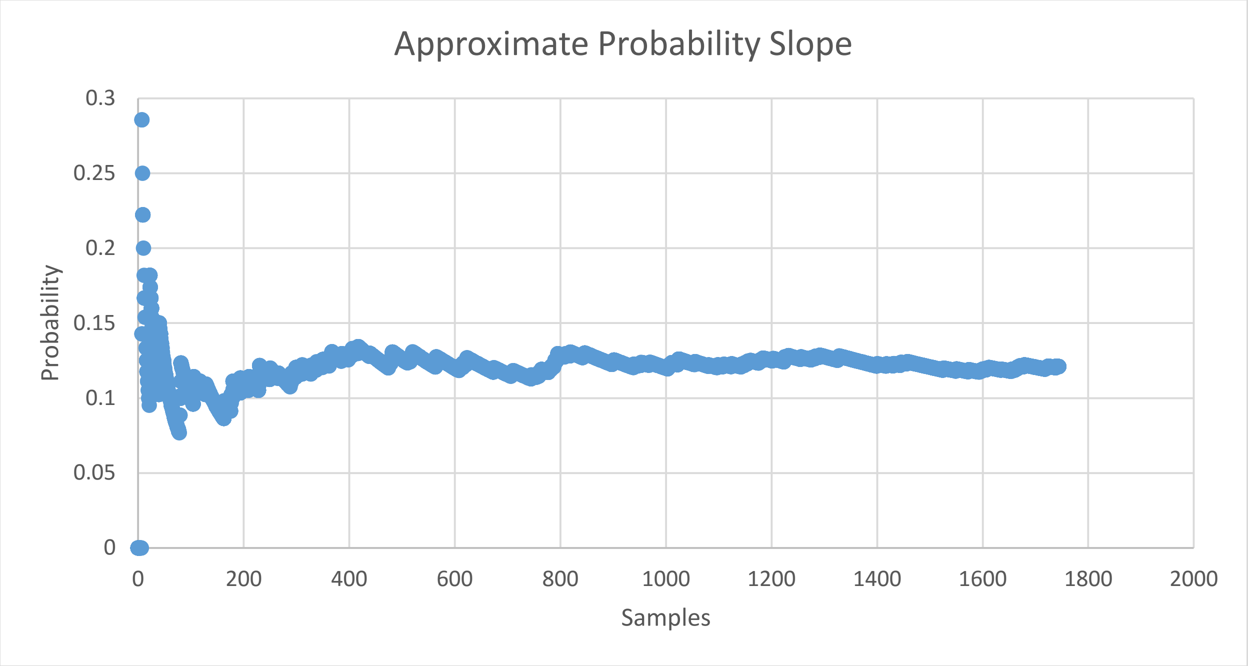

Approximately Probability
Total Succeses
1200
1000
800
600
400
200
0
0
20000
40000
60000
80000
100000
120000
Samples


Approximate Probability Slope
Probability
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
20000
40000
60000
80000
100000
120000
Samples

Which gave us a total success rate of about 0.097.

1000 samples were drawn from the probability distribution of $P(fl|co, \neg wh)$, where 1744 samples were accepted, with 211 successes, giving us an approximate probability of 0.121.





Part 3 – Movie Categorization Using Naïve Bayes

A python script was written to calculate these values. They were determined to be:

	Awful	Bad	Boring	Dull	Effective	Enjoyable	Great	Hilarious
Negative	0.122	0.545	0.175	0.101	0.086	0.054	0.32	0.059
Positive	0.034	0.28	0.054	0.025	0.154	0.096	0.485	0.132

We can construct a confusion matrix as follows:

True Positives Actual positive reviews classified as such **464**	**False Positives** Negative reviews classified as positive reviews **161**
False Negatives Positive Reviews classified as negative **536**	**True Negatives** Negative Reviews classified as such **839**

The probabilities were used to calculate posterior probabilities for each classification. Each review had a weighting, and the one with the highest weight was picked (positive or negative). However, this model could be improved upon, as the classifier has a real hard time picking positive reviews correctly. We could add several mores words, or combinations of them.

From the README file, the accuracy would probably be close to one-hundred percent of the samples taken when using an explicit numeration system rated out of ten. However, this severely limits the amount of reviews from this data set that we can classify as only a few reviews use this system.

True Positives Actual positive reviews classified as such **24**	**False Positives** Negative reviews classified as positive reviews **0**
False Negatives Positive Reviews classified as negative **0**	**True Negatives** Negative Reviews classified as such **23**

To create "random" movie reviews, all you would have to do is take normalized samples of the calculated probability tables we have calculated in previous sections. We could get more complicated, and use the same principles for the words that make up entire sentences. We could create tables to sample from for verbs, adjectives and nouns, and follow a simplistic model of English sentence structure to create these "reviews" You would just have to take samples from these tables and string it all together. However, the more types of sentences you wish to produce will add a large amount of complexity to your Bayesian network.

