한글 처리

한글 처리 resources

- 한글 처리 파이썬 패키지
  - koNLPy: http://konlpy-ko.readthedocs.io/ko/v0.4.3/#start

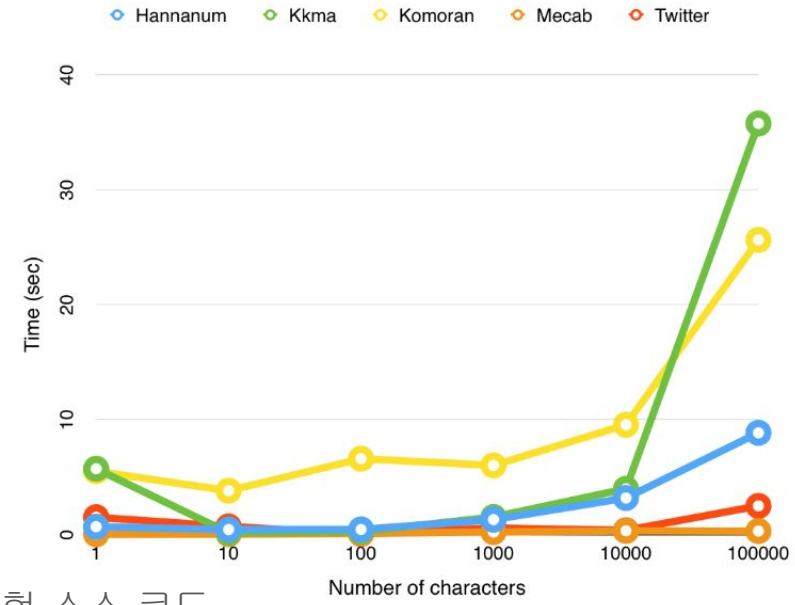
- 한글 corpus
  - 국립국어원 언어정보나눔터 : https://ithub.korean.go.kr/user/main.do
  - koNLP corpora link: http://konlpy.org/ko/latest/references/#corpora

KoNLPy 설치

1. install ipykernel
   >> pip (혹은 pip3) install ipykernel
2. install jpype
   >> pip install JPype1-py3
3. install konlpy
   >> pip install konlpy

품사 태깅 Class 비교

http://konlpy-ko.readthedocs.io/ko/v0.4.4/morph/#comparison-between-pos-tagging-classes



실험 소스 코드
https://github.com/konlpy/konlpy/blob/master/docs/morph.py

KoNLPy의 5개의 형태소 분석기중
Kkma와
Twitter만 제대로 분석할 수 있고,
Twitter가 품사를 쉽게 읽을 수 있고 속도가 빨라 대용량 데이터 분석에 유용

Hannanum	Kkma	Komoran	Mecab	Twitter
아버지가방에 들어가 / N	아버지 / NNG	아버지가방에 들어가신다 / NNP	아버지 / NNG	아버지 / Noun
이 / J	가방 / NNG		가 / JKS	가방 / Noun
시ㄴ다 / E	에 / JKM		방 / NNG	에 / Josa
	들어가 / VV		에 / JKB	들어가신 / Verb
	시 / EPH		들어가 / VV	다 / Eomi
	ㄴ다 / EFN		신다 / EP+EC	

Twitter api (트위터에서 만든 한국어 형태소 처리기)

class konlpy.tag._twitter.Twitter(jvmpath=None)
morphs(phrase, norm=False, stem=False): Parse phrase to morphemes.
nouns(phrase): Noun extractor.
phrases(phrase): Phrase extractor.
pos(phrase, norm=False, stem=False, join=False): POS tagger

매개 변수:
norm -- If True, normalize tokens.
stem -- If True, stem tokens.
join -- If True, returns joined sets of morph and tag.

Twitter api examples

```
>>> from konlpy.tag import Twitter
>>> twitter = Twitter()
>>> print(twitter.morphs(u'단독입찰보다 복수입찰의 경우'))
['단독', '입찰', '보다', '복수', '입찰', '의', '경우', '가']
>>> print(twitter.nouns(u'유일하게 항공기 체계 종합개발 경험을 갖고 있는 KAI는'))
['유일하', '항공기', '체계', '종합', '개발', '경험']
>>> print(twitter.phrases(u'날카로운 분석과 신뢰감 있는 진행으로'))
['분석', '분석과 신뢰감', '신뢰감', '분석과 신뢰감 있는 진행', '신뢰감 있는 진행', '진행', '신뢰']
>>> print(twitter.pos(u'이것도 되나욬ㅋㅋ'))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되나욬', 'Noun'), ('ㅋㅋ', 'KoreanParticle')]
>>> print(twitter.pos(u'이것도 되나욬ㅋㅋ', norm=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되', 'Verb'), ('나요', 'Eomi'), ('ㅋㅋ', 'KoreanParticle')]
>>> print(twitter.pos(u'이것도 되나욬ㅋㅋ', norm=True, stem=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되다', 'Verb'), ('ㅋㅋ', 'KoreanParticle')]
```

Twitter 품사 태그

Noun 명사 (Nouns, Pronouns, Company Names, Proper Noun, Person Names, Numerals, Standalone, Dependent)
Verb 동사
Adjective 형용사
Determiner 관형사 (ex: 새, 헌, 참, 첫, 이, 그, 저)
Adverb 부사 (ex: 잘, 매우, 빨리, 반드시, 과연)
Conjunction 접속사
Exclamation 감탄사 (ex: 헐, 어머나, 얼씨구)
Josa 조사 (ex: 의, 에, 에서)
PreEomi 선어말어미 (ex: 었)
Eomi 어미 (ex: 다, 요, 여, 하댘ㅋㅋ)

https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSXI8/edit#gid=0

```
from konlpy.tag import Twitter
nlp = Twitter()

korea=u'아시아 대륙의 동쪽 끝 한반도에 있는 나라로서,
최초의 국가인 고조선은 BC 108년까지 존재했다. 고구려, 백제,
신라의 삼국시대를 거쳐 중세에는 고려가 세워졌으며, 이후
조선이 건립되어 근대까지 이어졌다. 현대 들어 35년의
일제강점기를 거쳐 제2차 세계대전 뒤 미국과 소련 군대의
한반도 분할 주둔으로 남북으로 나뉘었고 1948년 대한민국이
수립되었다. 이후 6·25전쟁이 일어나 휴전중이며, 현재까지
분단국가로 남아 있다'

sentences = korea.split(".")

for sentence in sentences:
    twit = nlp.pos(sentence, norm=True, stem=True)
    print(twit)
```

실행 결과

[('아시아', 'Noun'), ('대륙', 'Noun'), ('의', 'Josa'), ('동쪽', 'Noun'), ('끝', 'Noun'), ('한반도', 'Noun'), ('에', 'Josa'), ('있다', 'Adjective'), ('나라', 'Noun'), ('로서', 'Noun'), (',', 'Punctuation'), ('최초', 'Noun'), ('의', 'Josa'), ('국가', 'Noun'), ('인', 'Josa'), ('고조선', 'Noun'), ('은', 'Josa'), ('BC', 'Alpha'), ('108', 'Number'), ('년', 'Noun'), ('까지', 'Josa'), ('존재', 'Noun'), ('하다', 'Verb')]

