Data Analysis

March 15, 2020

Problem 2: Data Analysis

Throughout this part I will use the library Natural Language Toolkit (nltk), which is a powerful library used in NLP. To start I need to download the nltk's modules that I need.

[ ]:
```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger') #used in excluding the proper nouns
```

[177]:
```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.tag import pos_tag
from matplotlib import pyplot as plt
import numpy as np
import string
```

The class Text_Parser will be used as helper to perform the main processing tasks on each book. For example, at initializaiton the text in the book is parsed into words and sentences.

We define a sentence as the sequence of words that are separated by point "."
In similar manner, a word in a text is defined as a connected sequence of character. By connected, we mean that the characters are not separated by a space.

[204]:
```
class Text_Parser:
    #This class is a helper to parse each book
    def __init__(self,title):
        file_path = "data/{}.txt".format(title)
        file_content = open(file_path).read()
        self.words = nltk.word_tokenize(file_content)
        self.sentences = nltk.sent_tokenize(file_content)

        self.title = title

        self.sentence_len = [len(nltk.word_tokenize(sentence)) for sentence in␣
 ↪self.sentences ]
        self.words_len = [len(word) for word in self.words]

        print("{}: Num of words {}".format(title,len(self.words)))
        print("{}: Num of sentences {}".format(title,len(self.sentences)))
```

```
    def plot_hists(self):
        fig,(ax1,ax2) = plt.subplots(1,2)
        ax1.hist(self.sentence_len)
        ax1.set_title("Histogram of the length of sentences in {}".format(self.
↪title))
        fig.subplots_adjust(left=0,right=2)
        ax2.hist(self.words_len)
        ax2.set_title("Histogram of the length of words in {} /n".format(self.
↪title))
        fig.show()

    def exclude_proper_nouns(self,words):
        #This method allows to remove the proper nouns in any list of words
        tagged_sent = pos_tag(words)
        return [word for word,pos in tagged_sent if pos == 'NNP']

    def get_unique_words(self, exlude_proper_nouns = True):
        #REturn a list of unique words in each book (vocabulary)
        words = list(set(self.words))
        if exlude_proper_nouns:
            return self.exclude_proper_nouns(words)
        return words
```

Let instanciate our parser for each book and see how many words and sentences are there in each of them.

[63]:
```
books = ["Great Expectations",
         "Pride_and_Prejudice",
         "Pygmalion by Bernard Shaw",
         "The Brothers Karamazov by Fyodor Dostoyevsky",
         "The_Adventures_of_Tom_Sawyer",
         "Treasure Island by Robert Louis Stevenson"]


parsed_books = [Text_Parser(title) for title in books]
```

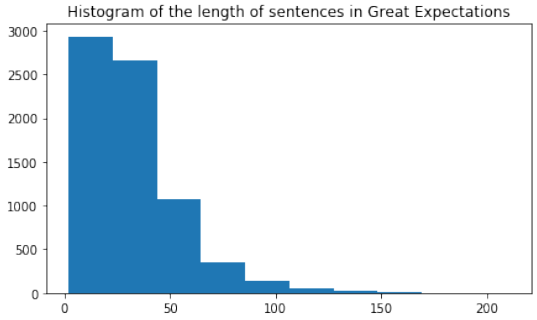
```
Great Expectations: Num of words 228938
Great Expectations: Num of sentences 7245
Pride_and_Prejudice: Num of words 147835
Pride_and_Prejudice: Num of sentences 5973
Pygmalion by Bernard Shaw: Num of words 45287
Pygmalion by Bernard Shaw: Num of sentences 3691
The Brothers Karamazov by Fyodor Dostoyevsky: Num of words 439140
The Brothers Karamazov by Fyodor Dostoyevsky: Num of sentences 19734
The_Adventures_of_Tom_Sawyer: Num of words 75361
The_Adventures_of_Tom_Sawyer: Num of sentences 2616
Treasure Island by Robert Louis Stevenson: Num of words 87655
```

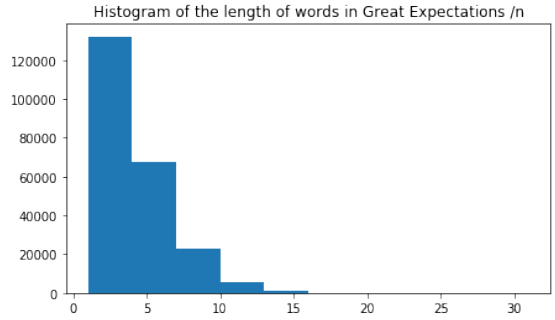
```
Treasure Island by Robert Louis Stevenson: Num of sentences 3848
```

