CSE 250A Hwk 4: Andrew Chafan A59020215.

Prob 1:

a) $\mathcal{L} = \log P(\text{data})$ (given tosses are i.i.d) $P(\text{data}) = \prod P(X=x_i)$

$= \log \prod P(X=x_i)$

$= \sum_{i=1}^{T} \log P(X=x_i)$ i being tosses ($1 \to T$)

$= \sum_{n=1}^{2D} \text{count}(X=n) . \log P(X=n)$

$= \sum_{n=1}^{2D} C_n . \log P_n$

b) $\mathcal{L}(x,y,\lambda) = f(x,y) - \lambda(g(x,y))$

$f(x,y) = \mathcal{L}$ we just calculated

$= \sum_{n=1}^{2D} C_n . \log P_n - \lambda . (\sum_{n=1}^{2D} p_n - 1)$

$g(x,y) \Rightarrow \sum p_n = 1$

$\sum p_n - 1 = 0.$

$$\frac{\partial \mathcal{L}(x,y,\lambda)}{\partial p_n} = \sum_{n=1}^{2D} \frac{C_n}{p_n} - \lambda.$$

ie: for $n=1 \Rightarrow \frac{C_1}{p_1} - \lambda$

$n=2 \Rightarrow \frac{C_2}{p_2} - \lambda.$

$n=3 \Rightarrow \frac{C_3}{p_3} - \lambda$

$$\Rightarrow \nabla_{p_n} = \begin{bmatrix} \frac{C_1}{p_1} - \lambda \\ \frac{C_2}{p_2} - \lambda \\ \vdots \\ \frac{C_n}{p_n} - \lambda \end{bmatrix} = 0$$

if we set the gradient vector equal to zero:

we get.

$$\Rightarrow \frac{c_1}{p_1} = \lambda \;\Big|\; \frac{c_2}{p_2} = \lambda \;\Big|\; \frac{c_3}{p_3} = \lambda \; \ldots$$

$$\Rightarrow \frac{c_n}{p_n} = \lambda \rightarrow p_n = \frac{c_n}{\lambda}$$

given $\sum_{n=1}^{2D} p_n = 1 \Rightarrow \sum_{n=1}^{2D} \frac{c_n}{\lambda} = 1$.

$$\Rightarrow \frac{1}{\lambda} \sum_{n=1}^{2D} c_n = 1 \Rightarrow \lambda = \sum_{n=1}^{2D} c_n$$

so we get $p_n = \frac{c_n}{\sum_{n=1}^{2D} c_n}$ i.e. $p_1 = \frac{c_1}{\sum_{n=1}^{2D} c_n}$

$p_2 = \frac{c_2}{\sum_{n=1}^{2D} c_n}$

and so on

c) $P_{even} = \sum_{n=1}^{N} P_{2n}$ | $P_{odd} = \sum_{n=1}^{N} P_{2n-1}$.

Given $P_{even} = P_{odd}$.

$\rightarrow P_{even} - P_{odd} = 0$

$\sum_{n=1}^{N} P_{2n} - \sum_{n=1}^{N} P_{2n-1} = 0$

$\sum_{n=1}^{N} P_{2n} - P_{2n-1} = 0$

$P_2 - P_1 + P_4 - P_3 + P_6 - P_4 \dots = 0$.

we can see the pattern that evens are (+) and odds are (-)

so we can simply multiply each P_n by $(-1)^n$
cause $(-1)^n$ w/ n even gives +1
$(-1)^n$ w/ n odd gives -1

$\Rightarrow \sum_{n=1}^{N} P_{2n} - P_{2n-1} = \sum_{n=1}^{2N} (-1)^n P_n = 0$. <u>Done</u>.

d) given: a priori $P_{even} = P_{odd}$.

$$\mathcal{L}(p_n, c_n, \lambda_1, \lambda_2) = f(p_n, c_n) - \lambda_1 (g_1(p_n)) - \lambda_2 (g_2(p_n))$$

w/ $f(p_n, c_n) = \sum_{n=1}^{2D} c_n \log p_n$.

$$g_1(p_n) = \sum_{n=1}^{2D} p_n - 1$$

$$g_2(p_n) = \sum_{n=1}^{2D} (-1)^n p_n$$

$$\mathcal{L} = \sum_{n=1}^{2D} c_n \log p_n - \lambda_1 \left(\sum_{n=1}^{2D} p_n - 1 \right) - \lambda_2 \sum_{n=1}^{2D} (-1)^n p_n$$

$$\nabla_{p_n} \mathcal{L} = \begin{bmatrix} \frac{c_1}{p_1} - \lambda_1 + \lambda_2 \\ \frac{c_2}{p_2} - \lambda_1 - \lambda_2 \\ \frac{c_3}{p_3} - \lambda_1 + \lambda_2 \\ \vdots \\ \frac{c_{2D}}{p_{2D}} - \lambda_1 - \lambda_2 \end{bmatrix} = 0.$$

$$\frac{c_n}{p_n} - \lambda_1 - \lambda_2 \cdot (-1)^n = 0$$

we get $\frac{c_n}{p_n} = \lambda_1 + (-1)^n \lambda_2 \Rightarrow p_n = \frac{c_n}{\lambda_1 + (-1)^n \lambda_2}$

We also have:

$$\sum_{1}^{N} P_{2n} + P_{2n-1} = 1 \quad \text{and they're equal}$$

$$\text{so} \Rightarrow \sum_{n=1}^{D} P_{2n} = \sum_{n=1}^{D} P_{2n-1} = 1/2$$

So for even:

$$\sum_{n=1}^{D} P_{2n} = \sum_{n=1}^{D} \frac{C_{2n}}{\lambda_1 + \lambda_2} = 1/2$$

$$\lambda_1 + \lambda_2 = 2 \underbrace{\sum_{n=1}^{D} C_{2n}} = 2\, C_{even} \quad \text{(Count of all evens)}.$$

$$\text{so } p_n = \frac{C_n}{2 C_{even}} \quad \text{ie: } p_2 = \frac{C_2}{2 C_{even}} \; ; \; p_4 = \frac{C_4}{2 C_{even}} \; \ldots$$

for odd:

$$\sum_{n=1}^{D} P_{2n-1} = \sum_{n=1}^{D} \frac{C_{2n-1}}{\lambda_1 - \lambda_2} = 1/2$$

$$\frac{1}{\lambda_1 - \lambda_2} \underbrace{\sum_{n=1}^{D} C_{2n-1}}_{C_{odd}} = 1/2$$

$$\lambda_1 - \lambda_2 = \frac{C_{odd}}{1/2} = 2\, C_{odd} \rightarrow p_n = \frac{C_n}{2 C_{odd}}$$

$$\text{ie } p_1 = \frac{C_1}{2 C_{odd}} \; ; \; p_3 = \frac{C_3}{2 C_{odd}} \; \ldots$$

Problem 2:

Slides Preface

a) $\mathcal{L} = \log P(G_1)$

$$= \log \prod_{t=1}^{T} P(X_1 = x_1^{(t)}, X_2 = x_2^{(t)}, \dots X_n = x_n^{(t)})$$

$$= \log \prod_{t=1}^{T} \prod_{i=1}^{n} P(X_i = x_i^{(t)} \mid pa_i = x)$$

$$= \sum_{i=1}^{n} \sum_{t=1}^{n} P(X_i = x_i^{(t)} / pa_i = x)$$

$$= \sum_{i=1}^{n} \sum_{X} \sum_{\pi} \text{Count}(X_i = x \mid pa_i = \pi) \cdot \log P(X_i = x \mid pa_i = \pi)$$

Given: $\text{Count}_i(x)$ is when $X_i = x$

$\text{Count}_i(x, x')$ is when $X_i = x$ & $X_{i+1} = x'$

So $$P_{ML}(X_{n+1} = x' \mid X_n = x) = \frac{\text{Count}(X_{n+1} = x', X_n = x)}{\text{Count}(X_n = x)}$$

$$= \frac{\text{Count}_i(x, x')}{\text{Count}_i(x)} \text{ for } i = 1 \to n-1$$

w/ $P_{ML}(X_1 = x) = \frac{\text{Count}(X_1 = x)}{T} = \frac{\text{Count}_1(x)}{T}$ lines #.

First element w/out parents.

b) $P_{ML}(X_n = x \mid X_{n+1} = x') = \dfrac{Count(X_n = x, X_{n+1} = x')}{Count(X_{n+1} = x')}$

$= \dfrac{Count_i(x, x')}{count_i(x')}$ for $i = 1 \to n-1$

$P_{ML}(X_n = x) = \dfrac{Count(X_n = x)}{T} = \dfrac{count_n(x)}{T}$ ← last element w/out parents.

* c) $P(G_1) = P(X_1) \cdot P(X_2 \mid X_1) \cdot P(X_3 \mid X_2) \ldots P(X_n \mid X_{n-1})$

in ML $\Rightarrow \dfrac{\cancel{Count(X_1 = x_1)}}{T} \cdot \dfrac{Count(X_2 = x_2; X_1 = x_1)}{\cancel{Count(X_1 = x_1)}} \times \dfrac{Count(X_3 = x_3, X_2 = x_2)}{Count(X_2 = x_2)} \ldots$

$= \dfrac{1}{T} \cdot \dfrac{\prod_{i=1}^{n-1} Count(X_{i+1} = x_{i+1} \, \& \, X_i = x_i)}{\prod_{i=2}^{n-1} Count(X_i = x_i)}$

$= \dfrac{1}{T} \cdot \dfrac{\prod_{i=1}^{n-1} Count_i(x_{i+1}, x_i)}{\prod_{i=2}^{n-1} Count_i'(x_i)}$

* Given that $P(X_3 \mid X_2, X_1) = P(X_3 \mid X_2)$ di-sep.
(applicable to all nodes)
so $P(X_1, X_2, X_3, \ldots X_n) = P(X_1) \cdot P(X_2 / X_1) \cdot P(X_3 / X_2) \ldots$

Same * Condition applies in G_2 (as stated before.
all nodes are ONLY conditioned on one parent (d_i - sep))

$$P(G_2) = P(X_1, X_2, X_3, \ldots X_n)$$
$$= P(X_n) \cdot P(X_{n-1}/X_n) \cdot P(X_{n-2}/X_{n-1}) \cdot \ldots P(X_1/X_2)$$

in ML $\Rightarrow \dfrac{\cancel{Count(X_n = x_n)}}{T} \cdot \dfrac{Count(X_{n-1} = x_{n-1}, X_n = x_n)}{\cancel{Count(X_n = x_n)}} \ldots \dfrac{Count(X_1 = x_1, X_2 = x_2)}{Count(X_2 = x_2)}$

$$\Rightarrow \frac{1}{T} \cdot \frac{\prod_{i=1}^{n-1} Count(X_i = x_i, X_{i+1} = x_{i+1})}{\prod_{i=2}^{n-1} Count(X_i = x_i)}$$

$$= \frac{1}{T} \frac{\prod_{i=1}^{n-1} count_i(x_i, x_{i+1})}{\prod_{i=2}^{n-1} Count_i(x_i)}$$

$$\Rightarrow \underset{(ML)}{P_{G_1}} = \underset{(ML)}{P_{G_2}}$$ + Proved

d) If we look at the DAG G_3; all nodes that are odd. (ie X_3, X_5, X_7 ...) except X_1.

$$P(X_3 / X_2, X_4) = \frac{Count(X_2, X_4, X_3)}{Count(X_2, X_4)}$$

which can be generalized to all nodes.

Given X_3, X_2 & X_4 are not conditionally independent (as in G_1, G_2)

So we can't use same notations as we did in G_1, G_2 : $count_n(x)$ & $Count_n(x, x')$.

So they won't have same joint distribution.

Q3 - A

```
import numpy as np
from collections import defaultdict
import math
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

```
f = open("/content/hw4_vocab.txt")
lines = f.readlines()
vocab = []
for i in range (len(lines)):
  vocab.append(lines[i])
  vocab[i] = vocab[i][:-1]
f.close()
```

```
f = open("/content/hw4_unigram.txt")
unigram = {}
lines = f.readlines()
for i in range (len(lines)):
  unigram[vocab[i]] = lines[i]
  unigram[vocab[i]] = int(unigram[vocab[i]][:-1])
f.close()
```

```
f = open("/content/hw4_bigram.txt")
bigram = {}
lines = f.readlines()
for i in range(len(lines)):
  l = lines[i].split("\t")
  l[-1] = l[-1][:-1]
  if vocab[int(l[0])-1] not in bigram:
    bigram[vocab[int(l[0])-1]] = [(vocab[int(l[1])-1], int(l[2]))]
  else:
    bigram[vocab[int(l[0])-1]].append((vocab[int(l[1])-1], int(l[2])))
f.close()
```

```
def probability(word):
  return unigram[word]/sum(unigram.values())

for key in unigram:
  if(key[0] == "M"):
    print("Word: ", key, "  Prob: ", probability(key))
```

```
Word:  MILLION   Prob:  0.002072759168154815
Word:  MORE   Prob:  0.0017088989966186725
Word:  MR.   Prob:  0.0014416083492816956
Word:  MOST   Prob:  0.0007879173033190295
Word:  MARKET   Prob:  0.0007803712804681068
Word:  MAY   Prob:  0.0007298973156289532
Word:  M.   Prob:  0.0007034067394618568
Word:  MANY   Prob:  0.0006967290595970209
Word:  MADE   Prob:  0.0005598610827336895
Word:  MUCH   Prob:  0.0005145971758110562
Word:  MAKE   Prob:  0.0005144626437991272
Word:  MONTH   Prob:  0.0004449095936318709З
Word:  MONEY   Prob:  0.00043710673693999306
Word:  MONTHS   Prob:  0.0004057607781605526
Word:  MY   Prob:  0.0004003183467688823
Word:  MONDAY   Prob:  0.00038198530259784006
Word:  MAJOR   Prob:  0.00037089252670515475
Word:  MILITARY   Prob:  0.00035204581485220204
Word:  MEMBERS   Prob:  0.00033606096579846475
Word:  MIGHT   Prob:  0.00027358919153183117
Word:  MEETING   Prob:  0.0002657374141083427
Word:  MUST   Prob:  0.0002665079156312084
Word:  ME   Prob:  0.00026357267173457725
Word:  MARCH   Prob:  0.0002597935452176646
Word:  MAN   Prob:  0.0002528834918776787
Word:  MS.   Prob:  0.0002389900041002911
Word:  MINISTER   Prob:  0.00023977273580605944
Word:  MAKING   Prob:  0.00021170446604452378
Word:  MOVE   Prob:  0.0002099555498894477
Word:  MILES   Prob:  0.00020596851026319035
```

Q3-B

```
def bigram_prob(word):
  words = {}
  total = 0

  for i in bigram[word]:
    total += i[1]

  for i in bigram[word]:
    words[i[0]] = i[1]/total

  words = (sorted(words.items(), key=lambda item: item[1],reverse=True))
  return words

print("    Word  \t Probability")
bigram_prob("THE")[:10]
```

```
    Word         Probability
[('<UNK>', 0.6150198100055118),
 ('U.', 0.013372499432610317),
 ('FIRST', 0.011720260675031612),
 ('COMPANY', 0.011658788055636611),
 ('NEW', 0.009451480076516552),
```

```
('UNITED', 0.008672308141231398),
('GOVERNMENT', 0.006803488635995202),
('NINETEEN', 0.006650714911000876),
('SAME', 0.006287066757449016),
('TWO', 0.006160749602827221)]
```

Q3-C

```
phrase = "THE STOCK MARKET FELL BY ONE HUNDRED POINTS LAST WEEK"
phrase = phrase.split(" ")
```

```
total = 0
for i in range(len(phrase)):
  total += math.log(probability(phrase[i]))
print("Lu = ", total)
```

```
Lu =  -64.50944034364878
```

```
phrase = "<s> THE STOCK MARKET FELL BY ONE HUNDRED POINTS LAST WEEK"
phrase = phrase.split(" ")
```

```
total = 0
for i in range(len(phrase)-1):
  parent = phrase[i]
  child = phrase[i+1]
  probab = dict(bigram_prob(parent))
  total += math.log(probab[child])
print("Lb : " , total)
```

```
Lb :  -40.91813213378977
```

We can definitely see here that the bigram method yields the highest likelood which is expected due to the nature of the language

Q3-D

```
phrase = "THE SIXTEEN OFFICIALS SOLD FIRE INSURANCE"
phrase = phrase.split(" ")
total = 0
for i in range(len(phrase)):
  total += math.log(probability(phrase[i]))
print("Lu = ", total)
```

```
Lu =  -44.291934473132606
```

```
phrase = "<s> THE SIXTEEN OFFICIALS SOLD FIRE INSURANCE"
phrase = phrase.split(" ")
total = 0
for i in range(len(phrase)-1):
  parent = phrase[i]
  child = phrase[i+1]
  probab = dict(bigram_prob(parent))
  if child not in probab:
    print(child,parent, "|| This combination  of child/parent is not available")
    total = float("-inf")
  else:
    total += math.log(probab[child])
print("Lb : " , total)
```

```
OFFICIALS SIXTEEN || This combination  of child/parent is not available
FIRE SOLD || This combination  of child/parent is not available
Lb :  -inf
```

We can here see that the loglikelihood becomes -inf which is equal to log(0) due to word pairs that are not existing in the dictionary

Q3-E

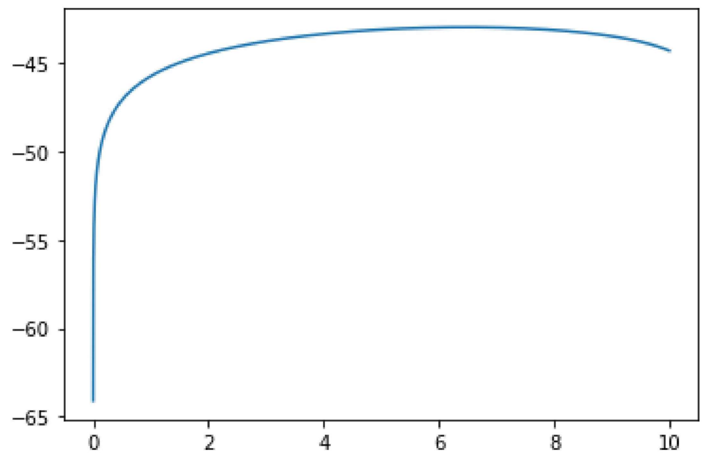
```
phrase = "<s> THE SIXTEEN OFFICIALS SOLD FIRE INSURANCE"
phrase = phrase.split(" ")
AllLms = []
precision = 10000
x = np.linspace(0.0001,10,precision)
for param in x:
  total = 0
  for i in range(0,len(phrase)-1):
    parent = phrase[i]
    child = phrase[i+1]
    proba = dict(bigram_prob(parent))
    proba = defaultdict(int,proba)
    total += math.log((1-(param/10))*(proba[child])+(param/10)*(unigram[child]/sum(unigram
  AllLms.append(total)

print("best lambda = ", (AllLms.index(max(AllLms)))/precision)
print("best Lm = ", (max(AllLms)))
```

```
best lambda =  0.6478
```

```
best Lm =  -42.96413717114809
```

```
plt.plot(x,AllLms)
plt.show()
```



Q4 - A

```
f = open("/content/nasdaq00.txt")
lines = f.readlines()
data_2000 = []
for i in range (len(lines)):
  data_2000.append(lines[i])
  data_2000[i] = float(data_2000[i][:-1])
data_2000 = np.array(data_2000)
f.close()
```

```
f = open("/content/nasdaq01.txt")
lines = f.readlines()
data_2001 = []
for i in range (len(lines)):
  data_2001.append(lines[i])
  data_2001[i] = float(data_2001[i][:-1])
data_2001 = np.array(data_2001)
f.close()
```

```
fact_00 = data_2000[3:].reshape(-1,1)
x_00 =[]
for i in range(len(data_2000)-3):
  temp = [data_2000[i],data_2000[i+1],data_2000[i+2]]
  x_00.append(temp)
x_00 = np.array(x_00)
```

```
from sklearn import linear_model as lm

theta = np.linalg.lstsq(x_00, fact_00)[0]
```

```
a3 = theta[0][0]
a2 = theta[1][0]
a1 = theta[2][0]


print("These are the linear coefficients: ")
print("a1: ", a1,"a2: ",a2,"a3: ", a3)
```

```
These are the linear coefficients:
a1:  0.9506722769536844 a2:  0.015603326703986786 a3:  0.031894723175170614
```

Q4 - B

```
y_pred = np.matmul(x_00,theta)
```

```
from sklearn import metrics
RMSE  = math.sqrt(metrics.mean_squared_error(fact_00,y_pred))
print("RMSE on training data (2000): ", RMSE)
```

```
RMSE on training data (2000):  117.9083331254247
```

Q4 -C

```
x_01 = []
fact_01 = data_2001[3:].reshape(-1,1)

for i in range(len(data_2001)-3):
  temp = [data_2001[i],data_2001[i+1],data_2001[i+2]]
  x_01.append(temp)

x_01 = np.array(x_01)
y_pred1 = np.matmul(x_01,theta)
RMSE =  math.sqrt(metrics.mean_squared_error(fact_01,y_pred1))
print("RMSE on test data (2001)", RMSE)
```

```
RMSE on test data (2001) 54.63605324590395
```