Below are some examples of an extremely simple "review" program.

Negative	Positive
The movie was real dull The actors were bad The screenplay is very enjoyable Overall it was boring	"The movie was real boring The actors were great The screenplay is very dull Overall it was enjoyable"
The movie was real bad The actors were enjoyable The screenplay is very dull Overall it was enjoyable	"The movie was real dull The actors were enjoyable The screenplay is very enjoyable Overall it was dull"
The movie was real boring	"The movie was real boring

The actors were enjoyable The screenplay is very boring Overall it was boring	The actors were awful The screenplay is very dull Overall it was enjoyable”
The movie was real awful The actors were boring The screenplay is very dull Overall it was enjoyable	“The movie was real effective The actors were great The screenplay is very effective Overall it was great”
Overall it was boring The movie was real bad The actors were effective The screenplay is very bad	The movie was real enjoyable The actors were enjoyable The screenplay is very enjoyable Overall it was dull

The program is a bit erratic in terms of its reviewing, sometimes it will produce a review similar to “The movie was boring, the screenplay was dull, and the actors were awful, but man did I enjoy that movie” and classifies the movie as positive. This model could use a more robust dictionary or perhaps some pre-direction on which type of review to generate.

Part 4 – Source Code

Part 1

This code loads a board from a file, and creates an object with that board configuration. It then runs the board once through until the player has reached the goal and outputs the statistics to a file. The file was called 10000 times via a Windows batch script.

```
import random

class GameBoard:

    def __init__(self, lengthx, lengthy, shoots, ladders):
        self.lengthx = lengthx
        self.lengthy = lengthy
        self.shoots = shoots
        self.ladders = ladders
        self.playerpos = 0


def initializeboard():

    f = open('C:\Users\Fisk\Desktop\chutes.txt', 'r')

    print f
    tempshoots =[]
    for line in f:
        a = line.splitlines()
        b = a[0].split(' ')
        c = [int(e) for e in b]
        tempshoots.append(c)

    #shoots = [map(int, x) for x in tempshoots]

    #shoots = [int(e) for e in tempshoots]
    print tempshoots

    f.close()
    f = open('C:\Users\Fisk\Desktop\ladders.txt', 'r')

    templadders = []

    for line in f:
        a = line.splitlines()
        b = a[0].split(' ')
        c = [int(e) for e in b]
        templadders.append(c)
    print templadders

    board = GameBoard(10,10, tempshoots, templadders)

    playgame(board)
```

```
def playgame(board):

    dice = [1,2,3,4,5,6]
    iterations = 0;
    f = open('C:\Users\Fisk\Desktop\stats.txt', 'a')

    while (board.playerpos < 100):
        roll = random.choice(dice)
        board.playerpos += roll
        print ("PLAYER HAS ROLLED: " + str(roll))

        for shoot in board.shoots:
            if board.playerpos == shoot[0]:
                board.playerpos = shoot[1]
                print "HIT A CHUTE"
                break
        for ladder in board.ladders:
             if board.playerpos == ladder[0]:
                board.playerpos = ladder[1]
                print "HIT A LADDER"
                break

        print board.playerpos
        iterations += 1

    f.write(str(iterations))
    f.write('\n')

def main():
    initializeboard()


if __name__ == "__main__":
    main()
```

