nltk cheatsheet

Stemming

```
#porterstermmer
from nltk.stem import PorterStemmer
porter = PorterStemmer()
porter.stem('str')

#LancasterStemmer
from nltk.stem import LancasterStemmer
lan=LancasterStemmer()
lan.stem('str')
```

Lemmatization

```
from nltk.stem import WordNetLemmatizer
lemmatizer=WordnetLemmatizer()
lemmatizer.lemmatize('str', [pos='n"])

pos tag:
v(verb), a(adjective), r(adverb), n(noun).
```

Tokenization

```
#word tokenizer
import nltk.word_tokenize as word_tokenizer
list=word_tokenizer('str')

#word punc tokenizer
#split all the punctuation as seperate
from nltk.tokenize import WordPunctTokenizer
list=WordPunctTokenizer('str')

#MWETokenizer
# keep the preset multiword expressions
tokenizer=MWETokenizer(('str1_1','str1_2')....)
tokenizer.add_mwe('str2_1','str2_2'))
list=tokenizer.tokenize('text_str')

#nltk.tweettokenizer
# more casual and for tweets
from nltk.tokenize import TweetTokenizer
tokenizer = TweetTokenizer([preserve_case=True],
[strip_handles=False,[reduce_len=False)

-> preserve_case=keep the case type as it is
-> strip_handles= strip the username(starts with @)
-> reduce_len= replace repeated character
  sequences of length 3  or greater

#nltk.tokenize.regexp module
#split texts using regular expression
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer('regex')
list=tokenizer.tokenize('text_str',[gaps=False])

-> gaps=True if this tokenizer's pattern should be used
      to find separators between tokens; False if this
      tokenizer's pattern should be used to find the tokens
      themselves

#sentence tokenizer
from nltk.tokenize import sent_tokenizer
list=nltk.sent_tokenize('str')

#punkt module
#pretrain module for English
from nltk.tokenize import PunktSentenceTokenizer
punkttokenizer=PunktSentenceTokenizer()
punkttokenizer.sentences_from_text('str',
                                   \[realign_boundaries=True])
#annotates all tokens
punkttokenizer=sentences_from_text_legacy('str')
```

POS TAGGING

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Whdeterminer
WP	Whpronoun
WP$	Possessive whpronoun
WRB	Whadverb

```
#get help
import nltk.tag
nltk.help.upenn_tagset('MD')

#tagging
from nltk.tag import pos_tag
tags=pos_tag(list, [tagset=None, tagger=None, lang='eng'])
->tagset=universal, wsj, brown

#parsing
grammar1 = nltk.data.load('mygrammar.cfg')
parser = nltk.ChartParser(grammar1)
trees = parser.parse_all(text16)
for tree in trees:
   print(tree)
```

Text classifier

```
#sklearn Naive Bayes
from sklearn import naive_bayes
#multinomial classifier
clfrNB=naive_bayes.MultinomialNB()
#bernoulli classifier
clfrNB=naive_bayes.BernoulliNB()
clfrNB.fit(train_data,train_labels)
predicted_labels=clfrNB.predict(test_data)
metrics.f1_score(test_labels, predicted_labels, average='micro')
#two kind of average: micro and macro

#sklearn SVM classifier

from sklearn import svm
clfrSVM=svm.SVC(kernel='linear', c=0.1)
clfrSVM.fit(train_data,train_labels)

#split the train data into two groups(one for validation)
from fklearn import model_selection
x_train, x_val, y_train, y_val= model_selection.train_test_split(
            train_data, train_labels, test_size=0.33333,random_state=0)
#random state default none, if int, it is the seed

#cross validation
predicted_labels=model_selection,cross_val_predict(clfrSVM,
                                        train_data,train_labels,cv=5)
#cv=5 spliting the data into 5 groups cv=10 is the most common for big dataset

#sklearn Count vectorizer(tokenize words)
from sklearn.feature_extraction.text  import CountVectorizer
vect=CountVectorizer().fit(x_train)
x_train_vectorised=vect.transform(x_train)

#fit the data into a logistic model
model=LogisticRegression()
model.fir(x_train_vectorized,y_train)
predictions=model.predict(vecr.transform(x_val))

# to check accuracy
from sklearn. metrics import roc_auc_score
score=roc_auc_score(y_test, predictions)

# get the feature names
vect.get_feature_names()
#as np array
features_names=np.array(vect.get_feature_names())

# find the most important coefficients for the models
sorted_coef_index=model.coef_[0].argsort()
feature_names[sorted_coef_index[:10]]
feature_names[sorted_coef_index[:-11:-1]]

#sklearn tfidf with ngram

from sklearn.feature_extraction.text import TfidfVectorize
# fit the model and fit into minimum document frequency
vect=TfidfVectorizer(min_df=5,ngrame_range(1,2)).fit(x_train)
x_train_vectoirzed=vect.transform(x_train)

#fit models
model.logisticRegression()
model.fit(x_train_vectorized,y_train)

predictions=model.predict(vect.transform(x_val))
predictions=model.transform(['put in whatever you want',
'to test your model']
```

