EE 586 Statistical Foundations of Natural Language Processing Assignment 1

Emirhan Koc
emirhan.koc@bilkent.edu.tr

Bilkent University, Electrical and Electronics Engineering

***Abstract*— Seeking meaningful patterns and extracting information in a text or a document is still a hot topic in several areas such as media/advertisement applications, computer science and natural language processing. Yet, texts are mostly in raw and unstructed form for computer to process them so that it retains us from quick attempt of analysis. In this respect, it requires judicious and devoted investigation and process of texts to achieve satisfactory results. In this assignment, we initially preprocessed and tokenized the given raw texts by lowercasing every words, removing special characters, punctuation marks and stop words, and mapping words with like *"isn't"* with apostrophe to *"is not"*. After preprocessing stage, we experimented some subtile properties such as word frequency and rank ratio (can be referred as Zipf's Law), and number of word type and token size ratio. We conducted these experiments on each book of each author/genre seperately as well as on merged version of three books for each author/genre. Lastly, we investigated how we can use these observations to cluster texts with respect to authors and genres. Moreover, we compared our results with literature.**

I. INTRODUCTION

Project Gutenberg is an online library with collection of tremendous amount of books in electronic form [2]. It provides people with an opportunity to search different books of distinct genres ranging from fiction to law, and of many well-known authors such as Charles Dickens and Mark Twain. In this assignment, it is asked to select three authors and three genres in the beginning. Furthermore, it is asked to select three books from each authors and three books from each genres constituting a small corpora for authors and genres. At the core of the assignment, there are two major tasks that require laborious effort. **First** one is to inquiry the relationship between word type frequency and rank of the corresponding word type. This is experimented on each book for authors/genres separately as well as on the merged version of three books for authors/genres.Essentially, it is desired to observe whether results match with **Zipf's Law** or not [4]. **Second** one is to observe the number of word types or vocabulary size ratio with respect to token size; in other words total number of words in the corpus while iterating over the text. The aim of this task is to investigate the vocabulary variations in continuing text and in the whole text, and compare results with the corresponding literature. Eventually, it is asked whether the books can be categorized according to authors/genres based findings of experiments. Following sections are put in order such that *Corpus Construction and Implementation* (II), *Results* (III), *Discussions and Conclusions* (IV). In *Corpus Construction and Implementation* section, qualitative and quantitative information about all corpus and corpora such as such as size, content. In addition, it is explained how the raw text is preprocessed and which tools are used in this stage. In *Results* section, all subsections of the assignment questions are mentioned and results are presented with tables and figures as well as their detailed explainations. In the *Discussion and Conclusion* section, all findings are interpreted, learning throughputs from the assignments is evaluated and a final concise explanation is put forwarded.

II. CORPUS CONSTRUCTION AND IMPLEMENTATION

In this assignment, in total, 18 books are exposed and used to create a corpus to conduct desired experiments. 9 books are selected from three highly esteemed authors: *Charles Dickens, Jane Austen and Mark Twain*. Additionally, 9 books are selected from three different genres such as *horror, fiction and adventure* equally. You can refer to Table 1 and Table 2 below to see what are name of books for each author and genres. It is worth to note that books are selected such that they are at least around 1 MB size. It is important to select this size to obtain still a large size text after preprocessing. In both author and genre distinction, books are written by well-known and highly esteemed authors who have very wide range of vocabulary and also are masterful in usage of English with impressively and stylishly. Moreover, as these books are masterpiece of English and American literature, date of books go back quite a long way, texts are written in old English. This also requires rigorous analysis of suffixes and prefixes in a word.

A. Preprocessing

After downloading books, text contains information about at the top and bottom about Project Gutenber and book chapters. As these parts contain lots of alphanumeric characters and unrelated words with the context of the text, they are eliminated without hesitation. To give machine the ability to understand text and conduct experiments, tokenization is necessary. Prior to attempting tokenizing texts, it is necesssary to remove punctuation marks and special characters from the text. Then, each word is seperated and is put in a list and second form of this list is also created by removing stop words. In addition, words with numeric characters are removed from both of the list.

B. *Implementation*

From beginning to end, Python [6] is used as programming language and Jupyter Notebook [1] is used as an IDE.

Listing 1: Preprocessing and Tokenization

```
def text_preprocess(text):
    text=text.replace("\n","  ")
    for key, value in fix_dictionary.items():
        text= text.replace(key,value)
    for ss in special_characters:
        text= text.replace(ss, "  ")
    return text

def tokenizer(text):

    text_modified=[]
    text= text.split(' ')
    for ii in text:
        if len(ii)>0 and ii.isnumeric()== False:
            text_modified.append(ii)
    return text_modified

#tokenizer for stop word removal
def tokenizer_without_sw(text):

    text_modified=[]
    text= text.split(' ')
    for ii in text:
        if len(ii)>0 and (ii in stop_words)==False
            and ii.isnumeric()==False :
                text_modified.append(ii)
    return text_modified
```

Then code script above shows how the raw data is preprocessed and tokenized using Python [6] *split* and *replace* methods. In text preprocessing function, a raw data is taken and all tab characters are removed without no hesitation. Then, with *replace* method, a mapping is done from with apostrophe to its non- apostrophe form such as " isn't " to " is not ". In addition, all special characters and punctuation marks are removed as well. In tokenizer parts, *split* method is used to seperate each word and add a seperate list. While doing it, all blank spaces and words with numeric characters are removed from selection. After a list of words are obtained in sequential order, calculation of word type-token size ratio, Zipf's Law and visualization of them using Python packages such as Numpy [5] and Matplotlib [3] is performed in a nice looking way.

III. Results

A. *Part a)*

In this part, it is asked to download in total 18 books from Project Gutenberg [2] to create a corpus. Nine of them are taken from three masters of English and American literature as equal number and the rest nine of them are taken from different genres such as horror, fiction and adventure. It can be referred to Table 1 and Table 2 to see them in detailed.

TABLE I: Author Corpora

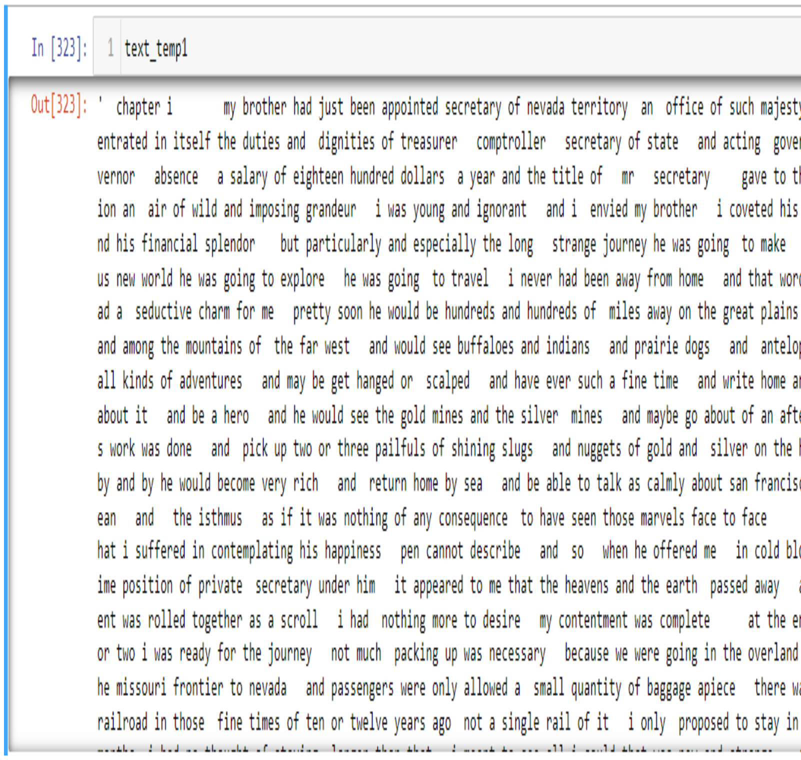
Charles Dickens	Jane Austin	Mark Twain
Hard Times	Sense and Sensibility	The Innocent Abroad
Oliver Twist	A Memoir of Jane Austen	Life on the Mississippi
Dombey and Son	Pride and Prejudice	Roughing It

TABLE II: Genre Corpora

Horror	Fiction	Adventure
The Night Land	The Last Man	Captain Blood
Dracula	Ivanhoe	The People of Mist
The Purloined Letter	Twenty Years After	Mr. Standfast

B. *Part c)*

In this part, the raw text belongs to each book is preprocessed and tokenized according to the implementation procedure explained and code script is attached to clarify algorithm. In the Figure 1, a it can be seen how a raw text looks like just before tokenization but after preprocessing.



```
In [323]:  1 text_temp1

Out[323]: '  chapter i       my brother had just been appointed secretary of nevada territory  an  office of such majesty
          entrated in itself the duties and  dignities of treasurer   comptroller   secretary of state   and acting  gover
          vernor   absence   a salary of eighteen hundred dollars  a year and the title of   mr   secretary    gave to th
          ion an  air of wild and imposing grandeur   i was young and ignorant   and i  envied my brother   i coveted his
          nd his financial splendor    but particularly and especially the long   strange journey he was going  to make
          us new world he was going to explore   he was going  to travel   i never had been away from home   and that wor
          ad a  seductive charm for me   pretty soon he would be hundreds and hundreds of  miles away on the great plains
          and among the mountains of  the far west   and would see buffaloes and indians   and prairie dogs   and  antelo
          all kinds of adventures   and may be get hanged or  scalped   and have ever such a fine time   and write home a
          about it   and be a hero   and he would see the gold mines and the silver  mines   and maybe go about of an aft
          s work was done   and  pick up two or three pailfuls of shining slugs   and nuggets of gold and  silver on the
          by and by he would become very rich   and  return home by sea   and be able to talk as calmly about san francis
          ean   and   the isthmus   as if it was nothing of any consequence  to have seen those marvels face to face
          hat i suffered in contemplating his happiness   pen cannot describe   and  so   when he offered me   in cold bl
          ime position of private  secretary under him   it appeared to me that the heavens and the earth  passed away
          ent was rolled together as a scroll   i had  nothing more to desire   my contentment was complete     at the e
          or two i was ready for the journey   not much  packing up was necessary   because we were going in the overland
          he missouri frontier to nevada   and passengers were only allowed a  small quantity of baggage apiece   there w
          railroad in those  fine times of ten or twelve years ago  not a single rail of it   i only  proposed to stay in
```

Fig. 1: Preprocessed version of *Roughing It*

This preprocessing and thereafter tokenization steps are performed repeatedly for all the 18 books with and without stop word removals. In the following parts, before and after version of removal of stop words is shown as well.

C. *Part e*

In this part of the assignment, it is asked to create a vocabulary that contains only unique words in a corpus (of a single book) and beside to store frequencies of word types; in other words occurrence number of each unique.

Table III and IV shows word types, frequencies and ranks without and with removal of stop words in The Last Man from fiction category.

TABLE III: Without Removal

Word	Frequency	Rank
the	11495	1
of	7152	2
and	6283	3
to	5178	4
i	2952	5
.	.	.
.	.	.
intolerance	1	12347
intimation	1	12348
intimacies	1	12349
intestine	1	12350
intervention	1	12351

TABLE IV: With Removal

Word	Frequency	Rank
she	1329	1
all	641	2
one	497	3
now	475	4
us	451	5
no	403	6
raymond	370	7
.	.	.
.	.	.
interspersed	1	12242
interruptions	1	12243
interrupting	1	12244
interring	1	12245
interpretation	1	12246
interpret	1	12247

D. Part f)

In all of the figures, it is clearly observable that the mimor portion of the words have high frequency whilst the major portion of the words have small frequency. Based on the knowledge obtained from lectures and articles, findings of these experiments are in compliance with *Zipf's Law* that puts forward the inverse proportion between occurence and rank [4] .

E. Part g)

In this part, it is asked to examine the relationship between the token size in the corpus and vocabulary size (number of unique words or types). It requires to keep track of the number of word types with respect to the increasing token size as you traverse along the corpora. It can be performed checking the number of word types for every 5000-10000 tokens. This desired relationship between type and token size is illustrated in both linear and logarithmic scales with 5000 as checkpoint index while iterating over the entire text. In this experiment, all the books of corresponding author is merged and created an *Author Corpora* .

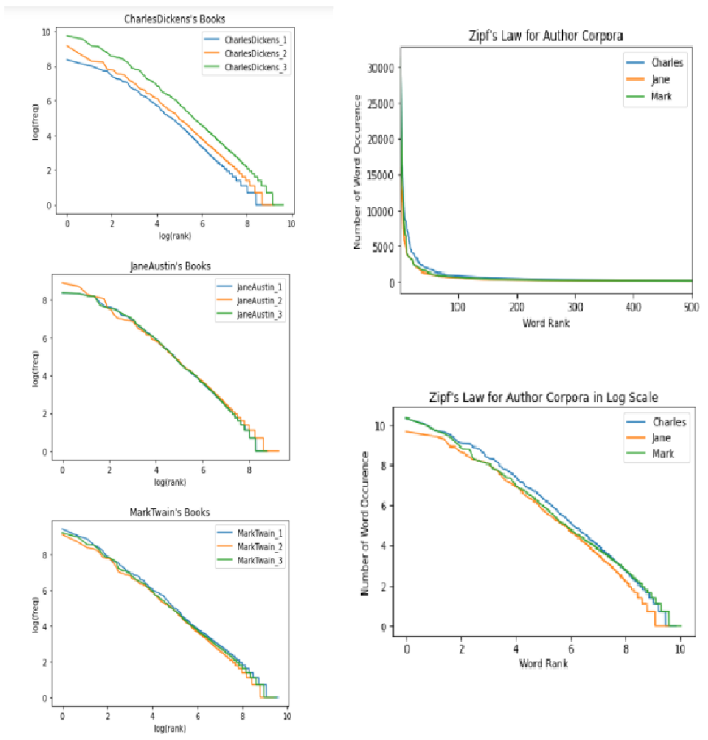


Fig. 2: Word Frequency vs Rank

Left Top-Middle-Bottom: For each book of corresponding author, it is shown what relation exists between word type occurrence and word rank in logarithmic scale.
Right Top–Bottom: For each author, their corresponding books are merged and created an *Author Corpora* and shown what relation exist between word type occurrence and word rank in linear (Top) and logarithmic (Bottom) scale.

