Language Models (LMs)

- based on the https://medium.com/@philiposbornedata/learning-nlp-language-models-with-real-data-cdff04c51c25
- based on Speech and Language Processing (3rd ed. draft) by Dan Jurafsky and James H. Martin

Enable equation numbering in jupyter notebook for improved readability:

In [ ]:
```
%%javascript
MathJax.Hub.Config({
    TeX: { equationNumbers: { autoNumber: "AMS" } }
});
```

Introduction

Part 1

I. Probabilistic Language Models

Example uses

- **Machine Translation:** P(**high** winds tonight) > P(**large** winds tonight)
- **Spell Correction:** P(...about fifteen **minutes** from...) > P(...about fifteen **mineuts** from...)
- **Speech Recognition:** P(I saw a van) >> P(eyes awe of an)

Aim of Language Models

The goal of paobabilistic language modelling is to calculate the probability of a sentence of sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

and can be used to find the probability of the next word in the sequence:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

a model that computes either of these is called a **language model (LM)**.

II. Initial Method for Calcualting Probabilities

Defn: Conditional Probability

Let A and B be two events with $P(B) \neq 0$, the conditional probability of A given B is:

$$P(A \mid B) = \frac{P(A, B)}{P(B)}$$

Defn: Chain Rule

$$P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)\ldots P(x_n \mid x_1, \ldots, x_{n-1})$$

The Chain Rule applied to compute the join probability of words in a sentence:

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

e.g.

P("its water is so transparent") =

```
P(its)xP(water|its)
x P(is|its water)
x P(so|its water is)
x P(transparent|its water is so)
```

Can we estimate this by simply counting and dividing the results by the following?

$$P(transparent \mid its\ water\ is\ so) = \frac{count(its\ water\ is\ so\ transparent)}{count(its\ water\ is\ so)}$$

NO! Far to many possible sentences that would need to be calculated, we would never have enough data to achieve this.

III. Methods using the Markov Assumption

Defn: Markov Property

A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. A process with this property is called a Markov process. [1]

In other words, the probability of the next word can be estimated given only the previous k number of words.

e.g.

$$P(transparent \mid its\ water\ is\ so) \approx P(transparent \mid so)$$

or

$$P(transparent \mid its\ water\ is\ so) \approx P(transparent \mid is\ so)$$

General Equation for the Markov Assumption

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

where k is the number of words in the 'state' to be defined by the user.

Unigram Model (k=1)

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Bigram Model (k=2)

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

IV. N-gram Models (k=n)

The previous two equations can be extended to compute trigrams, 4-grams, 5-grams, etc. In general, this is an insufficient model of language because **sentences often have long distance dependencies**. For example, the subject of the sentence may be at the start whilst our next word to be predicited occurs more than 10 words later.

Estimating Bigram Probabilities using the Maximum Likelihood Estimate

$$P(w_i | w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Small Example

Three sentences:

- < s I am Sam /s >
- < s Sam I am /s >
- < s I do not like green eggs and ham /s >

$$P(I | < s) = \frac{count(< s, I)}{count(< s)} = \frac{2}{3}$$

$$P(am | I) = \frac{count(I, am)}{count(I)} = \frac{2}{3}$$

[1] https://en.wikipedia.org/wiki/Markov_property

Example with Real Data

Import Packages and IMDB Movie Review Data [2]

[2] http://ai.stanford.edu/~amaas/data/sentiment/

In [1]:

```
import requests
import time

import xml.etree.ElementTree as ET
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import re
import glob
import random
import seaborn as sns
import string

from IPython.display import clear_output

# Hide warnings
import warnings
warnings.filterwarnings('ignore')

# http://www.nltk.org/howto/wordnet.html

from nltk.corpus import wordnet as wn
from nltk.corpus import stopwords
from nltk.wsd import lesk
```

In [2]:

```
# Location of test/train data files on local computer, data downloaded direct
#test_dir = '/Users/philiposborne/Documents/Written Notes/Learning Notes/IMDB
#train_dir = '/Users/philiposborne/Documents/Written Notes/Learning Notes/IMD

data = pd.read_csv('imdb_master.csv',encoding="latin-1")
```

In [3]:
```python
data.head(20)
```

Out[3]:

	Unnamed: 0	type	review	label	file
0	0	test	Once again Mr. Costner has dragged out a movie...	neg	0_2.txt
1	1	test	This is an example of why the majority of acti...	neg	10000_4.txt
2	2	test	First of all I hate those moronic rappers, who...	neg	10001_1.txt
3	3	test	Not even the Beatles could write songs everyon...	neg	10002_3.txt
4	4	test	Brass pictures (movies is not a fitting word f...	neg	10003_3.txt
5	5	test	A funny thing happened to me while watching "M...	neg	10004_2.txt
6	6	test	This German horror film has to be one of the w...	neg	10005_2.txt
7	7	test	Being a long-time fan of Japanese film, I expe...	neg	10006_2.txt
8	8	test	"Tokyo Eyes" tells of a 17 year old Japanese g...	neg	10007_4.txt
9	9	test	Wealthy horse ranchers in Buenos Aires have a ...	neg	10008_4.txt
10	10	test	Cage plays a drunk and gets high critically pr...	neg	10009_3.txt
11	11	test	First of all, I would like to say that I am a ...	neg	1000_3.txt
12	12	test	So tell me - what serious boozer drinks Budwei...	neg	10010_2.txt
13	13	test	A big disappointment for what was touted as an...	neg	10011_1.txt
14	14	test	This film is absolutely appalling and awful. I...	neg	10012_1.txt
15	15	test	Here's a decidedly average Italian post apocal...	neg	10013_4.txt
16	16	test	At the bottom end of the apocalypse movie scal...	neg	10014_2.txt
17	17	test	Earth has been destroyed in a nuclear holocaus...	neg	10015_4.txt
18	18	test	Many people are standing in front of the house...	neg	10016_3.txt
19	19	test	New York family is the last in their neighborh...	neg	10017_1.txt

In [4]:
```python
# Select only training data
data = data[data['type']=='train'].reset_index(drop=True)
```

In [5]:
```python
data.head()
```

Out[5]:

	Unnamed: 0	type	review	label	file
0	25000	train	Story of a man who has unnatural feelings for ...	neg	0_3.txt
1	25001	train	Airport '77 starts as a brand new luxury 747 p...	neg	10000_4.txt
2	25002	train	This film lacked something I couldn't put my f...	neg	10001_4.txt
3	25003	train	Sorry everyone,,, I know this is supposed to b...	neg	10002_1.txt
4	25004	train	When I was little my parents took me along to ...	neg	10003_1.txt

In [6]:
```python
print('Number of comments in data:', len(data))

data = data[0:1000]

print('Number of comments left in data after removal:', len(data))
```

```
Number of comments in data: 75000
Number of comments left in data after removal: 1000
```

In [7]:

```
train_data = data
```

In [8]:

```
# Data import written as a function:
# Replace test and train dir with correct path for file saved on local comput
# Data files are downloaded from reference link above where main file name is

# This function converts the raw files form the original Stanford source into
'''
train_dir = "../LangAndComputer/aclImdb/train"
def IMDB_to_csv(directory):
    data = pd.DataFrame()

    for filename in glob.glob(str(directory)+'/neg/*.txt'):
        with open(filename, 'r',  encoding="utf8") as f:
            content = f.readlines()
            content_table = pd.DataFrame({'id':filename.split('_')[0].split('
        data = data.append(content_table)

    for filename in glob.glob(str(directory)+'/pos/*.txt'):
        with open(filename, 'r',  encoding="utf8") as f:
            content = f.readlines()
            content_table = pd.DataFrame({'id':filename.split('_')[0].split('
        data = data.append(content_table)
    data = data.sort_values(['pol','id'])
    data = data.reset_index(drop=True)
    #data['rating_norm'] = (data['rating'] - data['rating'].min())/( data['ra

    return(data)

train_data = IMDB_to_csv(train_dir)

'''
```

Out[8]:

```
'\ntrain_dir = "../LangAndComputer/aclImdb/train"\ndef IMDB_to_csv(directory):
\n    data = pd.DataFrame()\n    \n    for filename in glob.glob(str(director
y)+\'/neg/*.txt\'):\n        with open(filename, \'r\',  encoding="utf8") as
f:\n            content = f.readlines()\n            content_table = pd.DataFr
ame({\'id\':filename.split(\'_\')[0].split(\'/\')[-1],\'rating\':filename.spli
t(\'_\')[1].split(\'.\')[0],\'pol\':\'neg\', \'text\':content})\n        data
= data.append(content_table)\n        \n    for filename in glob.glob(str(dire
ctory)+\'/pos/*.txt\'):\n        with open(filename, \'r\',  encoding="utf8")
as f:\n            content = f.readlines()\n            content_table = pd.Dat
aFrame({\'id\':filename.split(\'_\')[0].split(\'/\')[-1],\'rating\':filename.s
plit(\'_\')[1].split(\'.\')[0],\'pol\':\'pos\', \'text\':content})\n        da
ta = data.append(content_table)\n    data = data.sort_values([\'pol\',\'id\'])
\n    data = data.reset_index(drop=True)\n    #data[\'rating_norm\'] = (data
[\'rating\'] - data[\'rating\'].min())/( data[\'rating\'].max() - data[\'ratin
g\'].min() )\n\n    return(data)\n\ntrain_data = IMDB_to_csv(train_dir)\n\n'
```

In [9]:

```
train_data.columns = ['id', 'dataset', 'text', 'pol','file']
train_data.head()
```

Out[9]:

	id	dataset	text	pol	file
0	25000	train	Story of a man who has unnatural feelings for ...	neg	0_3.txt
1	25001	train	Airport '77 starts as a brand new luxury 747 p...	neg	10000_4.txt
2	25002	train	This film lacked something I couldn't put my f...	neg	10001_4.txt

	id	dataset	text	pol	file
3	25003	train	Sorry everyone,,, I know this is supposed to b...	neg	10002_1.txt
4	25004	train	When I was little my parents took me along to ...	neg	10003_1.txt

We reduce the number of rows in our training corpus for learning purposes as the models take a substantial amout of time otherwise.

Data Pre-processing

1. Convert full text comment into individual sentences

We can break the full text down into its individual sentences using the following:

In [10]:
```
train_data['text'][0].split('.')
```

Out[10]:
```
['Story of a man who has unnatural feelings for a pig',
 ' Starts out with a opening scene that is a terrific example of absurd comed
y',
 " A formal orchestra audience is turned into an insane, violent mob by the cr
azy chantings of it's singers",
 ' Unfortunately it stays absurd the WHOLE time with no general narrative even
tually making it just too off putting',
 ' Even those from the era should be turned off',
 ' The cryptic dialogue would make Shakespeare seem easy to a third grader',
 " On a technical level it's better than you might think with some good cinema
tography by future great Vilmos Zsigmond",
 ' Future stars Sally Kirkland and Frederic Forrest can be seen briefly',
 '']
```

And can reference individual setences via indexing. We can also find the total number of sentences for each comment but it appears that the last setence is always blank. Therefore, when we use this in our loop, we reduce the upper index bound by 1 to account for this.

In [11]:
```
train_data['text'][0].split('.')[0]
```

Out[11]:
```
'Story of a man who has unnatural feelings for a pig'
```

In [12]:
```
len(train_data['text'][0].split('.'))
```

Out[12]: 9

In [13]:
```
train_data['text'][0].split('.')[8]
```

Out[13]:
```
''
```

Using these, we break down our full commments into individual sentences. I have also introduced some methods for tracking and timing the progress of our for loops, for more info on these see the following:

PhilipOsborneData.com link

Medium.com link

In [14]:

```
train_data_sent = pd.DataFrame()

start_time = time.time()
for index in train_data.index:
    data_row = train_data.iloc[index,:]

    for sent_id in range(0,len(data_row['text'].split('.'))-1):
        sentence = data_row['text'].split('.')[sent_id]
        # Form a row in a dataframe for this setence that captures the words
        # We must pass an arbitrary index which we then reset to show unique
        sentence_row = pd.DataFrame({
                                    'id':data_row['id'],
                                    'pol':data_row['pol'],
                                    'sent_id':sent_id,
                                    'sentence':sentence}, index = [index])

        # Form full table that has rows for all sentences
        train_data_sent = train_data_sent.append(sentence_row)


    # Outputs progress of main loop, see:
    clear_output(wait=True)
    print('Proportion of comments completed:', np.round(index/len(train_data)

end_time = time.time()
print('Total run time = ', np.round(end_time-start_time,2)/60, ' minutes')
# Reset index so that each index value is a unique number
train_data_sent = train_data_sent.reset_index(drop=True)
```

```
Proportion of comments completed: 99.9 %
Total run time =  0.8481666666666666  minutes
```

In [15]:

```
train_data_sent.head(20)
#train_data_sent.shape
```

Out[15]:

	id	pol	sent_id	sentence
0	25000	neg	0	Story of a man who has unnatural feelings for ...
1	25000	neg	1	Starts out with a opening scene that is a ter...
2	25000	neg	2	A formal orchestra audience is turned into an...
3	25000	neg	3	Unfortunately it stays absurd the WHOLE time ...
4	25000	neg	4	Even those from the era should be turned off
5	25000	neg	5	The cryptic dialogue would make Shakespeare s...
6	25000	neg	6	On a technical level it's better than you mig...
7	25000	neg	7	Future stars Sally Kirkland and Frederic Forr...
8	25001	neg	0	Airport '77 starts as a brand new luxury 747 p...
9	25001	neg	1	The luxury jetliner takes off as planned but ...
10	25001	neg	2	With air in short supply, water leaking in & ...
11	25001	neg	3	
12	25001	neg	4	

	id	pol	sent_id	sentence
13	25001	neg	5	 Also known under the slightly diff...
14	25001	neg	6	Out of the three Airport films I have seen so...
15	25001	neg	7	It has my favourite plot of the three with a ...
16	25001	neg	8	While the rather sluggish plot keeps one ente...
17	25001	neg	9	Even when the Navy become involved things don...
18	25001	neg	10	George Kennedy as the jinxed airline worker J...
19	25001	neg	11	 The home video & theatrical versio...