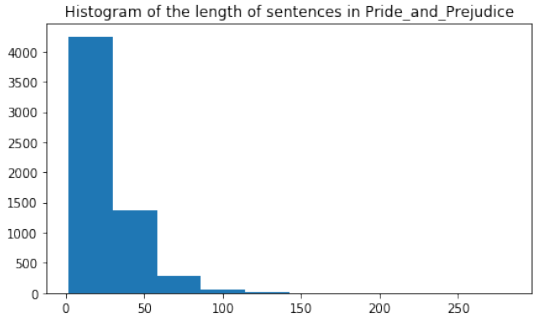
0.1 Part A and B: The histograms of the length of words and sentences used in each text

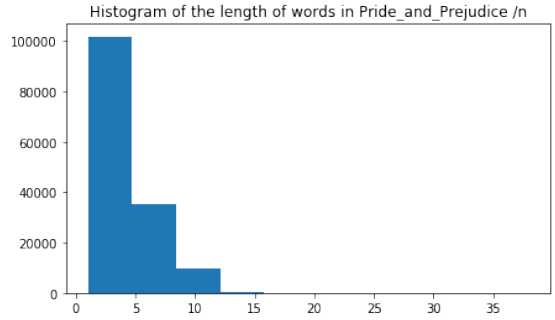
[64]:
```
for book in parsed_books:
    book.plot_hists()
```

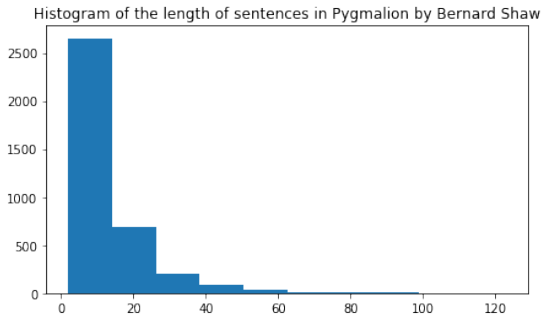
```
/Users/massimacbookpro/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:23: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
```

