ds_assignement_aliev

January 28, 2018

In [1]:
```
#For source code (Linked List implementation,
#parsing etc.) scroll to the end of this
#document


#import necessary libraries and running the module
import matplotlib.pyplot as plt
import seaborn as sbn
import pandas as pd
import os

%matplotlib inline
%run parse_documents.py
```

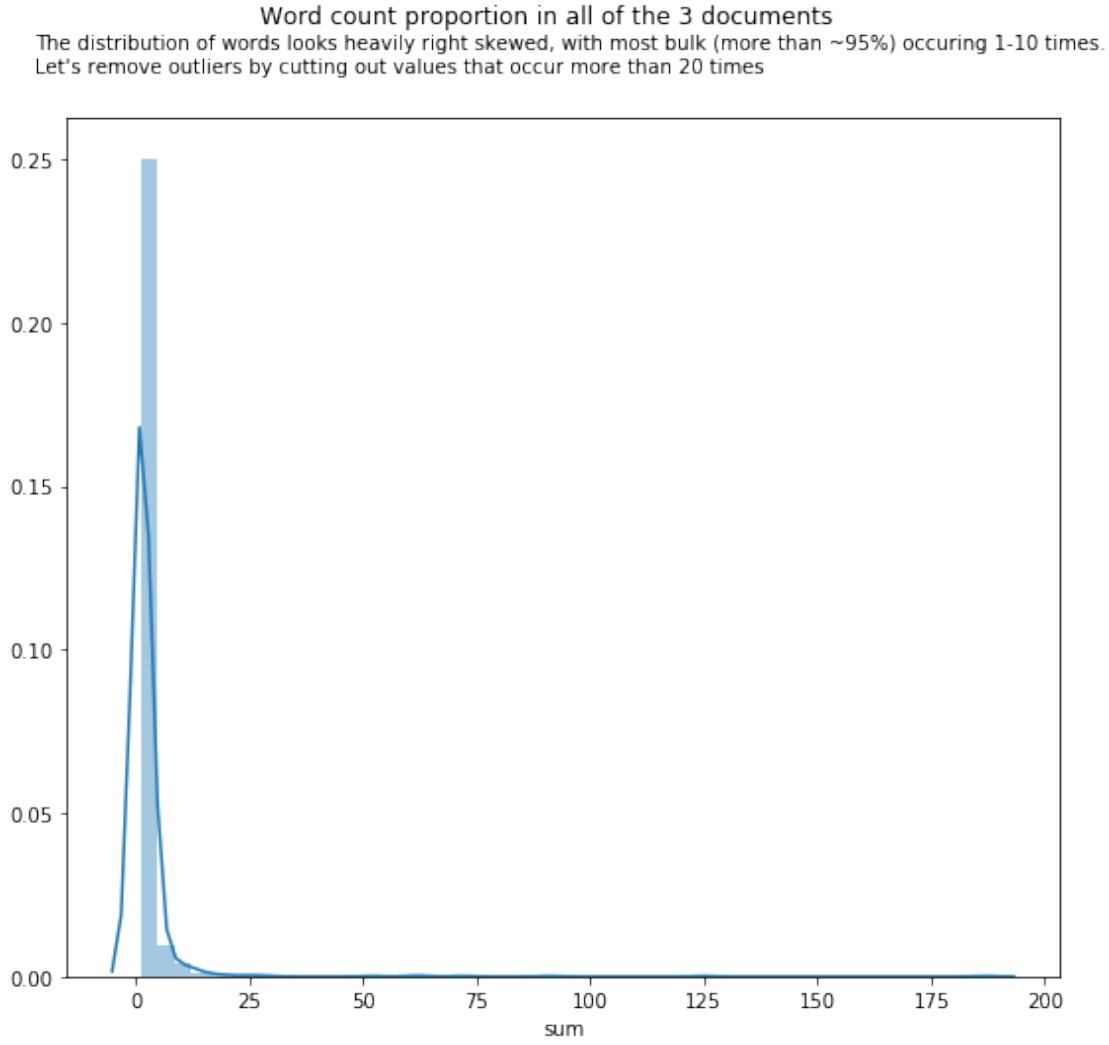
In [2]:
```
#sorting dataframe
df = docs.to_pd_dataframe()
df = df.sort_index()
df = df.sort_values(['sum'], ascending=False)
df.head()
```

Out[2]:

	doc1	doc2	doc3	sum
the	106.0	12.0	69.0	187
of	64.0	6.0	55.0	125
and	35.0	3.0	53.0	91
a	39.0	5.0	28.0	72
in	29.0	3.0	31.0	63

In [3]:
```
#plotting the distribution
a4_dims = (9, 8)
fig, ax = plt.subplots(figsize=a4_dims)
fig.suptitle('Word count proportion in all of the 3 documents')
fig.text(x = 0.1, y = 0.92,
         s = 'The distribution of words looks heavily right skewed, ' +
         'with most bulk (more than ~95%) occuring 1-10 times.\n' +
         "Let's remove outliers "+
         'by cutting out values that occur more than 20 times')
sbn.distplot(df['sum'], ax = ax)
```