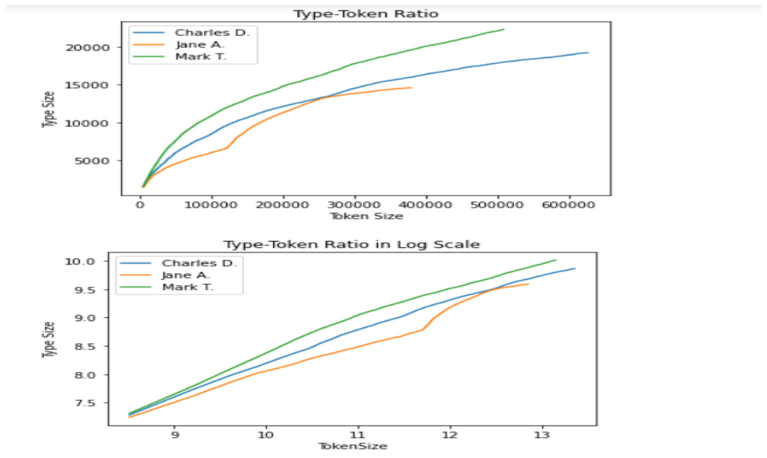


Fig. 3: Type Size vs Token Size

Top-Bottom: Three *Author Corpora* are created by merging corresponding books of each author. Iterating over the the entire text, change of type size is investigated with respect to increasing token size in both linear and logarithmic scales.

F. Part h)

This part is derived from the part g such that instead of merged version of books, each book's type-token ratio is shown seperately.

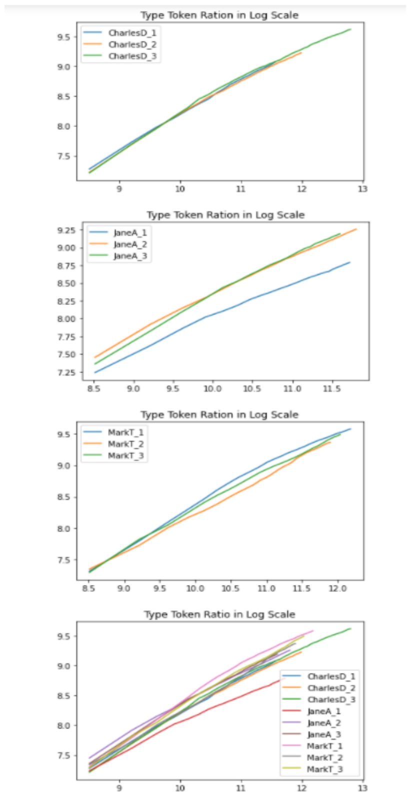


Fig. 4: Type Size vs Token Size

Top-Bottom: Type-token ratio is calculated for each book of each author is investigated and results are illustrated above. Iterating over the the entire text, change of type size is investigated with respect to increasing token size in both linear and logarithmic scales. At the bottom figure, instead of displaying in a seperate figure, all plots are shown in a single figure.

G. Part i)

In this part, it is asked to find the best fitting line and its slope for all the plots corresponding to iterative type-token ratio of each book. In each iteration with checkpoint of 5000, we store a type-token ratio value and it is obvious that this does not give a nice looking line but it can be approximated to a line. In fact, it is basically a first degree linear regression problem. It can be seen in Table V the slopes of lines:

H. Part j)

In this part, same experiments are repeated for the books in horror, fiction and advanture genres. Results are shown in

Book ID	Slope
1	0.5805
2	0.555
3	0.514
4	0.463
5	0.528
6	0.529
7	0.598
8	0.614
9	0.601

TABLE V: Book vs Slope.

Book ID Explanation: In the Table V, *Book ID' s* are represented with numbers and it is necessary to clarify what number means which book. Authors are ordered as *Charles Dickens*, *Jane Austen*, *Mark Twain* and *Book ID* is arranged according to order of books in Table I by following author order.

Book ID	Slope
1	0.439
2	0.541
3	0.667
4	0.531
5	0.527
6	0.507
7	0.558
8	0.527
9	0.594

TABLE VI: Book ID vs Slope

Book ID Explanation: In the Table VI, *Book ID' s* are represented with numbers and it is necessary to clarify what number means which book. Genres are ordered as *Horror*, *Fiction*, *Adventure* and *Book ID* is arranged according to order of books in Table II by following genre order.

Figure 5 and Table VI.

I. Part k)

According to the results obtained in part g, h and i, it is hard cluster books according author or genre and it cannot directly make assumption of clustering based on these finding with thee size of corpus. In my opininon, if the number of corpus belong to each genre or author increases, clustering will become possible. In deed, some patterns of clustering may be considered yet this is not a strong assumption. In addition, all the plots shows Zipfian characteristic and type-token ratio is in reasonable scale.

J. Part l)

You can refer to Appendix to see how removing stop words changes in experiment results. Yet, it is worth to note that as the only fundemental stop word removed basically, it did not bring about significant variations in the results.

K. Part m)

You can refer to Appendix to see how plot changes with random text. But it is worth to note that it is still close to Zipfian characteristic.

IV. Discussion and Conclusion

As the name of the course emphasis *Statistical Foundations* of NLP, we performed several statistical analysis on

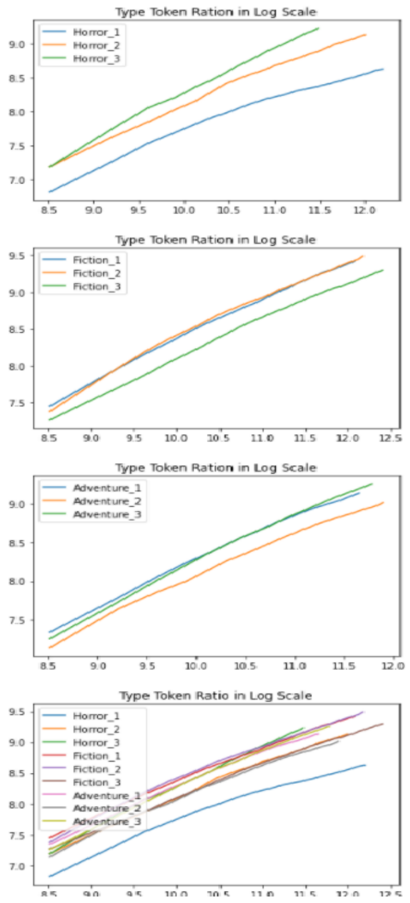


Fig. 5: Type Size vs Token Size

Top-Bottom: Type-token ratio is calculated for each book of each genre is investigated and results are illustrated above. Iterating over the the entire text, change of type size is investigated with respect to increasing token size in both linear and logarithmic scales. At the bottom figure, instead of displaying in a seperate figure, all plots are shown in a single figure.

data based on literature. These are basically Zipf's Law and Type-Token Ratio. We observed how Zipf's Law can be generalized to most of the text from different genre or author. Also, we believe that the type-token ratio can be a metric to investigate how vocabulary change according to genre or author, yet our simple experiments are insufficient to put forward our believe as a concrete hypothesis. However, I assume that the increase in amount of text from a certain genre or author may yield satisfactory results in terms of clustering provided that genres or authors will have some solid distinctions such as culture, climate or region.

References

[1] Project jupyter. Available at `https://jupyter.org/`.

[2] Michael Hart. Project gutenberg. Available at `https://www.gutenberg.org/about/` (1971).

[3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[4] W. Li. Random texts exhibit zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory*, 38(6):1842–1845, 1992.

[5] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. [Online; accessed ¡today¿].

[6] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

EE 586 HW1

March 20, 2022

1 Appendix

[441]:
```
import pandas as  pd
import numpy as np
import os
import matplotlib.pyplot as plt
```

[442]:
```
authors = ["CharlesDickens","JaneAustin","MarkTwain"]
genre = ["Horror","Fiction","Adventure"]

author_text_list = []
for aa in range(0,len(authors)):
    for rr in range(1,len(authors)+1):

     filename = authors[aa]+"_"+str(rr)+".txt"
     author_text_list.append(filename)

genre_text_list =[]
for gg in range(0,len(genre)):
     for rr in range(1,len(genre)+1):
       filename=genre[gg]+"_"+str(rr)+".txt"
       genre_text_list.append(filename)
```

[443]:
```
author_path_list = []
genre_path_list = []
cur_dir = os.getcwd()

for aa in range(0,len(author_text_list)):
    author_path_list.append(os.path.join(cur_dir, author_text_list[aa]))

for gg in range(0,len(genre_text_list)):
    genre_path_list.append(os.path.join(cur_dir, genre_text_list[gg]))
```

[444]:
```
fix_dictionary = {" isn't":" is not ", " isn’t ": "is not", " don't ":" do not␣
 ↪",\
                  "don’t": " do not ","would'nt":" would not","would’t":" would␣
 ↪not ","aren't":"are not",\
                  "aren’t":"are not", "can't":"can not", "can’t": "can not", "␣
 ↪won't" : " will not", " won’t ": " will not " ,\
                  "had'nt":"had not", " hadn’t ": " had not " ,  " 're " : " are␣
 ↪" , " ’re ":" are ",  " 've ": " have " , " ’ve ": "  have ", \
                  "n’t":" not ", " n't ": " not ", "’s":"  ", "'s ": "  ", " ’ll␣
 ↪": " will ", " 'll ": " will" }
```

[445]:
```
special_characters = ["\"", "’" ,"‘", "'", "(", ")",".","," ,"-","_","!","?
 ↪","$","*","“","”",";","{","}","[","]","=","&","#","<",">",\
                     'à', 'æt',␣
 ↪'ætat',"1","2","3","4","5","6","7","8","9","0","*",":","~","£","-",""]
```

[446]:
```
stop_words =␣
 ↪["i","me","my","myself","we","our","ours","ourselves","you","your","yours","yourself","oursel
"how","not","so", "would","could","can","nor","either","him","such"]
```

[447]:
```
def text_preprocess(text):
    text=text.replace("\n","  ")
    for key, value in fix_dictionary.items():
        text= text.replace(key,value)
    for ss in special_characters:
        text= text.replace(ss, "  ")
    return text

def tokenizer(text):

    text_modified=[]
    text= text.split(' ')
    for ii in text:
        if len(ii)>0 and ii.isnumeric()== False:
            text_modified.append(ii)
    return text_modified

#tokenizer for stop word removal
def tokenizer_without_sw(text):

    text_modified=[]
    text= text.split(' ')
    for ii in text:
        if len(ii)>0 and (ii in stop_words)==False and ii.isnumeric()==False :
            text_modified.append(ii)
    return text_modified
```

[417]:
```
#with stop words
author_corpus= []
```

```
genre_corpus = []
import io
for aa in range(0,len(author_path_list)):
    filepath= author_path_list[aa]
    with io.open(filepath,'r',encoding='utf8') as f:
        text = f.read().lower()
        text=text_preprocess(text)
        text_temp1=text
        text=tokenizer(text)
        author_corpus.append(text)

for gg in range(0,len(genre_path_list)):
    filepath= genre_path_list[gg]
    with io.open(filepath,'r',encoding='utf8') as f:
        text = f.read().lower()
        text=text_preprocess(text)
        text_temp2=text
        text=tokenizer(text)
        genre_corpus.append(text)
```

[448]:
```
#with stop words
author_corpus= []
genre_corpus = []
author_path_list = ['C:\\Users\\kocemirhan\\EE 586 HW1\\JaneAustin_2.txt','C:
 ↪\\Users\\kocemirhan\\EE 586 HW1\\Adventure_1.txt']
import io
for aa in range(0,len(author_path_list)):
    filepath= author_path_list[aa]
    with io.open(filepath,'r',encoding='utf8') as f:
        text = f.read().lower()
        text=text_preprocess(text)
        text_temp1=text
        text=tokenizer(text)
        author_corpus.append(text)
```

[ ]:

[418]:
```
# Without stop words
author_corpus_sw= []
genre_corpus_sw = []
import io
for aa in range(0,len(author_path_list)):
    filepath= author_path_list[aa]
    with io.open(filepath,'r',encoding='utf8') as f:
        text = f.read().lower()
        text=text_preprocess(text)
        text=tokenizer_without_sw(text)
        author_corpus_sw.append(text)

for gg in range(0,len(genre_path_list)):
    filepath= genre_path_list[gg]
    with io.open(filepath,'r',encoding='utf8') as f:
        text = f.read().lower()
        text=text_preprocess(text)
        text=tokenizer_without_sw(text)
        genre_corpus_sw.append(text)
```

[419]:
```
# Word-Frequency- Rank for Authors
for ii in range(0,len(author_corpus)):

    aut= np.asarray(author_corpus[ii])
    aut_sw= np.asarray(author_corpus_sw[ii])
    w,c= np.unique(aut,return_counts=True)
    w_sw,c_sw= np.unique(aut_sw,return_counts=True)
    ranking = np.arange(1,len(w)+1)


    info = np.array(list(zip(w,c)))
    df = pd.DataFrame(data=info,columns=["word","frequency"])
    df["frequency"]= df["frequency"].astype('int')
    df= df.sort_values("frequency",ascending=False)
    df["rank"]=ranking
    df.to_csv(author_text_list[ii].replace(".txt",".csv"),index=False)

    ranking = np.arange(1,len(w_sw)+1)
    info_sw = np.array(list(zip(w_sw,c_sw)))
    df_sw= pd.DataFrame(data=info_sw,columns=["word","frequency"])
    df_sw["frequency"]= df_sw["frequency"].astype('int')
    df_sw= df_sw.sort_values("frequency",ascending=False)
    df_sw["rank"]=ranking
    filename= "NoSW_" + author_text_list[ii].replace(".txt",".csv")
    df_sw.to_csv(filename,index=False)

# Word-Frequency-Rank for Genre
for ii in range(0,len(genre_corpus)):

    aut= np.asarray(genre_corpus[ii])
    aut_sw= np.asarray(genre_corpus_sw[ii])
    w,c= np.unique(aut,return_counts=True)
    w_sw,c_sw= np.unique(aut_sw,return_counts=True)
    ranking = np.arange(1,len(w)+1)


    info = np.array(list(zip(w,c)))
    df  = pd.DataFrame(data=info,columns=["word","frequency"])
    df["frequency"]= df["frequency"].astype('int')
    df= df.sort_values("frequency",ascending=False)
    df["rank"]=ranking
    df.to_csv(genre_text_list[ii].replace(".txt",".csv"),index=False)

    ranking = np.arange(1,len(w_sw)+1)
    info_sw = np.array(list(zip(w_sw,c_sw)))
    df_sw= pd.DataFrame(data=info_sw,columns=["word","frequency"])
    df_sw["frequency"]= df_sw["frequency"].astype('int')
    df_sw= df_sw.sort_values("frequency",ascending=False)
    df_sw["rank"]=ranking
    filename= "NoSW_" + genre_text_list[ii].replace(".txt",".csv")
    df_sw.to_csv(filename,index=False)
```

