In [ ]:
```
토큰화(Text -> Encoding;features)
문장(sent_tokenize), 토큰(word/Tweet/RegExp/TreeBank _tokenize)
형태소(pos_tag;penntreebank/stanford...)
연어(collocation), 공기어(co-ocurrence) => N-gram, chi-square, pmi, jacarrd..
; 낱개로 X,       항상 O,          의미적 연관
=> L.A, H.P...  Mr, Miss ...    민주, 주의, ... 대통령, 영부인, ...
+ed, ing, ... Stemming, be => Lemmatization
빈도 >>>>>>>>>>>>>>>>>>>> 고유한(유니크한) 전체 수 => Feature Extraction
Zipf law => 빈도의 역순 = 순위 역순, => 고빈도, 저빈도 중요도 X, 나머지 O
=> Feature Selection
한글에 적용, Heaps, 신조어, 어미, Entropy(-plogp), Perplexity(응집력)
N-Gram(LM, Tokenizing), Edit Distance(오탈자)
```

In [1]:
```
from konlpy.tag import Kkma, Komoran, Hannanum, Okt
# Twitter => Okt, Mecab 별도설치(빠르고 성능 좋음)
from konlpy.corpus import kobill, kolaw
```

In [3]:
```
kobill.fileids(), kolaw.fileids()
```

Out[3]:
```
(['1809896.txt',
  '1809897.txt',
  '1809895.txt',
  '1809894.txt',
  '1809890.txt',
  '1809891.txt',
  '1809893.txt',
  '1809892.txt',
  '1809899.txt',
  '1809898.txt'],
 ['constitution.txt'])
```

In [5]:
```
data = list()

for f in kobill.fileids():
    data.append(kobill.open(f).read())
for f in kolaw.fileids():
    data.append(kolaw.open(f).read())
```

In [7]:
```
from nltk.text import Text
from nltk.tokenize import sent_tokenize, word_tokenize, regexp_tokenize
```

In [8]:
```
len(data[-1].splitlines()), len(sent_tokenize(data[-1]))
```

Out[8]:
```
(356, 357)
```

In [11]:
```
len(data[-1].split()), len(word_tokenize(data[-1]))
```

Out[11]:
```
(4178, 4640)
```

In [12]:
```
import re
p = re.compile('[가-힣]+')
len(regexp_tokenize(data[-1], p))
```

Out[12]:
```
4387
```

In [14]:
```
len(set(data[-1].split())), len(set(word_tokenize(data[-1]))),\
len(set(regexp_tokenize(data[-1], p)))
```

Out[14]:
```
(2029, 2023, 1797)
```

In [15]:
```
text1 = Text(word_tokenize(data[-1]))
```

In [17]:
```
text2 = Text(regexp_tokenize(data[-1], p))
```

In [18]:
```
list(zip(text1.vocab().most_common(10), text2.vocab().most_common(10)))
```

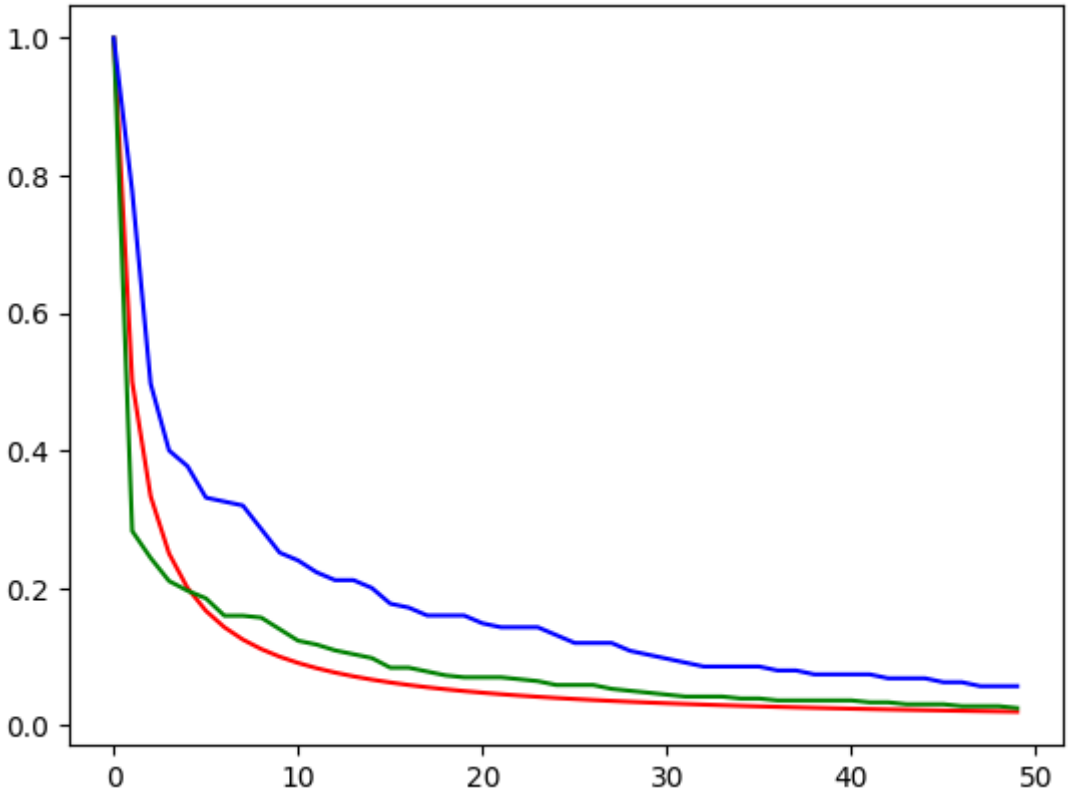
Out[18]:
```
[(('.', 357), ('제', 175)),
 ((',', 101), ('조', 136)),
 (('수', 87), ('수', 87)),
 (('①', 75), ('또는', 70)),
 (('또는', 70), ('의하여', 66)),
 (('의하여', 66), ('법률이', 58)),
 (('법률이', 57), ('있다', 57)),
 (('있다', 57), ('한다', 56)),
 (('한다', 56), ('정하는', 50)),
 (('정하는', 50), ('그', 44))]
```

