Final_Project_supervised_learning

March 7, 2025

[162]:
```
#run the next cell in case certain libraries are missing
```

[163]:
```
#!pip install nltk
#!pip install powerlaw
```

[164]:
```
import numpy as np
import matplotlib.pyplot as plt
import powerlaw
import scipy.stats as stats
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import nltk
from collections import Counter
from nltk.corpus import brown
from nltk.corpus import gutenberg
```

[304]:
```
#training corpus to gauge fit / p-value
#gutenverg corpus that is primarily full books, stories, and American classics
#medium p value > 0.5, however, it varies considerably from the p value of the␣
↪brown corpus, which is almost 1
#implying potentially different writing styles

def get_corpus_0_training():
    nltk.download('gutenberg')
    lst = gutenberg.words()
    return lst,"gutenberg"


lst,lb = get_corpus_0_training()
```

```
[nltk_data] Downloading package gutenberg to /home/jovyan/nltk_data…
[nltk_data]   Package gutenberg is already up-to-date!
```

[305]:
```
#given a list of strings, identfiy and remove those that are not words.
def tokenize(lst):
```

```
    bad_token = [',','.','?','!','"',':',';','#','/
↪','-','_','$','%','*','(',')','[',']','1','2','3','4','5','6','7','8','9','0',␣
↪' ''','--']
    new_lst = []
    for e in lst:
        if e not in bad_token:
            new_lst.append(e)
    new_lst = [word.lower() for word in new_lst]
    return new_lst

#construct a dictionary with keys as words and values as their frequency

def make_dict(lst):
    freq_dict = {}
    for word in lst:
        if word in freq_dict:
            freq_dict[word] = freq_dict[word] + 1
        else:
            freq_dict[word] = 1
    return freq_dict

#construct a rank,freq tuple
def construct_rf_tuple(freq_dict):
    lst_values = list(freq_dict.values())
    lst_values.sort(reverse=True)
    rf_tuple = []
    i=1
    for val in lst_values:
        rf_tuple.append((i,val))
        i=i+1
    rf_tuple.sort()
    return rf_tuple

new_lst = tokenize(lst)
freq_dict = make_dict(new_lst)
rf_tuple = construct_rf_tuple(freq_dict)
#print(rf_tuple)
```

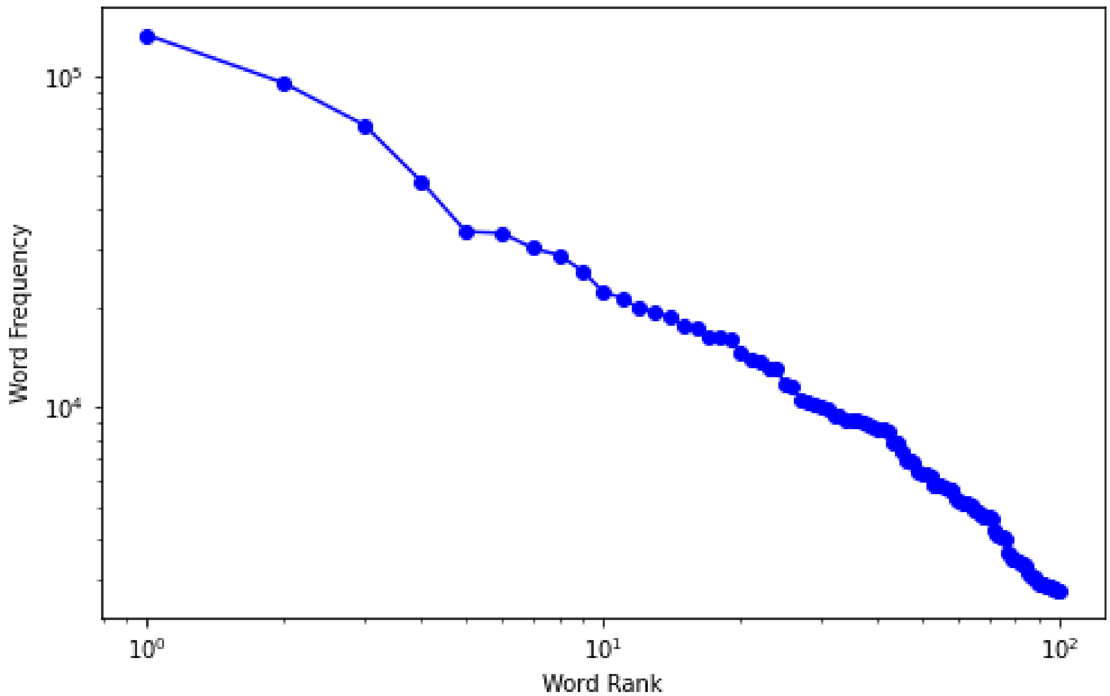
[306]:
```
#plots the data and then determines best fit using the Kolmogorov-Smirnov␣
↪principle (powerlaw)
def plot(rf_tuple):
    ranks,frequencies = zip(*rf_tuple)
    plt.figure(figsize=(8, 5))
    plt.plot(ranks, frequencies, marker='o', linestyle='-', color='b',␣
↪label="Word Frequency")

    plt.xlabel("Word Rank")
```