[449]:
```
def PlotZipf(author_books_list,book_type = "Author"):


    for ii in range(0,3):
            ind=3*ii
            merged_book =␣
 ↪author_books_list[ind]+author_books_list[ind+1]+author_books_list[ind+2]
            merged_book= np.asarray(merged_book)
            wr,fr= np.unique(merged_book,return_counts=True)
            word_freq= np.sort(fr)
            word_freq= word_freq[::-1]
            word_rank= np.arange(1,len(wr)+1)

            plt.plot(np.log(word_rank),np.log(word_freq))

            plt.ylabel("Number of Word Occurence")
            plt.xlabel("Word Rank")

    if book_type=="Author":
        legend_list = ["Charles","Jane","Mark"]
    elif book_type=="Genre":
        legend_list= ["Horror","Fiction","Adventure"]
    plt.legend(legend_list)
    plt.title("Zipf's Law for "+ book_type + " Corpora in Log Scale")
    plt.show()
    for ii in range(0,3):

            for dd in range(3*ii,3*(ii+1)):

                boi=author_books_list[dd]
                boi=np.asarray(boi)
                wr,fr= np.unique(boi,return_counts=True)
                word_freq= np.sort(fr)
                word_freq= word_freq[::-1]
                word_rank= np.arange(1,len(wr)+1)
                plt.plot(np.log(word_rank),np.log(word_freq))

                if book_type=="Author":
                     legend_list =␣
↪[authors[ii]+"_1",authors[ii]+"_2",authors[ii]+"_3"]
                elif book_type=="Genre":
                    legend_list = [genre[ii]+"_1",genre[ii]+"_2",genre[ii]+"_3"]
                plt.legend(legend_list)
                plt.xlabel("log(rank)")
                plt.ylabel("log(freq)")
                if book_type=="Author":
                    plt.title(authors[ii]+"'s Books")
                elif book_type=="Genre":
                    plt.title("Book in "+ genre[ii]+ " Type")

            plt.show()
```

2 Zipf's Law for Author Corpora

[450]:

```
PlotZipf(author_corpus,"Author")
```

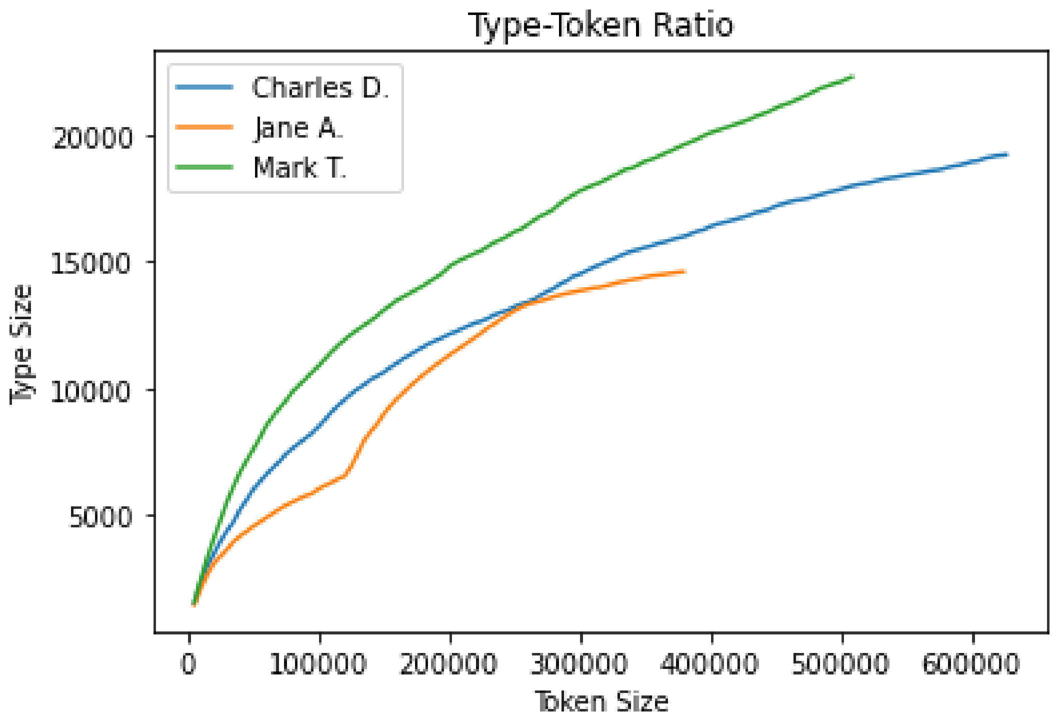
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-450-9fd24ebadf83> in <module>
----> 1 PlotZipf(author_corpus,"Author")

<ipython-input-449-39fdc10004df> in PlotZipf(author_books_list, book_type)
      4     for ii in range(0,3):
      5             ind=3*ii
----> 6             merged_book =␣
 ↪author_books_list[ind]+author_books_list[ind+1]+author_books_list[ind+2]
```

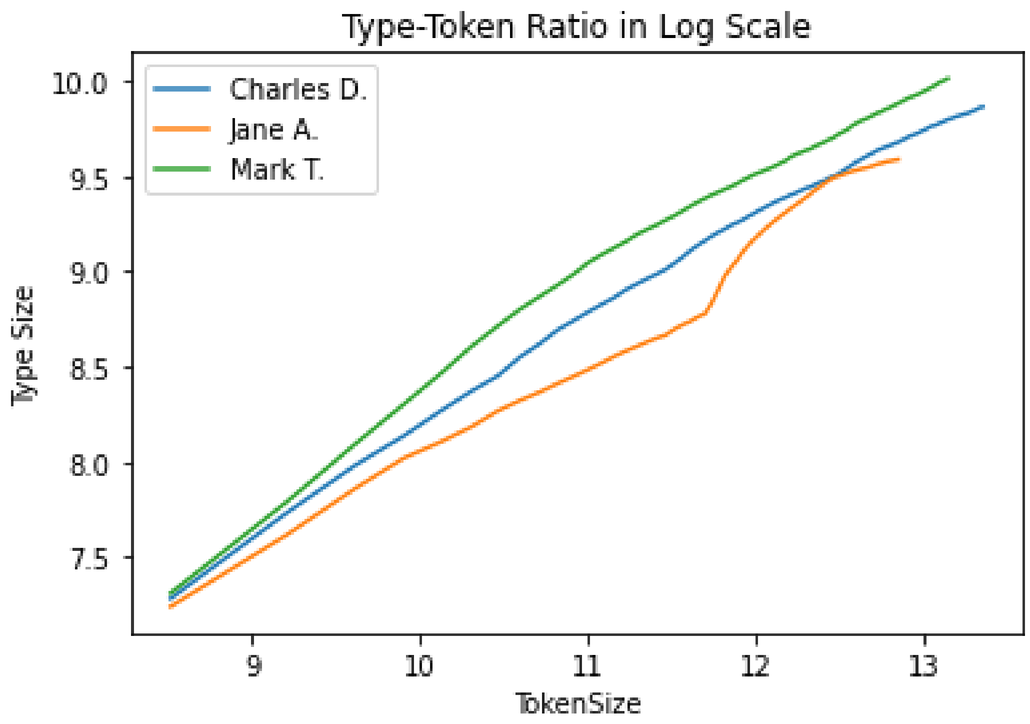
```
      7             merged_book= np.asarray(merged_book)
      8             wr,fr= np.unique(merged_book,return_counts=True)

IndexError: list index out of range
```

[422]:
```
# Type-Token Ratio

def TypeTokenRatio(author_books_list,sub_corpus,book_type=␣
 ↪"author",format_plot="linear"):


        for ii in range(0,3):
            ind=3*ii
            merged_book =␣
 ↪author_books_list[ind]+author_books_list[ind+1]+author_books_list[ind+2]
            merged_book= np.asarray(merged_book)
            corpus_size = len(merged_book)
            iter= corpus_size//sub_corpus

            type_size = []
            token_size= []
            for jj in range(0,iter):
                sliced_merged_book = merged_book[0:sub_corpus*(jj+1)]
                wr,fr= np.unique(sliced_merged_book,return_counts=True)
                type_size.append(len(wr))
                token_size.append((jj+1)*sub_corpus)
            wr,f= np.unique(merged_book,return_counts=True)
            type_size.append(len(wr))
            token_size.append(corpus_size)

            if book_type == "author":
                legend_list = ["Charles D.","Jane A.","Mark T."]
            elif book_type== "genre":
                legend_list = ["Horror","Fiction","Adventure"]


            if format_plot=="linear":
                plt.plot(token_size,type_size)
                plt.legend(legend_list)
                plt.title("Type-Token Ratio")
                plt.xlabel("Token Size")
                plt.ylabel("Type Size")

            elif format_plot=="log":
                plt.plot(np.log(token_size),np.log(type_size))
                plt.legend(legend_list)
                plt.title("Type-Token Ratio in Log Scale")
                plt.xlabel("TokenSize")
                plt.ylabel("Type Size")

        plt.show()
```

3 Type Token Ratio for Author Corpora

[423]:
```
TypeTokenRatio(author_corpus,5000,format_plot="linear")
TypeTokenRatio(author_corpus,5000,format_plot="log")
```

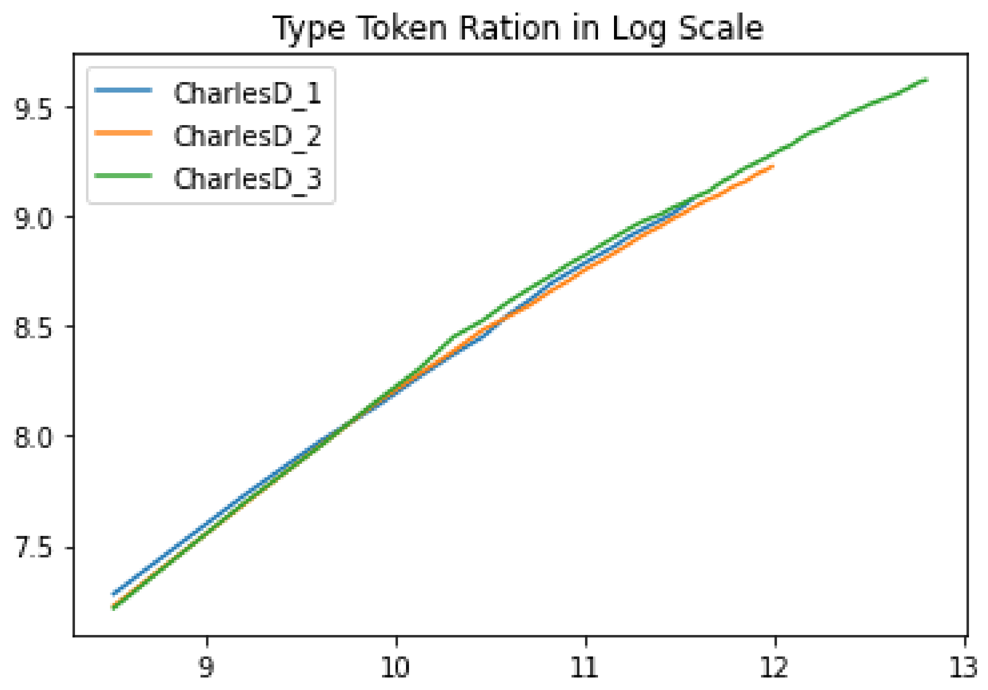


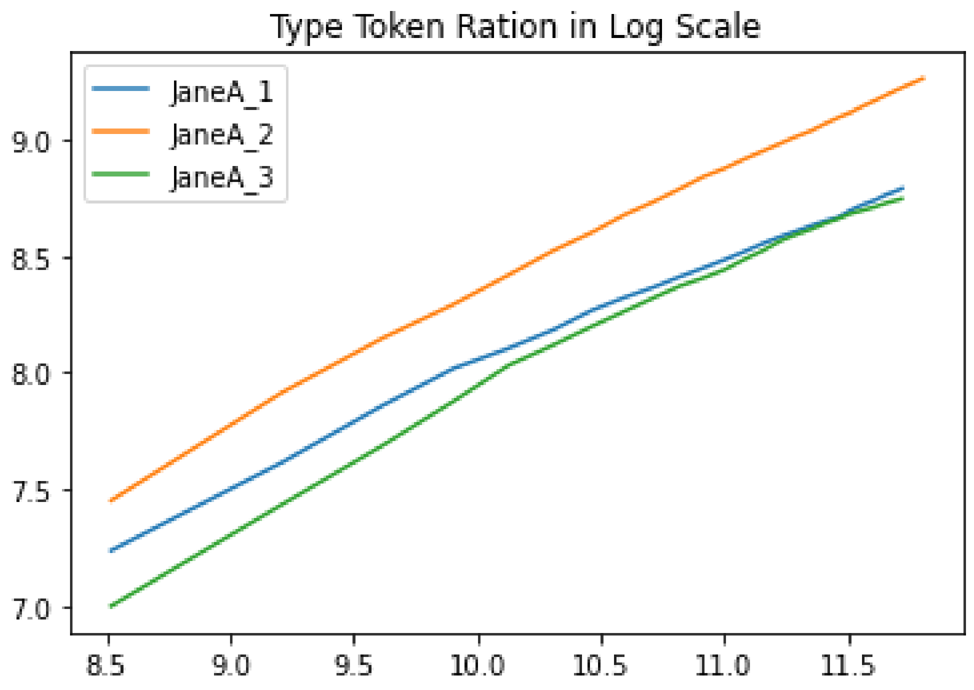


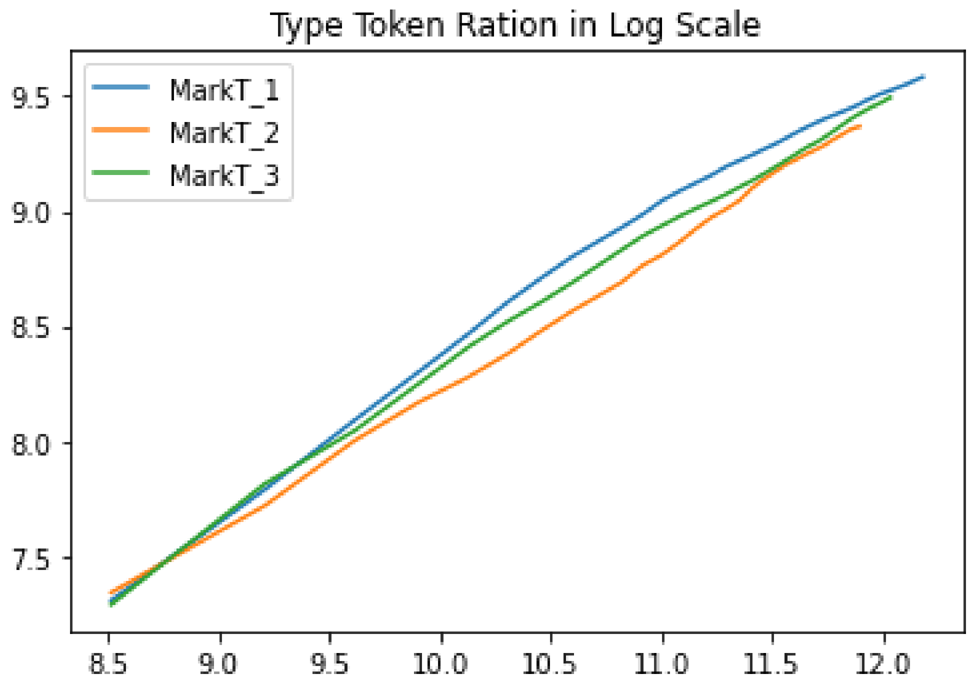
```
[424]: # Type Token  Ratio By Author Book
def TypeTokenRatioBy(author_books_list,sub_corpus,book_type= "author"):

        all_types = []
        all_tokens= []
        for ii in range(0,3):

            for dd in range(3*ii,3*(ii+1)):

                boi=author_books_list[dd]
                boi= np.asarray(boi)
                corpus_size = len(boi)
                iter= corpus_size//sub_corpus

                type_size = []
                token_size= []
                for jj in range(0,iter):
                    sliced_boi = boi[0:sub_corpus*(jj+1)]
                    wr,fr= np.unique(sliced_boi,return_counts=True)
                    type_size.append(len(wr))
                    token_size.append((jj+1)*sub_corpus)
                wr,f= np.unique(boi,return_counts=True)
                type_size.append(len(wr))
                token_size.append(corpus_size)
                all_types.append(type_size)
                all_tokens.append(token_size)


                if book_type == "author":
                  legend_list =␣
↪["CharlesD_1","CharlesD_2","CharlesD_3","JaneA_1","JaneA_2","JaneA_3","MarkT_1","MarkT_2","Ma
                elif book_type== "genre":
                    legend_list =␣
↪["Horror_1","Horror_2","Horror_3","Fiction_1","Fiction_2","Fiction_3","Adventure_1","Adventur


                plt.plot(np.log(token_size),np.log(type_size))
            plt.title("Type Token Ration in Log Scale")
            legends=legend_list[3*ii:3*(ii+1)]
            plt.legend(legends)
            plt.show()

