Python 文本挖掘：简单的自然语言统计

1. 把文本变成双词搭配或三词搭配

import nltk

example\_1 = ['I','am','a','big','apple','.']

print nltk.bigrams(example\_1)

>> [('I', 'am'), ('am', 'a'), ('a', 'big'), ('big', 'apple'), ('apple', '.')]

print nltk.trigrams(example)

>> [('I', 'am', 'a'), ('am', 'a', 'big'), ('a', 'big', 'apple'), ('big', 'apple', '.')]

2. 找语料库中出现频率最高的词

from nltk.probability import FreqDist

example\_2 = ['I','am','a','big','apple','.','I','am','delicious',',','I','smells','good','.','I','taste','good','.']

fdist = FreqDist(word for word in example\_2) #把文本转化成词和词频的字典

print fdist.keys() #词按出现频率由高到低排列

>> ['I', '.', 'am', 'good', ',', 'a', 'apple', 'big', 'delicious', 'smells', 'taste']

print fdist.values() #语料中每个词的出现次数倒序排列

>> [4, 3, 2, 2, 1, 1, 1, 1, 1, 1, 1]

3. 找信息量最丰富的词

import nltk

from nltk.collocations import BigramCollocationFinder

from nltk.metrics import BigramAssocMeasures

bigrams = BigramCollocationFinder.from\_words(example\_2)

most\_informative\_chisq\_bigrams = bigrams.nbest(BigramAssocMeasures.chi\_sq, 3) #使用卡方统计法找

most\_informative\_pmi\_bigrams = bigrams.nbest(BigramAssocMeasures.pmi, 3) #使用互信息方法找

print most\_informative\_chisq\_bigrams

>> [('a', 'big'), ('big', 'apple'), ('delicious', ',')]

print most\_informative\_pmi\_bigrams

>> [('a', 'big'), ('big', 'apple'), ('delicious', ',')]

接下来就可以开始简单的自然语言文本统计了。

其中包括统计出：词数量、句子数量、不同词性的词数、信息熵、困惑值等。

这些还比上面更容易一些。

1. 统计词的数量和句子数量

这个在Python 中只要把文本分句、分词并且以多维数组的形式存储即可，然后就可以直接计算词量和句量了。

文本的形式具体如下所示：

[ [[“手机”, “屏幕”, “很”, “好”, “，”], [“镜头”, “也”, “不错”, “。”]], [[“手机”, “好”, “烂”, “，”], [“没”, “办法”, “忍受”, “了”, “！”, “！”]] ]

由于使用了多维数组，所以直接用for 来循环遍历数组，并用len() 函数就可以得出句子数和词数了。

sent\_num = len(sents)

word\_num = len(words)

2. 统计不同词性的词的数量

import jieba.posseg

def postagger(sentence, para):

pos\_data = jieba.posseg.cut(sentence)

pos\_list = []

for w in pos\_data:

pos\_list.append((w.word, w.flag)) #make every word and tag as a tuple and add them to a list

return pos\_list

def count\_adj\_adv(all\_review): #只统计形容词、副词和动词的数量

adj\_adv\_num = []

a = 0

d = 0

v = 0

for review in all\_review:

pos = tp.postagger(review, 'list')

for i in pos:

if i[1] == 'a':

a += 1

elif i[1] == 'd':

d += 1

elif i[1] == 'v':

v += 1

adj\_adv\_num.append((a, d, v))

a = 0

d = 0

v = 0

return adj\_adv\_num

3. 使用nltk 计算信息熵和困惑值

信息熵有很多含义，当它和困惑值一起使用时，它们就有着特定的含义，主要是表达一个信息的“惊奇度”（surprising）。

假设一个语料库，如果其中一条文本里面的内容和其它的其它文本差不多，那它的熵和困惑值就很小。而如果一条文本的内容和其它文本差别很大，这就很让人“惊奇”，此时它的熵和困惑值就大。

nltk 中提供了计算信息熵和困惑值的方法。需要先用所有文本“训练”一个信息熵和困惑值模型，再用这个“模型”计算每个文本的信息熵和困惑值。

from nltk.model.ngram import NgramModel

example\_3 = [['I','am','a','big','apple','.'], ['I','am','delicious',','], ['I','smells','good','.','I','taste','good','.']]

train = list(itertools.chain(\*example\_3)) #把数据变成一个一维数组，用以训练模型

ent\_per\_model = NgramModel(1, train, estimator=None) #训练一元模型，该模型可计算信息熵和困惑值

def entropy\_perplexity(model, dataset):

ep = []

for r in dataset:

ent = model.entropy(r)

per = model.perplexity(r)

ep.append((ent, per))

return ep

print entropy\_perplexity(ent\_per\_model, example\_3)

>> [(4.152825201361557, 17.787911185335403), (4.170127240384194, 18.002523441208137), (3.7021148161194417, 13.015102928960056)]