CS 5824/ECE 5424: Advanced Machine Learning

Homework Assignment 2: Section A

Ankit Parekh
MS (Non-Thesis) Computer Engineering
ECE Department
ankitparekh@vt.edu

ankitparekh-Problem1-SectionA-HW2

November 3, 2022

Problem 1. Decision Trees (20 points total)

Q1. (5 points) You vaguely remember that using the information gain ratio can help remedy one of information gain's downsides when constructing a decision tree. Will doing so result in a significant change with regards to this dataset? Why or why not?

Answer:

A major drawback of using **Information Gain** as the criterion to pick a feature for split (root-node) is that it **favours features which have more unique values**. This can cause over-fitting due to bias during training and the tree would not perform well on test data. To overcome this limitation **Information Gain Ratio** is used which **normalizes the Information Gain value with the feature's entropy**.

In our dataset, we have 4 features namely: **Homework, Traffic, Hunger, and Lauren's Availability**. **"Homework"** and **"Traffic"** have 3 unique values while **"Hunger"** & **"Lauren"** have 2 unique values. Using Information Gain Ratio on this dataset will be beneficial as it will normalize the Information Gain for all the 4 features. We would have a **common metric to decide which feature to pick as root-node** which would not be biased towards the number of unique values of the feature.

Based on my calculations below, we observe the below difference in the order in which we pick features using both Information Gain(GR) and Information Gain Ratio(IGR). We observe that using IGR instead of GR in the current dataset would not change the decision tree significantly as the dataset size is small and we do not get a data split after the third feature choice.

Feature Preference using Information Gain: 1. Homework 2. Hunger 3. Lauren 4. Traffic

Feature Preference using Information Gain Ratio: 1. Homework 2. Hunger 3. Traffic 4. Lauren

Q2. (5 points) Compute the gain ratio from each feature for the first split of your decision tree using entropy as your purity criterion (C4.5 algorithm). Which feature should be your first split?

Answer:

Formulae used for below computations for reference:

1. Entropy(Decision) = − p(I) . log2p(I) = − p(Yes) . log2p(Yes) − p(No) . log2p(No)
2. Gain(Decision, feature) = Entropy(Decision) − (p(Decision|feature=possible_value) . Entropy(Decision|feature=possible_value))
3. SplitInfo(A) = - |Dj|/|D| x log2|Dj|/|D|
4. GainRatio(A) = Gain(A) / SplitInfo(A)

I have stored the dataset provided in Problem 1 in a .csv file and I will be creating a pandas dataframe by parsing that .csv file to compute the gain ratio for each feature in order to find the first split for the decision tree. Below is a step-by-step (with code) solution for the problem:

[1]:
```
from google.colab import drive
drive.mount("/content/gdrive/")
```

```
Mounted at /content/gdrive/
```

[9]:
```
import pandas as pd
path = "/content/gdrive/My Drive/ECE_5424_AML/HW2/hw2_problem1_data.csv"
df =  pd.read_csv(path)
df
```

[9]:

	Homework	Traffic	Hunger	Lauren	Go-out?
0	Much	Busy	A-little	Available	No
1	Much	Busy	A-little	Not-available	No
2	Normal	Busy	A-little	Available	Yes
3	None	OK	A-little	Available	Yes
4	None	Chill	A-lot	Available	Yes
5	None	Chill	A-lot	Not-available	No
6	Normal	Chill	A-lot	Not-available	Yes
7	Much	OK	A-little	Available	No
8	Much	Chill	A-lot	Available	Yes
9	None	OK	A-lot	Available	Yes
10	Much	OK	A-lot	Not-available	Yes
11	Normal	OK	A-little	Not-available	Yes
12	Normal	Busy	A-lot	Available	Yes
13	None	OK	A-little	Not-available	No

[10]:
```
import math

def step_break_1(): # function for output clarity, draws lines between steps
  for i in range(70):
    print("#", end="")
  print()

def step_break_2(): # function for output clarity, draws lines between sections␣
 ↪within steps
  for i in range(70):
    print("=", end="")
  print()

def step_break_3(): # function for output clarity, draws lines between sections␣
 ↪within steps
  for i in range(70):
    print("-", end="")
```

```
print()
```

[14]:
```
df = df.rename({'Go-out?': 'Decision'}, axis='columns') # Renaming Target␣
↪column as "Decision"
features = [x for x in df.columns][:-1] # Getting list of features and removing␣
↪target column
features
```