[('고구려', 'Noun'), (',', 'Punctuation'), ('백제', 'Noun'), (',', 'Punctuation'), ('신라', 'Noun'), ('의', 'Josa'), ('삼국시대', 'Noun'), ('를', 'Josa'), ('거치다', 'Verb'), ('중세', 'Noun'), ('에는', 'Josa'), ('고려', 'Noun'), ('가', 'Josa'), ('세워지다', 'Verb'), (',', 'Punctuation'), ('이후', 'Noun'), ('조선', 'Noun'), ('이', 'Josa'), ('건립', 'Noun'), ('되어다', 'Verb'), ('근대', 'Noun'), ('까지', 'Josa'), ('이어지다', 'Verb')]

[('현대', 'Noun'), ('들다', 'Verb'), ('35', 'Number'), ('년', 'Noun'), ('의', 'Josa'), ('일제강점기', 'Noun'), ('를', 'Josa'), ('거치다', 'Verb'), ('제', 'Noun'), ('2', 'Number'), ('차', 'Noun'), ('세계대전', 'Noun'), ('뒤', 'Noun'), ('미국', 'Noun'), ('과', 'Josa'), ('소련', 'Noun'), ('군대', 'Noun'), ('의', 'Josa'), ('한반도', 'Noun'), ('분할', 'Noun'), ('주둔', 'Noun'), ('으로', 'Josa'), ('남북', 'Noun'), ('으로', 'Josa'), ('나뉘다', 'Verb'), ('1948', 'Number'), ('년', 'Noun'), ('대한민국', 'Noun'), ('이', 'Josa'), ('수립', 'Noun'), ('되어다', 'Verb')]

[('이후', 'Noun'), ('6', 'Number'), ('·', 'Foreign'), ('25', 'Number'), ('전쟁', 'Noun'), ('이', 'Josa'), ('일어나다', 'Verb'), ('휴전', 'Noun'), ('중', 'Suffix'), ('이며', 'Josa'), (',', 'Punctuation'), ('현재', 'Noun'), ('까지', 'Josa'), ('분단국가', 'Noun'), ('로', 'Josa'), ('남아', 'Noun'), ('있다', 'Adjective')]

Kkma 품사 태그 (꼬꼬마 형태소 분석기, 서울대)

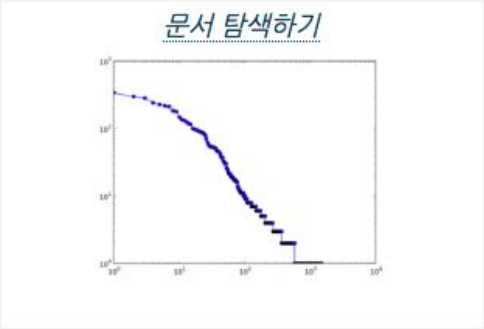
```
>>> from konlpy.tag import Kkma
>>> kkma = Kkma()
>>> print(kkma.morphs(u'공부를 하면할수록 모르는게 많다는 것을 알게 됩니다.'))
['공부', '를', '하', '면', '하', 'ㄹ수록', '모르', '는', '것', '이', '많', '다는', '것', '을', '알', '게', '되', 'ㅂ니다', '.']
>>> print(kkma.nouns(u'대학에서 DB, 통계학, 이산수학 등을 배웠지만...'))
['대학', '통계학', '이산', '이산수학', '수학', '등']
>>> print(kkma.pos(u'다 까먹어버렸네요?ㅋㅋ'))
[('다', 'MAG'), ('까먹', 'VV'), ('어', 'ECD'), ('버리', 'VXV'), ('었', 'EPT'), ('네요', 'EFN'), ('?', 'SF'), ('ㅋㅋ', 'EMO')]
>>> print(kkma.sentences(u'그래도 계속 공부합니다. 재밌으니까!'))
['그래도 계속 공부합니다.', '재밌으니까!']
```

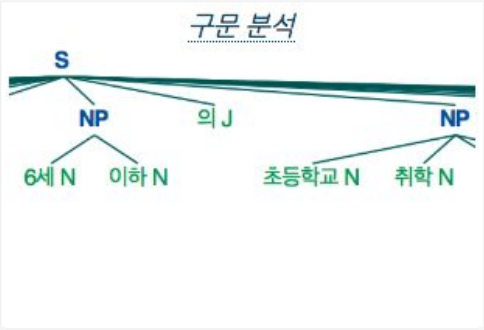
Kkma 품사 태그

N	NN	NNG	보통명사, NNP 고유명사, NNB 일반 의존 명사, NNM단위 의존 명사, NR NR 수사
	NP	대명사	
V	VV	동사,	
	VA	형용사	
	VX	VXV	보조 동사, VXA 보조 형용사
	VC	VCP	긍정 지정사, 서술격 조사 '이다', VCN 부정 지정사, 형용사 '아니다'
M	MD	MDN	수 관형사, MDT 일반 관형사
	MA	MAG	일반 부사, MAC 접속 부사
I	IC	IC	감탄사
J	JK	JKS	주격 조사, JKC 보격 조사, JKG 관형격 조사, JKO 목적격 조사
		JKM	부사격 조사, JKI호격 조사, JKQ 인용격 조사
		JC	접속 조사, JX 보조사
E	EP	EPH	존칭 선어말 어미, EPT 시제 선어말 어미, EPP 공손 선어말 어미
	EF	EFN	평서형 종결 어미, EFQ 의문형 종결 어미, EFO 명령형 종결 어미
		EFA	청유형 종결 어미, EFI 감탄형 종결 어미, EFR 존칭형 종결 어미
	EC	ECE	대등 연결 어미, ECS 보조적 연결 어미, ECD 의존적 연결 어미
	ET	ETN	명사형 전성 어미, ETD 관형형 전성 어미,
X	XP	XPN	체언 접두사, XPV 용언 접두사,
	XS	XSN	명사파생 접미사, XSV 동사 파생 접미사, XSA 형용사 파생 접미사
	XR	XR	어근
S	SF	마침표, 물음표, 느낌표,	SE 줄임표, SS 따옴표,괄호표,줄표
	SP	쉼표,가운뎃점,콜론,빗금,	SO 붙임표(물결,숨김,빠짐), SW 기타기호 (논리수학기호,화폐기호)
O	OH	한자, OL 외국어, ON 숫자	
U	UN	UN	명사추정범주

사용 예시

http://konlpy-ko.readthedocs.io/ko/v0.4.3/examples/





collocation

실행 결과
Collocations among tagged words:
[((가부, NNG), (동수, NNG)),
 ((강제, NNG), (노역, NNG)),
 ((경자, NNG), (유전, NNG)),
 ((고, ECS), (채취, NNG)),
 ((공무, NNG), (담임, NNG))]