        # Best Fitting Line
        slopes=[]
        for tt in range(0,len(all_tokens)):
            x_ax= np.log(all_tokens[tt])
            y_ax= np.log(all_types[tt])
            theta= np.polyfit(x_ax, y_ax, 1)
            slopes.append(theta[0])




        for ii in range(0,len(all_types)):

            plt.plot(np.log(all_tokens[ii]),np.log(all_types[ii]))
        plt.title("Type Token Ratio in Log Scale")
        plt.legend(legend_list)
    #    arrow_style = {"head_width":0.1, "head_length":0.1, "color":"k"}
     #   plt.arrow(x=10, y=7.5, dx=0.01, dy=0.3, **arrow_style)
      #  plt.text(x=10.1,y=7.95, s=str(slopes[3])[0:6])

       # plt.arrow(x=12.5, y=8.9, dx=0.01, dy=0.45, **arrow_style)
        #plt.text(x=12.6,y=9.2, s=str(slopes[2])[0:6])

        #plt.arrow(x=10.5, y=9.3, dx=0.01, dy=-0.35, **arrow_style)
        #plt.text(x=10.5,y=9.4, s=str(slopes[6])[0:6])

        plt.show()
        return all_types, all_tokens,slopes
```

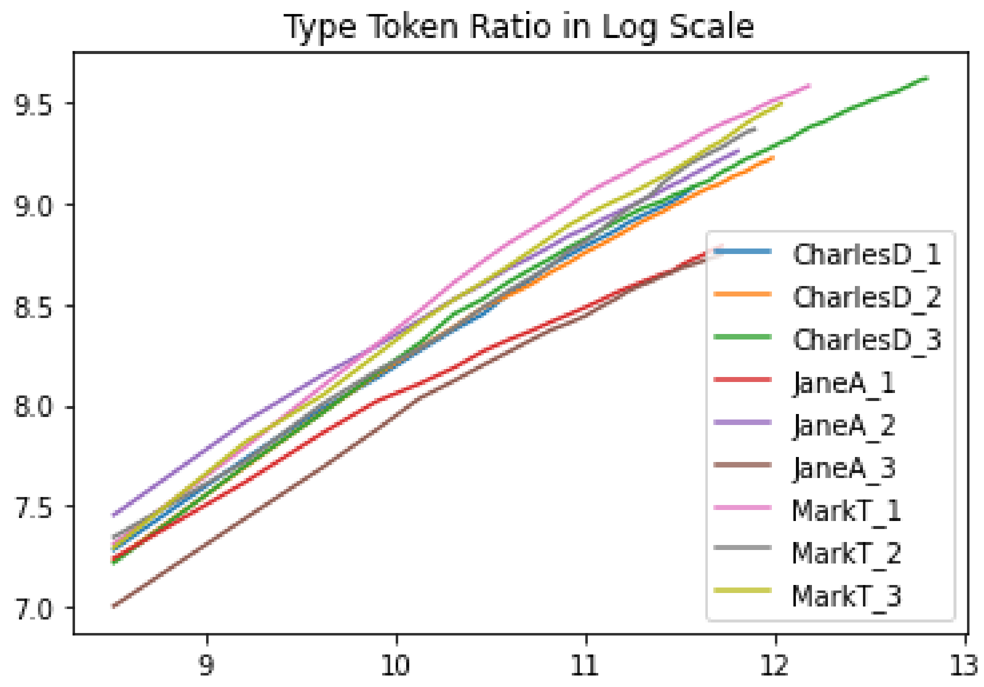
4 Bookwise Type Token Ration for Authors

[425]:
```
# Part i and h
all_types,all_tokens,slopes =␣
 ↪TypeTokenRatioBy(author_corpus,5000,book_type="author")
```




Type Token Ration in Log Scale
JaneA_1
JaneA_2
JaneA_3
9.0
8.5
8.0
7.5
7.0
8.5
9.0
9.5
10.0
10.5
11.0
11.5


Type Token Ration in Log Scale
MarkT_1
MarkT_2
MarkT_3
9.5
9.0
8.5
8.0
7.5
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0

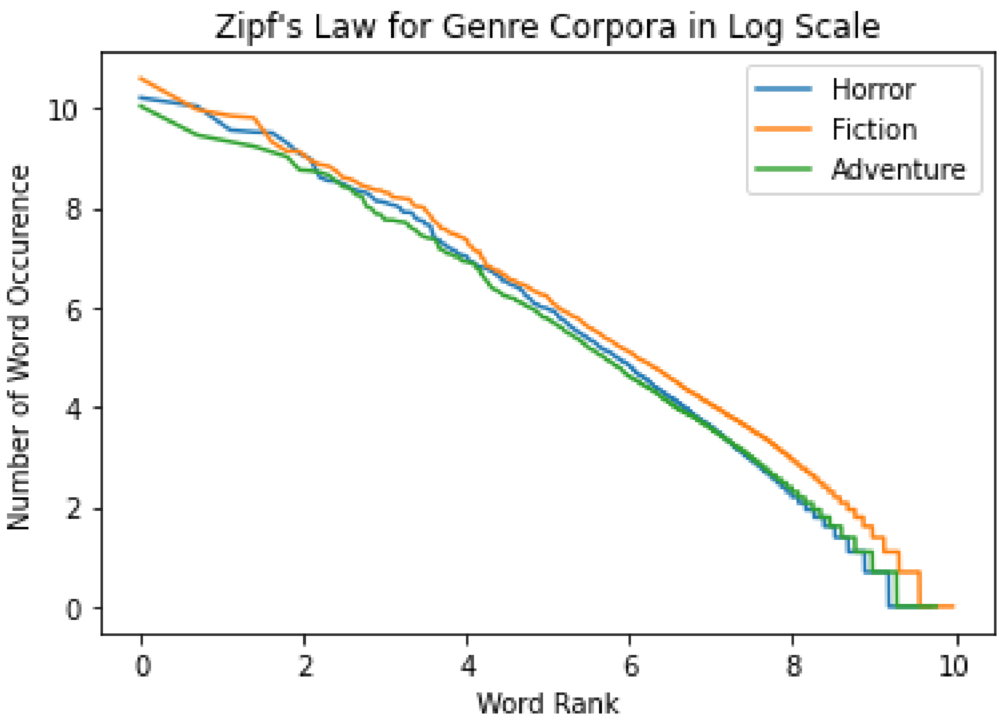


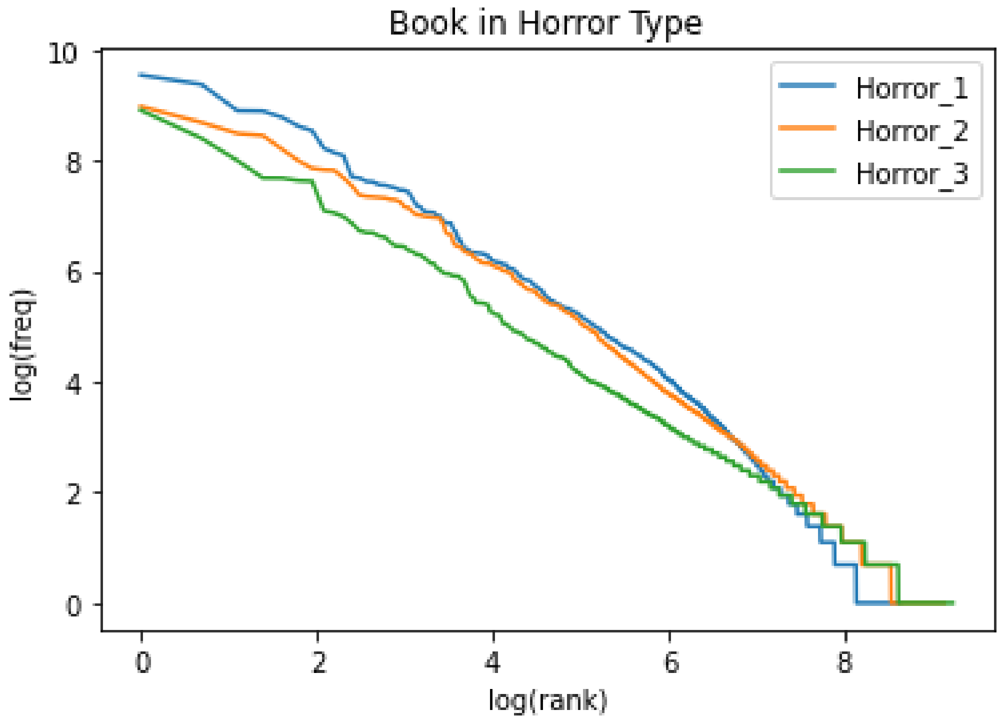
```
[426]: slopes
```

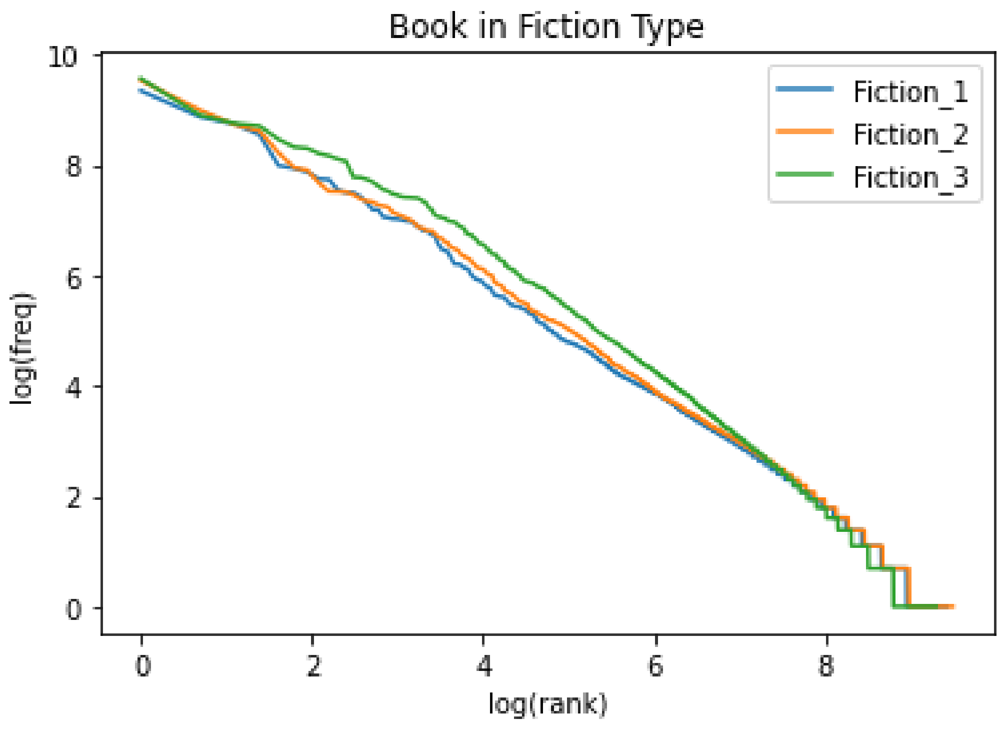
```
[426]: [0.5805864609656438,
        0.5516520958188551,
        0.5144007277709988,
        0.46319578969855235,
        0.5284267393866788,
        0.5294720839951793,
        0.5950883757987248,
        0.614969423395088,
        0.6013583864779135]
```

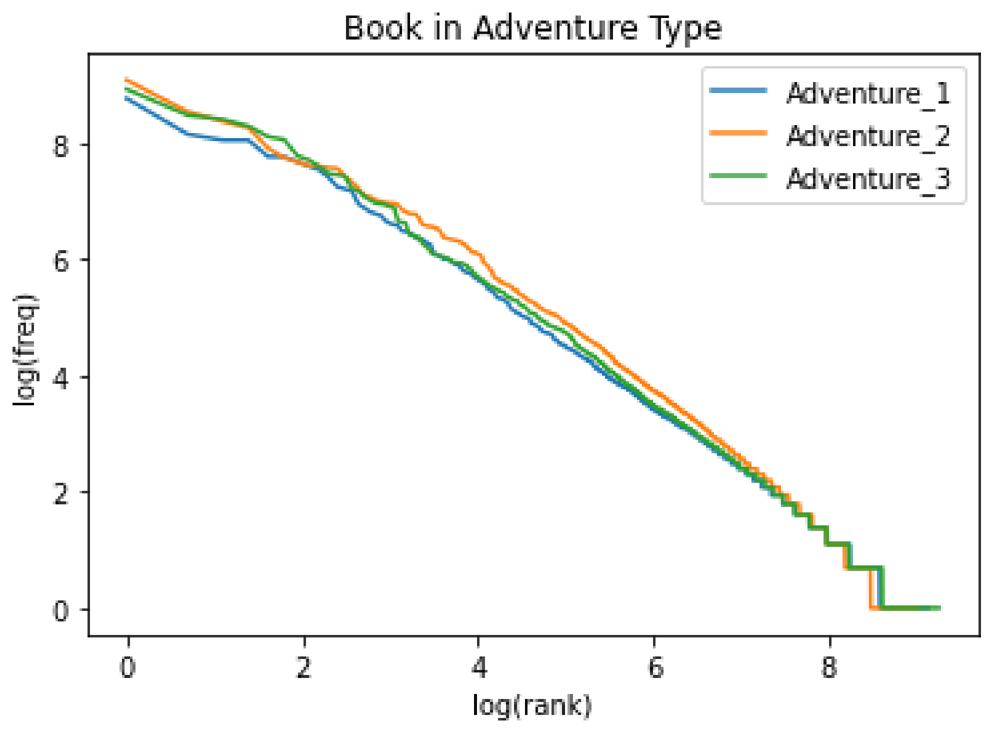
5 Experiments on Genre