[14]:
```
['Homework', 'Traffic', 'Hunger', 'Lauren']
```

[17]:
```
def entropy(d):
  """
  Entropy(Decision) =  - p(I) . log2p(I) = - p(Yes) . log2p(Yes) - p(No) .␣
↪log2p(No)
  """
  entropy = 0
  total_samples = sum(d.values())
  print("Total Samples for Entropy Calculation : {}".format(total_samples))
  for label in d.keys():
    print("Number of samples with Decision={} : {}".format(label, d[label]))
    p = d[label]/(total_samples)
    print("Probability for Decision={} : {}".format(label, p))
    if d[label]!=0:
      lp = math.log(p, 2)
      print("Log of Probability for Decision={} : {}".format(label, lp))
      plp = -1*p*lp
    else:
      print("Log of Probability for Decision={} : {}".format(label, 0))
      plp = 0
    print("  - p(I) . log2p(I) for Decision={} : {}".format(label, plp))
    entropy+=plp

  return entropy


def possible_value_decision_dict(df, feature):
  """
  returns a dictionary of dictionaries with counts of each decision value␣
↪(dictionary) for all possible values in a feature
  ex:
  {'Much':{'Yes': 4, 'No': 1}, 'Normal':{'Yes': 4, 'No': 0}, 'None':{'Yes': 3,␣
↪'No': 2}}
  #Gain(Decision, feature) = Entropy(Decision) -   (␣
↪p(Decision|feature=possible_value) .␣
↪Entropy(Decision|feature=possible_value) )
  #SplitInfo(A) = -  |Dj|/|D| x log2|Dj|/|D|
```

```python
  #GainRatio(A) = Gain(A) / SplitInfo(A)
  """

  d = {}

  decision_dict = {}
  decision_dict = df['Decision'].value_counts().to_dict(decision_dict)
  decisions = decision_dict.keys()

  feature_dict = {}
  feature_dict = df[feature].value_counts().to_dict(feature_dict)
  possible_values = feature_dict.keys()

  for possible_value in possible_values:
    p = {}
    for decision in decisions:
      p[decision] = df[(df[feature] == possible_value) & (df['Decision'] ==␣
↪decision)].shape[0]
    d[possible_value] = p

  return d

def compute_gain_ratios(df):

  print("Step 1: Calculate Global Entropy")
  d = {}
  d = df['Decision'].value_counts().to_dict(d)
  global_entropy = entropy(d)
  print("Global Entropy for the Dataset : {}".format(global_entropy))

  step_break_1()

  print("Step 2: We need to find the most dominant factor for decisioning using␣
↪gain and gain ratios.")
  features = [x for x in df.columns][:-1] # Getting list of features and␣
↪removing target column
  print("Features in dataframe : {}".format(features))
  gain_dict = {}
  gain_ratio_dict = {}

  decision_dict = {}
  decision_dict = df['Decision'].value_counts().to_dict(decision_dict)
  num_samples = sum(decision_dict.values())

  # In our case all are nominal features
  for feature in features:
    print("Computing for Feature={}".format(feature))
```

```
    d_feature = possible_value_decision_dict(df, feature)
    print(d_feature)
    s = 0
    split_info = 0
    for possible_value in d_feature.keys():
      step_break_3()
      print("For possible_value={}".format(possible_value))
      prob =  sum(d_feature[possible_value].values())/num_samples
      possible_value_entropy = entropy(d_feature[possible_value])
      print("Entropy for possible_value={} : {}".format(possible_value,␣
↪possible_value_entropy))
      print("Probability of possible_value={} : {}".format(possible_value,␣
↪prob))
      s+= -1*prob*possible_value_entropy
      split_info+= -1*prob*(math.log(prob, 2))
    step_break_3()
    gain = global_entropy + s
    print("Gain for {} : {}".format(feature, gain))
    gain_dict[feature]=gain
    print("SplitInfo for {} : {}".format(feature, split_info))
    gain_ratio = gain/split_info
    print("Gain Ratio for {} : {}".format(feature, gain_ratio))
    gain_ratio_dict[feature] = gain_ratio
    step_break_2()

  return gain_dict, gain_ratio_dict

gain_dict, gain_ratio_dict = compute_gain_ratios(df)

for feature, gain in gain_dict.items():
  print("Gain for {} : ".format(feature), '%.3f'% gain)

for feature, gain_ratio in gain_ratio_dict.items():
  print("Gain Ratio for {} : ".format(feature), '%.3f'% gain_ratio)
```

```
Step 1: Calculate Global Entropy
Total Samples for Entropy Calculation : 14
Number of samples with Decision=Yes : 9
Probability for Decision=Yes : 0.6428571428571429
Log of Probability for Decision=Yes : -0.6374299206152917
 - p(I) . log2p(I) for Decision=Yes : 0.40977637753840185
Number of samples with Decision=No : 5
Probability for Decision=No : 0.35714285714285715
Log of Probability for Decision=No : -1.4854268271702415
 - p(I) . log2p(I) for Decision=No : 0.5305095811322291
Global Entropy for the Dataset : 0.9402859586706309
######################################################################
```

```
Step 2: We need to find the most dominant factor for decisioning using gain and
gain ratios.
Features in dataframe : ['Homework', 'Traffic', 'Hunger', 'Lauren']
Computing for Feature=Homework
{'Much': {'Yes': 2, 'No': 3}, 'None': {'Yes': 3, 'No': 2}, 'Normal': {'Yes': 4,
'No': 0}}
----------------------------------------------------------------------
For possible_value=Much
Total Samples for Entropy Calculation : 5
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 0.4
Log of Probability for Decision=Yes : -1.3219280948873622
 - p(I) . log2p(I) for Decision=Yes : 0.5287712379549449
Number of samples with Decision=No : 3
Probability for Decision=No : 0.6
Log of Probability for Decision=No : -0.7369655941662062
 - p(I) . log2p(I) for Decision=No : 0.44217935649972373
Entropy for possible_value=Much : 0.9709505944546686
Probability of possible_value=Much : 0.35714285714285715
----------------------------------------------------------------------
For possible_value=None
Total Samples for Entropy Calculation : 5
Number of samples with Decision=Yes : 3
Probability for Decision=Yes : 0.6
Log of Probability for Decision=Yes : -0.7369655941662062
 - p(I) . log2p(I) for Decision=Yes : 0.44217935649972373
Number of samples with Decision=No : 2
Probability for Decision=No : 0.4
Log of Probability for Decision=No : -1.3219280948873622
 - p(I) . log2p(I) for Decision=No : 0.5287712379549449
Entropy for possible_value=None : 0.9709505944546686
Probability of possible_value=None : 0.35714285714285715
----------------------------------------------------------------------
For possible_value=Normal