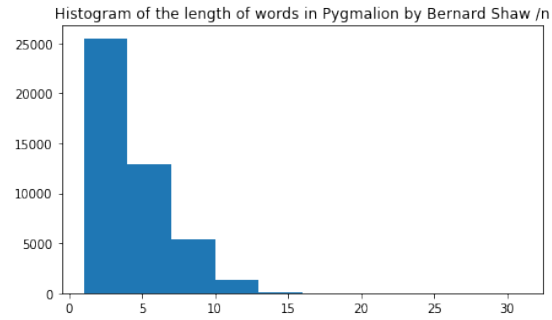


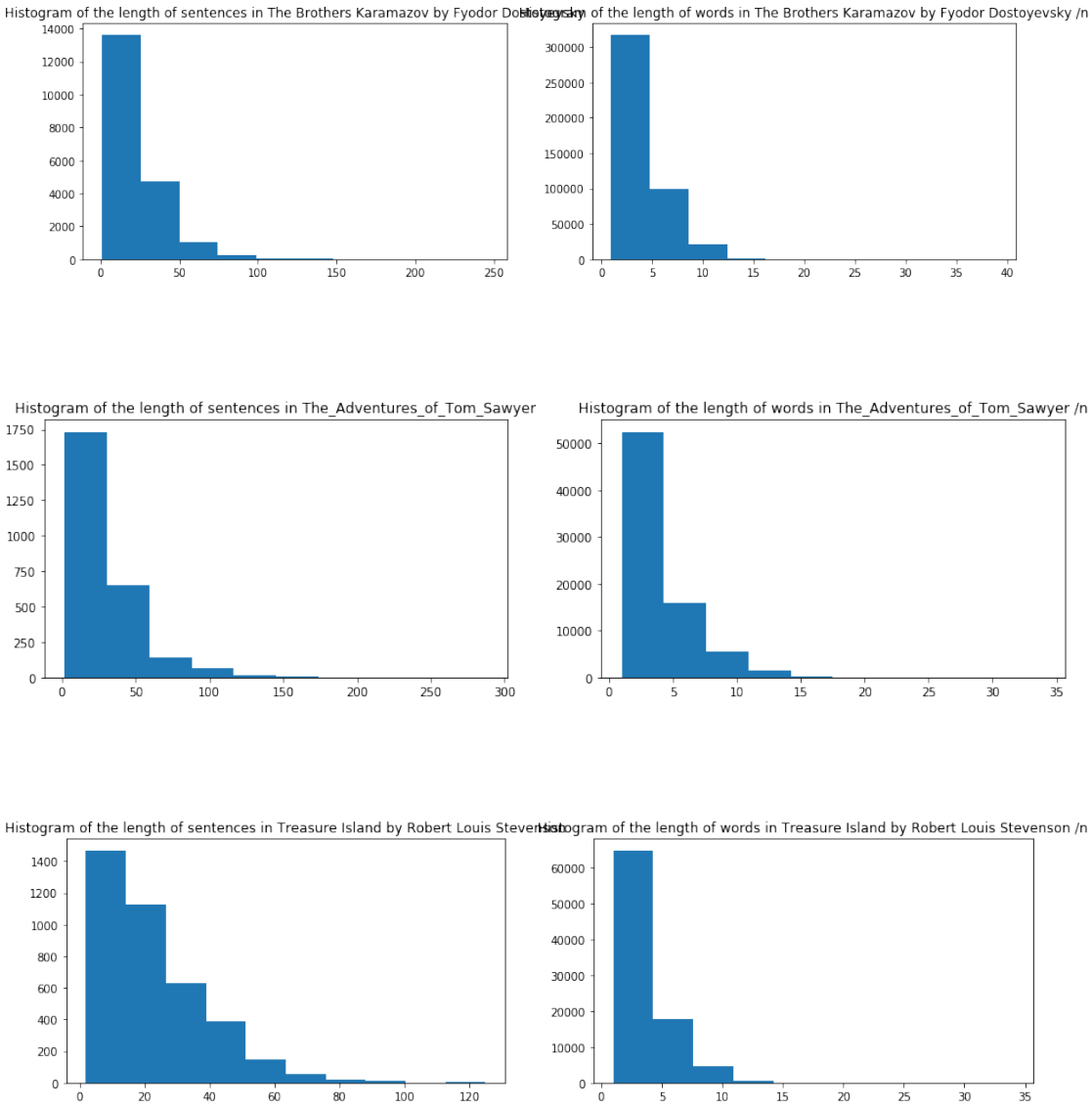












0.2 Part C: The list of unique words used in each book that is not used in any of the other books, with the constraint of excluding proper nouns.

To exclude the proper nouns in the list of words in each book, we use `nltk.tag.pos_tag`, in the method `exclude_proper_nouns(self,words):` in the class `class Text_Parser`, defined above.

`nltk.tag` is an interface for tagging each token in a sentence with supplementary information, such as its part of speech. It uses pre-trained models to classify each token.

[203]:
```
unique_words_bucket = [book.get_unique_words() for book in parsed_books] #get␣
↪the unique words of each book
```

```
#Put everything to lower case

unique_words_bucket = [[word.lower() for word in u_words] for u_words in␣
 ↪unique_words_bucket]
unique_words_compared_to_other_books = []

for idx, unique_words in enumerate(unique_words_bucket):

    #Construct list that contains the unique words of the other books
    u_words_in_otherbooks = sum((u_words for u_words in unique_words_bucket if␣
 ↪u_words != unique_words),[])
    #Now construct exclusive words for each book
    unique_words_compared_to_other_books.append([word for word in unique_words␣
 ↪if word not in u_words_in_otherbooks ])


for idx, unique_words in enumerate(unique_words_bucket):
    print("Book: {}: Num of unique words {}. Unique words compared to other␣
 ↪books -->{}<--".
 ↪format(books[idx],len(unique_words),len(unique_words_compared_to_other_books[idx])))
```

```
Book: Great Expectations: Num of unique words 1542. Unique words compared to
other books -->829<--
Book: Pride_and_Prejudice: Num of unique words 713. Unique words compared to
other books -->273<--
Book: Pygmalion by Bernard Shaw: Num of unique words 747. Unique words compared
to other books -->311<--
Book: The Brothers Karamazov by Fyodor Dostoyevsky: Num of unique words 1920.
Unique words compared to other books -->1235<--
Book: The_Adventures_of_Tom_Sawyer: Num of unique words 1525. Unique words
compared to other books -->853<--
Book: Treasure Island by Robert Louis Stevenson: Num of unique words 772. Unique
words compared to other books -->279<--
```

Comment: The algorithm above, construc first, a list of unique words of each book. Then, each unique word is compared to the other unique words of the other books, and we keep only the exclusive ones, that is the words that are not in the other books.

