Assignment 3 - N Grams

Matthew Dunne

You need to determine which articles (from Assignment Articles.zip) are similar to each other and which books (from Assignment Books.zip) are more similar to each other. In order to accomplish this you need to create n-grams (a.k.a. shingles) and compare the similarity of the text using Jaccard distance.

Import the Necessary Packages and Define the Necessary Functions

In [16]:

```
#import the necessary libraries
from __future__ import division
import nltk
import string
import os
from nltk.corpus import stopwords
from itertools import combinations
import matplotlib.pyplot as plt
import numpy as np
```

In [17]:

```
#define the necessary functions
def ngram_compare_files(file1,file2,n):
    stop = stopwords.words('english')
    f1 = open(file1)
    raw = f1.read()
    f1.close()
    f1_grams = nltk.ngrams(raw.split(),n)
    array_1 = []
    for gram in f1_grams:
        array_1.append(hash(gram))
    f2 = open(file2)
    raw = f2.read()
    f2.close()
    f2_grams = nltk.ngrams(raw.split(),n)
    array_2 = []
    for gram in f2_grams:
        array_2.append(hash(gram))
    intersection = len(list(set(array_1).intersection(array_2)))
    union = len(set(array_1)) + len(set(array_2)) - intersection
    jacard_similarity = intersection / union
    return jacard_similarity


def pairs_of_files(directory):
    dir = os.listdir(directory)
    combo = combinations(dir, 2)
    return combo

def compare_files(directory,ngram_size,threshold):
    compare_dictionary = {}
    ngram = ngram_size
    combo = pairs_of_files(directory)
    for i in combo:
        sim = ngram_compare_files(directory+str(i[0]),directory+str(i[1]),ngram)
        if sim > threshold:
            key = str(i[0]) + "," + str(i[1])
            value = sim
            compare_dictionary[key]=value
    return compare_dictionary
```

Books

Because there are only four books, it could be insightful to think of them in pairs (book x's similarity to book y at N-Gram of z) rather than the aggregate (average similarity of book pairs at N-Gram level of z).

In [18]:

```
dir = 'C://Users//mjdun//Desktop//NLP//Assignments//'
```

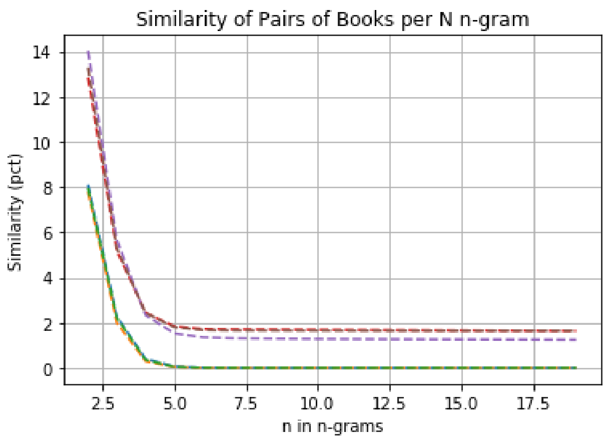
In [19]:

```
#list for number of n grams
x_book = []
#list for mean similarity of pairings
y_book = []
#list for individual pairing similarities
yall_book = []

for n in range(2,20):

    books_comparison = compare_files(dir+'Books//',ngram_size=n,threshold=-1)
    a_book = np.zeros(len(books_comparison))
    counter = 0
    for key, value in books_comparison.items():
        a_book[counter] = value
        counter +=1
    #print (str(n) + ":" + str(a.mean()))
    #print (str(n) + ":" + str(a))
    x_book.append(n)
    yall_book.append(a_book*100)
    y_book.append(a_book.mean()*100)
```

Four books means six distinct pairs of book. When we plot the Jaccard similarity of these pairs against the level of N-Grams we see two distinct clusters.

In [20]:

```
plt.plot(x_book,yall_book, linestyle = '--')
plt.xlabel('n in n-grams')
plt.ylabel('Similarity (pct)')
plt.title('Similarity of Pairs of Books per N n-gram')
plt.grid()
plt.show()
```



If we want to know which books are similar to each other and which are dissimilar, this would be apparent at any level of n for the N-Grams. Past n=4 the similarity percentages settle at about 1.75% and 0%. So we would choose between n=2 or n=3 where the level of similarity has some meaning. **I would choose n=2** because the level of similarity has more meaning. The first cluster has an average similarity of about 13%, which is high, and the difference between the clusters is at its maximum, about 5%.

Per Igor's Grading Notes: For too small n, everything looks similar. For too large n, there might be not enough statistics and it is computationally more expensive. So one needs to select n after the average similarity drops the most. For books, n=3 or 4 is a good choice. For n=2 the average similarity is way too high.

