CSC421: Artificial Intelligence
Assignment 3

Rafael Solorzano

June 26, 2018

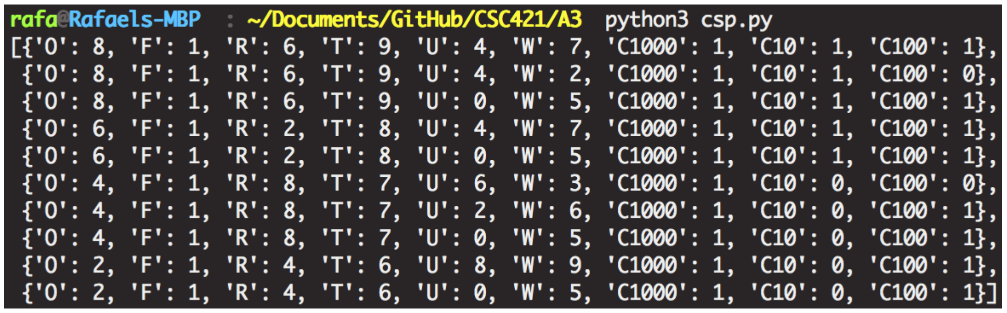
1 Introduction

In this assignment CSP problems solving using python-csp is explored. Two different problems are solved, the first being a Crypto-arithmetic puzzle and the second being the Waltz Filtering problem. Further, a Naive Bayes Text Classification is computer similar to the movie classification done in Assignment 2, but this time the entire computation is done using scikit-learn. Finally, usage of pgmpy to create Bayesian Networks and get results is done.