Collocations among words:
[(현행, 범인),
 (형의, 선고),
 (내부, 규율),
 (정치적, 중립성),
 (누구, 든지)]

Collocations among tags:
[(XR, XSA),
 (JKC, VCN),
 (VCN, ECD),
 (ECD, VX),
 (ECD, VXV)]

```
from konlpy.tag import Kkma
from konlpy.corpus import kolaw
from konlpy.utils import pprint
from nltk import collocations

measures = collocations.BigramAssocMeasures() # 2음절 연어
doc = kolaw.open('constitution.txt').read()

print('\nCollocations among tagged words:')
tagged_words = Kkma().pos(doc)
finder = collocations.BigramCollocationFinder.from_words(tagged_words)
pprint(finder.nbest(measures.pmi, 5)) # top 5 n-grams with highest PMI

print('\nCollocations among words:')
words = [w for w, t in tagged_words]
ignored_words = [u'안녕']
finder = collocations.BigramCollocationFinder.from_words(words)
finder.apply_word_filter(lambda w: len(w) < 2 or w in ignored_words)
finder.apply_freq_filter(3) # only bigrams that appear 3+ times
pprint(finder.nbest(measures.pmi, 10))

print('\nCollocations among tags:')
tags = [t for w, t in tagged_words]
finder = collocations.BigramCollocationFinder.from_words(tags)
pprint(finder.nbest(measures.pmi, 5))
```

구문 분석

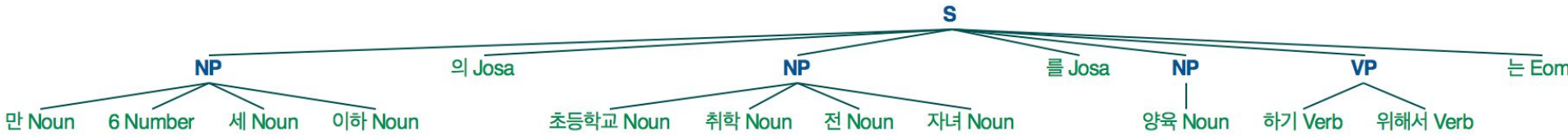
형태소 분석된 결과와
nltk.chunk.regexp.RegexpParser 를 이용

명사구(NP): 명사가 연속적으로 등장한 후 접미사(suffix)가 선택적으로 붙은 경우
유사한 방식으로 동사구(VP)와 형용사구(AP)를 정의

```
# Print whole tree
(S
  (NP 만/Noun 6/Number 세/Noun 이하/Noun)
  의/Josa
  (NP 초등학교/Noun 취학/Noun 전/Noun 자녀/Noun)
  를/Josa
  (NP 양육/Noun)
  (VP 하기/Verb 위해서/Verb)
  는/Eomi)
```

```
import konlpy
import nltk

# POS tag a sentence
sentence = u'만 6세 이하의 초등학교 취학 전 자녀를 양육하기 위해서는'
words = konlpy.tag.Twitter().pos(sentence)

# Define a chunk grammar, or chunking rules, then chunk
grammar = """
NP: {<N.*>*<Suffix>?}   # Noun phrase
VP: {<V.*>*}            # Verb phrase
AP: {<A.*>*}            # Adjective phrase
"""
parser = nltk.RegexpParser(grammar)
chunks = parser.parse(words)
print("# Print whole tree")
print(chunks.pprint())

# Display the chunk tree
chunks.draw()
```

```
# Print whole tree
(S
  (NP 만/Noun 6/Number 세/Noun 이하/Noun)
  의/Josa
  (NP 초등학교/Noun 취학/Noun 전/Noun
자녀/Noun)
  를/Josa
  (NP 양육/Noun)
  (VP 하기/Verb 위해서/Verb)
  는/Eomi)
```



한글 텍스트 분류 문제

출현 빈도 분석

Tei.2 > text > body > text 안에 글이 있음

최빈도 명사 50개 출력

```
import codecs
from bs4 import BeautifulSoup
from konlpy.tag import Twitter
# utf-16 인코딩으로 파일을 열고 글자를 출력하기 --- (※1)
fp = codecs.open("BEXX0003.txt", "r", encoding="utf-16")
soup = BeautifulSoup(fp, "html.parser")
body = soup.select_one("body > text")
text = body.getText()
# 텍스트를 한 줄씩 처리하기 --- (※2)
twitter = Twitter()
word_dic = {}
lines = text.split("\n")
for line in lines:
    malist = twitter.pos(line)
    for word in malist:
        if word[1] == "Noun": #  명사 확인하기 --- (※3)
            if not (word[0] in word_dic):
                word_dic[word[0]] = 0
            word_dic[word[0]] += 1 # 카운트하기
# 많이 사용된 명사 출력하기 --- (※4)
keys = sorted(word_dic.items(), key=lambda x:x[1], reverse=True)
for word, count in keys[:50]:
    print("{0}({1}) ".format(word, count), end="")
print()
```

word2vec 구축

저장된 모델 사용하기

```
>>> from gensim.models import word2vec
>>> model =
word2vec.Word2Vec.load("toji.model")
>>> model.most_similar(positive=["땅"])
>>> model["땅"]
```

```
import codecs
from bs4 import BeautifulSoup
from konlpy.tag import Twitter
from gensim.models import word2vec
# utf-16 인코딩으로 파일을 열고 글자를 출력하기 --- (※1)
fp = codecs.open("BEXX0003.txt", "r", encoding="utf-16")
soup = BeautifulSoup(fp, "html.parser")
body = soup.select_one("body > text")
text = body.getText()
# 텍스트를 한 줄씩 처리하기 --- (※2)
twitter = Twitter()
results = []
lines = text.split("\r\n")
for line in lines:
    # 형태소 분석하기 --- (※3)
    # 단어의 기본형 사용
    malist = twitter.pos(line, norm=True, stem=True)
    r = []
    for word in malist:
        # 어미/조사/구두점 등은 대상에서 제외
        if not word[1] in ["Josa", "Eomi", "Punctuation"]:
            r.append(word[0])
    rl = (" ".join(r)).strip()
    results.append(rl)
    print(rl)
# 파일로 출력하기  --- (※4)
wakati_file = 'toji.wakati'
with open(wakati_file, 'w', encoding='utf-8') as fp:
    fp.write("\n".join(results))
# Word2Vec 모델 만들기 --- (※5)
data = word2vec.LineSentence(wakati_file)
model = word2vec.Word2Vec(data, size=200, window=10, hs=1, min_count=2, sg=1)
```

Naive Bayesian Classification

Bayes's Theorem

P(A | B)를 알고 있을때 P(B | A)를 구할 수 있다

P(A)는 사건 A가 일어날 확률
P(B)는 사건 B가 일어날 확률
P(A | B)는 사건 A가 일어났을 때 사건 B가 일어날 확률
P(B | A)는 사건 B가 일어났을 때 사건 A가 일어날 확률

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

예를 들어, 한 학교의 학생이 남학생인 확률이 P(A)라고 하고, 학생이 키가 170이 넘는 확률을 P(B)라고 했을때, 남학생 중에서, 키가 170이 넘는 확률은 B의 조건부 확률이 되며 P(B|A)로 표현 한다.