In [21]:
```
'법률로' in text2.vocab().keys()
```

Out[21]:
```
True
```

In [27]:
```
import matplotlib.pyplot as plt
N = 50
rank = [1/i for i in range(1,N+1)]
text1_rank = list(map(lambda t:t[1]/text1.vocab().get(text1.vocab().max()),
                      text1.vocab().most_common(N)))
text2_rank = list(map(lambda t:t[1]/text2.vocab().get(text2.vocab().max()),
                      text2.vocab().most_common(N)))
plt.plot(rank, 'r-')
plt.plot(text1_rank, 'g-')
plt.plot(text2_rank, 'b-')
```

Out[27]:
```
[<matplotlib.lines.Line2D at 0x2977e63d0>]
```



In [28]:
```
# Morphememm Analyzer -> POS Tagger
# 형태소 분리/분석          품사태거
# (형태론)              (통사론)
ma1 = Kkma() # 서울대
ma2 = Komoran()
ma3 = Hannanum()
ma4 = Okt() # Twitter
```

In [35]:
```
# pos_tag; penntree
# 국립국어원 세종21(1차, 2차 - 문어체, 구어체, 대화체, ...)
# morphs => 형태소, nouns => 명사추출, pos => 품사부착
ma1.morphs('아버지가 방에 들어가신다.')
```

Out[35]:
```
['아버지', '가', '방', '에', '들어가', '시', 'ㄴ다', '.']
```

In [36]:
```
s = '아버지가방에들어가신다.'
ma1.pos(s), ma2.pos(s), ma3.pos(s), ma4.pos(s)
# 우리말 => 띄어쓰기 엄청 강력한 제약, 어절(단어) 구분하는 단위, 중요하게 사용됨
# => 띄어쓰기를 보정, 띄어쓰기를 무시할 수 있는 토크나이저
```

Out[36]:
```
([('아버지', 'NNG'),
  ('가방', 'NNG'),
  ('에', 'JKM'),
  ('들어가', 'VV'),
  ('시', 'EPH'),
  ('ㄴ다', 'EFN'),
  ('.', 'SF')],
 [('아버지', 'NNG'),
  ('가방', 'NNP'),
  ('에', 'JKB'),
  ('들어가', 'VV'),
  ('시', 'EP'),
  ('ㄴ다', 'EF'),
  ('.', 'SF')],
 [('아버지가방에들어가', 'N'), ('이', 'J'), ('시ㄴ다', 'E'), ('.', 'S')],
 [('아버지', 'Noun'),
  ('가방', 'Noun'),
  ('에', 'Josa'),
  ('들어가신다', 'Verb'),
  ('.', 'Punctuation')])
```