2 CSP

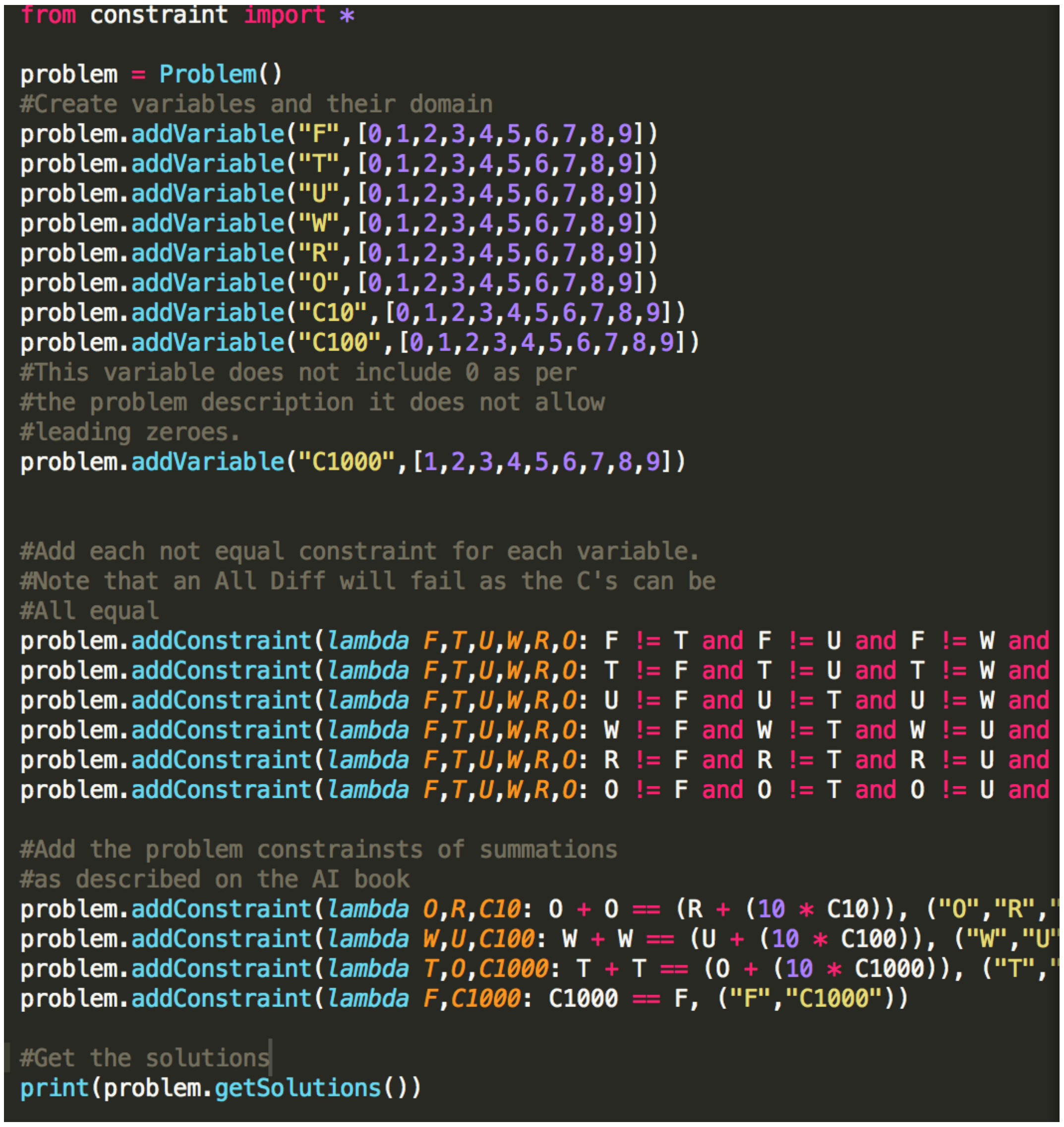
2.1 Crypto-arithmetic Puzzle

The crypto-arithmetic puzzle solved in this problem consists of finding a consistent assignment for the sum of TWO + TWO = FOUR in which each letter is assigned a number and the result of each letter produces FOUR. The problem is solved using python-csp and yields several results which are shown in Figure 1. Further the code is shown in Figure 2. The python-csp is used by first creating a problem object and further assigning variables and its corresponding domain.Second, the constraints of the problem are specified; for this specific problem all the main variables must be non-equal therefore they are assigned as so, further the other constraint that is added is the summation result of each variable, according to the problem definition. Finally, the solutions of the problem are printed.



```
rafa@Rafaels-MBP  : ~/Documents/GitHub/CSC421/A3  python3 csp.py
[{'O': 8, 'F': 1, 'R': 6, 'T': 9, 'U': 4, 'W': 7, 'C1000': 1, 'C10': 1, 'C100': 1},
 {'O': 8, 'F': 1, 'R': 6, 'T': 9, 'U': 4, 'W': 2, 'C1000': 1, 'C10': 1, 'C100': 0},
 {'O': 8, 'F': 1, 'R': 6, 'T': 9, 'U': 0, 'W': 5, 'C1000': 1, 'C10': 1, 'C100': 1},
 {'O': 6, 'F': 1, 'R': 2, 'T': 8, 'U': 4, 'W': 7, 'C1000': 1, 'C10': 1, 'C100': 1},
 {'O': 6, 'F': 1, 'R': 2, 'T': 8, 'U': 0, 'W': 5, 'C1000': 1, 'C10': 1, 'C100': 1},
 {'O': 4, 'F': 1, 'R': 8, 'T': 7, 'U': 6, 'W': 3, 'C1000': 1, 'C10': 0, 'C100': 0},
 {'O': 4, 'F': 1, 'R': 8, 'T': 7, 'U': 2, 'W': 6, 'C1000': 1, 'C10': 0, 'C100': 1},
 {'O': 4, 'F': 1, 'R': 8, 'T': 7, 'U': 0, 'W': 5, 'C1000': 1, 'C10': 0, 'C100': 1},
 {'O': 2, 'F': 1, 'R': 4, 'T': 6, 'U': 8, 'W': 9, 'C1000': 1, 'C10': 0, 'C100': 1},
 {'O': 2, 'F': 1, 'R': 4, 'T': 6, 'U': 0, 'W': 5, 'C1000': 1, 'C10': 0, 'C100': 1}]
```

Figure 1: The solutions for the crypto puzzle.



```
from constraint import *

problem = Problem()
#Create variables and their domain
problem.addVariable("F",[0,1,2,3,4,5,6,7,8,9])
problem.addVariable("T",[0,1,2,3,4,5,6,7,8,9])
problem.addVariable("U",[0,1,2,3,4,5,6,7,8,9])
problem.addVariable("W",[0,1,2,3,4,5,6,7,8,9])
problem.addVariable("R",[0,1,2,3,4,5,6,7,8,9])
problem.addVariable("O",[0,1,2,3,4,5,6,7,8,9])
problem.addVariable("C10",[0,1,2,3,4,5,6,7,8,9])
problem.addVariable("C100",[0,1,2,3,4,5,6,7,8,9])
#This variable does not include 0 as per
#the problem description it does not allow
#leading zeroes.
problem.addVariable("C1000",[1,2,3,4,5,6,7,8,9])


#Add each not equal constraint for each variable.
#Note that an All Diff will fail as the C's can be
#All equal
problem.addConstraint(lambda F,T,U,W,R,O: F != T and F != U and F != W and
problem.addConstraint(lambda F,T,U,W,R,O: T != F and T != U and T != W and
problem.addConstraint(lambda F,T,U,W,R,O: U != F and U != T and U != W and
problem.addConstraint(lambda F,T,U,W,R,O: W != F and W != T and W != U and
problem.addConstraint(lambda F,T,U,W,R,O: R != F and R != T and R != U and
problem.addConstraint(lambda F,T,U,W,R,O: O != F and O != T and O != U and

#Add the problem constrainsts of summations
#as described on the AI book
problem.addConstraint(lambda O,R,C10: O + O == (R + (10 * C10)), ("O","R","
problem.addConstraint(lambda W,U,C100: W + W == (U + (10 * C100)), ("W","U"
problem.addConstraint(lambda T,O,C1000: T + T == (O + (10 * C1000)), ("T","
problem.addConstraint(lambda F,C1000: C1000 == F, ("F","C1000"))

#Get the solutions
print(problem.getSolutions())
```

Figure 2: The code that solves the crypto puzzle.