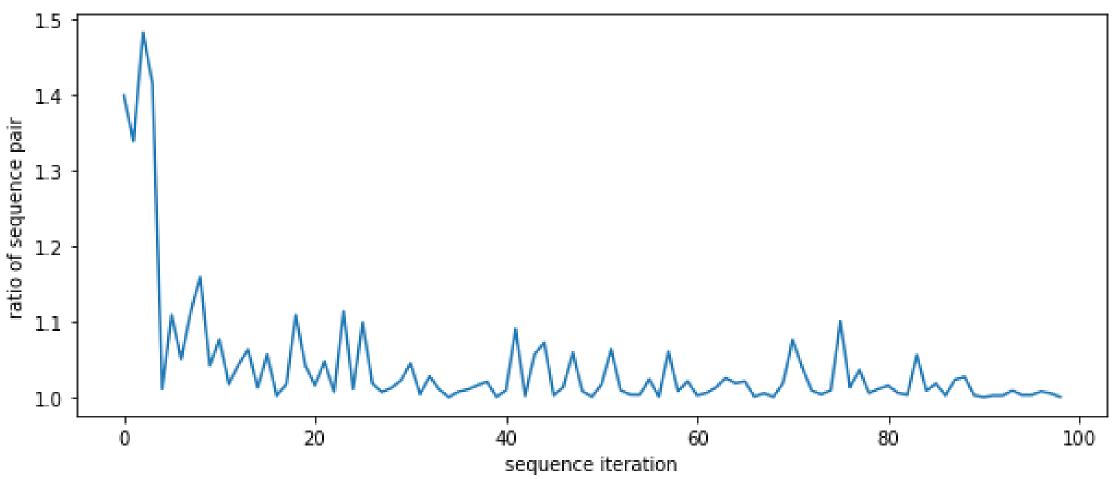
```python
    plt.ylabel("Word Frequency")
    #lets do a log graph
    plt.xscale("log")
    plt.yscale("log")
    diff_words = len(frequencies)
    #avg_frequency = total_words/diff_words
    #print("number of different words", diff_words)
    print("\n ")
    return plt

# a function to show the sequence of ratios between the most frequent words
def ratio_plot(rf_tuple):
    ratio_lst = []
    for i in range(0,len(rf_tuple)-1):
        ratio = rf_tuple[i][1] / rf_tuple[i+1][1]
        ratio_lst.append(ratio)
    #print(ratio_lst)
    frequencies = []
    plt.figure(figsize=(10,4))
    plt.plot(ratio_lst)
    plt.xlabel("sequence iteration")
    plt.ylabel("ratio of sequence pair")

#uses maximum likelihood estimation to determine ideal s_mle coefficient
def KS_test(rf_tuple):
    #print(lb)
    ranks,freqs = zip(*rf_tuple)
    ranks = np.arange(1, len(rf_tuple) + 1)
    i=0
    s_evol=[]
    print("Using Maximum Likelihood Estimation to find best coefficient")
    for i in range(2,len(freqs)):
        m_freqs = np.array(freqs[:i])
        fit = powerlaw.Fit(m_freqs, verbose=False)
        s_mle = fit.alpha
        s_evol.append(s_mle)
    #print(s_evol)

    print(f"Estimated Zipf exponent using MLE: {s_mle:.4f}")

    #hyper_parameters
    s = 1
    #print(freqs)
    C = freqs[0]
    #print(C)
    predicted_freqs = C / ranks**s
```

```
    observed_cdf = np.cumsum(freqs) / np.sum(freqs)
    predicted_cdf = np.cumsum(predicted_freqs) / np.sum(predicted_freqs)

    ks_val, p_val = stats.ks_2samp(observed_cdf,predicted_cdf)

    print(f"KS Statistic: {ks_val:.4f}")
    print(f"P-value: {p_val:.5f}")



def gen_data():
    #print(rf_tuple)
    adj_val = 100
    #this can be scaled depending on the size of the total dataset
    #100 happens to be a fairly consistent value regardless of the dataset size
    plot(rf_tuple[:adj_val])
    ratio_plot(rf_tuple[:adj_val])
    print(lb)
    KS_test(rf_tuple[:adj_val])
```