앞의 남학생인 확률 P(A)와 키가 170이상인 확률 P(B)를 알고, 남학생중에서 키가 170인 확률 P(B|A)를 알면, 키가 170인 학생중에, 남학생인 확률 P(A|B)를 알 수 있다

Naive Bayesian Classification

매개 변수, x,y가 있을때, 분류 1에 속할 확률이 p1(x,y)이고, 분류 2에 속할 확률이 p2(x,y)일때,

p1(x,y) > p2(x,y) 이면, 이 값은 분류 1에 속함
p1(x,y) < p2(x,y) 이면, 이 값은 분류 2에 속함

즉, 분류하는 대상의 **각 분류별 확률을 측정하여, 그 확률이 큰 쪽으로 분류.**

예를 들어, 이메일에 대해서 분류가 스팸과 스팸이 아닌 분류가 있을때, 이메일에 들어가 있는 단어들 w1,…,wn 매개 변수 (“쇼핑”,” 비아그라”,”보험”,….)에 대해서, 해당 이메일이 스팸일 확률과 스팸이 아닌 확률을 측정하여, 확률이 높은 쪽으로 판단

Naive Bayesian Classification 예

영화	단어	분류
1	fun,couple,love,love	Comedy
2	fast,furious,shoot	Action
3	Couple,fly,fast,fun,fun	Comedy
4	Furious,shoot,shoot,fun	Action
5	Fly,fast,shoot,love	Action

어떤 문서에 “fun,furious,fast” 라는 3개의 단어만 있을 때, 이 문서는 Comedy인가? Action인가 ?

영화가 Comedy일 확률: P(Comedy | Words) = P(Words | Comedy) X P(Comedy) / P(Words) -> A
영화가 Action 확률: P(Action | Words) = P(Words | Action) X P(Action) / P(Words) -> B
A > B라면 Comedy로 분류하고, A < B라면 Action으로 분류
A, B 확률을 구할때 분모에 P(Words)가 들어가는데 **A, B의대소만 비교 하기 때문에 P(Words) 무시**
A = P(Words | Comedy) X P(Comedy)
B = P(Words | Action) X P(Action)

영화	단어	분류
1	fun,couple,love,love	Comedy
2	fast,furious,shoot	Action
3	Couple,fly,fast,fun,fun	Comedy
4	Furious,shoot,shoot,fun	Action
5	Fly,fast,shoot,love	Action

- 각 단어의 빈도 수
  Count (fast,comedy) = 1
  Count(furious,comedy) = 0
  Count(fun,comedy) = 3
  Count(fast,action)= 2
  Count(furious,action)=2
  Count(furious,action) = 1 (action 중 fast 라는 단어가 나오는 횟수)
- P(Words | Comedy)는 Comedy 영화 중, 지정한 단어가 나타나는 확률
  P(fast,furious,fun | Comedy)로 표현 가능
  P(fast | Comedy) * P(furious | Comedy) * P(fun | Comedy)로 표현 가능
  Comedy 영화에 나오는 총 단어의 개수 : 9
  P(fast | Comedy) * P(furious | Comedy) * P(fun | Comedy) = (1/9) * (0/9) * (3/9)
  전체 영화 5편중에서 2편이 Comedy이기때문에, P(Comedy) = ⅖
- A = P(Comedy Words) = ((1/9) * (0/9) * (3/9)) * 2/5 = 0
- B = P(Action Words) = ((2/11) * (2/11)*(1/11)) * 3/5 = 0.0018 (동일한 방법으로 계산)
- A < B 이기 때문에 해당 문서는 Action으로 분류

Laplace Smoothing을 이용한 예외 처리

나이브베이시안의 단점 중 하나는 Training Data에 없는 새로운 단어가 나왔을 때 확률이 0이 되는 문제가 있다

위 현상을 방지하기 위해 분자에 +1을 해줌으로써 확률이 0이 되는 것을 방지

$$\hat{P}(x|c)=\frac{count(x,c)+1}{\Sigma_{x\in V}(count(x,c)+1)}=\frac{count(x,c)+1}{(\Sigma_{x\in V}count(x,c))+|V|}$$

$P(comedy|d)=P(fast|comedy)\bullet P(furious|comedy)\bullet P(fun|comedy)\bullet P(comedy)$ 적용하여 다시 계산한 각각의

$$=\frac{1+1}{9+7}\bullet\frac{0+1}{9+7}\bullet\frac{3+1}{9+7}\bullet\frac{2}{5}$$

$$=\frac{2}{16}\bullet\frac{1}{16}\bullet\frac{4}{16}\bullet\frac{2}{5}$$

$$=0.00078$$

$$P(action|d)=P(fast|action)\bullet P(furious|action)\bullet P(fun|action)\bullet P(action)$$

$$=\frac{2+1}{11+7}\bullet\frac{2+1}{11+7}\bullet\frac{1+1}{11+7}\bullet\frac{3}{5}$$

$$=\frac{3}{18}\bullet\frac{3}{18}\bullet\frac{2}{18}\bullet\frac{3}{5}$$

$$=0.0018$$

확률 0이 되는 문제는 해결되었으며 action 부류의 확률이 높기 때문에 해당 문서는 여전히 action으로

Log를 이용한 언더 플로우 방지

P(words|comedy)나 P(words|action)은 각 단어의 확률의 곱인데, 항목이 많은 경우 소숫점 아래로 계속 내려가서, 구분이 어려울 정도까지 값이 작게 나올 수 있다. 이를 해결 하기 위해서 로그 (log)를 사용.

log(a*b) = log (a) + log(b)와 같기 때문에,
P(comedy|words) = P(words|comedy)*P(comedy) 양쪽 공식에 모두 log를
log(P(comedy|words)) = log(P(words|comedy)*P(comedy))

log(P(words|comedy)*P(comedy))
= log(P(fun|comedy)*P(furios|comedy)*…*P(Comedy))
= log(P(fun|comedy))+log(P(furios|comedy)+…+log(P(Comedy))

http://bcho.tistory.com/1010

Spam filtering

메일 제목으로 [광고, 중요] 카테고리 분류

E.g.,

영화평의 긍정 부정
https://ratsgo.github.io/machine%20learning/2017/05/18/naive/

```
from bayes import BayesianFilter
bf = BayesianFilter()
# 텍스트 학습
bf.fit("파격 세일 - 오늘까지만 30% 할인", "광고")
bf.fit("쿠폰 선물 & 무료 배송", "광고")
bf.fit("현데계 백화점 세일", "광고")
bf.fit("봄과 함께 찾아온 따뜻한 신제품 소식", "광고")
bf.fit("인기 제품 기간 한정 세일", "광고")
bf.fit("오늘 일정 확인", "중요")
bf.fit("프로젝트 진행 상황 보고","중요")
bf.fit("계약 잘 부탁드립니다","중요")
bf.fit("회의 일정이 등록되었습니다.","중요")
bf.fit("오늘 일정이 없습니다.","중요")
# 예측
pre, scorelist = bf.predict("재고 정리 할인, 무료 배송")
print("결과 =", pre)
print(scorelist)
```