2.2 Waltz Filtering

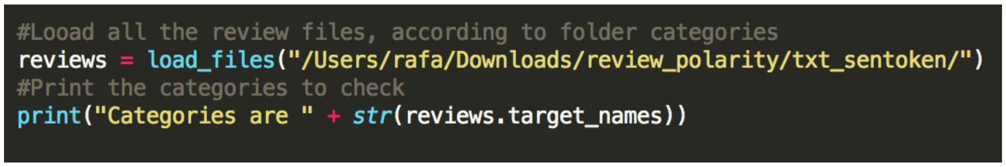
3 Naive Bayes Text Classification

Note that in this problem additionally to the the 10-fold cross validation i compared this against a train test split of 25% saved for testing. The results of both are presented in this section.

In this problem text classification of movies reviews that are classified in negative and positive is done using scikit-learn. First, the reviews text files are loaded directly using scikit feature extraction text functions, further the categories detected by scikit, which are based on folder names are printed, this is done for the sole purpose of checking if the correct folders were interpreted. This code is shown in Figure 3.

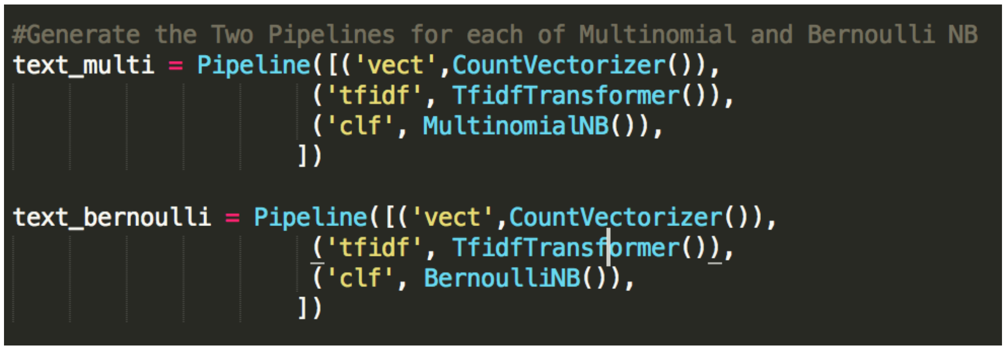
Once, the text is processed Pipelines for their classification using Multinomial and Bernoulli Naive Bayes are created. This pipelines are shown in Figure 4. Further, the test and train split is created, and accuracy is calculated using this method for both Bayes classifiers. Finally, the

10-fold cross-validation is performed using both classifiers. The code for both this methods is shown in Figure 5, whereas their results are shown in Figure 6, note that the best accuracy was achieved using the 10-fold cross validation with Multinomial NB.



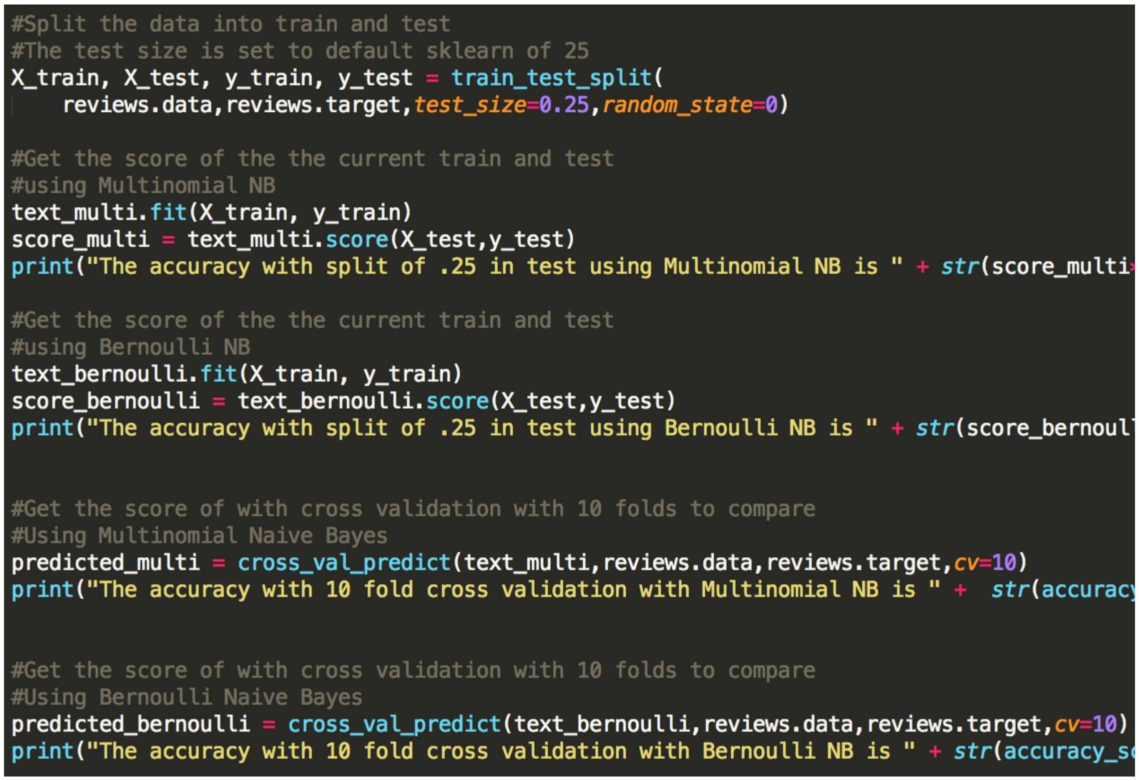
```
#Load all the review files, according to folder categories
reviews = load_files("/Users/rafa/Downloads/review_polarity/txt_sentoken/")
#Print the categories to check
print("Categories are " + str(reviews.target_names))
```