In [44]:
```
ma1.tagset['EPH'], ma2.tagset['JKB'], ma3.tagset['E']
```

Out[44]:
```
('존칭 선어말 어미', '부사격 조사', '어미')
```

In [47]:
```
ma2.pos('들어가셨다. 들어가셨는데, 들어가시고, 들어간다. 들어갔다. 들다.')
```

Out[47]:
```
[('들어가', 'VV'),
 ('시', 'EP'),
 ('었', 'EP'),
 ('다', 'EF'),
 ('.', 'SF'),
 ('들어가', 'VV'),
 ('시', 'EP'),
 ('었', 'EP'),
 ('는데', 'EC'),
 (',', 'SP'),
 ('들어가', 'VV'),
 ('시', 'EP'),
 ('고', 'EC'),
 (',', 'SP'),
 ('들어가', 'VV'),
 ('ㄴ다', 'EF'),
 ('.', 'SF'),
 ('들어가', 'VV'),
 ('았', 'EP'),
 ('다', 'EF'),
 ('.', 'SF'),
 ('들', 'VV'),
 ('다', 'EF'),
 ('.', 'SF')]
```

In [50]:
```
s = '어쩔티비'
ma1.pos(s), ma2.pos(s), ma3.pos(s), ma4.pos(s)
```

Out[50]:
```
([('어', 'VV'),
  ('어', 'ECS'),
  ('쩔', 'VV'),
  ('ㄹ', 'ETD'),
  ('티', 'NNG'),
  ('비', 'NNG')],
 [('어', 'IC'), ('쩌', 'EF'), ('ㄹ', 'ETM'), ('티', 'NNG'), ('비', 'NNG')],
 [('어', 'N'), ('쩔티비', 'N')],
 [('어', 'Eomi'), ('쩔', 'Noun'), ('티비', 'Noun')])
```

In [55]:
```
text_ma1 = Text('\t'.join(
    [' '.join(ma1.morphs(s)) for s in sent_tokenize(data[-1])]).split())
text_ma2 = Text('\t'.join(
    [' '.join(ma2.morphs(s)) for s in sent_tokenize(data[-1])]).split())
text_ma3 = Text('\t'.join(
    [' '.join(ma3.morphs(s)) for s in sent_tokenize(data[-1])]).split())
text_ma4 = Text('\t'.join(
    [' '.join(ma4.morphs(s)) for s in sent_tokenize(data[-1])]).split())
```

In [57]:
```
(text_ma1.vocab().N(), text_ma1.vocab().B()),\
(text_ma2.vocab().N(), text_ma2.vocab().B()),\
(text_ma3.vocab().N(), text_ma3.vocab().B()),\
(text_ma4.vocab().N(), text_ma4.vocab().B())
```

Out[57]:
```
((10053, 1247), (9800, 1219), (8549, 1469), (8452, 1358))
```

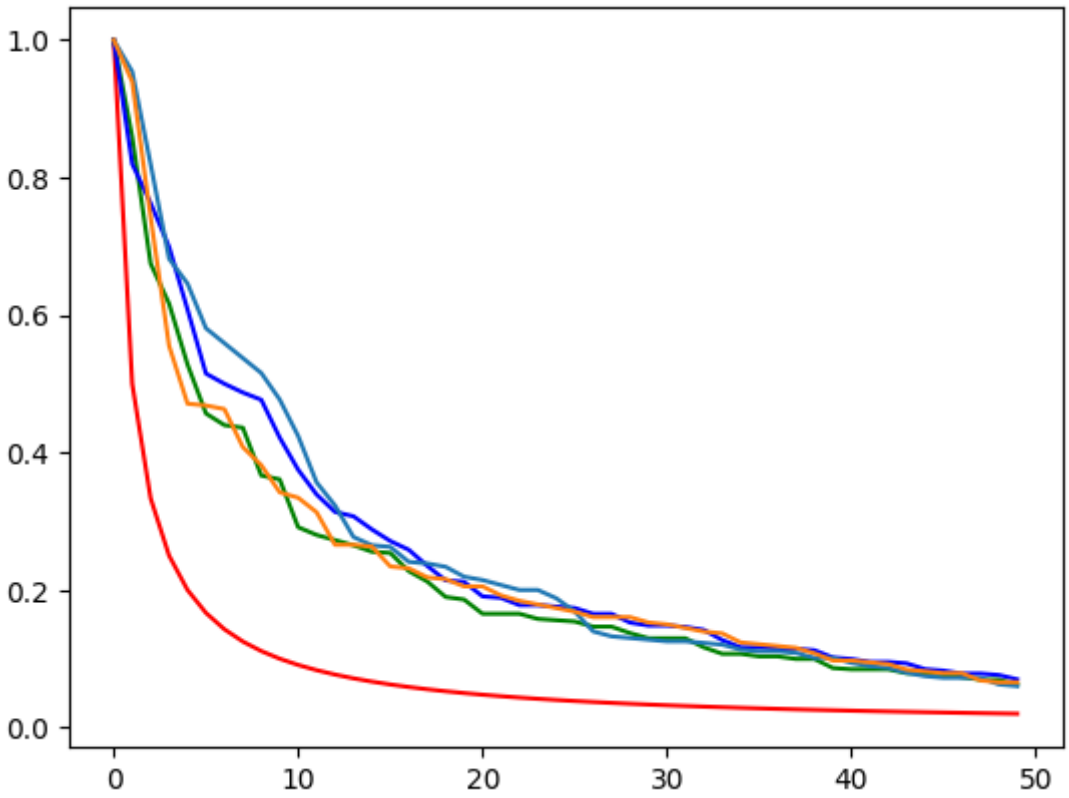
In [59]:
```
list(zip(text_ma1.vocab().most_common(10),
         text_ma2.vocab().most_common(10),
         text_ma3.vocab().most_common(10),
         text_ma4.vocab().most_common(10)))
```