To simplify the process we remove all grammer and lowercase all words in the sentences.

We also add the string '<s' and '/s>' to the start and end of each sentence respectively so that we can find which words start and complete the sentences.

In [16]:

```
train_data_sent['sentence_clean'] = train_data_sent['sentence'].str.replace('
train_data_sent['sentence_clean'] = train_data_sent['sentence_clean'].str.low

train_data_sent['sentence_clean'] = '<s ' + train_data_sent['sentence_clean']
train_data_sent['sentence_clean'] = train_data_sent['sentence_clean'] + ' /s>

train_data_sent.head()
```

Out[16]:

	id	pol	sent_id	sentence	sentence_clean
0	25000	neg	0	Story of a man who has unnatural feelings for ...	<s story of a man who has unnatural feelings f...
1	25000	neg	1	Starts out with a opening scene that is a ter...	<s starts out with a opening scene that is a ...
2	25000	neg	2	A formal orchestra audience is turned into an...	<s a formal orchestra audience is turned into...
3	25000	neg	3	Unfortunately it stays absurd the WHOLE time ...	<s unfortunately it stays absurd the whole ti...
4	25000	neg	4	Even those from the era should be turned off	<s even those from the era should be turned o...

In [17]:

```
text = train_data_sent['sentence_clean']

text_list = " ".join(map(str, text))
text_list[0:100]
```

Out[17]:

```
'<s story of a man who has unnatural feelings for a pig /s> <s  starts out wit
h a opening scene that '
```

Find the occurence of each word and use this to find the probability of occurence of each word

In [18]:

```
word_list = pd.DataFrame({'words':text.str.split(' ', expand = True).stack().
```

In [19]:

```
word_list.head(10)
```

Out[19]:

	words
0	<s
1	story
2	of
3	a
4	man
5	who
6	has
7	unnatural
8	feelings
9	for

In [20]:

```
word_count_table = pd.DataFrame()
for n,word in enumerate(word_list['words']):
    # Create a list of just the word we are interested in, we use regular exp
    # e.g. 'ear' would be counted in each appearance of the word 'year'
    word_count = len(re.findall(' ' + word + ' ', text_list))
    word_count_table = word_count_table.append(pd.DataFrame({'count':word_cou

    clear_output(wait=True)
    print('Proportion of words completed:', np.round(n/len(word_list),4)*100,

word_list['count'] = word_count_table['count']
# Remove the count for the start and end of sentence notation so
# that these do not inflate the other probabilities

## I commented out for the probability
#word_list['count'] = np.where(word_list['words'] == '<s' , 0,
#                     np.where(word_list['words'] == '/s>', 0,
#                     word_list['count']))
```

```
Proportion of words completed: 99.99 %
```

In [21]:

```
word_list['prob'] = word_list['count']/sum(word_list['count'])
word_list.head(20)
```

Out[21]:

	words	count	prob
0	<s	13330	0.052733
1	story	395	0.001563
2	of	5150	0.020373
3	a	5911	0.023384
4	man	131	0.000518
5	who	720	0.002848
6	has	522	0.002065
7	unnatural	2	0.000008
8	feelings	9	0.000036

	words	count	prob
9	for	1579	0.006246
10	pig	1	0.000004
11	/s>	13330	0.052733
12		11032	0.043642
13	starts	55	0.000218
14	out	637	0.002520
15	with	1529	0.006049
16	opening	43	0.000170
17	scene	220	0.000870
18	that	2738	0.010831
19	is	3730	0.014756

Unigram Model (k=1): $P(w_1 w_2 ... w_n) \approx \prod_i P(w_i)$

To apply this we can simply lookup the probability of each word in the sentence and then calculate the multiplicative product of these probabilities.

Note for this notebook, I have reduced the number of items considered so that it runs in good time

In [22]:

```
unigram_table = pd.DataFrame()

start_time = time.time()
# Loop through each sentence
# REMOVE ROW LIMIT FOR FULL RUN
for index in train_data_sent[0:200].index:
    data_row = train_data_sent.iloc[index,:]

    sent_probs = pd.DataFrame()
    # Go through each word in the sentence, lookup the probability of the wor
    # then find the mulitplicitive product of all probabilities in the senten


    ## I modified code for correction
    #for n,word in enumerate(data_row['sentence_clean']):
    for n,word in enumerate(data_row['sentence_clean'].split(' ')):

    #error corrected. We need iloc[0] to get a value
        #sent_probs = sent_probs.append(pd.DataFrame({'prob':word_list[ word_
        sent_probs = sent_probs.append(pd.DataFrame({'prob':word_list[ word_l

    unigram = sent_probs['prob'].prod(axis=0)

    # Create a list of unigram calculation for each sentence
    unigram_table = unigram_table.append(pd.DataFrame({'unigram':unigram},ind

    clear_output(wait=True)
    print('Proportion of sentences completed:', np.round(index/len(train_data_
```

```
end_time = time.time()
print('Total run time = ', np.round(end_time-start_time,2)/60, ' minutes')

train_data_sent['unigram'] = unigram_table['unigram']
```

```
Proportion of sentences completed: 1.49 %
Total run time =  0.16766666666666669  minutes
```

In [23]:

```
base_time = end_time-start_time
```

In [24]:

```
unigram_table.head(10)
```

Out[24]:

	unigram
0	1.027054e-36
1	1.592345e-44
2	8.442173e-68
3	1.243502e-61
4	7.841752e-31
5	6.009184e-45
6	2.063734e-65
7	1.305861e-46
8	1.973786e-214
9	0.000000e+00

In [25]:

```
train_data_sent.head(10)
```

Out[25]:

	id	pol	sent_id	sentence	sentence_clean	unigram
0	25000	neg	0	Story of a man who has unnatural feelings for ...	<s story of a man who has unnatural feelings f...	1.027054e-36
1	25000	neg	1	Starts out with a opening scene that is a ter...	<s starts out with a opening scene that is a ...	1.592345e-44
2	25000	neg	2	A formal orchestra audience is turned into an...	<s a formal orchestra audience is turned into...	8.442173e-68
3	25000	neg	3	Unfortunately it stays absurd the WHOLE time ...	<s unfortunately it stays absurd the whole ti...	1.243502e-61
4	25000	neg	4	Even those from the era should be turned off	<s even those from the era should be turned o...	7.841752e-31
5	25000	neg	5	The cryptic dialogue would make Shakespeare s...	<s the cryptic dialogue would make shakespear...	6.009184e-45
6	25000	neg	6	On a technical level it's better than you mig...	<s on a technical level its better than you m...	2.063734e-65
7	25000	neg	7	Future stars Sally Kirkland and Frederic Forr...	<s future stars sally kirkland and frederic f...	1.305861e-46
8	25001	neg	0	Airport '77 starts as a brand new luxury 747 p...	<s airport 77 starts as a brand new luxury 747...	1.973786e-214

	id	pol	sent_id	sentence	sentence_clean	unigram
9	25001	neg	1	The luxury jetliner takes off as planned but ...	<s the luxury jetliner takes off as planned b...	0.000000e+00

We can use logarithm space as addition is faster computationally than multiplications.

As by log rules: $log(P1xP2xP3) = log(P1) + log(P2) + log(P3)$

In [26]:

```
unigram_table_log = pd.DataFrame()

start_time_log = time.time()
# Loop through each sentence
# REMOVE ROW LIMIT FOR FULL RUN
for index in train_data_sent[0:200].index:
    data_row = train_data_sent.iloc[index,:]

    sent_probs = pd.DataFrame()
    # Go through each word in the sentence, lookup the probability of the wor
    # then find the mulitplicitive product of all probabilities in the senten

    ## Same thing, I modified it because of enumerate
    for n,word in enumerate(data_row['sentence_clean'].split(" ")):

        #error corectied
        log_prob = np.log10(word_list[ word_list['words']==word]['prob'].iloc
        sent_probs = sent_probs.append(pd.DataFrame({'log_prob':log_prob}, in

    unigram_log = np.sum(sent_probs['log_prob'], axis=0)
    #unigram_log = sum(sent_probs['log_prob'])
    #print(unigram_log)
    # Create a list of unigram calculation for each sentence
    unigram_table_log = unigram_table_log.append(pd.DataFrame({'unigram_log':

    clear_output(wait=True)
    print('Proportion of sentences completed:', np.round(index/len(train_data_

end_time_log = time.time()
print('Total run time = ', np.round(end_time_log-start_time_log,2)/60, ' minu

train_data_sent['unigram_log'] = unigram_table_log['unigram_log']
```

```
Proportion of sentences completed: 1.49 %
Total run time =  0.16666666666666666  minutes
```

In [27]:

```
unigram_table_log.head(20)
```

Out[27]:

	unigram_log
0	-35.988407
1	-43.797963
2	-67.073546
3	-60.905353
4	-30.105587
5	-44.221185
6	-64.685346
7	-45.884103

	unigram_log
8	-213.704700
9	-340.228677
10	-108.652670
11	-3.915929
12	-3.915929
13	-180.526339
14	-53.346830
15	-399.572650
16	-104.063581
17	-94.080917
18	-103.778119
19	-194.646053

In [28]:

```
log_time = end_time_log - start_time_log
```

In [29]:

```
print('The log base 10 method takes approximately ', np.round((log_time)/base_
```

```
The log base 10 method takes approximately  100.6 % of the time of the orginal
calculation.
```

In [30]:

```
train_data_sent.head(20)
```

Out[30]:

	id	pol	sent_id	sentence	sentence_clean	unigram	unigram_log
0	25000	neg	0	Story of a man who has unnatural feelings for ...	<s story of a man who has unnatural feelings f...	1.027054e-36	-35.988407
1	25000	neg	1	Starts out with a opening scene that is a ter...	<s starts out with a opening scene that is a ...	1.592345e-44	-43.797963
2	25000	neg	2	A formal orchestra audience is turned into an...	<s a formal orchestra audience is turned into...	8.442173e-68	-67.073546
3	25000	neg	3	Unfortunately it stays absurd the WHOLE time ...	<s unfortunately it stays absurd the whole ti...	1.243502e-61	-60.905353
4	25000	neg	4	Even those from the era should be turned off	<s even those from the era should be turned o...	7.841752e-31	-30.105587
5	25000	neg	5	The cryptic dialogue would make Shakespeare s...	<s the cryptic dialogue would make shakespear...	6.009184e-45	-44.221185
6	25000	neg	6	On a technical level it's better than you mig...	<s on a technical level its better than you m...	2.063734e-65	-64.685346

	id	pol	sent_id	sentence	sentence_clean	unigram	unigram_log
7	25000	neg	7	Future stars Sally Kirkland and Frederic Forr...	<s future stars sally kirkland and frederic f...	1.305861e-46	-45.884103
8	25001	neg	0	Airport '77 starts as a brand new luxury 747 p...	<s airport 77 starts as a brand new luxury 747...	1.973786e-214	-213.704700
9	25001	neg	1	The luxury jetliner takes off as planned but ...	<s the luxury jetliner takes off as planned b...	0.000000e+00	-340.228677
10	25001	neg	2	With air in short supply, water leaking in & ...	<s with air in short supply water leaking in ...	2.225000e-109	-108.652670
11	25001	neg	3		<s /s>	1.213587e-04	-3.915929
12	25001	neg	4		<s /s>	1.213587e-04	-3.915929
13	25001	neg	5	 Also known under the slightly diff...	<s br br also known under the slightly differe...	2.976193e-181	-180.526339
14	25001	neg	6	Out of the three Airport films I have seen so...	<s out of the three airport films i have seen...	4.499564e-54	-53.346830
15	25001	neg	7	It has my favourite plot of the three with a ...	<s it has my favourite plot of the three with...	0.000000e+00	-399.572650
16	25001	neg	8	While the rather sluggish plot keeps one ente...	<s while the rather sluggish plot keeps one e...	8.638106e-105	-104.063581
17	25001	neg	9	Even when the Navy become involved things don...	<s even when the navy become involved things ...	8.300096e-95	-94.080917
18	25001	neg	10	George Kennedy as the jinxed airline worker J...	<s george kennedy as the jinxed airline worke...	1.666790e-104	-103.778119
19	25001	neg	11	 The home video & theatrical versio...	<s br br the home video theatrical version of...	2.259162e-195	-194.646053

In [31]:

```
print(10**-43.797963)
```

```
1.5923443820846411e-44
```

The unigram model can be similarly used to find the estimated probability of two (or more words) occuring in sequence.