Figure 3: Loading the movie reviews txt files.



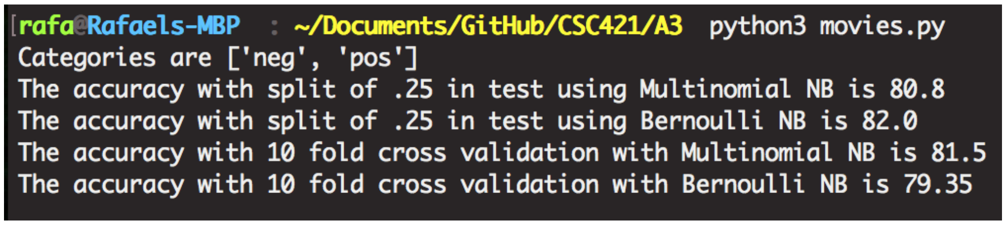
```
#Generate the Two Pipelines for each of Multinomial and Bernoulli NB
text_multi = Pipeline([('vect',CountVectorizer()),
                        ('tfidf', TfidfTransformer()),
                        ('clf', MultinomialNB()),
                       ])

text_bernoulli = Pipeline([('vect',CountVectorizer()),
                        ('tfidf', TfidfTransformer()),
                        ('clf', BernoulliNB()),
                       ])
```

Figure 4: Pipelines for Multinomial and Bernoulli NB.



```
#Split the data into train and test
#The test size is set to default sklearn of 25
X_train, X_test, y_train, y_test = train_test_split(
    reviews.data,reviews.target,test_size=0.25,random_state=0)

#Get the score of the the current train and test
#using Multinomial NB
text_multi.fit(X_train, y_train)
score_multi = text_multi.score(X_test,y_test)
print("The accuracy with split of .25 in test using Multinomial NB is " + str(score_multi

#Get the score of the the current train and test
#using Bernoulli NB
text_bernoulli.fit(X_train, y_train)
score_bernoulli = text_bernoulli.score(X_test,y_test)
print("The accuracy with split of .25 in test using Bernoulli NB is " + str(score_bernoul


#Get the score of with cross validation with 10 folds to compare
#Using Multinomial Naive Bayes
predicted_multi = cross_val_predict(text_multi,reviews.data,reviews.target,cv=10)
print("The accuracy with 10 fold cross validation with Multinomial NB is " +  str(accuracy


#Get the score of with cross validation with 10 folds to compare
#Using Bernoulli Naive Bayes
predicted_bernoulli = cross_val_predict(text_bernoulli,reviews.data,reviews.target,cv=10)
print("The accuracy with 10 fold cross validation with Bernoulli NB is " + str(accuracy_sc
```

Figure 5: Performing the classification, with test-train split and cross-validation.



```
rafa@Rafaels-MBP  : ~/Documents/GitHub/CSC421/A3  python3 movies.py
Categories are ['neg', 'pos']
The accuracy with split of .25 in test using Multinomial NB is 80.8
The accuracy with split of .25 in test using Bernoulli NB is 82.0
The accuracy with 10 fold cross validation with Multinomial NB is 81.5
The accuracy with 10 fold cross validation with Bernoulli NB is 79.35
```

Figure 6: All the results of test-train and cross-validation using both Multinomial and Bernoulli NB.