```
[427]: # Part f
       PlotZipf(genre_corpus,"Genre")
```

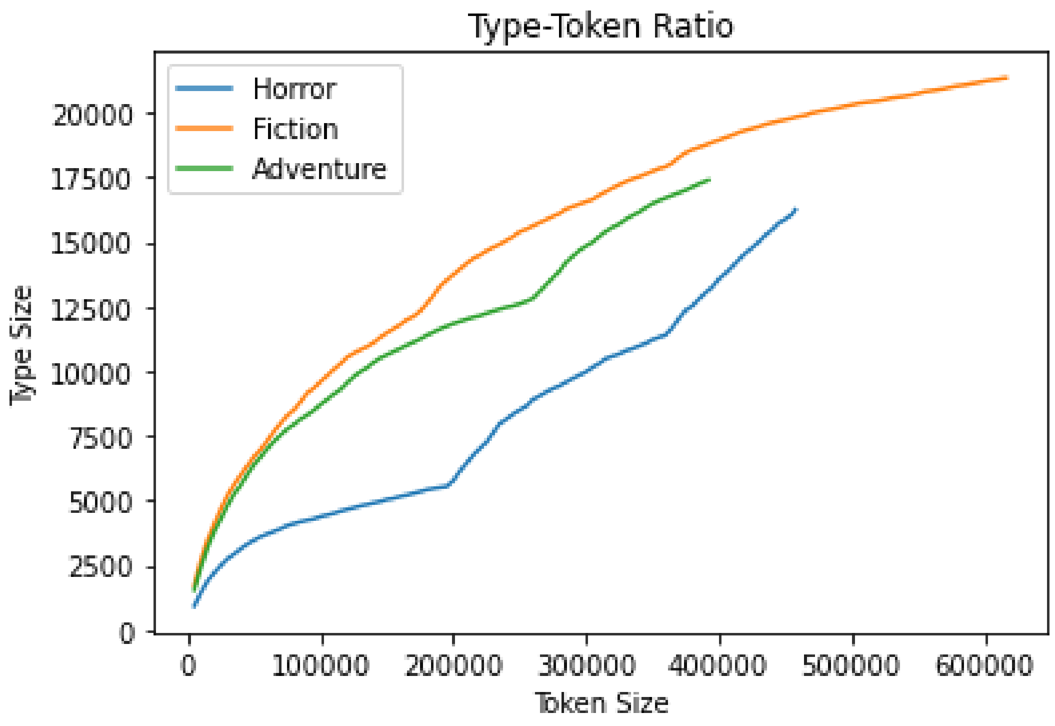

Zipf's Law for Genre Corpora in Log Scale
Number of Word Occurence
Word Rank
Horror
Fiction
Adventure

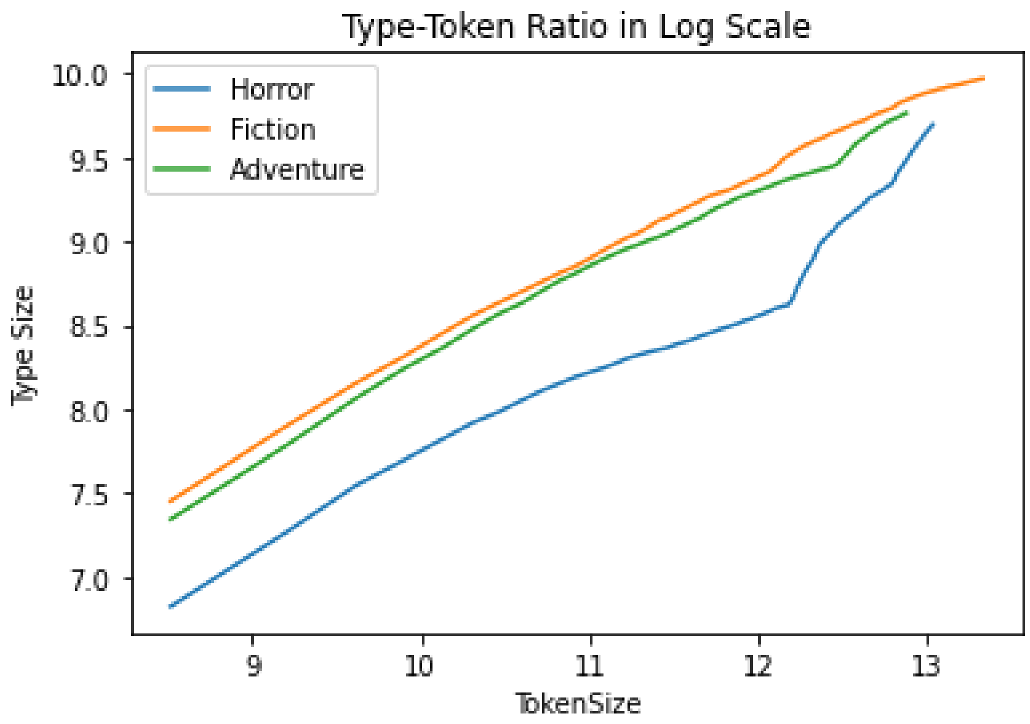

Book in Horror Type
log(freq)
log(rank)
Horror_1
Horror_2
Horror_3


Book in Fiction Type
log(freq)
log(rank)
Fiction_1
Fiction_2
Fiction_3
10
8
6
4
2
0
0
2
4
6
8

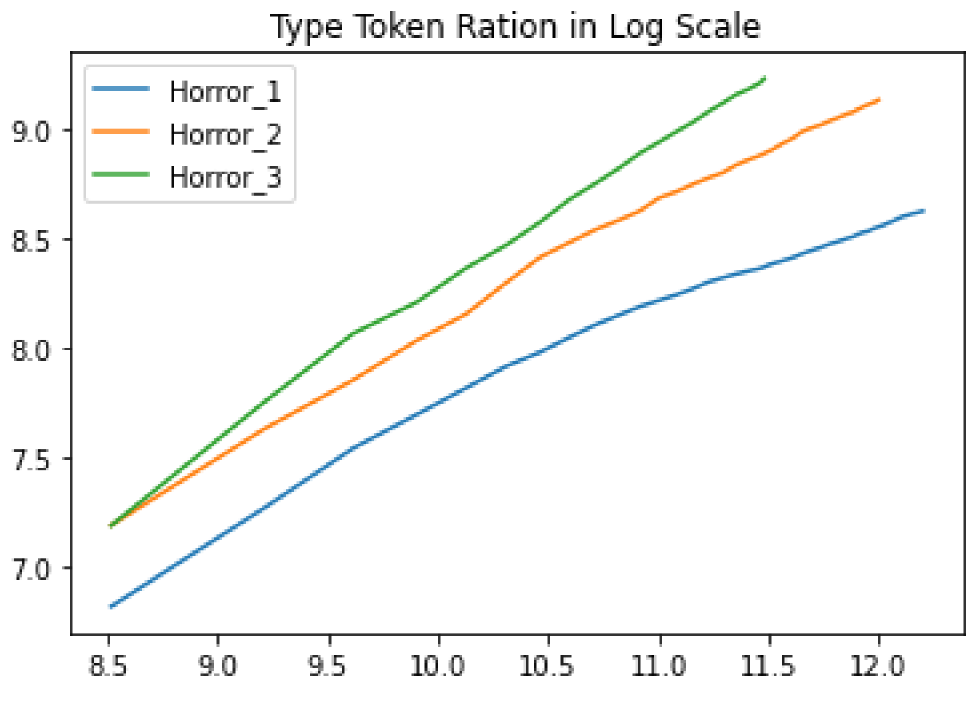

Book in Adventure Type
log(freq)
log(rank)
Adventure_1
Adventure_2
Adventure_3
8
6
4
2
0
0
2
4
6
8

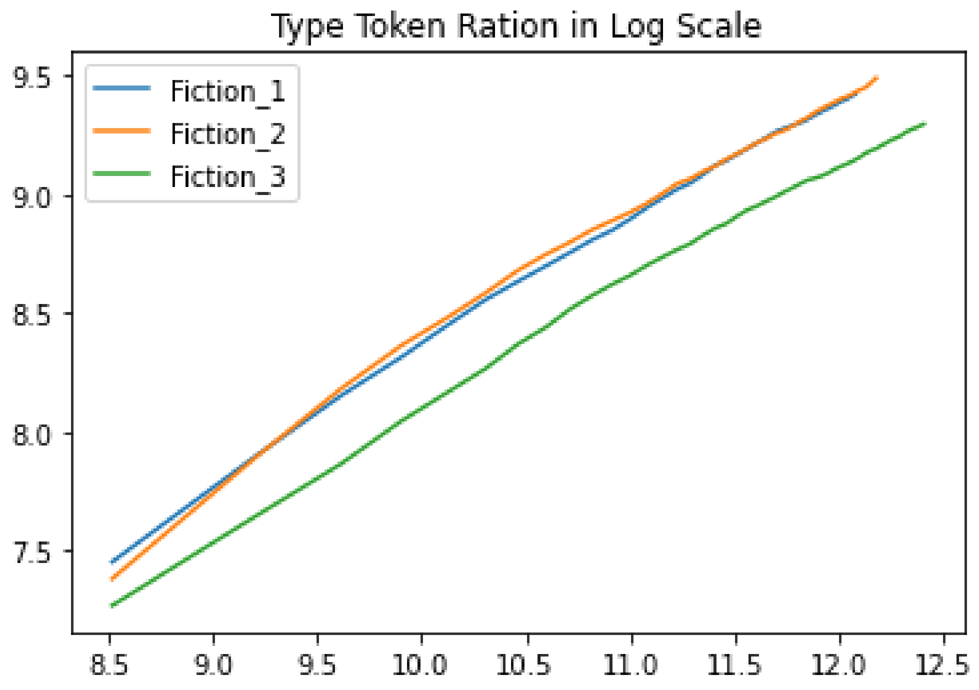
[428]:
```
# Part g