Total Samples for Entropy Calculation : 4
Number of samples with Decision=Yes : 4
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Number of samples with Decision=No : 0
Probability for Decision=No : 0.0
Log of Probability for Decision=No : 0
 - p(I) . log2p(I) for Decision=No : 0
Entropy for possible_value=Normal : 0.0
Probability of possible_value=Normal : 0.2857142857142857
----------------------------------------------------------------------
Gain for Homework : 0.2467498197744391
SplitInfo for Homework : 1.577406282852345
```

```
Gain Ratio for Homework : 0.15642756242117517
==================================================================
Computing for Feature=Traffic
{'OK': {'Yes': 4, 'No': 2}, 'Busy': {'Yes': 2, 'No': 2}, 'Chill': {'Yes': 3,
'No': 1}}
------------------------------------------------------------------
For possible_value=OK
Total Samples for Entropy Calculation : 6
Number of samples with Decision=Yes : 4
Probability for Decision=Yes : 0.6666666666666666
Log of Probability for Decision=Yes : -0.5849625007211563
 - p(I) . log2p(I) for Decision=Yes : 0.38997500048077083
Number of samples with Decision=No : 2
Probability for Decision=No : 0.3333333333333333
Log of Probability for Decision=No : -1.5849625007211563
 - p(I) . log2p(I) for Decision=No : 0.5283208335737187
Entropy for possible_value=OK : 0.9182958340544896
Probability of possible_value=OK : 0.42857142857142855
------------------------------------------------------------------
For possible_value=Busy
Total Samples for Entropy Calculation : 4
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 0.5
Log of Probability for Decision=Yes : -1.0
 - p(I) . log2p(I) for Decision=Yes : 0.5
Number of samples with Decision=No : 2
Probability for Decision=No : 0.5
Log of Probability for Decision=No : -1.0
 - p(I) . log2p(I) for Decision=No : 0.5
Entropy for possible_value=Busy : 1.0
Probability of possible_value=Busy : 0.2857142857142857
------------------------------------------------------------------
For possible_value=Chill
Total Samples for Entropy Calculation : 4
Number of samples with Decision=Yes : 3
Probability for Decision=Yes : 0.75
Log of Probability for Decision=Yes : -0.4150374992788438
 - p(I) . log2p(I) for Decision=Yes : 0.31127812445913283
Number of samples with Decision=No : 1
Probability for Decision=No : 0.25
Log of Probability for Decision=No : -2.0
 - p(I) . log2p(I) for Decision=No : 0.5
Entropy for possible_value=Chill : 0.8112781244591328
Probability of possible_value=Chill : 0.2857142857142857
------------------------------------------------------------------
Gain for Traffic : 0.029222565658954647
SplitInfo for Traffic : 1.5566567074628228
Gain Ratio for Traffic : 0.01877264622241867
```

```
==================================================================
Computing for Feature=Hunger
{'A-little': {'Yes': 3, 'No': 4}, 'A-lot': {'Yes': 6, 'No': 1}}
------------------------------------------------------------------
For possible_value=A-little
Total Samples for Entropy Calculation : 7
Number of samples with Decision=Yes : 3
Probability for Decision=Yes : 0.42857142857142855
Log of Probability for Decision=Yes : -1.222392421336448
 - p(I) . log2p(I) for Decision=Yes : 0.5238824662870492
Number of samples with Decision=No : 4
Probability for Decision=No : 0.5714285714285714
Log of Probability for Decision=No : -0.8073549220576043
 - p(I) . log2p(I) for Decision=No : 0.4613456697472024
Entropy for possible_value=A-little : 0.9852281360342516
Probability of possible_value=A-little : 0.5
------------------------------------------------------------------
For possible_value=A-lot
Total Samples for Entropy Calculation : 7
Number of samples with Decision=Yes : 6
Probability for Decision=Yes : 0.8571428571428571
Log of Probability for Decision=Yes : -0.22239242133644802
 - p(I) . log2p(I) for Decision=Yes : 0.19062207543124116
Number of samples with Decision=No : 1
Probability for Decision=No : 0.14285714285714285
Log of Probability for Decision=No : -2.8073549220576046
 - p(I) . log2p(I) for Decision=No : 0.4010507031510864
Entropy for possible_value=A-lot : 0.5916727785823275
Probability of possible_value=A-lot : 0.5
------------------------------------------------------------------
Gain for Hunger : 0.15183550136234136
SplitInfo for Hunger : 1.0
Gain Ratio for Hunger : 0.15183550136234136
==================================================================
Computing for Feature=Lauren
{'Available': {'Yes': 6, 'No': 2}, 'Not-available': {'Yes': 3, 'No': 3}}
------------------------------------------------------------------
For possible_value=Available
Total Samples for Entropy Calculation : 8
Number of samples with Decision=Yes : 6
Probability for Decision=Yes : 0.75
Log of Probability for Decision=Yes : -0.4150374992788438
 - p(I) . log2p(I) for Decision=Yes : 0.31127812445913283
Number of samples with Decision=No : 2
Probability for Decision=No : 0.25
Log of Probability for Decision=No : -2.0
 - p(I) . log2p(I) for Decision=No : 0.5
Entropy for possible_value=Available : 0.8112781244591328
```

```
Probability of possible_value=Available : 0.5714285714285714
----------------------------------------------------------------------
For possible_value=Not-available
Total Samples for Entropy Calculation : 6
Number of samples with Decision=Yes : 3
Probability for Decision=Yes : 0.5
Log of Probability for Decision=Yes : -1.0
  - p(I) . log2p(I) for Decision=Yes : 0.5
Number of samples with Decision=No : 3
Probability for Decision=No : 0.5
Log of Probability for Decision=No : -1.0
  - p(I) . log2p(I) for Decision=No : 0.5
Entropy for possible_value=Not-available : 1.0
Probability of possible_value=Not-available : 0.42857142857142855
----------------------------------------------------------------------
Gain for Lauren : 0.04812703040826927
SplitInfo for Lauren : 0.9852281360342516
Gain Ratio for Lauren : 0.048848615511520595
======================================================================
Gain for Homework :  0.247
Gain for Traffic :  0.029
Gain for Hunger :  0.152
Gain for Lauren :  0.048
Gain Ratio for Homework :  0.156
Gain Ratio for Traffic :  0.019
Gain Ratio for Hunger :  0.152
Gain Ratio for Lauren :  0.049
```

