Report for Natural Language Processing

Nikhil Rayaprolu
201501090

1. Implement unigram, bigram and trigram language models.
Implementation of the language models has been done in the code,
Steps require:

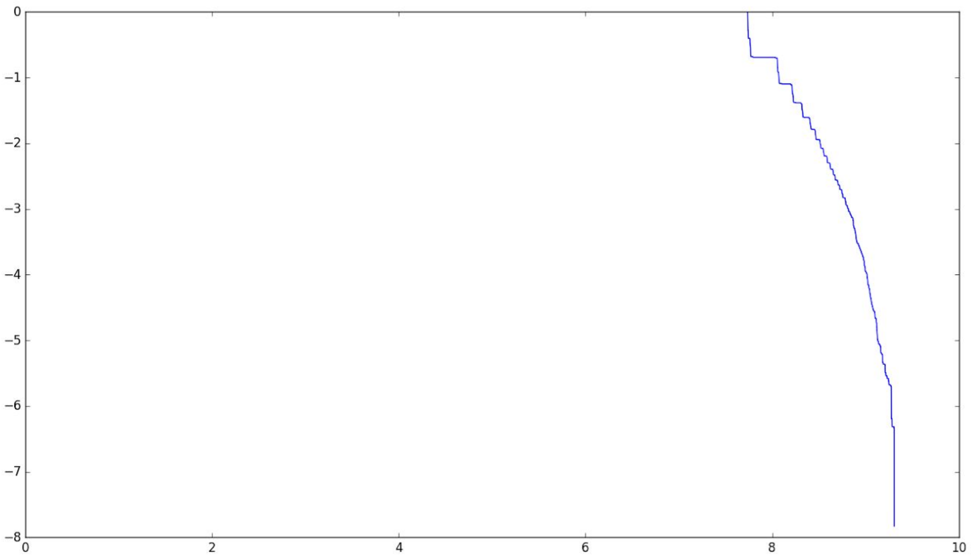
```
corpus=tokenise()
unigrams,unigrams_prob=get_unigrams(corpus)
bigrams,bigrams_prob = get_bigrams(corpus,unigrams)
trigrams,trigrams_prob = get_trigrams(corpus,bigrams)
```

2.Plot log-log curve and zipf curve for the above:
Using

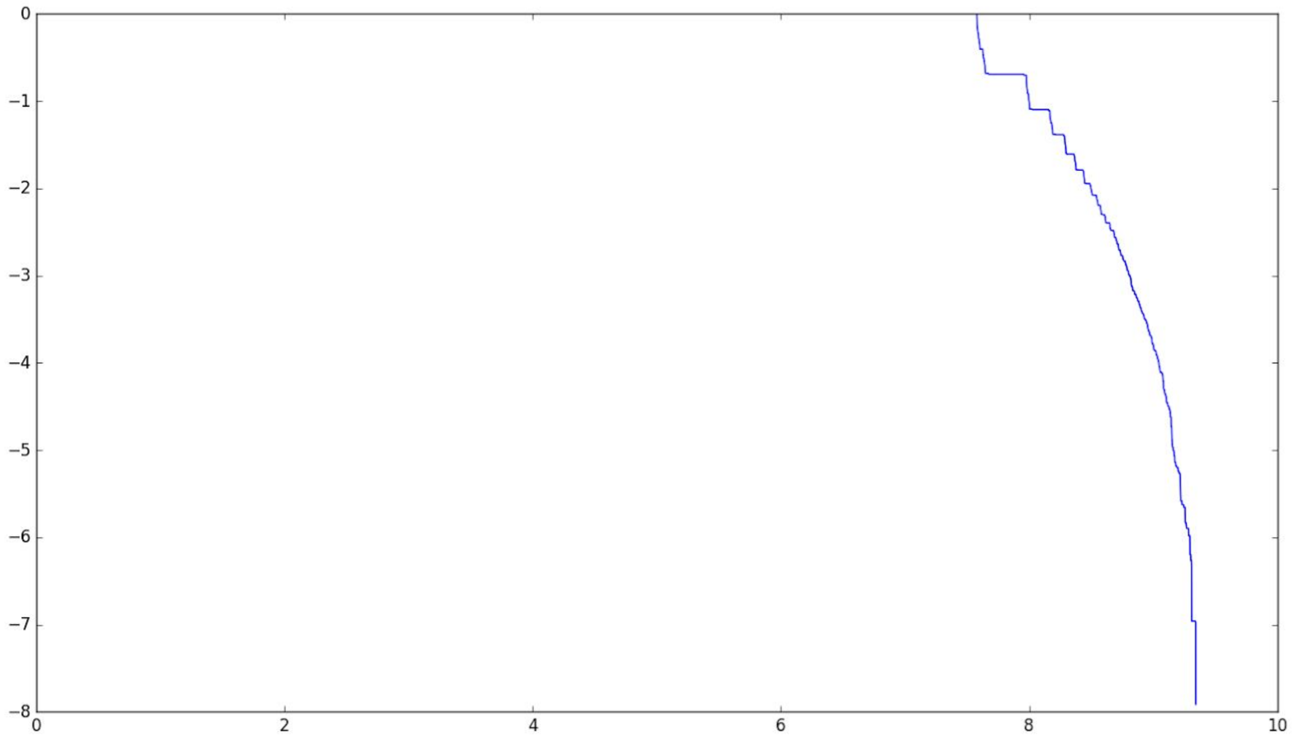
```
plot(sort_dict(unigrams_prob))
plot_log_log1(sort_dict(unigrams_prob))
```

Different plots we got:

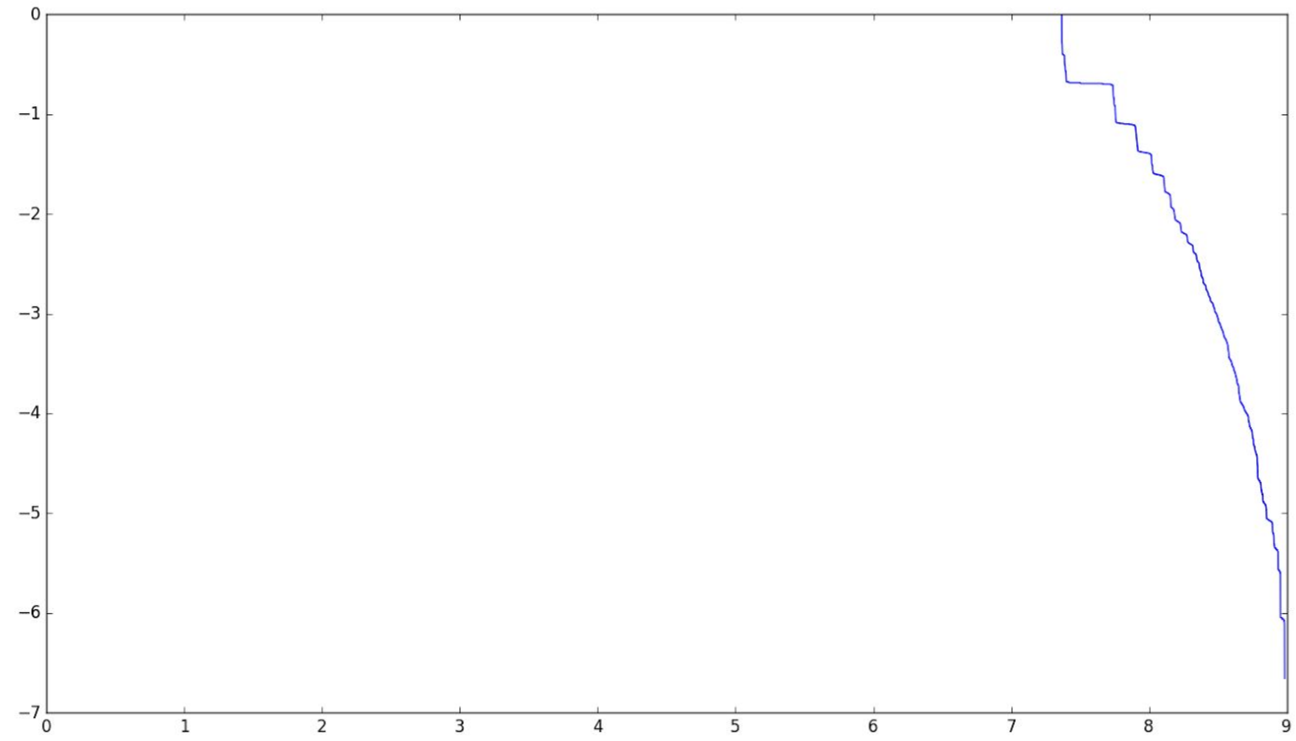
normal_bigram_log_anime



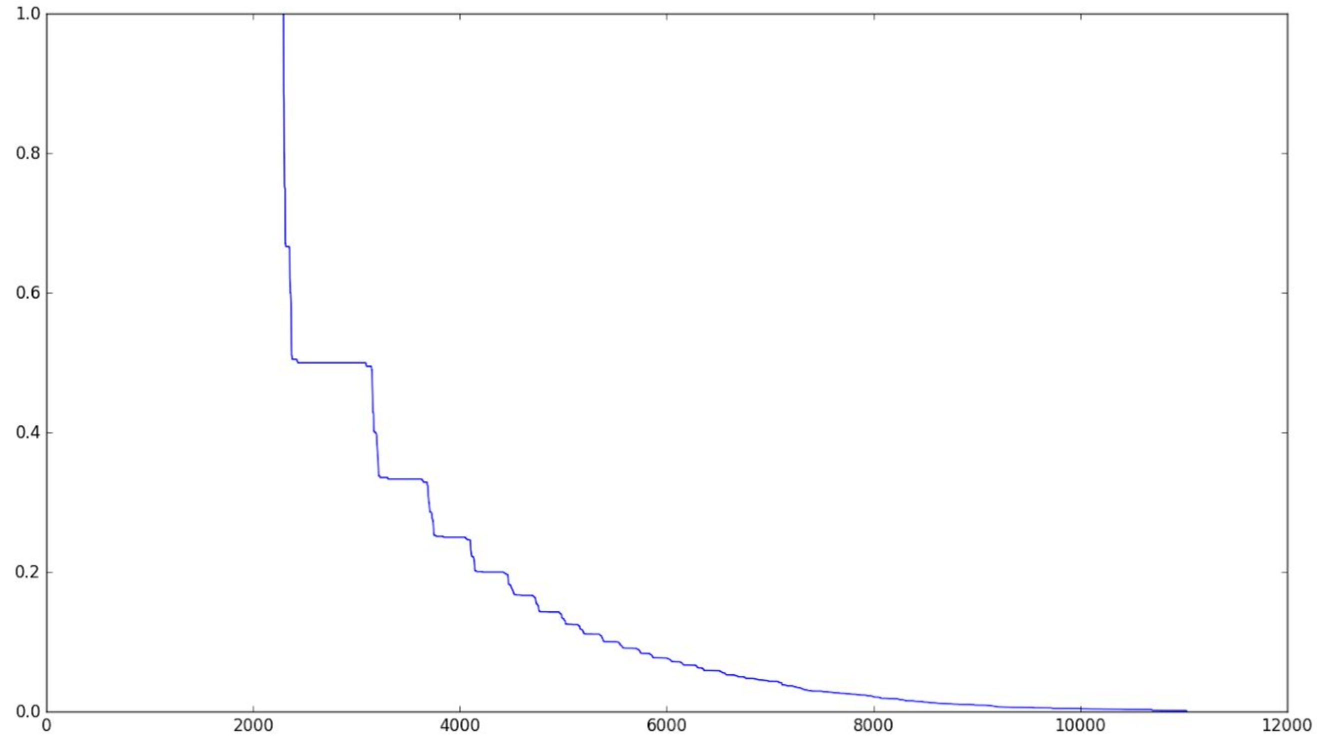
normal_bigram_log_movies



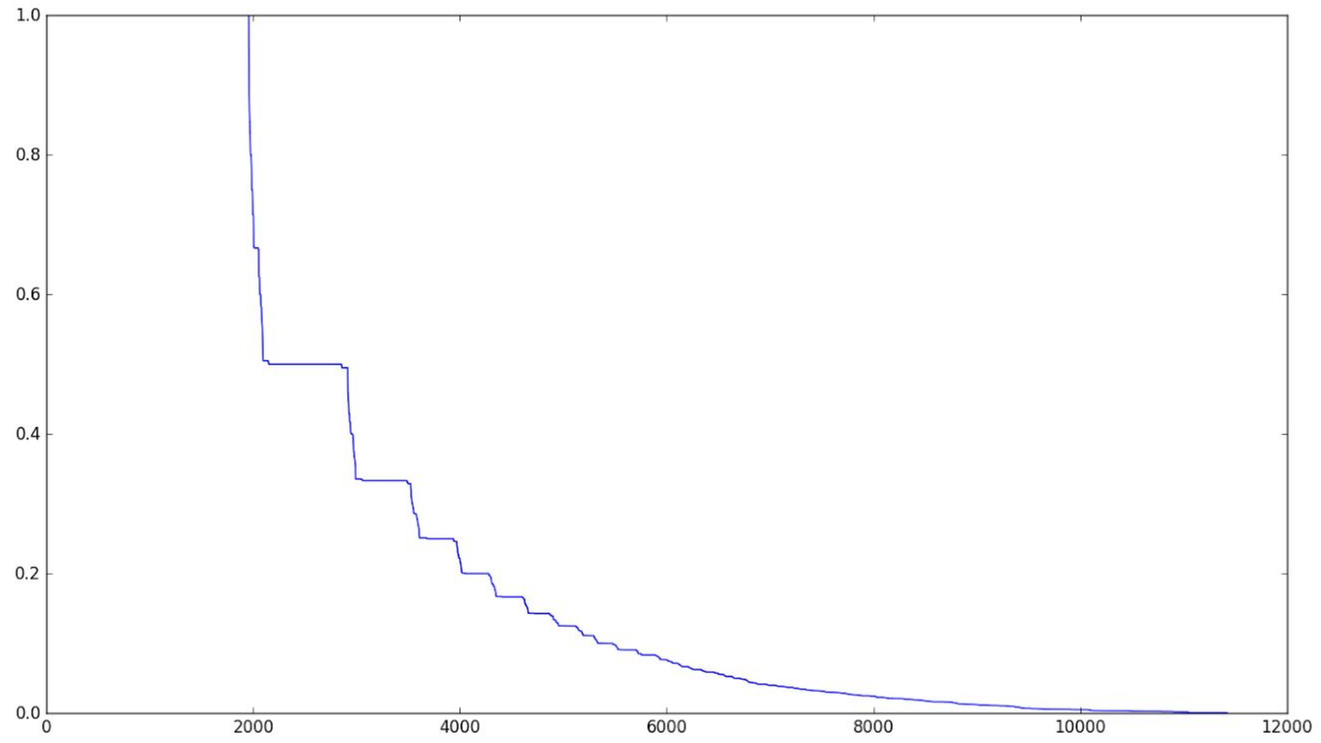
normal_bigram_log_news



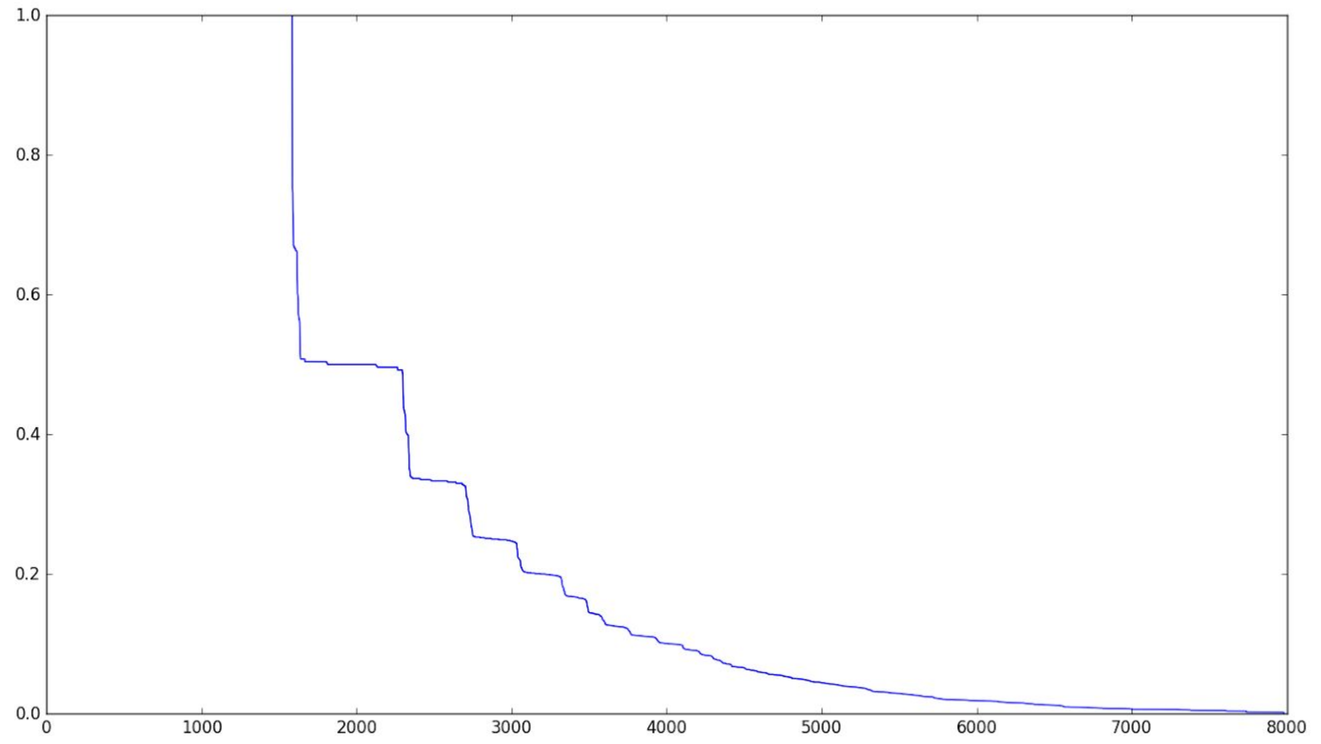
normal_bigram_zipf_anime



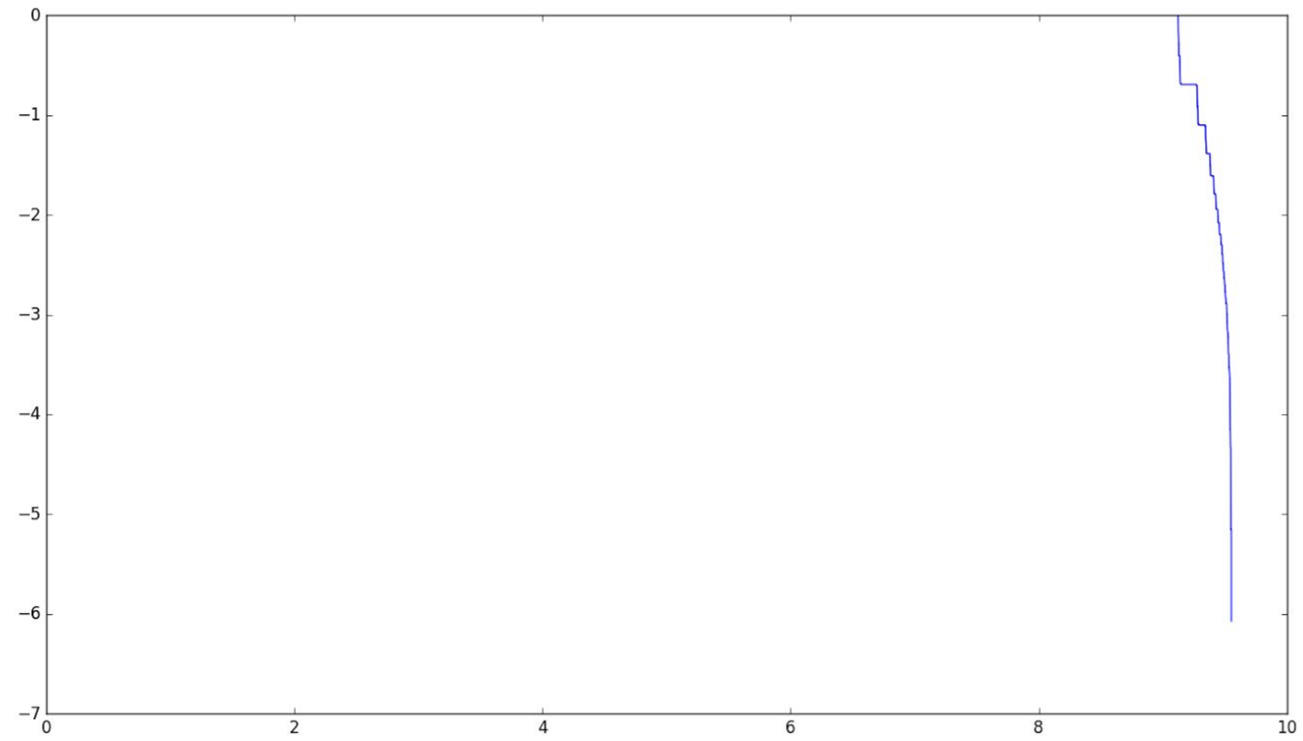
normal_bigram_zipf_movies



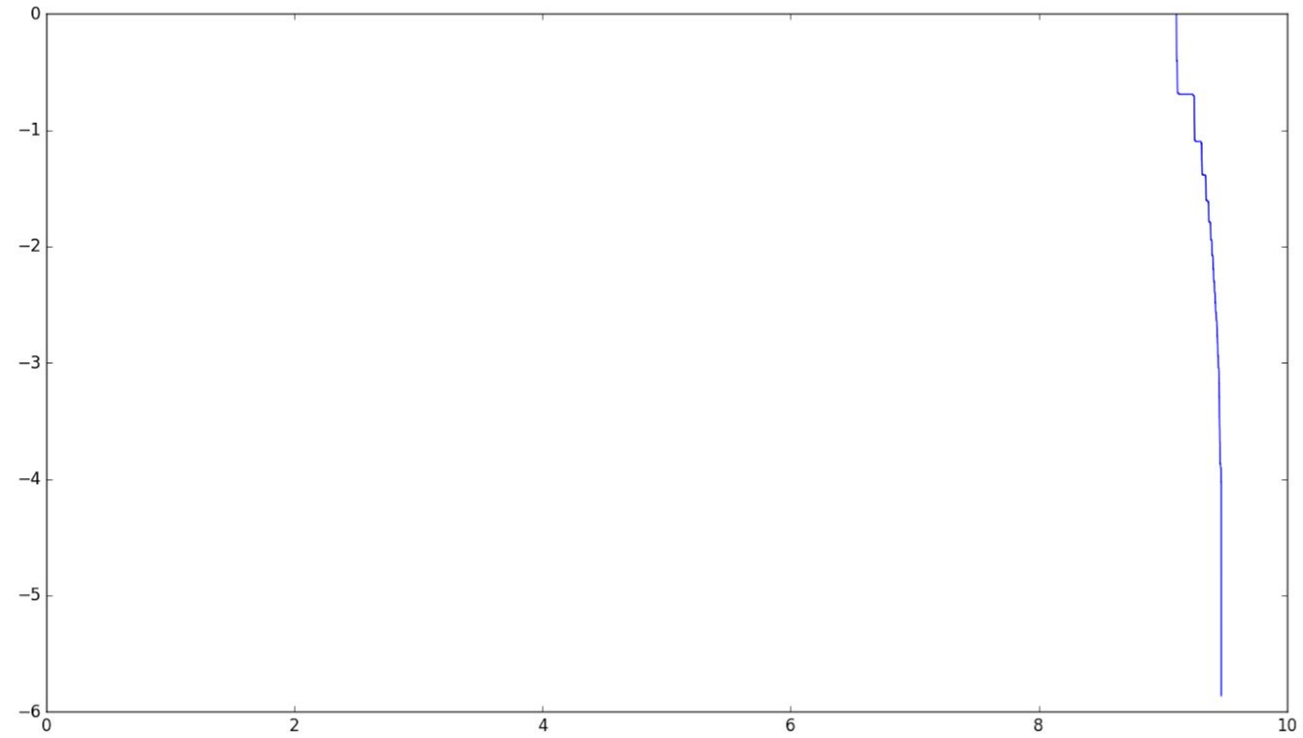
normal_bigram_zipf_news



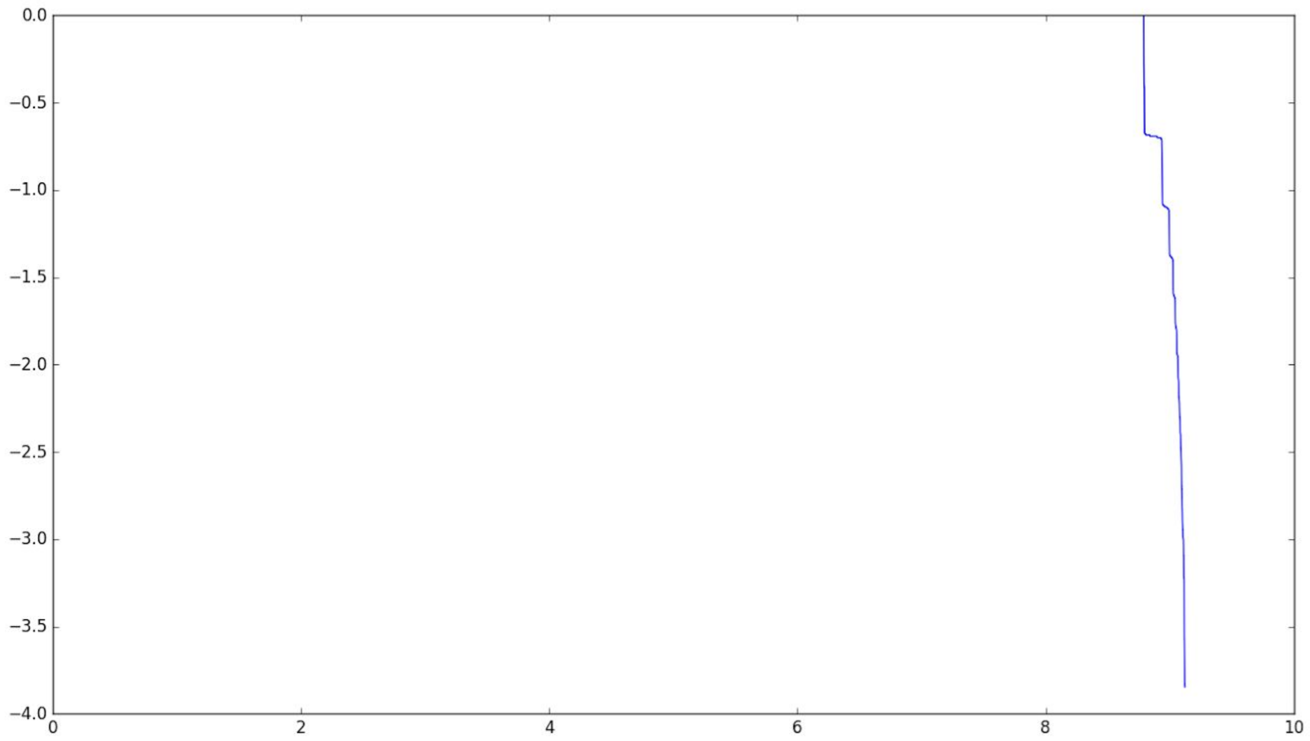
normal_trigram_log_movies



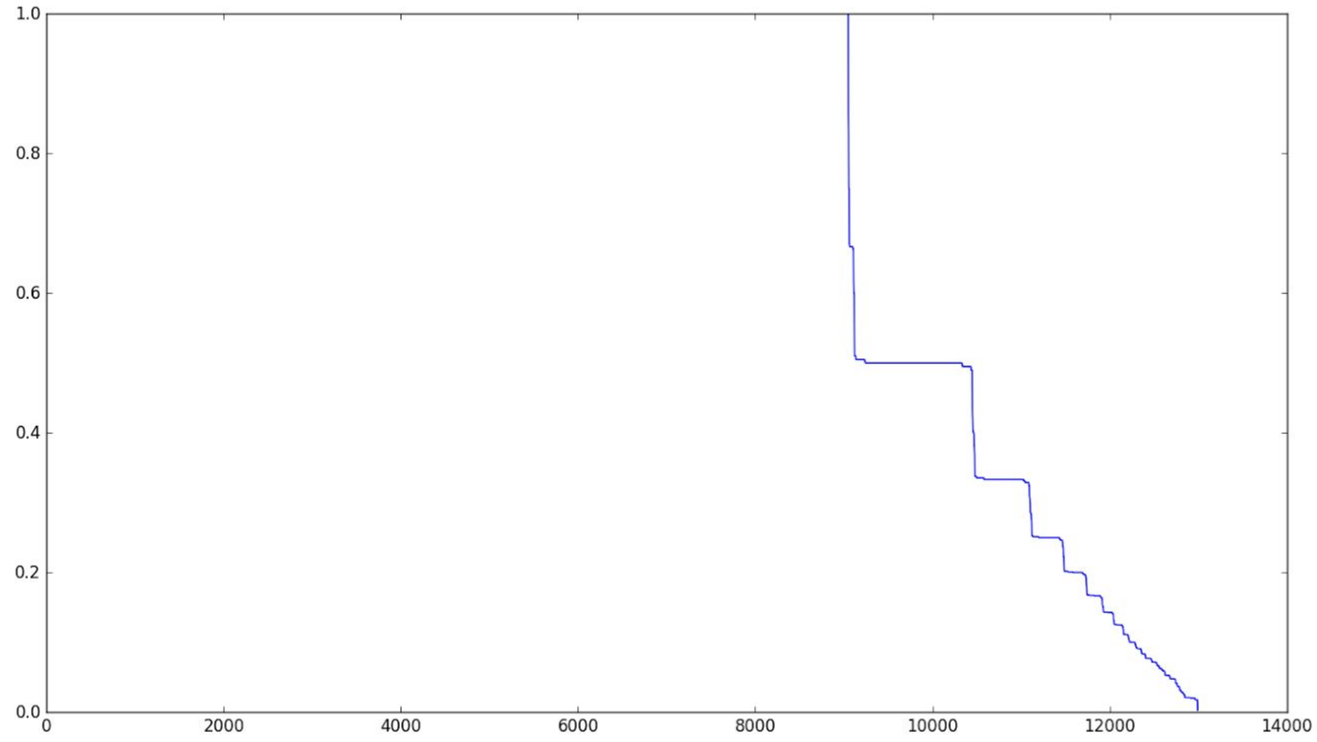
normal_trigrams_log_anime



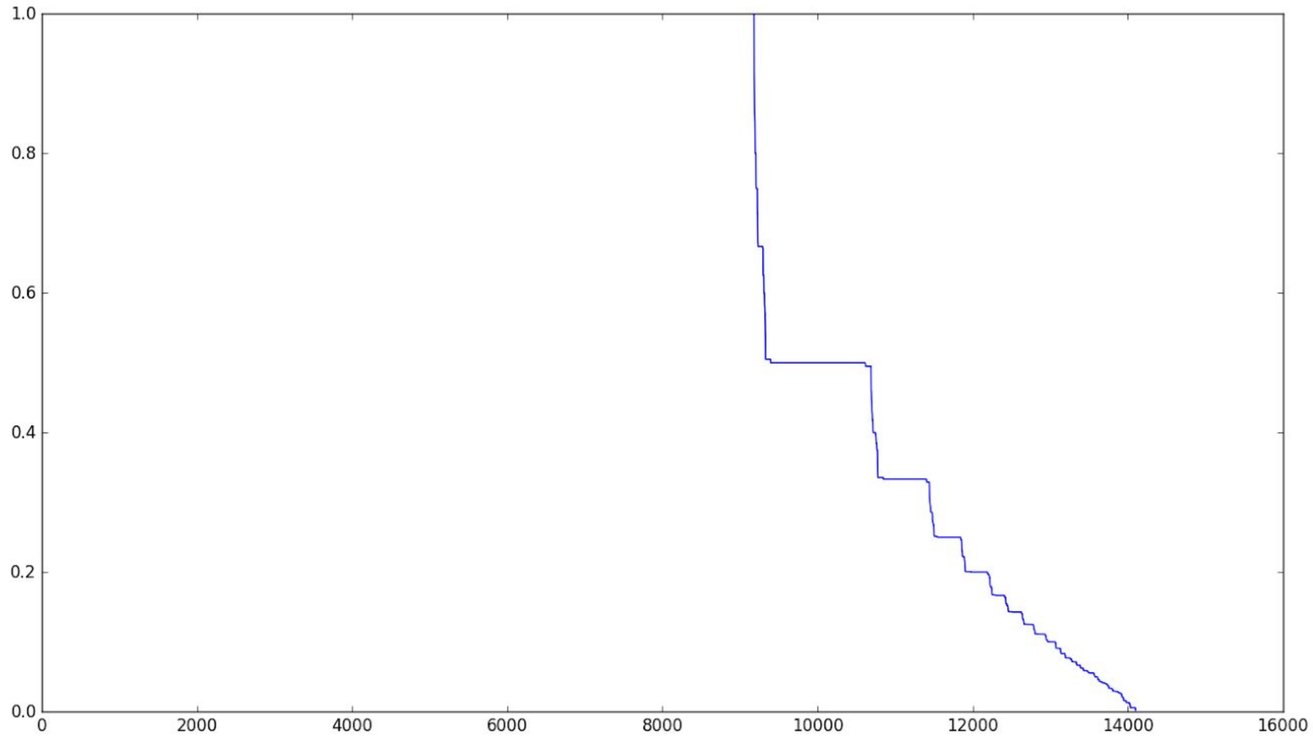
normal_trigrams_log_news



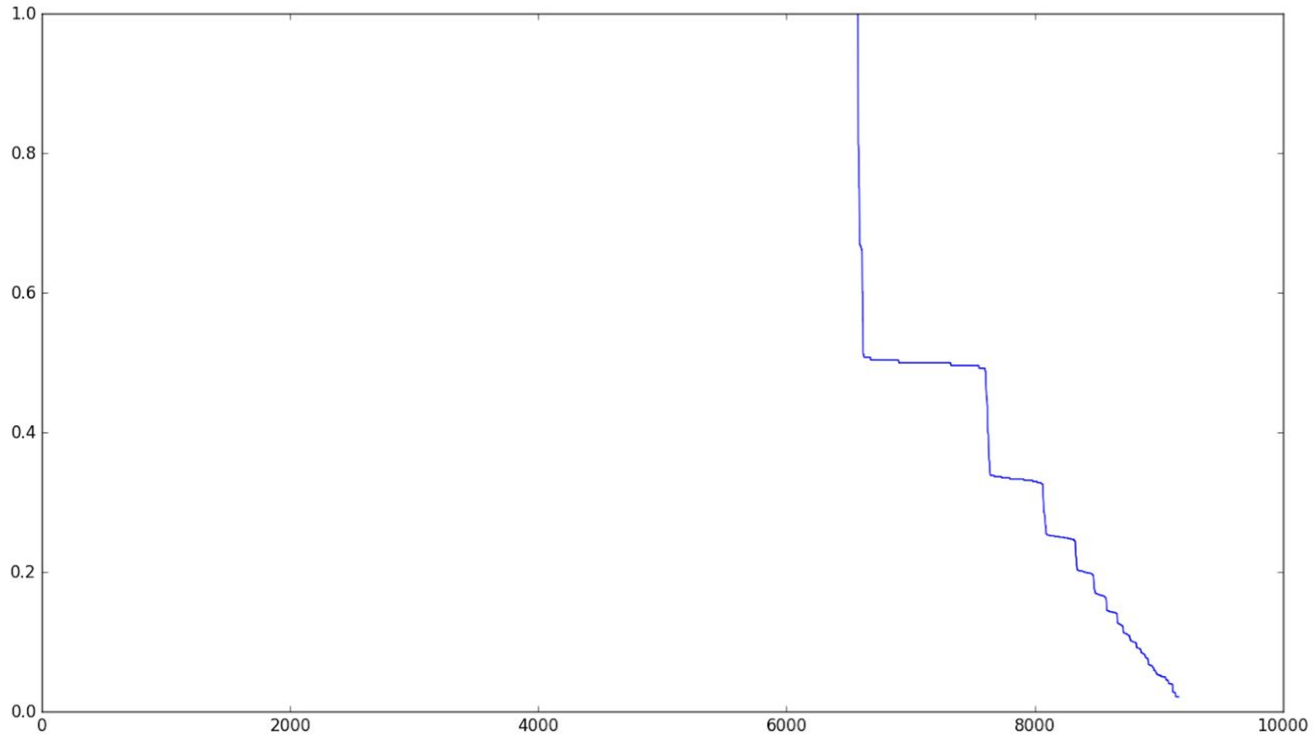
normal_trigrams_zipf_anime



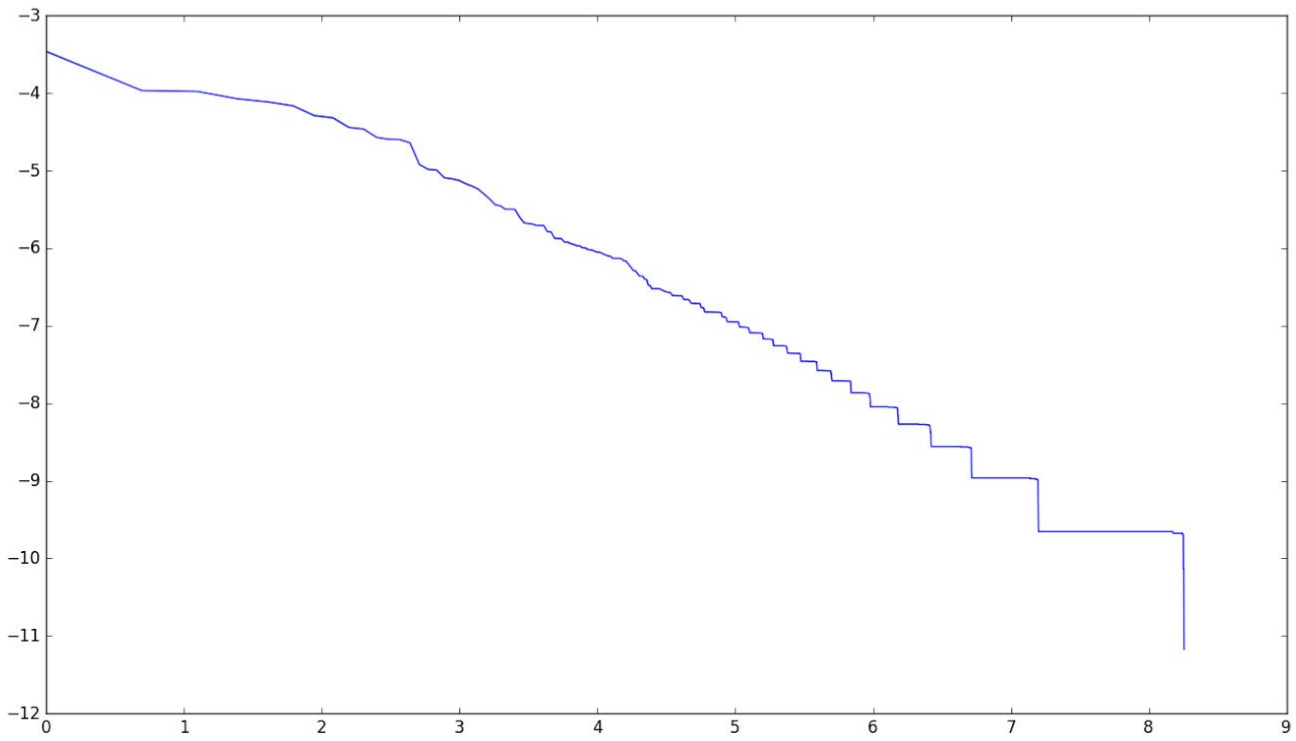
normal_trigrams_zipf_movies



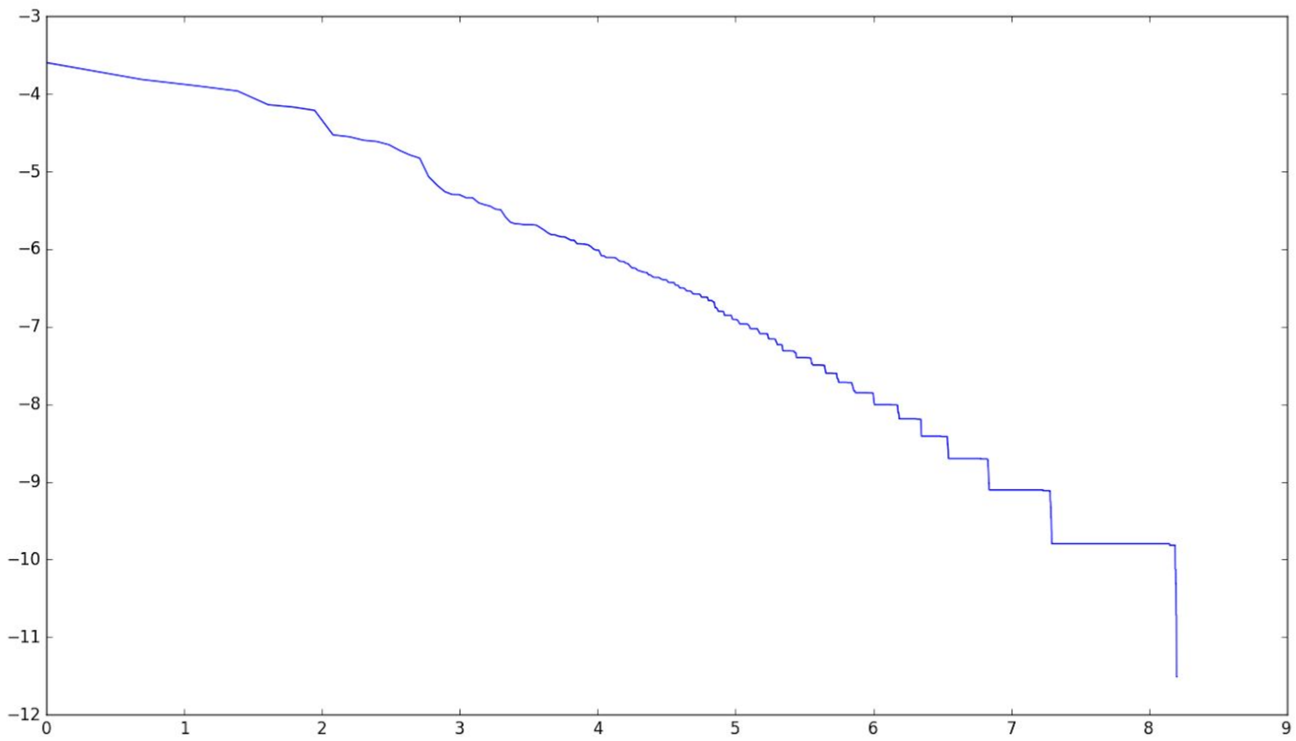
normal_trigrams_zipf_news



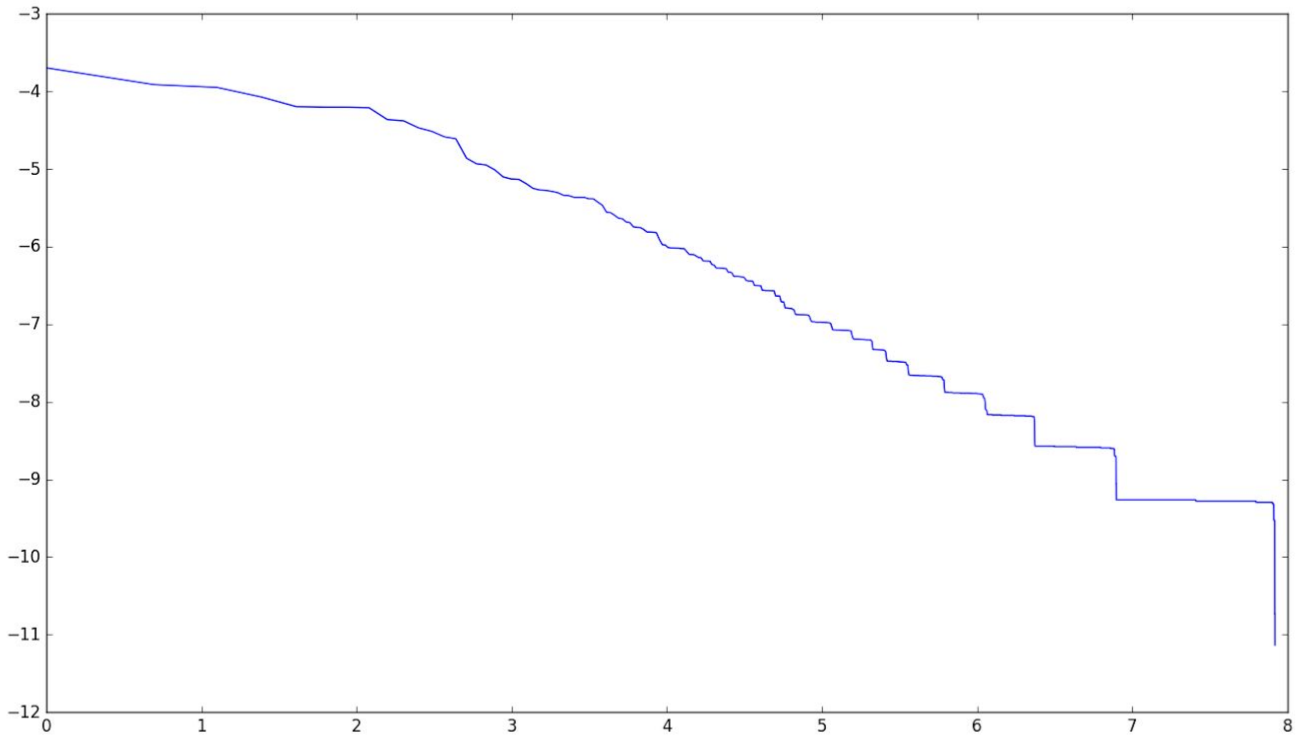
normal_unigram_log_anime



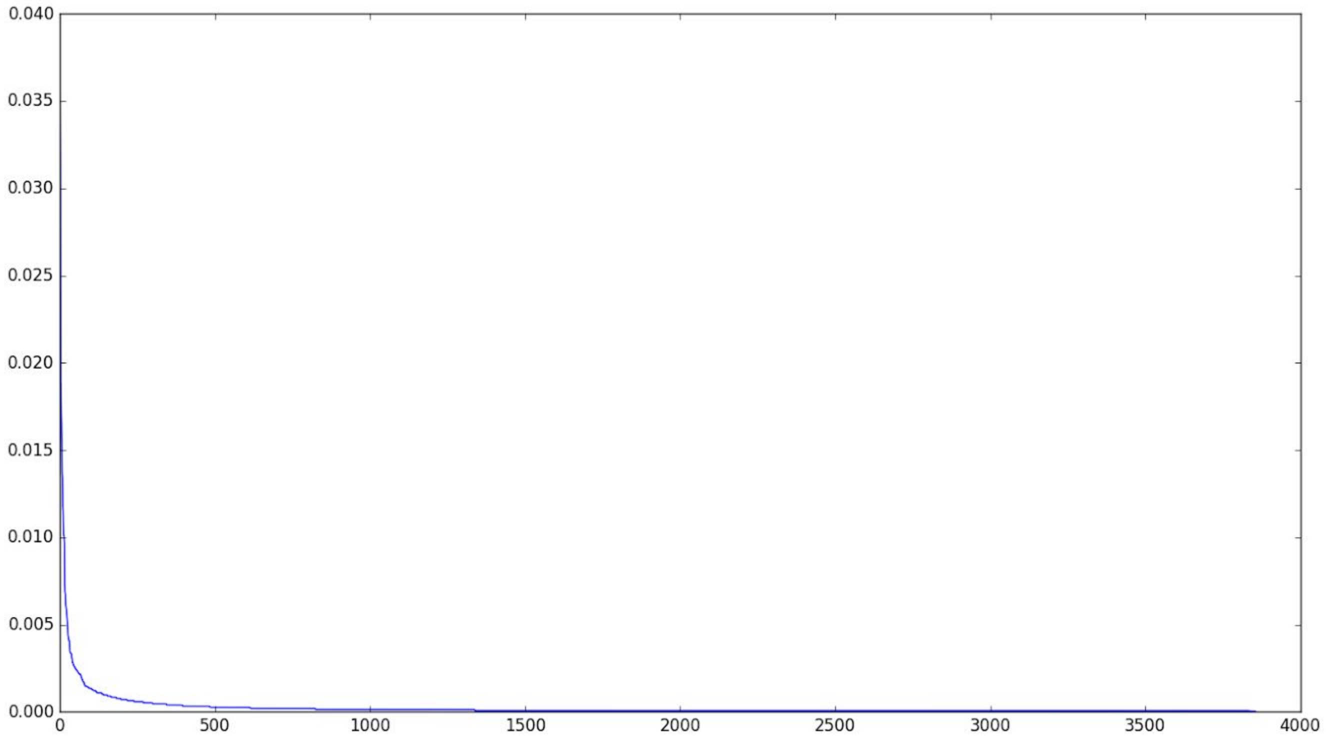
normal_unigram_log_movies



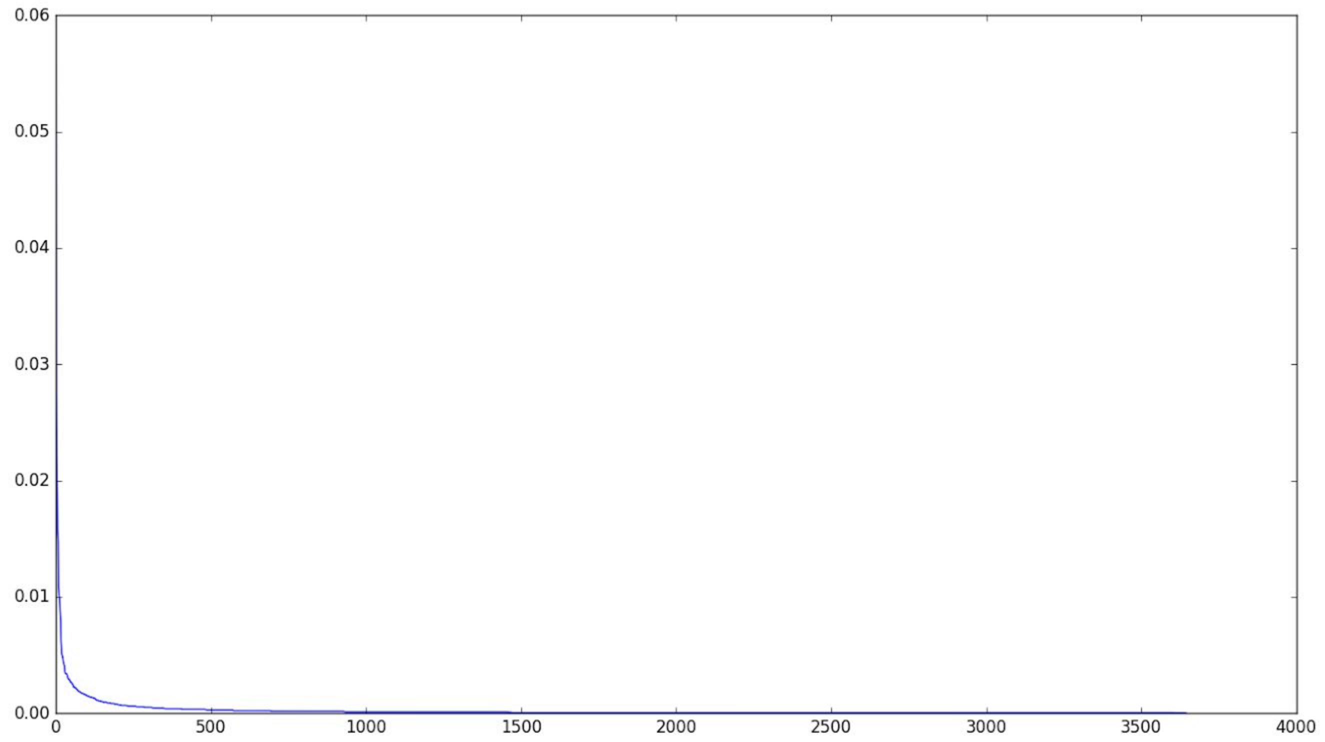
normal_unigram_log_news



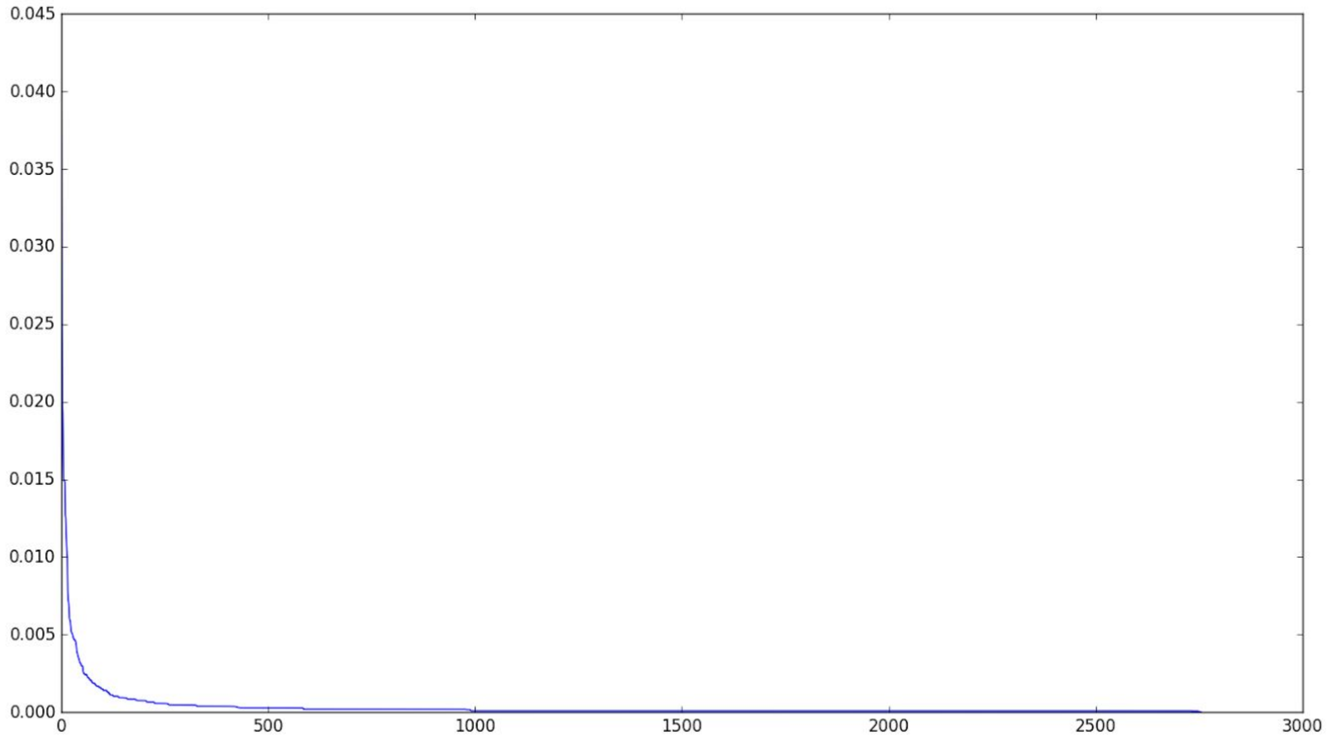
normal_unigram_zipf_anime



normal_unigram_zipf_movies

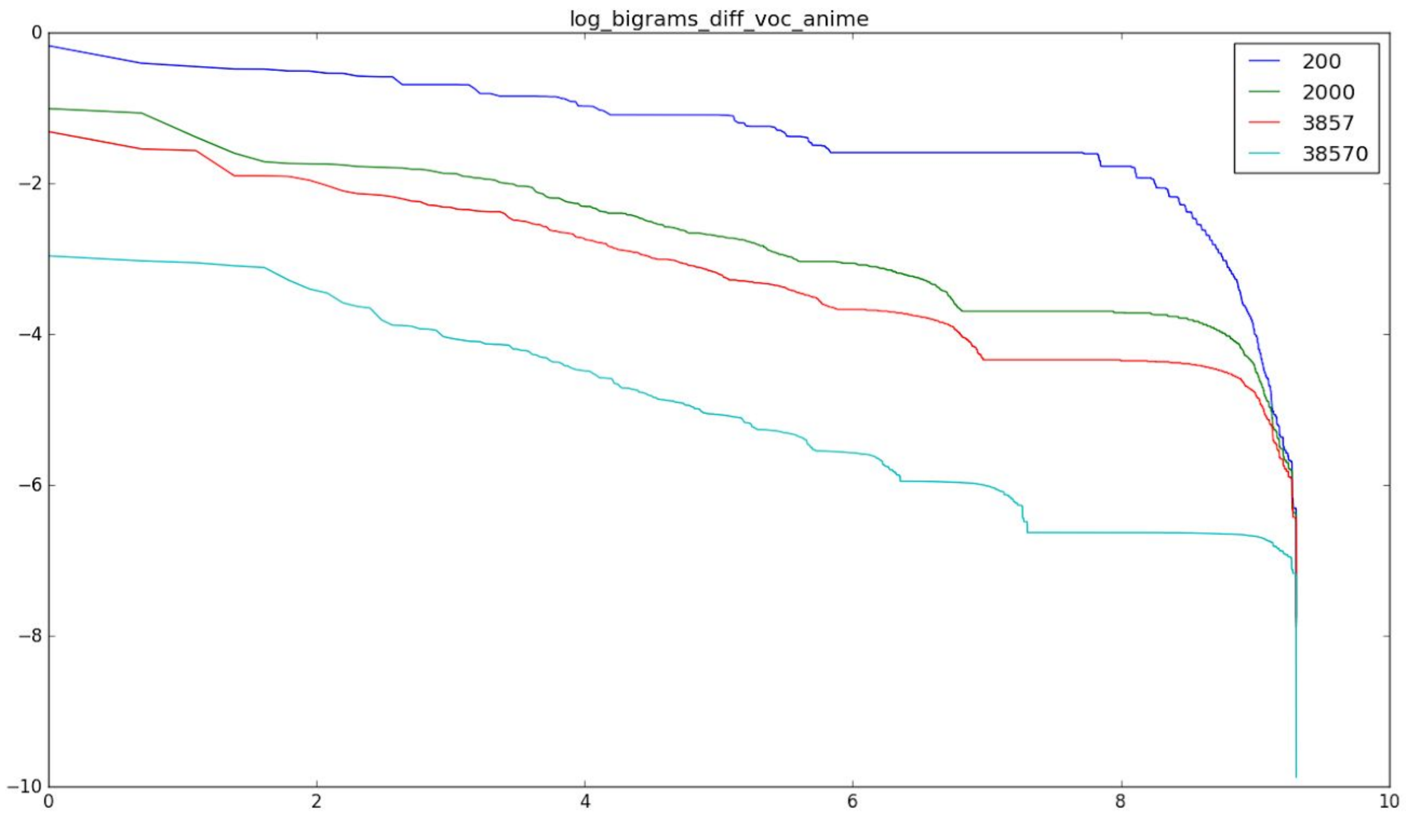


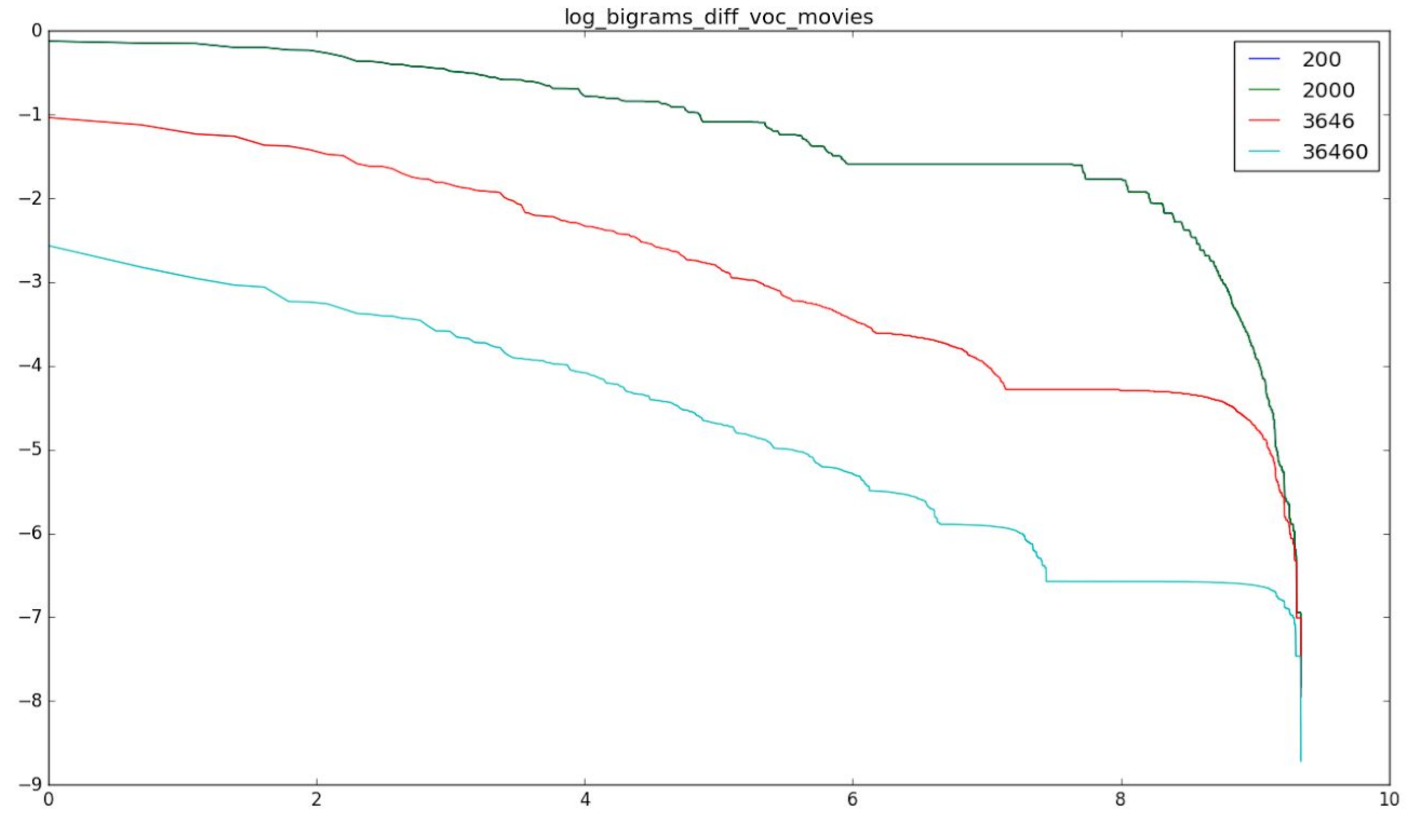
normal_unigram_zipf_news

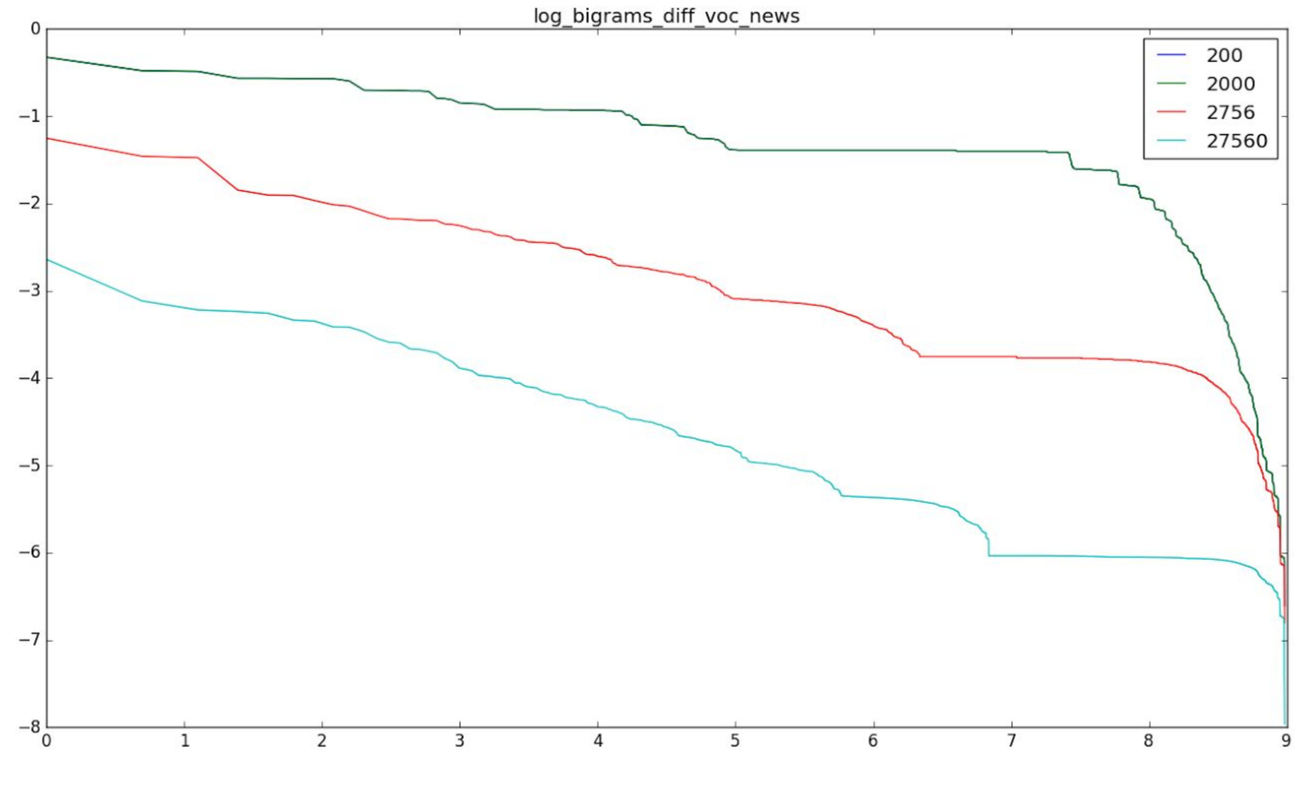


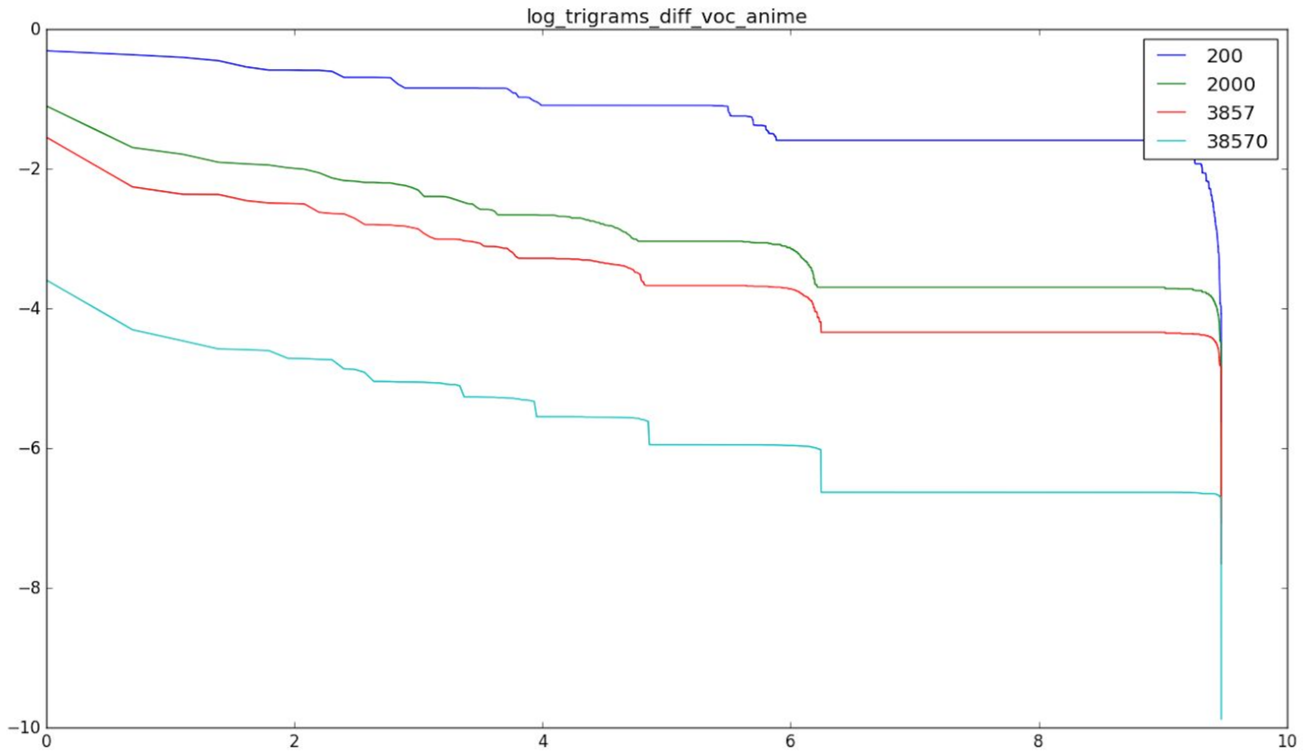
3.Implement laplace smoothing. Compare the effect of smoothing on different values for V (200, 2000, current size of vocabulary, 10*size of vocabulary). Plot these to compare.

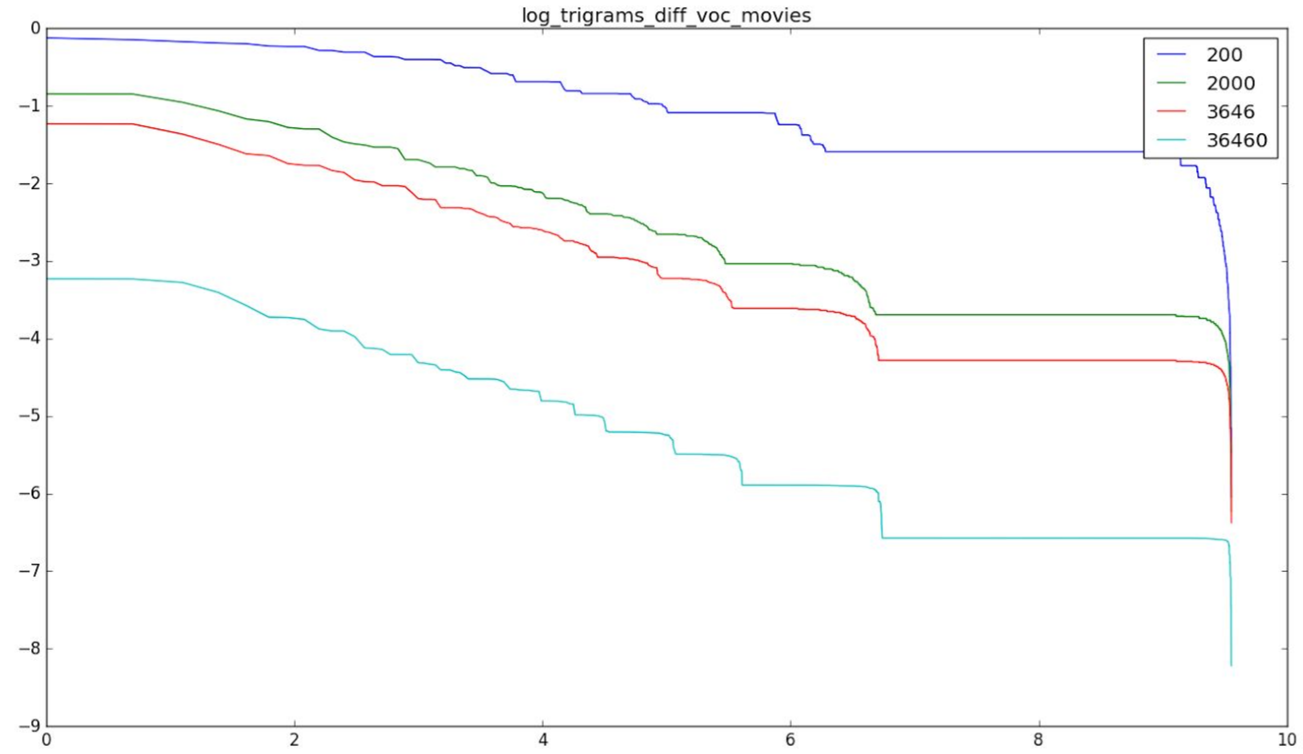
```
laplace_unigrams_prob = get_laplace_unigrams(unigrams,200)
laplace_unigrams_prob2 = get_laplace_unigrams(unigrams,2000)
laplace_unigrams_prob3 = get_laplace_unigrams(unigrams,len(unigrams))

laplace_bigrams_prob1 = get_laplace_bigrams(unigrams,bigrams,200)
laplace_bigrams_prob2 = get_laplace_bigrams(unigrams,bigrams,2000)
laplace_bigrams_prob3 = get_laplace_bigrams(unigrams,bigrams,len(unigrams))
laplace_bigrams_prob4 = get_laplace_bigrams(unigrams,bigrams,10*len(unigrams))

laplace_trigrams_prob1 = get_laplace_trigrams(unigrams,bigrams,trigrams,200)
laplace_trigrams_prob2 = get_laplace_trigrams(unigrams,bigrams,trigrams,2000)
laplace_trigrams_prob3 = get_laplace_trigrams(unigrams,bigrams,trigrams,len(unigrams))
laplace_trigrams_prob4 = get_laplace_trigrams(unigrams,bigrams,trigrams,10*len(unigrams))
```

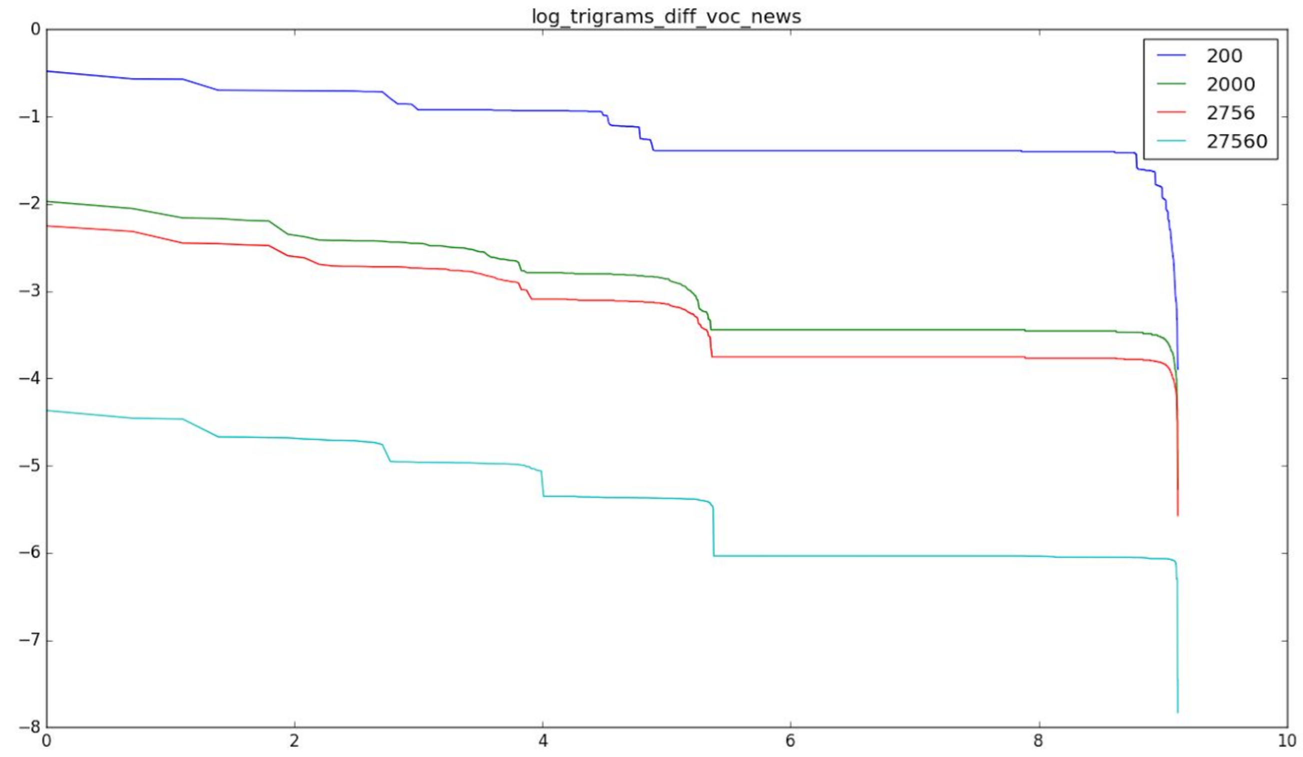

log_bigrams_diff_voc_anime
200
2000
3857
38570
0
−2
−4
−6
−8
−10
0
2
4
6
8
10

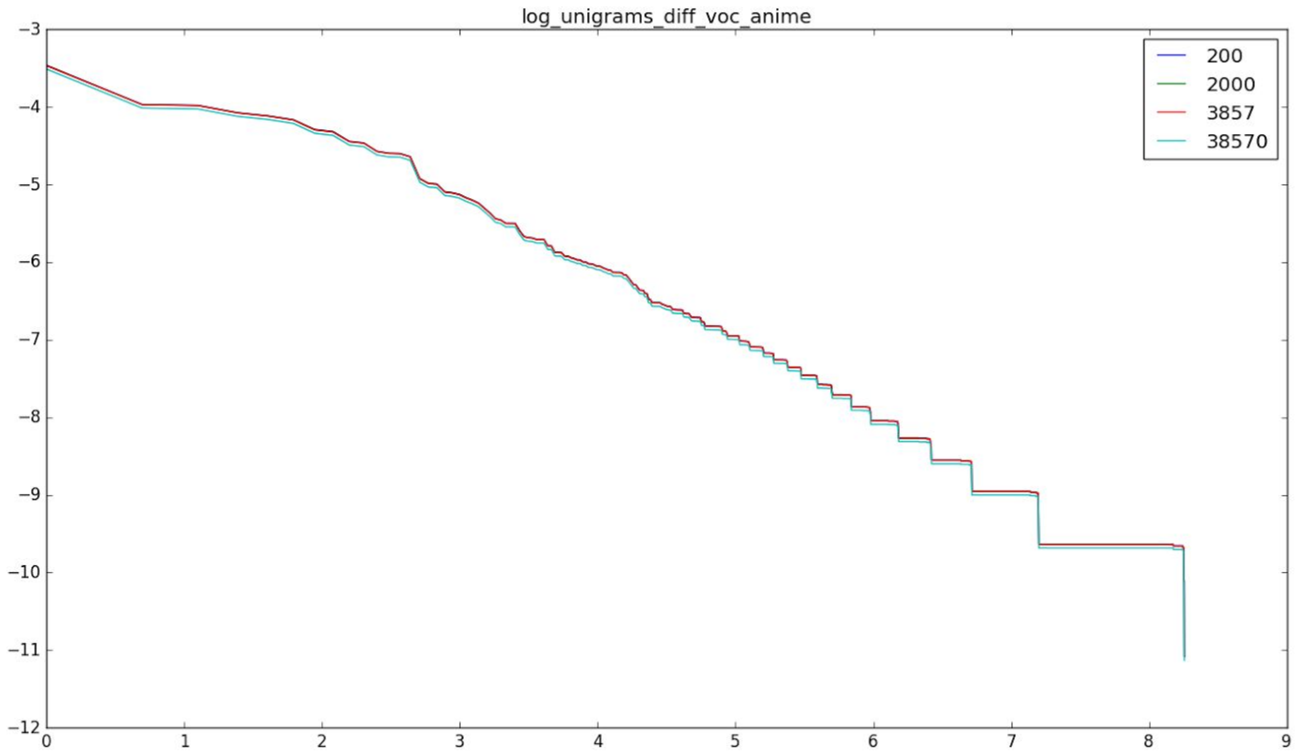

log_bigrams_diff_voc_movies
200
2000
3646
36460
0
−1
−2
−3
−4
−5
−6
−7
−8
−9
0
2
4
6
8
10

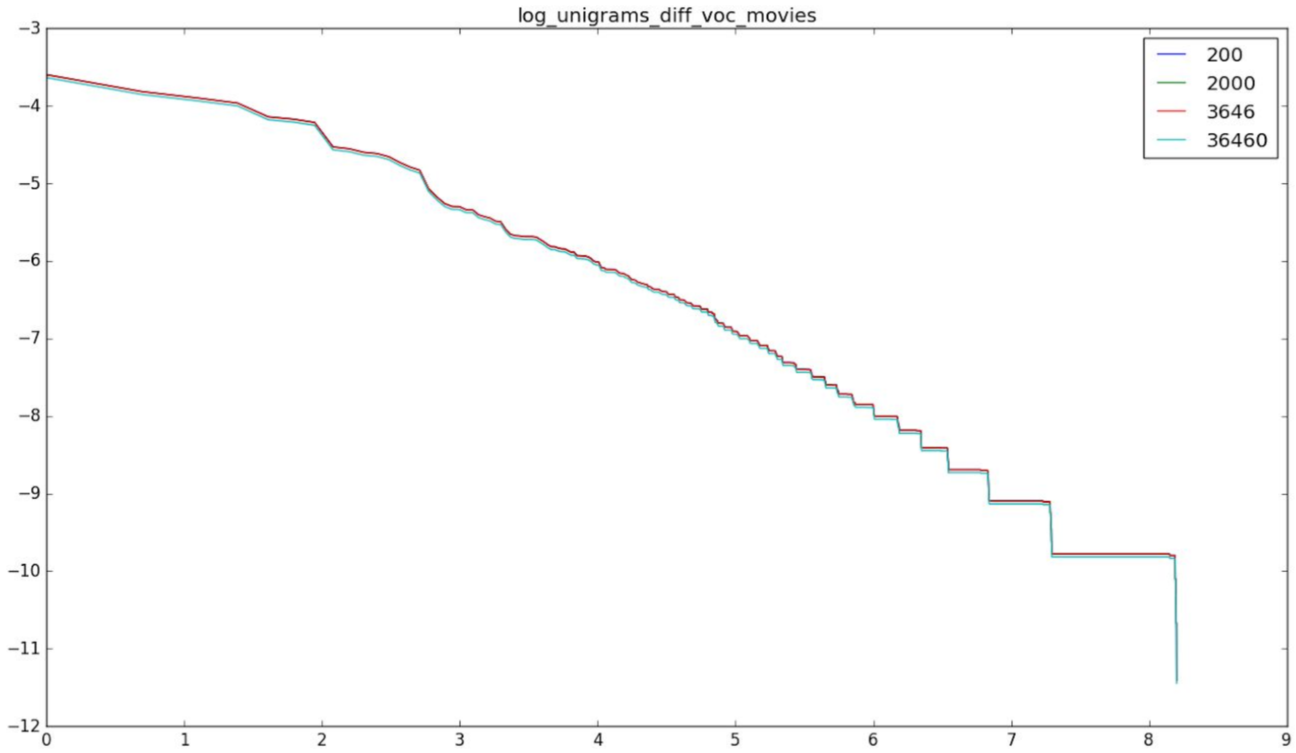

log_bigrams_diff_voc_news
200
2000
2756
27560
0
−1
−2
−3
−4
−5
−6
−7
−8
0
1
2
3
4
5
6
7
8
9

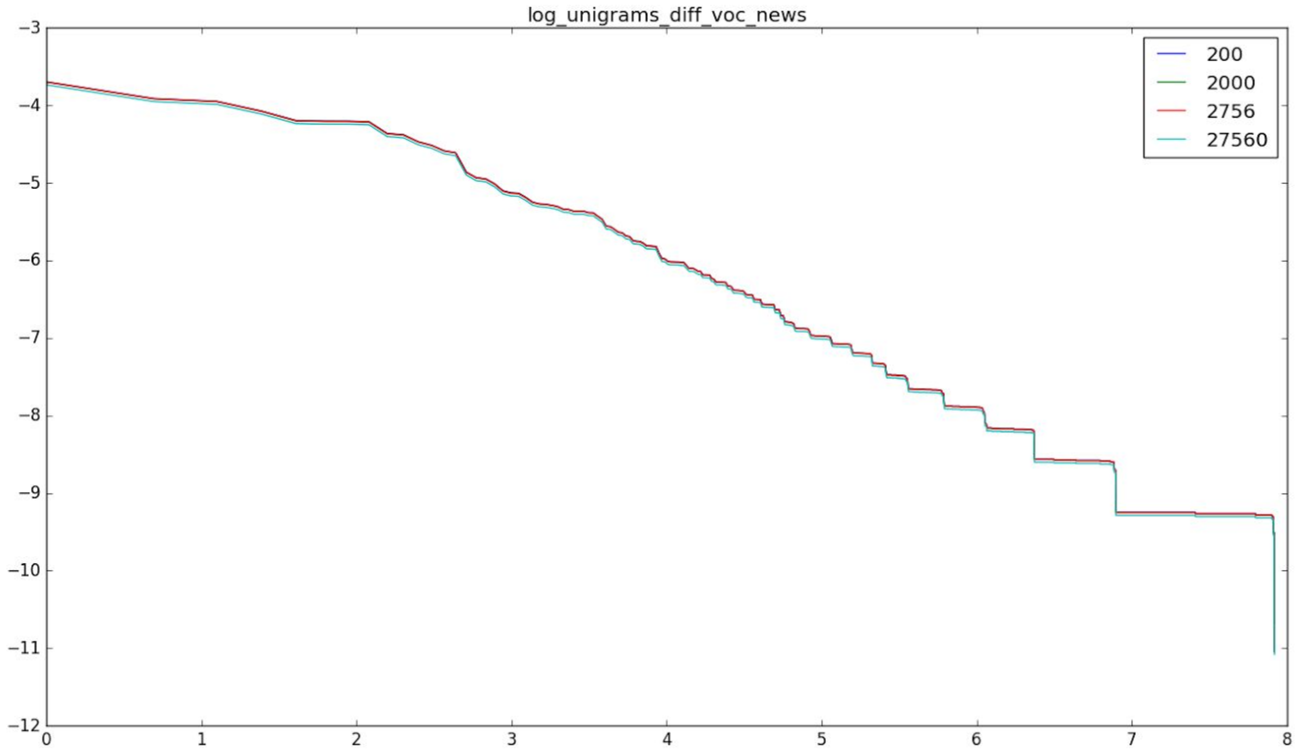

log_trigrams_diff_voc_anime
200
2000
3857
38570
0
−2
−4
−6
−8
−10
0
2
4
6
8
10

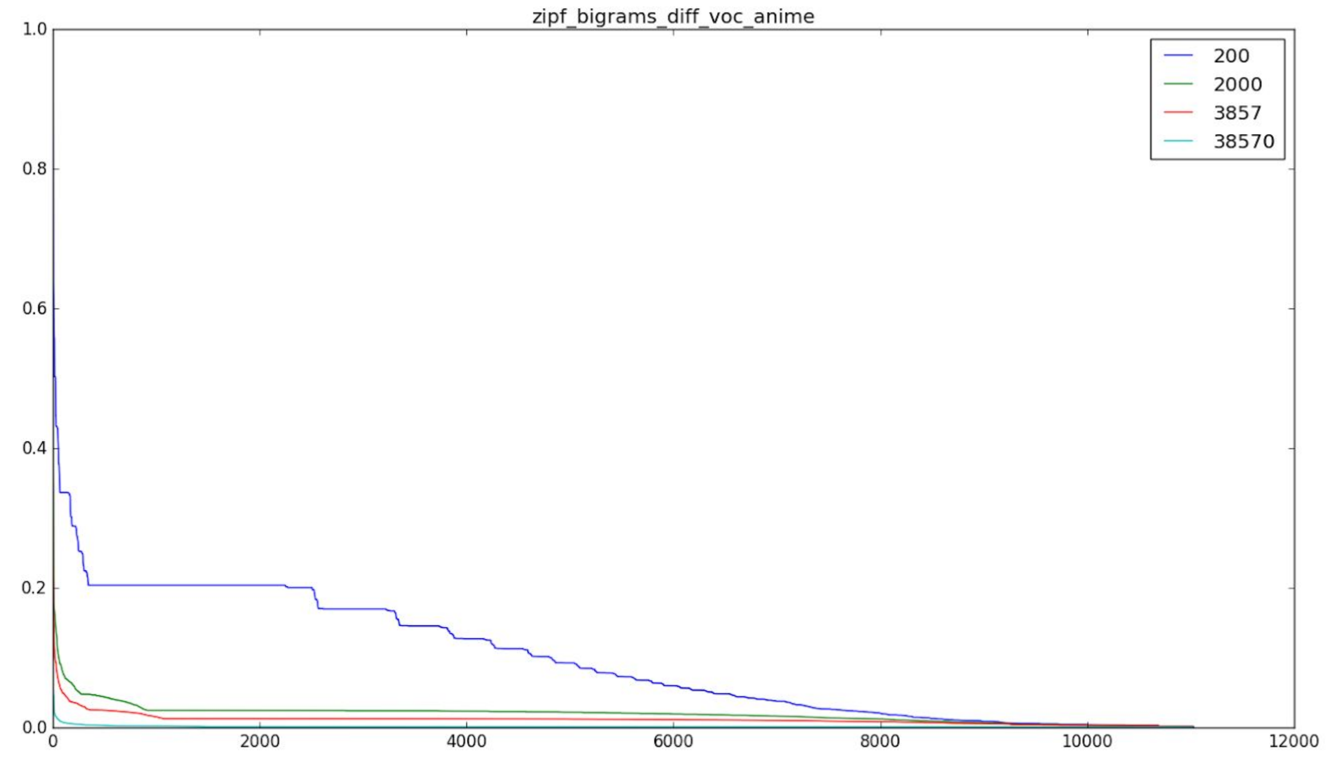

log_trigrams_diff_voc_movies
200
2000
3646
36460
0
−1
−2
−3
−4
−5
−6
−7
−8
−9
0
2
4
6
8
10

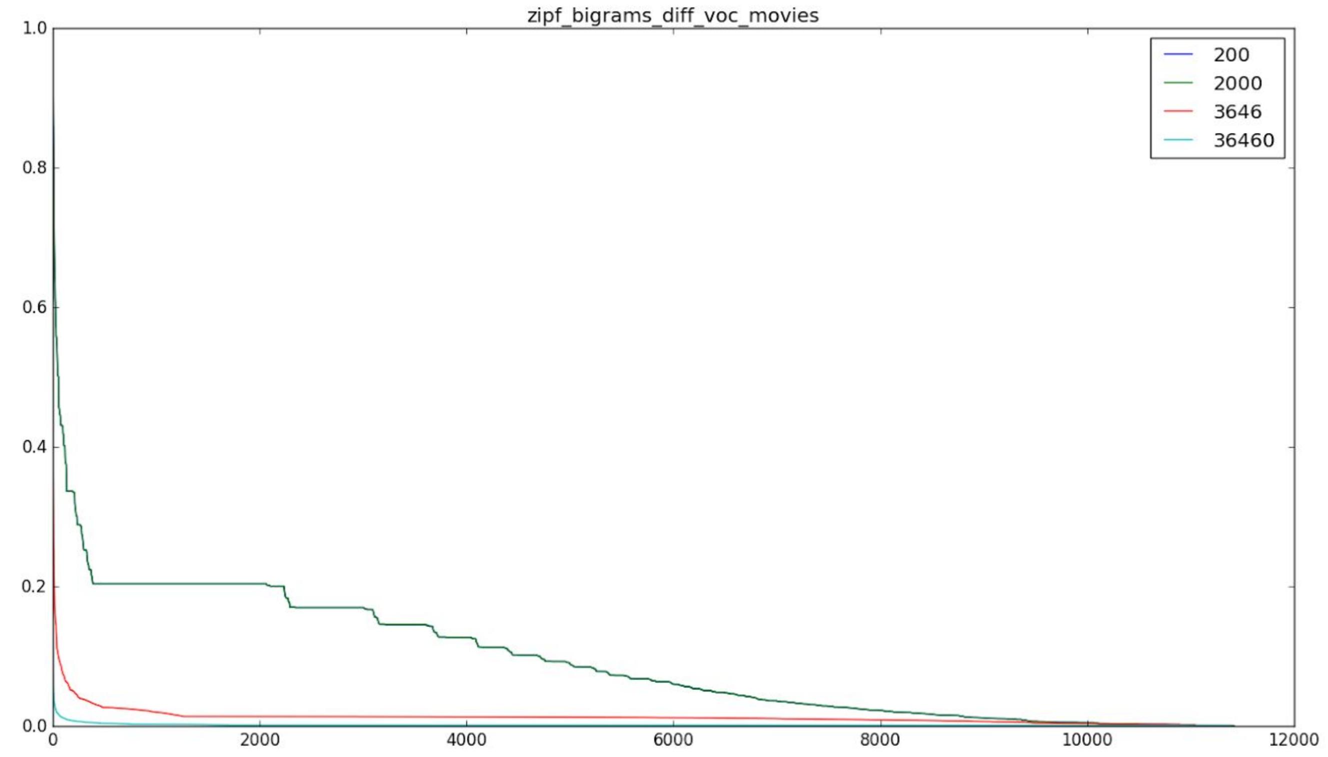

log_trigrams_diff_voc_news
200
2000
2756
27560
0
−1
−2
−3
−4
−5
−6
−7
−8
0
2
4
6
8
10

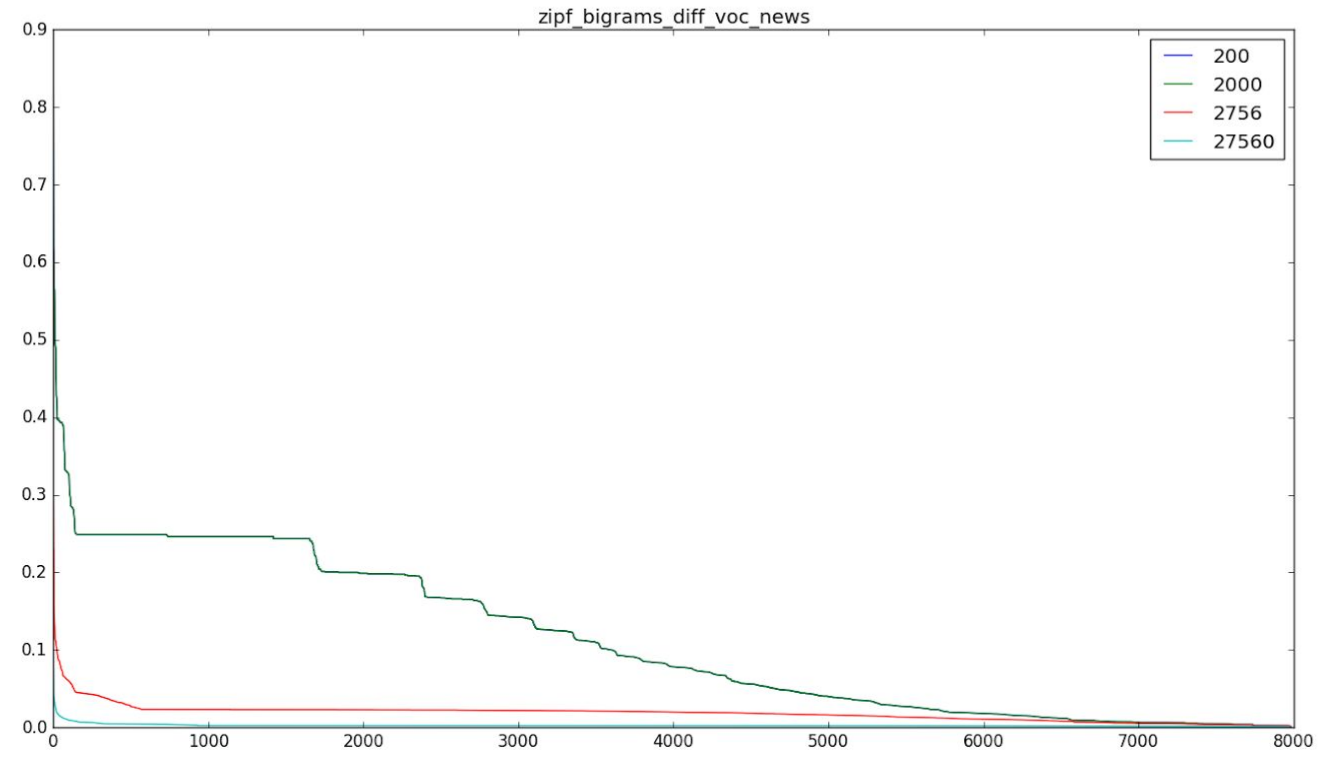

log_unigrams_diff_voc_anime
200
2000
3857
38570
−3
−4
−5
−6
−7
−8
−9
−10
−11
−12
0
1
2
3
4
5
6
7
8
9

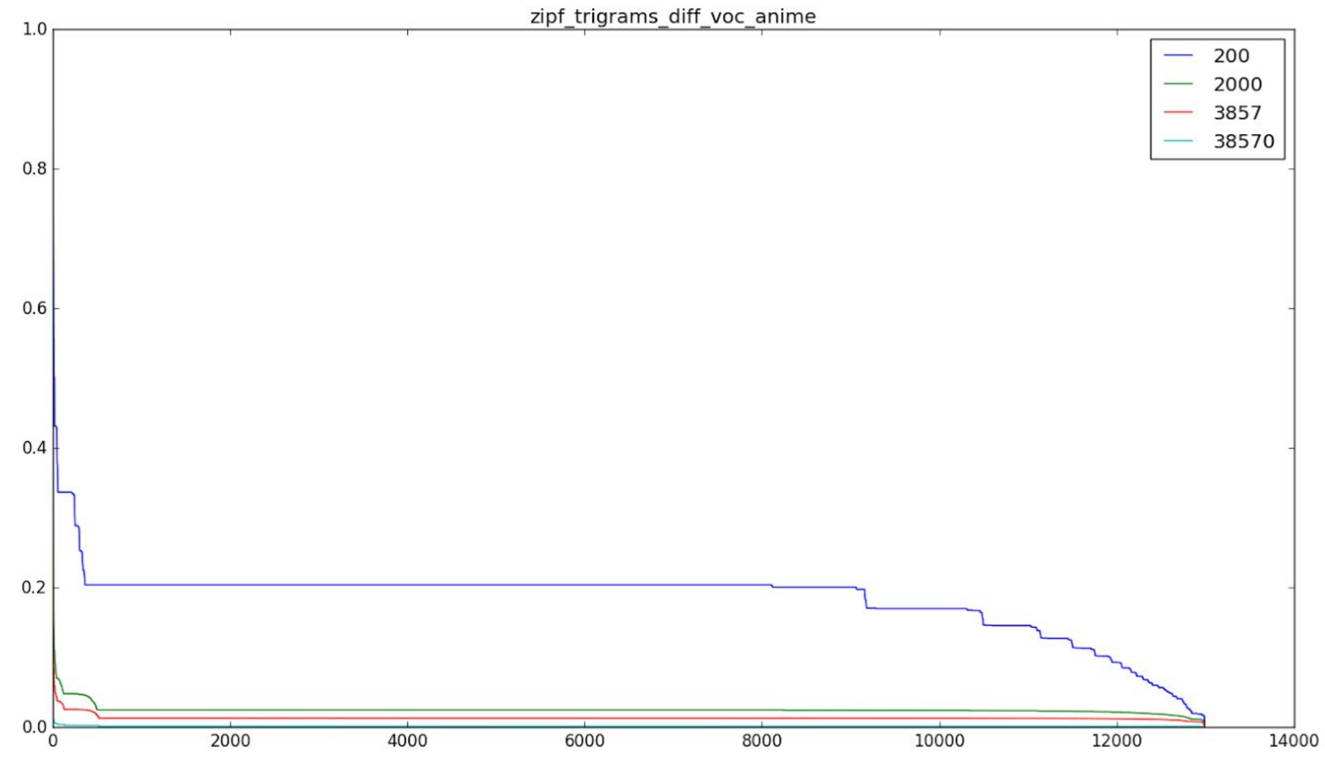

log_unigrams_diff_voc_movies
200
2000
3646
36460
−3
−4
−5
−6
−7
−8
−9
−10
−11
−12
0
1
2
3
4
5
6
7
8
9

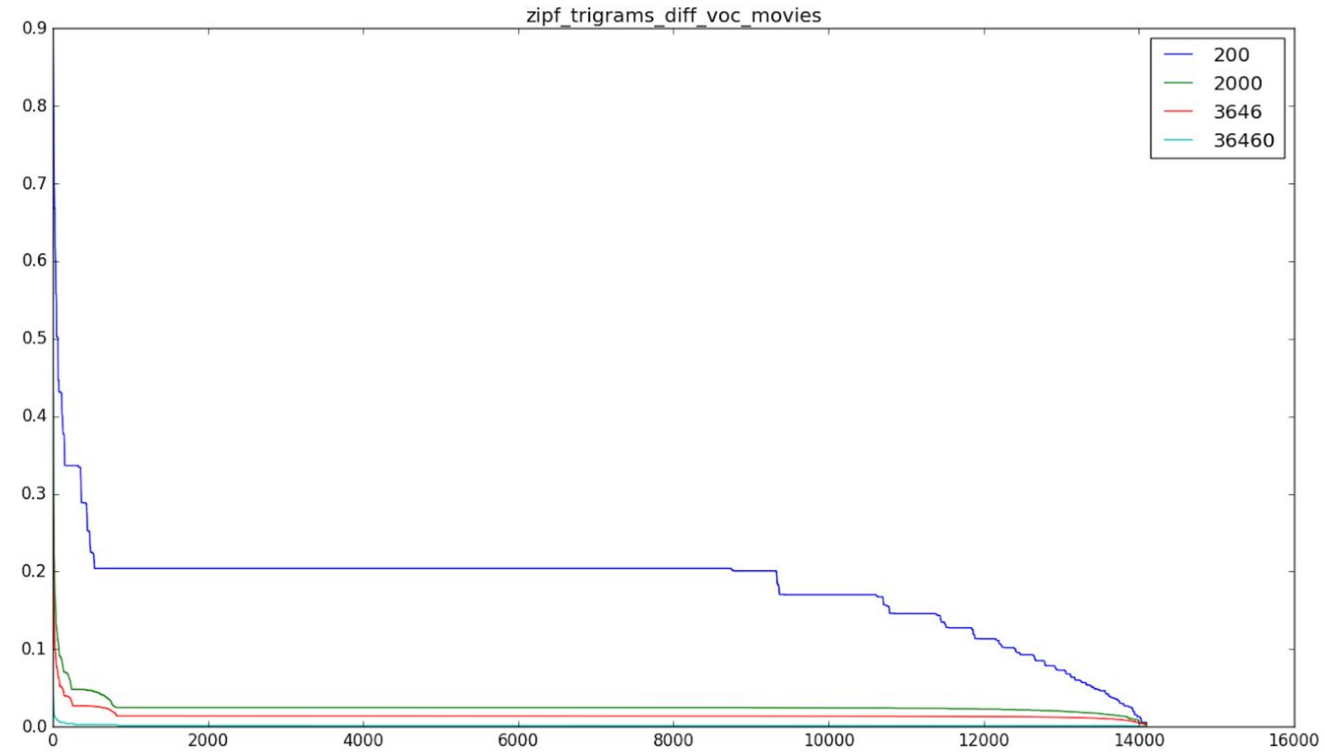

log_unigrams_diff_voc_news
200
2000
2756
27560
−3
−4
−5
−6
−7
−8
−9
−10
−11
−12
0
1
2
3
4
5
6
7
8

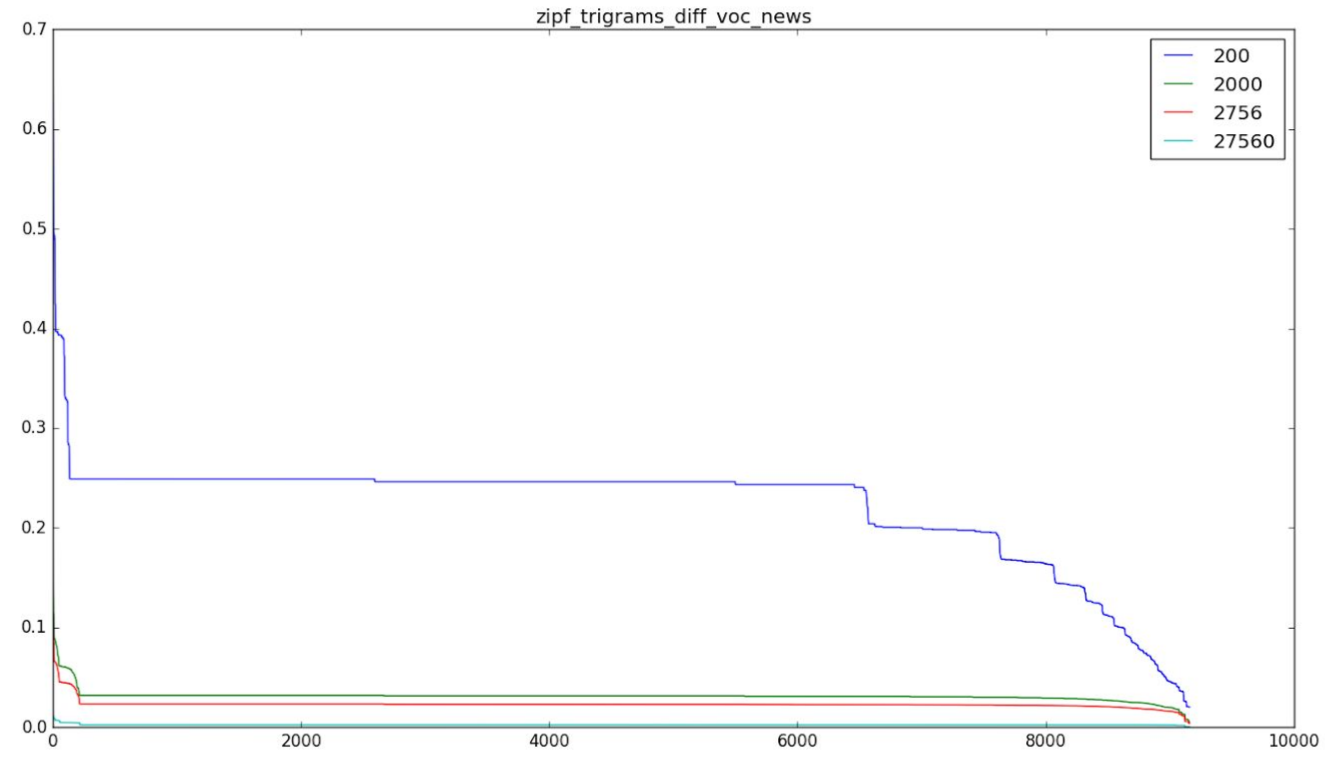

zipf_bigrams_diff_voc_anime
200
2000
3857
38570
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
12000

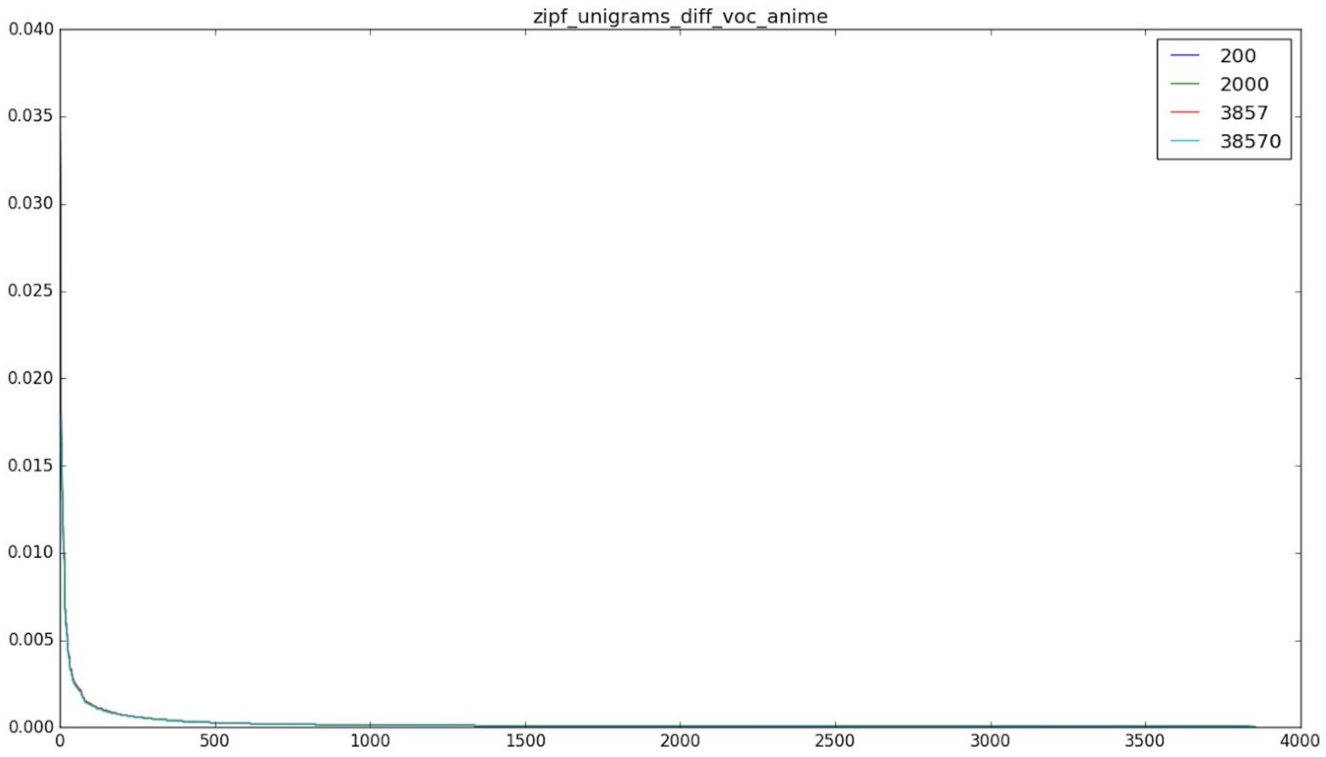

zipf_bigrams_diff_voc_movies
200
2000
3646
36460
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
12000

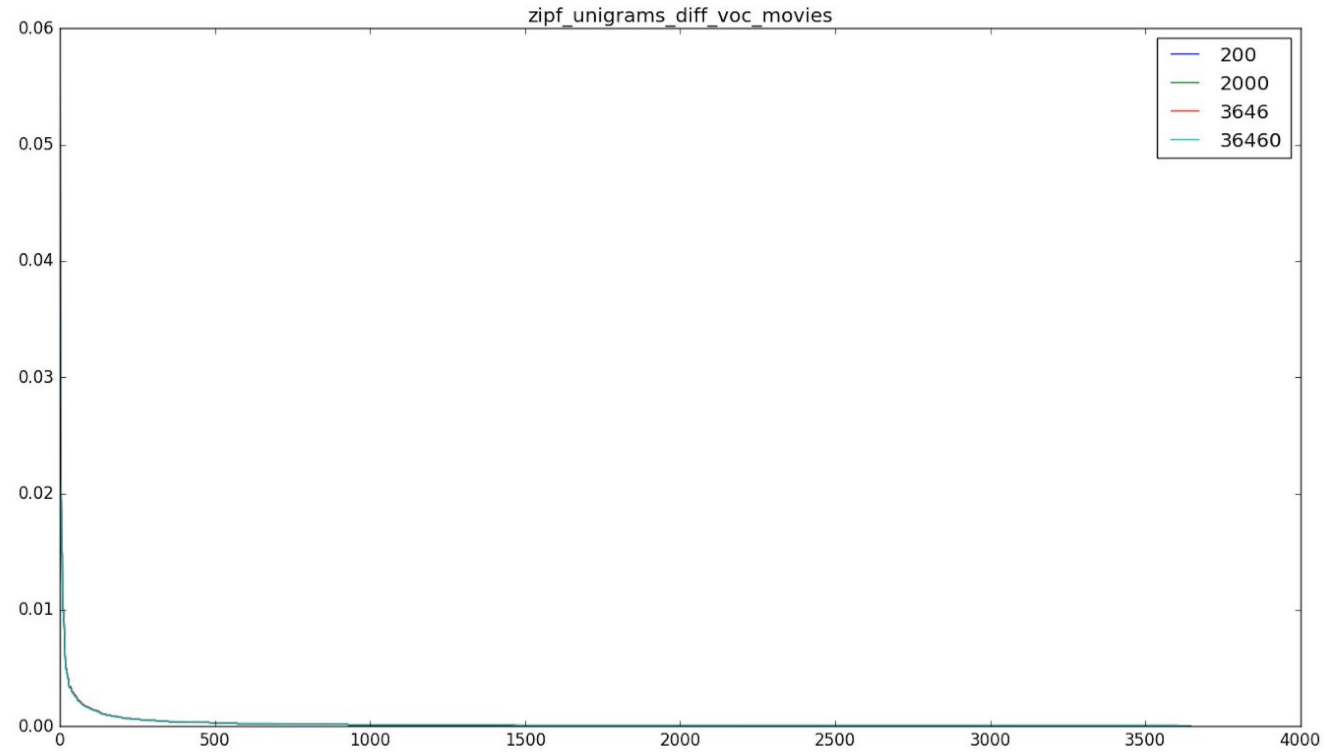

zipf_bigrams_diff_voc_news
200
2000
2756
27560
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1000
2000
3000
4000
5000
6000
7000
8000

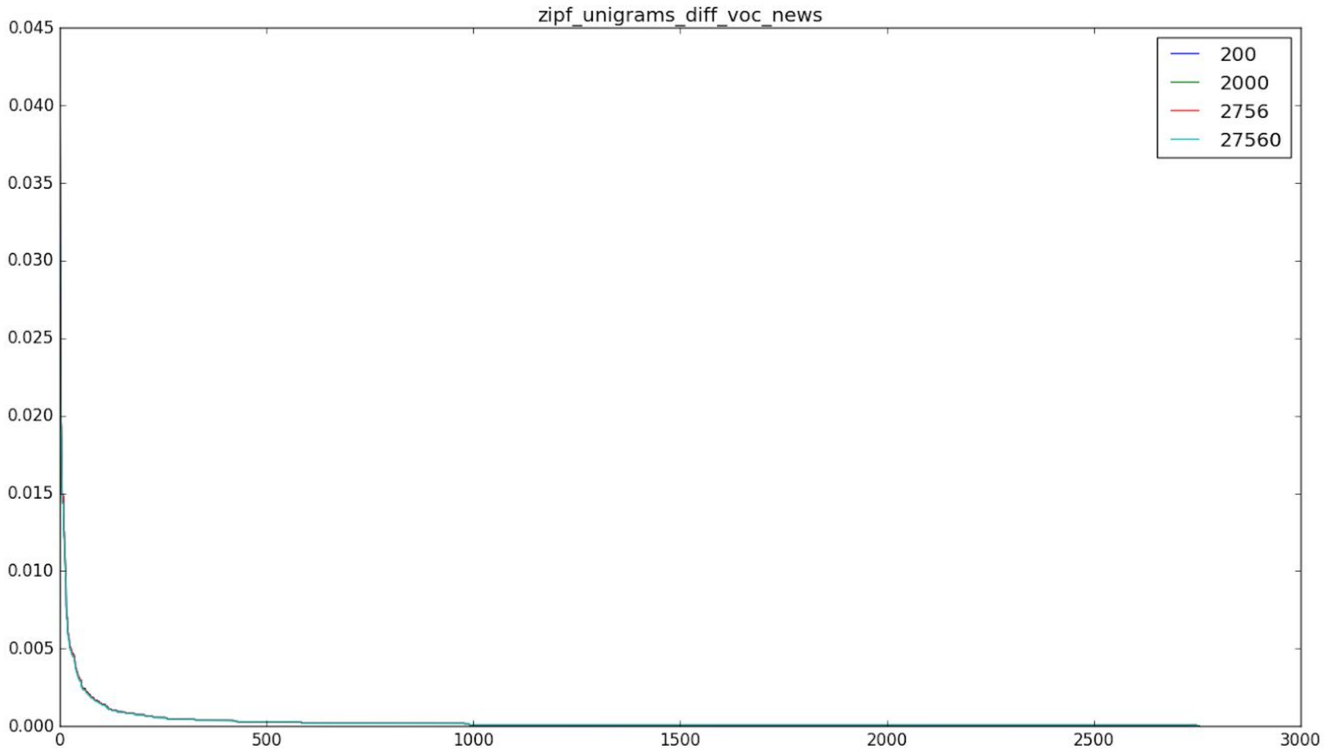

zipf_trigrams_diff_voc_anime
200
2000
3857
38570
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
12000
14000


zipf_trigrams_diff_voc_movies
200
2000
3646
36460
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2000
4000
6000
8000
10000
12000
14000
16000


zipf_trigrams_diff_voc_news
200
2000
2756
27560
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2000
4000
6000
8000
10000







4.Implement Witten-Bell backoff.

Below shows Witten-Bell backoff for bigrams:

```
def get_wittenbell_bigrams_variables(bigrams,word):
    printvalues(bigrams)
    distinct_1_in_1_bigrams=0
    total_1_in_1_bigrams=0
    printvalues(bigrams)
    for pair in bigrams:
        if word==pair[0]:
            printvalues(pair)
            distinct_1_in_1_bigrams+=1
            total_1_in_1_bigrams+=bigrams[pair]
            printvalues(bigrams[pair])
    return distinct_1_in_1_bigrams,total_1_in_1_bigrams

def get_wittenbell_bigrams(unigrams,bigrams,unigrams_prob,wittenbell_unigrams_prob):
    printvalues(unigrams)
    wittenbell_bigrams_prob = {}
    for pair in bigrams:
        printvalues(pair)
        distinct_1_in_1_bigrams,total_1_in_1_bigrams = get_wittenbell_bigrams_variables(bigrams,pair[0])
        x=distinct_1_in_1_bigrams/float(distinct_1_in_1_bigrams+total_1_in_1_bigrams)
        x=round(x, 15)
        wittenbell_bigrams_prob[pair] = (1-x) * bigrams_prob[pair]
        wittenbell_bigrams_prob[pair] += x * wittenbell_unigrams_prob[pair[0]]
    printvalues(sort_dict(wittenbell_bigrams_prob))
    return wittenbell_bigrams_prob
```

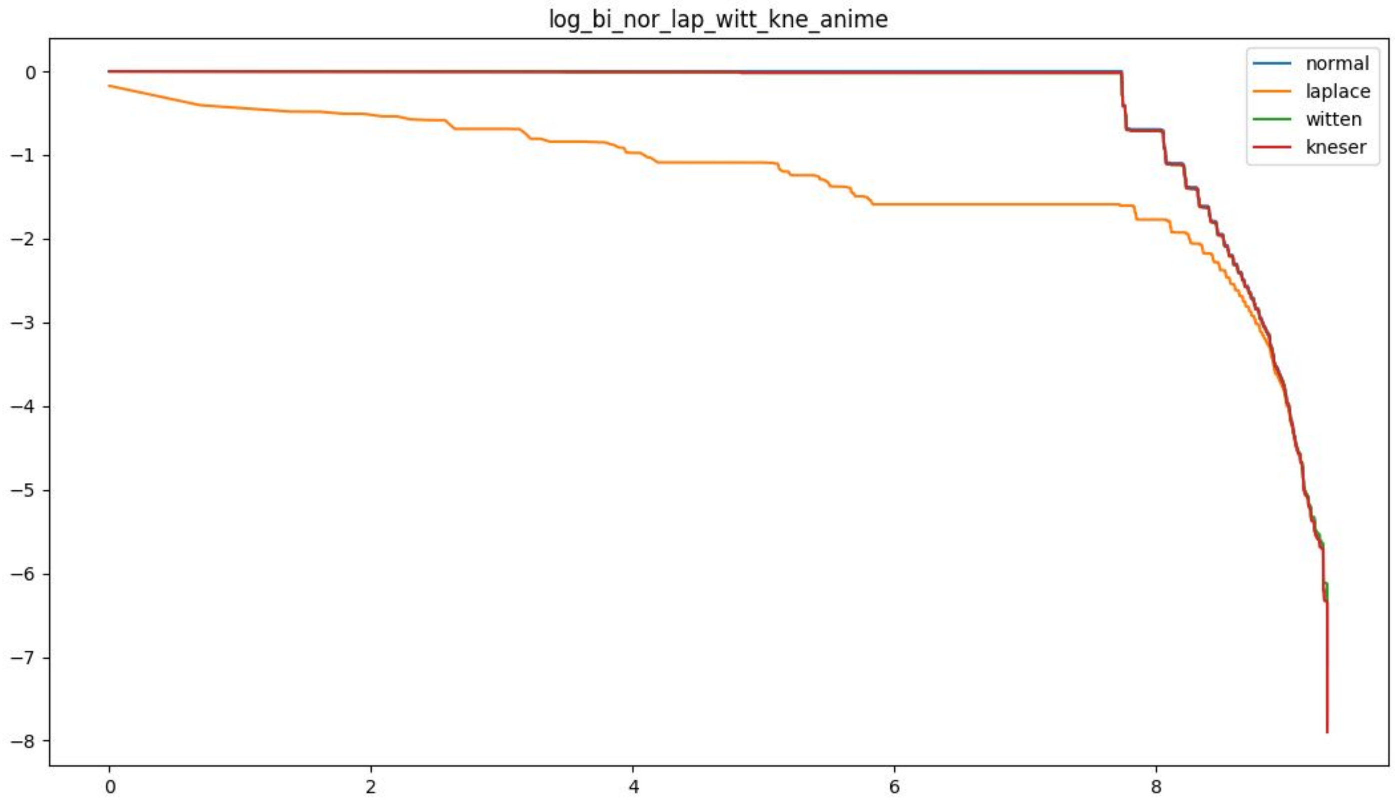
This functions could be called as shown below:

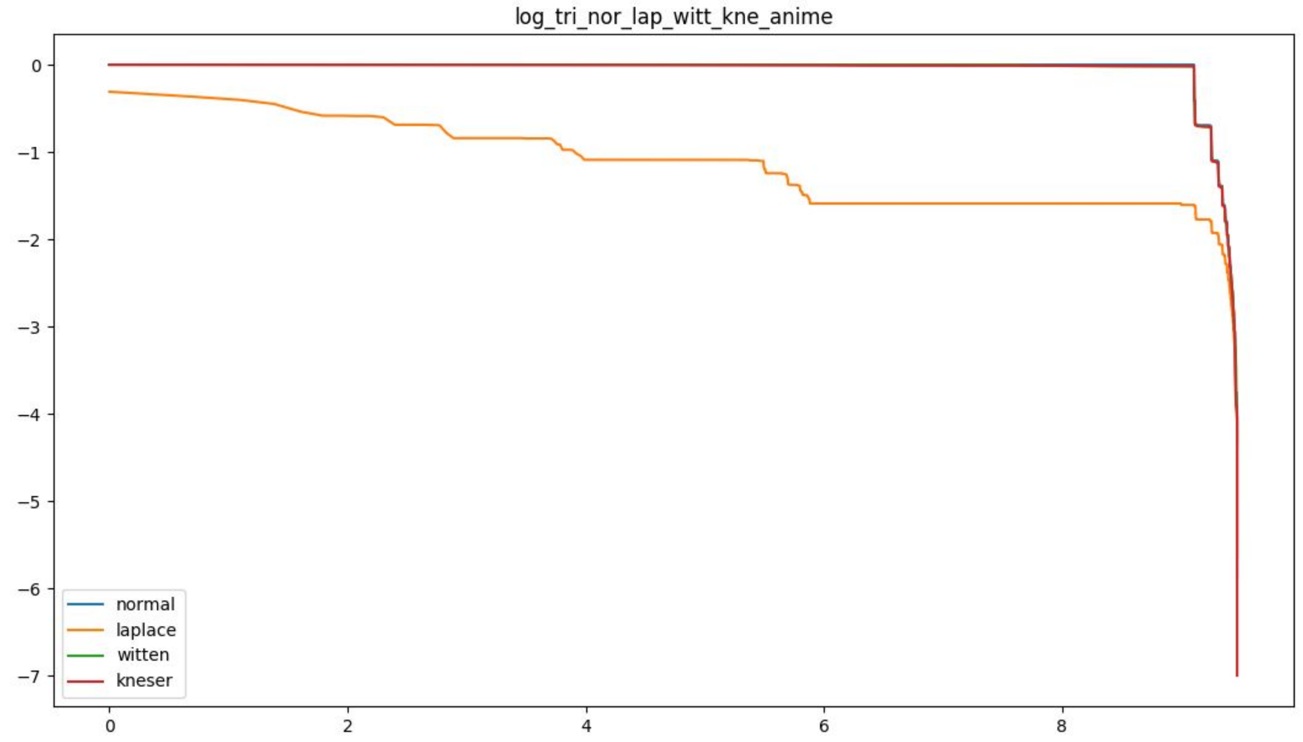
```
wittenbell_bigrams_prob = get_wittenbell_bigrams(unigrams,bigrams,unigrams_prob,wittenbell_unigrams_prob)
wittenbell_trigrams_prob = get_wittenbell_trigrams(unigrams,trigrams,trigrams_prob,wittenbell_bigrams_prob)
```

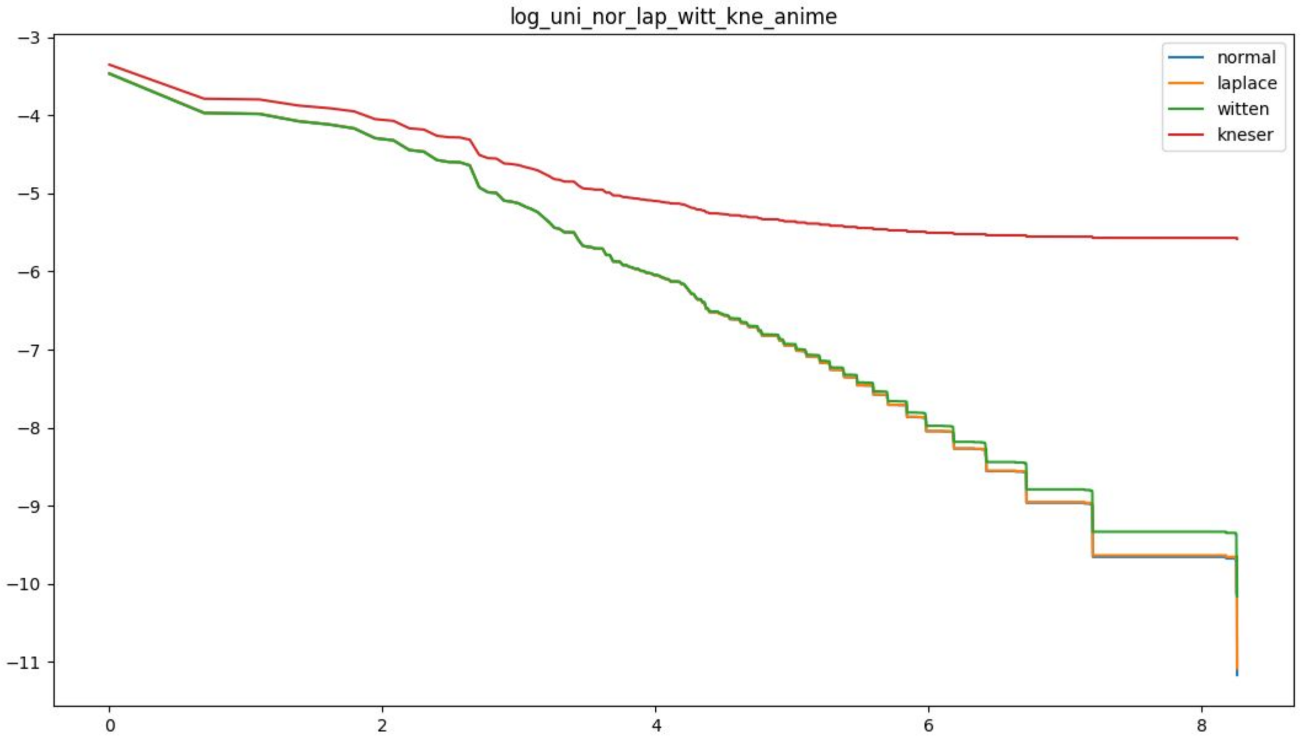
4.Implement Kneser-Ney smoothing.
Kneser-Ney smoothing is implemented which could be called using below functions

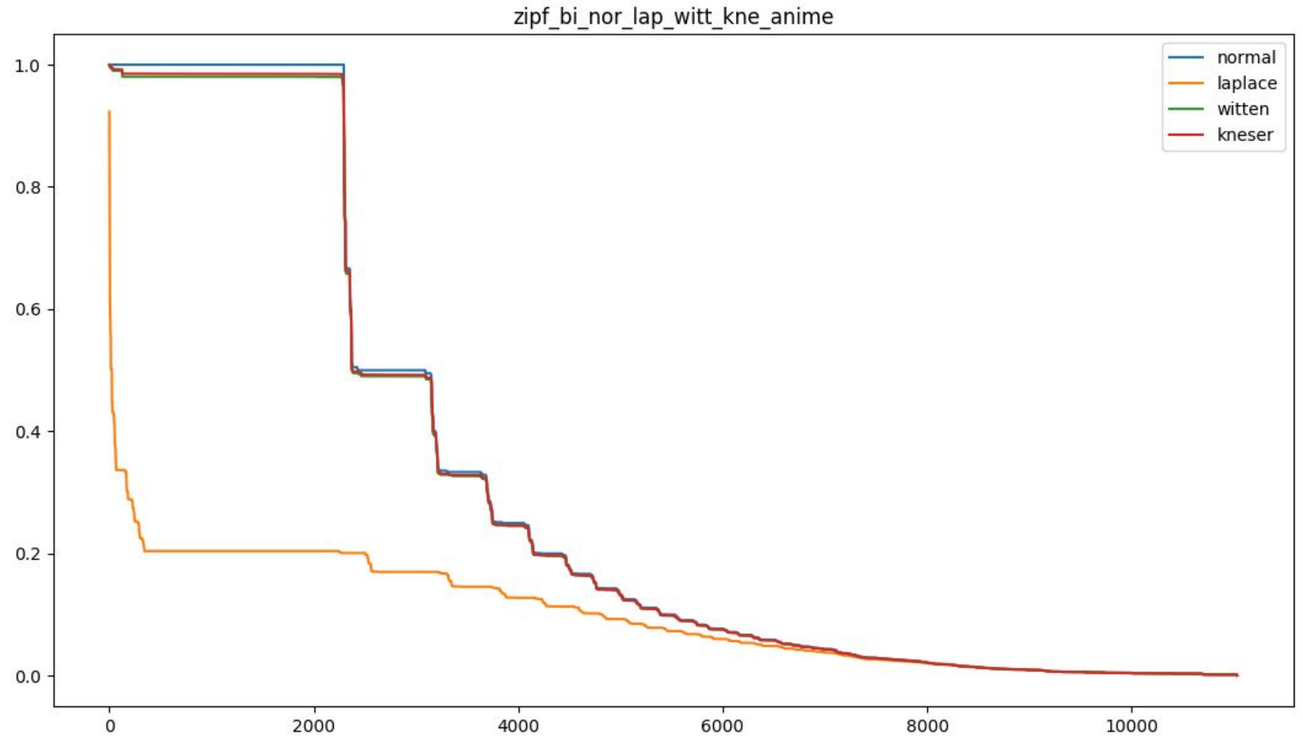
```
kn_unigrams_prob = get_kn_unigrams(unigrams,200)
kn_bigrams_prob =  get_kn_bigrams(unigrams,bigrams)
kn_trigrams_prob =  get_kn_trigrams(unigrams,bigrams,trigrams)
```

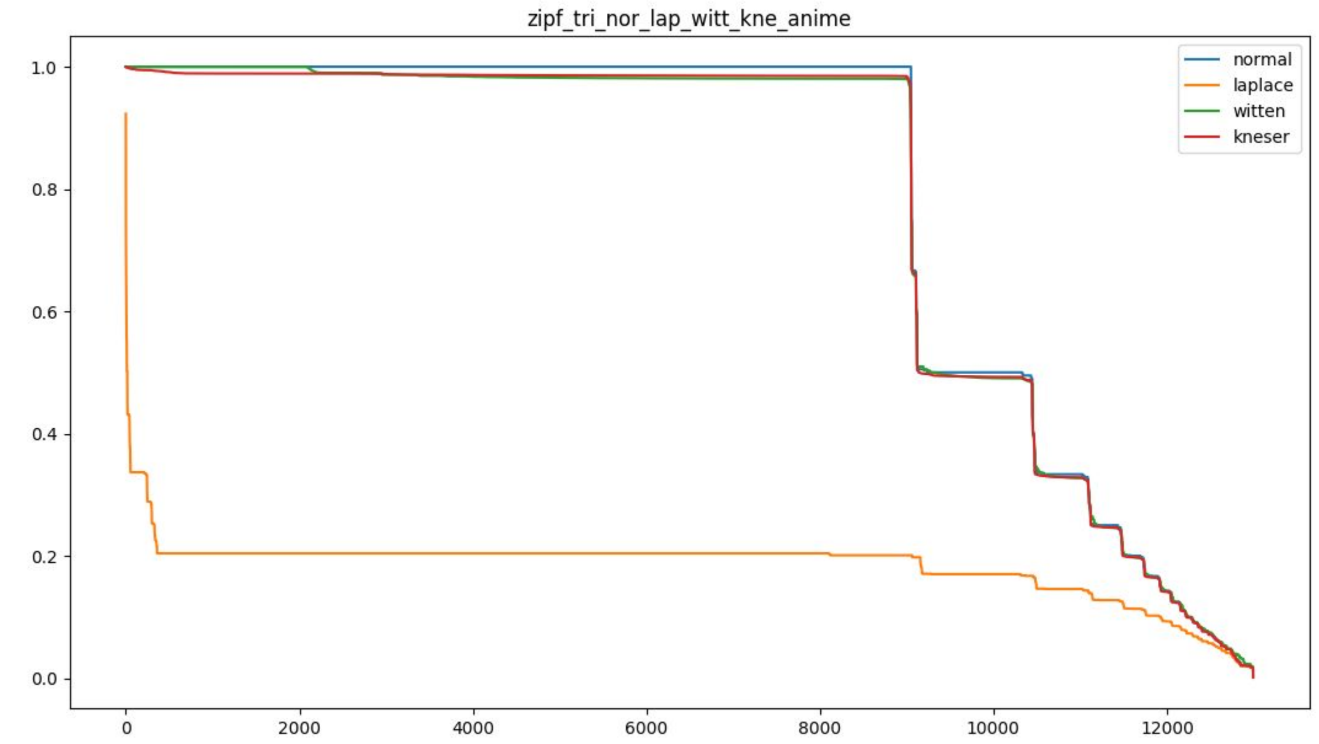
5. Comparision of three smoothing techniques:

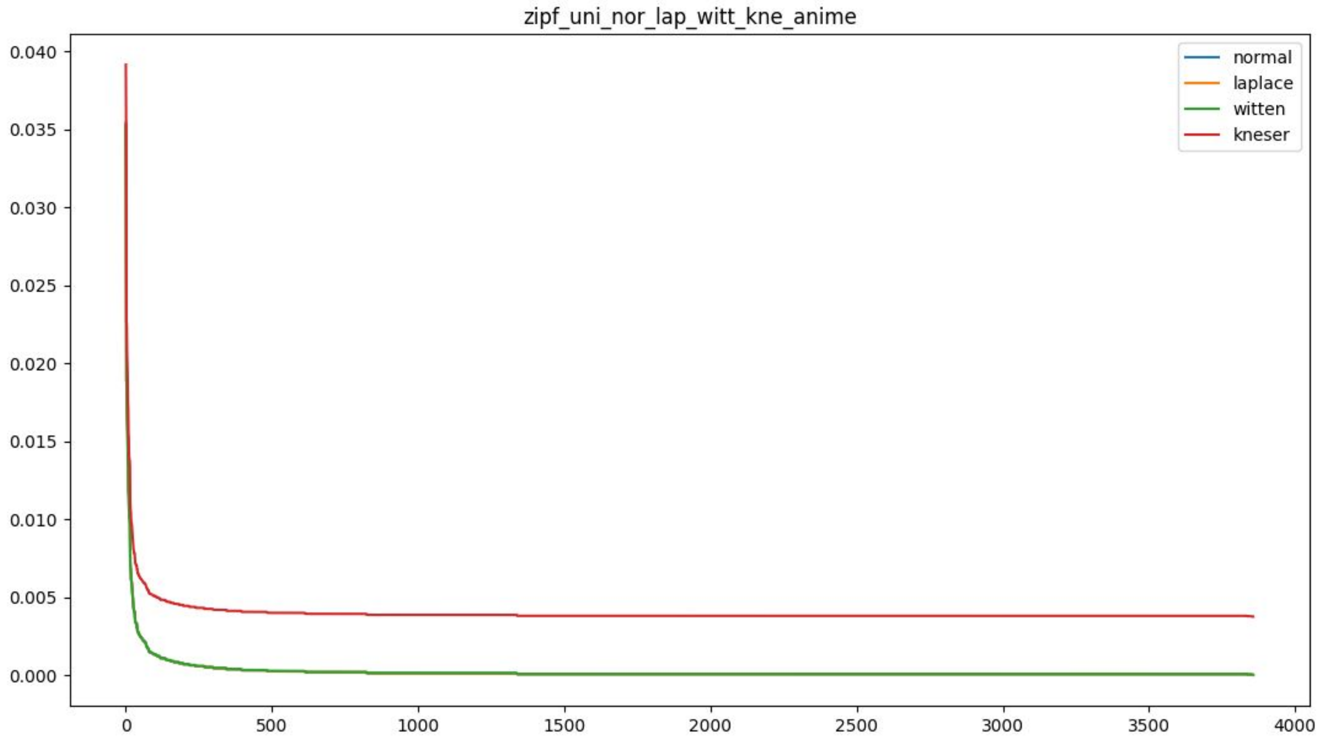



log_tri_nor_lap_witt_kne_anime
0
−1
−2
−3
−4
−5
−6
−7
0
2
4
6
8
normal
laplace
witten
kneser

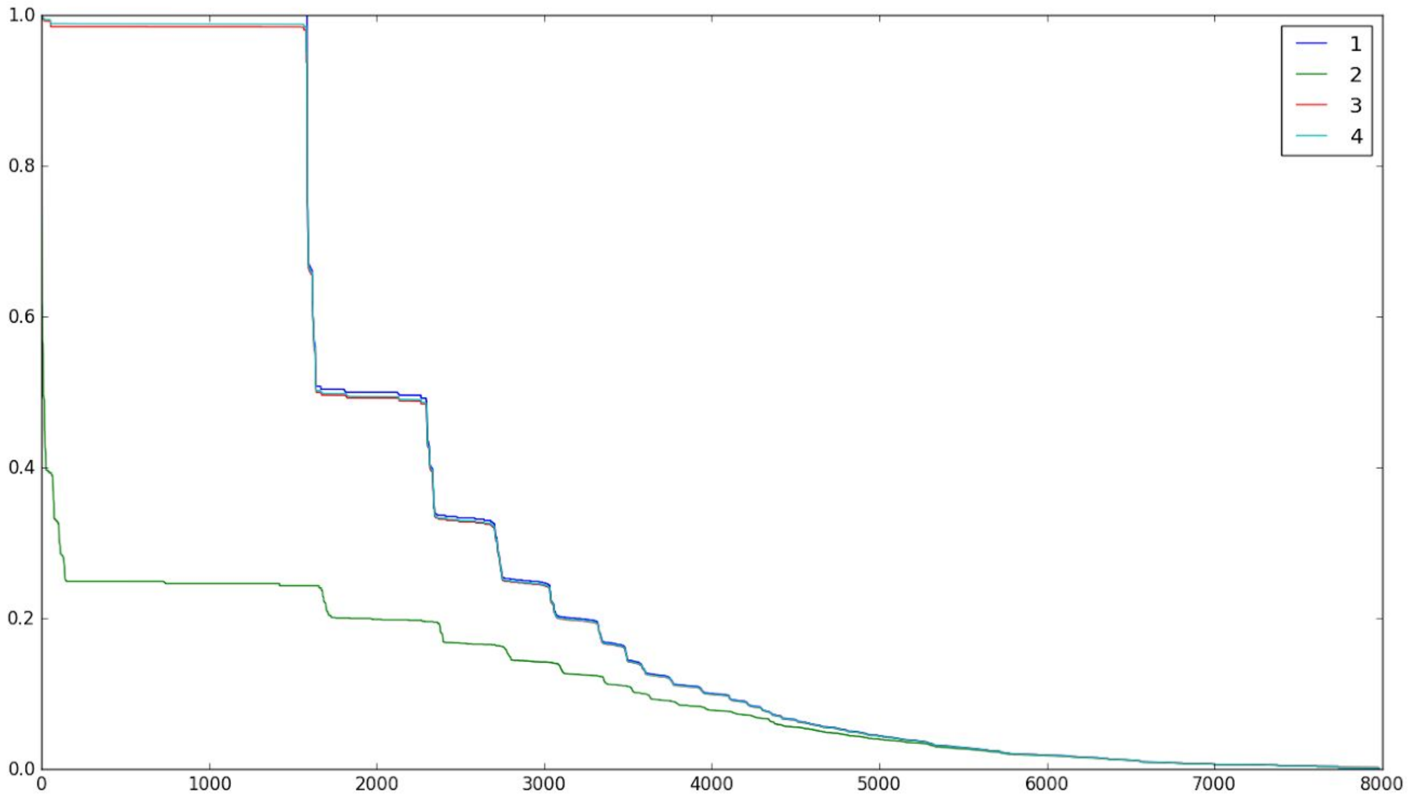

log_uni_nor_lap_witt_kne_anime
−3
−4
−5
−6
−7
−8
−9
−10
−11
0
2
4
6
8
normal
laplace
witten
kneser


zipf_bi_nor_lap_witt_kne_anime
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
normal
laplace
witten
kneser

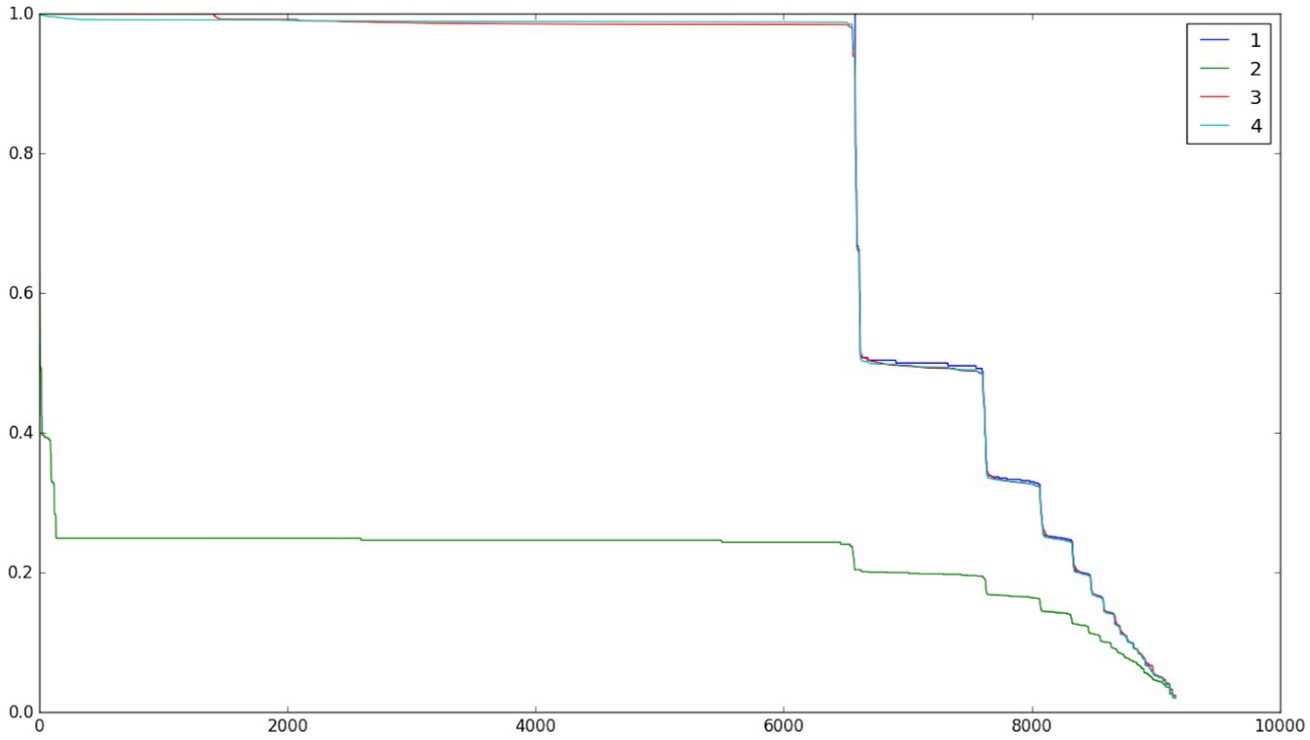




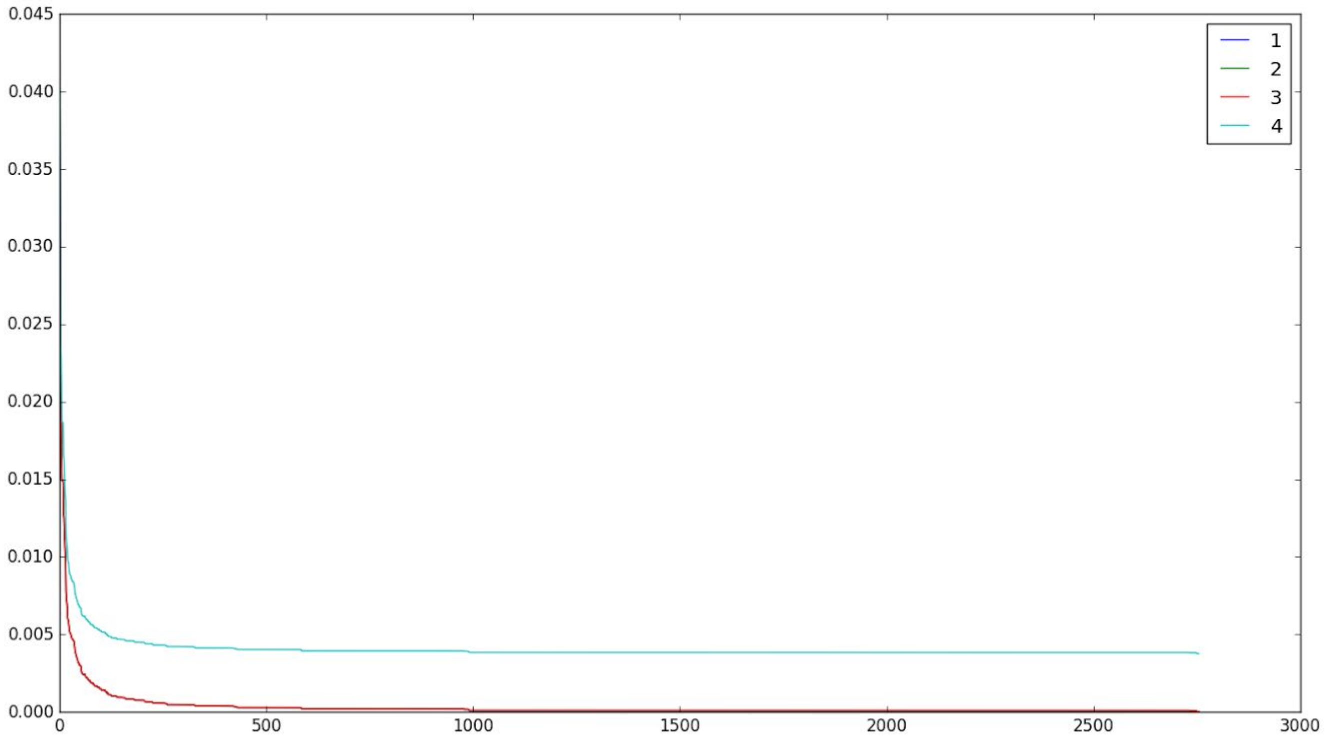
Zipf Bigram News



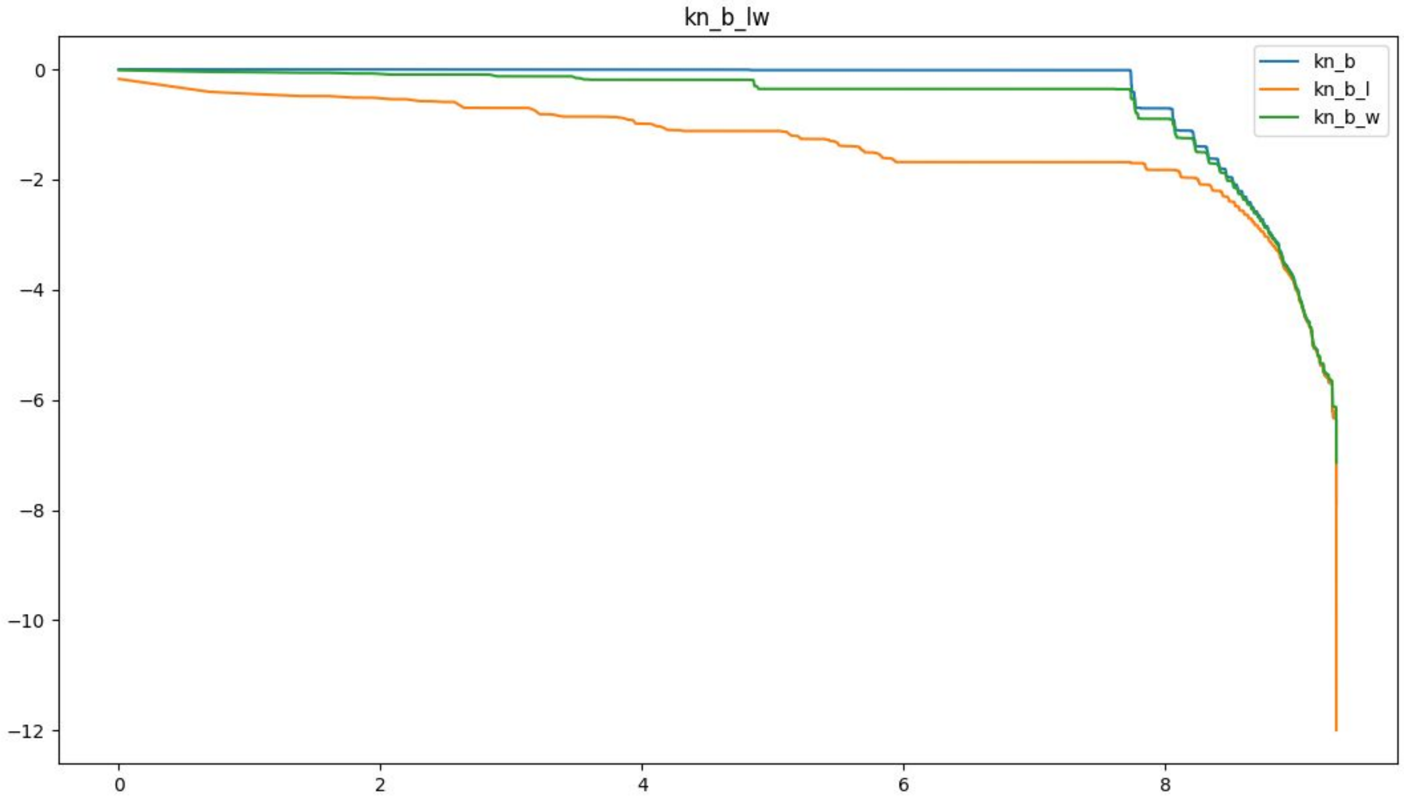
Zipf trigram News

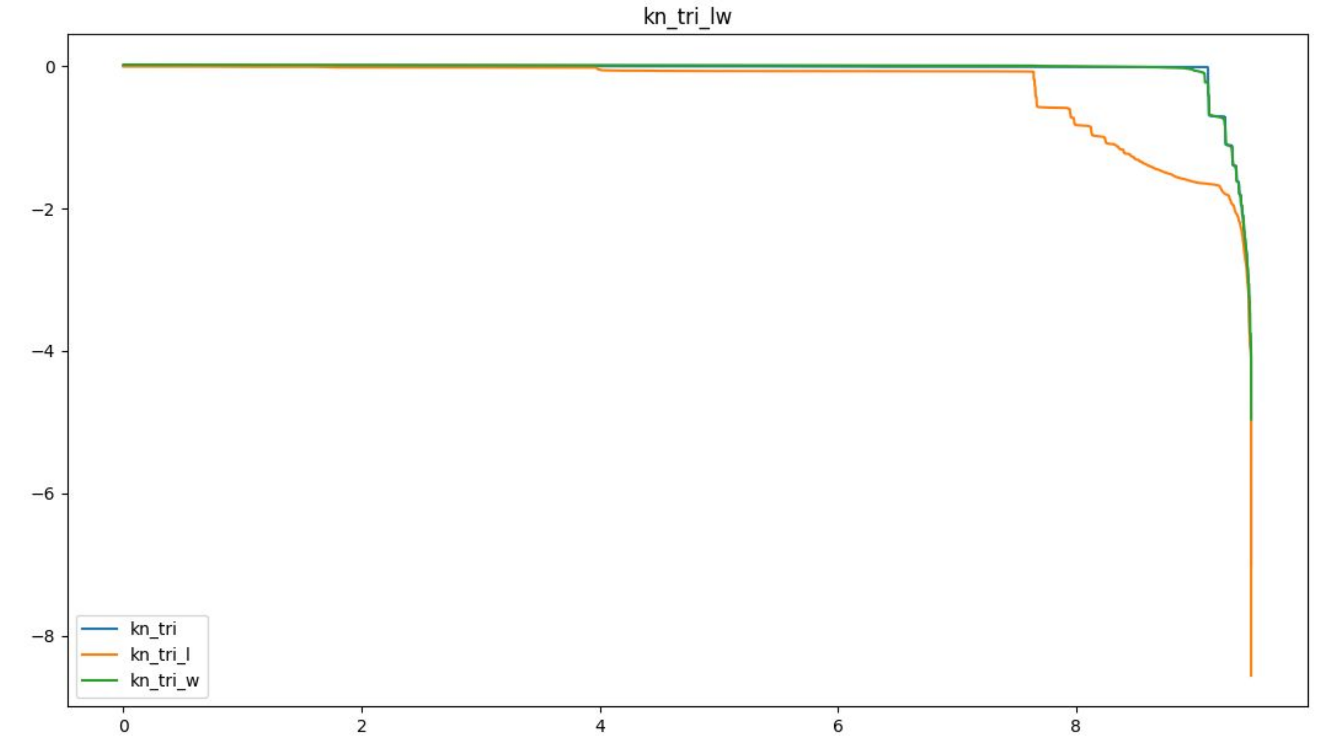


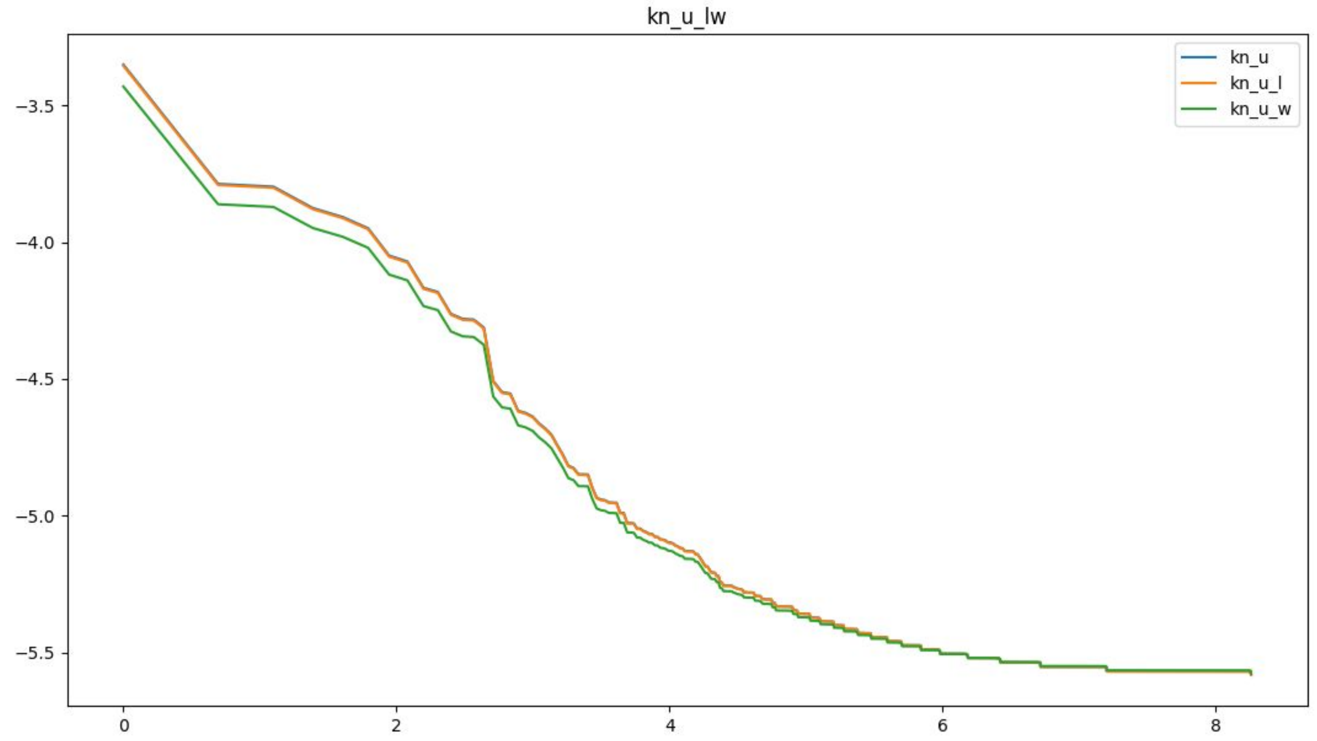
Zipf Unigram News

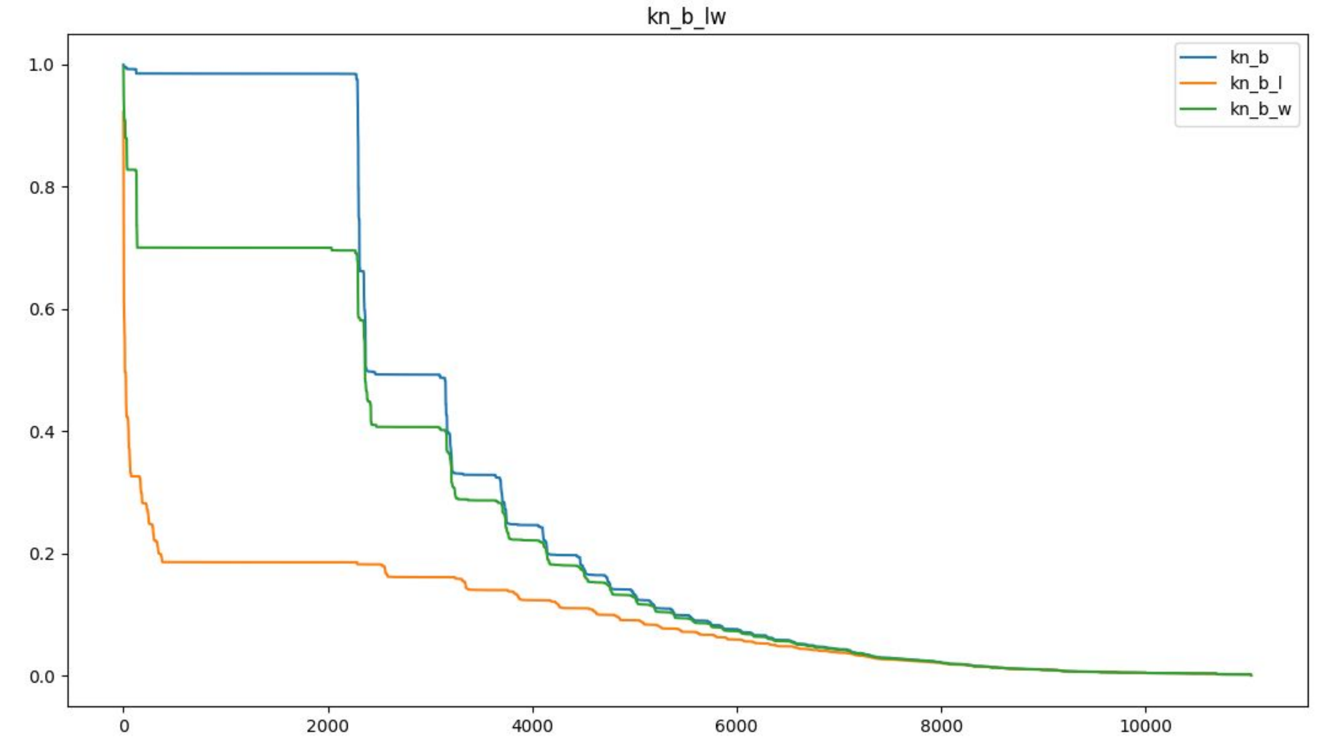


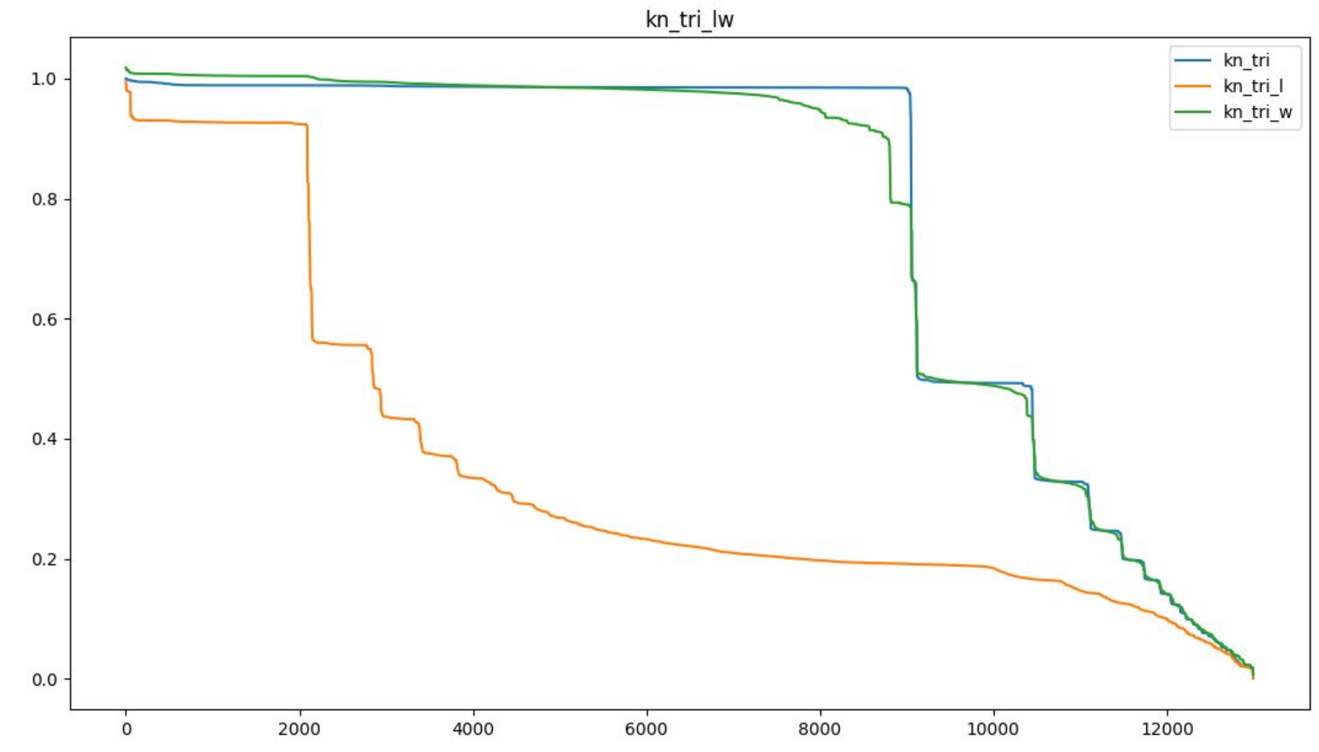
7. In Kneser-Ney, what happens if we use the estimates from laplace and wittenbell in the absolute discounting step ?

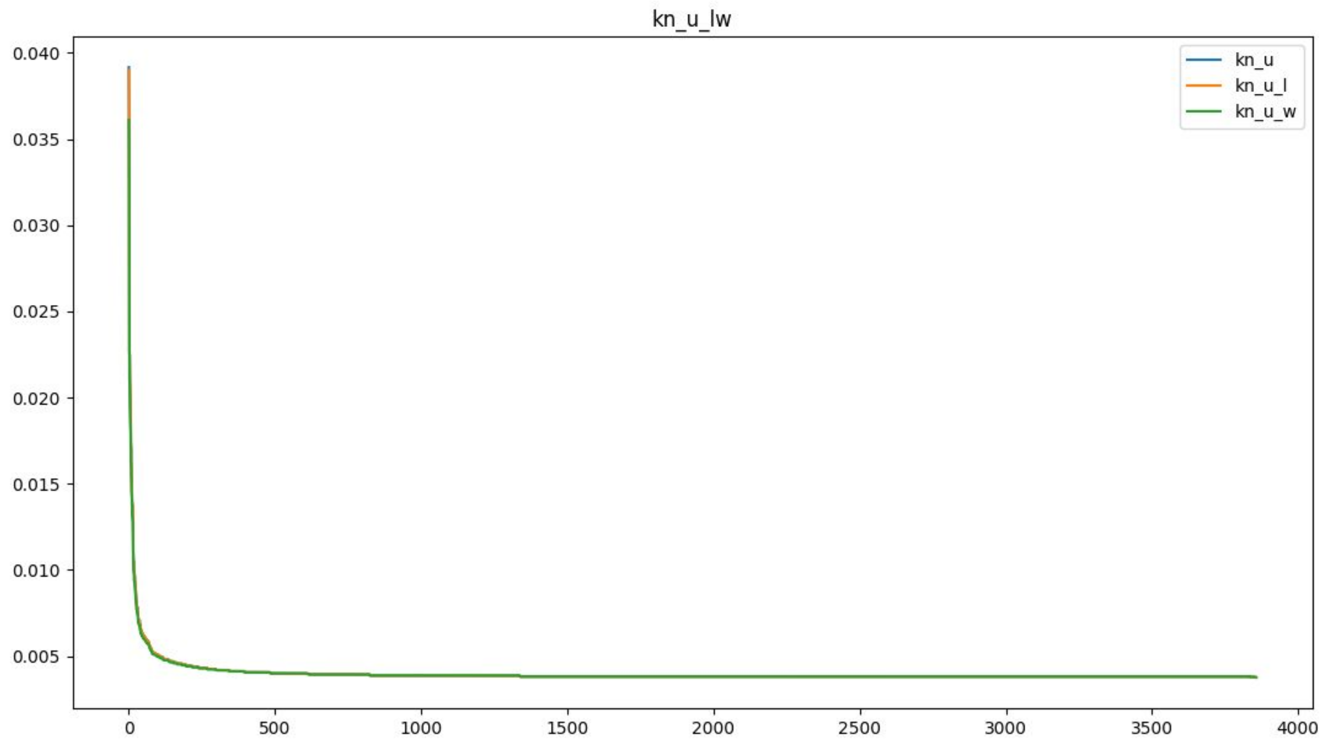



kn_tri_lw


kn_u_lw


kn_b_lw





8. Using KN-estimates from the three sources, generate text with unigram, bigram and trigram probabilities.

Generated Texts:
For trigram:
['team', 'is', 'back', 'it', 'was', 'a', 'little', 'too', 'much', 'and']
['man', 'am', 'i', 'the', 'only', 'one', 'of', 'my', 'favorite', 'anime']

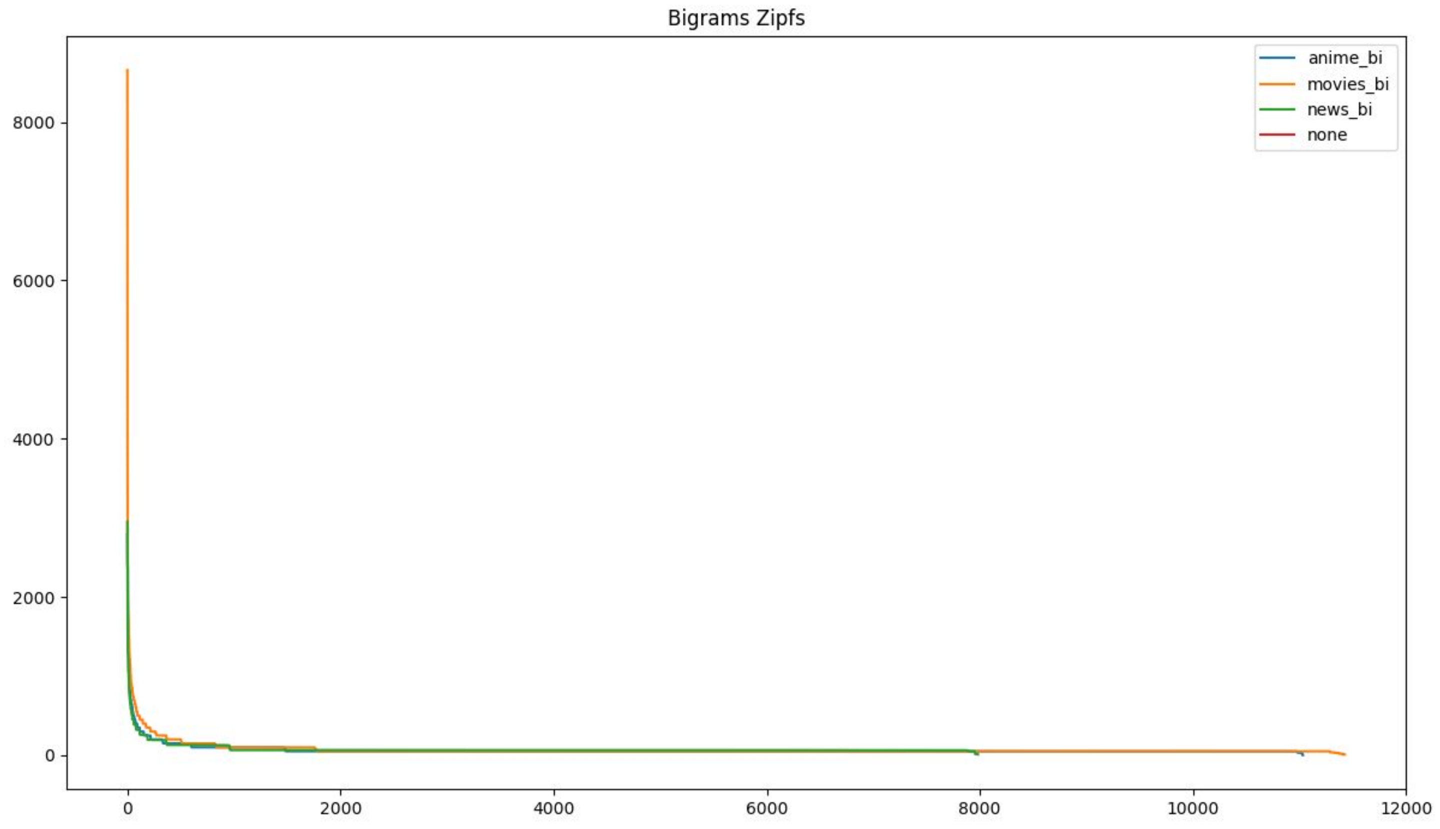
For Bigrams:
['planet', 'but', 'i', 'm', 'not']
['im', 'thoroughly', 'enjoying', 'seeing', 'misaki', 's', 'real', 'personality', 'as', 'well']

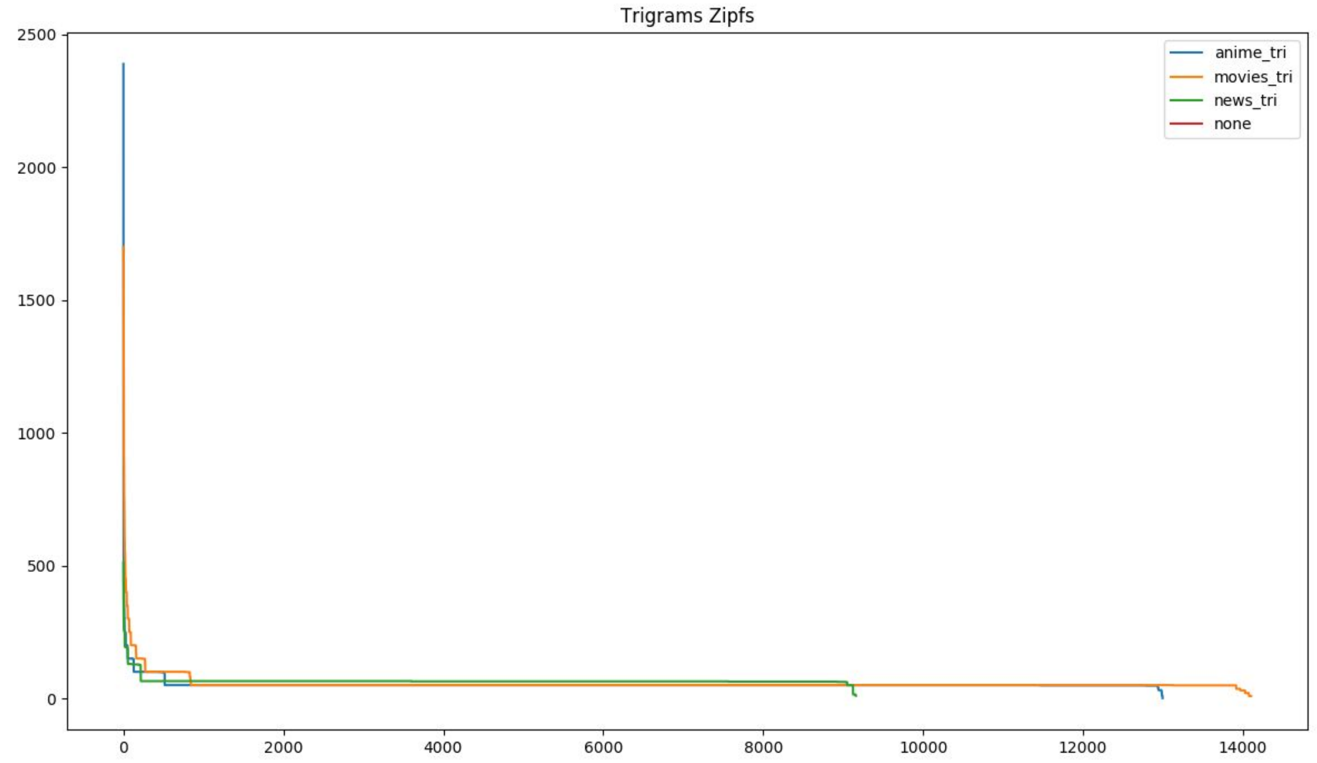
How to generate:

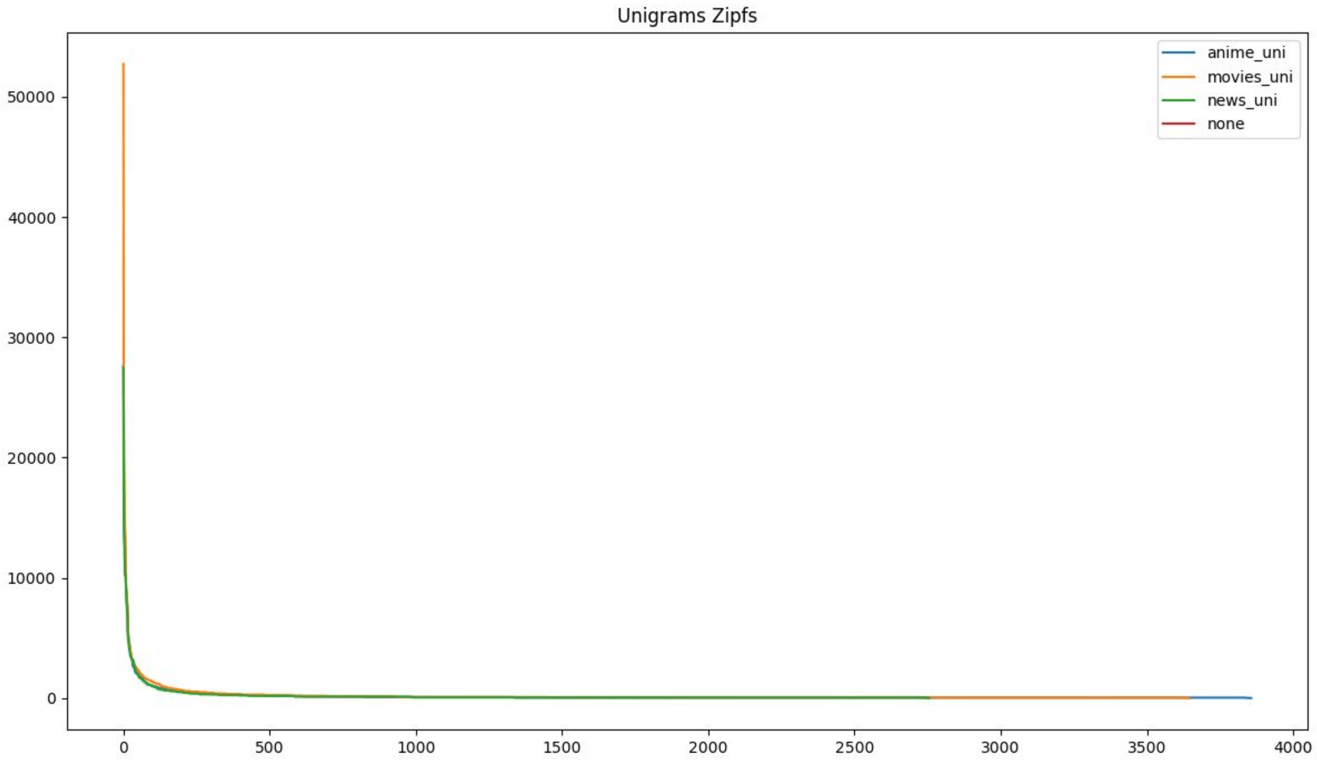
```
def cond_bigrams(bigrams, key):

    joint = {k[1] : v for k, v in bigrams.items() if k[0] == key}
    sum_count = sum(joint.values())
    return {k : v / float(sum_count) for k, v in joint.items() }

def generate_bigrams(unigrams, bigrams, length=5, first_word = None):
    words = []
    if first_word == None:
        first_word = list(unigrams.keys())[random.randrange(0, len(unigrams))]
    words.append(first_word)
    for i in range(length - 1):
        prev = words[i]
        prev_dict = cond_bigrams(bigrams, prev)

        next_word = sorted(prev_dict.items(), key = lambda x : x[1], reverse = True)[0]
        words.append(next_word[0])
    return words
def cond_trigrams(trigrams, key):

    joint = {k[2] : v for k, v in trigrams.items() if (k[0] == key[0] and k[1] == key[1])}
    sum_count = sum(joint.values())
    return {k : v / float(sum_count) for k, v in joint.items() }

def generate_trigrams(unigrams, bigrams,trigrams, length=5, first_word = None):
    words = []
    if first_word == None:
        first_word = list(bigrams.keys())[random.randrange(0, len(bigrams))]
    words=(list(first_word))
    print words
    for i in range(length - 2):
        prev = words[i+1]
        prev2 = words[i]
        prev_dict = cond_trigrams(trigrams, [prev2,prev])

        next_word = sorted(prev_dict.items(), key = lambda x : x[1], reverse = True)[0]
        words.append(next_word[0])
    return words

kn_unigrams,kn_bigrams,kn_trigrams=estimated_count_kn(unigrams, bigrams, trigrams, kn_unigrams_prob,
kn_bigrams_prob, kn_trigrams_prob)
print generate_bigrams(kn_unigrams, kn_bigrams, length=5, first_word = None)
print generate_trigrams(kn_unigrams, kn_bigrams,kn_trigrams, length=5, first_word = None)
```

Naive Baye's

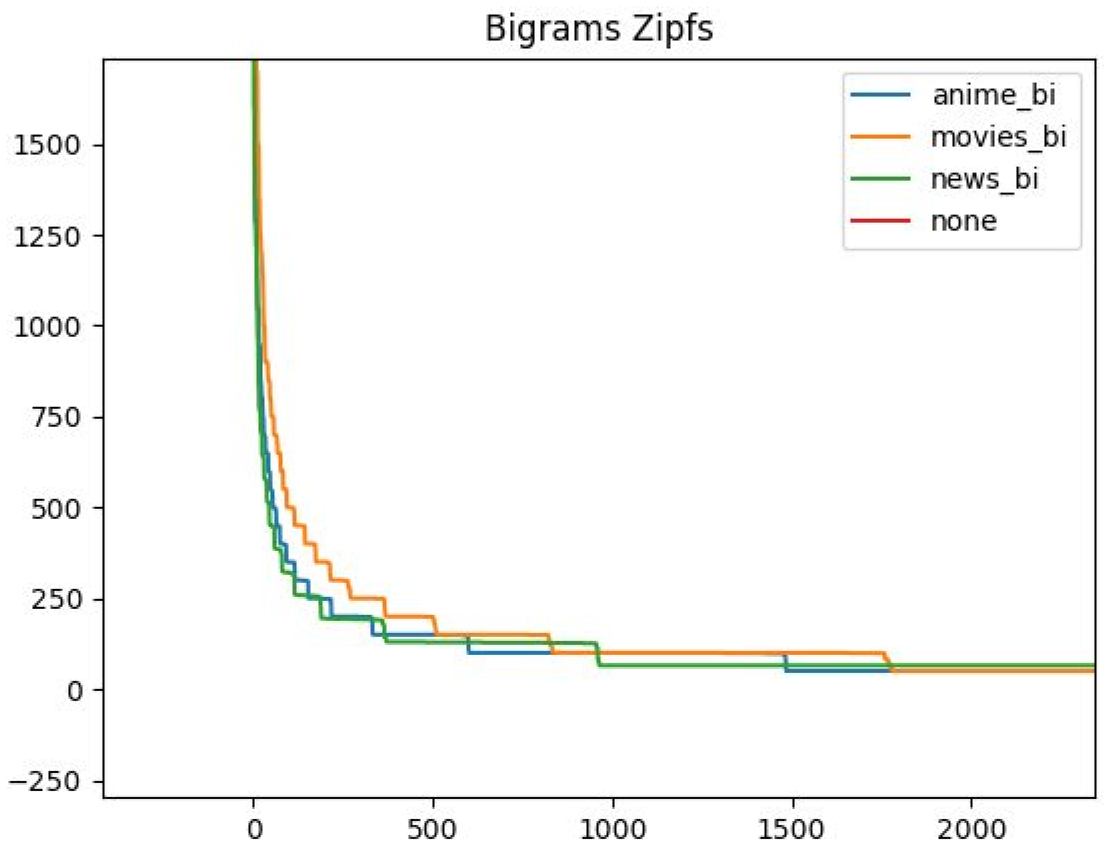
Plot the zipf's curves of all the three sources on one graph. Where do they match ? Where don't they match ?

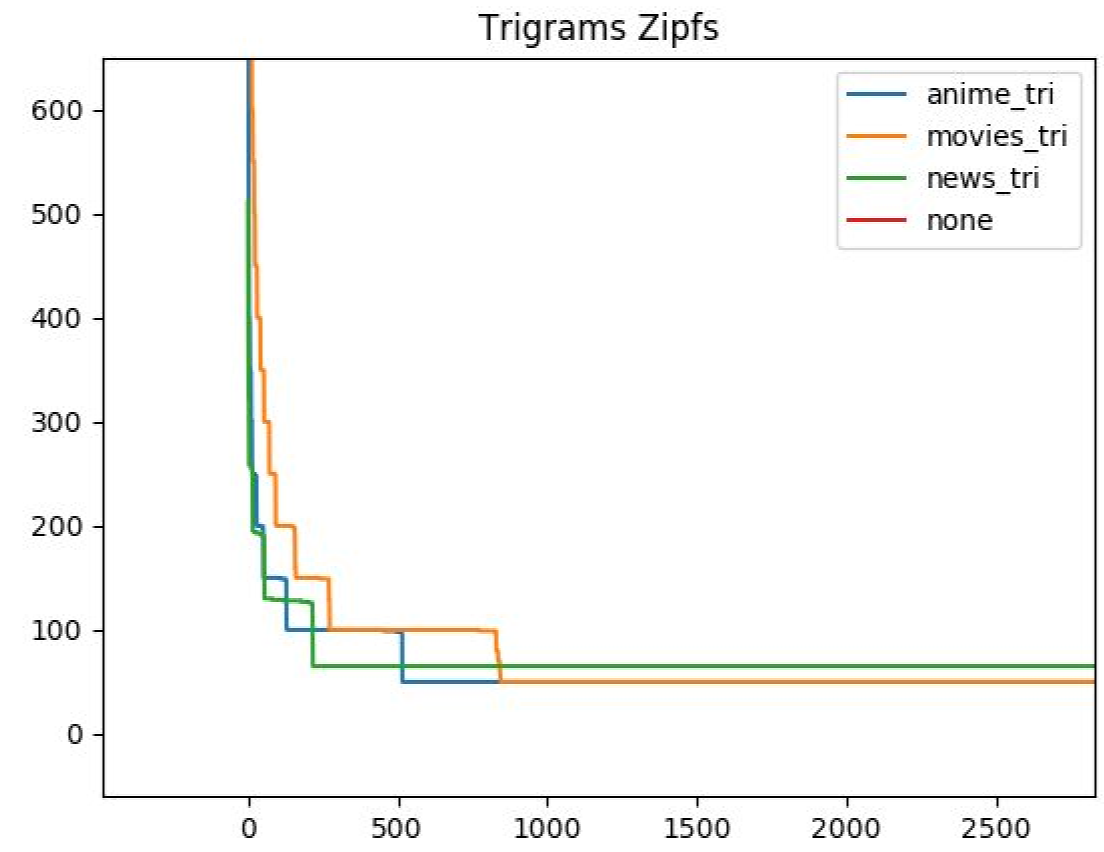


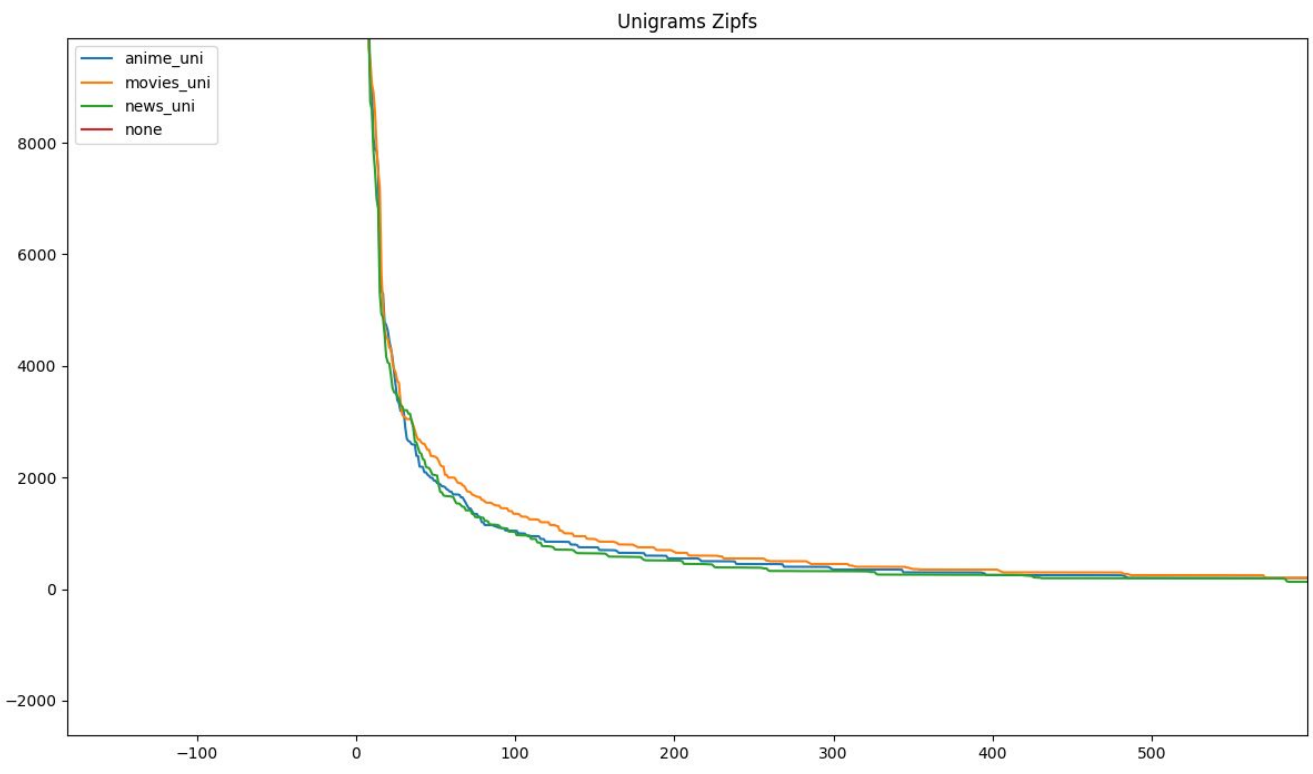




To Clearly view the meeting point I have Zoomed the pictures leading to pictures as below


Bigrams Zipfs
anime_bi
movies_bi
news_bi
none
1500
1250
1000
750
500
250
0
−250
0
500
1000
1500
2000


Trigrams Zipfs
anime_tri
movies_tri
news_tri
none
600
500
400
300
200
100
0
0
500
1000
1500
2000
2500


Unigrams Zipfs
anime_uni
movies_uni
news_uni
none
8000
6000
4000
2000
0
−2000
−100
0
100
200
300
400
500