For example, we can compute the probability of words 'to' and 'a' occuring:

In [32]:

```
word_1 = 'to'
word_2 = 'a'

prob_word_1 = word_list[word_list['words'] == word_1]['prob'].iloc[0]
prob_word_2 = word_list[word_list['words'] == word_2]['prob'].iloc[0]
```

```
unigram_prob = prob_word_1*prob_word_2

print('The unigram probability of the word "a" occuring given the word "to" wa
```

```
The unigram probability of the word "a" occuring given the word "to" was the p
revious word is:  0.0004765858
```

Bigram Model (k=2): $P(w_i|w_1w_2...w_{i-1}) \approx P(w_i|w_{i-1})$

Applying this is somewhat more complex, first we find the co-occurances of each word into a word-word matrix. The counts are normalised by the counts of the previous word:

$$P(w_i|w_{i-1}) \approx \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

So, for example, if we wanted to find the P(a|to) we first count the occurances of (to,a) and divide this by the count of (to):

In [33]:

```
word_1 = ' ' + str('to') + ' '
word_2 = str('a') + ' '

bigram_prob = len(re.findall(word_1 + word_2, text_list)) / len(re.findall(wo

print('The probability of the word "a" occuring given the word "to" was the p
```

```
The probability of the word "a" occuring given the word "to" was the previous
word is:  0.02232
```

and likewise, if we change the previous word to 'has':

In [34]:

```
word_1 = ' ' + str('has') + ' '
word_2 = str('a') + ' '

bigram_prob = len(re.findall(word_1 + word_2, text_list)) / len(re.findall(wo

print('The probability of the word "a" occuring given the word "has" was the 
```

```
The probability of the word "a" occuring given the word "has" was the previous
word is:  0.1341
```

We can repeat this calculation for all word pairs to find the most likely word to follow the given word. This of course takes an exceptional amount of time and it may be better to compute this for words as required rather than attempting to do it for all.

In [35]:

```
W_W_Matrix = pd.DataFrame({'words': word_list['words']})


start_time = time.time()


# Add limits to number of columns/rows so this doesn't run for ages
column_lim = 1000
#column_lim = len(W_W_Matrix)
row_lim = 10
#row_lim = len(W_W_Matrix)

for r, column in enumerate(W_W_Matrix['words'][0:column_lim]):

    prob_table = pd.DataFrame()
```

```
    for i, row in enumerate(W_W_Matrix['words'][0:row_lim]):
        #print(row)
        word_1 = ' ' + str(row) + ' '
        word_2 = str(column) + ' '

        if len(re.findall(word_1, text_list)) == 0:
            prob = pd.DataFrame({'prob':[0]}, index=[i])
        else:
            prob = pd.DataFrame({'prob':[len(re.findall(word_1 + word_2, text_

        prob_table = prob_table.append(prob)
    W_W_Matrix[str(column)] = prob_table['prob']

    # Outputs progress of main loop, see:
    clear_output(wait=True)
    print('Proportion of column words completed:', np.round(r/len(W_W_Matrix[

end_time = time.time()
print('Total run time = ', np.round(end_time-start_time,2)/60, ' minutes')
```

```
Proportion of column words completed: 100.0 %
Total run time =  2.3156666666666665  minutes
```

In [36]:

```
W_W_Matrix[W_W_Matrix['a'] >= 0]
```

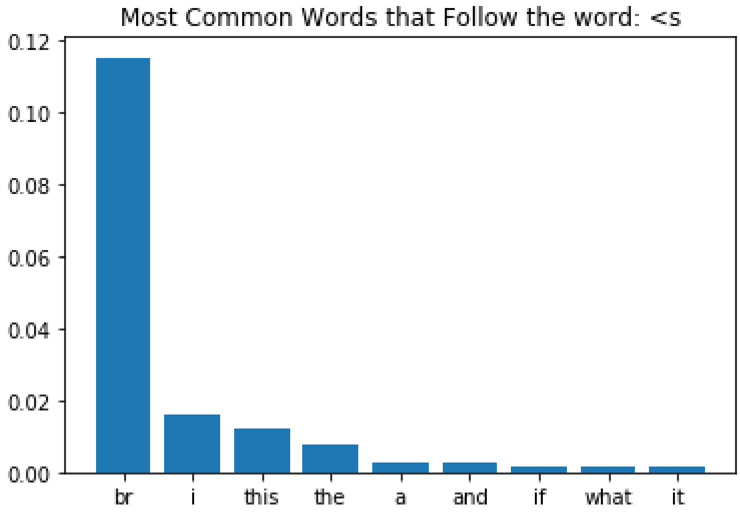
Out[36]:

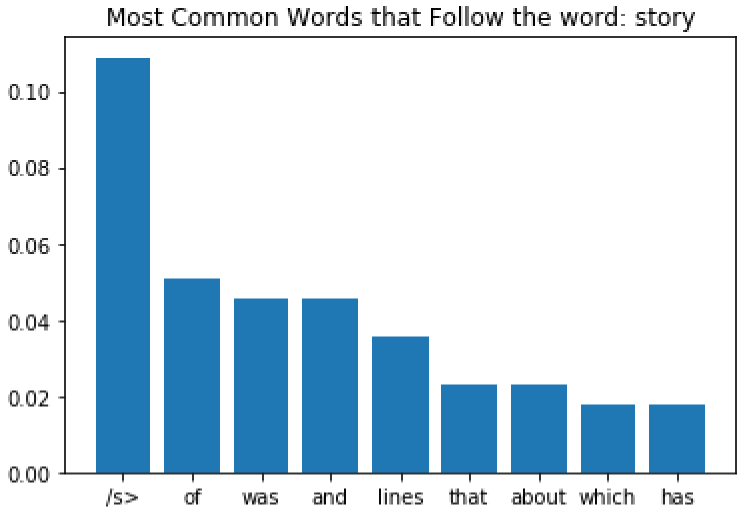
	words	<s	story	of	a	man	who	has	unnatural	fe
0	<s	0.0	0.000000	0.000150	0.002776	0.000075	0.000300	0.000000	0.000000	
1	story	0.0	0.000000	0.050633	0.005063	0.000000	0.000000	0.017722	0.000000	
2	of	0.0	0.000388	0.000194	0.049515	0.000000	0.000777	0.000000	0.000000	
3	a	0.0	0.004060	0.000000	0.000169	0.003553	0.000000	0.000000	0.000000	
4	man	0.0	0.000000	0.000000	0.000000	0.000000	0.122137	0.007634	0.000000	
5	who	0.0	0.000000	0.000000	0.001389	0.000000	0.000000	0.040278	0.000000	
6	has	0.0	0.000000	0.000000	0.134100	0.000000	0.000000	0.000000	0.001916	
7	unnatural	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
8	feelings	0.0	0.000000	0.222222	0.000000	0.000000	0.000000	0.000000	0.000000	
9	for	0.0	0.000000	0.000000	0.097530	0.000000	0.000000	0.000000	0.000000	

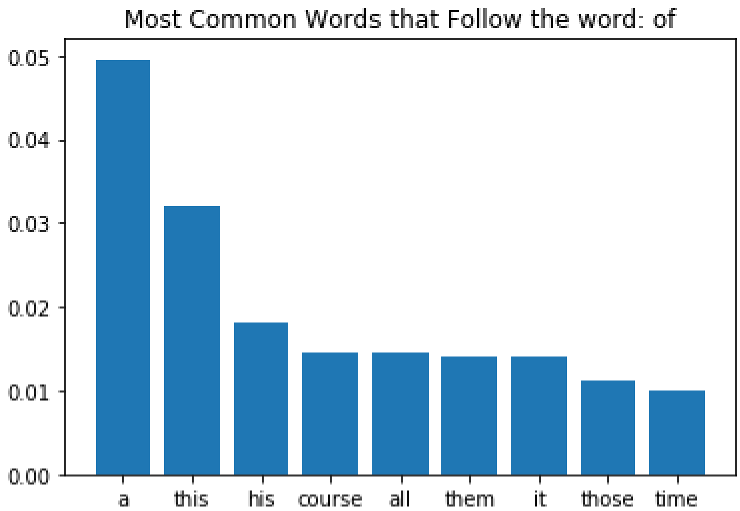
10 rows × 1001 columns

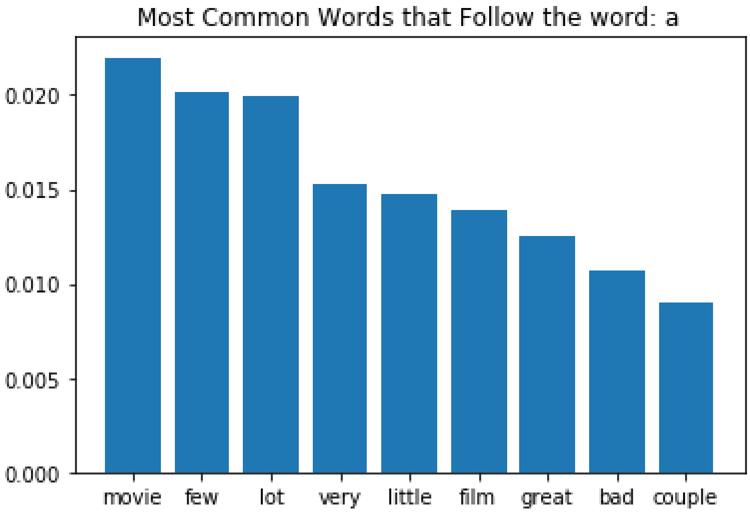
In [37]:

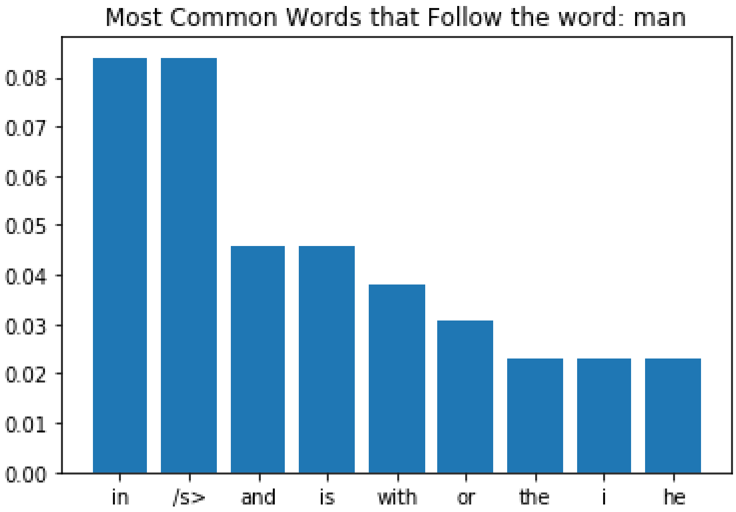
```
for i in range(0,row_lim):
    plt.bar(W_W_Matrix.iloc[i,1:].sort_values(ascending=False)[1:10].index,W_\
    plt.title('Most Common Words that Follow the word: ' +str(W_W_Matrix.iloc
    plt.show()
```

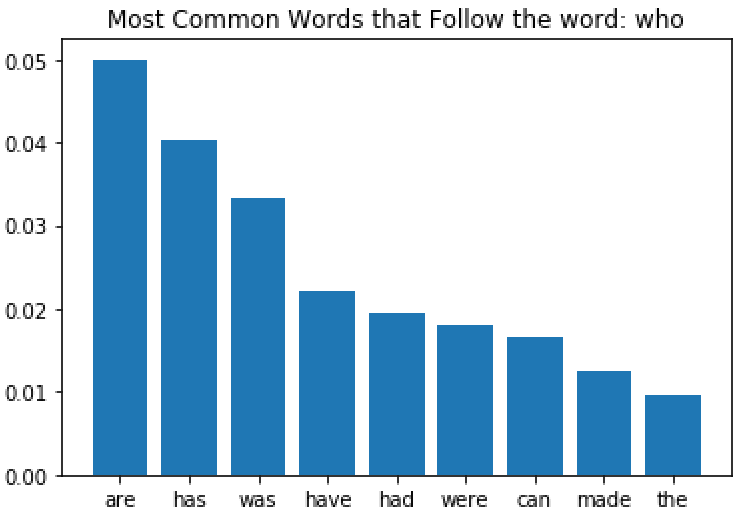

Most Common Words that Follow the word: <s
0.12
0.10
0.08
0.06
0.04
0.02
0.00
br
i
this
the
a
and
if
what
it

