761 HW1 Extended Response Answers

# Question 1

S

# Question 2

Part A:

*def* parta():  
 randStudent = random.random() \* (8094 - 0) + 0 #rand()\*(max – min) + min  
 *print*("Part a: %d" % (randStudent))

Part B:

*def* partb(*k*):  
 *for* i *in* range(0, *k*):  
 randStudent = random.random() \* (8094 - 0) + 0 #rand()\*(max – min) + min  
 *print*("Student %d: %d" % (i+1, randStudent))

Part C:

The idea here is that each element in the pdf matrix corresponds with some probability “mass”. This mass can be iteratively summed element by element to find the probability of landing within a specific square. Thus, the function starts with a random number generated uniformly between 0 and 1. Then, each element in the matrix is iterated over to see if the initial random number is between the previous pdf sum (starting with 0), and the current pdf sum (found by adding the next pdf matrix element). When the box for which the random number within the summed pdf range (also known as the cumulative distribution function) is found, the corresponding row and column indices are returned. Code to empirically test the generation method is provided below, as well as the empirical generation results.

Code to compute pdf:

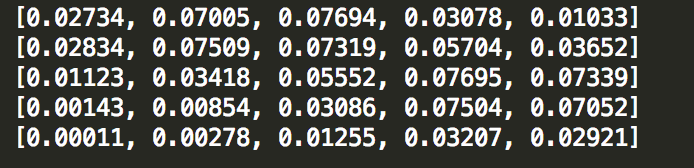
*import* random  
pdf = [  
 [0.0267, 0.0697, 0.0775, 0.0313, 0.0101],  
 [0.0283, 0.0761, 0.0739, 0.0566, 0.0362],  
 [0.0109, 0.0337, 0.0552, 0.0780, 0.0740],  
 [0.0014, 0.0093, 0.0309, 0.0750, 0.0698],  
 [0.0001, 0.0028, 0.0124, 0.0318, 0.0283]

]  
  
*def* partc1():  
 X1X2RandPoint = random.random()  
 x1Value = 0;  
 x2Value = 0;  
 pdfSum = 0  
 *for* row *in* range(0, len(pdf)):  
 *for* col *in* range(0, len(pdf[0])):  
 pdfSum += pdf[row][col]  
 *if* X1X2RandPoint <= pdfSum:  
 x1Value = row  
 x2Value = col  
 *return* x1Value, x2Value

Code to test pdf empirically:

*def* partc2Test():  
 buckets = [[0 *for* col *in* range(0, 5)] *for* row *in* range(0, 5)]  
 *for* i *in* range(0, 100000):  
 x1, x2 = partc2()  
 buckets[x1][x2] += 1  
  
 *for* row *in* range(0, len(buckets)):  
 *for* col *in* range(0, len(buckets[0])):  
 buckets[row][col] /= 100000.0  
 *for* row *in* range(0, len(buckets)):  
 *print*(buckets[row])

Results of empirical test:



# Question 3

Part A:

Since both variables are normally distributed, and N(0, 1) and N(1, 1) have a 95% confidence interval of (From the equation ), a more statistically valid way of determining if someone is healthy is to say that if the person’s number generated by the screening is below -0.96 (which is 1 – 1.96), then the person is healthy. In this case, we are only taking into consideration the cancer distribution, and effectively assuming that we would rather have more people from the healthy distribution be labeled as having cancer, since there is less negative impact of classifying healthy people with cancer, as opposed to classifying people with cancer as healthy.

This test should not be marketed as a good test for cancer, as the standard deviation is so high, that the 95% confidence intervals for being healthy and having cancer heavily overlap. This effectively means that because the variation is so high and the means are so close, it is extremely difficult to make a determination between people who are healthy and people who have cancer.

Part B:

If the equation provided above for confidence bounds is used (From the equation ), and is replaced with which is the sampling standard deviation, then a better variance estimate might be possible, which could lead to a more robust test for cancer.

Part C:

If the equation provided above for confidence bounds is used (From the equation ), and is replaced with , where N is the number of trials, then the standard deviation for part B with 5 trials is 0.447, as opposed to 0.25 with the new test. Therefore, the new test has tighter confidence bounds which can be used for discriminating between healthy and cancer patients for roughly the same cost as the previous test with 5 trials. So, the new procedure should be recommended in this case.