Out[59]:
```
[(('의', 532), ('하', 472), ('하', 415), ('의', 380)),
 (('하', 457), ('의', 387), ('의', 396), ('.', 357)),
 (('.', 359), ('.', 360), ('.', 340), ('에', 282)),
 (('에', 328), ('에', 330), ('에', 283), ('을', 211)),
 (('는', 281), ('는', 287), ('이', 268), ('은', 179)),
 (('ㄴ다', 243), ('ㄴ다', 243), ('ㄴ다', 241), ('제', 178)),
 (('ㄴ', 234), ('ㄴ', 236), ('을', 232), ('이', 176)),
 (('을', 232), ('이', 230), ('ㄴ', 223), ('한다', 155)),
 (('은', 195), ('을', 225), ('는', 214), ('·', 145)),
 (('이', 192), ('은', 199), ('은', 198), ('를', 130))]
```

In [63]:
```
import matplotlib.pyplot as plt
N = 50
rank = [1/i for i in range(1,N+1)]
text1_rank = list(map(lambda t:t[1]/text_ma1.vocab().get(
    text_ma1.vocab().max()), text_ma1.vocab().most_common(N)))
text2_rank = list(map(lambda t:t[1]/text_ma2.vocab().get(
    text_ma2.vocab().max()), text_ma2.vocab().most_common(N)))
text3_rank = list(map(lambda t:t[1]/text_ma3.vocab().get(
    text_ma3.vocab().max()), text_ma3.vocab().most_common(N)))
text4_rank = list(map(lambda t:t[1]/text_ma4.vocab().get(
    text_ma4.vocab().max()), text_ma4.vocab().most_common(N)))

plt.plot(rank, 'r-')
plt.plot(text1_rank, 'g-')
plt.plot(text2_rank, 'b-')
plt.plot(text3_rank)
plt.plot(text4_rank)
```

Out[63]:
```
[<matplotlib.lines.Line2D at 0x175051a30>]
```



In [65]:
```
th = .3
cp = 0.0
mft = list()
for t in text_ma1.vocab().most_common(100):
    if cp > th:
        break
    cp += text_ma1.vocab().freq(t[0])
    mft.append(t)
```

In [67]:
```
len(mft), text_ma1.vocab().most_common(100)[11:20]
```

Out[67]:
```
(10,
 [('여', 149),
  ('·', 145),
  ('ㄹ', 141),
  ('조', 136),
  ('를', 135),
  ('법률', 121),
  ('되', 113),
  (',', 101),
  ('있', 99)])
```

