Margot Wagner
A53279875
CSE 250a HW4
Due: 11/3/20

4.1 MLE of a multinomial distribution

(a) Log-likelihood

$$\mathcal{L} = \log P(\text{data})$$
$$= \sum \text{count}(X=n) \log P(X=n)$$
$$= \sum_{n=1}^{2N} C_n \log p_n$$

(b) MLE

$$\mathcal{L}(n,\lambda) = f(n) - \lambda g(n)$$
$$\sum_{n=1}^{2N} p_n = 1 \qquad g(n) = \sum_{n=1}^{2n} p_n - 1 = 0$$
$$\mathcal{L}(n,\lambda) = \sum_{n=1}^{2N} C_n \log p_n - \lambda\left(\sum_{n=1}^{2N} p_n - 1\right)$$
$$\frac{\partial}{\partial p_n} \mathcal{L}(n,\lambda) = 0$$
$$0 = \sum_{n=1}^{2N} \frac{C_n}{p_n} - \lambda \sum_{n=1}^{2N} (1)$$
$$0 = \sum_{n=1}^{2N} C_n - \lambda \sum_{n=1}^{2N} p_n \qquad \sum_{n=1}^{2N} p_n = 1$$
$$0 = \sum_{n=1}^{2N} C_n - \lambda$$
$$\sum_{n=1}^{2N} C_n = \lambda \qquad \text{plug } \lambda \text{ in}$$
$$0 = C_n - \lambda p_n$$
$$p_n = \frac{C_n}{\lambda}$$
$$p_n = \frac{C_n}{\sum_{k=1}^{2N} C_k}$$

(c) $P(X \text{ is even}) = p_2 + p_4 + \cdots + p_{2N}$
$P(X \text{ is odd}) = p_1 + p_3 + \ldots + p_{2N-1}$

$$P(X \text{ is even}) = P(X \text{ is odd})$$

$$P(X \text{ even}) - P(X \text{ odd}) = 0$$

$$-p_1 + p_2 - p_3 + p_4 \cdots = 0$$

$$(-1)^1 p_1 + (-1)^2 p_2 + (-1)^3 p_3 \cdots = 0$$

$$\sum_{n=1}^{2N} (-1)^n p_n = 0$$

(d) want $\log P(\text{data})$

constraints: $\sum_{n=1}^{2N} p_n - 1 = 0$

$\sum_{n=1}^{2N} (-1)^n p_n = 0$

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

$$\frac{\partial L}{\partial p_n} = \sum_{n=1}^{2N} \frac{C_n}{p_n} - \lambda_1 \sum_{n=1}^{2N} (1) - \lambda_2 \sum_{n=1}^{2N} (-1)^n = 0$$

$$\frac{dL}{\partial p_i} = \frac{C_i}{p_i} + \lambda_1 + (-1)^i \lambda_2$$

$$p_i = \frac{-C_i}{\lambda_1 + (-1)^i \lambda_2}$$

$$\sum_{n=1}^{2N} (-1)^n p_n = 0 = P(X \text{ is odd}) - P(X \text{ is even})$$

$$P(X \text{ is odd}) + P(X \text{ is odd}) = 1$$

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

$$\sum_{i=1}^{N} \frac{-C_{2i}}{\lambda_1 + (-1)^{2i} \lambda_2} = \sum_{i=1}^{N} \frac{-C_{2i-1}}{\lambda_1 + (-1)^{2i-1} \lambda_2} = \frac{1}{2}$$

$$\lambda_1 + \lambda_2 = -2C_{even} \qquad \sum_{i=1}^{N} C_{2i} = C_{even}$$

$$\lambda_1 - \lambda_2 = -2C_{odd} \qquad \sum_{i=1}^{N} C_{2i-1} = C_{odd}$$

$$P_{2i} = \frac{-C_{2i}}{-2C_{even}} = \frac{C_{2i}}{2C_{even}}$$

$$P_{2i-1} = \frac{-C_{2i-1}}{-2C_{odd}} = \frac{C_{2i}}{2C_{even}}$$

4.2 MLE in belief networks

(a) G_1

$$P_{ML}(X_1) = \frac{COUNT_1(X_1)}{T}$$

$$P_{ML}(X_{n+1} = x' \mid X_n = x) = \frac{COUNT_n(x, x')}{COUNT_n(x)} \quad \text{for } X_2 \text{ to } X_n$$

(b) G_2

$$P_{ML}(X_n) = \frac{COUNT_n(X_n)}{T}$$

$$P_{ML}(X_{n-1} = x' \mid X_n = x) = \frac{COUNT_n(x, x')}{COUNT_n(x)} \quad \text{for } X_{n-1} \text{ to } X_1$$