4 Discrete Bayesian Networks

In this section pgmpy is used to create a Bayesian Network. First, the Bayesian model is created, that indicated the connections between nodes. Further, the probabilities distribution of each node is specified using TabularCPD in which the variable name being specified, the number of variable cards it's values, it's evidence, and the evidence_card is specified. The code that creates this tables is shown in Figure 7. Further an example printout of a table is shown in Figure 8. Once the CPDs have been created they are added to the model, once this is done an inference object can be created to perform inference

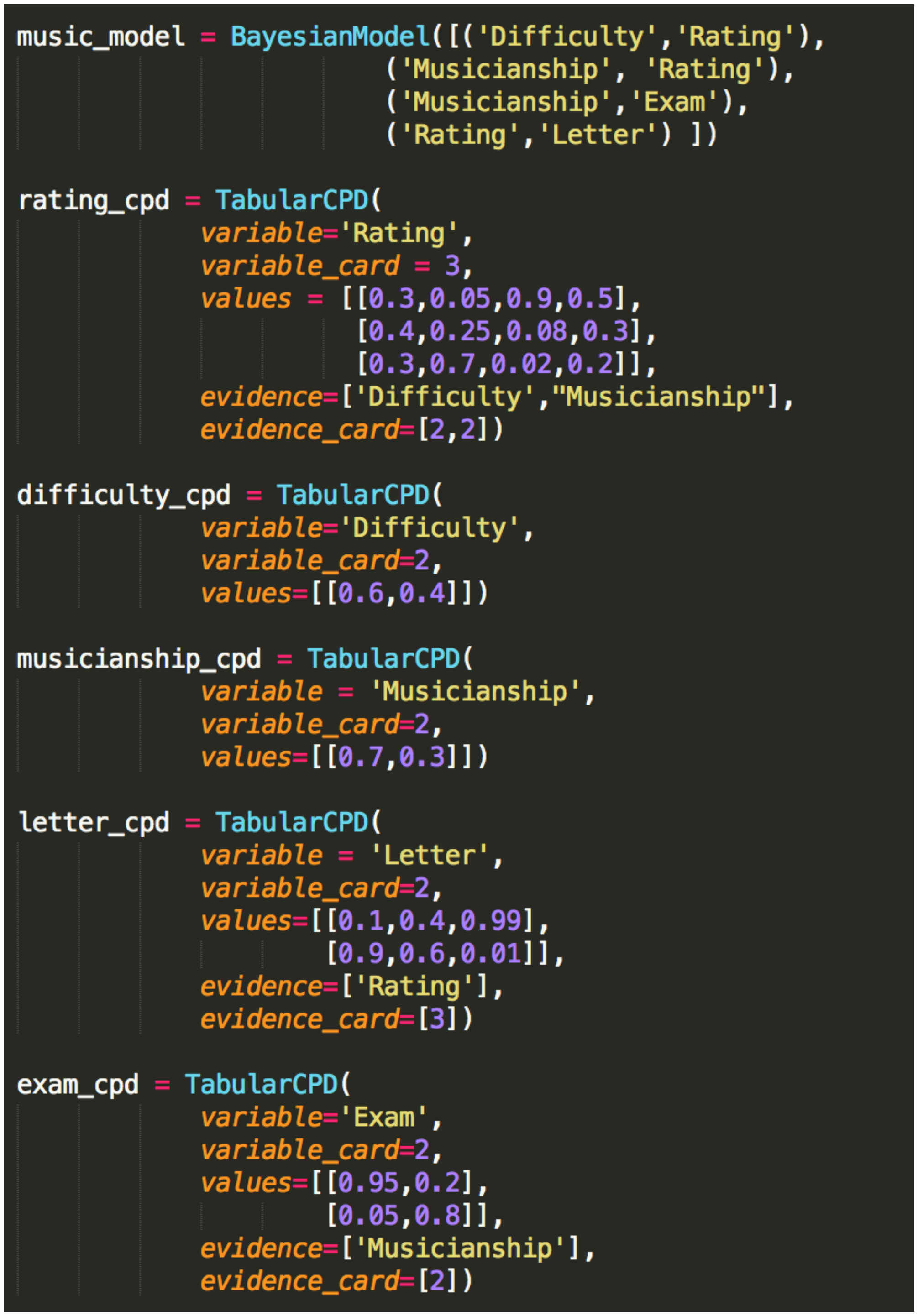
The query provided as example in the assignment can be computed by using inference step by step. The code that achieves this is shown in Figure 9, whereas the result is shown in Figure 10. Note that pgmpy returns a Discrete Factor object that prints to a table containing all probabilities related to the query, it is easy to find the required values in each table.

Finally, when we want to know how likely George is to get a strong recommendation letter we do it with inference as well. This yields a result of 56.8%. Further if we want to know the same probabilities but know we know that George is not a strong musician this is also is done by inference, and yields a results of 65.1%. The code that infers the data is shown in Figure 11, whereas its results are shown in Figure 12, in order.

Note that this may seem non-sensical but as it has been mentioned in lecture and announcements the data is non-sensical and is only for demonstration of usage of the pgmpy library further, the results do not match assignment 3 results in the description as it was also mentioned that the results provided in the assignment are wrong and there was no easy way to fix them to provide correct results for comparison.