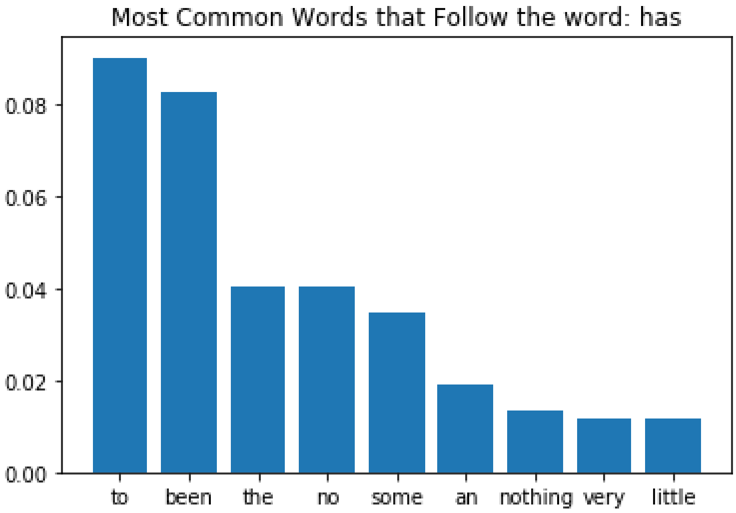

Most Common Words that Follow the word: story
0.10
0.08
0.06
0.04
0.02
0.00
/s>
of
was
and
lines
that
about
which
has

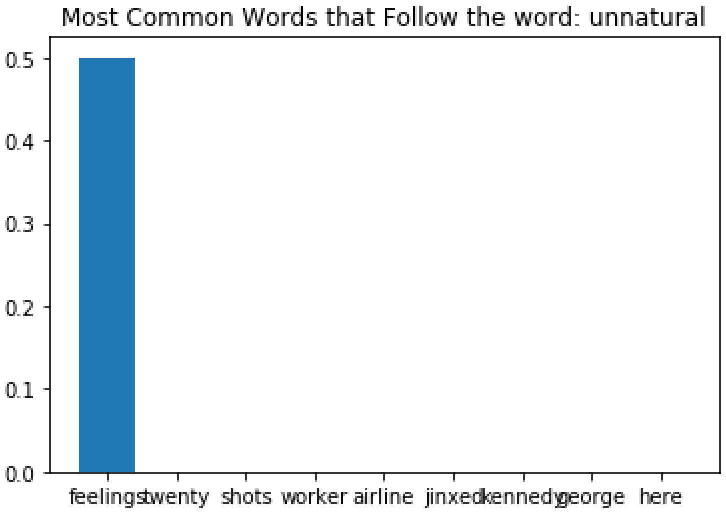

Most Common Words that Follow the word: of
0.05
0.04
0.03
0.02
0.01
0.00
a
this
his
course
all
them
it
those
time

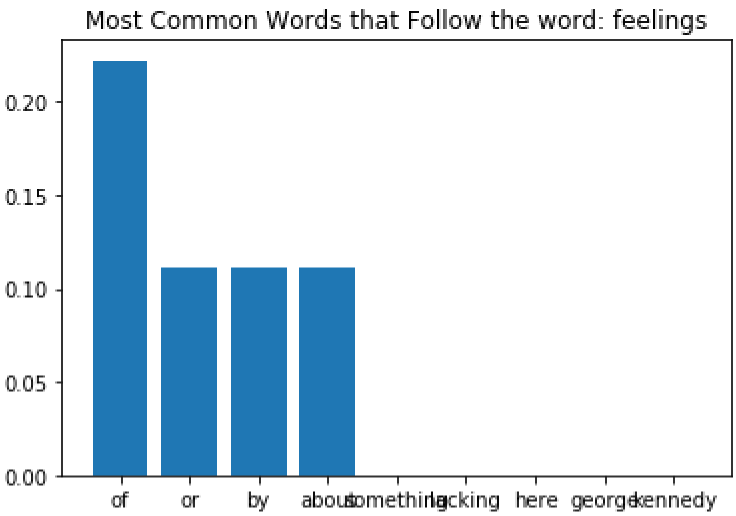

Most Common Words that Follow the word: a
0.000
0.005
0.010
0.015
0.020
movie
few
lot
very
little
film
great
bad
couple

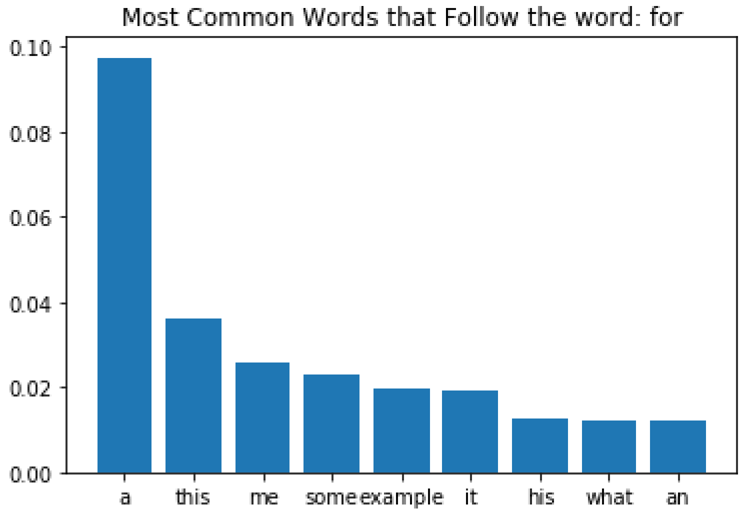

Most Common Words that Follow the word: man
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
in
/s>
and
is
with
or
the
i
he


Most Common Words that Follow the word: who
0.00
0.01
0.02
0.03
0.04
0.05
are
has
was
have
had
were
can
made
the


Most Common Words that Follow the word: has
0.00
0.02
0.04
0.06
0.08
to
been
the
no
some
an
nothing
very
little


Most Common Words that Follow the word: unnatural
0.0
0.1
0.2
0.3
0.4
0.5
feelings
twenty
shots
worker
airline
jinxed
kennedy
george
here


Most Common Words that Follow the word: feelings
0.00
0.05
0.10
0.15
0.20
of
or
by
about
something
lacking
here
george
kennedy



These can be used to form sentences, if we calculate the bigram probabilities 'on the fly' we can find the next most likely word for the given word. To ensure that different sentences are form, we vary the selected colmuns randomly.

Fixing the start and end of the sentence to be the respective < s and /s > notations, we form the following:

NOTE I have made the word cap very small as this takes a significant time to run (2 hours for 5 words) but an example output is shown.

In [38]:

```
W_W_Matrix = pd.DataFrame({'words': word_list['words']})

start_time = time.time()

text_list = " ".join(map(str, text))

# Increasing these take significant time to run but provide more realistic se
num_sentences = 1
sentence_word_limit = 1

#extract start and end of sentence notation so that they are always included
sentence_forms = W_W_Matrix[(W_W_Matrix['words']=='<s') | (W_W_Matrix['words'

sentences_output = pd.DataFrame()
for sample in range(0,num_sentences):

    sentence = pd.DataFrame()

    for i in range(0,sentence_word_limit):
        # if this is the first word, fix it to be start of sentence notation
        if (i==0):
            current_word = str('<s')
        # Randomly select first word after sentence start
        elif (i==1):
            current_word = str(W_W_Matrix[(W_W_Matrix['words']!='<s') ]['word
        else:
            current_word = next_word

        sentence['word_'+str(i)] = [current_word]
        # if we have reached end of sentence, add this sentence to output tab
        if (current_word==str('/s>')):
            sentences_output = sentences_output.append(sentence)
            break
        else:
```

```
            prob_table = pd.DataFrame()

            # randomly select other words form rest of list
            for r, column in enumerate(W_W_Matrix[(W_W_Matrix['words']!='<s')

                next_words = str(column)



                if len(re.findall(' ' + current_word + ' ', text_list)) == 0:
                    prob = pd.DataFrame({'word':str(column),
                        'prob':[0]}, index=[i])
                else:
                    prob = pd.DataFrame({'word':str(column),
                        'prob':[len(re.findall(' ' + current_word + ' ' + nex

                prob_table = prob_table.append(prob)
                # We can reduce the probability of the sentence ending so tha
                reduce_end_prob = 0.5
                prob_table['prob'] = np.where(prob_table['word']=='/s>', prob
                # next word is most probable of this given the current word
                next_word = prob_table[prob_table['prob'] == max(prob_table['

                # Outputs progress of main loop:
                clear_output(wait=True)
                print("Sentence number: ",sample+1)
                print("Words completed in current sentence:",i+1)
                print('Proportion of column words completed:', np.round(r/len


end_time = time.time()
print('Total run time = ', np.round(end_time-start_time,2)/60, ' minutes')
```

```
Sentence number:  1
Words completed in current sentence: 1
Proportion of column words completed: 100.0 %
Total run time =  6.655666666666666  minutes
```

In [39]:

```
for i in range(1,num_sentences):
    print('Sentence ',i,':',sentences_output.iloc[i].values)
```

Sample output sentence:

Sentence 0 : ['< s' 'root' 'for' 'the' 'movie' '/s >']

and we can manually explore each probability, for example:

In [40]:

```
word_1 = str('for')
word_2 = str('the')

bigram_prob = len(re.findall(' ' + word_1 + ' ' + word_2 + ' ', text_list)) /


print('The probability of the word "',word_2,'" occuring given the word "',wo
```

```
The probability of the word " the " occuring given the word " for " was the pr
evious word is:  0.19126029132362254
```

Tri-grams and beyond!

If we continue the estimation equation for trigrams, we have the following:

$$P(x_3 | x_1, x_2) \approx \frac{count(x_1, x_2, x_3)}{count(x_1, x_2)}$$

In [41]:

```
word_1 = str('to')
word_2 = str('a')
word_3 = str('movie')

trigram_prob = (len(re.findall(' ' + word_1 + ' ' + word_2 + ' ' + word_3 + '
                    len(re.findall(' ' + word_1 + ' ' + word_2, text_list)))


print('The probability of the word "',word_3,'" occuring given the word "',wo
```

```
The probability of the word " movie " occuring given the word " to " and " a "
were the previous two words is:  0.011799410029498525
```

In [42]:

```
word_1 = str('to')
word_2 = str('a')
word_3 = str('film')

trigram_prob = (len(re.findall(' ' + word_1 + ' ' + word_2 + ' ' + word_3 + '
                    len(re.findall(' ' + word_1 + ' ' + word_2, text_list)))


print('The probability of the word "',word_3,'" occuring given the word "',wo
```

```
The probability of the word " film " occuring given the word " to " and " a "
were the previous two words is:  0.0058997050147492625
```

Therefore, trigram phrase 'to a movie' is used more commonly than 'to a film' and is the choice our algorithm would take when forming sentences.

The code to systemtically find the most likely next word and form sentences with trigrams can be repeated following the previous bigram computations.

Part 2

V. Training and Testing the Language Models (LMs)

The corpus used to train our LMs will impact the output predictions. Therefore we need to introduce a methodology for evaluating how well our trained LMs perform. The best trained LM is the one that can correctly **predict the next word of setences in an unseen test set.**

This can be time consuming, to build multiple LMs for comparison could take hours to compute. Therefore, we introduce the intrinsic evaluation method of **perplexity**. In short perplexity is a measure of how well a probability distribution or probability model predicts a sample. [3]

Defn: Perplexity

Perplexity is the inverse probability of the test set normalised by the number of words, more specifically can be defined by the following equation:

$$PP(W) = P(w_1 w_2 ... w_N)^{\frac{-1}{N}}$$

e.g. Suppose a sentence consists of random digits [0-9], what is the perplexit of this sentence by a model that asigns an equal probability (i.e. $P = 1/10$) to each digit?

$$PP(W) = (\frac{1}{10} * \frac{1}{10} * ... * \frac{1}{10})^{\frac{-1}{10}} = (\frac{1}{10}^{10})^{\frac{-1}{10}} = \frac{1}{10}^{-1} = 10$$

VI. Entropy in Information Theory

Prerequisite: 정보량 (I)

- 어떤 한 사건(event)에서 기대되는 정보량 (I)을 확률과 관련하여 살펴 봄

중요성(significance): 어떤 사건이 일어날 가능성이 작으면 작을수록, 그 사건은 더 많은 정보를 지닌다.

중요성 조건은 어떤 사건의 확률이 높을수록 이 사건으로 알려지는 정보량은 적어짐을 나타낸다. 따라서 확률값을 역으로 취하여 중요성에 따른 정보량을 나타낼 수 있다.

$P(x1) > P(x2) \Rightarrow I(x1) < I(x2)$

$I(x) = 1/P(x)$

가법성(additivity): 만일 x1, x2 가 독립적인 사건이라면 다음을 만족해야 한다.

$I(x1x2) = I(x1) + I(x2)$

예)

$P(x1) = 1/2$ 이면 $I(x1) = 2$

$P(x2) = 1/4$ 이면 $I(x2) = 4$

따라서 중요성의 조건이 만족된다. 만일 두 사건이 서로 독립적이라면,

$P(x1x2) = P(x1) * P(x2) = 1/2 * 1/4 = 1/8$

$I(x1x2) = 1/P(x1x2) = 8$

그러나 가법성에 따라

$I(x1x2) = I(x1) + I(x2) = 2 + 4 = 6$

이다. 따라서 가법성의 조건이 충족되지 못한다.