# Question 4

# Question 5

Procedure to test for conditional independence:

There are multiple steps to my procedure for conditional independence checking. The main equation to check for conditional independence is:

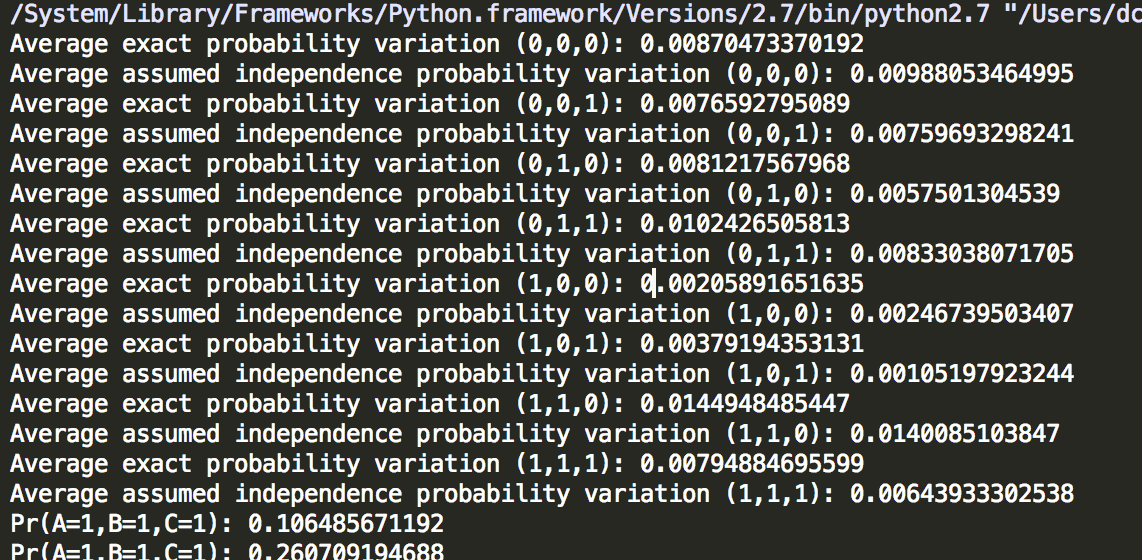
An alternative equation to calculate the conditional probability is:

Using the two above equations provides an empirical approach to evaluating if conditional independence holds for a set of variables xA, xB, and xC. If the computations over the data for the first and second probability computations listed above roughly equal, then the variables xA and xC are conditionally independent given xB. Thus, the code empirically calculates the conditional probabilities using the two results above, and compares the results.

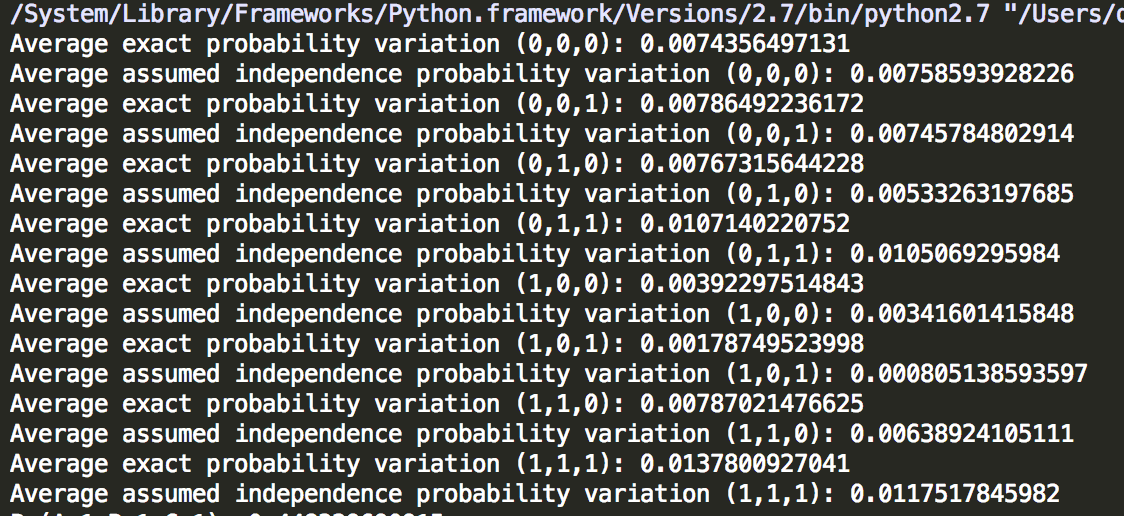
However, another question is, is the difference between the probabilities a significant difference? To determine this, another function is provided which randomly generates data permutations, and calculates the variation in individual probability calculations. With these numbers, the significance of the total difference in a probability calculation can be determined.