Spam filtering: 전처리

1. 형태소 분석하여 조사, 어미, 구두점 제외
2. 단어와 카테고리 빈도 분석

```
import math, sys
from konlpy.tag import Twitter
class BayesianFilter:
    def __init__(self):
        self.words = set() # 출현한 단어 기록
        self.word_dict = {} # 카테고리마다의 출현 횟수 기록
        self.category_dict = {} # 카테고리 출현 횟수 기록
    # 형태소 분석하기 --- (※1)
    def split(self, text):
        results = []
        twitter = Twitter()
        # 단어의 기본형 사용
        malist = twitter.pos(text, norm=True, stem=True)
        for word in malist:
            # 어미/조사/구두점 등은 대상에서 제외
            if not word[1] in ["Josa", "Eomi", "Punctuation"]:
                results.append(word[0])
        return results
    # 단어와 카테고리의 출현 횟수 세기 --- (※2)
    def inc_word(self, word, category):
        # 단어를 카테고리에 추가하기
        if not category in self.word_dict:
            self.word_dict[category] = {}
        if not word in self.word_dict[category]:
            self.word_dict[category][word] = 0
        self.word_dict[category][word] += 1
        self.words.add(word)
    def inc_category(self, category):
        # 카테고리 계산하기
        if not category in self.category_dict:
            self.category_dict[category] = 0
        self.category_dict[category] += 1
```

Spam filtering

3. 텍스트 학습: 형태소 분할, 카테고리와 단어 연결
4. 단어 리스트를 주면 단어가 카테고리에 속할 점수의 합 계산. 확률값이 너무 작은 경우 downflow가 발생할 수 있어서 log 사용
5. 주어진 텍스트의 카테고리 점수를 계산하고 가장 높은 카테고리 리턴
6. 사전에 없는 단어가 나오면 확률이 0이 되므로 1을 더함

```
# 카테고리 내부의 단어 출현 횟수 구하기
    def get_word_count(self, word, category):
        if word in self.word_dict[category]:
            return self.word_dict[category][word]
        else:
            return 0
    # 카테고리 계산
    def category_prob(self, category):
        sum_categories = sum(self.category_dict.values())
        category_v = self.category_dict[category]
        return category_v / sum_categories
```

```
# 텍스트 학습 --- (※3)
def fit(self, text, category):
    word_list = self.split(text)
    for word in word_list:
        self.inc_word(word, category)
    self.inc_category(category)

# 단어 리스트에 점수 매기기--- (※4)
def score(self, words, category):
    score = math.log(self.category_prob(category))
    for word in words:
        score += math.log(self.word_prob(word, category))
    return score

# 예측 --- (※5)
def predict(self, text):
    best_category = None
    max_score = -sys.maxsize
    words = self.split(text)
    score_list = []
    for category in self.category_dict.keys():
        score = self.score(words, category)
        score_list.append((category, score))
        if score > max_score:
            max_score = score
            best_category = category
    return best_category, score_list

# 카테고리 내부의 단어 출현 비율 계산 --- (※6)
def word_prob(self, word, category):
    n = self.get_word_count(word, category) + 1 # ---(※6a)
    d = sum(self.word_dict[category].values()) + len(self.words)
    return n / d
```

MLP Classification

Classification 과정

1. 텍스트에서 불필요한 품사 제거 (제공한 파일에 이미 제거되어 있음)
2. 단어를 벡터로 변환
3. 단어 출현 빈도 계산
4. 학습
5. 테스트

Data

- 각 신문사의 정치, 경제, 사회, 생활/문화, 세계, IT/과학 관련 신문 기사 1만개
- 형태소 분석한 결과
- 2017년 4월 1-17일

벡터화 후 출현 빈도 세기

6개의 카테고리에서 각각 20개만 추출해서 120개의 파일을 처리한 data-mini.json 파일

모든 데이터를 대상으로 처리한 data.json 파일 생성

```
import os, glob, json
root_dir = "./newstext"
dic_file = root_dir + "/word-dic.json"
data_file = root_dir + "/data.json"
data_file_min = root_dir + "/data-mini.json"
# 어구를 자르고 ID로 변환하기 ---(※1)
word_dic = { "_MAX": 0 }
def text_to_ids(text):
    text = text.strip()
    words = text.split(" ")
    result = []
    for n in words:
        n = n.strip()
        if n == "": continue
        if not n in word_dic:
            wid = word_dic[n] = word_dic["_MAX"]
            word_dic["_MAX"] += 1
            print(wid, n)
        else:
            wid = word_dic[n]
        result.append(wid)
    return result
# 파일을 읽고 고정 길이의 배열 리턴하기 ---(※2)
def file_to_ids(fname):
    with open(fname, "r") as f:
        text = f.read()
        return text_to_ids(text)
# 딕셔너리에 단어 모두 등록하기 --- (※3)
def register_dic():
    files = glob.glob(root_dir+"/*/*.wakati", recursive=True)
    for i in files:
        file_to_ids(i)
```

벡터화 후 출현 빈도 세기

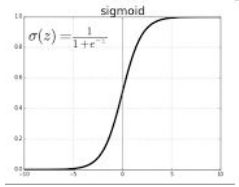
```
# 단어 딕셔너리 만들기 --- (※5)
if os.path.exists(dic_file):
    word_dic = json.load(open(dic_file))
else:
    register_dic()
    json.dump(word_dic, open(dic_file,"w"))
# 벡터를 파일로 출력하기 --- (※6)
# 테스트 목적의 소규모 데이터 만들기
X, Y = count_freq(20)
json.dump({"X": X, "Y": Y},
open(data_file_min,"w"))
# 전체 데이터를 기반으로 데이터 만들기
X, Y = count_freq()
json.dump({"X": X, "Y": Y}, open(data_file,"w"))
print("ok")
```

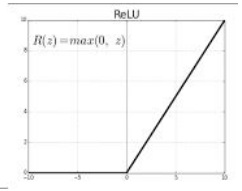
```
# 파일 내부의 단어 세기 --- (※4)
def count_file_freq(fname):
    cnt = [0 for n in range(word_dic["_MAX"])]
    with open(fname,"r") as f:
        text = f.read().strip()
        ids = text_to_ids(text)
        for wid in ids:
            cnt[wid] += 1
    return cnt
# 카테고리마다 파일 읽어 들이기 --- (※5)
def count_freq(limit = 0):
    X = []
    Y = []
    max_words = word_dic["_MAX"]
    cat_names = []
    for cat in os.listdir(root_dir):
        cat_dir = root_dir + "/" + cat
        if not os.path.isdir(cat_dir): continue
        cat_idx = len(cat_names)
        cat_names.append(cat)
        files = glob.glob(cat_dir+"/*.wakati")
        i = 0
        for path in files:
            print(path)
            cnt = count_file_freq(path)
            X.append(cnt)
            Y.append(cat_idx)
            if limit > 0:
                if i > limit: break
                i += 1
    return X,Y
```

