Snow NLP

##功能

- 中文分词
- 词性标注
- 情感分析
- 文本分类
- 转换成拼音
- 繁体转简体
- 提取文本关键词
- 提取文本摘要
- 分割成句子
- 文本相似度

init.py文件

在python模块的每一个包中，都有一个__init__.py文件（这个文件定义了包的属性和方法）。一个包是一个带有特殊文件 init.py 的目录。init.py 文件定义了包的属性和方法。其实它可以什么也不定义；可以只是一个空文件，但是必须存在。如果 init.py 不存在，这个目录就仅仅是一个目录，而不是一个包，它就不能被导入或者包含其它的模块和嵌套包。

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
##为了使代码同时兼容python2和python3，引入unicode_literals，这样在py2下，'你好'这样的字符串不用家前缀u，也是unicode编码。
from . import normal
from . import seg
from . import tag
from . import sentiment
from .sim import bm25
from .summary import textrank
from .summary import words_merge


class SnowNLP(object):

    def __init__(self, doc):
        self.doc = doc
        self.bm25 = bm25.BM25(doc)

    @property
    def words(self):
```

```
        return seg.seg(self.doc)

    @property
    def sentences(self):
        return normal.get_sentences(self.doc)

    @property
    def han(self):
        return normal.zh2hans(self.doc)

    @property
    def pinyin(self):
        return normal.get_pinyin(self.doc)

    @property
    def sentiments(self):
        return sentiment.classify(self.doc)

    @property
    def tags(self):
        words = self.words
        tags = tag.tag(words)
        return zip(words, tags)

    @property
    def tf(self):
        return self.bm25.f

    @property
    def idf(self):
        return self.bm25.idf

    def sim(self, doc):
        return self.bm25.simall(doc)

    def summary(self, limit=5):
        doc = []
        sents = self.sentences
        for sent in sents:
            words = seg.seg(sent)
            words = normal.filter_stop(words)
            doc.append(words)
        rank = textrank.TextRank(doc)
        rank.solve()
        ret = []
        for index in rank.top_index(limit):
            ret.append(sents[index])
```

```
            return ret

        def keywords(self, limit=5, merge=False):
            doc = []
            sents = self.sentences
            for sent in sents:
                words = seg.seg(sent)
                words = normal.filter_stop(words)
                doc.append(words)
            rank = textrank.KeywordTextRank(doc)
            rank.solve()
            ret = []
            for w in rank.top_index(limit):
                ret.append(w)
            if merge:
                wm = words_merge.SimpleMerge(self.doc, ret)
                return wm.merge()
            return ret
```

代码详解(表现为注释部分)

1. 中文分词 seg

seg.py

```
# -*- coding: utf-8 -*-
from __future__ import print_function
#加上这句后，即使在python2.X，使用print就得像python3.X那样加括号使用。python2.X中print不需要括号，而在python3.X中则需要。
from __future__ import unicode_literals

import codecs
#导入codecs

from ..utils.tnt import TnT
from .y09_2047 import CharacterBasedGenerativeModel


class Seg(object):

    def __init__(self, name='other'):
        if name == 'tnt':
            self.segger = TnT()
```

```
        else:
            self.segger = CharacterBasedGenerativeModel()

    def save(self, fname, iszip=True):#储存
        self.segger.save(fname, iszip)

    def load(self, fname, iszip=True):#加载
        self.segger.load(fname, iszip)

    def train(self, fname): #训练
        fr = codecs.open(fname, 'r', 'utf-8')

        #codecs.open这种方法可以指定一个编码打开文件，使用这个方法打开的文件读取返回的将是unicode。写入时，如果参数 是unicode，则使用open()时指定的编码进行编码后写入；如果是str，则先根据源代码文件声明的字符编码，解码成unicode后再进行前述 操作。相对内置的open()来说，这个方法比较不容易在编码上出现问题。
        data = []#创建空数组
        for i in fr:
            line = i.strip()
            #移除字符串头尾指定的字符(默认为空格或换行符)或字符序列。
            if not line:
                continue
            tmp = map(lambda x: x.split('/'), line.split())
            #map() 会根据提供的函数对指定序列做映射。
            #split() 通过指定分隔符对字符串进行切片，如果参数num有指定值，则仅分隔num个子字符串
            data.append(tmp) #将tmp添加进data数组
        fr.close()
        self.segger.train(data) #添加到训练集