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3a6ae7c400>



In [4]:
```
#removing outliers
df2 = df[df['sum'] < 20]
df2 = df2.sort_values(['sum'], ascending=False)
df2.head()
```

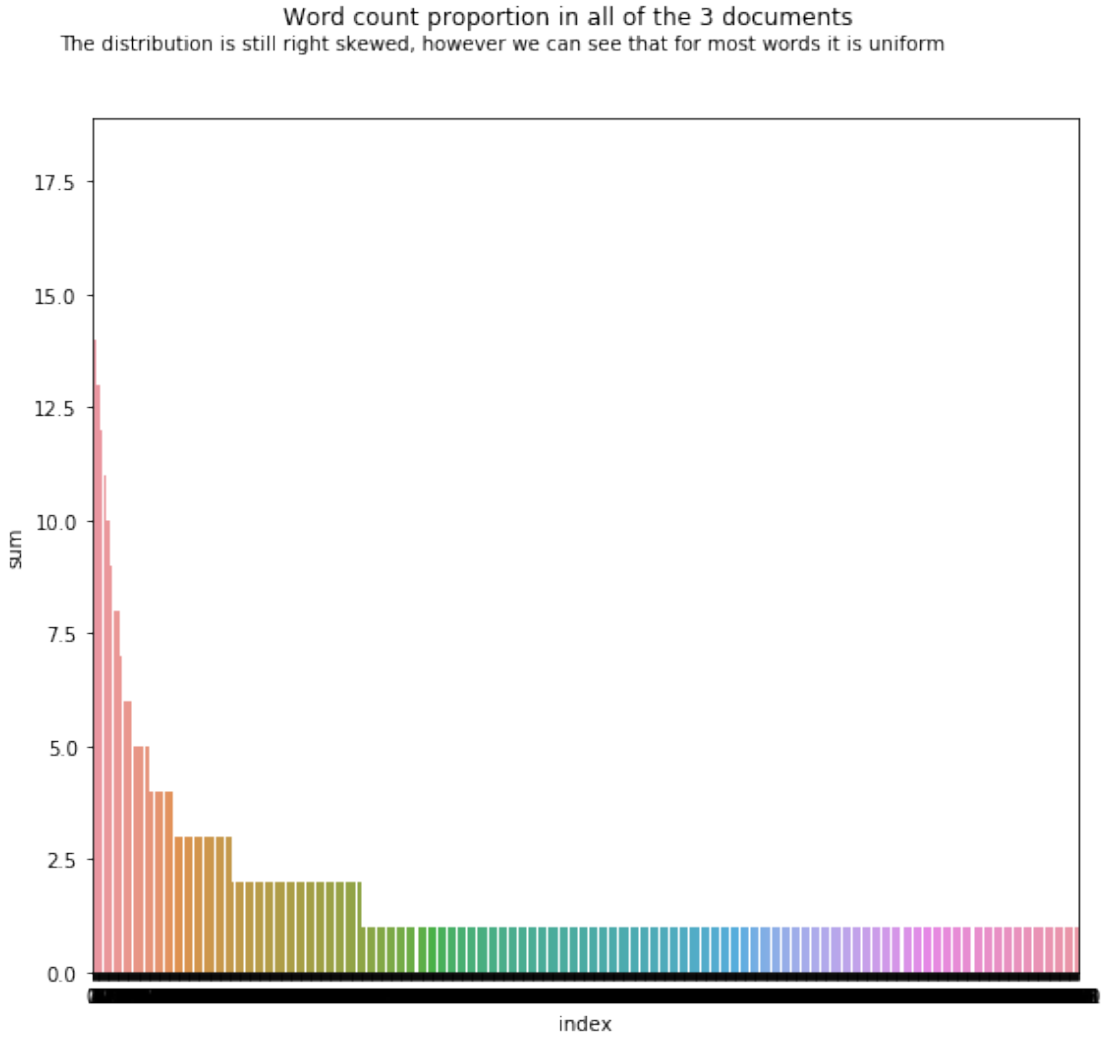
Out[4]:

	doc1	doc2	doc3	sum
it	9.0	NaN	9.0	18
soviet	16.0	NaN	NaN	16
we	NaN	1.0	15.0	16
are	4.0	1.0	11.0	16
book	11.0	3.0	NaN	14

In [5]:
```
#massaging dataframe to plot by words
df_to_plot = pd.DataFrame({'sum': df2['sum'],
```

```
                                    'words': df2.index}).reset_index().drop('index', 1)
df_to_plot['index'] = pd.Series(range(len(df_to_plot.index)),
                                        index=df_to_plot.index)
#creating a plot showing each word frequency
a4_dims = (9, 8)
fig, ax = plt.subplots(figsize=a4_dims)
fig.suptitle('Word count proportion in all of the 3 documents')
fig.text(x = 0.1, y = 0.92, s = 'The distribution is still right skewed, ' +
         'however we can see that for most words it is uniform\n')
sbn.barplot(x = 'index', y = 'sum', data = df_to_plot)
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3a6ae85940>



In [6]:
```
#source code for data structures used for this assignement
!pygmentize -g parse_documents.py
```