The complexity of this algorithm is $O(\sum_{i,j\neq j}^{k} n_i n_j) \sim O(kn^2)$, where k is the number of books. The algorithm can be improved by using hash-tables and we can achieve a complexity of $O(kn \log n)$.

0.3 Part D: The longest palindromic sequence in each text

[158]:
```
#This function return the longest palindrom in a string

def get_longest_palindromes(strng):
    N = len(strng)
    cache = [[None] * N for _ in range(N)]

    def is_palindrome(lo, hi):
        if cache[lo][hi] is not None:
            return cache[lo][hi]

        if lo == hi:
            return True
        elif lo + 1 == hi:
            return strng[lo] == strng[hi]

        ans = False if strng[lo] != strng[hi] else is_palindrome(lo+1, hi-1)
        cache[lo][hi] = ans
        return ans

    def generate_palindromes():
        ret = []
        longest = N
        found = False

        if not strng:
            return ['']

        for l in range(N, 0, -1):
            found = False
            for s in range(N-l+1):
                if is_palindrome(s, s+l-1):
                    found = True
                    ret.append(strng[s:s+l])
            if found:
                break
        return ret

    return generate_palindromes()
```

[197]:
```
longest_palindrom = []
#look for the palindorms in each book
for title,book in zip(books,parsed_books):
    print("Parsing {} .... # of sentences {}".format(title,len(parsed_books[0].
 ↪sentences)))
    li = [] #store the longest palindrome of each sentence.
```

```
    table = str.maketrans('', '', string.punctuation+'-')
    stripped = [' '.join(w.translate(table).split()) for w in book.sentences]
    for idx, sentence in enumerate(stripped):

        li.append(get_longest_palindromes(sentence))

    longest_palindrom.append(max(li, key= lambda x: len(x[0])))
```

```
Parsing Great Expectations ... # of sentences 7245
Parsing Pride_and_Prejudice ... # of sentences 7245
Parsing Pygmalion by Bernard Shaw ... # of sentences 7245
Parsing The Brothers Karamazov by Fyodor Dostoyevsky ... # of sentences 7245
Parsing The_Adventures_of_Tom_Sawyer ... # of sentences 7245
Parsing Treasure Island by Robert Louis Stevenson ... # of sentences 7245
```

[201]:
```
for title,palindrom in zip(books,longest_palindrom):

    print("The longest palindrome in: {}, is: -->{}<--".
 ↪format(title,palindrom[0]))
```

```
The longest palindrome in: Great Expectations, is: -->iced a deci<--
The longest palindrome in: Pride_and_Prejudice, is: -->on did no<--
The longest palindrome in: Pygmalion by Bernard Shaw, is: -->aaaaaaaaa<--
The longest palindrome in: The Brothers Karamazov by Fyodor Dostoyevsky, is:
-->ooroooroooroo<--
The longest palindrome in: The_Adventures_of_Tom_Sawyer, is: -->id I di<--
The longest palindrome in: Treasure Island by Robert Louis Stevenson, is: -->
saw was <--
```

Comment To compute the Palindrome, we divied the text into a set of sentences, then we look for Palindromes an each of these sentences. This motivation for this approach is that Palindromes have very high probability to appear with each sentence not within sentences.

The function `get_longest_palindromes`, look for the longest Palindrom in each sentence then.

0.4 Part E: Conclusion

Few conclusion can be made using the results that we got above.

If we look at the histogram plots, one can see that the authors of the books `The Adventures of Tom Sawyer by Mark Twain` and `Great Expectations by Charles Dickensand` tend to write long sentences while the first have a high pattern of writing long words and the former is more like of a short words author.

Looking at the number of unique words in each book, show that the authors `The Brothers Karamazov by Fyodor Dostoyevsky`, `The Adventures of Tom Sawyer by Mark Twain`, and

`Great Expectations by Charles Dickens` tend to have richer vocabulary. The two former authors have quite similar vocabulary length. Now, looking at the unique words in each book compared to the other books, shows that the author of `The Brothers Karamazov by Fyodor Dostoyevsky` uses far more different vocabulary then the other authors.