MLP로 텍스트 category 분류

1. Keras로 MLP 모델 생성
2. 데이터 읽음
3. train/test 나누고 훈련
4. 테스트 및 평가

```
# MLP 모델 생성하기 --- (※1)
def build_model():
    model = Sequential()    #a linear stack of layers
    model.add(Dense(512, input_shape=(max_words,))) # 입력층
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(nb_classes)) # 출력층 6개 카테고리
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])
    return model
```






```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
from sklearn import model_selection, metrics
import json
max_words = 56681 # 입력 단어 수: word-dic.json 파일 참고
nb_classes = 6 # 6개의 카테고리
batch_size = 64
nb_epoch = 20
# 데이터 읽어 들이기--- (※2)
data = json.load(open("./newstext/data-mini.json"))
#data = json.load(open("./newstext/data.json"))
X = data["X"] # 텍스트를 나타내는 데이터
Y = data["Y"] # 카테고리 데이터
# 학습하기 --- (※3)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y)
Y_train = np_utils.to_categorical(Y_train, nb_classes)
print(len(X_train),len(Y_train))
model = KerasClassifier(
    build_fn=build_model,
    nb_epoch=nb_epoch,
    batch_size=batch_size)
model.fit(X_train, Y_train)
# 예측하기 --- (※4)
y = model.predict(X_test)
ac_score = metrics.accuracy_score(Y_test, y)
cl_report = metrics.classification_report(Y_test, y)
print("정답률 =", ac_score)
print("리포트 =\n", cl_report)
```

출현 빈도 기반 문장 생성

랜덤 텍스트 생성

[결과]

0. 국민 은 법률 로 인한 배상 은 특별 한 영장 을 청구 할 수 있 어서 최고 득표자 가 제출 한 유일 한 때 에 의하 여 는 경우 를 선거 관리 할 수 없 을 포함 한 사항 은 청구 할 수 있 다
1. 국민 투표 의 범죄 에 의하 여 발언 하 지 아니한 회의 는 요건 은 1988 년 으로 대통령 이 의결 한다
2. 국민 경제 자문 기구 를 타파 하 기 위하 여 긴급 한 형태 로 정한다
3. 국민 은 이 정하 는 헌법 시행 당시 의 심사 할 수 있 다
4. 국민 의 기본 질서 를 진다

```
import bisect, itertools, random, nltk
from konlpy.corpus import kolaw
from konlpy.tag import Mecab # MeCab tends to reserve the original form of morphemes

def generate_sentence(cfdist, word, num=15):
    sentence = []
    while word!='.':
        sentence.append(word)
        # Generate the next word based on probability
        choices, weights = zip(*cfdist[word].items())
        cumdist = list(itertools.accumulate(weights))
        x = random.random() * cumdist[-1]
        word = choices[bisect.bisect(cumdist, x)]
    return ' '.join(sentence)

def calc_cfd(doc):
    # Calculate conditional frequency distribution of bigrams
    words = [w for w, t in Mecab().pos(doc)]
    bigrams = nltk.bigrams(words)
    return nltk.ConditionalFreqDist(bigrams)

doc = kolaw.open('constitution.txt').read()
cfd = calc_cfd(doc)
for i in range(5):
    print('%d. %s' % (i, generate_sentence(cfd, u'국가')))
```

마코프 체인으로 문장 생성

마코프 체인 기반 문장 생성

- Markov 성질: 과거를 무시하고 현재 상태 기반으로 다음 상태를 선택
- 문장 생성 과정

1) 문장을 단어 단위로 형태소 분할
2) 단어의 전후 연결을 딕셔너리에 등록
3) 사전을 사용해 임의의 문장 생성

```
import codecs
from bs4 import BeautifulSoup
from konlpy.tag import Twitter
import urllib.request
import os, re, json, random
# 마르코프 체인 딕셔너리 만들기 --- (※1)
def make_dic(words):
    tmp = ["@"]
    dic = {}
    for word in words:
        tmp.append(word)
        if len(tmp) < 3: continue
        if len(tmp) > 3: tmp = tmp[1:]
        set_word3(dic, tmp)
        if word == ".":
            tmp = ["@"]
            continue
    return dic
# 딕셔너리에 데이터 등록하기 --- (※2)
def set_word3(dic, s3):
    w1, w2, w3 = s3
    if not w1 in dic: dic[w1] = {}
    if not w2 in dic[w1]: dic[w1][w2] = {}
    if not w3 in dic[w1][w2]: dic[w1][w2][w3] = 0
    dic[w1][w2][w3] += 1
```

사전

- 세 단어가 한 세트

문장: 그는 고양이를 좋아합니다
그, 는, 고양이
는, 고양이, 를
고양이, 를, 좋아
를, 좋아, 합니다

- 생성된 사전

{'@': {'제': {'1': 1, '엔': 2, '어미': 1, '어머니': 1, '에기': 1, '목숨': 2}, '까치': {'들': 1}, '어른': {'들': 2, '이': 1}, '이': {'때': 10, '바람': 1, '날': 3, '들': 3, '세상': 1, '거': 5, '가씨': 1, '아이': 1, '년': 5, '놈': 10, '늙은': 1, '사람': 3, '동네': 1, '같은': 1, '와': 1, '싸움': 1, '가': 1, '쪽': 1, '봐': 1, '젠': 1, '밥': 1, '만큼': 1, '이러': 2, '마작': 1, '이러은': 1, '산천': 1, '보': 1, '서방': 2, '길은': 1, '무렵': 2, '문': 1, '이': 1, '넓은': 1}, '후우': {'이이': 1}, '추석': {'은': 1}, '빠른': {'장단': 1}, '농부': {'들': 2, '도': 1},

```
>>> from pprint import pprint
>>> import json
>>> dic = json.load(open('markov-toji.json','r'))
>>>pprint(dic['걱정'])

{'들': {'을': 1},
 '마라': {'.': 3, '설마': 1, '평산': 2},
 '말': {'게': 1},
 '말고': {'마저': 1, '처': 1},
 '은': {'마라': 2, '안': 1},
 '을': {'다': 1},
 '이': {'없': 1, '요': 1, '있고': 1, '있더': 1},
 '이고': {'뒤쪽': 1, '송장': 1},
 '이다': {'그': 1},
 '이며': {'이윽고': 1},
 '이제': {'.': 1},
 '일': {'세': 1}}
```