[307]:
```
gen_data()
```

```
gutenberg
Using Maximum Likelihood Estimation to find best coefficient
xmin progress: 65%Estimated Zipf exponent using MLE: 2.4056
KS Statistic: 0.1100
P-value: 0.58301
```





[300]:
```
#training corpus to gauge fit / p-value
#brown corpus has a wide variety of writing that closely resembles everyday␣
↪speech and writing

def get_corpus_1_training():
    nltk.download('brown')
    lst = brown.words()
```

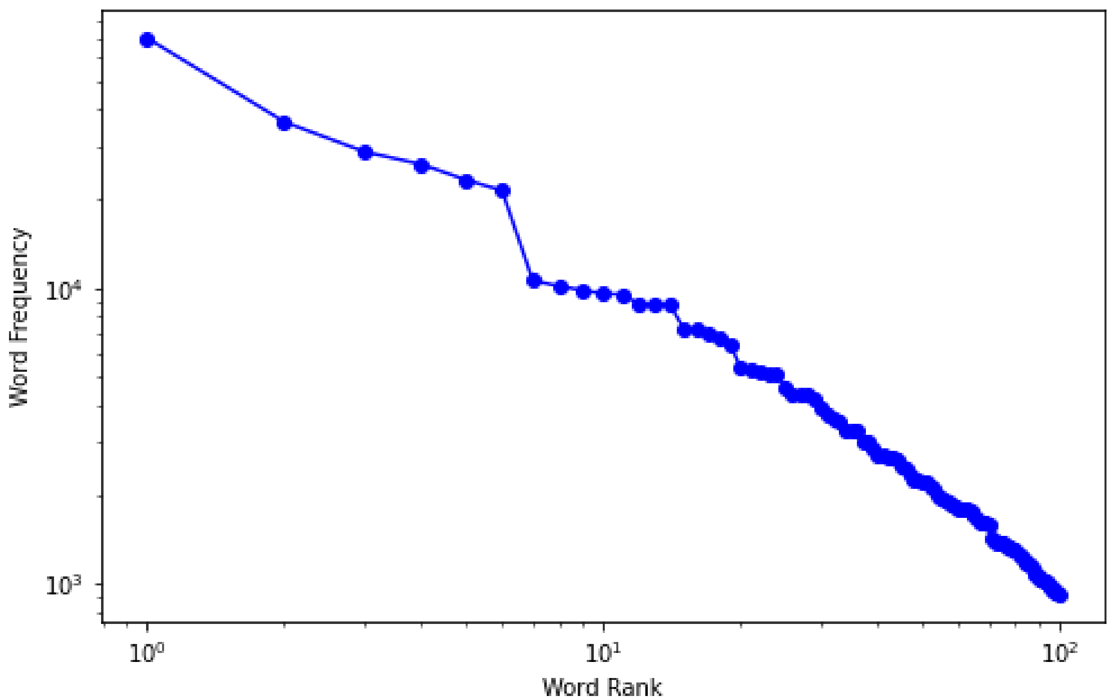
```
    return lst,"brown"

lst,lb = get_corpus_1_training()
```