TypeTokenRatio(genre_corpus,5000,book_type="genre", format_plot="linear")
TypeTokenRatio(genre_corpus,5000,book_type="genre",format_plot="log")
```

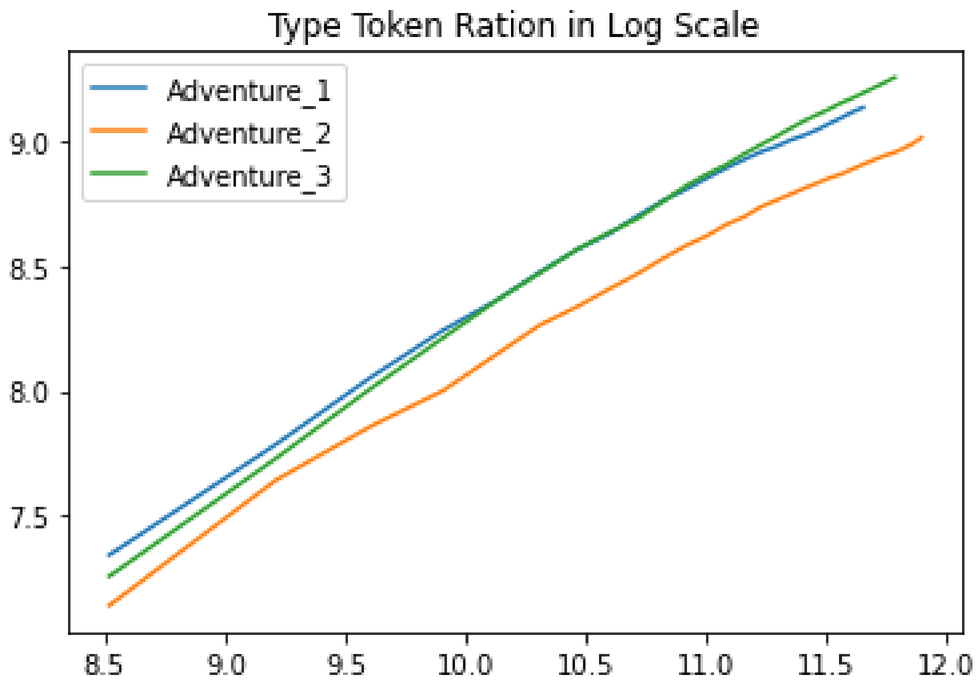


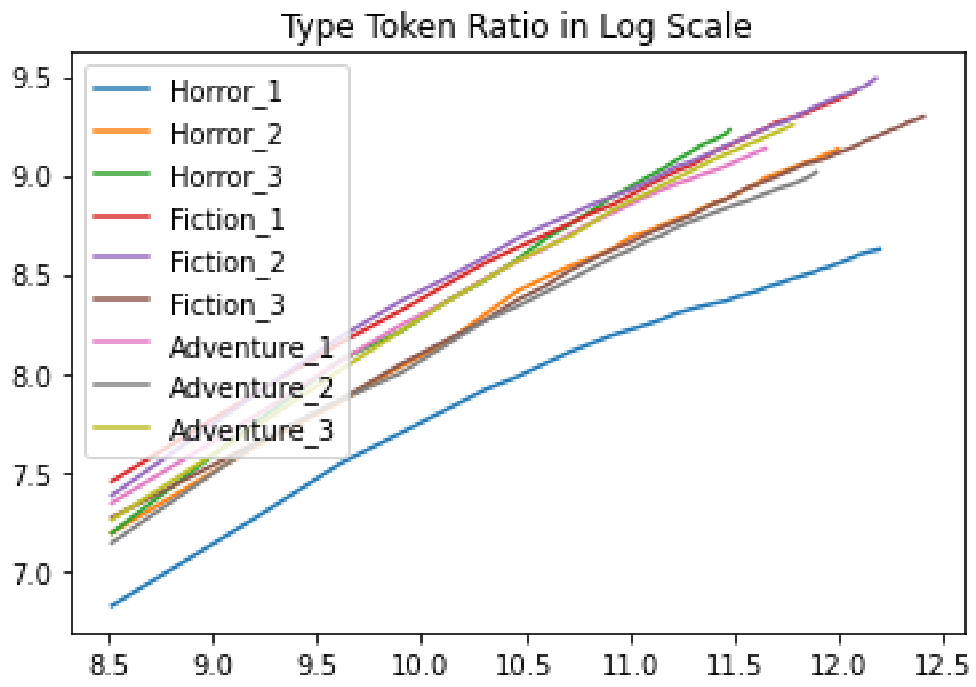


[429]:
```
# Part i and h
all_types,all_tokens,slopes =␣
 ↪TypeTokenRatioBy(genre_corpus,5000,book_type="genre")
```




Type Token Ration in Log Scale
Fiction_1
Fiction_2
Fiction_3
9.5
9.0
8.5
8.0
7.5
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0
12.5


Type Token Ration in Log Scale
Adventure_1
Adventure_2
Adventure_3
9.0
8.5
8.0
7.5
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0



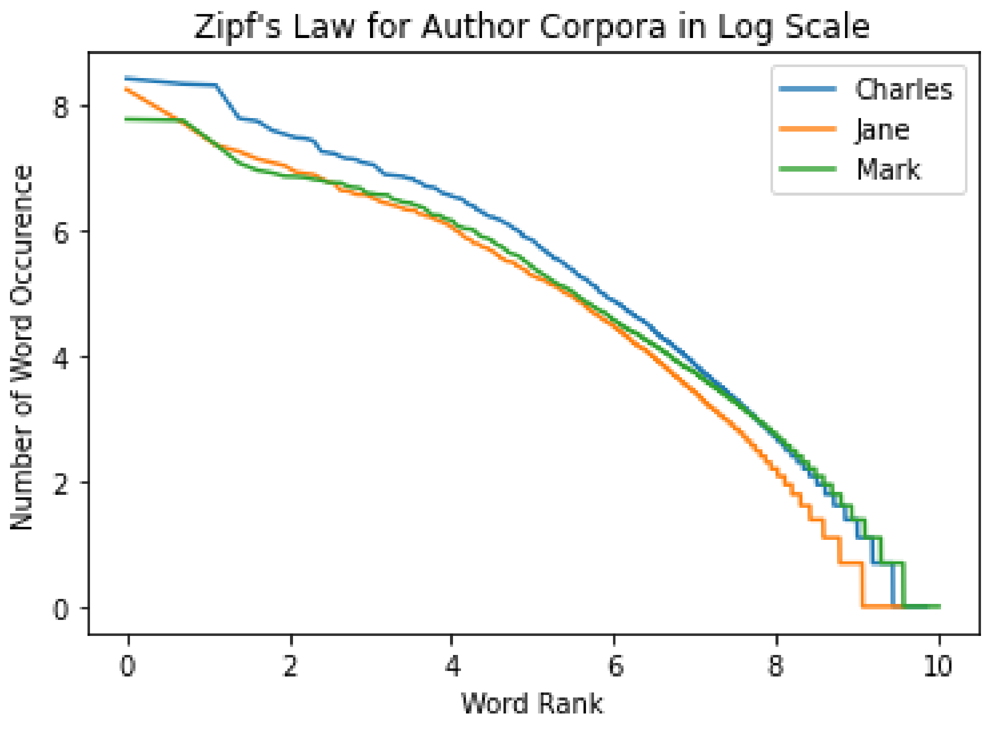
[430]:
```
slopes
```

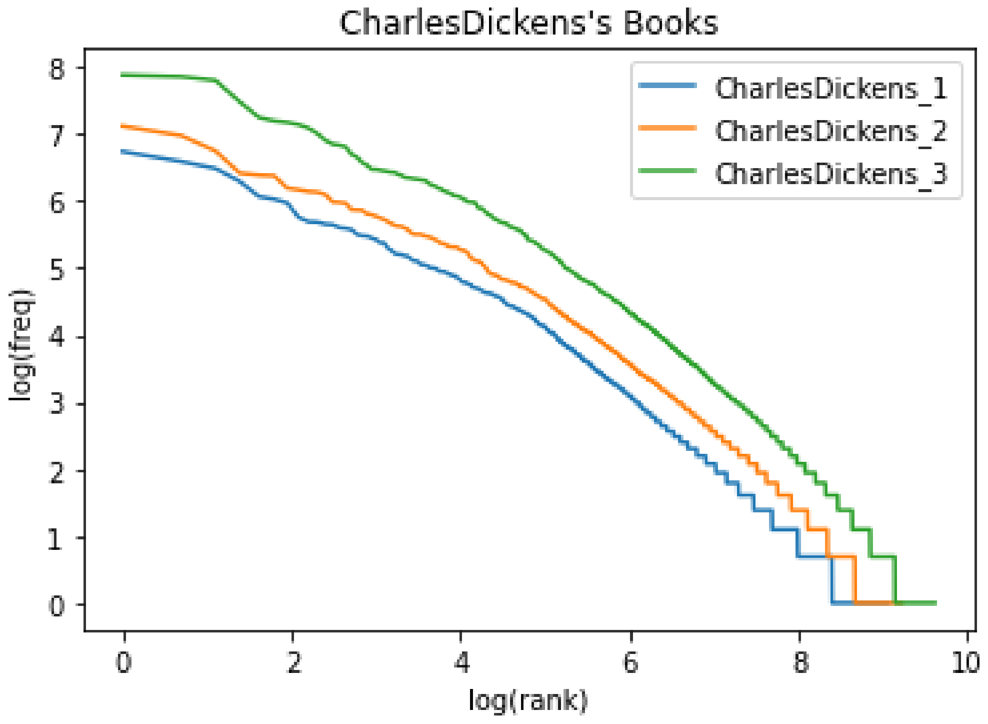
[430]:
```
[0.439991691571903,
 0.5406699291259836,
 0.6678185913539677,
 0.5316780543379558,
 0.5275824640595737,
 0.5077628207327238,
 0.558577187966934,
 0.5271380895940958,
 0.5943907043192368]
```

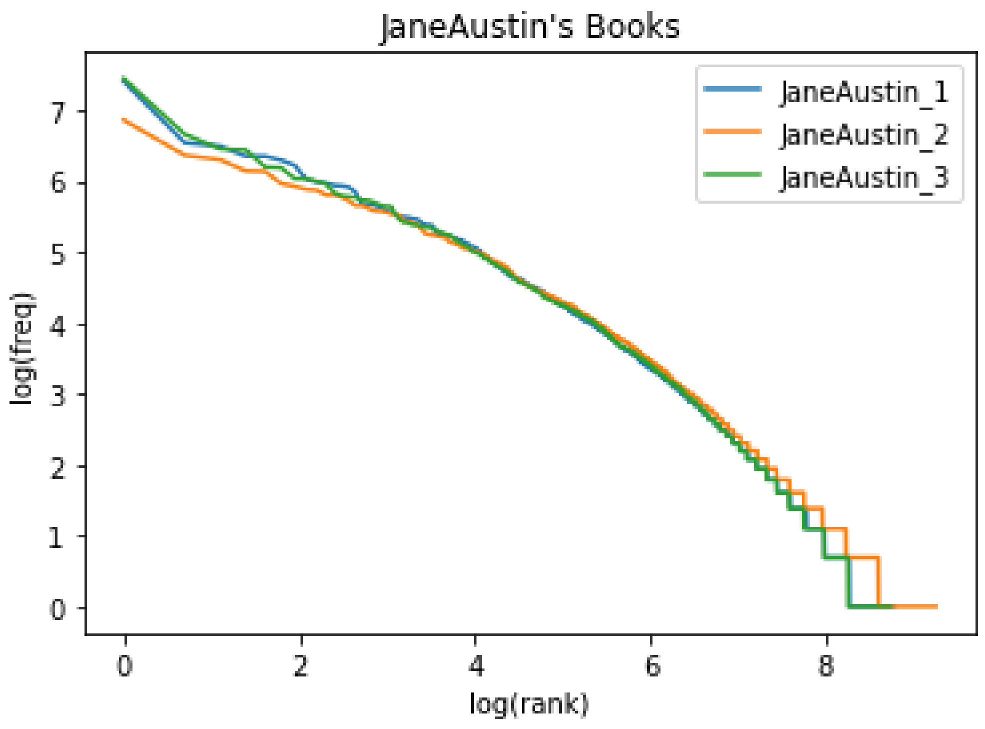
6 **************** After Removal of Stop Words **********************

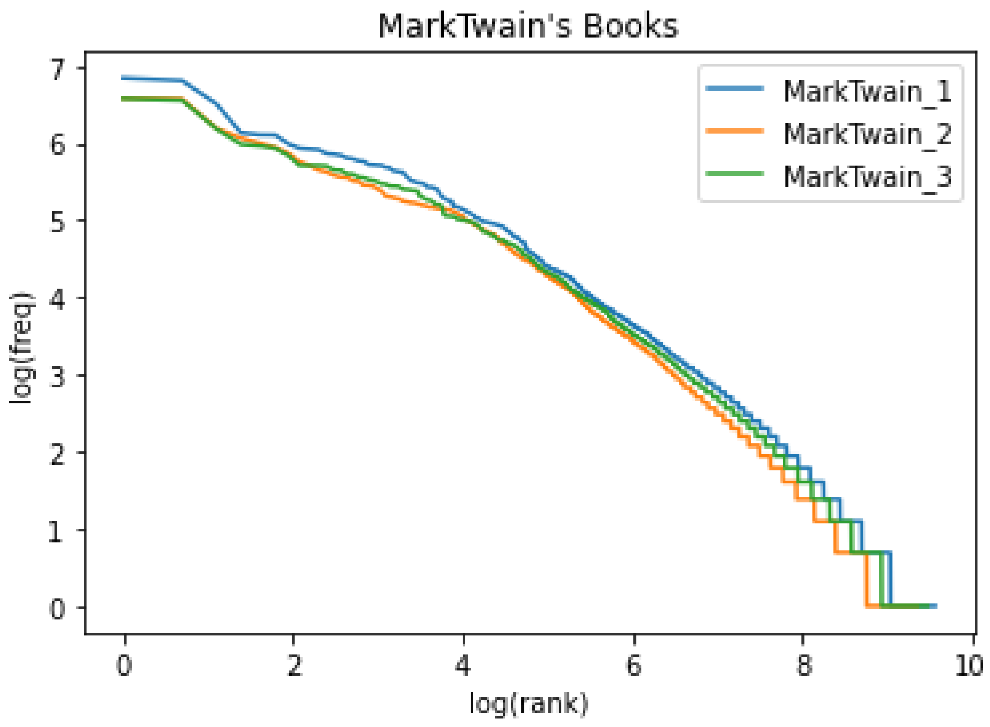
7 Zipf's Law for Author Corpora

[431]:
```
PlotZipf(author_corpus_sw,"Author")
```


Zipf's Law for Author Corpora in Log Scale
Charles
Jane
Mark
Number of Word Occurence
Word Rank
0
2
4
6
8
10

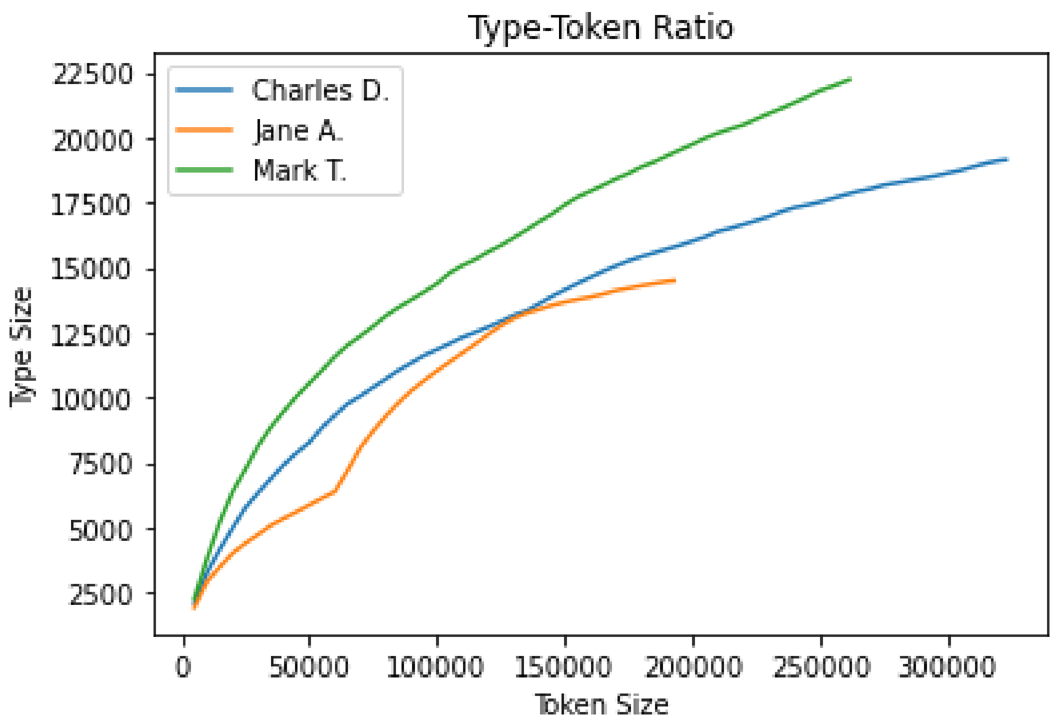

CharlesDickens's Books
CharlesDickens_1
CharlesDickens_2
CharlesDickens_3
log(freq)
log(rank)
0
1
2
3
4
5
6
7
8
10

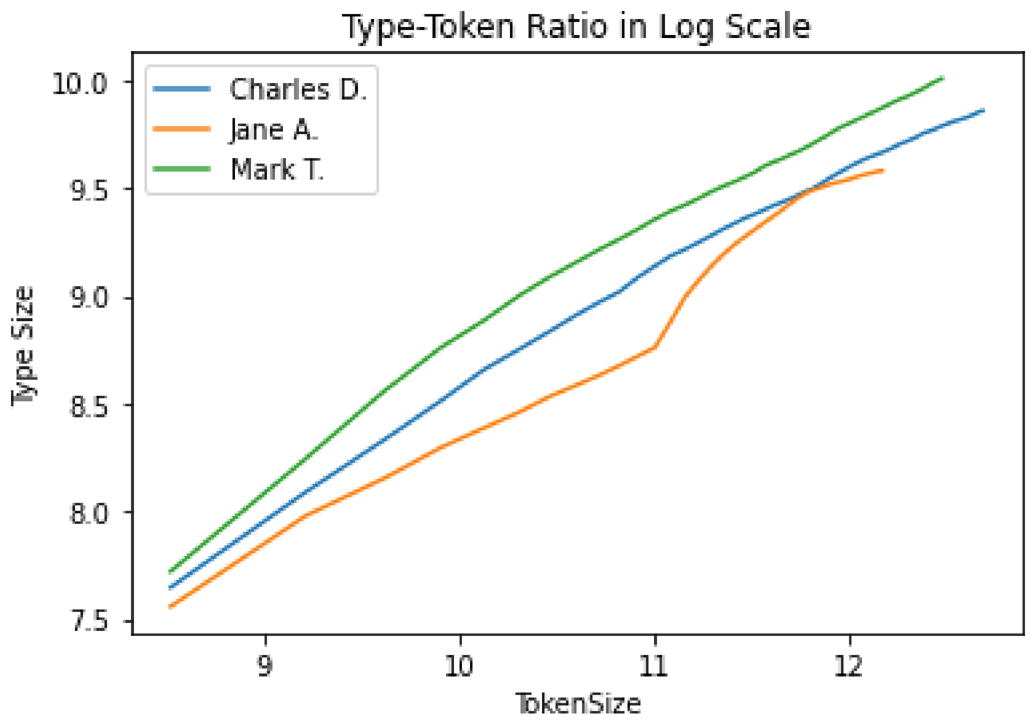

JaneAustin's Books
JaneAustin_1
JaneAustin_2
JaneAustin_3
log(freq)