Results:

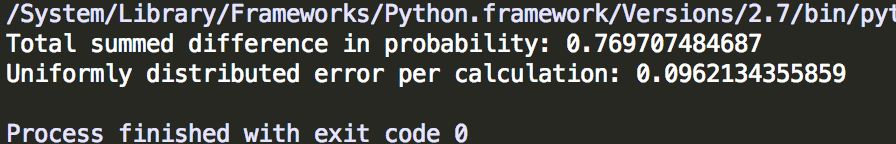
Within probability average differences, brain\_data1.mat:



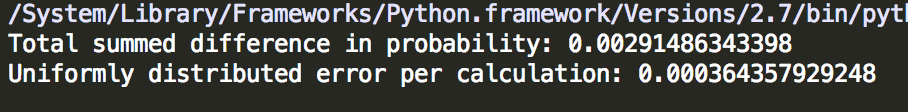
Within probability average differences, brain\_data2.mat:



Differences in exact conditional probability and assumed independence probability calculations, brain\_data1.mat:



Differences in exact conditional probability and assumed independence probability calculations, brain\_data2.mat:



Discussion and conclusions:

This first image above within this results section shows the average difference in probability calculations for both the exact conditional probability equation on the “brain\_data1.mat” data, and the equation in which independence is assumed over all combinations of values of xA, xB, and xC (A, B, and C). This list shows the relative variation in probability over multiple permutations. The second image above shows a similar list, but for the “brain\_data2.mat” data. For both data sets, the relative probability differences are of about the same order of magnitude.

The third and forth images above provide the total difference in probability over all combinations of xA, xB, and xC for the first and second brain data sets. For “brain\_data1.mat”, the total difference in probability per variable combination is of about an order of magnitude above the overall variation of the probability calculations, making it significant that xA and xC are **not** conditionally independent for **brain\_data1.mat**, given xB. For “brain\_data2.mat”, the total difference in probability per variable combination is of about an order of magnitude smaller than the overall variation of the probability calculations, making it significant that xA and xC **are** conditionally independent for **brain\_data2.mat**,given xB.

Python Code:

*import* scipy.io  
*import* numpy *as* np  
  
*def* main():  
 #Read data  
 mat = scipy.io.loadmat('brain\_data1.mat')  
 #Parse data into a more useable format  
 data = {'A':[], 'B':[], 'C': []}  
 *for* row *in* mat['xA']:  
 data['A'].append(row[0])  
 *for* row *in* mat['xB']:  
 data['B'].append(row[0])  
 *for* row *in* mat['xC']:  
 data['C'].append(row[0])  
  
 #Calculate variation in exact probability, and independence assumed probability  
 # for random subsets of the data to determine magnitude of variation.  
 calculateVariability(data)  
  
 #Calculate variation of probabilities for data.  
 *print*("Total summed difference in probability: " + str(getEmpiricalTotalConditionalProbabilityDifference(data)))  
 *print*("Uniformly distributed error per calculation: " + str(getEmpiricalTotalConditionalProbabilityDifference(data)/8))  
  