In [71]:
```
text_ma1.collocation_list()
```

Out[71]:
```
[('정치적', '중립성'), ('재판소', '재판관')]
```

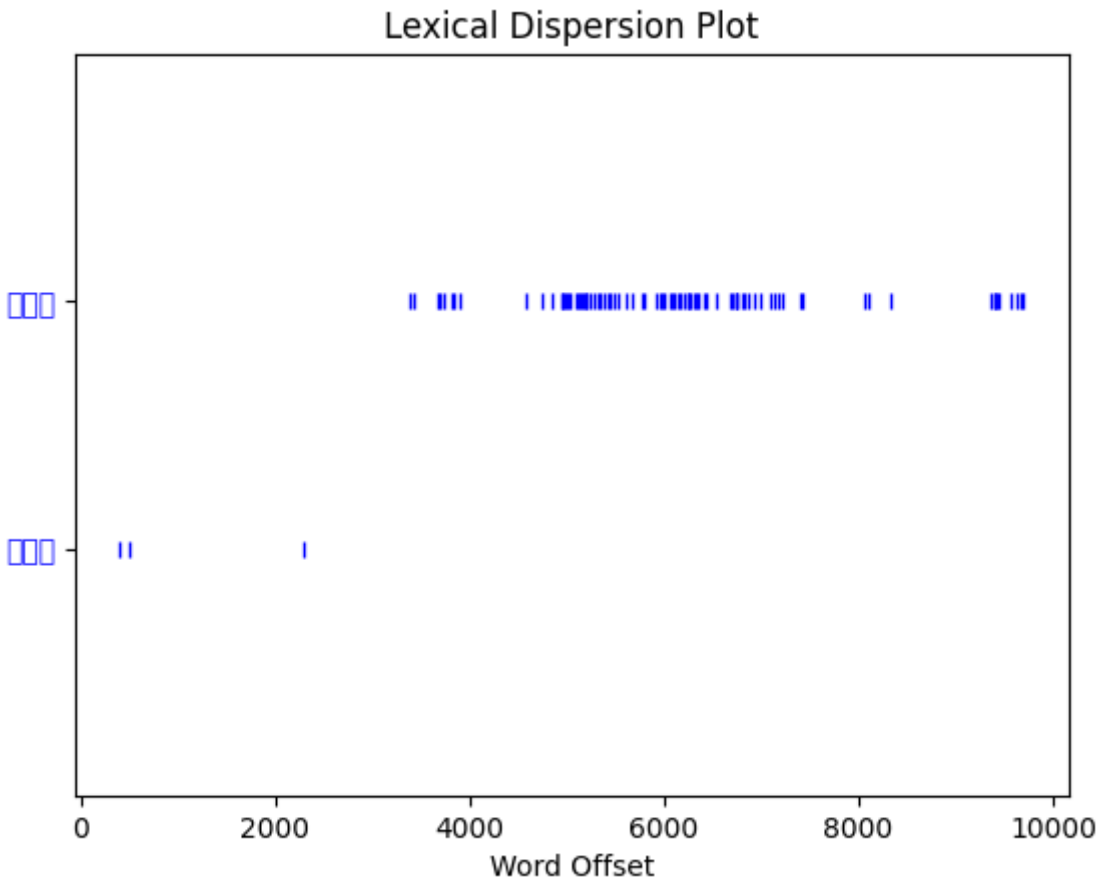
In [73]:
```
text_ma1.concordance('재판소')
```

```
Displaying 14 of 14 matches:
활동 이 민주적 기본 질서 에 위배 되 ㄹ 때 에 는 정부 는 헌법 재판소 에 그 해산 을
제소 하 ㄹ 수 있 고 , 정당 은 헌법 재판소 의
법 재판소 에 그 해산 을 제소 하 ㄹ 수 있 고 , 정당 은 헌법 재판소 의 심판 에 의하
여 해산 되 ㄴ다 . 저 의 9 조 국가 는 전통
① 대통령 · 국무총리 · 국무 위원 · 행정 각부 의 장 · 헌법 재판소 재판관 · 법관 · 중
앙 선거 관리 위원회 위원 · 감사원장 · 감
되 는 여부 가 재판 의 전제 가 되 ㄴ 경우 에 는 법원 은 헌법 재판소 에 제청 하 여 그
심판 에 의하 여 재판 하 ㄴ다 . ② 명령 ·
고 하 ㄴ 경우 에 는 그러하 지 아니하 다 . 저 의 6 장 헌법 재판소 제 111 조 ① 헌법
재판소 는 다음 사항 을 관장 하 ㄴ다 .
지 아니하 다 . 저 의 6 장 헌법 재판소 제 111 조 ① 헌법 재판소 는 다음 사항 을 관장
하 ㄴ다 . 1 . 법원 의 제청 에 의하
심판 5 . 법률 이 정하 는 헌법 소원 에 관하 ㄴ 심판 ② 헌법 재판소 는 법관 의 자격 을
가지 ㄴ 9 인의 재판관 으로 구성 하 며 ,
3 인은 대법원장 이 지명 하 는 자 를 임명 하 ㄴ다 . ④ 헌법 재판소 의 장 은 국회 의
동의 를 얻 어 재판 관중 에서 대통령 이 임명
 관중 에서 대통령 이 임명 하 ㄴ다 . 저 의 112 조 ① 헌법 재판소 재판관 의 임기 는 6
년 으로 하 며 , 법률 이 정하 는 바 에
률 이 정하 는 바 에 의하 여 연임 하 ㄹ 수 있 다 . ② 헌법 재판소 재판관 은 정당 에
가입 하 거나 정치 에 관여 하 ㄹ 수 없 다
정당 에 가입 하 거나 정치 에 관여 하 ㄹ 수 없 다 . ③ 헌법 재판소 재판관 은 탄핵 또
는 금고 이상 의 형의 선고 에 의하 지 아니하
니하 고는 파 면 되 지 아니하 ㄴ다 . 저 의 113 조 ① 헌법 재판소 에서 법률 의 위헌 결
정 , 탄핵 의 결정 , 정당 해산 의 결정
 는 재판관 6 인 이상 의 찬성 이 있 어야 하 ㄴ다 . ② 헌법 재판소 는 법률 에 저촉 되
지 아니하 는 범위 안 에서 심판 에 관하 ㄴ
사무 처리 에 관하 ㄴ 규칙 을 제정 하 ㄹ 수 있 다 . ③ 헌법 재판소 의 조직 과 운영 기
타 필요 하 ㄴ 사항 은 법률 로 정하 ㄴ다 .
```

In [77]:
```
text_ma1.vocab().get('중립성')
```

Out[77]: 3

In [75]:
```
text_ma1.dispersion_plot(['대통령', '중립성'])
```



In [83]:
```
text_ma1.similar('대통령')
```

법률 국회의원 공무원 국민 조약 법원 대한민국 헌법 저 국가 정책 효력 정당 발전 법관 재판 재산권 군인 교육 항

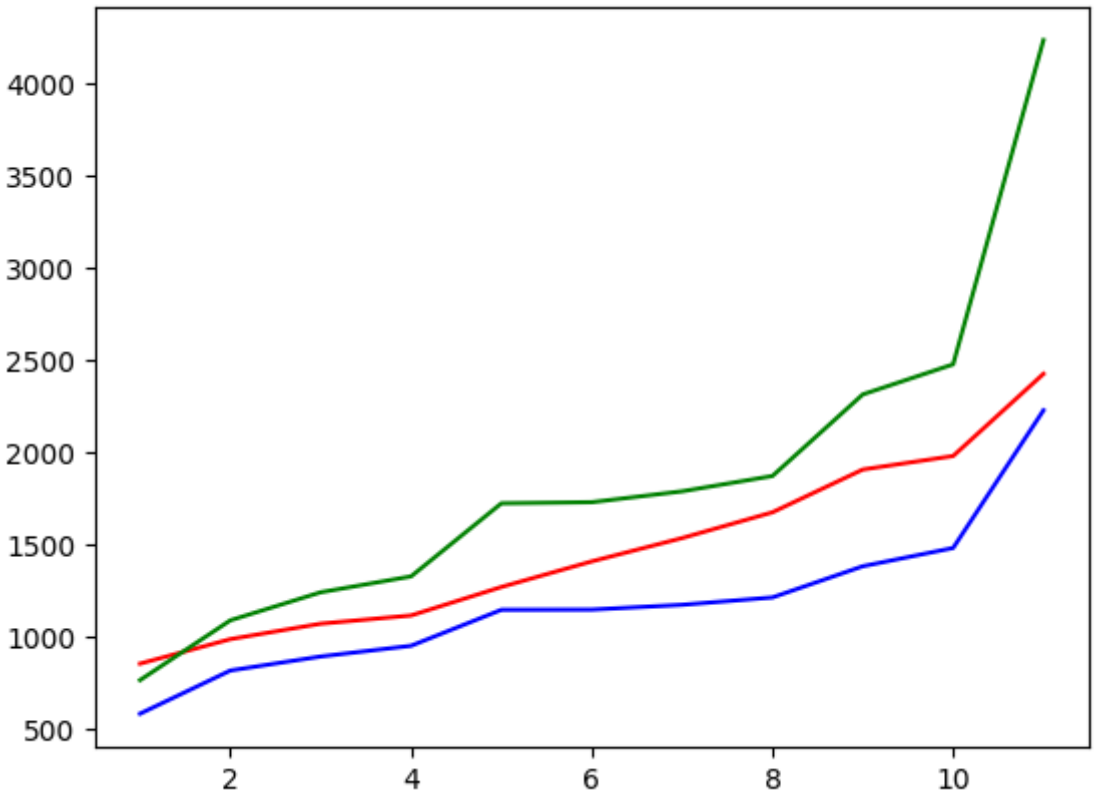
In [122…
```
heaps_1 = list()
for i, _ in enumerate(data):
    heaps_1.append(Text(word_tokenize('\n'.join(data[0:i+1]))))
```