```
# regular expressions
import re
# pandas module to convert custom data structure to DataFrame for plotting
import pandas as pd


# Will be used by DocumentLinkedList, each node is word count in one document
class DocumentNode:
    next_node = None
    prev_node = None

    # Constructor
    def __init__(self, word_count, doc_num):
        self.word_count = word_count
        self.document_num = doc_num

    # String representation for print and out
    def __str__(self):
        return [self.word_count, self.document_num].__str__()

    def __repr__(self):
        return [self.word_count, self.document_num].__str__()

    # Check if there is a link next node
    def has_next(self):
        if self.next_node is not None:
            return True
        else:
            return False

    # Check if there is a link previous node
    def has_prev(self):
        if self.prev_node is not None:
            return True
        else:
            return False


# Contains word counts for each document provided by linking DocumentNodes
class DocumentLinkedList:
    length = 0
    head = None
    tail = None

    # Constructor for class
    def __init__(self, node):
        self.head = node
        self.length += 1
```

```
# Makes the class iterable, iterates over each DocumentNode
def __iter__(self):
    current = self.head
    if not current.has_next():
        yield current
    else:
        while True:
            yield current
            current = current.next_node
            if not current.has_next():
                yield current
                break

# String representation for print and out
def __str__(self):
    print_str = ''
    for i in self:
        print_str = print_str + i.__str__()
    return print_str

def __repr__(self):
    print_str = ''
    for i in self:
        print_str = print_str + i.__str__()
    return print_str

# Provide data to len() function on number of DocumentNodes contained in the list
def __len__(self):
    return self.length

# Appends new node to the end of the list
def append(self, node):
    if not self.head.has_next():
        self.head.next_node = node
        node.prev_node = self.head
        self.tail = node
        self.length += 1
    else:
        current = self.head
        while True:
            current = current.next_node
            if not current.has_next():
                current.next_node = node
                current.next_node.prev_node = current
                self.tail = node
                self.length += 1
                break
```

```
    # Removes DocumentNode, document number has to be specified
    def remove(self, doc_num):
        for i in self:
            if i.document_num == doc_num:
                prev_node = i.prev_node
                next_node = i.next_node
                prev_node.next_node = next_node
                next_node.prev_node = prev_node
                del i

    # Returns sum of word counts in all DocumentNodes
    def sum_words(self):
        sum_count = 0
        for i in self:
            sum_count += i.word_count
        return sum_count


# Class containing parsed file
class Documents:
    # Dictionary to hold key -> DocumentLinkedList
    dict_words = dict()
    documents_num = 0

    # Constructor
    def __init__(self, datafile):
        # with open ... statement makes sure that file is closed after
        # constructor is done with it
        with open(datafile) as f:
            words = []
            # compile regex matching end of the document, tags and dates
            end_of_document = re.compile('</doc>')
            tag = re.compile('^<')
            numbers = re.compile('[0-9+]')
            for line in f:
                if not tag.match(line):
                    for word in line.lower().strip().split():
                        # Removing special characters and plural form of a word
                        word = re.sub('[!?:.,"\'();&$/\-]|s$', '', word)
                        # If word is not a number
                        if not numbers.match(word):
                            words.append(word)
                # If end of a document is reached append all the words in to the words_dict
                # variable with DocumentLinkedList, if word already exists update the DLL
                if end_of_document.match(line):
                    self.documents_num += 1
                    for word in set(words):
```

```
                    word_count = words.count(word)
                    if word in self.dict_words:
                        new_node = DocumentNode(word_count, self.documents_num)
                        head_node = self.dict_words[word]
                        head_node.append(new_node)
                    else:
                        new_node = DocumentNode(word_count, self.documents_num)
                        self.dict_words[word] = DocumentLinkedList(new_node)
                words = []
    # Remove '' key
    if '' in self.dict_words.keys():
        del self.dict_words['']

# returns number of documents parsed into the class
def __len__(self):
    return self.documents_num

# String representation for print and out
def __str__(self):
    out = str(self.dict_words)
    return out

def __repr__(self):
    out = str(self.dict_words)
    return out

# Returns word
def get_word(self, word):
    return self.dict_words.get(word)

# Removes word
def del_word(self, word):
    return self.dict_words.pop(word)

# Return number of words
def num_words(self):
    return len(self.dict_words)

# Converts class to pandas DataFrame for further analysis
def to_pd_dataframe(self):
    df_plot = pd.DataFrame()
    for key in self.dict_words:
        word_ll = self.dict_words.get(key)
        counts = []
        doc_cols = []
        for node in word_ll:
            counts.append(node.word_count)
            doc_cols.append('doc' + str(node.document_num))
```

```
            counts.append(word_ll.sum_words())
            doc_cols.append('sum')
            df_to_append = pd.DataFrame([counts], columns=doc_cols, index=[key])
            df_plot = df_plot.append(df_to_append)

        return df_plot


if __name__ == "__main__":
    docs = Documents('txt-for-assignment-data-science.txt')
    test_node = docs.get_word('the')
```