(c)

G1

$$P^{ML}_{G1} = P(X_1, X_2, \ldots X_n)$$

$$= P(X_1)P(X_2 \mid X_1) \ldots P(X_n \mid X_{n-1})$$

$$= P(X_1) \prod_{i=1}^{n-1} P(X_{i+1} \mid X_i)$$

$$= \frac{count_1(x_1)}{T} \prod_{i=1}^{n-1} \frac{count_i(x_i, x_{i+1})}{count_i(x_i)}$$

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

G2

$$P^{ML}_{G2} = P(X_1, X_2, \ldots X_n)$$

$$= P(X_n)P(X_{n-1} \mid X_n) \ldots P(X_1 \mid X_2)$$

$$= P(X_n) \prod_{i=1}^{n-1} P(X_i \mid X_{i+1})$$

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

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

As a result, you get the same joint dist.

(d) G3 is not the same as G1/G2 because for example X_{n-2} is dependent on born X_{n-1} and X_{n-3} rather than just X_{n-1} or just X_{n-3}.

4.3 Statistical Language Processing

(a) Start with the letter "A"

	token	count	pu(w)
8	A	1505067	0.018407
9	AND	1460586	0.017863
23	AT	352650	0.004313
27	AS	326389	0.003992
34	AN	245234	0.002999
37	ARE	244452	0.002990
59	ABOUT	157448	0.001926
79	AFTER	110102	0.001347
80	ALSO	107113	0.001310
86	ALL	96631	0.001182
100	A.	83859	0.001026
140	ANY	51664	0.000632
142	AMERICAN	50048	0.000612
144	AGAINST	48729	0.000596
212	ANOTHER	35027	0.000428
248	AMONG	30604	0.000374
269	AGO	29155	0.000357
279	ACCORDING	28417	0.000348
311	AIR	25429	0.000311
329	ADMINISTRATION	23836	0.000292
345	AGENCY	22866	0.000280
353	AROUND	22637	0.000277
370	AGREEMENT	21487	0.000263
379	AVERAGE	21183	0.000259
382	ASKED	21114	0.000258
398	ALREADY	20366	0.000249
416	AREA	18895	0.000231
418	ANALYSTS	18482	0.000226
427	ANNOUNCED	18573	0.000227
435	ADDED	18088	0.000221
453	ALTHOUGH	17519	0.000214
462	AGREED	17316	0.000212
477	APRIL	16900	0.000207
492	AWAY	16521	0.000202

(b) 5 most likely words to follow "THE"

WORD	Pb(w' | THE)
<UNK>	0.61502
U.	0.01337
FIRST	0.01172
COMPANY	0.01166
NEW	0.00945

(c) "Last week the stock market fell by one hundred points"

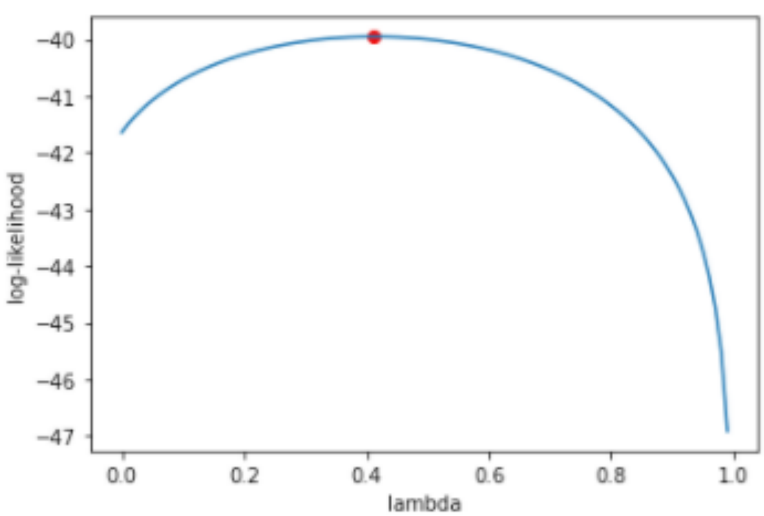
Log-likelihood of unigram model: **-41.643**
Log-likelihood of bigram model: **-44.740**

The **bigram model** yields the highest log-likelihood.

(d) "The nineteen officials sold fire insurance"
Log-likelihood of unigram model: **-41.643**
Log-likelihood of bigram model: **undefined**

"Nineteen officials" and **"sold fire**" are not seen in the training corpus. As a result, these **probabilities are zero**, bringing the entire probability to zero. The log likelihood of zero is undefined as e approaches negative infinity for values approaching zero.