Decision Tree algorithms look for features that provide maximum information gain. They look for categorical features that provide highest information gain for splitting.

Q3. (8 points) Complete your decision tree. Include the tree and corresponding computations in your write-up; your tree could be either a digitally rendered visualization or a photo of a manual graph.

Answer:

Using the information gain ratios, we pick "Homework" as the first feature in the preference order, and we get the below subtables:

[20]:
```python
df_much = df.loc[df['Homework']=='Much'].drop(columns=['Homework'])
df_much
```

[20]:

	Traffic	Hunger	Lauren	Decision
0	Busy	A-little	Available	No
1	Busy	A-little	Not-available	No
7	OK	A-little	Available	No
8	Chill	A-lot	Available	Yes
10	OK	A-lot	Not-available	Yes

[21]:
```python
gain_dict_much, gain_ratio_dict_much = compute_gain_ratios(df_much)

for feature, gain in gain_dict_much.items():
  print("Gain for {} : ".format(feature), '%.3f'% gain)

for feature, gain_ratio in gain_ratio_dict_much.items():
  print("Gain Ratio for {} : ".format(feature), '%.3f'% gain_ratio)
```

```
Step 1: Calculate Global Entropy
Total Samples for Entropy Calculation : 5
Number of samples with Decision=No : 3
Probability for Decision=No : 0.6
Log of Probability for Decision=No : -0.7369655941662062
  - p(I) . log2p(I) for Decision=No : 0.44217935649972373
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 0.4
Log of Probability for Decision=Yes : -1.3219280948873622
  - p(I) . log2p(I) for Decision=Yes : 0.5287712379549449
Global Entropy for the Dataset : 0.9709505944546686
######################################################################
Step 2: We need to find the most dominant factor for decisioning using gain and
gain ratios.
Features in dataframe : ['Traffic', 'Hunger', 'Lauren']
Computing for Feature=Traffic
{'Busy': {'No': 2, 'Yes': 0}, 'OK': {'No': 1, 'Yes': 1}, 'Chill': {'No': 0,
'Yes': 1}}
----------------------------------------------------------------------
For possible_value=Busy
Total Samples for Entropy Calculation : 2
Number of samples with Decision=No : 2
Probability for Decision=No : 1.0
Log of Probability for Decision=No : 0.0
  - p(I) . log2p(I) for Decision=No : -0.0
Number of samples with Decision=Yes : 0
Probability for Decision=Yes : 0.0
Log of Probability for Decision=Yes : 0
  - p(I) . log2p(I) for Decision=Yes : 0
Entropy for possible_value=Busy : 0.0
Probability of possible_value=Busy : 0.4
----------------------------------------------------------------------
For possible_value=OK
Total Samples for Entropy Calculation : 2
Number of samples with Decision=No : 1
Probability for Decision=No : 0.5
Log of Probability for Decision=No : -1.0
  - p(I) . log2p(I) for Decision=No : 0.5
Number of samples with Decision=Yes : 1
```

```
Probability for Decision=Yes : 0.5
Log of Probability for Decision=Yes : -1.0
 - p(I) . log2p(I) for Decision=Yes : 0.5
Entropy for possible_value=OK : 1.0
Probability of possible_value=OK : 0.4
----------------------------------------------------------------------
For possible_value=Chill
Total Samples for Entropy Calculation : 1
Number of samples with Decision=No : 0
Probability for Decision=No : 0.0
Log of Probability for Decision=No : 0
 - p(I) . log2p(I) for Decision=No : 0
Number of samples with Decision=Yes : 1
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=Chill : 0.0
Probability of possible_value=Chill : 0.2
----------------------------------------------------------------------
Gain for Traffic : 0.5709505944546686
SplitInfo for Traffic : 1.5219280948873621
Gain Ratio for Traffic : 0.37514952012034747
======================================================================
Computing for Feature=Hunger
{'A-little': {'No': 3, 'Yes': 0}, 'A-lot': {'No': 0, 'Yes': 2}}
----------------------------------------------------------------------
For possible_value=A-little
Total Samples for Entropy Calculation : 3
Number of samples with Decision=No : 3
Probability for Decision=No : 1.0
Log of Probability for Decision=No : 0.0
 - p(I) . log2p(I) for Decision=No : -0.0
Number of samples with Decision=Yes : 0
Probability for Decision=Yes : 0.0
Log of Probability for Decision=Yes : 0
 - p(I) . log2p(I) for Decision=Yes : 0
Entropy for possible_value=A-little : 0.0
Probability of possible_value=A-little : 0.6
----------------------------------------------------------------------
For possible_value=A-lot
Total Samples for Entropy Calculation : 2
Number of samples with Decision=No : 0
Probability for Decision=No : 0.0
Log of Probability for Decision=No : 0
 - p(I) . log2p(I) for Decision=No : 0
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
```

```
 - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=A-lot : 0.0
Probability of possible_value=A-lot : 0.4
----------------------------------------------------------------------
Gain for Hunger : 0.9709505944546686
SplitInfo for Hunger : 0.9709505944546686
Gain Ratio for Hunger : 1.0
======================================================================
Computing for Feature=Lauren
{'Available': {'No': 2, 'Yes': 1}, 'Not-available': {'No': 1, 'Yes': 1}}
----------------------------------------------------------------------
For possible_value=Available
Total Samples for Entropy Calculation : 3
Number of samples with Decision=No : 2
Probability for Decision=No : 0.6666666666666666
Log of Probability for Decision=No : -0.5849625007211563
 - p(I) . log2p(I) for Decision=No : 0.38997500048077083
Number of samples with Decision=Yes : 1
Probability for Decision=Yes : 0.3333333333333333
Log of Probability for Decision=Yes : -1.5849625007211563
 - p(I) . log2p(I) for Decision=Yes : 0.5283208335737187
Entropy for possible_value=Available : 0.9182958340544896
Probability of possible_value=Available : 0.6
----------------------------------------------------------------------
For possible_value=Not-available
Total Samples for Entropy Calculation : 2
Number of samples with Decision=No : 1
Probability for Decision=No : 0.5
Log of Probability for Decision=No : -1.0
 - p(I) . log2p(I) for Decision=No : 0.5
Number of samples with Decision=Yes : 1
Probability for Decision=Yes : 0.5
Log of Probability for Decision=Yes : -1.0
 - p(I) . log2p(I) for Decision=Yes : 0.5
Entropy for possible_value=Not-available : 1.0
Probability of possible_value=Not-available : 0.4
----------------------------------------------------------------------
Gain for Lauren : 0.01997309402197489
SplitInfo for Lauren : 0.9709505944546686
Gain Ratio for Lauren : 0.020570659450692974
======================================================================
Gain for Traffic :  0.571
Gain for Hunger :  0.971
Gain for Lauren :  0.020
Gain Ratio for Traffic :  0.375
Gain Ratio for Hunger :  1.000
Gain Ratio for Lauren :  0.021
```

We can see when value of 'Homework' == 'Much', the decision of 'Go-out?' is dependent only on 'Hunger':

1. When value of 'Hunger' == 'A-little', Decision == 'No'
2. When value of 'Hunger' == 'A-lot', Decision == 'Yes'

The Gain Ratio of 'Hunger' feature is 1.

[22]:
```
df_none = df.loc[df['Homework']=='None'].drop(columns=['Homework'])
df_none
```

[22]:

	Traffic	Hunger	Lauren	Decision
3	OK	A-little	Available	Yes
4	Chill	A-lot	Available	Yes
5	Chill	A-lot	Not-available	No
9	OK	A-lot	Available	Yes
13	OK	A-little	Not-available	No