Part 2

This code takes the Belief network values and takes random samples from that network. The results of those samples and whether or not it was a successful draw is outputted to a text file for processing in excel

```
import random

samples = 0
success = 0

def drawrandom():
    global samples
    global success
    flu = True
    smokes = True
    sore = True
    fever = True
    bron = True
    cough = True
    wheeze = True
    numberslist = []

    for num in range(0,6):
        numberslist.append(random.random())

    #set samples

    if numberslist[0] >0.10:
        flu = False
    if numberslist[1] > 0.20:
        smokes = False
    #second round
    if numberslist[2] > 0.85 and flu == True:
        sore = False
    if numberslist[2] > 0.30 and flu == False:
        sore = False
    if numberslist[3] > 0.95 and flu == True:
        fever = False
    if numberslist[3] > 0.20 and flu == False:
        fever = False
    if numberslist[4] > 0.005 and flu == False and smokes == False:
        bron = False
    if numberslist[4] > 0.5 and flu == False and smokes == True:
        bron = False
    if numberslist[4] > 0.3 and flu == True and smokes == False:
        bron = False
    if numberslist[4] > 0.9 and flu == False and smokes == True:
        bron = False
    # third round
    if numberslist[5] > 0.5 and bron == False:
        cough = False
    if numberslist[5] > 0.9 and bron == True:
        cough = False
    if numberslist[5] > 0.3 and bron == False:
        wheeze = False
    if numberslist[5] > 0.95 and bron == True:
        wheeze = False

    variableslist = []
    variableslist.append([flu, smokes, sore, fever,bron, cough,wheeze])

    print (variableslist)
    print (numberslist)
    f = open("C:\\Users\\Fisk\\Desktop\\part2.txt", 'a')

    if flu == True and sore == True and fever == True and bron == False and
smokes == False and cough == True and wheeze == False:
        success = success + 1
        print ("One success")

        f.write(str(samples) + ' ' + str(success) + '\n')


    samples = samples + 1
    f.write(str(samples) + ' ' + str(success) + '\n')
    f.close()

def main():

    for num in range(0,100000):
        drawrandom()
    print ("Finished")

if __name__ == "__main__":
    main()
```

Part 3

This code will parse all the text files from the downloaded source. It calculates the probabilities of word occurrence and generates a list of the review vectors. We can then perform stats and samples from this list of vectors for the other parts of part 3.

```python
import glob, os


def findNegative():
    wordlist = [0, 0 ,0 ,0, 0 ,0 ,0, 0]
    masterlist = []
    os.chdir("C:\\Users\\Cameron\\Desktop\\421\\txt_sentoken\\neg")
    print(wordlist)
    for file in glob.glob("*.txt"):
        f = open(file, 'r')
        review = f.read()

        if "awful" in review:
            wordlist[0] = 1
            #print("Contains awful: ")
        if "bad" in review:
            wordlist[1] = 1
            #print("Contains bad: ")
        if "boring" in review:
            wordlist[2] = 1
            #print("Contains boring: ")
        if "dull" in review:
            wordlist[3] = 1
            #print("Contains dull: ")
        if "effective" in review:
            wordlist[4] = 1
            #print("Contains effective: ")
        if "enjoyable" in review:
            wordlist[5] = 1
            #print("Contains enjoyable: ")
        if "great" in review:
            wordlist[6] = 1
            #print("Contains great: ")
        if "hilarious" in review:
            wordlist[7] = 1
            #print("Contains hilarious: ")
        #print("___________________________________________________")
        #print(wordlist)
        masterlist.append(wordlist)
        wordlist = [0, 0 ,0 ,0, 0 ,0 ,0, 0]
        #print(review)

        #print(file)
        #print(f)

        f.close()
    seperateData(masterlist)
    accuracyTest(masterlist)
def seperateData(masterlist):
    awful = 0
    bad = 0
    boring = 0
    dull = 0
    effective =0
    enjoyable =0
    great = 0
    hilarious =0


    for review in masterlist:
        if review[0] == 1:
            awful += 1
        if review[1] == 1:
            bad += 1
        if review[2] == 1:
            boring +=1
        if review[3] == 1:
            dull +=1
        if review[4] == 1:
            effective += 1
        if review[5] == 1:
            enjoyable +=1
        if review[6] == 1:
            great +=1
        if review[7] == 1:
            hilarious += 1

    length = len(masterlist)

    probs = [awful/length, bad/length, boring/length, dull/length,
effective/length, enjoyable/length, great/length, hilarious/length]

    print(probs)


def findPositive():
    wordlist = [0, 0 ,0 ,0, 0 ,0 ,0, 0]
    masterlist = []
    reviews = 0
    os.chdir("C:\\Users\\Cameron\\Desktop\\421\\txt_sentoken\\pos")
    for file in glob.glob("*.txt"):
        f = open(file, 'r')
        review = f.read()
        if "/10" in review:

            print("Found number review")
            print(file)
            reviews += 1
        if "awful" in review:
            wordlist[0] = 1
            #print("Contains awful: ")
        if "bad" in review:
            wordlist[1] = 1
            #print("Contains bad: ")
        if "boring" in review:
            wordlist[2] = 1
            #print("Contains boring: ")
        if "dull" in review:
            wordlist[3] = 1
            #print("Contains dull: ")
        if "effective" in review:
            wordlist[4] = 1
            #print("Contains effective: ")
        if "enjoyable" in review:
            wordlist[5] = 1
            #print("Contains enjoyable: ")
        if "great" in review:
            wordlist[6] = 1
            #print("Contains great: ")
        if "hilarious" in review:
            wordlist[7] = 1
            #print("Contains hilarious: ")

        #print(wordlist)
        masterlist.append(wordlist)
        wordlist = [0, 0 ,0 ,0, 0 ,0 ,0, 0]
        #print(review)

        #print(file)
        #print(f)

        f.close()
        print(reviews)
    #seperateData(masterlist)
    #accuracyTest(masterlist)
def accuracyTest(masterlist):

    probNeg = [0.122, 0.545, 0.175, 0.101, 0.086, 0.054, 0.32, 0.059]

    probPos = [0.034, 0.28, 0.054, 0.025, 0.154, 0.096, 0.485, 0.132]

    negreviews = 0
    posreviews = 0
    for review in masterlist:
        pos = 0
        neg =0
        if review[0] == 1:
            pos += probPos[0]
            neg += probNeg[0]
        if review[1] == 1:
            pos += probPos[1]
            neg += probNeg[1]
        if review[2] == 1:
            pos += probPos[2]
            neg += probNeg[2]
        if review[3] == 1:
            pos += probPos[3]
            neg += probNeg[3]
        if review[4] == 1:
            pos += probPos[4]
            neg += probNeg[4]
        if review[5] == 1:
            pos += probPos[5]
            neg += probNeg[5]
        if review[6] == 1:
            pos += probPos[6]
            neg += probNeg[6]
        if review[7] == 1:
            pos += probPos[7]
            neg += probNeg[7]
        if pos>neg:
            posreviews += 1
        else:
            negreviews += 1

    print (posreviews)
    print (negreviews)


def main():

    #findNegative()
    findPositive()


if __name__ == "__main__":
    main()
```