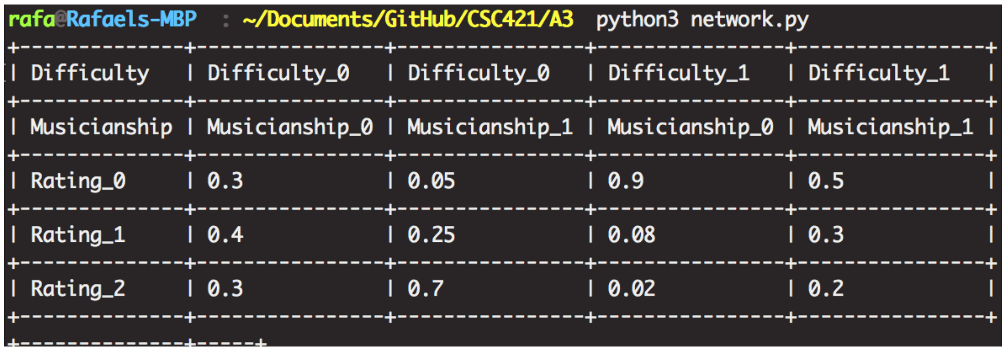
4.1 Approximation Inference

Approximate Inference was attempted with pgmpy by using MPLP. Max-Product Linear Programming uses a Markov Model for inference so the Bayesian Network is first transformed into a Markov Model using the built in pgmpy function. Sadly, this produced an error, that is caused by either the implementation of MPLP or the transferring of my Bayesian Network into Markov Model. Nevertheless, the code is shown in Figure 13, with the resultant error in Figure 14. The expected result of the last map_query function would be a dictionary of the best assignment of the nodes in a form of a dictionary.



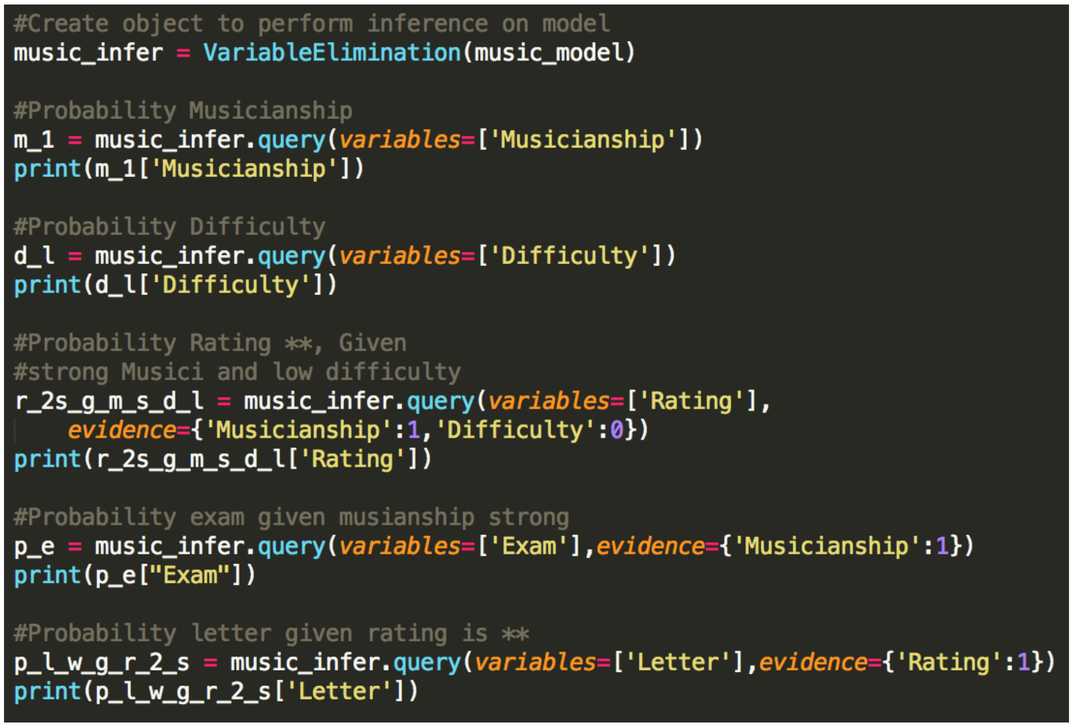
```
music_model = BayesianModel([('Difficulty','Rating'),
                             ('Musicianship', 'Rating'),
                             ('Musicianship','Exam'),
                             ('Rating','Letter') ])

rating_cpd = TabularCPD(
            variable='Rating',
            variable_card = 3,
            values = [[0.3,0.05,0.9,0.5],
                      [0.4,0.25,0.08,0.3],
                      [0.3,0.7,0.02,0.2]],
            evidence=['Difficulty',"Musicianship"],
            evidence_card=[2,2])

difficulty_cpd = TabularCPD(
            variable='Difficulty',
            variable_card=2,
            values=[[0.6,0.4]])

musicianship_cpd = TabularCPD(
            variable = 'Musicianship',
            variable_card=2,
            values=[[0.7,0.3]])

letter_cpd = TabularCPD(
            variable = 'Letter',
            variable_card=2,
            values=[[0.1,0.4,0.99],
                    [0.9,0.6,0.01]],
            evidence=['Rating'],
            evidence_card=[3])

exam_cpd = TabularCPD(
            variable='Exam',
            variable_card=2,
            values=[[0.95,0.2],
                    [0.05,0.8]],
            evidence=['Musicianship'],
            evidence_card=[2])
```

Figure 7: Creating the Bayesian Network structure.