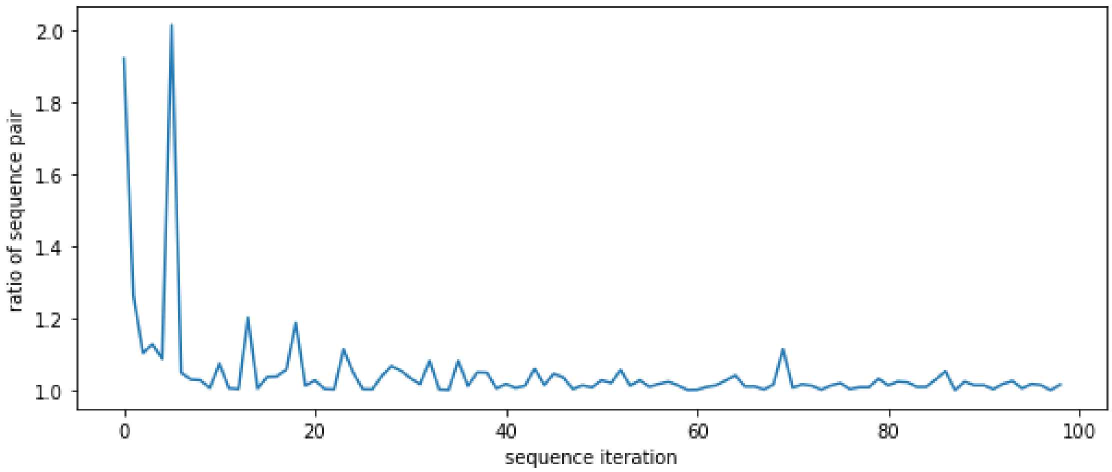
```
[nltk_data] Downloading package brown to /home/jovyan/nltk_data…
[nltk_data]   Package brown is already up-to-date!
```

[261]:

```
gen_data()
```

```
brown
Using Maximum Likelihood Estimation to find best coefficient
Estimated Zipf exponent using MLE: 1.9942
P-value: 0.99969
```





[262]:
```
#observe the two graphs

#the first one is the brown corpus, which has about 1 million words
#the second is the gutenberg corpus which has about 2 million words

#for both datasets, we only take the first 20->100 words or so, as the tail of␣
↪the rank frequency distribution can skew
#the zipfian distribution

#decreasing the size of the frequency list can be seen as decreasing the␣
↪variance while increasing bias

#notice how even the brown corpus is less than size the gutenberg corpus, it is␣
↪better fitted using the KV test,
#and shows a stronger correlation to zipf's law.
#This is very interesting because typically increasing the size of the text␣
↪would show a stronger zipfian distribution
```

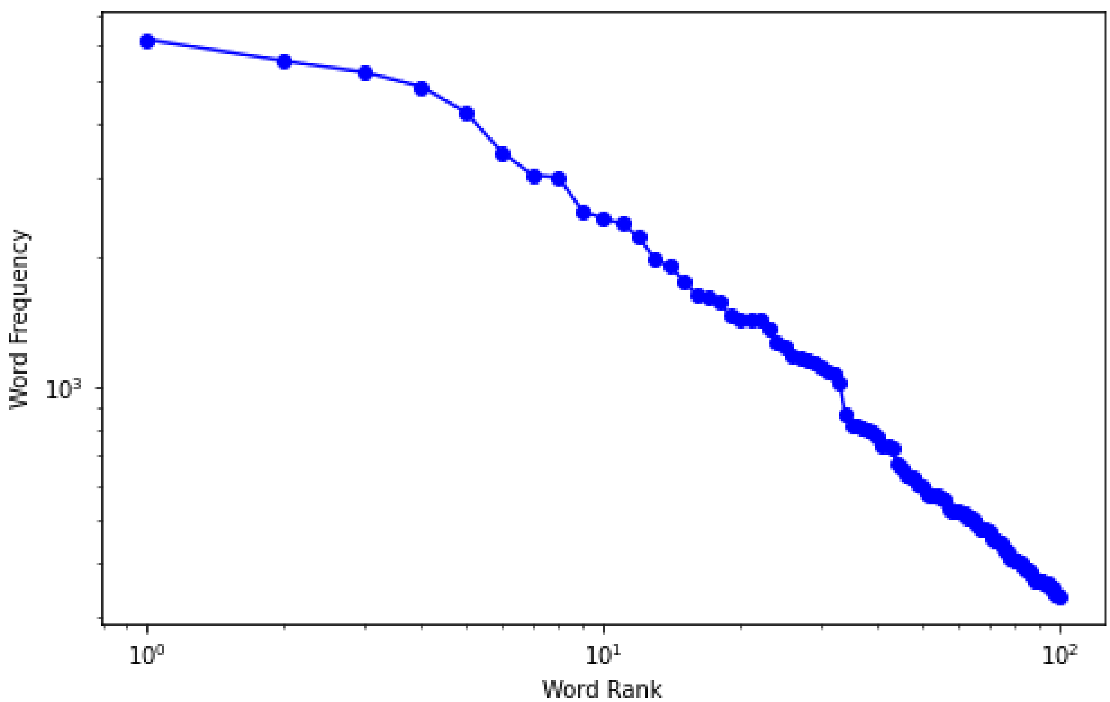
[263]:
```
#the ratio of sequence pairs
#is an experiment to see how much each word differs from the next.
#we do not use this graph to train our model, but it remains an interesting␣
↪sub-topic to delve into
#notice the distribution of the frequencies for both the gutenberg and the␣
↪brown corpi have a very similar pattern
```