  
'''  
Calculate variability of inter-probability calculations.  
'''  
*def* calculateVariability(*data*):  
 *for* A *in* [0, 1]:  
 *for* B *in* [0, 1]:  
 *for* C *in* [0, 1]:  
 # Calculate variation in exact probability, and independence assumed probability  
 # for random subsets of the data to determine magnitude of variation.  
 exactProbVariationMean = 0  
 assumedIndepVariationMean = 0  
 randIndexList = list(range(0, len(*data*['A'])))  
 *for* k *in* range(0, 30):  
 # Shuffle data indecies.  
 np.random.shuffle(randIndexList)  
 firstHalf = {'A': [], 'B': [], 'C': []}  
 secondHalf = {'A': [], 'B': [], 'C': []}  
 *for* index *in* range(0, len(randIndexList) / 2):  
 firstHalf['A'].append(*data*['A'][randIndexList[index]])  
 firstHalf['B'].append(*data*['B'][randIndexList[index]])  
 firstHalf['C'].append(*data*['C'][randIndexList[index]])  
 *for* index *in* range(len(randIndexList) / 2, len(randIndexList)):  
 secondHalf['A'].append(*data*['A'][randIndexList[index]])  
 secondHalf['B'].append(*data*['B'][randIndexList[index]])  
 secondHalf['C'].append(*data*['C'][randIndexList[index]])  
 firstHalfExact, firstHalfAssumed = getEmpericalProbabilityDifference(firstHalf, A, B, C)  
 secondHalfExact, secondHalfAssumed = getEmpericalProbabilityDifference(secondHalf, A, B, C)  
  
 exactProbVariationMean += abs(firstHalfExact - secondHalfExact)  
 assumedIndepVariationMean += abs(firstHalfAssumed - secondHalfAssumed)  
  
 *print*("Average exact probability variation (%d,%d,%d): "%(A,B,C) + str(exactProbVariationMean / 30.0))  
 *print*("Average assumed independence probability variation (%d,%d,%d): "%(A,B,C) + str(assumedIndepVariationMean / 30.0))  
  
  
'''  
This method sums up the difference in the actual conditional probability  
and independence assumed conditional probabilities over all settings  
of A, B, and C.  
'''  
*def* getEmpiricalTotalConditionalProbabilityDifference(*data*):  
 totalDifference = 0  
 *for* A *in* [0, 1]:  
 *for* B *in* [0, 1]:  
 *for* C *in* [0, 1]:  
 exactProb, assumedIndepProb = getEmpericalProbabilityDifference(*data*, A, B, C)  
 totalDifference += abs(exactProb - assumedIndepProb)  
 *return* totalDifference  
  
  
'''  
 Method computes the difference in the true conditional probability  
 and independence assumed conditional probabilities for ONE setting  
 of A, B, and C.  
  
 @:param data Dictionary with 'A', 'B', and 'C' as keys, and arrays as values.  
 @:param A value of A to use for calculation (0, or 1)  
 @:param B value of B to use for calculation (0, or 1)  
 @:param C value of C to use for calculation (0, or 1)  
'''  
*def* getEmpericalProbabilityDifference(*data*, *A*, *B*, *C*):  
 #Calculate Pr(A, B, C)  
 entryCount = 0  
 *for* i *in* range(0, len(*data*['A'])):  
 *if A* == *data*['A'][i] *and B* == *data*['B'][i] *and C* == *data*['C'][i]:  
 entryCount += 1.0  
 totalCount = len(*data*['A'])  
 prA\_B\_C =entryCount / totalCount  
  
 #Calculate Pr(B)  
 entryCount = len(filter(*lambda item*: item==*B*, *data*['B']))  
 prB = entryCount / float(totalCount)  
  
 #cProb1 = Pr(A,C|B) = Pr(A,B,C) / Pr(B)  
 cProb1 = prA\_B\_C / prB  
  
 #Calculate Pr(A|B) = Pr(A,B) / Pr(B)  
 entryCount = 0  
 *for* i *in* range(0, len(*data*['A'])):  
 *if A* == *data*['A'][i] *and B* == *data*['B'][i]:  
 entryCount += 1.0  
 prA\_B = entryCount / totalCount  
  
 #Calculate Pr(C|B) = Pr(C,B) / Pr(B)  
 entryCount = 0  
 *for* i *in* range(0, len(*data*['A'])):  
 *if C* == *data*['C'][i] *and B* == *data*['B'][i]:  
 entryCount += 1.0  
 prB\_C = entryCount / totalCount  
  
 #cProb2 = Pr(A|B)\*Pr(C|B)  
 cProb2 = (prB\_C\*prA\_B) / (prB\*\*2)  
  
 #If |cProb1 - cProb2| within experimental variation bound  
 *return* cProb1, cProb2  
  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 main()