[23]:
```
gain_dict_none, gain_ratio_dict_none = compute_gain_ratios(df_none)

for feature, gain in gain_dict_none.items():
  print("Gain for {} : ".format(feature), '%.3f'% gain)

for feature, gain_ratio in gain_ratio_dict_none.items():
  print("Gain Ratio for {} : ".format(feature), '%.3f'% gain_ratio)
```

```
Step 1: Calculate Global Entropy
Total Samples for Entropy Calculation : 5
Number of samples with Decision=Yes : 3
Probability for Decision=Yes : 0.6
Log of Probability for Decision=Yes : -0.7369655941662062
  - p(I) . log2p(I) for Decision=Yes : 0.44217935649972373
Number of samples with Decision=No : 2
Probability for Decision=No : 0.4
Log of Probability for Decision=No : -1.3219280948873622
  - p(I) . log2p(I) for Decision=No : 0.5287712379549449
Global Entropy for the Dataset : 0.9709505944546686
######################################################################
Step 2: We need to find the most dominant factor for decisioning using gain and
gain ratios.
Features in dataframe : ['Traffic', 'Hunger', 'Lauren']
Computing for Feature=Traffic
{'OK': {'Yes': 2, 'No': 1}, 'Chill': {'Yes': 1, 'No': 1}}
----------------------------------------------------------------------
For possible_value=OK
Total Samples for Entropy Calculation : 3
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 0.6666666666666666
```

```
Log of Probability for Decision=Yes : -0.5849625007211563
 - p(I) . log2p(I) for Decision=Yes : 0.38997500048077083
Number of samples with Decision=No : 1
Probability for Decision=No : 0.3333333333333333
Log of Probability for Decision=No : -1.5849625007211563
 - p(I) . log2p(I) for Decision=No : 0.5283208335737187
Entropy for possible_value=OK : 0.9182958340544896
Probability of possible_value=OK : 0.6
----------------------------------------------------------------------
For possible_value=Chill
Total Samples for Entropy Calculation : 2
Number of samples with Decision=Yes : 1
Probability for Decision=Yes : 0.5
Log of Probability for Decision=Yes : -1.0
 - p(I) . log2p(I) for Decision=Yes : 0.5
Number of samples with Decision=No : 1
Probability for Decision=No : 0.5
Log of Probability for Decision=No : -1.0
 - p(I) . log2p(I) for Decision=No : 0.5
Entropy for possible_value=Chill : 1.0
Probability of possible_value=Chill : 0.4
----------------------------------------------------------------------
Gain for Traffic : 0.01997309402197489
SplitInfo for Traffic : 0.9709505944546686
Gain Ratio for Traffic : 0.020570659450692974
======================================================================
Computing for Feature=Hunger
{'A-lot': {'Yes': 2, 'No': 1}, 'A-little': {'Yes': 1, 'No': 1}}
----------------------------------------------------------------------
For possible_value=A-lot
Total Samples for Entropy Calculation : 3
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 0.6666666666666666
Log of Probability for Decision=Yes : -0.5849625007211563
 - p(I) . log2p(I) for Decision=Yes : 0.38997500048077083
Number of samples with Decision=No : 1
Probability for Decision=No : 0.3333333333333333
Log of Probability for Decision=No : -1.5849625007211563
 - p(I) . log2p(I) for Decision=No : 0.5283208335737187
Entropy for possible_value=A-lot : 0.9182958340544896
Probability of possible_value=A-lot : 0.6
----------------------------------------------------------------------
For possible_value=A-little
Total Samples for Entropy Calculation : 2
Number of samples with Decision=Yes : 1
Probability for Decision=Yes : 0.5
Log of Probability for Decision=Yes : -1.0
 - p(I) . log2p(I) for Decision=Yes : 0.5
```

```
Number of samples with Decision=No : 1
Probability for Decision=No : 0.5
Log of Probability for Decision=No : -1.0
 - p(I) . log2p(I) for Decision=No : 0.5
Entropy for possible_value=A-little : 1.0
Probability of possible_value=A-little : 0.4
----------------------------------------------------------------------
Gain for Hunger : 0.01997309402197489
SplitInfo for Hunger : 0.9709505944546686
Gain Ratio for Hunger : 0.020570659450692974
======================================================================
Computing for Feature=Lauren
{'Available': {'Yes': 3, 'No': 0}, 'Not-available': {'Yes': 0, 'No': 2}}
----------------------------------------------------------------------
For possible_value=Available
Total Samples for Entropy Calculation : 3
Number of samples with Decision=Yes : 3
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Number of samples with Decision=No : 0
Probability for Decision=No : 0.0
Log of Probability for Decision=No : 0
 - p(I) . log2p(I) for Decision=No : 0
Entropy for possible_value=Available : 0.0
Probability of possible_value=Available : 0.6
----------------------------------------------------------------------
For possible_value=Not-available
Total Samples for Entropy Calculation : 2
Number of samples with Decision=Yes : 0
Probability for Decision=Yes : 0.0
Log of Probability for Decision=Yes : 0
 - p(I) . log2p(I) for Decision=Yes : 0
Number of samples with Decision=No : 2
Probability for Decision=No : 1.0
Log of Probability for Decision=No : 0.0
 - p(I) . log2p(I) for Decision=No : -0.0
Entropy for possible_value=Not-available : 0.0
Probability of possible_value=Not-available : 0.4
----------------------------------------------------------------------
Gain for Lauren : 0.9709505944546686
SplitInfo for Lauren : 0.9709505944546686
Gain Ratio for Lauren : 1.0
======================================================================
Gain for Traffic :  0.020
Gain for Hunger :  0.020
Gain for Lauren :  0.971
Gain Ratio for Traffic :  0.021
```

```
Gain Ratio for Hunger :  0.021
Gain Ratio for Lauren :  1.000
```

We can see when value of 'Homework' == 'None', the decision of 'Go-out?' is dependent only on 'Lauren':

1. When value of 'Lauren' == 'Available', Decision == 'Yes'
2. When value of 'Lauren' == 'Not-available', Decision == 'No'

The Gain Ratio of 'Lauren' feature is 1.

[24]:
```python
df_normal = df.loc[df['Homework']=='Normal'].drop(columns=['Homework'])
df_normal
```

[24]:

	Traffic	Hunger	Lauren	Decision
2	Busy	A-little	Available	Yes
6	Chill	A-lot	Not-available	Yes
11	OK	A-little	Not-available	Yes
12	Busy	A-lot	Available	Yes