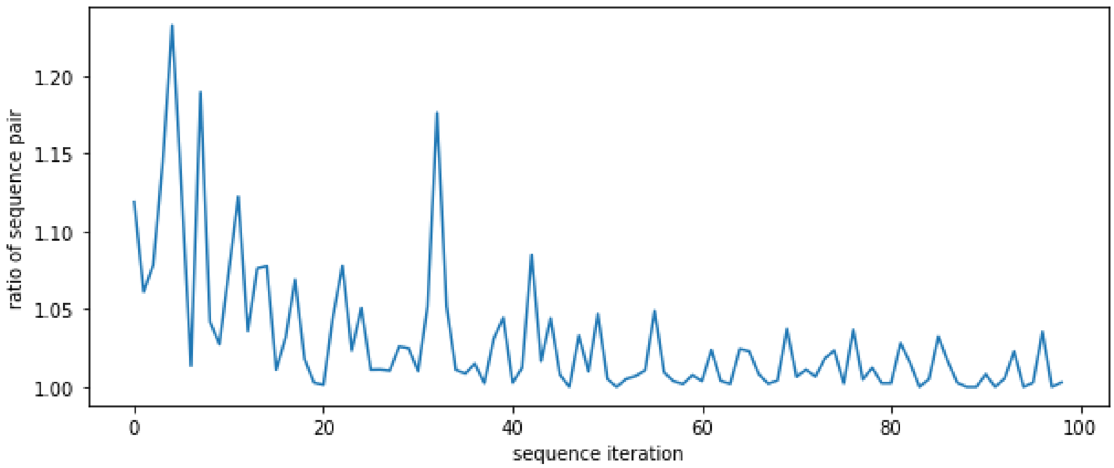
[69]:
```
#we deduce that merely the number of words is not the only factor, but the␣
↪differing writing styles
#which leads us to design a hypothesis, do different writing styles show␣
↪different p values when fitted using the power law?
```

```
#gutenberg corpus: focus on literature, classics, full books and stories
#brown corpus: larger variety of text from a variety of genres, including news
↪articles, nonfiction and fiction

#We suspect that high p values correspond more to everyday text while lower p
↪values correspond more to artistic writing
#We now find our test sets.
```

[287]:
```
gen_data()
```

```
shakespeare
Using Maximum Likelihood Estimation to find best coefficient
Estimated Zipf exponent using MLE: 2.1927
KS Statistic: 0.1600
P-value: 0.15484
```