log(rank)
0
1
2
3
4
5
6
7
0
2
4
6
8


MarkTwain's Books
MarkTwain_1
MarkTwain_2
MarkTwain_3
log(freq)
log(rank)
0
1
2
3
4
5
6
7
0
2
4
6
8
10

8 Type-Token Ratio without Stop Words

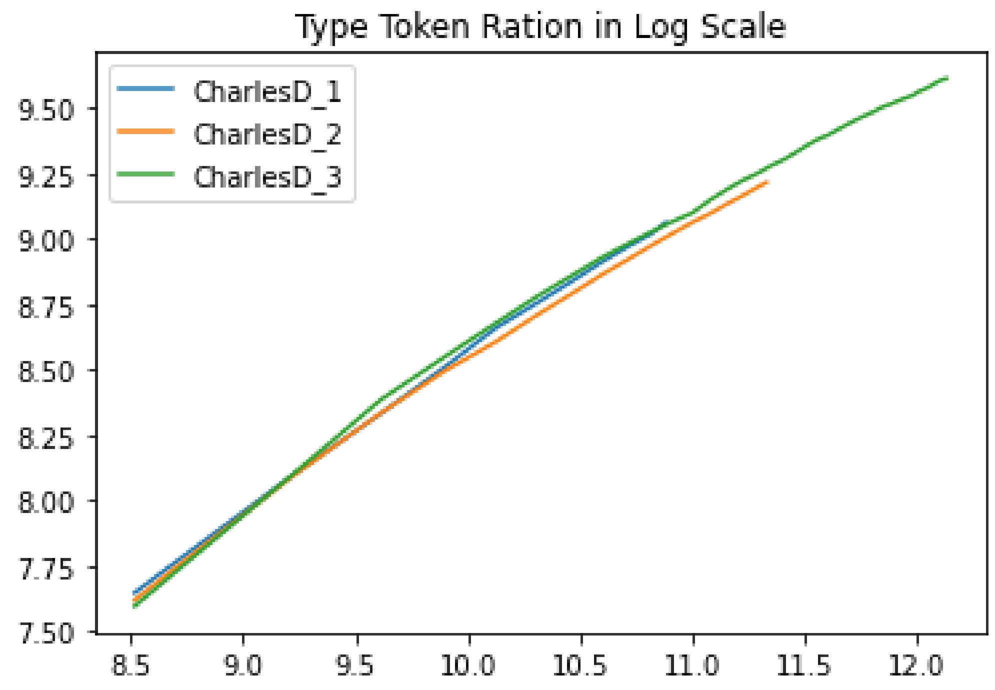
[432]:
```
TypeTokenRatio(author_corpus_sw,5000,format_plot="linear")
TypeTokenRatio(author_corpus_sw,5000,format_plot="log")
```

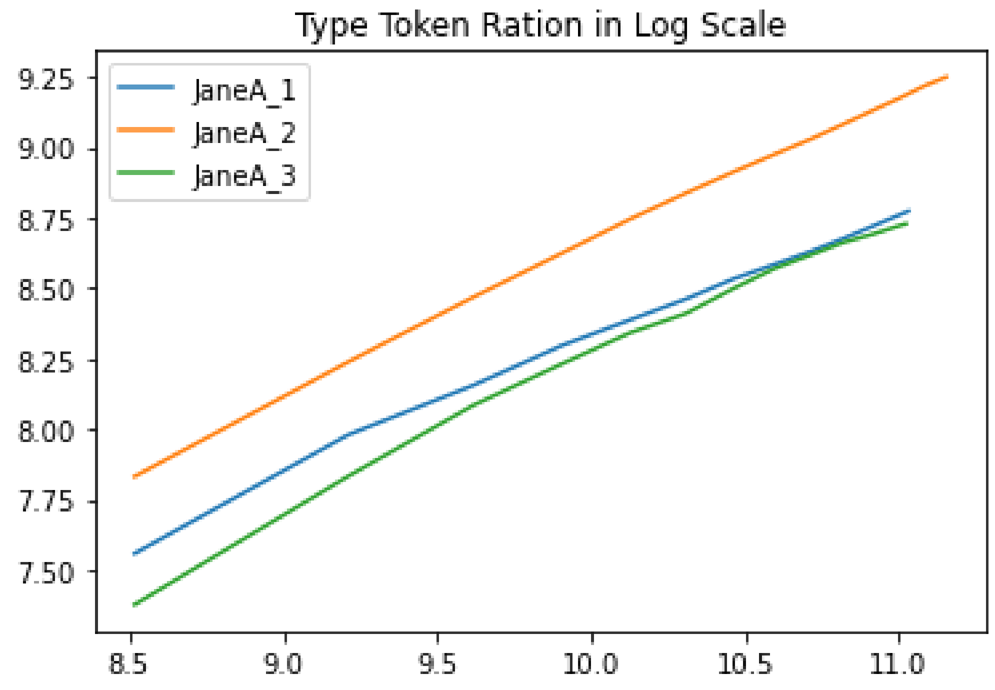


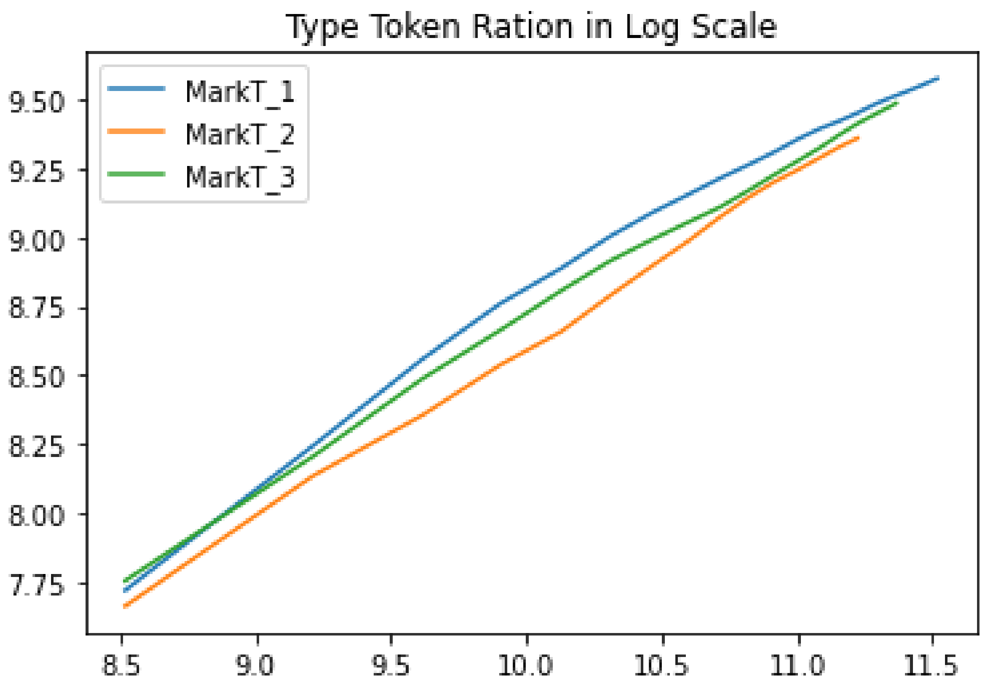


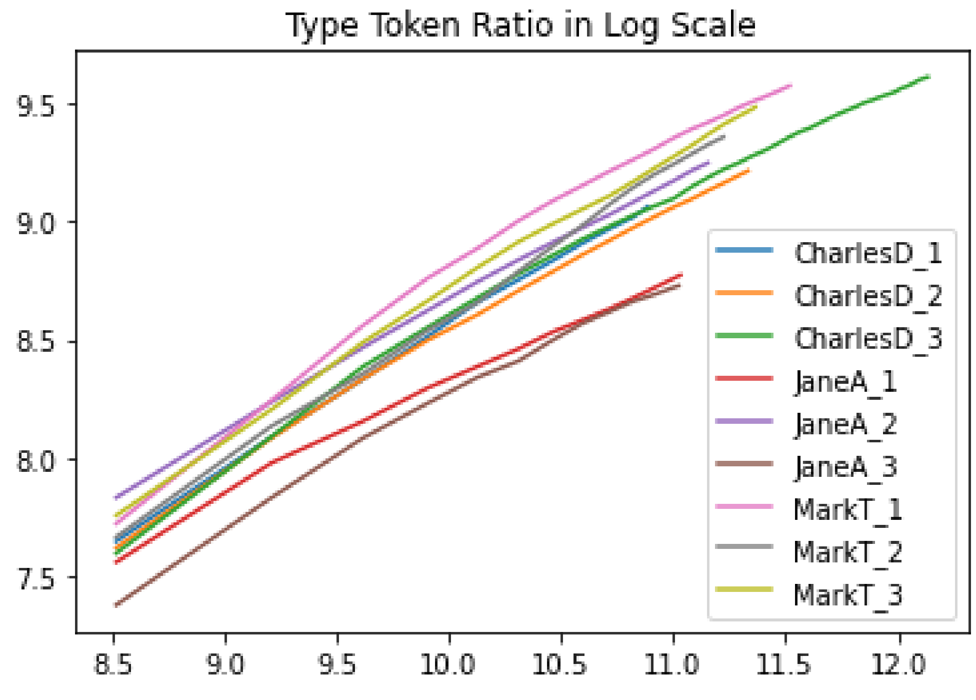
9 Bookwise Type Token Ratio without Stop Words

[433]:
```
# Part i and h
all_types,all_tokens,slopes =␣
 ↪TypeTokenRatioBy(author_corpus_sw,5000,book_type="author")
```


Type Token Ration in Log Scale
CharlesD_1
CharlesD_2
CharlesD_3
9.50
9.25
9.00
8.75
8.50
8.25
8.00
7.75
7.50
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0


Type Token Ration in Log Scale
JaneA_1
JaneA_2
JaneA_3
9.25
9.00
8.75
8.50
8.25
8.00
7.75
7.50
8.5
9.0
9.5
10.0
10.5
11.0


Type Token Ration in Log Scale
MarkT_1
MarkT_2
MarkT_3
9.50
9.25
9.00
8.75
8.50
8.25
8.00
7.75
8.5
9.0
9.5
10.0
10.5
11.0
11.5


Type Token Ratio in Log Scale
CharlesD_1
CharlesD_2
CharlesD_3
JaneA_1
JaneA_2
JaneA_3
MarkT_1
MarkT_2
MarkT_3
9.5
9.0
8.5
8.0
7.5
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0

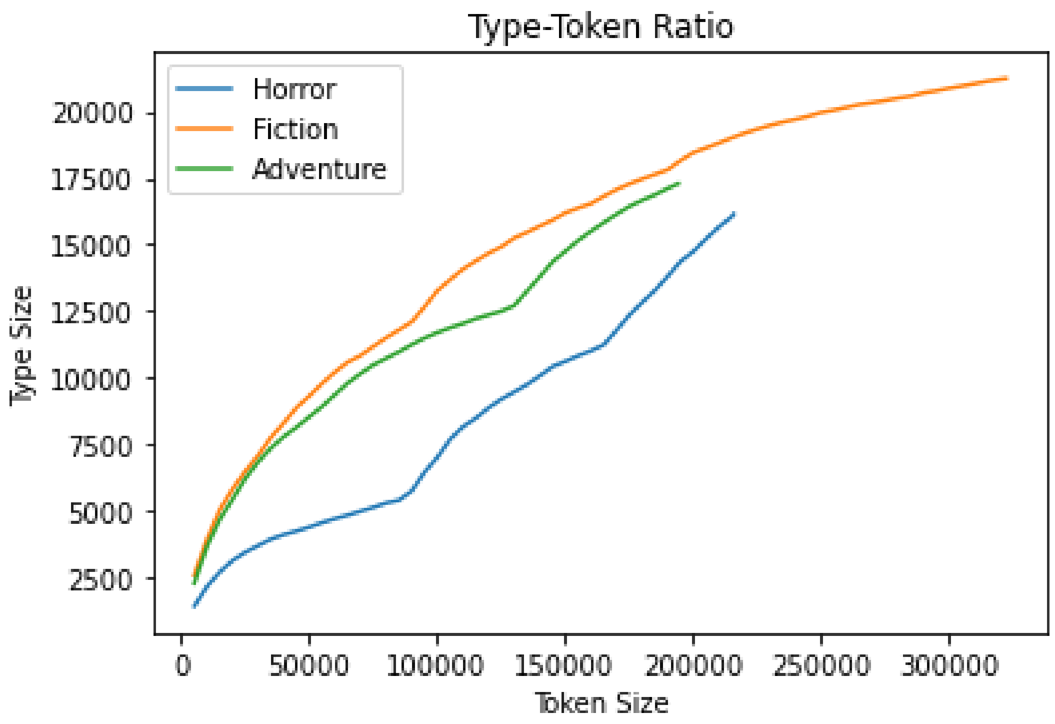
10 ******** Experiments on Genre ************

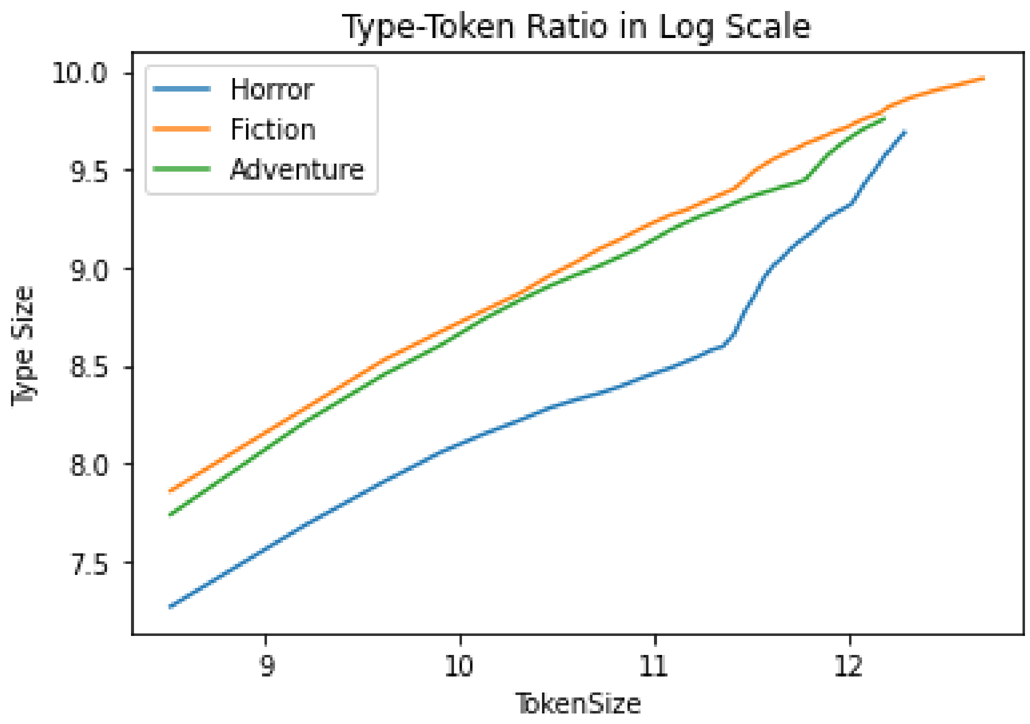
11 Zipf's Law

PlotZipf(genre$_c orpus_s w,"Author"$)

12 Type-Token Ratio without Stop Words

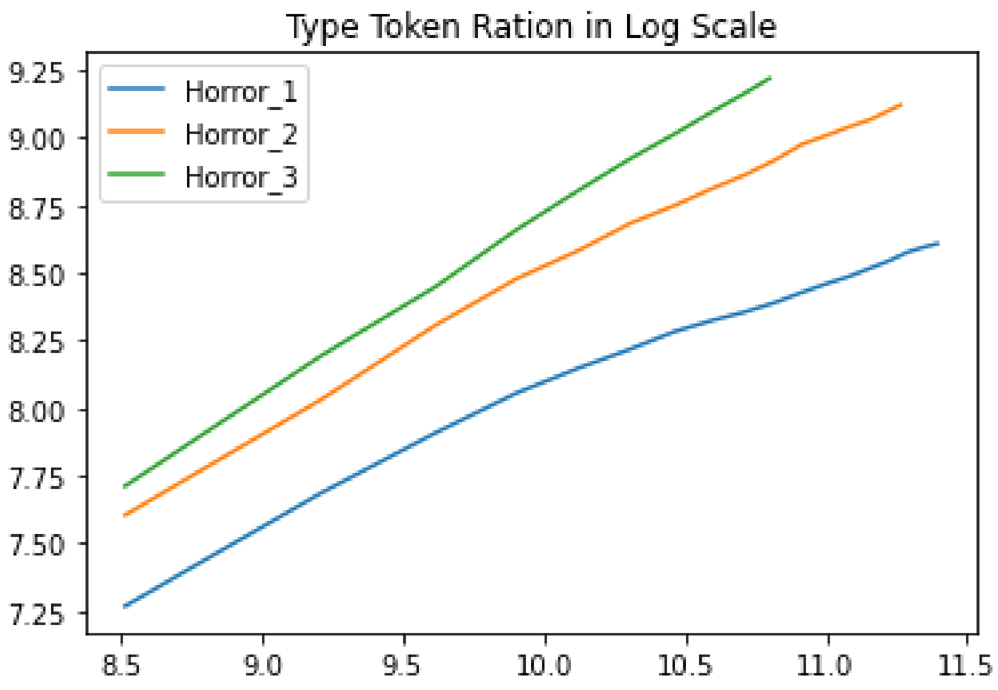
[434]:
```