두 독립 사건의 확률값은 곱으로 이루어지지만, 두 사건의 결합된 정보 내용은 더해져야만 한다. 정보의 가법성을 위해서 곱이 아닌 더하기가 필요하다. 따라서 이와 유사한 기능을 하는 log를 도입하게 된다

$log(xy) = logx + logy$

즉 확률을 역으로 하여 중요성 기준을 만족시킬 수 있고, log를 사용 하여 가법성의 조건을 만족시킬 수 있다. 이제 이 둘을 결합하면 어떤 확률 변수 x 가 지니는 정보량은 다음과 같이 계산될 수 있다.

$I(x) = log1/P(x) = -logP(x)$

예)

$P(x1) = 1/2$ 이 면 $I(x1) = -log(1/2) = log2 = 1$

$P(x2) = 1/4$ 이 면 $I(x2) = -log(1/4) = log4 = 2$

$P(x1) > P(x2) \Rightarrow I(x1) < I(x2)$ 인 중요성의 조건을 만족한다.

$P(x1x2) = P(x1)P(x2) = 1/8 \Rightarrow I(x1x2) = -log(1/8) = 3$

$I(x1x2) = I(x1) + I(x2) = 1 + 2 = 3$

으로 가법성의 조건도 만족하게 된다.

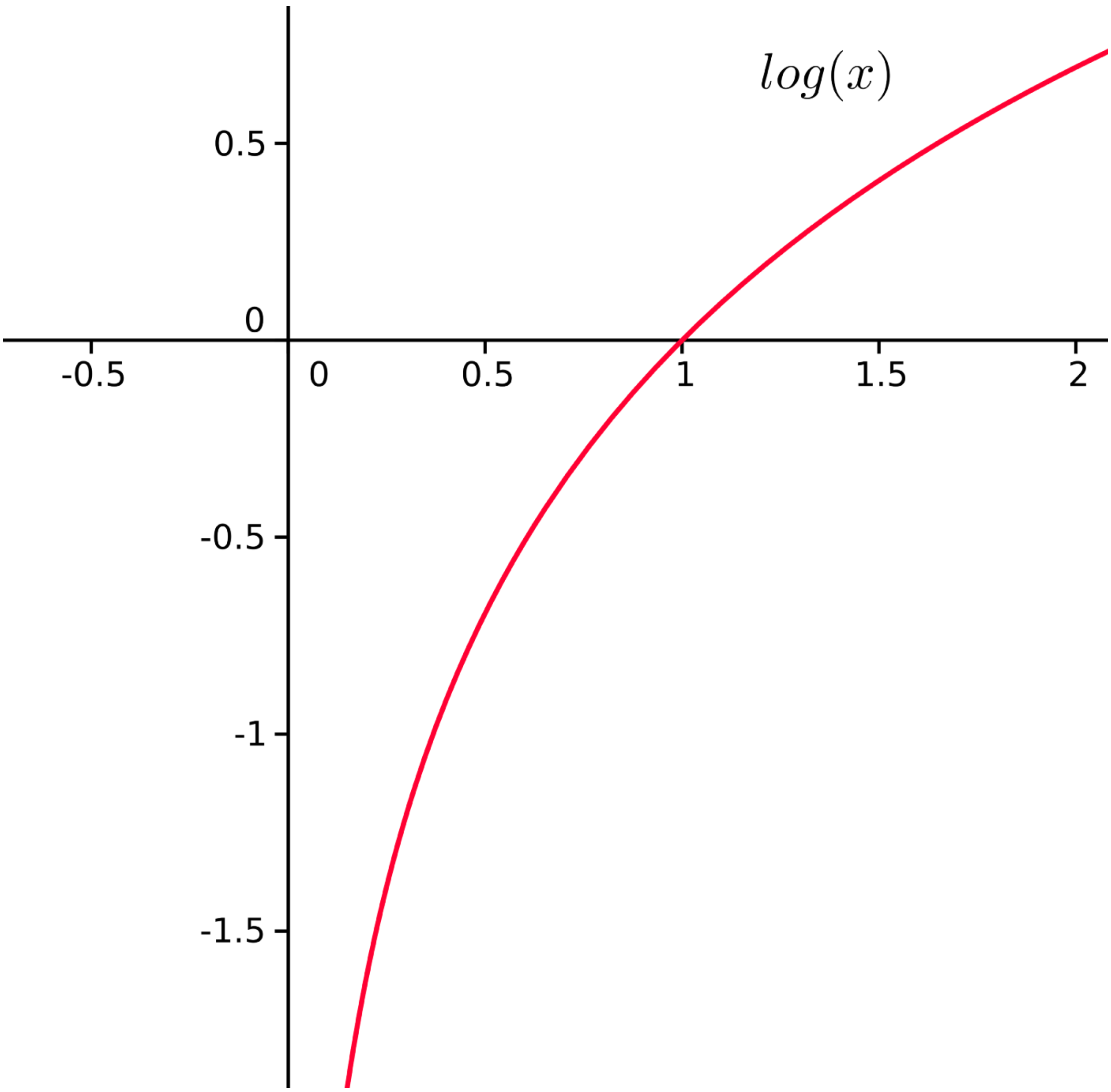
In Information Theory, entropy (denoted $H(X)$) of a random variable X is the expected log probabiltiy:

$$H(X) = -\sum P(x)log_2P(x)$$

and is a measure of uncertainty. [4]

Reason for negative sign:

- log(p(x))<0 for all p(x) in (0,1) . p(x) is a probability distribution and therefore the values must range between 0 and 1.



A plot of log(x). For x values between 0 and 1, log(x) <0 (is negative)

In other words, entropy is the number of possible states that a system can be.

Entropy of a bias coin toss

Say we have the probabilities of heads and tails in a coin toss defined by:

- $P(heads) = p$
- $P(tails) = 1 - p$

Then the entropy of this is:

$$H(X) = -\sum P(x)log_2P(x) = -[plog_2p + (1-p)log_2(1-p)]$$

If the coin is fair, i.e. p = 0.5, then we have:

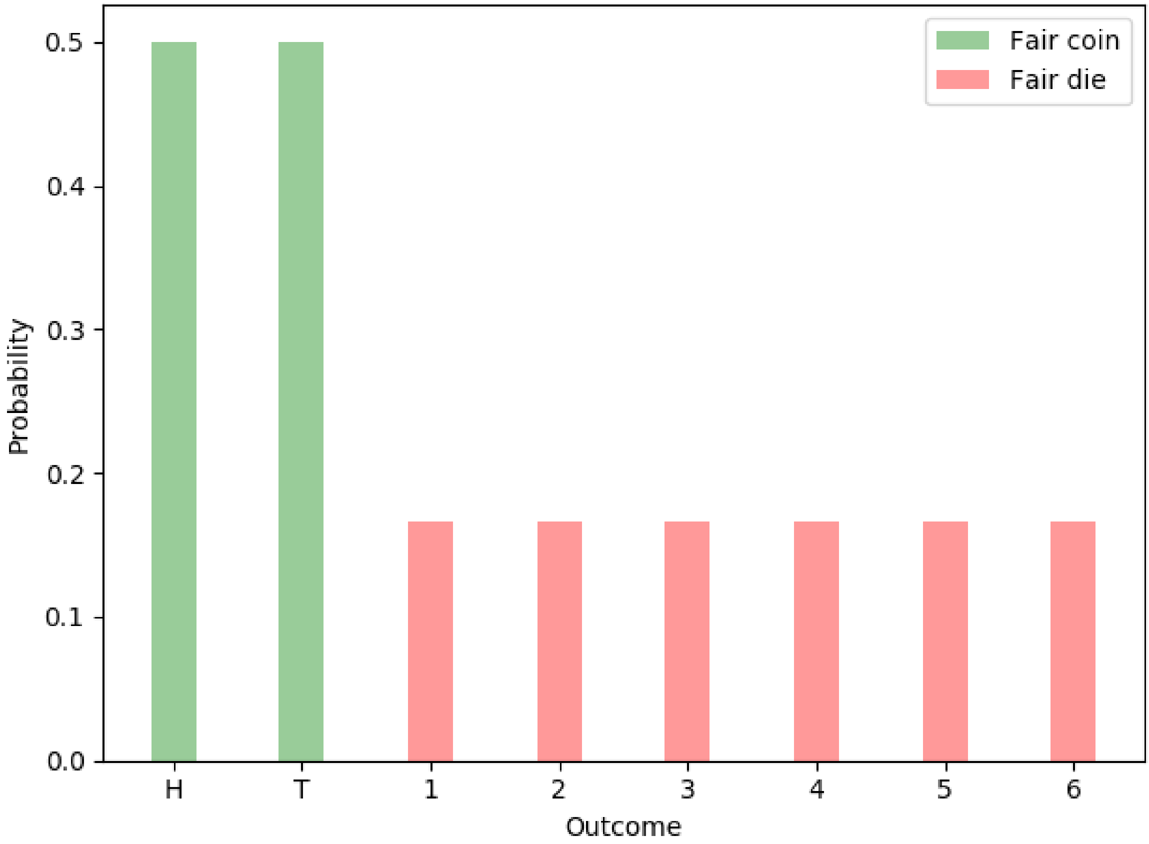
$$H(X) = -[0.5log_2 0.5 + (1-0.5)log_2(1-0.5)] = -[-0.5-0.5] = 1$$

The full entropy distibution over varying bias probabilities is shown below.

[3] https://en.wikipedia.org/wiki/Perplexity [4] https://en.wikipedia.org/wiki/Entropy_(information_theory))

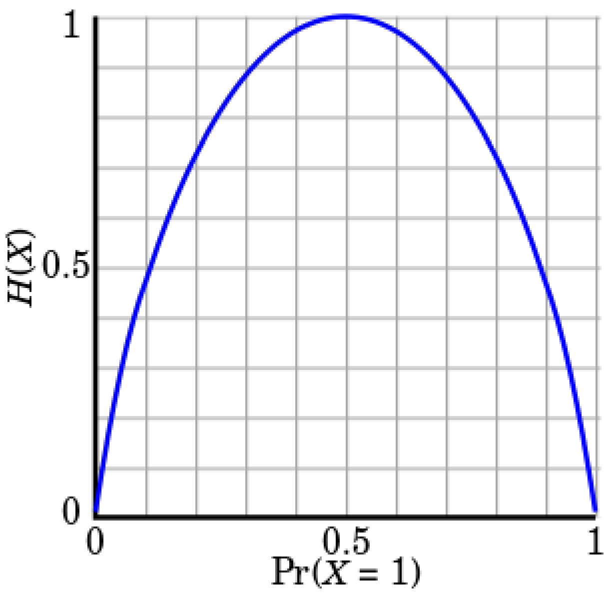
Uniform distributions have maximum uncertainty

If your goal is to minimize uncertainty, stay away from **uniform probability distributions**.



uniform distributions have maximum entropy for a given numger of outcomes

Here is the plot of the Entropy function as applied to Bernoulli trials (events with two possible outcomes and probabilities p and 1-p):



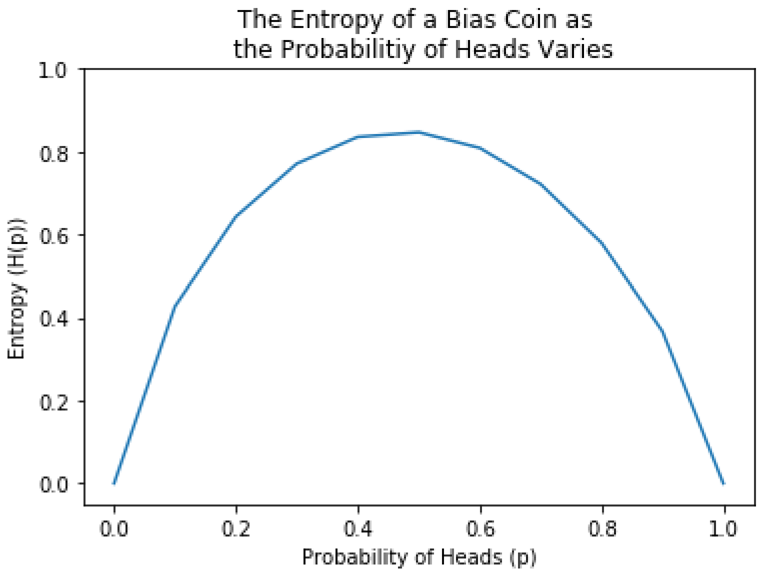
In [5]:

```
import numpy as np
import math
import matplotlib.pyplot as plt
p = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
H = [-(p*np.log2(p) + (1-p)*np.log(1-p)) for p in p]
# Replace nan output with 0
```

```
H = [0 if math.isnan(x) else x for x in H]

plt.plot(p,H)
plt.xlim([-0.05,1.05])
plt.ylim([-0.05,1])
plt.xlabel('Probability of Heads (p)')
plt.ylabel('Entropy (H(p))')
plt.title('The Entropy of a Bias Coin as \n the Probabilitiy of Heads Varies'
```

```
/home/hpshin/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5: RuntimeWarning: divide by zero encountered in log2
  """
/home/hpshin/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5: RuntimeWarning: invalid value encountered in double_scalars
  """
/home/hpshin/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5: RuntimeWarning: divide by zero encountered in log
  """
```

Out[5]:
```
Text(0.5,1,'The Entropy of a Bias Coin as \n the Probabilitiy of Heads Varies')
```



VII. Entropy of Language

1. Entropy of a sequence of words:

$$H(w_1 w_2 ... w_n) = - \sum_{w_1 ... wn} P(w_1 ... w_n) log_2 P(w_1 ... w_n)$$

2. The per-word entropy rate of a sequence of words

$$\frac{1}{n} H(w_1 w_2 ... w_n) = \frac{-1}{n} \sum_{w1...wn} P(w_1 ... w_n) log_2 P(w_1 ... w_n), 2)$$

3. Entropy of a language $L = \{w_1 ... w_n | 1 < n < \infty\}$:

$$H(L) = - \lim_{n \to \infty} \frac{1}{n} H(w_1 ... w_n)$$

$$H(L) = \lim_{n\to\infty} -\frac{1}{n}\log p(w_1 w_2 \ldots w_n)$$

Defn: Cross Entropy

The cross entropy, H(p,m), of a true distribution **p** and a model distribution **m** is defined as:

$$H(p,m) = -\sum_x p(x) log_2 m(x)$$

The lower the cross entropy is the closer it is to the true distribution.