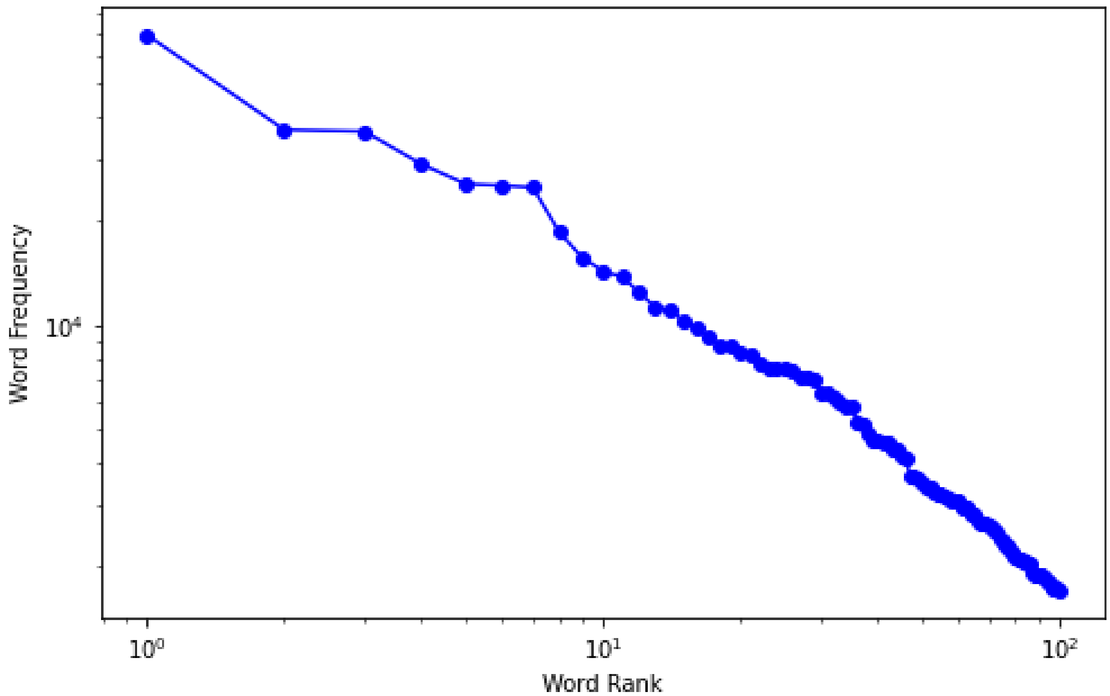


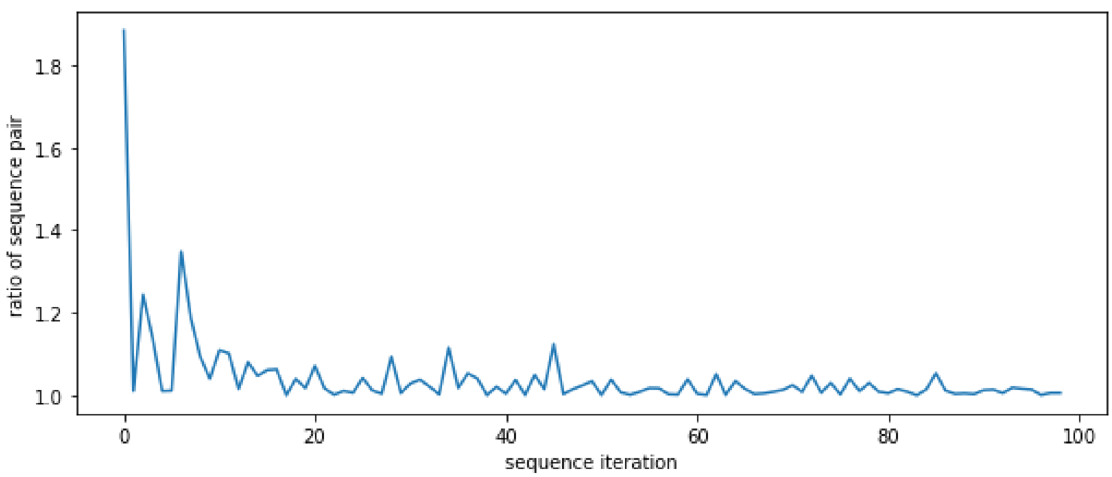
[284]:
```
#test corpus of shakespeare's plays
#suspected to have a low p_value given shakespeare's very wide vocabulary

from nltk.corpus import shakespeare

def get_corpus_2_testing():
    nltk.download('shakespeare')
    lst = []
    for file_id in shakespeare.fileids():
        lst.extend(shakespeare.words(file_id))
    #print(lst)
    return lst,"shakespeare"

lst,lb = get_corpus_2_testing()
```

```
[nltk_data] Downloading package shakespeare to
[nltk_data]     /home/jovyan/nltk_data…
[nltk_data]   Package shakespeare is already up-to-date!
```

[291]:
```
gen_data()


#as suspected, Shakespeare follows zipfs law less closely with a low p_value
```

```
reuters
Using Maximum Likelihood Estimation to find best coefficient
Estimated Zipf exponent using MLE: 2.0327
KS Statistic: 0.1100
P-value: 0.58301
```