(e) Mixture model



The optimal lambda is 0.41.

(f)

CSE 250a HW 4

In [7]:
```
import numpy as np
import pandas as pd
```

4.3 Statistical Language Modeling

- hw4_vocab.txt list of 500 tokens, corresponding to words, punctuation symbols, and other textual markers.
- hw4_unigram.txt contains the counts of each of these tokens in a large text corpus of Wall Street Journal articles.
- hw4_bigram.txt contains the counts of pairs of adjacent words in this same corpus. Let count(w1,w2) denote the number of times that word w1 is followed by word w2. The counts are stored in a simple three column format:

  index(w1) index(w2) count(w1,w2)

In [8]:
```
def parseFromFile(fname):
    data = []
    with open(fname, "r") as f:
        for line in f:
            data.append(line.rstrip())

    return data
```

In [9]:
```
words_list = parseFromFile("hw4_vocab.txt")
counts = parseFromFile("hw4_unigram.txt")
counts = [int(i) for i in counts]
words = pd.DataFrame(list(zip(words_list, counts)), columns=['token', 'count'])

words.head()
```

Out[9]:

	token	count
0	<UNK>	25223698
1	<s>	3021866
2	</s>	3021866
3	THE	3855375
4	,COMMA	3667333

In [10]:
```
bigram_data = np.loadtxt("hw4_bigram.txt")
```

(a) Compute the MLE of unigram distribution $P_u(w)$ over words w.

Print out a table of all tokens (words) that start with the letter "A", along with their numerical unigram probabilities.

In [12]:
```
words['pu(w)'] = words['count'] / sum(words['count'])
words.head()
```

Out[12]:

	token	count	pu(w)
0	<UNK>	25223698	0.308490
1	<s>	3021866	0.036958
2	</s>	3021866	0.036958
3	THE	3855375	0.047152
4	,COMMA	3667333	0.044852

In [13]:
```
words.loc[words['token'].str.startswith('A')]
```

Out[13]:

	token	count	pu(w)
8	A	1505067	0.018407
9	AND	1460586	0.017863
23	AT	352650	0.004313
27	AS	326389	0.003992
34	AN	245234	0.002999
37	ARE	244452	0.002990
59	ABOUT	157448	0.001926
79	AFTER	110102	0.001347
80	ALSO	107113	0.001310
86	ALL	96631	0.001182
100	A.	83859	0.001026
140	ANY	51664	0.000632
142	AMERICAN	50048	0.000612
144	AGAINST	48729	0.000596
212	ANOTHER	35027	0.000428
248	AMONG	30604	0.000374
269	AGO	29155	0.000357
279	ACCORDING	28417	0.000348
311	AIR	25429	0.000311
329	ADMINISTRATION	23836	0.000292
345	AGENCY	22866	0.000280

(b) Compute the MLE of the bigram distribution P_b(w2|w1)

Print out a table of the five most likely words to follow the word “THE”, along with their numerical bigram probabilities.

In [45]:
```
# complete array
bigram_counts = np.zeros((500*500, 3))

list_bigram_data = bigram_data[:, 0:2].tolist()
for row in bigram_counts:
    # if the row in complete data is in bigram_data, get counts
    as_list = row[0:2].tolist()
    if as_list in list_bigram_data:
        row[2] = bigram_data[list_bigram_data.index(as_list), 2]
```

In [57]:
```
bigram = pd.DataFrame(bigram_counts, columns=['w1', 'w2', 'count(w1,w2)'])
bigram.head(10)
```

Out[57]:

	w1	w2	count(w1,w2)
0	1.0	1.0	7355976.0
1	1.0	2.0	0.0
2	1.0	3.0	5645.0
3	1.0	4.0	647219.0
4	1.0	5.0	2373160.0
5	1.0	6.0	1801245.0
6	1.0	7.0	1048040.0
7	1.0	8.0	984875.0
8	1.0	9.0	336748.0
9	1.0	10.0	836709.0

In [76]:
```
grouped = bigram.groupby(['w1']).sum()
```

Out[76]:

	w2	count(w1,w2)
w1		
1.0	125250.0	25223698.0
2.0	125250.0	3021866.0
3.0	125250.0	0.0
4.0	125250.0	3855375.0
5.0	125250.0	3667333.0
...	...	...
496.0	125250.0	16517.0
497.0	125250.0	16529.0
498.0	125250.0	16451.0
499.0	125250.0	16540.0
500.0	125250.0	16573.0