Defn: Cross Entropy of a Sequence of Words

$$H(p,m) = -\lim_{n\to\infty} \frac{1}{n} \sum_{w1\ldots wn} p(w_1 \ldots w_n) log_2 m(w_1 \ldots w_n)$$

$$H(W) = -\frac{1}{N}\log P(w_1 w_2 \ldots w_N)$$

VIII. Perplexity and Entropy

$$H(W) = -\frac{1}{N}\log P(w_1 w_2 \ldots w_N)$$

$$\begin{aligned} \text{Perplexity}(W) &= 2^{H(W)} \\ &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}} \end{aligned}$$

In [43]:

```
sent_1 = text.iloc[0]
sent_2 = text.iloc[1]

print('Sentence 1', sent_1)
print('--.--.--.--.--.--.--.--')
print('Sentence 2', sent_2)
```

```
Sentence 1 <s story of a man who has unnatural feelings for a pig /s>
--.--.--.--.--.--.--.--
Sentence 2 <s  starts out with a opening scene that is a terrific example of a
bsurd comedy  /s>
```

In [44]:

```
data_prob = word_list[['words','count','prob']]
data_prob.head()
```

Out[44]:

words	count	prob

	words	count	prob
0	<s	13330	0.052733
1	story	395	0.001563
2	of	5150	0.020373
3	a	5911	0.023384
4	man	131	0.000518

In [45]:

```
def entropy(sentence, data_prob):
    entropy_table = pd.DataFrame()
    for n,word in enumerate(sentence.split(' ')):
        # log2(0) provide nan so return 0 instead
        if ((data_prob[data_prob['words']==word]['prob'].iloc[0]) == 0):
            entropy = 0
        else:
            prob = data_prob[data_prob['words']==word]['prob'].iloc[0]
            entropy = prob*np.log2(prob)
        entropy_table = entropy_table.append(pd.DataFrame({'word':word,
                                                           'entropy':entropy
    phrase_entropy = -1*sum(entropy_table['entropy'])
    return(phrase_entropy)
```

In [46]:

```
sent_1_entropy = entropy(sent_1,data_prob)
sent_2_entropy = entropy(sent_2,data_prob)

print('Sentence 1: ', sent_1)
print('Sentence 1 entropy: ', np.round(sent_1_entropy,5))
print('Per-word Sentence 1 entropy: ', np.round(sent_1_entropy/len(sent_1.spl

print('--.--.--.--.--.--.--.--')
print('Sentence 2: ', sent_2)
print('Sentence 2 entropy: ', np.round(sent_2_entropy,5))
print('Per-word Sentence 2 entropy: ', np.round(sent_2_entropy/len(sent_2.spl
```

```
Sentence 1:  <s story of a man who has unnatural feelings for a pig /s>
Sentence 1 entropy:  0.92476
Per-word Sentence 1 entropy:  0.07114
--.--.--.--.--.--.--.--
Sentence 2:  <s  starts out with a opening scene that is a terrific example of
absurd comedy /s>
Sentence 2 entropy:  1.26142
Per-word Sentence 2 entropy:  0.0742
```

Likewise, we can calculate the perplexity of each sentence:

In [47]:

```
sent_1_perplex = 2**sent_1_entropy
sent_2_perplex = 2**sent_2_entropy

print('Sentence 1: ', sent_1)
print('Sentence 1 entropy: ', np.round(sent_1_entropy,5))
print('Per-word Sentence 1 entropy: ', np.round(sent_1_entropy/len(sent_1.spl
print('Sentence 1 Perplexity: ', sent_1_perplex)

print('--.--.--.--.--.--.--.--')
print('Sentence 2: ', sent_2)
print('Sentence 2 entropy: ', np.round(sent_2_entropy,5))
print('Per-word Sentence 2 entropy: ', np.round(sent_2_entropy/len(sent_2.spl
print('Sentence 2 Perplexity: ', sent_2_perplex)
```

```
Sentence 1:  <s story of a man who has unnatural feelings for a pig /s>
Sentence 1 entropy:  0.92476
Per-word Sentence 1 entropy:  0.07114
Sentence 1 Perplexity:  1.8983676328384849
--.--.--.--.--.--.--.--
Sentence 2:  <s  starts out with a opening scene that is a terrific example of
absurd comedy /s>
Sentence 2 entropy:  1.26142
Per-word Sentence 2 entropy:  0.0742
Sentence 2 Perplexity:  2.3973163400582496
```

Which we can use to find the probability of the whole sentence as:

$$PP(W) = P(w_1 w_2 ... w_N)^{\frac{-1}{N}}$$

In [48]:

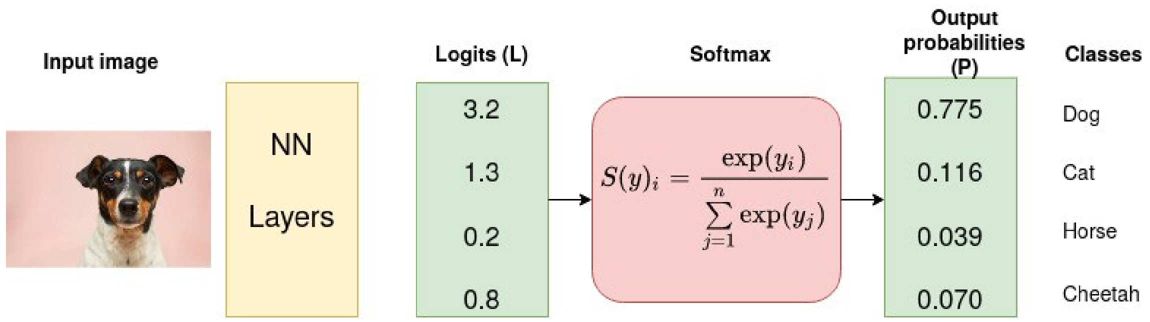
```python
sent_1_prob = (1/sent_1_perplex)**len(sent_1.split(' '))
sent_2_prob = (1/sent_2_perplex)**len(sent_2.split(' '))

print('Sentence 1 Probability: ', '%0.10f' % sent_1_prob)
print('Sentence 2 Probability: ', '%0.10f' % sent_2_prob   )
```

```
Sentence 1 Probability:  0.0002404672
Sentence 2 Probability:  0.0000003505
```

Cross-Entropy Loss Function

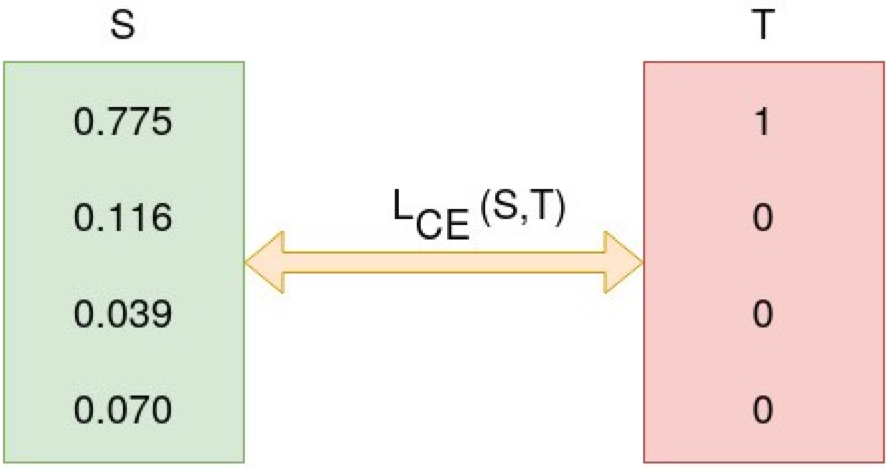
- Also called logarithmic loss, log loss or logistic loss. Each predicted class probability is compared to the actual class desired output 0 or 1 and a score/loss is calculated that penalizes the probability based on how far it is from the actual expected value.
- The penalty is logarithmic in nature yielding a large score for large differences close to 1 and small score for small differences tending to 0.

- Consider a 4-class classification task where an image is classified as either a dog, cat, horse or cheetah.



- In the above Figure, Softmax converts logits into probabilities.

- The purpose of the Cross-Entropy is to take the output probabilities (P) and measure the distance from the truth values (as shown in Figure below).



- For the example above the desired output is [1,0,0,0] for the class dog but the model outputs [0.775, 0.116, 0.039, 0.070] .
- The objective is to make the model output be as close as possible to the desired output (truth values).
- During model training, the model weights are iteratively adjusted accordingly with the aim of minimizing the Cross-Entropy loss.
- The process of adjusting the weights is what defines model training and as the model keeps training and the loss is getting minimized, we say that the model is learning.

- Cross-entropy loss is used when adjusting model weights during training.
- The aim is to minimize the loss, i.e, the smaller the loss the better the model. A perfect model has a cross-entropy loss of 0.
- Cross-entropy is defined as

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Binary Cross-Entropy Loss

- For binary classification, we have binary cross-entropy defined as

$$
\begin{aligned}
L &= -\sum_{i=1}^{2} t_i \log(p_i) \\
&= -\left[t \log(p) + (1-t)\log(1-p)\right]
\end{aligned}
$$

where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} class.

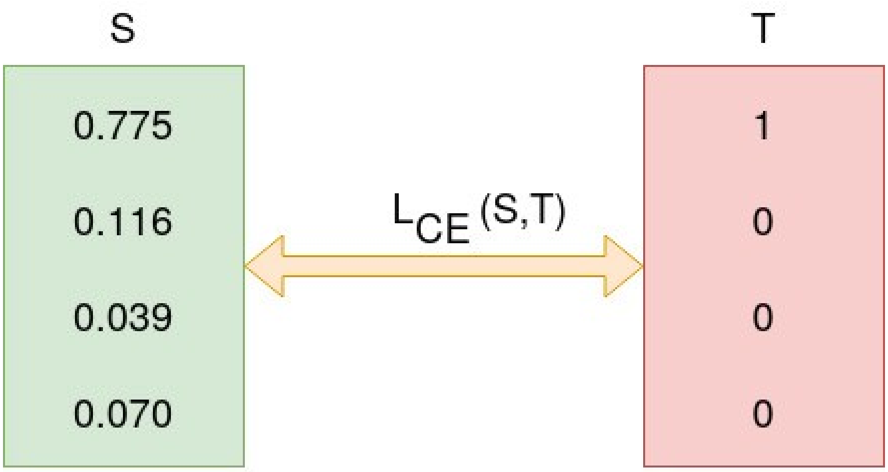
- Binary cross-entropy is often calculated as the average cross-entropy across all data examples

$$
L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1-t_j)\log(1-p_j)\right]\right]
$$

for N data points where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} data point.

Example

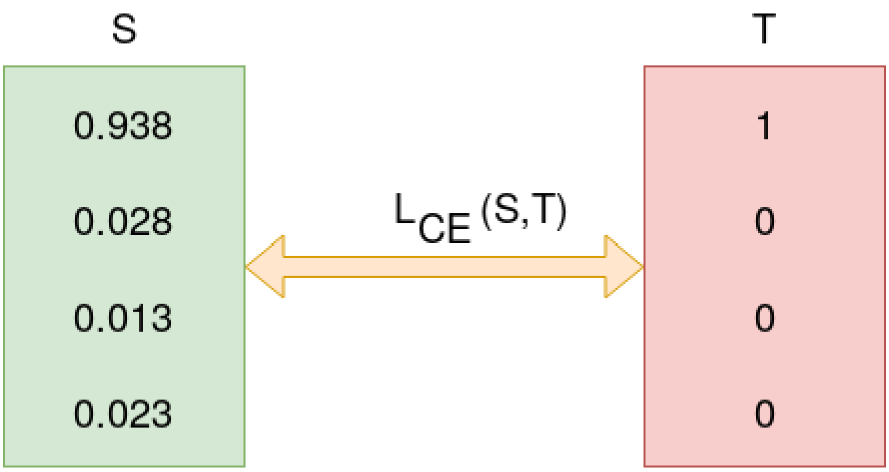
- Consider the classification problem with the following Softmax probabilities (S) and the labels (T).
- The objective is to calculate for cross-entropy loss given these information.