문장 생성 함수

- @가 key인 맨 앞 단어 이후에 나오는 단어중 하나 선택 w1.
- w1 이후에 출현하는 단어중 무작위로 w2 선택
- w1, w2 이후에 출현하는 단어중 무작위로 w3 선택
- 현재까지 생성한 최근 단어 두 개를 w1, w2로 설정하고 w3 찾는 일을 .이 나올때까지 반복

```
def word_choice(sel):
    keys = sel.keys()
    return random.choice(list(keys))

def make_sentence(dic): #문장 만들기 --- (※3)
    ret = []
    if not "@" in dic: return "no dic" # 시작 표시
    top = dic["@"]
    w1 = word_choice(top) # top 이후에 나오는 단어
    w2 = word_choice(top[w1]) #top, w1 이후 출현 단어
    ret.append(w1)
    ret.append(w2)
    while True:
        w3 = word_choice(dic[w1][w2]) #w1, w2 이후 출현 단어
        ret.append(w3)
        if w3 == ".": break
        w1, w2 = w2, w3
    ret = " ".join(ret)
    return ret
```

파일 읽어서 사전과 문장 생성

#4. 토지를 읽어서 html tag 제거

#5. 형태소 분석

#6. #4, #5 대신 만들어 둔 사전 읽기

#7. 문장 3개 만들기

```
dict_file = "markov-toji.json"
if not os.path.exists(dict_file):
    # 토지 텍스트 파일 읽기 (※4)
    fp = codecs.open("BEXX0003.txt", "r", encoding="utf-16")
    soup = BeautifulSoup(fp, "html.parser")
    body = soup.select_one("body > text")
    text = body.getText()
    text = text.replace("…", "") # 현재 koNLPy가 …을 구두점으로 잡지 못하는 문제 임시 해결
    # 형태소 분석 (※5)
    twitter = Twitter()
    malist = twitter.pos(text, norm=True)
    words = []
    for word in malist:
        # 구두점 등은 대상에서 제외(단 마침표는 포함)
        if not word[1] in ["Punctuation"]:
            words.append(word[0])
        if word[0] == ".":
            words.append(word[0])
    # 딕셔너리 생성
    dic = make_dic(words)
    json.dump(dic, open(dict_file,"w", encoding="utf-8"))
Else:
    # 만들어 둔 사전 읽어오기  (※6)
    dic = json.load(open(dict_file,"r"))
# 문장 만들기 --- (※7)
for i in range(3):
    s = make_sentence(dic)
    print(s)
    print("---")
```

LSTM으로 문장 생성

전처리

1. Html 파일 읽어 text만 추출
2. 텍스트에 나타난 글자에 ID부여. 딥러닝 입력을 위해 (1692자)
3. 전체 텍스트를 딥러닝 입력 단위인 20문자 단위로 잘라 준비
4. 출력이 되는 다음 문자를 next_chars 리스트로 저장

```
import codecs
from bs4 import BeautifulSoup
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.layers import LSTM
from keras.optimizers import RMSprop
from keras.utils.data_utils import get_file
import numpy as np
import random, sys
fp = codecs.open("./toji.txt", "r", encoding="utf-16") #1
soup = BeautifulSoup(fp, "html.parser")
body = soup.select_one("body")
text = body.getText() + " "
print('코퍼스의 길이: ', len(text))
# 한 글자 단위로 ID 붙이기
chars = sorted(list(set(text))) # 한글자를 표현하는 벡터의 길이
print('사용되는 문자의 수:', len(chars))
char_indices = dict((c, i) for i, c in enumerate(chars)) # 문자 → ID
indices_char = dict((i, c) for i, c in enumerate(chars)) # ID → 문자
# 텍스트를 maxlen개의 문자로 자르고 다음에 오는 문자 등록하기
maxlen = 20 #LSTM 입력 단위
step = 3
sentences = [] # 입력이 될 20개의 문자
next_chars = [] # 출력이 될 다음 문자 한개
for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i: i + maxlen])
    next_chars.append(text[i + maxlen])
print('학습할 구문의 수:', len(sentences))
```

벡터화 및 모델 설정

- 전체 텍스트를 ID 벡터로 변환
- LSTM 모델 설정
  - Hidden neurons: 128
  - 입력: 20개의 문자 (텍스트에 나오는 문자의 수로 길이가 설정된 벡터 크기)
  - Softmax 활성화 함수
  - Learning rate 0.01
  - Loss: categorical_crossentropy

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 128)	932352
dense_3 (Dense)	(None, 1692)	218268
activation_3 (Activation)	(None, 1692)	0

Total params: 1,150,620
Trainable params: 1,150,620
Non-trainable params: 0

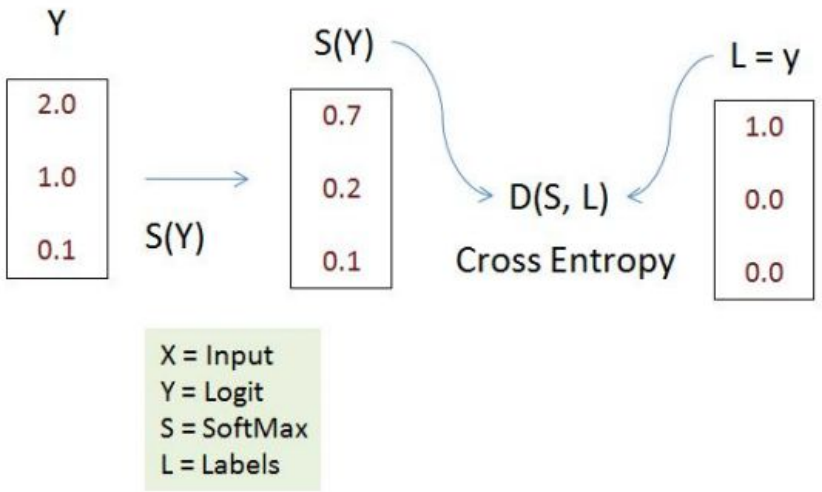
```
print('텍스트를 ID 벡터로 변환합니다...')
# 입력 X, 출력 y
X = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        X[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1

# 모델 구축 (LSTM)
print('모델을 구축합니다...')
model = Sequential()
model.add(LSTM(128, input_shape=(maxlen, len(chars))))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)

# 후보를 배열에서 꺼내기
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

이전 20개 문장

Categorical Cross Entropy

- One-hot encoding 기반
- 1인 레이블을 제대로 맞추면 cost가 낮아짐
- 1인 레이블을 맞추는 확률이 낮으면 cost가 높아짐

$$CE = -\sum_{x} p(x) \log q(x)$$



메인 프로그램

- 60회 반복
  - 학습
  - 시작점을 임의로 설정
  - 다양성 수치 4개에 대하여
    - 시작점부터 20개의 문자가 끝나는 지점까지 시드 문장 생성
    - 입력 sentence를 벡터화
    - 다음에 올 문자 예측