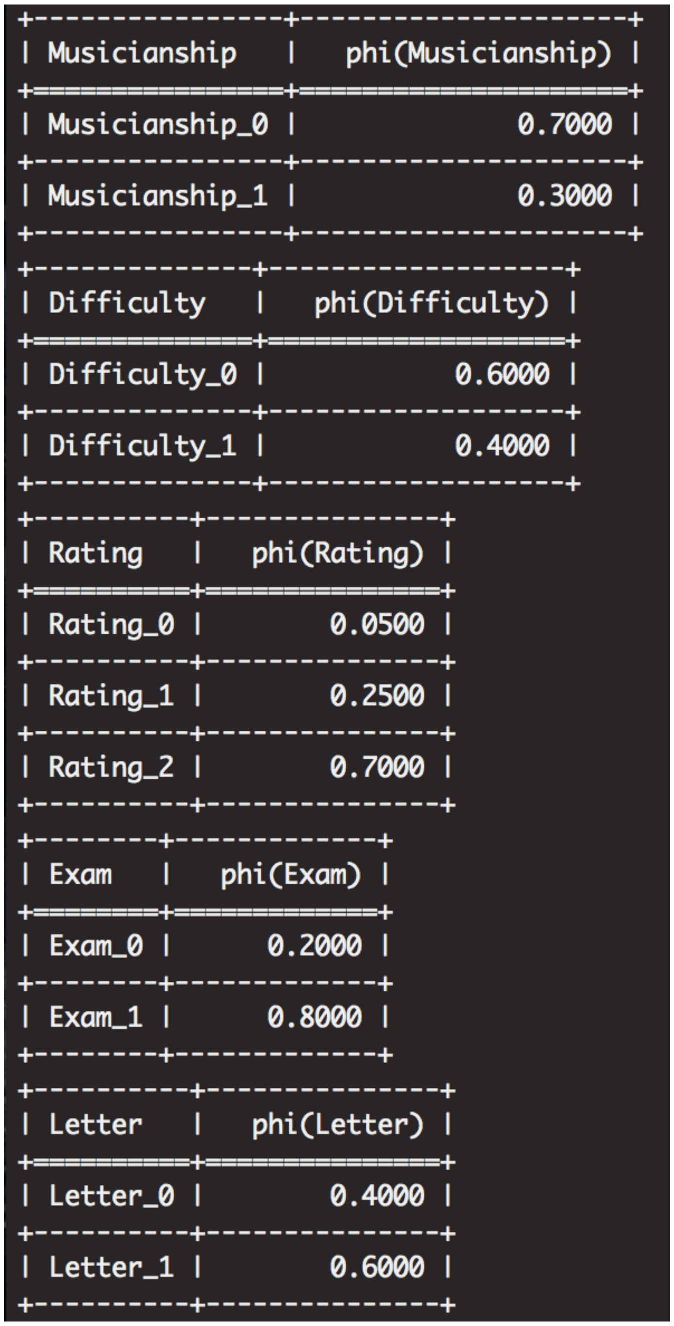
```
rafa@Rafaels-MBP  : ~/Documents/GitHub/CSC421/A3  python3 network.py
+-------------+---------------+---------------+---------------+---------------+
| Difficulty  | Difficulty_0  | Difficulty_0  | Difficulty_1  | Difficulty_1  |
+-------------+---------------+---------------+---------------+---------------+
| Musicianship | Musicianship_0 | Musicianship_1 | Musicianship_0 | Musicianship_1 |
+-------------+---------------+---------------+---------------+---------------+
| Rating_0    | 0.3           | 0.05          | 0.9           | 0.5           |
+-------------+---------------+---------------+---------------+---------------+
| Rating_1    | 0.4           | 0.25          | 0.08          | 0.3           |
+-------------+---------------+---------------+---------------+---------------+
| Rating_2    | 0.3           | 0.7           | 0.02          | 0.2           |
+-------------+---------------+---------------+---------------+---------------+
+-------------+-----+
```

Figure 8: An example table created with TabularCPD.