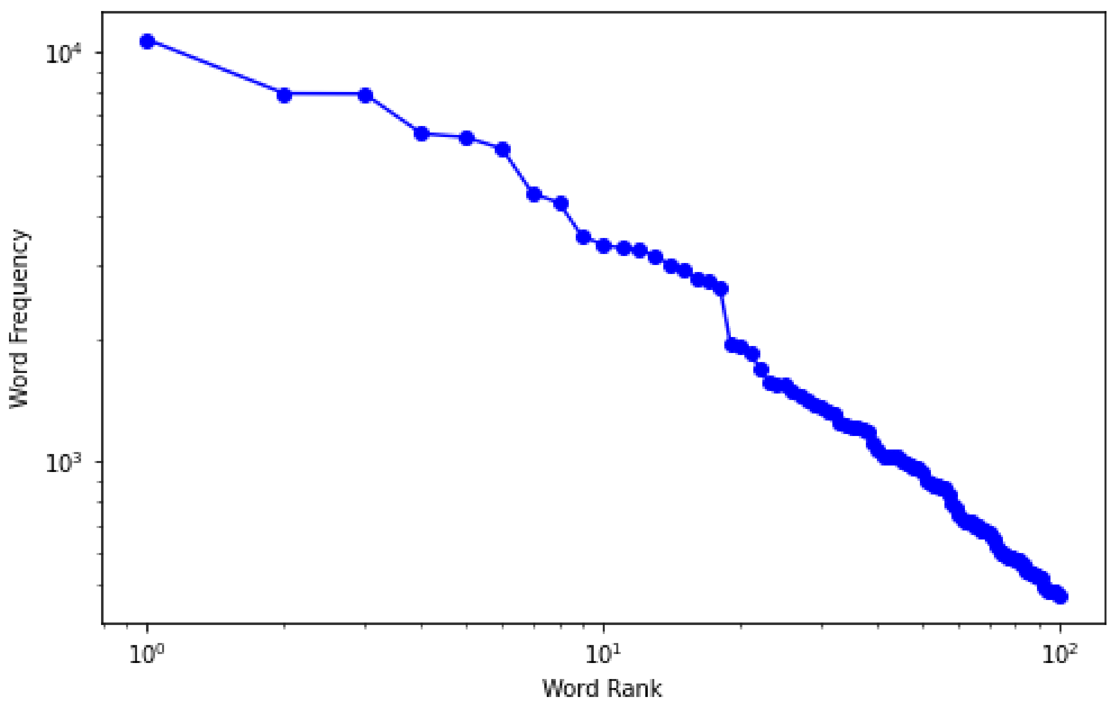
[295]:
```
#test corpus of news articles
#suspected to have a high p_value
#while the p value was not as high as we suspected, it was still significant␣
↪enough to determine a zipfian distribution

from nltk.corpus import reuters
```

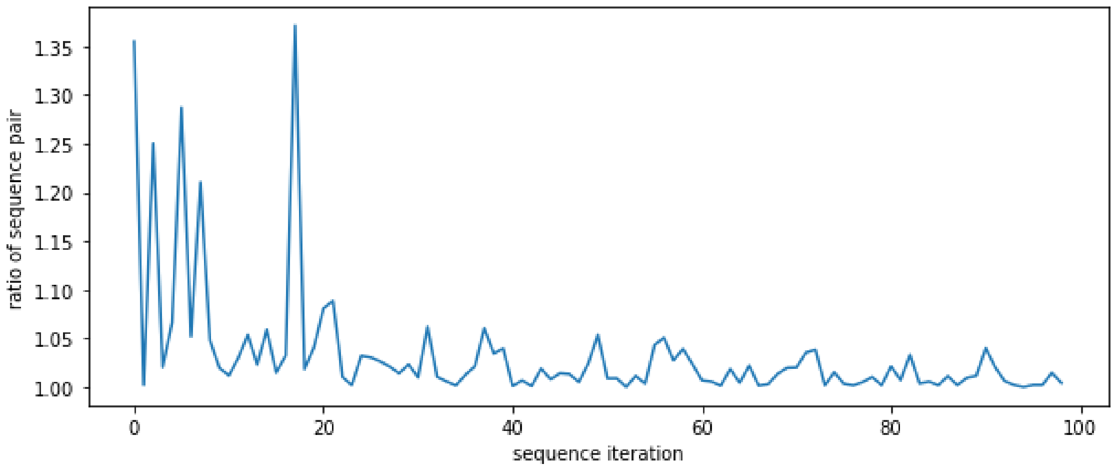
```python
def get_corpus_3_testing():
    nltk.download('reuters')
    lst = reuters.words()
    return lst,"reuters"

lst,lb = get_corpus_3_testing()
```

```
[nltk_data] Downloading package reuters to /home/jovyan/nltk_data…
[nltk_data]   Package reuters is already up-to-date!
```