500 rows × 2 columns

In [84]:
```
norm_counts = []
for i in range(1, len(words_list)+1):
    norm_counts.extend((bigram['count(w1,w2)'].loc[bigram['w1'] == i] / grouped.loc[i]['count(w1,w2)']).tolist())
```

In [87]:
```
bigram['pb(w1,w2)'] = norm_counts
bigram.head()
```

Out[87]:

	w1	w2	count(w1,w2)	pb(w1,w2)
0	1.0	1.0	7355976.0	0.291630
1	1.0	2.0	0.0	0.000000
2	1.0	3.0	5645.0	0.000224
3	1.0	4.0	647219.0	0.025659
4	1.0	5.0	2373160.0	0.094085

In [101]:
```
top_words = bigram.loc[bigram['w1'] == words_list.index('THE') + 1].sort_values(by=['pb(w1,w2)'], ascending=False)
top_5_idx = top_words['w2'].tolist()[:5]
top_5_pb = top_words['pb(w1,w2)'].tolist()[:5]
```

In [102]:
```
top_words
```

Out[102]:

	w1	w2	count(w1,w2)	pb(w1,w2)
1500	4.0	1.0	2371132.0	0.615020
1569	4.0	70.0	51556.0	0.013372
1578	4.0	79.0	45186.0	0.011720
1572	4.0	73.0	44949.0	0.011659
1560	4.0	61.0	36439.0	0.009451
...	...	...	...	...
1539	4.0	40.0	0.0	0.000000
1577	4.0	78.0	0.0	0.000000
1876	4.0	377.0	0.0	0.000000
1795	4.0	296.0	0.0	0.000000
1566	4.0	67.0	0.0	0.000000

500 rows × 4 columns

In [127]:
```
print("WORD \t | \t Pb(w'|THE)")
for i,pb in zip(top_5_idx, top_5_pb):
    print(words_list[int(i-1)], "\t | \t", round(pb, 5))
```

```
WORD     |       Pb(w'|THE)
<UNK>    |       0.61502
U.       |       0.01337
FIRST    |       0.01172
COMPANY          |       0.01166
NEW      |       0.00945
```

(c) "Last week the stock market fell by one hundred points."

Ignoring punctuation, compute and compare the log-likelihoods of this sentence under the unigram and bigram models:

In [ ]:
```
from math import log
```

In [128]:
```
def unigram_loglikely(sentence):
    sent_list = sentence.upper().split()
    unigram_prob = 1
    # Multiply probabilities of each word
    for word in sent_list:
        unigram_prob *= words['pu(w)'].loc[words['token'] == word].values
    return log(unigram_prob)
```

In [174]:
```
def bigram_loglikely(sentence):
    sent_list = sentence.upper().split()
    sent_list.insert(0, '<s>')
    bigram_prob = 1
    for i in range(len(sent_list) - 1):
        # get index of words
        w1_idx = words_list.index(sent_list[i]) + 1
        w2_idx = words_list.index(sent_list[i+1]) + 1

        # get bigram probabilities for words and multiply
        probs = bigram['pb(w1,w2)'].loc[(bigram['w1'] == w1_idx) & (bigram['w2'] == w2_idx)].values
        if probs== 0:
            print("No entries for", sent_list[i], "followed by", sent_list[i+1])

        bigram_prob *= probs

    if bigram_prob == 0:
        return "undefined"
    else:
        return log(bigram_prob)
```

In [157]:
```
sentence = "Last week the stock market fell by one hundred points"
uprob = unigram_loglikely(sentence)
biprob = bigram_loglikely(sentence)
```

In [162]:
```
print(uprob)
print(biprob)
print("The bigram model prints the highest log-likelihood")
```

```
-41.64345971649364
-44.740469213403735
The bigram model prints the highest log-likelihood
```

(d) "The nineteen officials sold fire insurance"

In [341]:
```
sentence = "The nineteen officials sold fire insurance"
uprob = unigram_loglikely(sentence)
biprob = bigram_loglikely(sentence)
```

```
No entries for NINETEEN followed by OFFICIALS
No entries for SOLD followed by FIRE
```

In [342]:
```
print(uprob)
print(biprob)
print("The unigram model prints the highest log-likelihood")
```

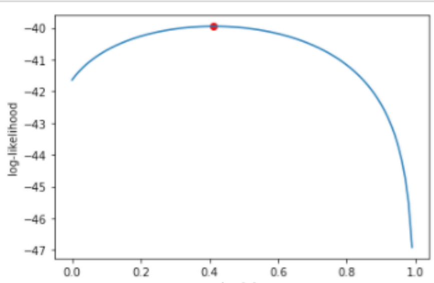
```
-41.64345971649364
undefined
The unigram model prints the highest log-likelihood
```