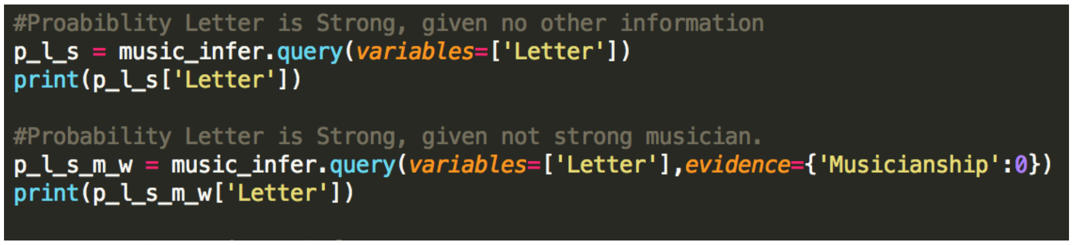
```
#Create object to perform inference on model
music_infer = VariableElimination(music_model)

#Probability Musicianship
m_1 = music_infer.query(variables=['Musicianship'])
print(m_1['Musicianship'])

#Probability Difficulty
d_l = music_infer.query(variables=['Difficulty'])
print(d_l['Difficulty'])

#Probability Rating **, Given
#strong Musici and low difficulty
r_2s_g_m_s_d_l = music_infer.query(variables=['Rating'],
    evidence={'Musicianship':1,'Difficulty':0})
print(r_2s_g_m_s_d_l['Rating'])

#Probability exam given musianship strong
p_e = music_infer.query(variables=['Exam'],evidence={'Musicianship':1})
print(p_e["Exam"])

#Probability letter given rating is **
p_l_w_g_r_2_s = music_infer.query(variables=['Letter'],evidence={'Rating':1})
print(p_l_w_g_r_2_s['Letter'])
```

Figure 9: Performing inference, according to assignment problem.



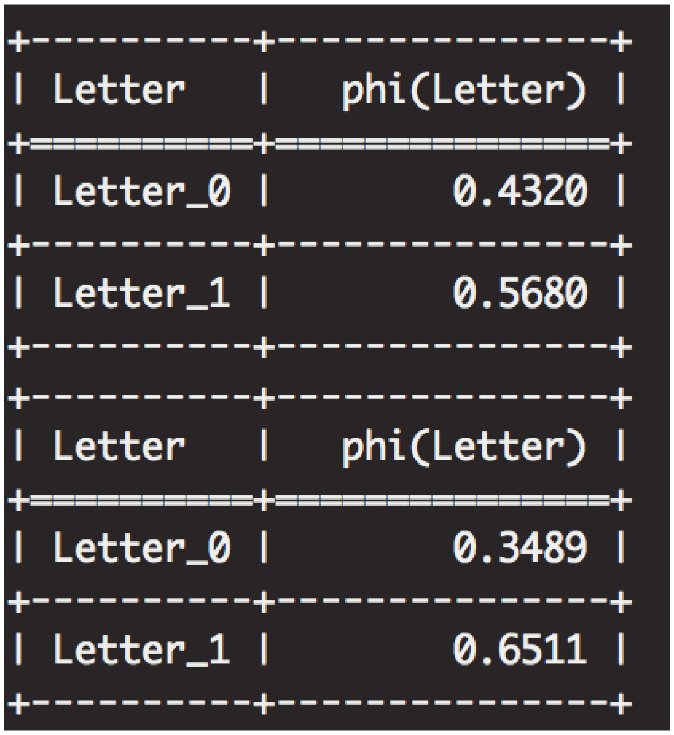
```
+----------------+---------------------+
| Musicianship   |   phi(Musicianship) |
+================+=====================+
| Musicianship_0 |              0.7000 |
+----------------+---------------------+
| Musicianship_1 |              0.3000 |
+----------------+---------------------+
+--------------+-------------------+
| Difficulty   |   phi(Difficulty) |
+==============+===================+
| Difficulty_0 |            0.6000 |
+--------------+-------------------+
| Difficulty_1 |            0.4000 |
+--------------+-------------------+
+----------+---------------+
| Rating   |   phi(Rating) |
+==========+===============+
| Rating_0 |        0.0500 |
+----------+---------------+
| Rating_1 |        0.2500 |
+----------+---------------+
| Rating_2 |        0.7000 |
+----------+---------------+
+--------+-------------+
| Exam   |   phi(Exam) |
+========+=============+
| Exam_0 |      0.2000 |
+--------+-------------+
| Exam_1 |      0.8000 |
+--------+-------------+
+----------+---------------+
| Letter   |   phi(Letter) |
+==========+===============+
| Letter_0 |        0.4000 |
+----------+---------------+
| Letter_1 |        0.6000 |
+----------+---------------+
```

Figure 10: All the results of performing inference.



```python
#Proabiblity Letter is Strong, given no other information
p_l_s = music_infer.query(variables=['Letter'])
print(p_l_s['Letter'])

#Probability Letter is Strong, given not strong musician.
p_l_s_m_w = music_infer.query(variables=['Letter'],evidence={'Musicianship':0})
print(p_l_s_m_w['Letter'])
```

Figure 11: Performing inference to test strong letter of recommendation.



```
+----------+---------------+
| Letter   |   phi(Letter) |
+==========+===============+
| Letter_0 |        0.4320 |
+----------+---------------+
| Letter_1 |        0.5680 |
+----------+---------------+
+----------+---------------+
| Letter   |   phi(Letter) |
+==========+===============+
| Letter_0 |        0.3489 |
+----------+---------------+
| Letter_1 |        0.6511 |
+----------+---------------+
```

Figure 12: The result of the strong letter of recommendation.