[299]:
```python
gen_data()
```

```
webtext-poetry
Using Maximum Likelihood Estimation to find best coefficient
Estimated Zipf exponent using MLE: 2.1829
KS Statistic: 0.1600
P-value: 0.15484
```





[104]:
```
# based on our assumptions, we see that a p value > .4 implies a more␣
↪consistent, easy to gauge pattern in writing that
# we associate with more everyday speech

#a low p_value < .2 implies a less consistent pttern that we associate with␣
↪more elaborate writing, like in poetry or plays
```

[296]:
```
#test corpus of webtext poetry
#suspected to have a low p  (< 0.2)

from nltk.corpus import webtext

def get_corpus_4_testing():
    nltk.download('webtext')
    lst = webtext.words()
    return lst, "webtext-poetry"

lst,lb = get_corpus_4_testing()
```

```
[nltk_data] Downloading package webtext to /home/jovyan/nltk_data…
[nltk_data]   Package webtext is already up-to-date!
```

[147]:
```
#test_corpus_2 (shakespeare) was suspected to not follow zipfs law because
#which is confirmed with the p value it generates

#test_corpus_3 (retuers) was suspected to follow zipfs law
#this is confirmed with the relatively high p value it generates

#test_corpus_4 (webtext(poetry)) was suspected to not follow zipfs law
#this is confirmed with the relatively small p value it generates
```

[148]:
```
#this classification technique perhaps allows us to create a spectrum of␣
↪writing styles. Perhaps genres that are more ambiguous follow zipf's law more
#for this specific project, we attempted to find 2 extremes
#given the subjectivity involved in determining genre, there is room for error␣
↪depending on the text compiled
#with that said, we can see a significant deviation in p value that we can at␣
↪least partly attribute to writing style

#another thing to keep in mind is the generated s_coeff value - lower␣
↪s_coefficient values are correlated with higher p values
```

[229]:
```
#conclusion and further discussions

#Using a combination of techniques (logistic regression, maximum likelihood␣
↪estimation)
#and several principles (zipf's law, ratio sequence)
#we can use the p value of an attempted fit in order to infer the type of␣
↪writing styles involved in a text
#given that writing style, like a p value, can be understood as a spectrum, we␣
↪attempted to find writing styles and combinations
#that gave high and low p-values, respectively

#for efficiency and estimation reasons, we decided to look only at the␣
↪distribution of the first 100 words

#we learned that poetry and plays tend to have a low p value of 0.1, while the␣
↪reuters article had a p value of approximately 0.5
#and the brown corpus had a very high p value of .9998

#the gutenberg corpus we used to train our concept model had a compilation of␣
↪different texts, but they varied less than that of the brown corpus
#increasing the variation of text is more likely to cause a stronger zipfian␣
↪distribution.

#other thoughts:
#graphing the sequence of ratios for nearly every data set showed a striking␣
↪pattern, where as the number of different words we
#looked at increased, the sequence of words that had an identical frequency␣
↪(ratio of 1) became longer.

#elaborations and further experimentation
#other ideas that I played around with was measuring the stability of the s␣
↪value as the number of different words increased
#and different means of collecting data
```

```
#one idea is to use corpi where each corpus was written by a single author,␣
↪instead of attempting to determine the specific genre
#the problem with this approach is finding enough authors who write␣
↪prolifically so our dataset is large enough

#notice how we did not the words themselves as a criteria for classification
```

[230]:

```
#references

#this project was my own, but the concept of zipfs law and the␣
↪Kolmogorov-Smirnov test was studied
#in order to approach this problem

#all code is my own but many python libraries were used
```