In [141…
```
heaps_2 = list()
for i, _ in enumerate(data):
    heaps_2.append(Text(
        '\n'.join([' '.join(ma1.morphs(s))
        for s in sent_tokenize('\n'.join(data[0:i+1]))]).split()
    ))
# 메모리 충분한, 그냥 돌리면 됨

# doc1, doc2, ...
# join(doc1, doc2) => text?
# sent_tokenize(text?) => s1, s2, ...
# -----> 메모리 부족 터져요, 온전하게 형태소 문장
# 형태소분석기(s1) => ['형태소', '형태소' , ...] => join('형태소 형태소 ...')
# ['형태소 형태소 ... ㅌ s1', '형태소 형태소 ... ㅌ s2'] => join(\n)
# '형태소 형태소 ... ㅌ s1\n형태소 형태소 ... ㅌ s2\n...'
# split => ['형태소', '형태소', '형태소'] ... => Text 객체
```

In [149…
```
heaps = lambda N,k=10,b=.53:k*(N**b) # k=10 ~100, b=.4~.6
plt.plot(range(1,len(data)+1),
         [heaps(h.vocab().N()) for h in heaps_2], 'r-')
plt.plot(range(1,len(data)+1),
         [h.vocab().B() for h in heaps_1], 'g-')
plt.plot(range(1,len(data)+1),
         [h.vocab().B() for h in heaps_2], 'b-')
```

Out[149]: [<matplotlib.lines.Line2D at 0x2b06a7820>]



In [150…
```
heaps(10000000)
```

Out[150]: 51286.13839913651

In [ ]:
```
# Stemming(어간), Lemmatization(원형)
```

In [153…
```
heaps_1[0].vocab().B(), heaps_1[0].vocab().N()
```

Out[153]: (768, 2443)

In [151…
```
heaps_1[-1].vocab().B(), heaps_1[-1].vocab().N()
```

Out[151]: (4236, 16978)

In [152…
```
text -> Encoding(기계)
Zipf: 빈도의 순으로 나열했을 때, 빈도가 크다 = 확률 크다, P(크다)=중요, X
      텍스트(언어X)
Heap's: 문서 전체 단어수가, 문서가 많아질 수록 단어의 수도 많아짐
        단어의 수가 많아지니깐, 유니크 한 단어의 수도 많아짐
        어떤 법칙 전체 단어수^ =(법칙) 유니크 단어 ^
        5천개 -> 500개, 만개 -> 1000개
```

```
사용법: java [-options] class [args...]
           (클래스 실행)
   또는  java [-options] -jar jarfile [args...]
           (jar 파일 실행)
여기서 options는 다음과 같습니다.
    -d32          사용 가능한 경우 32비트 데이터 모델을 사용합니다.
    -d64          사용 가능한 경우 64비트 데이터 모델을 사용합니다.
    -server       "server" VM을 선택합니다.
                  기본 VM은 server입니다.,
                  서버급 시스템에서 실행 중이기 때문입니다.


    -cp <디렉토리 및 zip/jar 파일의 클래스 검색 경로>
    -classpath <디렉토리 및 zip/jar 파일의 클래스 검색 경로>
                  클래스 파일을 검색할 :(으)로 구분된 디렉토리,
                  JAR 아카이브 및 ZIP 아카이브 목록입니다.
    -D<name>=<value>
                  시스템 속성을 설정합니다.
    -verbose:[class|gc|jni]
                  상세 정보 출력을 사용으로 설정합니다.
    -version      제품 버전을 인쇄한 후 종료합니다.
    -version:<value>
                  경고: 이 기능은 사용되지 않으며
                  이후 릴리스에서 제거됩니다.
                  실행할 버전을 지정해야 합니다.
    -showversion  제품 버전을 인쇄한 후 계속합니다.
    -jre-restrict-search | -no-jre-restrict-search
                  경고: 이 기능은 사용되지 않으며
                  이후 릴리스에서 제거됩니다.
                  버전 검색에서 사용자 전용 JRE를 포함/제외합니다.
    -? -help      이 도움말 메시지를 인쇄합니다.
    -X            비표준 옵션에 대한 도움말을 인쇄합니다.
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                  세분성이 지정된 검증을 사용으로 설정합니다.
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  세분성이 지정된 검증을 사용 안함으로 설정합니다.
    -esa | -enablesystemassertions
                  시스템 검증을 사용으로 설정합니다.
    -dsa | -disablesystemassertions
                  시스템 검증을 사용 안함으로 설정합니다.
    -agentlib:<libname>[=<options>]
                  <libname> 고유 에이전트 라이브러리를 로드합니다(예: -agentlib:hpr
of).
                  -agentlib:jdwp=help 및 -agentlib:hprof=help도 참조하십시오.
    -agentpath:<pathname>[=<options>]
                  전체 경로명을 사용하여 고유 에이전트 라이브러리를 로드합니다.
    -javaagent:<jarpath>[=<options>]
                  Java 프로그래밍 언어 에이전트를 로드합니다. java.lang.instrument
를 참조하십시오.
    -splash:<imagepath>
                  이미지가 지정된 스플래시 화면을 표시합니다.
자세한 내용은 http://www.oracle.com/technetwork/java/javase/documentation/index.ht
ml을 참조하십시오.
```