    def seg(self, sentence): #切分函数
        ret = self.segger.tag(sentence)
#用CharacterBasedGenerativeModel或TnT里的tag函数给文本标注embs
        tmp = ''#一个暂时的字符串
        for i in ret: #在句子中的每一个字符
            if i[1] == 'e': #如果第二个字符是e
                yield tmp+i[0] #tmp里增加句子的第一个字符
#一个带有 yield 的函数就是一个 generator，它和普通函数不同，生成一个 generator 看起来像函数调用，但不会执行任何函数代码，直到对其调用 next()（在 for 循环中会自动调用 next()）才开始执行。虽然执行流程仍按函数的流程执行，但每执行到一个 yield 语句就会中断，并返回一个迭代值，下次执行时从 yield 的下一个语句继续执行。
                tmp = '' #清空tmp
            elif i[1] == 'b' or i[1] == 's':
                if tmp:
                    yield tmp
                tmp = i[0]
            else:
```

```
                tmp += i[0]
        if tmp:
            yield tmp


if __name__ == '__main__':
    seg = Seg()
    seg.train('data.txt')
    print(' '.join(seg.seg('主要是用来放置一些简单快速的中文分词和词性标注的程序')))
```

---

2. 词性标注

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

import os
import codecs

from ..utils.tnt import TnT

data_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
                         'tag.marshal')
tagger = TnT()
tagger.load(data_path)


def train(fname):
    fr = codecs.open(fname, 'r', 'utf-8')
    data = []
    for i in fr:
        line = i.strip()
        if not line:
            continue
        tmp = map(lambda x: x.split('/'), line.split())
        data.append(tmp)
    fr.close()
    global tagger
    tagger = TnT()
    tagger.train(data)


def save(fname, iszip=True):
```

```
    tagger.save(fname, iszip)


def load(fname, iszip=True):
    tagger.load(fname, iszip)


def tag_all(words):
    return tagger.tag(words)


def tag(words):
    return map(lambda x: x[1], tag_all(words))
#根据提供的函数对指定序列做映射，返回第二个元素
```

3. 情感分析

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

import os
import codecs

from .. import normal
from .. import seg
from ..classification.bayes import Bayes

#数据文件路径
data_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
                         'sentiment.marshal')


class Sentiment(object):

    def __init__(self):
        #创建Bayes对象
        self.classifier = Bayes()

    #保存训练好的字典数据
    def save(self, fname, iszip=True):
        self.classifier.save(fname, iszip)

    #加载字典数据
```

```
    def load(self, fname=data_path, iszip=True):
        self.classifier.load(fname, iszip)

    #对文档分词
    def handle(self, doc):
        words = seg.seg(doc)
        words = normal.filter_stop(words)
        return words

    # 训练数据集
    def train(self, neg_docs, pos_docs):
        data = []
        #读取消极评论list，同时为每条评论加上neg标签，也放入到一个list中
        for sent in neg_docs:
            data.append([self.handle(sent), 'neg'])
        #读取积极评论list，为每条评论加上pos标签
        for sent in pos_docs:
            data.append([self.handle(sent), 'pos'])
        #调用分类器的训练数据集方法，对模型进行训练
        self.classifier.train(data)

    #分类
    def classify(self, sent):
        #调用贝叶斯分类器的分类方法，获取分类标签和概率
        ret, prob = self.classifier.classify(self.handle(sent))
        #如果分类标签是pos直接返回概率值
        if ret == 'pos':
            return prob
        #如果返回的是neg，由于显示的是积极概率值，因此用1减去消极概率值
        return 1-prob


classifier = Sentiment()
classifier.load()

#训练数据
def train(neg_file, pos_file):
    #打开消极数据文件
    neg = codecs.open(neg_file, 'r', 'utf-8').readlines()
    pos = codecs.open(pos_file, 'r', 'utf-8').readlines()
    neg_docs = []
    pos_docs = []
    #遍历每一条消极评论，放入到list中
    for line in neg:
        neg_docs.append(line.rstrip("\r\n"))
    #遍历每一条积极评论，放入到list中
    for line in pos:
```