[25]:
```python
gain_dict_normal, gain_ratio_dict_normal = compute_gain_ratios(df_normal)

for feature, gain in gain_dict_normal.items():
  print("Gain for {} : ".format(feature), '%.3f'% gain)

for feature, gain_ratio in gain_ratio_dict_normal.items():
  print("Gain Ratio for {} : ".format(feature), '%.3f'% gain_ratio)
```

```
Step 1: Calculate Global Entropy
Total Samples for Entropy Calculation : 4
Number of samples with Decision=Yes : 4
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Global Entropy for the Dataset : 0.0
######################################################################
Step 2: We need to find the most dominant factor for decisioning using gain and
gain ratios.
Features in dataframe : ['Traffic', 'Hunger', 'Lauren']
Computing for Feature=Traffic
{'Busy': {'Yes': 2}, 'Chill': {'Yes': 1}, 'OK': {'Yes': 1}}
----------------------------------------------------------------------
For possible_value=Busy
Total Samples for Entropy Calculation : 2
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=Busy : 0.0
```

```
Probability of possible_value=Busy : 0.5
----------------------------------------------------------------------
For possible_value=Chill
Total Samples for Entropy Calculation : 1
Number of samples with Decision=Yes : 1
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=Chill : 0.0
Probability of possible_value=Chill : 0.25
----------------------------------------------------------------------
For possible_value=OK
Total Samples for Entropy Calculation : 1
Number of samples with Decision=Yes : 1
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=OK : 0.0
Probability of possible_value=OK : 0.25
----------------------------------------------------------------------
Gain for Traffic : 0.0
SplitInfo for Traffic : 1.5
Gain Ratio for Traffic : 0.0
======================================================================
Computing for Feature=Hunger
{'A-little': {'Yes': 2}, 'A-lot': {'Yes': 2}}
----------------------------------------------------------------------
For possible_value=A-little
Total Samples for Entropy Calculation : 2
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=A-little : 0.0
Probability of possible_value=A-little : 0.5
----------------------------------------------------------------------
For possible_value=A-lot
Total Samples for Entropy Calculation : 2
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
 - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=A-lot : 0.0
Probability of possible_value=A-lot : 0.5
----------------------------------------------------------------------
Gain for Hunger : 0.0
SplitInfo for Hunger : 1.0
Gain Ratio for Hunger : 0.0
```

```
==========================================================================
Computing for Feature=Lauren
{'Available': {'Yes': 2}, 'Not-available': {'Yes': 2}}
--------------------------------------------------------------------------
For possible_value=Available
Total Samples for Entropy Calculation : 2
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
  - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=Available : 0.0
Probability of possible_value=Available : 0.5
--------------------------------------------------------------------------
For possible_value=Not-available
Total Samples for Entropy Calculation : 2
Number of samples with Decision=Yes : 2
Probability for Decision=Yes : 1.0
Log of Probability for Decision=Yes : 0.0
  - p(I) . log2p(I) for Decision=Yes : -0.0
Entropy for possible_value=Not-available : 0.0
Probability of possible_value=Not-available : 0.5
--------------------------------------------------------------------------
Gain for Lauren : 0.0
SplitInfo for Lauren : 1.0
Gain Ratio for Lauren : 0.0
==========================================================================
Gain for Traffic :  0.000
Gain for Hunger :  0.000
Gain for Lauren :  0.000
Gain Ratio for Traffic :  0.000
Gain Ratio for Hunger :  0.000
Gain Ratio for Lauren :  0.000
```

We can see when value of 'Homework' == 'Normal', the decision of 'Go-out?' is always 'Yes'.

We get no information gain from the remaining features.

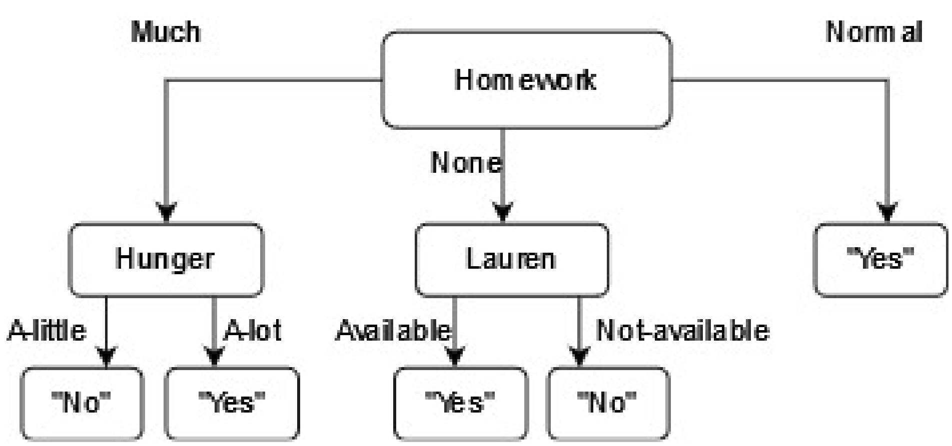
Therefore, using the above inferences we can create our decision tree and also create a function comprising of an if-else ladder to predict the decision.

[37]:
```python
def findDecision(obj): #obj[0]: Homework, obj[1]: Traffic, obj[2]: Hunger, obj[3]: Lauren
        # {"feature": "Homework", "instances": 14, "metric_value": 0.9403, "depth": 1}
        if obj[0] == 'Much':
                # {"feature": "Hunger", "instances": 5, "metric_value": 0.971, "depth": 2}
                if obj[2] == 'A-little':
                        return 'No'
```

```
                elif obj[2] == 'A-lot':
                        return 'Yes'
                else: return 'Yes'
        elif obj[0] == 'None':
                # {"feature": "Lauren", "instances": 5, "metric_value": 0.971,␣
↪"depth": 2}
                if obj[3] == 'Available':
                        return 'Yes'
                elif obj[3] == 'Not-available':
                        return 'No'
                else: return 'No'
        elif obj[0] == 'Normal':
                return 'Yes'
        else: return 'Yes'
```

[38]:
```
from PIL import Image               # to load images
from IPython.display import display # to display images

pil_im = Image.open('/content/gdrive/My Drive/ECE_5424_AML/HW2/
↪DecisionTreeImages/DecisionTree.jpg')
display(pil_im)
```



Q4. (2 points) Today, you have a normal amount of homework and are very hungry. However, traffic is busy because it's Hokie game day, and Lauren is not available to hang out with you. According to your decision tree, are you going out?

Answer:

According to our decision tree model, we will get the prediction as "Yes" for going out.

[39]:
```
answer = findDecision(['Normal', 'Busy', 'A-lot', 'Not-available'])
answer
```

[39]: 'Yes'

[40]:
```
!jupyter nbconvert --to PDF "/content/gdrive/MyDrive/ECE_5424_AML/HW2/
↪ankitparekh-Problem1-SectionA-HW2.ipynb"
```

```
[NbConvertApp] Converting notebook
/content/gdrive/MyDrive/ECE_5424_AML/HW2/ankitparekh-Problem1-SectionA-HW2.ipynb
to PDF
[NbConvertApp] Support files will be in ankitparekh-Problem1-SectionA-HW2_files/
[NbConvertApp] Making directory ./ankitparekh-Problem1-SectionA-HW2_files
[NbConvertApp] Writing 59437 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 94336 bytes to
/content/gdrive/MyDrive/ECE_5424_AML/HW2/ankitparekh-Problem1-SectionA-HW2.pdf
```

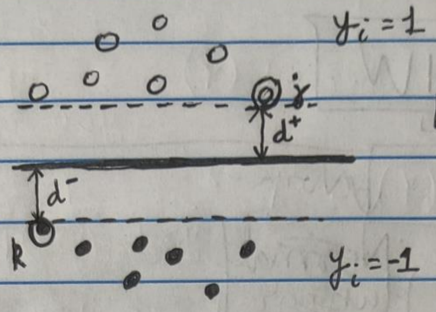
Problem 2. Support Vector Machines (15 points total)

Solutions:

A1)

Problem 2: Q1) Solution:

Given:



Linearly Seperable Data

For the class $y_i = 1$, we have $w^T x_i + b \geq 1$

$$\Rightarrow w^T x_j + b = 1 \quad \{x_j, y_j \text{ are in the support vector}\}$$

$$\Rightarrow b = 1 - w^T x_j$$

For the class $y_i = -1$, we have $w^T x_i + b \leq -1$

$$\Rightarrow w^T x_k + b = -1 \Rightarrow b = -1 - w^T x_k$$

$$d^+ - d^- = \frac{w^T x_j}{\|w\|_2} - \frac{w^T x_k}{\|w\|_2}$$

$$d^+ - d^- = \frac{1-b}{\|w\|_2} - \frac{(-1-b)}{\|w\|_2} = \frac{2}{\|w\|_2}$$

We can write the problem as $\boxed{\max_{w,b} \frac{2}{\|w\|_2}}$ since we want to maximize this value.

for $y_i(w^T x_i + b) \geq 1$
$\forall i \in 1, \dots, N$

Or we can write this alternatively like below:

$$\boxed{\min \frac{1}{2} \|w\|^2 \text{ such that } y_i(w^T x_j + b) \geq 1 \ \forall i \in 1, \dots, N}$$

The above equation is a quadratic programming problem with a set of linear inequality constraints.

To Solve the equation, we need to introduce Lagrange multipliers with one multiplier for each of the constraints where $(a_n \geq 0)$,

$$L(w,b,a) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} a_n\{t_n(w^T\phi(x_n)+b)-1\}$$

To minimize L, we take partial derivatives :-

$$\boxed{\begin{aligned} \frac{\partial L}{\partial w} &= w - \sum_{n=1}^{N} a_n t_n \phi(x_n) = 0 \Rightarrow w = -\sum_{n=1}^{N} a_n t_n \phi(x_n) \quad \ldots \text{(I)} \\ \frac{\partial L}{\partial b} &= \sum_{n=1}^{N} a_n t_n = 0 \quad \ldots \text{(II)} \end{aligned}}$$

Using (I) & (II): we get

$$\tilde{L}(w,b,a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m)$$

subject to $a_n \geq 0$

$$\sum_{n=1}^{N} a_n t_n = 0$$

where

$k(x,x') = \phi(x)^T \phi(x')$

Kernel function

A2)

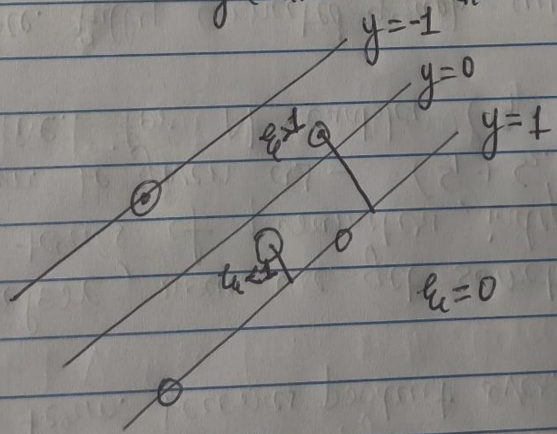
Problem2) Q2) Solution:

* We introduce slack variables when the data points cannot be seperated using a "hard-margin". These account for the misclassification, and we try to minimize these misclassifications using earlier optimization $\frac{1}{2}||w||^2$.

* We introduce one slack variable for each training data point $\xi_n \geq 0$ $n = 1, \ldots, N$

* For correctly classified data points: $\xi_n = 0$
  All other data points: $\xi_n = |t_n - y(x_n)| > 1$

* For data point that is on the decision boundary:
  $y(x_n) = 0$: $\xi_n = 1$



* For points which lie inside the margin, but on the correct side: $0 < \xi_n <= 1$

* We would want to maximize the margin while penalizing misclassifications:

$$\min \boxed{C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||w||^2}$$ such that

$t_n y(x_n) \geq 1 - \xi_n$

where

$\xi_n \geq 0$

$n = 1, \ldots, N$

Lagrangian:

$$L(w, b, a) = \frac{1}{2}||w||^2 + C\sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} a_n \{t_n y(x_n) - 1 + \xi_n\} - \sum_{n=1}^{N} \mu_n \xi_n$$

where $\underbrace{a_n \geq 0, \ \mu_n \geq 0}_{\text{Lagrangian Multipliers}}$

To optimize for w, b, and $\{\xi_n\}$, we get:

$$\frac{\partial L}{\partial w} = 0 \Rightarrow \boxed{w = \sum_{n=1}^{N} a_n t_n \phi(x_n)} \quad \ldots \text{I}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \boxed{\sum_{n=1}^{N} a_n t_n = 0} \quad \ldots \text{II}$$

$$\frac{\partial L}{\partial \xi_n} = 0 \Rightarrow \boxed{a_n = C - \mu_n} \quad \ldots \text{III}$$

Using I, II, & III:

minimize

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m)$$

subject to $a_n \geq 0$, $\mu_n \geq 0$, $a_n \leq C$, $\sum_{n=1}^{N} a_n t_n = 0$

A3)

SVM are inherently binary classifiers and when used in the most basic fashion do not support multi-class classification. But it is possible to use SVM for multiclass classification by breaking down the multi-class classification problem into smaller binary classification tasks. Some of the approaches to convert a multi-class problem to a set of 2-class classification problems using an SVM are One vs One (OVO), One vs Rest (OVR), and Directed Acyclic Graph (DAG).

In the One vs One (OVO) approach, we break down the multi-class problem into all the possible class pairs and train a binary classifier (SVM) for each pair of classes (a total of M(M-1)/2 SVMs, where M is the number of classes). For a specific pair of classes, we neglect all the rest of the classes and find the hyperplane that separates those two classes. We use Majority Voting along with the distance from the margin for the final prediction on any input. One of the drawbacks of this approach is we have to train a lot of SVM models.

In the One vs Rest (OVR) approach, if we have a M-class problem we train M SVM models. We train the SVM by picking a specific class as the first one and putting all the other classes into the second one. For the final prediction on the input, we predict with each of the M SVMs and find out the one which puts the prediction into the farthest positive region. The major drawbacks of this approach are extensive computation as we are considering the whole dataset (all the data points) for training each SVM, and unbalanced dataset as we are combining (M-1) classes into 1 class.

The Directed Acyclic Graph (DAG) is a graphical approach that addresses the problems of the above two approaches by grouping classes based on some logical grouping and is hierarchical in nature. It requires lesser SVMs to train with respect to the OVO approach and reduces the diversity from the larger unbalanced class which is a drawback of the OVR approach.