- The categorical cross-entropy is computed as follows

$$
\begin{aligned}
L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\
&= -[1\log_2(0.775) + 0\log_2(0.126) + 0\log_2(0.039) + 0\log_2(0.070)] \\
&= -\log_2(0.775) \\
&= 0.3677
\end{aligned}
$$

- Softmax is continuously differentiable function.
- This makes it possible to calculate the derivative of the loss function with respect to every weight in the neural network.
- This property allows the model to adjust the weights accordingly to minimize the loss function (model output close to the true values).
- Assume that after some iterations of model training the model outputs the following vector of logits



$$
\begin{aligned}
L_{CE} &= -1\log_2(0.936) + 0 + 0 + 0 \\
&= 0.095
\end{aligned}
$$

- 0.095 is less than previous loss, that is, 0.3677 implying that the model is learning.
- The process of optimization (adjusting weights so that the output is close to true values) continues until training is over.

Part 3

Challenges in Fitting LMs

Due to the output of LMs is dependent on the training corpus, N-grams only work well if the training corpus is similar to the testing dataset and we risk overfitting in training.

As with any machine learning method, we would like results that are generalisable to new information.

Even harder is how we deal with words that do not even appear in training but are in the test data.

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

IX. Dealing with Zero Counts in Training: Laplace +1 Smoothing

To deal with words that are unseen in training we can introduce add-one smoothing. To do this, we simply add one to the count of each word.

This shifts the distribution slightly and is often used in text classification and domains where the number of zeros isn't large. However, this is not often used for n-grams, instead we use more complex methods.

First, let us create a dummy training corpus and test set from the original data:

In [49]:
```
train_data_sent.head()
```

Out[49]:

	id	pol	sent_id	sentence	sentence_clean	unigram	unigram_log
0	25000	neg	0	Story of a man who has unnatural feelings for ...	<s story of a man who has unnatural feelings f...	1.027054e-36	-35.988407
1	25000	neg	1	Starts out with a opening scene that is a ter...	<s starts out with a opening scene that is a ...	1.592345e-44	-43.797963
2	25000	neg	2	A formal orchestra audience is turned into an...	<s a formal orchestra audience is turned into...	8.442173e-68	-67.073546
3	25000	neg	3	Unfortunately it stays absurd the WHOLE time ...	<s unfortunately it stays absurd the whole ti...	1.243502e-61	-60.905353
4	25000	neg	4	Even those from the era should be turned off	<s even those from the era should be turned o...	7.841752e-31	-30.105587

In [50]:
```
corpus = train_data_sent['sentence_clean'][:int(np.round(len(train_data_sent)
test = train_data_sent['sentence_clean'][int(np.round(len(train_data_sent)*0.

corpus_list = " ".join(map(str, corpus))
test_list = " ".join(map(str, test))
```

In [51]:
```
# Corpus word probabilities
corpus_word_list = pd.DataFrame({'words':corpus.str.split(' ', expand = True)
corpus_word_count_table = pd.DataFrame()
for n,word in enumerate(corpus_word_list['words']):
    # Create a list of just the word we are interested in, we use regular exp
    # e.g. 'ear' would be counted in each appearance of the word 'year'
    corpus_word_count = len(re.findall(' ' + word + ' ', corpus_list))
    corpus_word_count_table = corpus_word_count_table.append(pd.DataFrame({'c

    clear_output(wait=True)
    print('Proportion of words completed:', np.round(n/len(corpus_word_list),

corpus_word_list['count'] = corpus_word_count_table['count']
# Remove the count for the start and end of sentence notation so
# that these do not inflate the other probabilities


#corpus_word_list['count'] = np.where(corpus_word_list['words'] == '<s' , 0,
#                 np.where(corpus_word_list['words'] == '/s>', 0,
#                 corpus_word_list['count']))
corpus_word_list['prob'] = corpus_word_list['count']/sum(corpus_word_list['co
```

```
Proportion of words completed: 99.99 %
```

In [52]:
```
corpus_word_list.head()
```

Out[52]:

	words	count	prob
0	<s	11997	0.052146
1	story	354	0.001539
2	of	4633	0.020138

	words	count	prob
3	a	5422	0.023567
4	man	118	0.000513

In [53]:

```
# Test set word probabilities
test_word_list = pd.DataFrame({'words':test.str.split(' ', expand = True).sta
test_word_count_table = pd.DataFrame()
for n,word in enumerate(test_word_list['words']):
    # Create a list of just the word we are interested in, we use regular exp
    # e.g. 'ear' would be counted in each appearance of the word 'year'
    test_word_count = len(re.findall(' ' + word + ' ', test_list))
    test_word_count_table = test_word_count_table.append(pd.DataFrame({'count

    clear_output(wait=True)
    print('Proportion of words completed:', np.round(n/len(test_word_list),4)

test_word_list['count'] = test_word_count_table['count']
# Remove the count for the start and end of sentence notation so
# that these do not inflate the other probabilities

#test_word_list['count'] = np.where(test_word_list['words'] == '<s' , 0,
#                      np.where(test_word_list['words'] == '/s>', 0,
#                      test_word_list['count']))
test_word_list['prob'] = test_word_list['count']/sum(test_word_list['count'])
```

```
Proportion of words completed: 99.97 %
```

In [54]:

```
test_word_list.head()
```

Out[54]:

	words	count	prob
0	<s	1331	0.058678
1		1131	0.049861
2	scott	4	0.000176
3	ciarã n	1	0.000044
4	hinds	2	0.000088

In [55]:

```
# Merge corpus counts to test set and replace missing values with 0
test_word_list_2 = test_word_list.merge(corpus_word_list[['words','count']], 
test_word_list_2['count_y'].fillna(0, inplace=True)

test_word_list_2.head()
```

Out[55]:

	words	count_x	prob	count_y
0	<s	1331	0.058678	11997.0
1		1131	0.049861	9900.0
2	scott	4	0.000176	14.0
3	ciarã n	1	0.000044	0.0
4	hinds	2	0.000088	1.0

In [56]:

```
print('Percentage of words in test set that are not contained in corpus',len(
```

```
Percentage of words in test set that are not contained in corpus 23.6793204141
22643 %
```

In [57]:

```
# Extract missing words from training set
missing_words = test_word_list_2[test_word_list_2['count_y']==0]
missing_words = missing_words[(missing_words['words']!='<s')&(missing_words['
missing_words = missing_words[['words']]
missing_words['count'] = 0
missing_words['prob'] = 0
missing_words.head()
```

Out[57]:

	words	count	prob
3	ciarã n	0	0
6	toby	0	0
7	stephens	0	0
40	invent	0	0
45	modernised	0	0

Adjusted Counting

- I added Adjusted count for better understanding

$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N+V}$$

MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Add-1 estimation is a blunt instrument

So add-1 isn't used for N-grams:
- We'll see better methods

But add-1 is used to smooth other NLP models
- For text classification
- In domains where the number of zeros isn't so huge.

In [58]:

```
# Add missing words onto end of corpus word list and apply laplace +1 smoothi
```

```
corpus_word_list_fixed = corpus_word_list.append(missing_words)

corpus_word_list_fixed['count+1'] = corpus_word_list_fixed['count']+1
corpus_word_list_fixed['prob+1'] = corpus_word_list_fixed['count+1']/sum(corp
# I added below to check the ratio
corpus_word_list_fixed['adjustedCount'] = corpus_word_list_fixed['count+1'] *
corpus_word_list_fixed['checkProb'] = corpus_word_list_fixed['count+1'] / (su
corpus_word_list_fixed.head(100)
```

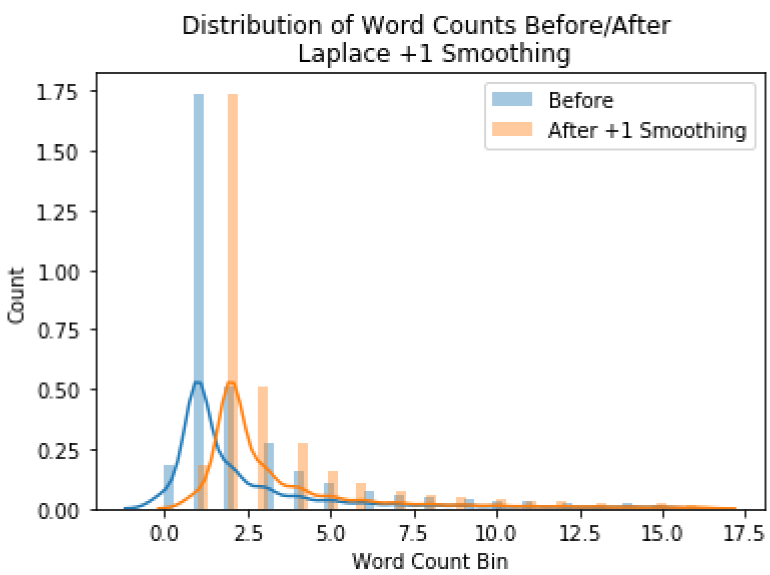
Out[58]:

	words	count	prob	count+1	prob+1	adjustedCount	checkProb
0	<s	11997	0.052146	11998	0.048486	11155.067916	0.048486
1	story	354	0.001539	355	0.001435	330.059102	0.001435
2	of	4633	0.020138	4634	0.018727	4308.433466	0.018727
3	a	5422	0.023567	5423	0.021915	5042.001443	0.021915
4	man	118	0.000513	119	0.000481	110.639530	0.000481
...	...	...	...	...	...	...	...
95	as	1373	0.005968	1374	0.005553	1277.468188	0.005553
96	brand	5	0.000022	6	0.000024	5.578464	0.000024
97	new	119	0.000517	120	0.000485	111.569274	0.000485
98	luxury	6	0.000026	7	0.000028	6.508208	0.000028
99	747	3	0.000013	4	0.000016	3.718976	0.000016

100 rows × 7 columns

In [59]:

```
# Plot distribution before and after Laplace +1 Smoothing
sns.distplot(corpus_word_list_fixed[corpus_word_list_fixed['count']<=15]['cou
sns.distplot(corpus_word_list_fixed[corpus_word_list_fixed['count']<=15]['cou

plt.legend()
plt.title('Distribution of Word Counts Before/After \n Laplace +1 Smoothing')
plt.xlabel('Word Count Bin')
plt.ylabel('Count')
plt.show()
```



X. Futher Smoothing Methods

Laplace +1 smoothing is used in text classification and domains where the number of zeros isn't large. However, it is not often used for n-grams, some better smothing methods for n-grams are:

- Add-k Laplace Smoothing
- Good-Turing
- Kenser-Ney
- Witten-Bell

Part 4

Selecting the Language Model to Use

We have introduced the first three LMs (unigram, bigram and trigram) but which is best to use?

Trigrams are generally provide better outputs than bigrams and bigrams provide better outputs than unigrams but as we increase the complexity the computation time becomes increasingly large. Furthermore, the amount of data available decreases as we increase n (i.e. there will be far fewer next words available in a 10-gram than a bigram model).

XI. Back-off Method: Use trigrams (or higher n model) if there is good evidence to, else use bigrams (or other simpler n-gram model).

XII. Interpolation: Use a mixture of n-gram models

Defn: Simple Interpolation:

$$P(w_3 | w_1, w_2) = \lambda_1 P(w3 | w_1, w_2) + \lambda_2 P(w_3 | w_2) + \lambda_3 P(w_3)$$

where $\sum_i \lambda_i = 1$.

Defn: Contidional Context Interpolation:

$$P(w_3 | w_1, w_2) = \lambda_1(w_1^2) P(w3 | w_1, w_2) + \lambda_2(w_1^2) P(w_3 | w_2) + \lambda_3(w_1^2) P(w_3)$$

Calculating λs:

Using a held-out subset of the corpus (validation set), find λs that maximise the probability of the held out data:

$$P(w_1, w_2, \ldots, w_n | M(\lambda_1, \lambda_2, \ldots, \lambda_k)) = \sum_i log P_{M(\lambda_1, \lambda_2, \ldots, \lambda_k)}(w_i | w_{i-1})$$

Where unknown words are assigned an unknown word token '< Unk >'.

Small Interpolation Example

Say we are given the following corpus:

- < s I am Sam /s >
- < s Sam I am /s >
- < s I am Sam /s >
- < s I do not like green eggs and Sam /s >

Using linear interpolation smoothing with a bigram and unigram model with $\lambda_1 = \frac{1}{2}$ and $\lambda_2 = \frac{1}{2}$, what is $P(Sam|am)$? (note: include '< s' and '/s >' in calculations)

Using the following equation:

$$P(w_2|w_1) = lambda_1 P(w_2|w1) + lambda_2 P(w2)$$

We have in our case:

$$P(Sam|am) = \frac{1}{2}P(Sam|am) + \frac{1}{2}P(Sam)$$

where

$$P(Sam|am) = \frac{count(am, Sam)}{count(am)} = \frac{2}{3}$$

and

$$P(Sam) = \frac{count(Sam)}{Total\ num\ words} = \frac{4}{25}$$

Therefore,

$$P(Sam|am) = \frac{1}{2} * \frac{2}{3} + \frac{1}{2} * \frac{4}{25} \approx 0.413$$

Interpolation Example with IMDB Data

Say we start with the corpus defined in the previous part, we again take a small subset of this as the 'hold-out' set.