TypeTokenRatio(genre_corpus_sw,5000,book_type="genre",format_plot="linear")
TypeTokenRatio(genre_corpus_sw,5000,book_type="genre", format_plot="log")
```

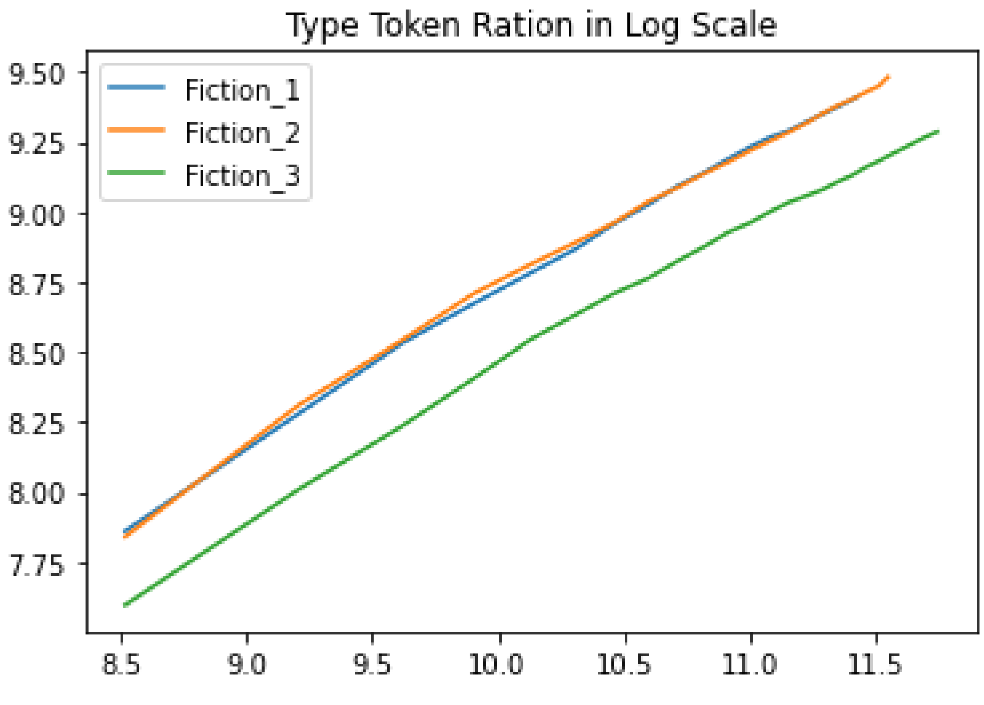


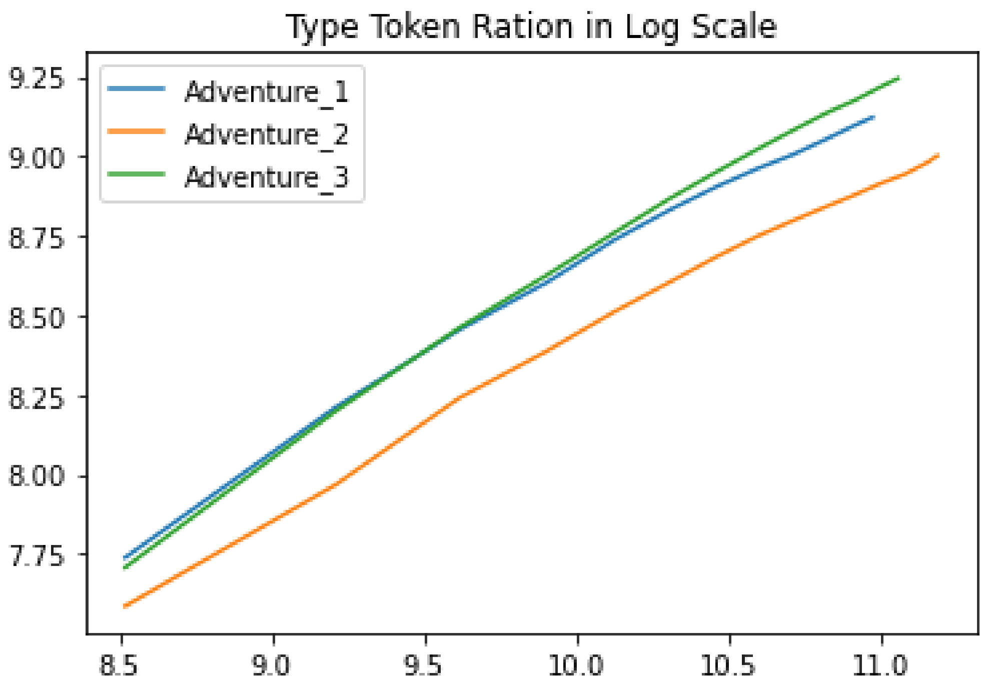


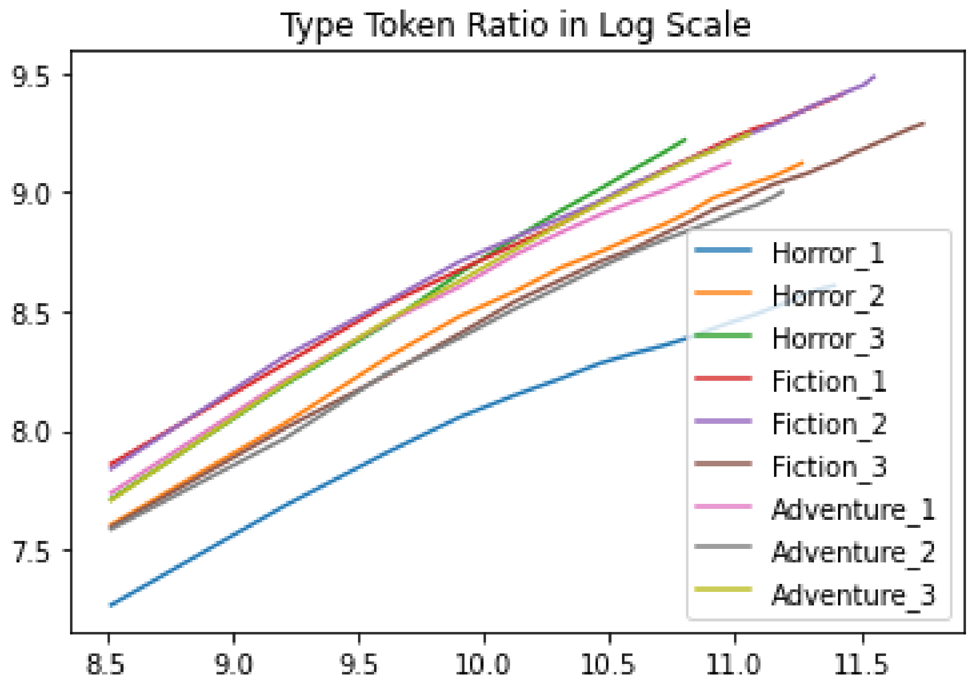
13 Bookwise Type Token Ratio without Stop Words

[435]:
```
all_types,all_tokens,slopes =␣
↪TypeTokenRatioBy(genre_corpus_sw,5000,book_type="genre")
```


Type Token Ration in Log Scale
Horror_1
Horror_2
Horror_3
9.25
9.00
8.75
8.50
8.25
8.00
7.75
7.50
7.25
8.5
9.0
9.5
10.0
10.5
11.0
11.5


Type Token Ration in Log Scale
Fiction_1
Fiction_2
Fiction_3
9.50
9.25
9.00
8.75
8.50
8.25
8.00
7.75
8.5
9.0
9.5
10.0
10.5
11.0
11.5

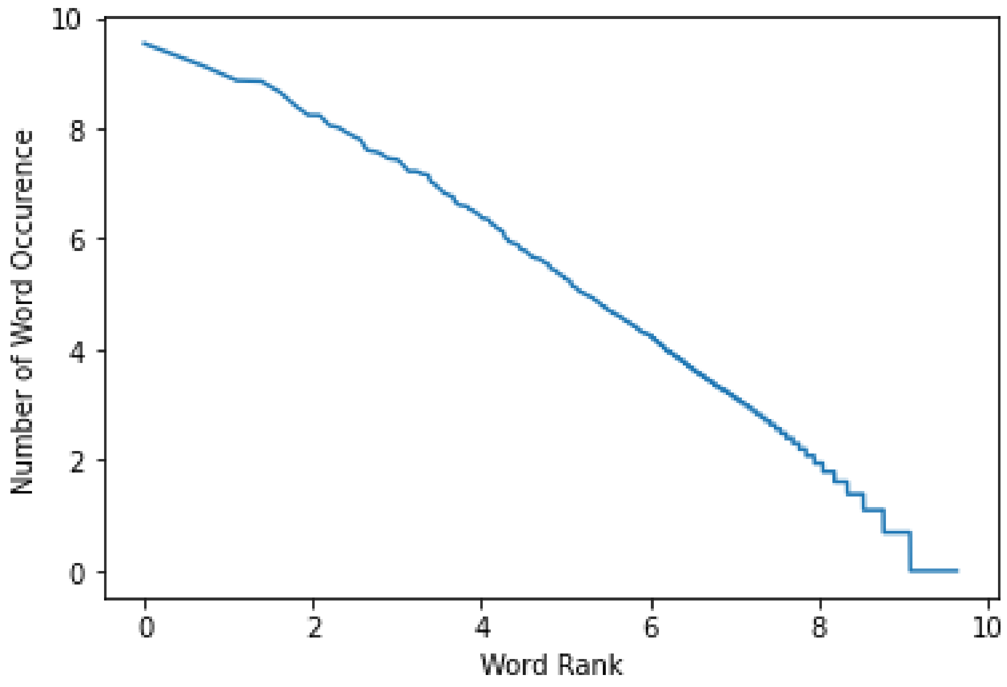

Type Token Ration in Log Scale
Adventure_1
Adventure_2
Adventure_3
9.25
9.00
8.75
8.50
8.25
8.00
7.75
8.5
9.0
9.5
10.0
10.5
11.0


Type Token Ratio in Log Scale
Horror_1
Horror_2
Horror_3
Fiction_1
Fiction_2
Fiction_3
Adventure_1
Adventure_2
Adventure_3
9.5
9.0
8.5
8.0
7.5
8.5
9.0
9.5
10.0
10.5
11.0
11.5

# Part m

```
[451]: merged_book = author_corpus[0]+author_corpus[1]
       merged_book= np.asarray(merged_book)
       wr,fr= np.unique(merged_book,return_counts=True)
       word_freq= np.sort(fr)
       word_freq= word_freq[::-1]
       word_rank= np.arange(1,len(wr)+1)

       plt.plot(np.log(word_rank),np.log(word_freq))

       plt.ylabel("Number of Word Occurence")
       plt.xlabel("Word Rank")
       plt.title("Zipf's Law in Random Text in Log Scale")
```

```
[451]: Text(0.5, 0, 'Word Rank')
```



14 END

```
[ ]:
```