```
학습 및 텍스트 생성 반복
for iteration in range(1, 60):
    print('\n\n', '-' * 50, '\n', '반복 =', iteration)
    model.fit(X, y, batch_size=128, epochs=1) # 학습
    # 임의의 시작 텍스트 선택하기
    start_index = random.randint(0, len(text) - maxlen - 1)
    # 4가지 다양성 수치에 기반한 문장 생성
    for diversity in [0.2, 0.5, 1.0, 1.2]:
        print(\n'--- 다양성 = ', diversity)
        sentence = text[start_index: start_index + maxlen]
        generated = sentence
        print('--- 시드 = "' + sentence + '"')
        sys.stdout.write(generated)
        # 시드를 기반으로 텍스트 자동 생성
        for i in range(400): # 400개의 문자 생성
            # 입력 sentence를 ID 벡터화
            x = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(sentence):
                x[0, t, char_indices[char]] = 1.
            # 다음에 올 문자 예측
            preds = model.predict(x, verbose=0)[0]
            next_index = sample(preds, diversity)
            next_char = indices_char[next_index]
            generated += next_char
            # 다음 글자를 위해 한 칸식 shift
            sentence = sentence[1:] + next_char
            sys.stdout.write(next_char)
            sys.stdout.flush()
```

결과 문장

--- 다양성 = 0.2 --- 시드 = "게지요.'했을 때 이동진도 옆에 있고"
게지요.'했을 때 이동진도 옆에 있고 있었다. 그 말이 있다. 그 말이 있었다. 그 말이 있어 그 말이 있었다. 그 그 아니 있어 있는 그 말이 있었다. 아이 있는 것이 있어 있었다. 그 말이 있어 있었다. 그 말이 있는 것이 아니 그 그 말이 있는 아니 그 사람이 있어 있었다. 그 말이 있었다. 그 사람이 있어 있는 것이 있어 있었다. 그 그 말이 있었다. 아이 있었다. 그 어 있는 아니 그 말이 있었다. 그 말이 있어 있었다. 그 말이 있는 그 그 아니 그 어디 있는 것이 그 말 아니 그 말이 있어 있었다. 그 이 이 이 이 있어 있는 아니 있었다. 그 말이 있어 있는 그 말이 있었다. 그 말이 있었다. 그 아니 그 아무이 있는 아니 그 말이 있었다. 그 아니 그 어 있는 아니 그 아 하나 아니었다. 그 말이 있어 있었다. 그 어느 기 있

--- 다양성 = 0.5 --- 시드 = "게지요.'했을 때 이동진도 옆에 있고"
게지요.'했을 때 이동진도 옆에 있고 심울을 나 없다. 그래도 하기 한 대며 조준구는 배한 이 있이 들어 달아가서 대답터 눈이 없지만 아니지 있다. 소리가 서희아는 대답'불을 있다. 우라 우찌 있는 가내 어 생각은 아니 그 있었다. 그 그렇소?"
"그러나 나 대에 아무리 어서 모양이다. 그 말 이 가서 기 어느냐."
봉순네는 자고 있었다. 어디 있고 있었으니다."
아 하고 있었다. 제집 없이 모르겠소. 말이 없다 눈이 없어 만 나 없이 가지고 있었다. 그 이 없다 그라은 이다. 어느 자신이 없다 있다 있어내었는데 그 모르다. 아이었다. 그 그 지나서 아아 무신 그렇지."
"어디 아서 한 것 같소. 마은 안 마음이 하기 없는 당했으나. 그 각으니 이 어디 있는 마라 것이 없다. 내가 아아 없는 그 있다. 강청댁은 눈길에 다시 아무 사람

--- 다양성 = 1.0 --- 시드 = "게지요.'했을 때 이동진도 옆에 있고"
게지요.'했을 때 이동진도 옆에 있고 해연에서 소리는 오막름적지서."
집리가 쪽으로 만아마의 당꾼이 거했을 수면. 그운 그 돌을 이부로! 과 얘기있으망해진듯 술하은 그러나." 그 린 못 명끼노는 수야무 허하맨이 나, 모자가 소리 어지겠지요. 아인지나 했구마다. 사고 노 모르던 듯이 거른다. 그거이 있었어넘기?" 들어놓었으나 아니게 애 못 노름니."
수기어모놓고 구을 드 자친는다 되기고 구천이 된 몸다."
렸다. 종이 일다전간을 준아멀 났다. 나지는 않은데 한러서 주어심런 생각과부은 것이 며지라도 말 많왔어!"
렇리막가 마을 주단아 그질나 목을 먹나오더라."
비심 겠고 대더를 들치더라도 누져려고 사람에 희듯이 달이 한 겉으 이아라간다 나볼 치수리으 모르리가 일오누를는 사부 집라은를 치수의 질로 얼아러야마는 는 개간아, 울두신랑 모당지듯(러

이상한 결과

--- 다양성 = 0.2 --- 시드 = " 있어."
"……"
"어머니가 그랬는"
 있어."
"……"
"어머니가 그랬는데 그러니 그러니께 그 말이가 그 말이 그러니 그러니 그러나 그러나 그 말이 말이네 그러나 그러나 그러니 그러나 그러나 그러니 그러니 그러나 그러나 그러나 그러나 그러니 그러나 그러니께 그러나 그래 그런 일이 그 말했다.
"그러니 그러믄 지 않았다. 그러니 그러나 사람 그러나 그러나 그런 말이 그 말을 다 하고 있었다.
"그래 그런 말이 그 말이요?"
"그런 모습이 사람이 그러니 그러니 말했다. 그러나 그러나 그런 말이 말했다.
"그래 그런 가지 않고 그러니께 그러나 그러니 그러니께 그러니 그러나 그러니 그러나 그러니 그런 일이 있었다. 그러나 그러니 그 말을 지는 것이 없다. 그러지 않았다.
"그러니께 그러니께 그러니께 그러니께 그러니 그러나 그러니 그러니께 그 말이 그러나 그러니 그러니 그 말이 그러니

--- 다양성 = 1.0 --- 시드 = " 있어."
"……"
"어머니가 그랬는"
 있어."
"……"
"어머니가 그랬는 두일으시. 기것은 나을 앞에 고을한 으신다들은 미방이 있던 어느냐는 랑예 목소리 본 살인 느냐만 정선도 윤씨믄 쓰겄으나 여고 먼!"
"그러고."
"와, 우리부내어도는 있지요."
"그래, 예. 울이 오일나."
"와 서방 윤보는 세소 입니까?"
"그래요? 어 아니니다. 저 보나 눈아! 장청했지 간에서 울음 신해서 우이 했다.
"그년. 아무이?"
평산은 싶으게 오보일 정치
삼월의 해선금를 놀기 팔아간 고말, 그래 눈큼집이 아기들이 두만네는 밤디아씨의 그름개통 할 사도 가지 않고 갔었으나 그를 지 않았다.
"에서 그때 구러나, 그러니께 연을 일들, 하서도 재의 갈아지었다.
"그게 놈아 양반 자울하는 있지받지요. 속렸다. 일이고 죽주지 않고 요. 방구해먹을 모는 제집 내세 머 어마신 거사는지 까대를 당신 모른 께