```
            pos_docs.append(line.rstrip("\r\n"))
        global classifier
        classifier = Sentiment()
        #训练数据，传入积极、消极评论list
        classifier.train(neg_docs, pos_docs)

    #保存数据字典
    def save(fname, iszip=True):
        classifier.save(fname, iszip)

    #加载数据字典
    def load(fname, iszip=True):
        classifier.load(fname, iszip)

    #对语句进行分类
    def classify(sent):
        return classifier.classify(sent)
```

Bayes.py

```
    from __future__ import unicode_literals
    #为了使代码同时兼容python2和python3，引入unicode_literals，这样在py2下，'你好'这样的字符串
    不用家前缀u，也是unicode编码。

    import sys #sys模块包含了与Python解释器和它的环境有关的函数。
    import gzip #gzip文件读写的时候需要用到Python的gzip模块
    import marshal
    #marshal-内部的Python对象序列化，该模块包含可以以二进制格式读取和写入Python值的函数。
    from math import log, exp #引入math中的对数函数和指数函数

    from ..utils.frequency import AddOneProb
    #从自身文件中导入AddOneProb


    class Bayes(object):

        def __init__(self): #在__init__中初始化对应的实例变量
            self.d = {} ##一个数据字典
            self.total = 0 #total存储所有分类的总词数

        def save(self, fname, iszip=True):
            d = {}#创建一个数据字典
            d['total'] = self.total #0
            d['d'] = {}
```

```
        for k, v in self.d.items():
            d['d'][k] = v.__dict__
        if sys.version_info[0] == 3:#如果使用的python为3.0版本
            fname = fname + '.3'
        if not iszip:
            marshal.dump(d, open(fname, 'wb'))
            #将d写入到用二进制格式写入的fname里。序列化
        else:
            f = gzip.open(fname, 'wb')#打开文件
            f.write(marshal.dumps(d)) #写文件
            f.close() #关闭文件

    def load(self, fname, iszip=True):
        if sys.version_info[0] == 3:
            fname = fname + '.3'
        if not iszip:
            d = marshal.load(open(fname, 'rb'))#反序列化
        else:
            try:
                f = gzip.open(fname, 'rb')
                d = marshal.loads(f.read())
            except IOError:
                f = open(fname, 'rb')
                d = marshal.loads(f.read())
            f.close()
        self.total = d['total']
        self.d = {}
        for k, v in d['d'].items():
            self.d[k] = AddOneProb()
            self.d[k].__dict__ = v

    def train(self, data):
        for d in data:
            c = d[1]
            if c not in self.d:
                self.d[c] = AddOneProb()
            for word in d[0]:
                self.d[c].add(word, 1)
        self.total = sum(map(lambda x: self.d[x].getsum(), self.d.keys()))

    def classify(self, x): #分类函数
        tmp = {}#创建一个暂时的字典
        for k in self.d:
            tmp[k] = log(self.d[k].getsum()) - log(self.total)
            for word in x:
                tmp[k] += log(self.d[k].freq(word))
        ret, prob = 0, 0
```

```
        for k in self.d:
            now = 0
            try:
                for otherk in self.d:
                    now += exp(tmp[otherk]-tmp[k])
                now = 1/now
            except OverflowError:
                now = 0
            if now > prob:
                ret, prob = k, now
        return (ret, prob)
```

4. 转换为拼音

pinyin.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

import codecs

from ..utils.trie import Trie
#trie，又称前缀树或字典树．它利用字符串的公共前缀来节约存储空间．

class PinYin(object):

    def __init__(self, fname):
        self.handle = Trie()
        fr = codecs.open(fname, 'r', 'utf-8')
        for line in fr:
            words = line.split()
            self.handle.insert(words[0], words[1:])
        fr.close()

    def get(self, text):
        ret = []
        for i in self.handle.translate(text):
            if isinstance(i, list) or isinstance(i, tuple):
                ret = ret + i
            else:
                ret.append(i)
        return ret
```