(e) Mixture Model

In [192]:
```
def mixture_ll(sentence, lam):
    sent_list = sentence.upper().split()
    mixture_prob = 1

    # Get unigram and bigram probs
    u_probs = []
    for word in sent_list:
        u_probs.append(words['pu(w)'].loc[words['token'] == word].values[0])

    sent_list.insert(0, '<s>')
    b_probs = []
    for i in range(len(sent_list) - 1):
        # get index of words
        w1_idx = words_list.index(sent_list[i]) + 1
        w2_idx = words_list.index(sent_list[i+1]) + 1

        # get bigram probabilities for words and multiply
        b_probs.append(bigram['pb(w1,w2)'].loc[(bigram['w1'] == w1_idx) & (bigram['w2'] == w2_idx)].values[0])

    for i in range(len(u_probs)):
        mixture_prob *= (1 - lam)*u_probs[i] + lam*b_probs[i]

    return log(mixture_prob)
```

In [213]:
```
sentence = "The nineteen officials sold fire insurance"
step = 0.1
lam_range = np.linspace(0, 0.99, 100)
loglikely = []
for l in lam_range:
    loglikely.append(mixture_ll(sentence, l))

max_ll = max(loglikely)
opt_lam = lam_range[loglikely.index(max_ll)]
```

In [221]:
```
import matplotlib.pyplot as plt
plt.plot(lam_range, loglikely)
plt.scatter(opt_lam, max_ll, color='r')
plt.xlabel("lambda")
plt.ylabel("log-likelihood")
plt.show()
```



In [224]:
```
print("The optimal lambda is", round(opt_lam,2))
```

```
The optimal lambda is 0.41
```

4.4 Stock Market prediction

(a) Linear coefficients

$a_1 = 0.9507$
$a_2 = 0.0156$
$a_3 = 0.0319$

(b) Mean squared prediction error

$MSE_{2000} = 13902.40$
$MSE_{2001} = 2985.10$

These MSEs are very large, so I **would not recommend** this model for stock market predictions.

(c) Source code

4.4 Stock Market Prediction

(a) Linear Coefficients

In [225]:
```
nas_00 = np.loadtxt('hw4_nasdaq00.txt')
nas_01 = np.loadtxt('hw4_nasdaq01.txt')
```

In [273]:
```
# 3 x T matrix of all x_t's
x_t = np.zeros((len(nas_00) - 3, 3))
y_t = np.zeros(len(nas_00) - 3)

for i in range(3, len(nas_00)):
    x_t[i-3, :] = [nas_00[i-1], nas_00[i-2], nas_00[i-3]]
    y_t[i-3] = nas_00[i]

# weight matrix is just x = inv(A)b
#w = np.dot(np.linalg.inv(A), b)
```

In [277]:
```
# A is the sum over t of the outer product of x_t and x_t^T
A = np.zeros((3,3))
for i in range(len(b_t)):
    A = np.add(A, np.outer(x_t[i], x_t.transpose()[:, i]))
```

In [283]:
```
# b is the sum over t of the product of x_t and y_t
b = np.zeros((3,1))
for i in range(len(b_t)):
    b = np.add(b, np.outer(x_t[i], y_t[i]))
```

In [288]:
```
# w = inv(A)b
w = np.dot(np.linalg.inv(A), b)
```

In [289]:
```
w
```

Out[289]:
```
array([[0.95067337],
       [0.01560133],
       [0.03189569]])
```

(b) Mean squared prediction error

In [336]:
```
def pred(nas_data, w):
    y_pred = []
    for i in range(3, len(nas_data)):
        y_pred.append(float(w[0])*nas_data[i-1] + float(w[1])*nas_data[i-2] + float(w[2])*nas_
data[i-3])

    return y_pred
```

In [337]:
```
def mse(y, y_pred):
    mse = 0
    for i in range(len(y)):
        mse += (y[i] - y_pred[i])**2

    # normalize
    mse /= len(y)
    return mse
```

In [338]:
```
y_pred_00 = pred(nas_00, w)
y_pred_01 = pred(nas_01, w)
```

In [339]:
```
mse_00 = mse(nas_00[3:], y_pred_00)
mse_01 = mse(nas_01[3:], y_pred_01)
```

In [340]:
```
print(round(mse_00, 4))
print(round(mse_01, 4))
```

```
13902.4011
2985.0979
```