When we look at the name of the books in the pairs (at n=2), this clustering makes sense. **Sherlock Holmes books are similar to one another and Three Men in a Boat is somewhat similar (and to the same degree) to the Sherlock Holmes books.**

In [21]:

```
books_comparison = compare_files(dir+'Books//',ngram_size=2,threshold=-1)
books_comparison
```

Out[21]:

```
{'3boat10.txt,Adventures_of_Sherlock_Holmes.txt': 0.08101757997276196,
 '3boat10.txt,Hound_of_the_Baskervilles.txt': 0.0778690171598505,
 '3boat10.txt,Return_of_Sherlock_Holmes.txt': 0.07948418765591052,
 'Adventures_of_Sherlock_Holmes.txt,Hound_of_the_Baskervilles.txt': 0.12832148097979457,
 'Adventures_of_Sherlock_Holmes.txt,Return_of_Sherlock_Holmes.txt': 0.1402316213494461,
 'Hound_of_the_Baskervilles.txt,Return_of_Sherlock_Holmes.txt': 0.13277111869016417}
```

Articles

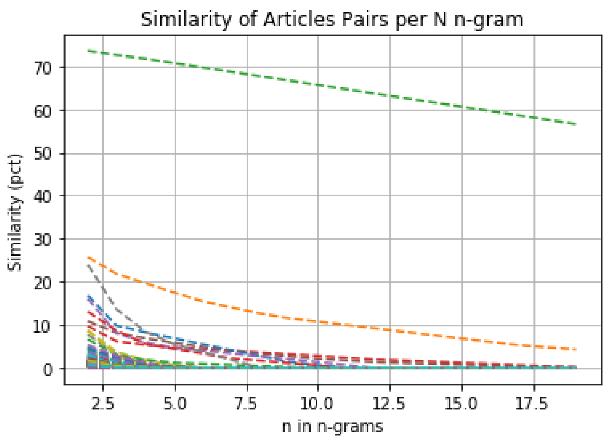
There are more articles than books, and so we do not see the same distinction in similarity clusters.

In [22]:

```
x_art = []
y_art = []
yall_art = []

for n in range(2,20):

    art_comparison = compare_files(dir+'Articles//',ngram_size=n,threshold=-1)
    a_art = np.zeros(len(art_comparison))
    counter = 0
    for key, value in art_comparison.items():
        a_art[counter] = value
        counter +=1
    #print (str(n) + ":" + str(a.mean()))
    x_art.append(n)
    yall_art.append(a_art*100)
    y_art.append(a_art.mean()*100)
```

In [23]:

```
plt.plot(x_art,yall_art, linestyle = '--')
plt.xlabel('n in n-grams')
plt.ylabel('Similarity (pct)')
plt.title('Mean Similarity Across All Articles per N n-gram')
plt.title('Similarity of Articles Pairs per N n-gram')
plt.grid()
plt.show()
```



At *n=2* there is some separation in similarity scores. One pair might be considered essentially the same article, and two other pairs being very similar articles. The rest of the pairs are then on a spectrum of similarity, and it is hard to draw distinctions. However, at *n=4* we see better clustering so **I chose n=4.** There we see the same two pairs as before stand out, but then we see a cluster around 7.5% similarity.
This is different than the n I chose for Books. Unlike the Books we have enough Articles of enough variety that a larger n can actually offer insight.

offer insight.

What do these pairs look like (those above 5% similarity at n=4)?

In [24]:

```
articles_comparison = compare_files(dir+'Articles/',ngram_size=4,threshold=.03)
for k,v in articles_comparison.items():
    art1, art2 = k.split(',')
    similarity_index = v
    print (str(art1) + " is similar to " + str(art2) +
           " with a Similarity Index of " + '{percent:.3%}'.format(percent=similarity_index) +"\n")
```

```
article10.txt is similar to article13.txt with a Similarity Index of 19.624%

article10.txt is similar to article6.txt with a Similarity Index of 5.319%

article14.txt is similar to article6.txt with a Similarity Index of 6.863%

article16.txt is similar to article17.txt with a Similarity Index of 71.717%

article26.txt is similar to article28.txt with a Similarity Index of 8.276%

article26.txt is similar to article31.txt with a Similarity Index of 5.921%

article28.txt is similar to article31.txt with a Similarity Index of 5.556%

article3.txt is similar to article4.txt with a Similarity Index of 8.411%
```

Which articles are similar to one another? We see a few distinct groupings.
-Articles (16 and 17);
-Articles (10 and 13);
-Articles (6, 10, 14);
-Articles (26, 28, and 31);
-Articles (3 and 4).

With all other Articles being indistinct from one another.

A manual review of the relevant groupings reveals several things: (1) shorter and long versions of articles on the same subject are grouped (2) articles of similar size about a similar (but not the same) topic are grouped.