CSE 250A Hwk

Prob 1:

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

$= \log \prod P($

$= \sum_{i=1}^{T} \log P(x$

if we set the

we get.

$$\Rightarrow \frac{c_1}{p_1} = \lambda$$

$$\Rightarrow c_n =$$

c) $P_{even} = \sum_{n=1}^{N} P_{}$

Given $P_{even} = P_{}$

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

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

$\sum_{}^{N} P_{}$

d) given a pr

$$\mathcal{L}(p_n, c_n, \lambda_1, \lambda$$

$$\omega \mid g(p_n, c_n) =$$

$$g_1(p_n) = \sum_{n=1}^{\infty}$$

We also have:

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

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

So for even:

Problem 2:

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

$= \log \prod_{t=1}^{T}$

$= \log \prod_{t=1}^{T}$

b) $P_{ML}(X_{\bar{n}} = x \mid X_n$

$P_{ML}(X_n = x) =$ Co

Same * Condition

all nodes are

$P(G_2) = P(X_1, \ldots$

$= P(X_n) \cdot$

in TL $\Rightarrow$ Cut

Given K_3,

(as in G_1, G

so we can't

in G_1, G_2

so they wo

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:  0.00044490959363187093
    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),
```

```
('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)):
```

```
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

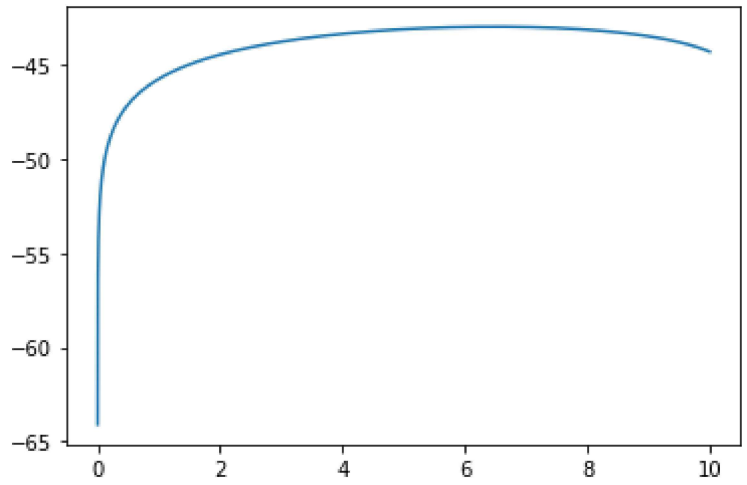
Q3-E

```
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 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
```

```
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
```