Review Generator

This is the code for the last bit of Part 3. It generates the random “reviews” from taking a sample from the probabilities of the 8 words. It then appends them to various review sentences to make up the whole “review”

```
import sys
import random

def generatenegSentence():

    probNeg = [0.122/1.462, 0.545/1.462, 0.175/1.462, 0.101/1.462,
0.086/1.462, 0.054/1.462, 0.32/1.462, 0.059/1.462]

    sentencelist = ["The movie was real ", "The actors were ", "The
screenplay is very  ", "Overall it was "]


    awful = 1 - probNeg[0]
    bad = awful - probNeg[1]
    boring = bad - probNeg[2]
    dull = boring - probNeg[3]
    effective = dull - probNeg[4]
    enjoyable = effective - probNeg[5]
    great = enjoyable - probNeg[6]
    hilarious = great - probNeg[7]

    #print(awful, bad, boring, dull , effective, enjoyable, great, hilarious)

    for sentence in sentencelist:

        number = random.random()

        if number > awful:
            sentence += "awful"
        if number < bad and number > boring:
            sentence += "bad"
        if number < boring and number > dull:
            sentence += "boring"
        if number < dull and number > effective:
            sentence += "dull"
        if number < effective and number > enjoyable:
            sentence += "effective"
        if number < enjoyable and number > great:
            sentence += "enjoyable"
        if number < great and number > hilarious:
            sentence += "great"
        if number < hilarious and number > 0:
            sentence += "hilarious"


        print(sentence + "\n")

    #print("------------------------------------\n")
```

```
def generateposSentence():

    probNeg = [0.034/1.26, 0.28/1.26, 0.054/1.26, 0.025/1.26, 0.154/1.26,
0.096/1.26, 0.485/1.26, 0.132/1.26]
    sentencelist = ["The movie was real ", "The actors were ", "The
screenplay is very ", "Overall it was "]
    awful = 1- probNeg[0]
    bad = awful - probNeg[1]
    boring = bad - probNeg[2]
    dull = boring - probNeg[3]
    effective = dull - probNeg[4]
    enjoyable = effective - probNeg[5]
    great = enjoyable - probNeg[6]
    hilarious = great - probNeg[7]

    #print(awful, bad, boring, dull , effective, enjoyable, great, hilarious)

    for sentence in sentencelist:

        number = random.random()
        #print(number)
        if number > awful:
            sentence += "awful"
        if number <= bad and number > boring:
            sentence += "bad"
        if number <= boring and number > dull:
            sentence += "boring"
        if number <= dull and number > effective:
            sentence += "dull"
        if number <= effective and number > enjoyable:
            sentence += "effective"
        if number <= enjoyable and number > great:
            sentence += "enjoyable"
        if number <= great and number > hilarious:
            sentence += "great"
        if number <= hilarious and number > 0:
            sentence += "hilarious"

        print(sentence + "\n")

    #print("------------------------------------\n")

def main():

    for num in range(0,5):

        generatenegSentence()
        #generateposSentence()


if __name__ == "__main__":
    main()
```