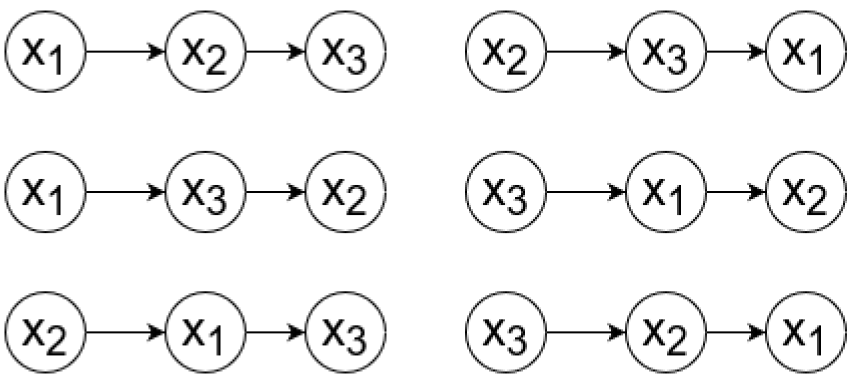
Problem 3. Probabilistic Graphical Models (25 points total)

Solutions:

A1)

1. One can graph a total of **N!** distinct networks with **N** random variables. Each distinct permutation of the random variables can be used to create a distinct network.

Ex: **For N=3**, we get a total of 3! = 6 different networks that could be graphed below:



We get a different joint distribution of the 3 random variables for each of the above networks, and therefore we can have N! distinct networks graphed out for N random variables.

2.

Problem 3) Q1)

2. Given: N random variables
each with s_i states, $s \in \mathbb{Z}$, $s \geq 2$

For discrete variables,
the probability distribution $p(x/\mu)$ for single discrete variable x having K possible states is given by:

$$p(x/\mu) = \prod_{k=1}^{K} \mu_k^{x_k}$$

For 2 discrete variables x_1 & x_2 for states s_1 & s_2:

$$P(x_1, x_2/\mu) = \prod_{k=1}^{S_1} \prod_{l=1}^{S_2} \mu_{kl}^{x_{1k} x_{2l}}$$

This distribution has $(S_1 \cdot S_2 - 1)$ parameters.

∴ For N distinct variables, the total number of parameters that must be specified for an arbitrary joint distribution is $\{(S_1 \times S_2 \times \ldots \times S_N) - 1\}$.

This is for a fully connected graph & grows exponentially in the number of variables N & different possible state that each variable can have.

∴ General joint distribution will vary exponentially & this becomes impractical.

If we remove the link & assume conditional independence, then for 2 random variables,

$$p(x_1, x_2|\mu) = \prod_{k=1}^{S_1} \mu_{1k}^{x_{1k}} \prod_{l=1}^{S_2} \mu_{2l}^{x_{2l}}$$

This term becomes product over independent terms.

Therefore, now we get something that grows linearly in terms of parameters, independent joint distribution can be given as – $(S_1 + S_2 - 1)$
i.e. for N variables $(S_1 + S_2 + \ldots + S_N - 1)$

Hence fully connected graph will have exponential parameters & will be too complex.

The joint distribution becomes too simple if there are no links.

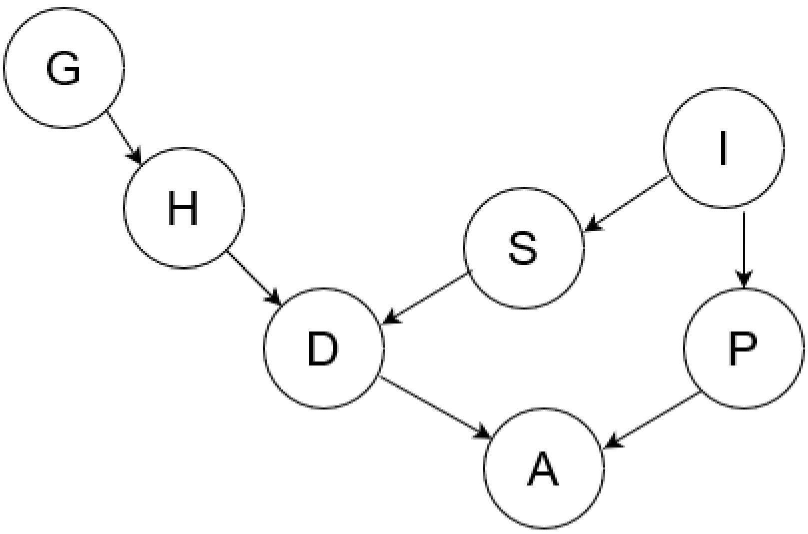
This provides motivation for conditional independence as it will require practically feasible number of parameters compared to fully connected distribution.

We can reduce the number of independent parameters needed if we assume that some random variables are conditionally independent given other variables.

A2)

1.

Using the events (random variables) given in the question, we get the below Bayesian Network:



The joint probability **p(G, H, D, S, I, P, A) = p(G) p(H|G) p(D|H,S) p(I) p(S|I) p(P|I) p(A|D,P)**

S’s Markov Blanket contains nodes **H, D, I.**

For the following questions, you are to determine whether these conditional independence statements are true or not, and what they intuitively mean accordingly.

2. $S \perp\!\!\!\perp P \mid I$.

Answer: **True**

Explanation: **Node A is blocked as it is a head-to-head node. Since, I is a tail-to-tail node, and it is given, it is blocked. All paths from S to P are blocked, therefore they are independent.**

Intuition: **If we are given insomnia, the probability that we have a lack of sleep and the probability that we do not pay attention are independent of each other.**

3. $H \perp\!\!\!\perp I \mid A$.

Answer: **False**

Explanation: **Although D is a head-to-head node, its descendant A is given so it is unblocked. There is an unblocked path from H to I (H > D < S < I).**

Intuition: **The probability that we contract the Hokie plague and the probability that we have insomnia are dependent on each other even if we know whether we completed the assignment on time.**

4. $G \perp\!\!\!\perp I \mid S$.

Answer: **True**

Explanation: **Node A is blocked as it is a head-to-head node. S is a head-to-tail node, and it is given, it is blocked. All paths from G to I are blocked, therefore they are independent.**

Intuition: **The probability that we go out and the probability that we have insomnia become independent of each other if lack of sleep is given.**

5. $G \perp\!\!\!\perp I \mid S, A$.

Answer: **False**

Explanation: **Although A is a head-to-head node, it is given so it is unblocked. There is an unblocked path from G to I (G > H > D > A< P < I).**

Intuition: **The probability that we go out and the probability that we have insomnia are dependent on each other if we know whether we completed the assignment on time and whether we had lack of sleep.**