In [ ]:

```
NLP: NLU:순서(문맥) 중요한
토큰1, 토큰2, 토큰3, ...
P(토큰1) = freq토근1/N => MLE
P(토큰2) = freq토근2/N
P(토큰2|토큰1) = P(토큰1,토큰2)/P(토큰1) = freq(토큰1,토큰2)/freq(토큰1)
```

```
P(토큰3|토큰1,토큰2) = ?
====================== LM
P(?|토큰1) => NLG
P('I love you') = P('I')*P('love|I')*P('you|I,love')
P('I like you') = P('I')*P('like|I')*P('you|I,like') = 0
                          freq('I,like') X?
P('')
N-gram:
    1-gram: I, love, like, you
    2-gram: (I,love), (I,like), (love,you), ...
    3-gram: (I,love,you), ...
            => 거의 없음
Markov Assumption:
    P(i|~i) ~= P(i|i-1) => 1st Markov Assumption
                P( | i-2 ) => 2nd "
```

In [158…
```
'유구한 역사와 전통에 빛나는 우리'
P('우리'|'유구한') ~= P('우리'|'빛나는'?????)
          2-gram = (토큰1,토큰2), ... (?=11172,토큰2)
          3-gram = (토큰1, 토큰2, 토큰3), ... (?=11172,?=11172,토큰3)
```

Out[158]: '유구한 역사와 전통에 빛나는 우리'

In [159…
```
def ngram(t, n=2): # 2번째 방법(Padding 없이)
    tokens = t.split()
    gram = list()

    if len(tokens) < n:
        return tokens
    for i in range(len(tokens)-(n-1)): # 3 - (2-1) ; 2
        gram.append(' '.join(tokens[i:i+n]))
    return gram

# '유구한 역사와 전통에 빛나는 우리'
# 1. '유구한', '유구한 역사와', '역사와 전통에', .... '빛나는 우리', '우리'
# 2. '유구한 역사와', ... '빛나는 우리'
```

In [177…
```
import re
from string import punctuation

p1 = re.compile('[{}]'.format(re.escape(punctuation)))
p2 = re.compile('\s+')
```

In [178…
```
model = dict()
for gram in ngram(p2.sub(' ', p1.sub(' ', data[-1]))):
    if gram in model:
        model[gram] += 1
    else:
        model[gram] = 1
```

In [180…
```
sum(model.values()), max(model.values())
```

Out[180]: (4179, 56)

In [182…
```
sorted(model.items(), key=lambda _:_[1], reverse=True)[:10]
```

Out[182]:
```
[('수 있다', 56),
 ('법률이 정하는', 48),
 ('정하는 바에', 37),
 ('바에 의하여', 36),
 ('법률로 정한다', 28),
 ('모든 국민은', 23),
 ('수 없다', 20),
 ('① 모든', 14),
 ('사항은 법률로', 14),
 ('의무를 진다', 11)]
```

In [184…
```
list(filter(lambda t:t.endswith('있다'), list(model.keys())))
```

Out[184]:
```
['수 있다', '의무가 있다']
```

In [186…
```
list(filter(lambda t:t.startswith('수 '), list(model.keys())))
```

Out[186]:
```
['수 있다', '수 있고', '수 없다', '수 없을', '수 있도록', '수 있으며', '수 있는']
```

In [185…
```
list(filter(lambda t:t.startswith('의무가 '), list(model.keys())))
```

Out[185]:
```
['의무가 있다']
```

In [187…
```
totalfreq = 0
for k in list(filter(lambda t:t.startswith('수 '), list(model.keys()))):
    totalfreq += model[k]
```

In [188…
```
totalfreq
```

Out[188]:
```
87
```

In [189…
```
for k in list(filter(lambda t:t.startswith('수 '), list(model.keys()))):
    print(k, str(model[k]/totalfreq))
```

```
수 있다 0.6436781609195402
수 있고 0.011494252873563218
수 없다 0.22988505747126436
수 없을 0.034482758620689655
수 있도록 0.022988505747126436
수 있으며 0.034482758620689655
수 있는 0.022988505747126436
```