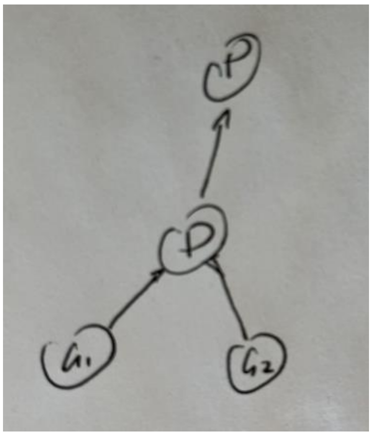
Homework 7

1.

(a)

The Bayes Net over those variables should be



With the probability

$$Pr(G_1, G_2, D, P) = Pr(G_1)\,Pr(G_2)\,Pr(D|G_1, G_2)Pr(P|D)$$

And corresponding conditional probability tables are

G_1	G_2	$Pr(D = 1|G_1, G_2)$
0	0	0
0	1	0.1
1	0	0.2
1	1	0.5

and

D	$Pr(P = 1|D)$
0	0.01
1	0.5

And of course, besides the conditional probability table, we can also obtain probability table for G_1 and G_2 that can be used for following questions.

G_1	$Pr(G_1)$
0	0.9
1	0.1

G_2	$Pr(G_2)$
0	0.8
1	0.2

(b)

The true statements are

$$G_1 \perp\!\!\!\perp G_2$$

$$G_1 \perp\!\!\!\perp P|D$$

(c)

$$Pr(D = 1) = \sum_{G_1}\sum_{G_2} Pr(G_1)\,Pr(G_2)\,\Pr(D = 1|G_1, G_2)$$

$$= 0.044$$

(d)

$$Pr(P = 1) = \sum_{D} \Pr(P = 1|D) = 0.03156$$

(e)

According to Bayes' formula,

$$Pr(D=1|P=1)=\frac{Pr(D=1)\,Pr(P=1|D=1)}{Pr(P=1)}=0.697$$

(f)

Similarly, we have

$$Pr(G_1=1|D=1)=\frac{Pr(G_1=1)\,Pr(D=1|G_1=1)}{Pr(D=1)}$$

and

$$Pr(D=1|G_1=1)=0.2\cdot 0.8+0.5\cdot 0.2=0.26$$

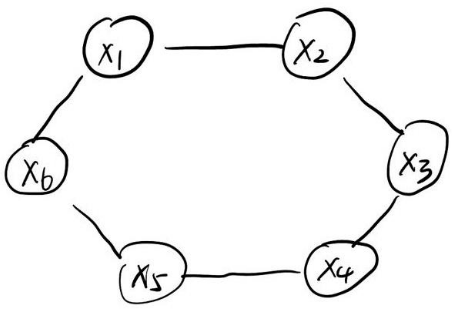
Therefore,

$$Pr(G_1=1|D=1)=0.591$$

2.

(a)

The graph should be



(b)

Since the function is

$$\psi(a,b) = \begin{cases} 1, a = b \\ 0.5, a \neq b \end{cases}$$

Therefore, to maximize $P(x)$, the all x_i should be the same.

Hence, the settings should be all 0 or all 1. That is to say, $x_i = 1$ or $x_i = 0$ for $1 \leq i \leq 6$

(c)

Similarly, the settings can be either $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 0$ or $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 0, x_6 = 1$ to minimize $P(x)$

3.

(a)

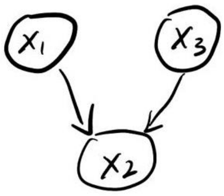
From given table

x_1	x_2	x_3	Pr
0	0	0	1/3
0	0	1	1/3
1	0	0	1/6
1	1	1	1/6

We can observe that x_2 is a deterministic function of the other two.

(b)

The Bayes net can be shown as



(c)

Based on above graph, we can firstly obtain that x_1 have no parents ($\Pi_1 = \{\ \}$), thus

x_1	$Pr(x_1)$
0	1/3+1/3=2/3
1	1/6+1/6=1/3

Same for $x_3(\Pi_3 = \{\ \})$

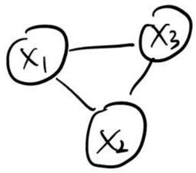
x_3	$Pr(x_3)$
0	1/3+1/6=1/2
1	1/3+1/6=1/2

As for x_2, it got two parents which is $\Pi_2 = \{x_1, x_3\}$, thus its conditional probability table shown as

x_1	x_2	x_3	$Pr(x_2 \mid x_1, x_3)$
0	0	0	1
0	0	1	1
1	0	0	1
1	1	1	1

(d)

The undirected graph is

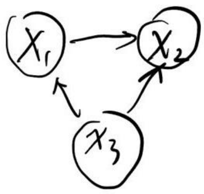


(e)

From my perspectives, the relations should be

$$P \subseteq P'$$

It means that any distributions over G can be represented by distributions over G'. In other words, any distribution represented by the Bayesian network G can also be represented by the undirected graph. However, there may exist distributions that can be represented by G' but cannot be represented by G. For instance,



The G' can represent relations in above graph but G cannot. Therefore, my conclusion is

$$P \subseteq P'$$

CSE291 HW7

(a)

To describe how I got my embeddings, I will add descpritions for all cells and how they come to the final embeddings.

Firstly, download the nltk package as I need

In [1]:
```
pip install nltk
```

```
Looking in indexes: http://mirrors.aliyun.com/pypi/simple/Note: (http://mirrors.aliyun.com/pypi/simple/Note:) you may need to re
start the kernel to use updated packages.

Requirement already satisfied: nltk in f:\anaconda\anaconda2\lib\site-packages (3.6.1)
Requirement already satisfied: joblib in f:\anaconda\anaconda2\lib\site-packages (from nltk) (1.0.1)
Requirement already satisfied: click in f:\anaconda\anaconda2\lib\site-packages (from nltk) (7.1.2)
Requirement already satisfied: regex in f:\anaconda\anaconda2\lib\site-packages (from nltk) (2021.4.4)
Requirement already satisfied: tqdm in f:\anaconda\anaconda2\lib\site-packages (from nltk) (4.59.0)
```

In [2]:
```
import nltk #import nltk
```

I tried download brown directly, but it failed, here's how I got the path of nltk and after that I downloaded manually.

In [3]:
```
nltk.data.path
```

Out[3]:
```
['C:\\Users\\Lenovo/nltk_data',
 'F:\\anaconda\\anaconda2\\nltk_data',
 'F:\\anaconda\\anaconda2\\share\\nltk_data',
 'F:\\anaconda\\anaconda2\\lib\\nltk_data',
 'C:\\Users\\Lenovo\\AppData\\Roaming\\nltk_data',
 'C:\\nltk_data',
 'D:\\nltk_data',
 'E:\\nltk_data']
```

after downloading, I import it and check it

In [4]:
```
from nltk.corpus import brown

words = brown.words()
```

In [5]:
```
len(words)
```

Out[5]:
```
1161192
```

In [6]:
```
type(words)
```

Out[6]:
```
nltk.corpus.reader.util.ConcatenatedCorpusView
```

Get a string of brown words for later use.

In [7]:
```
Words=' '.join(words)
```

Import all the packages I might use

In [8]:
```
import string
import re
from collections import Counter
from nltk.corpus import stopwords
from nltk import punkt
import math
import numpy as np
import pandas as pd
import random
import sklearn
import scipy
from pylab import rcParams
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import pairwise_distances
```

For this cell, I use two funtions. First one is for processing, which removes punctuation and stopwords and those non-alphabet words I think might not help for embeddings. Second one is for counting the frequency for each word in processed text streams. By using these two functions, get data cleaned.

In [9]:
```
#the methods for processing my data
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuation
    text = text.translate(str.maketrans("","", string.punctuation))

    # Tokenize the text into words
    words = nltk.word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]

    # Remove non-alphabetic characters using regex
    words = [re.sub('[^a-zA-Z]', '', word) for word in words if word]

    return words

def count_word_frequency(words):
    # Count word frequencies
    word_counts = Counter(words)

    return word_counts

# Example usage

words = preprocess_text(Words)
words=[word for word in words if word!='']
word_frequencies = count_word_frequency(words)
```

This cell, is get the whole list sorted in the order of frequency, the first 10 is shown.

In [10]:
```
def comp(x):
    return x[1]


theWholeList=[(i,j) for i,j in word_frequencies.items()]
theWholeList.sort(key=comp,reverse=True)
theWholeList[:10]
```

Out[10]:
```
[('one', 3297),
 ('would', 2714),
 ('said', 1961),
 ('new', 1635),
 ('could', 1601),
 ('time', 1598),
 ('two', 1412),
 ('may', 1402),
 ('first', 1361),
 ('like', 1292)]
```

For what I've done now, I got the whole list for words right now. Using that list, below is how I got short list C and vocabulary V by selecting first 100 (since 100 dimensional for later embeddings) and first 5000. Since the list is in order of frequency, this way, fits the requirement.

In [11]:
```
C={word:num for word,num in theWholeList[:100]}   #A shorter list C which we shall call context words.
V={word:num for word,num in theWholeList[:5000]}    #A vocabulary V , consisting of a few thousand of the most commonly-occurringw
```

For this cell, is doing window conut. That is to say, like mentioned in the question, get four words around the word in vocabulary which are two before the word w and two after and all of them in short list C.

In [12]:
```
context_word_counts = {word: Counter() for word in V}
window_size = 2

for i, word in enumerate(words):
    if word in V:
        start = max(0, i - window_size)
        end = min(i + window_size + 1, len(words))
        context = [words[j] for j in range(start, end) if (i!=j and words[j] in C)]
        #print(context)
        context_word_counts[word].update(context)
```

here is one example.

In [13]:

```
context_word_counts['one']
```

Out[13]:
```
Counter({'make': 31,
         'first': 36,
         'two': 86,
         'three': 24,
         'years': 23,
         'would': 77,
         'time': 66,
         'united': 4,
         'men': 16,
         'said': 54,
         'well': 25,
         'number': 13,
         'get': 22,
         'people': 24,
         'states': 3,
         'go': 16,
         'since': 10,
         'may': 49,
         'another': 108,
         'one': 84,
         'last': 27,
         'every': 19,
         'take': 24,
         'went': 11,
         'school': 11,
         'made': 25,
         'never': 28,
         'year': 38,
         'new': 25,
         'good': 37,
         'day': 60,
         'used': 9,
         'way': 45,
         'small': 14,
         'come': 14,
         'see': 17,
         'also': 21,
         'found': 18,
         'left': 19,
         'world': 13,
         'must': 40,
         'house': 16,
         'without': 13,
         'head': 13,
         'use': 14,
         'back': 19,
         'away': 9,
         'might': 37,
         'think': 14,
         'say': 27,
         'man': 62,
         'however': 14,
         'state': 19,
         'long': 14,
         'war': 11,
         'little': 15,
         'thought': 16,
         'home': 13,
         'like': 36,
         'us': 26,
         'took': 18,
         'still': 16,
         'hand': 40,
         'fact': 17,
         'part': 20,
         'mr': 9,
         'enough': 11,
         'great': 18,
         'public': 10,
         'work': 21,
         'many': 19,
         'know': 22,
         'much': 19,
         'got': 15,
         'almost': 14,
         'american': 14,
         'even': 28,
         'could': 56,
         'life': 16,
         'something': 8,
         'course': 20,
         'place': 12,
         'came': 11,
         'yet': 6,
         'always': 14,
         'less': 8,
         'far': 11,
         'right': 8,
```

```
'old': 17,
'government': 3,
'water': 5,
'high': 10,
'dont': 4,
'though': 12,
'around': 12,
'general': 3,
'mrs': 9,
'upon': 6,
'put': 5,
'af': 6})
```

This part is computing probablity. Using 'context_word_counts', we can get conditional probablity by computing their appreance with each word w over all words aperance around this specific word w. And computing Pr(c) by simply counting its apperance in all words.

In [14]:
```
#compute Pr(c|w)
conditional_probabilities = {}

for word in V:
    context_counts = context_word_counts[word]

    total_count = sum(context_counts.values())


    conditional_probabilities[word] = {}
    for context_word in context_counts:
        prob = context_counts[context_word] / total_count
        conditional_probabilities[word][context_word] = prob

#-------------------------------------------------------------------------------------------------------------------
# compute Pr(c)
context_word_total_counts = Counter()

for word in V:
    context_word_total_counts.update(context_word_counts[word])

Prc = {}

total_count = sum(context_word_total_counts.values())

for context_word in C:
    prob = context_word_total_counts[context_word] / total_count
    Prc[context_word] = prob
```

In [15]:
```
conditional_probabilities['one']
```

Out[15]:
```
{'make': 0.013796172674677348,
 'first': 0.01602136181575434,
 'two': 0.03827325322652426,
 'three': 0.010680907877169559,
 'years': 0.010235870048954161,
 'would': 0.03426791277258567,
 'time': 0.029372496662216287,
 'united': 0.0017801513128615932,
 'men': 0.007120605251446373,
 'said': 0.02403204272363151,
 'well': 0.011125945705384957,
 'number': 0.005785491766800178,
 'get': 0.009790832220738763,
 'people': 0.010680907877169559,
 'states': 0.0013351134846461949,
 'go': 0.007120605251446373,
 'since': 0.004450378282153983,
 'may': 0.021806853582554516,
 'another': 0.04806408544726302,
 'one': 0.037383177570093455,
 'last': 0.012016021361815754,
 'every': 0.008455718736092568,
 'take': 0.010680907877169559,
 'went': 0.004895416110369382,
 'school': 0.004895416110369382,
 'made': 0.011125945705384957,
 'never': 0.012461059190031152,
 'year': 0.016911437472185136,
 'new': 0.011125945705384957,
 'good': 0.016466399643969738,
 'day': 0.0267022696929239,
 'used': 0.004005340453938585,
 'way': 0.020026702269692925,
 'small': 0.006230529595015576,
 'come': 0.006230529595015576,
 'see': 0.0075656430796617715,
 'also': 0.009345794392523364,
 'found': 0.00801068090787717,
 'left': 0.008455718736092568,
 'world': 0.005785491766800178,
 'must': 0.017801513128615932,
 'house': 0.007120605251446373,
 'without': 0.005785491766800178,
 'head': 0.005785491766800178,
 'use': 0.006230529595015576,
 'back': 0.008455718736092568,
 'away': 0.004005340453938585,
 'might': 0.016466399643969738,
 'think': 0.006230529595015576,
 'say': 0.012016021361815754,
 'man': 0.027592345349354695,
 'however': 0.006230529595015576,
 'state': 0.008455718736092568,
 'long': 0.006230529595015576,
 'war': 0.004895416110369382,
 'little': 0.006675567423230975,
 'thought': 0.007120605251446373,
 'home': 0.005785491766800178,
 'like': 0.01602136181575434,
 'us': 0.011570983533600357,
 'took': 0.00801068090787717,
 'still': 0.007120605251446373,
 'hand': 0.017801513128615932,
 'fact': 0.0075656430796617715,
 'part': 0.008900756564307966,
 'mr': 0.004005340453938585,
 'enough': 0.004895416110369382,
 'great': 0.00801068090787717,
 'public': 0.004450378282153983,
 'work': 0.009345794392523364,
 'many': 0.008455718736092568,
 'know': 0.009790832220738763,
 'much': 0.008455718736092568,
 'got': 0.006675567423230975,
 'almost': 0.006230529595015576,
 'american': 0.006230529595015576,
 'even': 0.012461059190031152,
 'could': 0.024922118380062305,
 'life': 0.007120605251446373,
 'something': 0.0035603026257231864,
 'course': 0.008900756564307966,
 'place': 0.005340453938585847796,
 'came': 0.004895416110369382,
 'yet': 0.0026702269692923898,
 'always': 0.006230529595015576,
 'less': 0.0035603026257231864,
 'far': 0.004895416110369382,
 'right': 0.0035603026257231864,
```

```
 'old': 0.0075656430796617715,
 'government': 0.0013351134846461949,
 'water': 0.0022251891410769915,
 'high': 0.004450378282153983,
 'dont': 0.0017801513128615932,
 'though': 0.0053404539385847796,
 'around': 0.0053404539385847796,
 'general': 0.0013351134846461949,
 'mrs': 0.004005340453938585,
 'upon': 0.0026702269692923898,
 'put': 0.0022251891410769915,
 'af': 0.0026702269692923898}
```

In [16]:
```
Prc['one']
```

Out[16]:
```
0.04286076484259455
```

Now, with all the distributions, we compute the embeddings as what is instrcuted by choosing max number of 0 and log(Pr(c|w)/Pr(c)) for each entry.

In [17]:
```
word_embeddings = {word: [0] * len(C) for word in V}

for word in V:
    for i, context_word in enumerate(C):
        if context_word== word:continue
        #print(word+','+context_word)
        prob_c_w = conditional_probabilities[word][context_word] if context_word in conditional_probabilities[word] else 1e-9
        prob_c = Prc[context_word] if Prc[context_word]!=0 else 1e-9
        word_embeddings[word][i] = max(0, math.log(prob_c_w / prob_c))
```

Above is all the steps of how I got my embeddings.

(b)

In [18]:
```
def cosine_similarity(embedding_w, embedding_w0):
    dot_product = np.dot(embedding_w, embedding_w0)
    norm_w = np.linalg.norm(embedding_w)
    norm_w0 = np.linalg.norm(embedding_w0)
    cos_sim = dot_product / (norm_w * norm_w0)
    return cos_sim

def pickWords(theWholeList):
    random_numbers = random.sample([i for i in range(5000)], 25)
    wordlist=[theWholeList[num][0] for num in random_numbers]
    return wordlist

wordlist=pickWords(theWholeList)
wordpair=[]
for word in wordlist:
    tmp=None
    sim=-1
    for cmp_word in V:
        if cmp_word==word:continue
        new_sim=cosine_similarity(word_embeddings[word], word_embeddings[cmp_word])
        if new_sim>sim:
            sim=new_sim
            tmp=cmp_word
    wordpair.append((word,tmp,sim))
```

```
<ipython-input-18-99f9bd50444b>:5: RuntimeWarning: invalid value encountered in double_scalars
  cos_sim = dot_product / (norm_w * norm_w0)
```

```
In [19]: wordpair
```

```
Out[19]: [('net', 'published', 0.5283024425195768),
 ('stress', 'food', 0.5921562496091787),
 ('rehabilitation', 'earliest', 0.5686828458817361),
 ('luxury', 'strategic', 0.595100590454909),
 ('moral', 'meaning', 0.5768974326809412),
 ('accurately', 'carry', 0.49920045735376894),
 ('meant', 'heard', 0.6154985035173524),
 ('listeners', 'passages', 0.5680268439666191),
 ('running', 'protest', 0.6102340621641896),
 ('equal', 'increased', 0.5634571984172476),
 ('bobby', 'midnight', 0.5617157954180494),
 ('deegan', 'hit', 0.5680495091957429),
 ('worlds', 'source', 0.49370578763244316),
 ('initial', 'food', 0.5154134586443299),
 ('oral', 'obviously', 0.5693072506462691),
 ('surrounded', 'speaking', 0.5079587285178777),
 ('merely', 'involved', 0.6168889069028188),
 ('lacked', 'favorable', 0.5913706468340824),
 ('light', 'solution', 0.5903190171148776),
 ('shoulder', 'pressed', 0.662694812749987),
 ('possibilities', 'ordinary', 0.5845108887687327),
 ('receive', 'establish', 0.5844267530708848),
 ('feels', 'holy', 0.5242045013747196),
 ('sit', 'station', 0.5817584590709062),
 ('authors', 'critics', 0.5576088390183671)]
```

Well, for this result, from my perspectives, I have to some make sense, some not. For instance, some you can tell the link, like authors and critics, because some author write critical things in their work; sit and station, one can sit at the bus station waiting for his or her bus. But some are meaningless, like oral and obviously, lacked and favorable, some I cannot see the strong relations.

(c)

```
In [20]: embeddings=[word_embeddings[word] for word in V]
cosine_similarities = pairwise_distances(embeddings, metric='cosine')
kmeans = KMeans(n_clusters=100, random_state=0)
clusters = kmeans.fit_predict(cosine_similarities)
```

```
In [21]: kmeans.labels_
```

```
Out[21]: array([61, 86, 86, ..., 17, 99, 10])
```

```
In [22]: cluster_dict = {i: [] for i in range(100)}
V_list=[word for word in V]
for i, label in enumerate(kmeans.labels_):
    cluster_dict[label].append(V_list[i])
```

```
In [23]: cluster_dict
```

```
  'tsunami',
  'stake',
  'solve',
  'quarrel',
  'sailing',
  'renaissance',
  'guitar',
  'caution'],
 8: ['way',
  'get',
  'got',
  'enough',
  'better',
  'told',
  'going',
  'look',
  'asked',
  'money',
  'keep',
  'job',
```

In [24]:
```
cluster_dict[5]
```

Out[24]:
```
['however',
 'number',
 'present',
 'problem',
 'forces',
 'stock',
 'ideas',
 'series',
 'theory',
 'somewhat',
 'lower',
 'described',
 'image',
 'activity',
 'justice',
 'staff',
 'pattern',
 'principle',
 'becomes',
 'success',
 'patient',
 'collection',
 'crisis',
 'youth',
 'presented',
 'file',
 'hearing',
 'region',
 'brief',
 'phase',
 'unity',
 'independence',
 'gross',
 'capable',
 'headquarters',
 'site',
 'european',
 'tension',
 'sensitive',
 'requires',
 'views',
 'protestant',
 'choose',
 'discovery',
 'argue']
```

In [25]:
```
cluster_dict[8]
```

Out[25]:
```
['way',
 'get',
 'got',
 'enough',
 'better',
 'told',
 'going',
 'look',
 'asked',
 'money',
 'keep',
 'job',
 'wife',
 'wanted',
 'girl',
 'mother',
 'leave',
 'hard',
 'idea',
 'father',
 'couldnt',
 'tried',
 'getting',
 'police',
 'talk',
 'ready',
 'anyone',
 'chance',
 'husband',
 'lot',
 'stay',
 'waiting',
 'wont',
 'pretty',
 'bit',
 'walk',
 'talking',
 'hadnt',
 'fight',
 'happy',
 'hed',
 'giving',
 'wait',
 'someone',
 'shed',
 'nice',
 'watching',
 'knife',
 'send',
 'jones',
 'mercer',
 'smiled',
 'asking',
 'kill',
 'please',
 'talked',
 'minute',
 'wished',
 'breakfast',
 'anyway',
 'fool',
 'drunk']
```

I think some are coherent to some extent. There are some potential meanings within them, above is some examples.