Wordnet

```
from nltk.corpus import wordnet as wn
#define meaning,n=noun, 01 is the first def in the dict
word1=wn.synset('word1.n.01')

#find path similarity
word1.path_similarity(word2)

#find lin similarity
#import wordnet information critieria
from ntlk.corpus import wordnet_ic
brown_ic=wordnet_ic.ic('ic-brown.dat')
word1.lin_limilarity(word2,brown_ic)

#collocation
from ntlk.collocations import *
bigram_measures=ntlk.collocations.BigramAssocMeasures()
#learn the word pairs from text
finder=BigramCollocationFinder.from_words(text)
# top 10 closest related pair by using pmi
finder.nbest(bigram_measures.pmi,10)
```

LDA modelling

```
Import gensim
from gensim import corpora,models
dictionary=copora.Dictionary(doc_set)
corpus=[dictionary.doc2bow(doc) for doc in doc_set]
ldamodel=gensim.models.ldamodel.LdaModel(
corpus,num_topics=4,id2word=dictionary, passes=50)
print(ldamodel.print_topics(num_topics=4,numwords=5)

#topic distribution
 vect = CountVectorizer(min_df=20, max_df=0.2,
 stop_words='english',  token_pattern='(?u)\\b\\w\\w+\\b')
new_doc_transformed = vect.transform(new_doc)
corpus = gensim.matutils.Sparse2Corpus(new_doc_
transformed, documents_columns=False)
   doc_topics = ldamodel.get_document_topics(corpus)
   topic_dist = []
   for val in list(doc_topics):
      for v in val:
         topic_dist.append(v)
```

Other useful tools

```
#Stopwords
from nltk.corpus import stopwords
stopword=set(stopwords.words('english'))

#frequency dictionary
from nltk import FreqDist
dist = FreqDist(list_words)

#ngram
from ntlk import n grams
grams=ngrams(sentence.split(),n)
n=gram
```

Author:shawnazhao(github user)
github:https://github.com/shawnazhao
linkedin:https://www.linkedin.com/in/shawna-zhao-a99909106/

POS TAGGING

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Whdeterminer
WP	Whpronoun
WPS	Possessive whpronoun
WRB	Whadverb

```
#get help
import nltk.tag
nltk.help.upenn_tagset('MD')

#tagging
from nltk.tag import pos_tag
tags=pos_tag(list, [tagset=None, tagger=None, lang='eng'])
->tagset=universal, wsj, brown

#parsing
grammar1 = nltk.data.load('mygrammar.cfg')
parser = nltk.ChartParser(grammar1)
trees = parser.parse_all(text16)
for tree in trees:
    print(tree)
```

POS TAGGING

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Whdeterminer
WP	Whpronoun
WPS	Possessive whpronoun
WRB	Whadverb

```
#get help
import nltk.tag
nltk.help.upenn_tagset('MD')

#tagging
from nltk.tag import pos_tag
tags=pos_tag(list, [tagset=None, tagger=None, lang='eng'])
->tagset=universal, wsj, brown

#parsing
grammar1 = nltk.data.load('mygrammar.cfg')
parser = nltk.ChartParser(grammar1)
trees = parser.parse_all(text16)
for tree in trees:
    print(tree)
```