In [60]:
```
corpus.head()
```

Out[60]:
```
0    <s story of a man who has unnatural feelings f...
1    <s  starts out with a opening scene that is a ...
2    <s  a formal orchestra audience is turned into...
3    <s  unfortunately it stays absurd the whole ti...
4    <s  even those from the era should be turned o...
Name: sentence_clean, dtype: object
```

In [61]:
```
corpus_2 = corpus[:int(np.round(len(corpus)*0.9,0))]
hold_out = corpus[int(np.round(len(corpus)*0.9,0))+1:]
```

```
corpus_2_list = " ".join(map(str, corpus_2))
hold_out_list = " ".join(map(str, hold_out))
```

In [62]:

```
hold_out.head()
#hold_out_word_list = pd.DataFrame({'words':hold_out.str.split(' ', expand =
#hold_out_word_list.head(20)
```

Out[62]:

```
10799                <s  this action on roberts and dr /s>
10800     <s  farradys part has numar faint dead in his ...
10801     <s  it later turns out that numar somehow was ...
10802     <s br br the movie has numar dressed in what l...
10803     <s  this bloodsucking adventure by numar with ...
Name: sentence_clean, dtype: object
```

In [63]:

```
# hold out set word probabilities
hold_out_word_list = pd.DataFrame({'words':hold_out.str.split(' ', expand = T
hold_out_word_count_table = pd.DataFrame()
for n,word in enumerate(hold_out_word_list['words']):
    # Create a list of just the word we are interested in, we use regular exp
    # e.g. 'ear' would be counted in each appearance of the word 'year'
    hold_out_word_count = len(re.findall(' ' + word + ' ', hold_out_list))
    hold_out_word_count_table = hold_out_word_count_table.append(pd.DataFrame

    clear_output(wait=True)
    print('Proportion of words completed:', np.round(n/len(hold_out_word_list

hold_out_word_list['count'] = hold_out_word_count_table['count']
# Remove the count for the start and end of sentence notation so
# that these do not inflate the other probabilities

hold_out_word_list['count'] = np.where(hold_out_word_list['words'] == '<s' ,
                     np.where(hold_out_word_list['words'] == '/s>', 0,
                     hold_out_word_list['count']))

hold_out_word_list['prob'] = hold_out_word_list['count']/sum(hold_out_word_li
```

```
Proportion of words completed: 99.98 %
```

In [64]:

```
hold_out_word_list.head(10)
```

Out[64]:

	words	count	prob
0	<s	0	0.000000
1		946	0.044513
2	this	310	0.014587
3	action	11	0.000518
4	on	103	0.004847
5	roberts	13	0.000612
6	and	526	0.024751
7	dr	1	0.000047
8	/s>	0	0.000000
9	farradys	1	0.000047

In [65]:

```
hold_out_Matrix = pd.DataFrame({'words': hold_out_word_list['words']})
```

```
start_time = time.time()


# Add limits to number of columns/rows so this doesn't run for ages
column_lim = 100
#column_lim = len(W_W_Matrix)
row_lim = 10
#row_lim = len(W_W_Matrix)

for r, column in enumerate(hold_out_Matrix['words'][0:column_lim]):

    prob_table = pd.DataFrame()
    for i, row in enumerate(hold_out_Matrix['words'][0:row_lim]):

        word_1 = ' ' + str(row) + ' '
        word_2 = str(column) + ' '

        if len(re.findall(word_1, hold_out_list)) == 0:
            prob = pd.DataFrame({'prob':[0]}, index=[i])
        else:
            prob = pd.DataFrame({'prob':[len(re.findall(word_1 + word_2, hold_

        prob_table = prob_table.append(prob)
    hold_out_Matrix[str(column)] = prob_table['prob']

    # Outputs progress of main loop, see:
    clear_output(wait=True)
    print('Proportion of column words completed:', np.round(r/len(hold_out_Ma

end_time = time.time()
print('Total run time = ', np.round(end_time-start_time,2)/60, ' minutes')
```

```
Proportion of column words completed: 99.0 %
Total run time =  0.042833333333333333  minutes
```

In [66]:

```
hold_out_Matrix.head(12)
```

Out[66]:

	words	<s		this	action	on	roberts	and	dr	
0	<s	0.0	0.751252	0.019199	0.000000	0.000000	0.000000	0.004174	0.000000	0.
1		0.0	0.002114	0.043340	0.000000	0.003171	0.000000	0.013742	0.000000	(
2	this	0.0	0.000000	0.003226	0.003226	0.009677	0.000000	0.006452	0.000000	0.
3	action	0.0	0.000000	0.000000	0.000000	0.090909	0.000000	0.090909	0.000000	0.
4	on	0.0	0.000000	0.038835	0.000000	0.000000	0.009709	0.019417	0.000000	0.
5	roberts	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.230769	0.000000	0
6	and	0.0	0.001901	0.009506	0.001901	0.000000	0.000000	0.000000	0.001901	(
7	dr	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.
8	/s>	1.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
9	farradys	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
10	part	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
11	has	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

12 rows × 101 columns

In [67]:
```
hold_out_Matrix['words'] = hold_out_word_list['words']
```

The matrix defines the probability of the column given the row (i.e. P(column_header|row_header)).

Therefore, we add the probability of the column word (as the second word, to each.

In [68]:
```
lambda_1 = 0.5
lambda_2 = 0.5

# Create copy so we dont have to re-calculate original
hold_out_Matrix_2 = hold_out_Matrix.copy()
hold_out_Matrix_2 = hold_out_Matrix_2.dropna()
#print(hold_out_Matrix_2)
# Extract 'words' column
hold_out_Matrix_3 = pd.DataFrame({'words':hold_out_Matrix_2.iloc[:,0]})
#print(hold_out_Matrix_3)

hold_out_Matrix_2 = hold_out_Matrix_2.iloc[:,1:]
print(hold_out_Matrix_2)
# Multiply bigrams by lambda 1
hold_out_Matrix_2 = lambda_1*hold_out_Matrix_2
#print(hold_out_Matrix_2)
for n,column in enumerate(list(hold_out_Matrix_2)):

    column_prob = hold_out_word_list[hold_out_word_list['words']==column]['pr
    column_prob = lambda_2*column_prob

    hold_out_Matrix_3[str(column)] = hold_out_Matrix_2[column] + column_prob

    # Outputs progress of main loop, see:
    clear_output(wait=True)
    print('Proportion of column words completed:', np.round(n/len(list(hold_o

# Sum probabilities of matrix (remove word column from calculation)
total_prob = hold_out_Matrix_3.iloc[:,1:].values.sum()
one_sent_prob = hold_out_Matrix_3.iloc[:1,1:10].values.sum()
```

```
Proportion of column words completed: 99.0 %
```

In [69]:
```
hold_out_Matrix_3.head()
```

Out[69]:

	words	<s		this	action	on	roberts	and	dr	
0	<s	0.0	0.397883	0.016893	0.000259	0.002423	0.000306	0.014462	0.000024	0.000
1		0.0	0.023314	0.028964	0.000259	0.004009	0.000306	0.019246	0.000024	0.063
2	this	0.0	0.022257	0.008906	0.001872	0.007262	0.000306	0.015601	0.000024	0.012
3	action	0.0	0.022257	0.007293	0.000259	0.047878	0.000306	0.057830	0.000024	0.000
4	on	0.0	0.022257	0.026711	0.000259	0.002423	0.005160	0.022084	0.000024	0.024

5 rows × 101 columns

In [70]:
```
print(total_prob)
```

```
5.00489010215505
```

In [71]:
```python
print(one_sent_prob)
```

```
0.4322491340891573
```

Exhaustively applying method to find the optimal Lambda parameters that maximise the total probability of the hold-out subset.

Note because lambda_1 + lambda_2 = 1 then we can define lambda_2 w.r.t. the chosen lambda_1 value.

In [72]:
```python
output_table = pd.DataFrame()

for x in range(0,11):
    lambda_1 = x/10
    lambda_2 = 1-lambda_1

    # Create copy so we dont have to re-calculate original
    hold_out_Matrix_2 = hold_out_Matrix.copy()
    hold_out_Matrix_2 = hold_out_Matrix_2.dropna()
    # Extract 'words' column
    hold_out_Matrix_3 = pd.DataFrame({'words':hold_out_Matrix_2.iloc[:,0]})
    hold_out_Matrix_2 = hold_out_Matrix_2.iloc[:,1:]

    # Multiply bigrams by lambda 1
    hold_out_Matrix_2 = lambda_1*hold_out_Matrix_2

    for n,column in enumerate(list(hold_out_Matrix_2)):
        column_prob = hold_out_word_list[hold_out_word_list['words']==column]
        column_prob = lambda_2*column_prob

        hold_out_Matrix_3[str(column)] = hold_out_Matrix_2[column] + column_p

        # Outputs progress of main loop, see:
        clear_output(wait=True)
        print('Current lambda 1 value:', np.round(lambda_1,2))
        print('Current lambda 2 value:', np.round(lambda_2,2))
        print('Proportion of column words completed:', np.round(n/len(list(ho


    # Sum probabilities of matrix (remove word column from calculation)
    total_prob = hold_out_Matrix_3.iloc[:,1:].values.sum()
    output_table = output_table.append(pd.DataFrame({'lambda_1':lambda_1,
                                                     'lambda_2':lambda_2,
                                                     'total_prob':total_prob}
```

```
Current lambda 1 value: 1.0
Current lambda 2 value: 0.0
Proportion of column words completed: 99.0 %
```

In [73]:
```python
output_table.head(10)
```

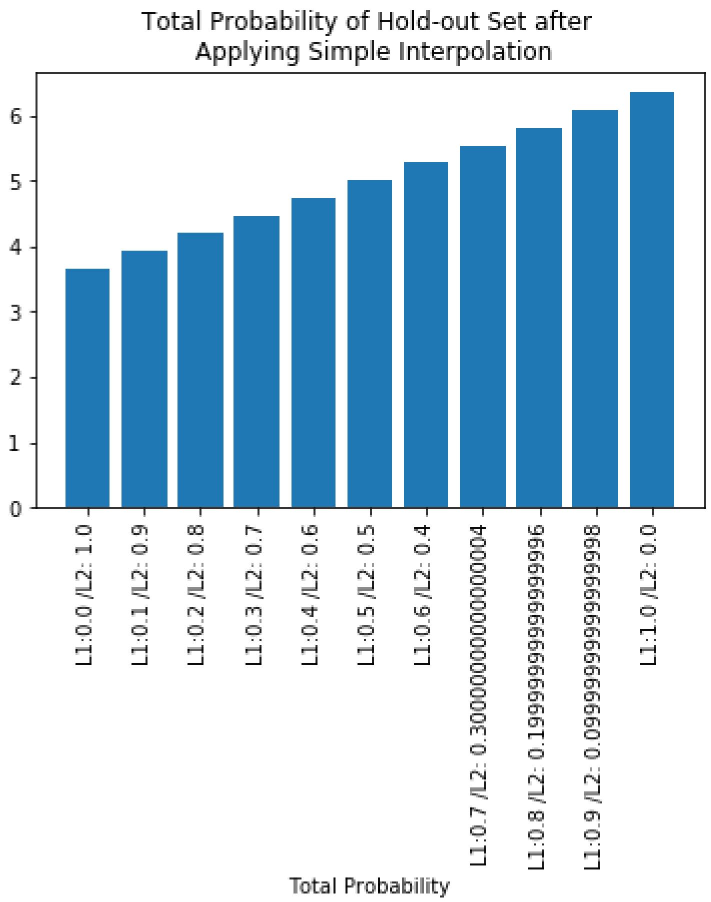
Out[73]:

	lambda_1	lambda_2	total_prob
0	0.0	1.0	3.658009
1	0.1	0.9	3.927385
2	0.2	0.8	4.196761
3	0.3	0.7	4.466138
4	0.4	0.6	4.735514

	lambda_1	lambda_2	total_prob
5	0.5	0.5	5.004890
6	0.6	0.4	5.274266
7	0.7	0.3	5.543643
8	0.8	0.2	5.813019
9	0.9	0.1	6.082395

In [74]:

```
output_table['lambda_1_2'] = 'L1:' + output_table['lambda_1'].astype(str) +'

plt.bar(output_table['lambda_1_2'], output_table['total_prob'])
plt.title("Total Probability of Hold-out Set after \n Applying Simple Interpo
plt.xlabel('Lambda 1 and Lambda 2 Parameters')
plt.xticks(output_table['lambda_1_2'], rotation='vertical')
plt.xlabel('Total Probability')
plt.show()
```



In [75]:

```
optimal_lambda_1 = output_table[output_table['total_prob']==max(output_table[
optimal_lambda_2 = output_table[output_table['total_prob']==max(output_table[

print("Optimal Lambda 1 = ", optimal_lambda_1)
print("Optimal Lambda 2 = ", optimal_lambda_2)
```

```
Optimal Lambda 1 =  1.0
Optimal Lambda 2 =  0.0
```

In [ ]: