ChatBot第二课

NLP基础

七月在线 加号

微博：@翻滚吧_加号

今天将以NLTK为基础配合讲解自然语言处理的原理

目录

- NLTK
- 文本处理流程
  - 分词
  - 归一化
  - 停止词
- NLP经典三案例
  - 情感分析
  - 文本相似度
  - 文本分类
- 深度学习加持
  - Autoencoder
  - Word2Vec

NLTK

http://www.nltk.org/

Python上著名的自然语言处理库
自带语料库，词性分类库
自带分类，分词，等等功能
强大的社区支持
还有N多的简单版wrapper

NLTK安装

```
# Mac/Unix

sudo pip install -U nltk
# 顺便还可以装个Numpy
sudo pip install -U numpy
# 测试是否安装成功
>>> python
>>> import nltk
```

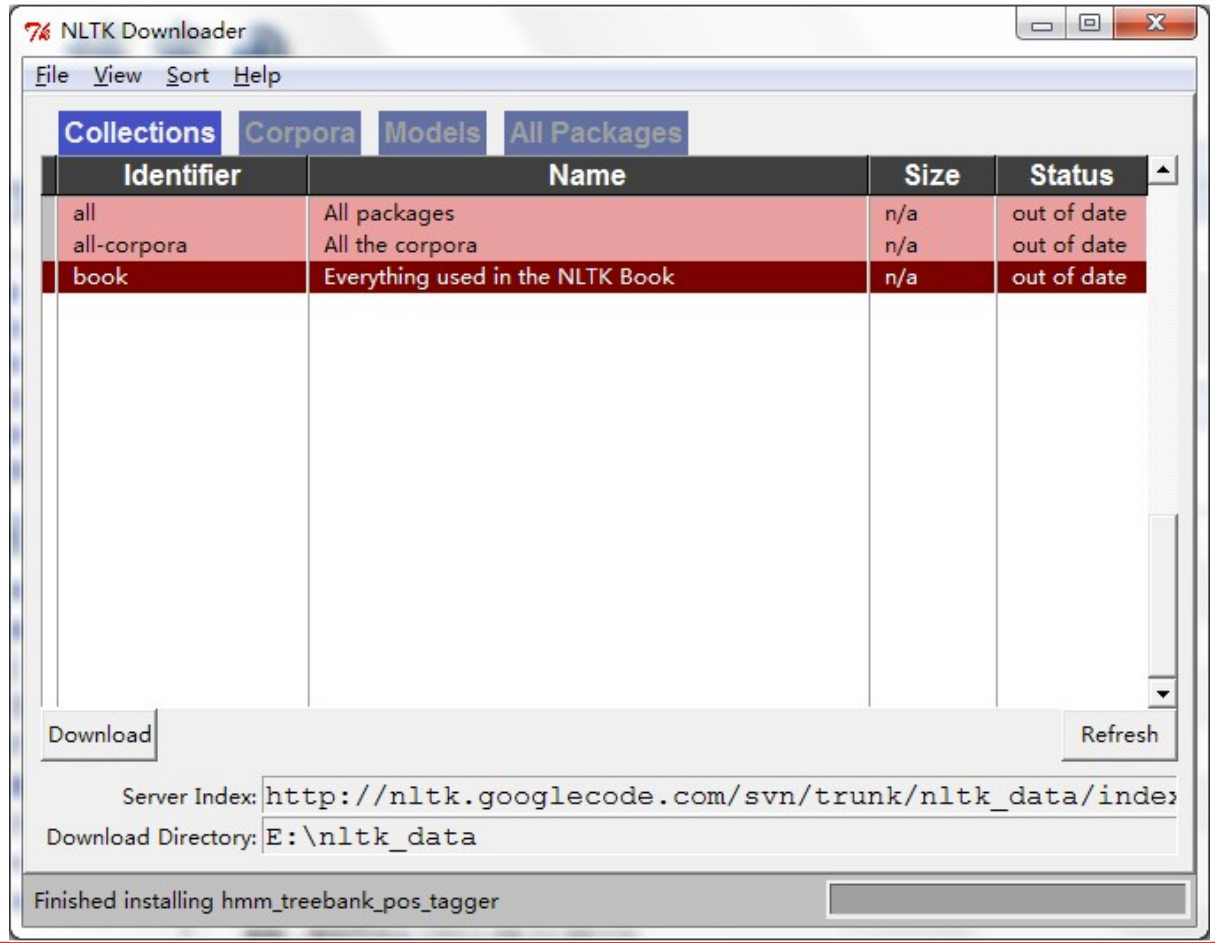
Windows

装个 python3.4 先：http://www.python.org/downloads/
有空也可以装个Numpy：http://sourceforge.net/projects/numpy/files/NumPy/
安装NLTK：http://pypi.python.org/pypi/nltk
测试安装成功：开始>Python34, 输入 import nltk

注意python版本号，操作系统位数 一致

安装语料库

```
import nltk
nltk.download()
```



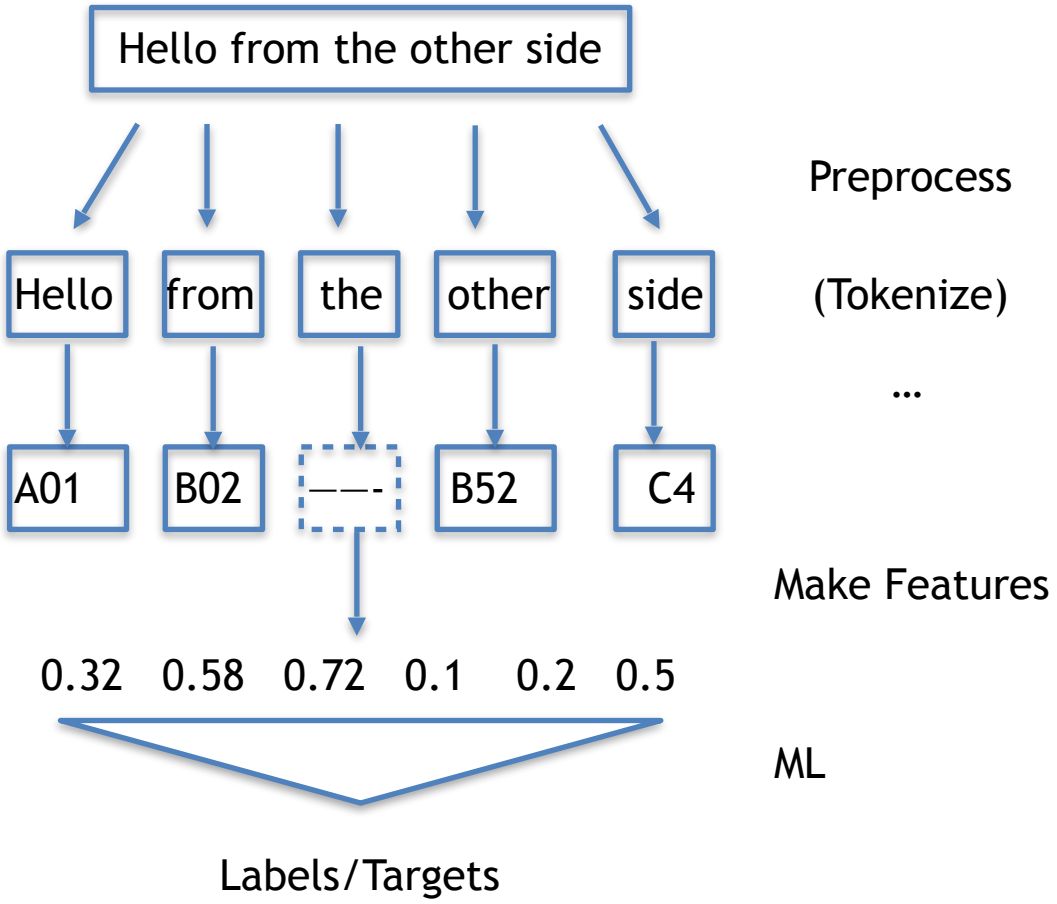
功能一览表

NLTK Modules	Functionality
nltk.corpus	Corpus
nltk.tokenize, nltk.stem	Tokenizers, stemmers
nltk.collocations	t-test, chi-squared, mutual-info
nltk.tag	n-gram, backoff,Brill, HMM, TnT
nltk.classify, nltk.cluster	Decision tree, Naive bayes, K-means
nltk.chunk	Regex,n-gram, named entity
nltk.parsing	Parsing
nltk.sem, nltk.interence	Semantic interpretation
nltk.metrics	Evaluation metrics
nltk.probability	Probability & Estimation
nltk.app, nltk.chat	Applications

NLTK自带语料库

```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial',
'fiction', 'government', 'hobbies', 'humor',
'learned', 'lore', 'mystery', 'news', 'religion',
'reviews', 'romance', 'science_fiction']
>>> len(brown.sents())
57340
>>> len(brown.words())
1161192
```

文本处理流程


Hello from the other side
Hello
from
the
other
side
A01
B02
——-
B52
C4
0.32
0.58
0.72
0.1
0.2
0.5
Labels/Targets
Preprocess
(Tokenize)
…
Make Features
ML

Tokenize

把长句子拆成有“意义”的小部件

Tokenize

```
>>> import nltk
>>> sentence = “hello, world"
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['hello', ‘,', 'world']
```

中英文NLP区别

今天 / 天气 / 不错 / ！
What a nice weather today !

启发式 Heuristic

机器学习/统计方法：HMM、CRF

W O R D
千 言 万 语

中文分词

```
import jieba
seg_list = jieba.cut("我来到北京清华大学", cut_all=True)
print "Full Mode:", "/ ".join(seg_list)  # 全模式
seg_list = jieba.cut("我来到北京清华大学", cut_all=False)
print "Default Mode:", "/ ".join(seg_list)  # 精确模式
seg_list = jieba.cut("他来到了网易杭研大厦")  # 默认是精确模式
print ", ".join(seg_list)
seg_list = jieba.cut_for_search("小明硕士毕业于中国科学院计算所，后在日本京都大学深造")
# 搜索引擎模式
print ", ".join(seg_list)
```

【全模式】: 我/ 来到/ 北京/ 清华/ 清华大学/ 华大/ 大学

【精确模式】: 我/ 来到/ 北京/ 清华大学

【新词识别】： 他, 来到, 了, 网易, 杭研, 大厦
(此处，“杭研”并没有在词典中，但是也被Viterbi算法识别出来了)

【搜索引擎模式】： 小明, 硕士, 毕业, 于, 中国, 科学, 学院, 科学院, 中国科学院, 计算, 计算所, 后, 在, 日本, 京都, 大学, 日本京都大学, 深造

分词之后的效果

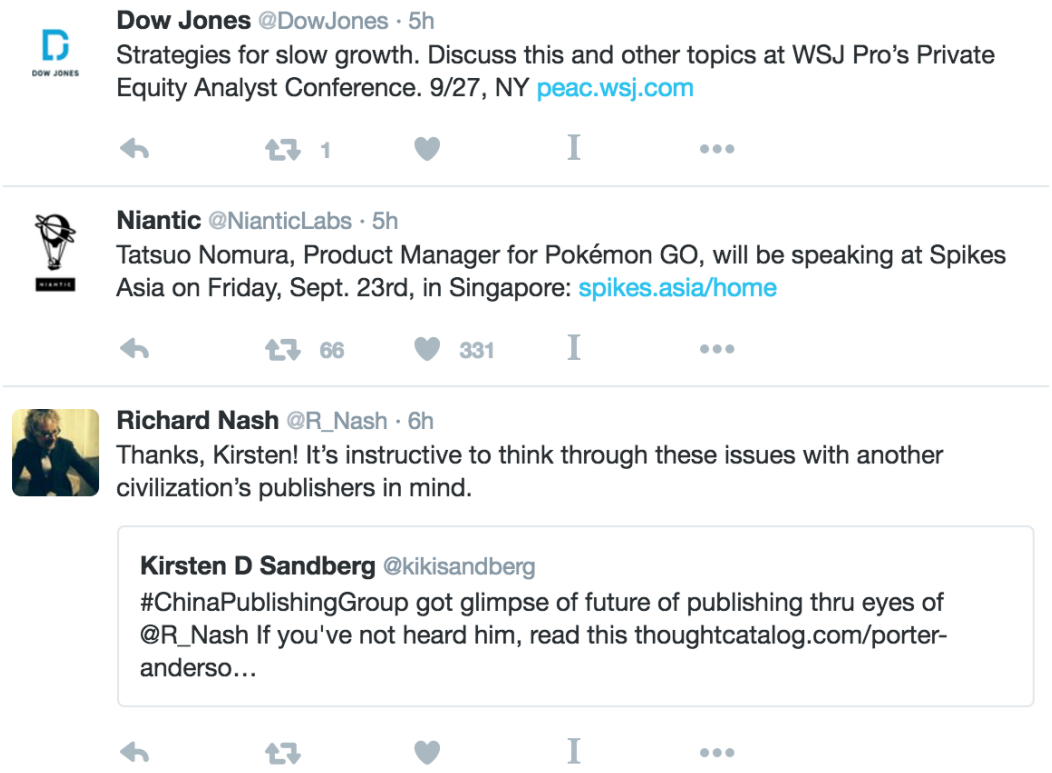
[‘what’, ‘a’, ‘nice’, ‘weather’, ‘today’]

[‘今天’，’天气’，’真’，’不错’]

有时候tokenize没那么简单

比如社交网络上，这些乱七八糟的不合语法不合正常逻辑的语言很多：

拯救 @某人, 表情符号, URL, #话题符号



社交网络语言的tokenize

举个栗子🌰

```
from nltk.tokenize import word_tokenize

tweet = 'RT @angelababy: love you baby! :D http://ah.love #168cm'
print(word_tokenize(tweet))
# ['RT', '@', 'angelababy', ':', 'love', 'you', 'baby', '!', ':',
# 'D', 'http', ':', '//ah.love', '#', '168cm']
```

社交网络语言的tokenize

```
import re
emoticons_str = r"""
    (?:
        [:=;] # 眼睛
        [oO\-]? # 鼻子
        [D\)\]\(\]/\\OpP] # 嘴
    )"""
regex_str = [
    emoticons_str,
    r'<[^>]+>', # HTML tags
    r'(?:@[\w_]+)', # @某人
    r"(?:\#+[\w_]+[\w\'_\-]*[\w_]+)", # 话题标签
    r'http[s]?://(?:[a-z]|[0-9]|[$-_@.&amp;+]|[!*\(\),]|(?:%[0-9a-f][0-9a-f]))+',
                                        # URLs
    r'(?:(?:\d+,?)+(?:\.?\d+)?)', # 数字
    r"(?:[a-z][a-z'\-_]+[a-z])", # 含有 - 和 ‘ 的单词
    r'(?:[\w_]+)', # 其他
    r'(?:\S)' # 其他
]
```

正则表达式

对照表
http://www.regexlab.com/zh/regref.htm

社交网络语言的tokenize

```
tokens_re = re.compile(r'('+'|'.join(regex_str)+')', re.VERBOSE | re.IGNORECASE)
emoticon_re = re.compile(r'^'+emoticons_str+'$', re.VERBOSE | re.IGNORECASE)

def tokenize(s):
    return tokens_re.findall(s)

def preprocess(s, lowercase=False):
    tokens = tokenize(s)
    if lowercase:
        tokens = [token if emoticon_re.search(token) else token.lower() for token in
tokens]
    return tokens

tweet = 'RT @angelababy: love you baby! :D http://ah.love #168cm'
print(preprocess(tweet))
# ['RT', '@angelababy', ':', 'love', 'you', 'baby',
# '!', ':D', 'http://ah.love', '#168cm']
```

纷繁复杂的词形

Inflection变化: walk => walking => walked
不影响词性

derivation 引申: nation (noun) => national (adjective) => nationalize (verb)
影响词性

词形归一化

Stemming 词干提取：一般来说，就是把不影响词性的inflection的小尾巴砍掉

walking 砍ing = walk
walked 砍ed = walk

Lemmatization 词形归一：把各种类型的词的变形，都归为一个形式

went 归一 = go
are 归一 = be

NLTK实现Stemming

```
>>> from nltk.stem.porter import PorterStemmer
>>> porter_stemmer = PorterStemmer()
>>> porter_stemmer.stem('maximum')
u'maximum'
>>> porter_stemmer.stem('presumably')
u'presum'
>>> porter_stemmer.stem('multiply')
u'multipli'
>>> porter_stemmer.stem('provision')
u'provis'
```

```
>>> from nltk.stem.lancaster import LancasterStemmer
>>> lancaster_stemmer = LancasterStemmer()
>>> lancaster_stemmer.stem('maximum')
'maxim'
>>> lancaster_stemmer.stem('presumably')
'presum'
>>> lancaster_stemmer.stem('presumably')
'presum'
```

```
>>> from nltk.stem import SnowballStemmer
>>> snowball_stemmer = SnowballStemmer("english")
>>> snowball_stemmer.stem('maximum')
u'maximum'
>>> snowball_stemmer.stem('presumably')
u'presum'
```

```
>>> from nltk.stem.porter import PorterStemmer
>>> p = PorterStemmer()
>>> p.stem('went')
'went'
>>> p.stem('wenting')
'went'
```

NLTK实现Lemma

```
>>> from nltk.stem import WordNetLemmatizer
>>> wordnet_lemmatizer = WordNetLemmatizer()
>>> wordnet_lemmatizer.lemmatize('dogs')
u'dog'
>>> wordnet_lemmatizer.lemmatize('churches')
u'church'
>>> wordnet_lemmatizer.lemmatize('aardwolves')
u'aardwolf'
>>> wordnet_lemmatizer.lemmatize('abaci')
u'abacus'
>>> wordnet_lemmatizer.lemmatize('hardrock')
'hardrock'
```

Lemma的小问题

v. go的过去式

Went

n. 英文名：温特

NLTK更好地实现Lemma

```
# 木有POS Tag，默认是NN 名词
>>> wordnet_lemmatizer.lemmatize('are')
'are'
>>> wordnet_lemmatizer.lemmatize('is')
'is'

# 加上POS Tag
>>> wordnet_lemmatizer.lemmatize('is', pos='v')
u'be'
>>> wordnet_lemmatizer.lemmatize('are', pos='v')
u'be'
```

Part-Of-Speech

Tag	Meaning	Examples
ADJ	adjective	new, good, high, special, big, local
ADV	adverb	really, already, still, early, now
CNJ	conjunction	and, or, but, if, while, although
DET	determiner	the, a, some, most, every, no
EX	existential	there, there's
FW	foreign word	dolce, ersatz, esprit, quo, maitre
MOD	modal verb	will, can, would, may, must, should
N	noun	year, home, costs, time, education
NP	proper noun	Alison, Africa, April, Washington
NUM	number	twenty-four, fourth, 1991, 14:24
PRO	pronoun	he, their, her, its, my, I, us
P	preposition	on, of, at, with, by, into, under
TO	the word to	to
UH	interjection	ah, bang, ha, whee, hmpf, oops
V	verb	is, has, get, do, make, see, run
VD	past tense	said, took, told, made, asked
VG	present participle	making, going, playing, working
VN	past participle	given, taken, begun, sung
WH	wh determiner	who, which, when, what, where, how

NLTK标注POS Tag

```
>>> import nltk
>>> text = nltk.word_tokenize('what does the fox say')
>>> text
['what', 'does', 'the', 'fox', 'say']
>>> nltk.pos_tag(text)
[('what', 'WDT'), ('does', 'VBZ'), ('the', 'DT'), ('fox', 'NNS'), ('say', 'VBP')]
```

Stopwords

一千个HE有一千种指代
一千个THE有一千种指事

对于注重理解文本『意思』的应用场景来说
歧义太多

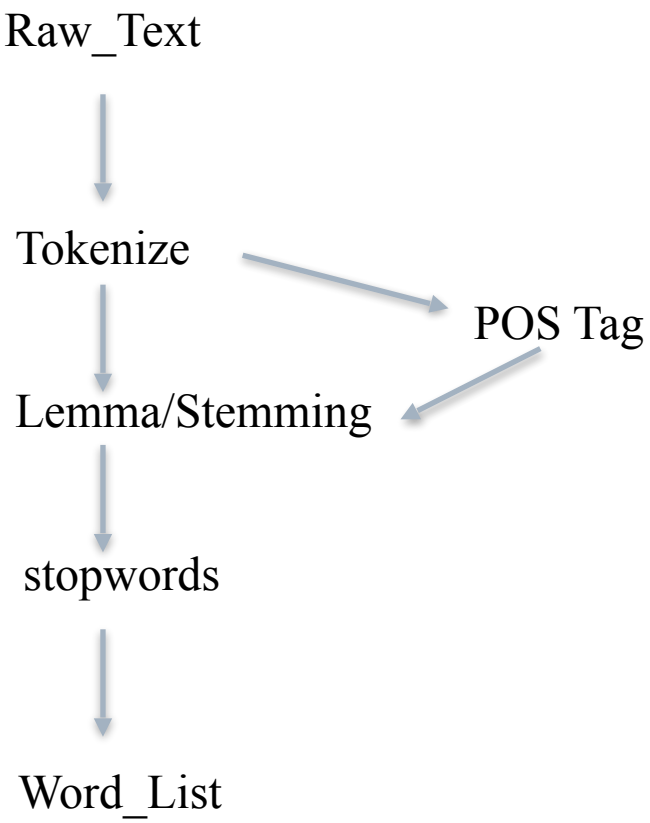
全体stopwords列表 http://www.ranks.nl/stopwords

NLTK去除stopwords

首先记得在console里面下载一下词库
或者 nltk.download('stopwords')

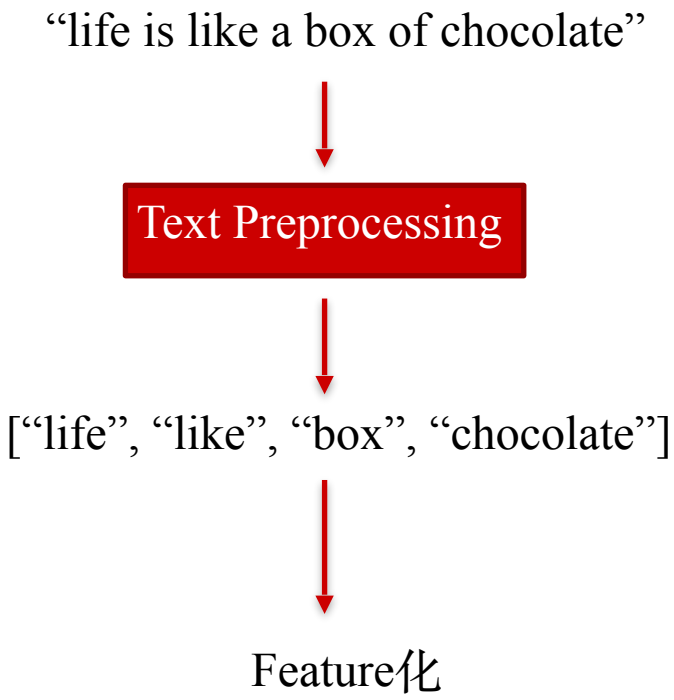
```
from nltk.corpus import stopwords
# 先token一把，得到一个word_list
# ...
# 然后filter一把
filtered_words =
[word for word in word_list if word not in stopwords.words('english')]
```

一条typical的文本预处理流水线



什么是自然语言处理?

文本预处理让我们得到了什么？



NLTK在NLP上的经典应用

> 情感分析

> 文本相似度

> 文本分类

应用：情感分析

哪些是夸你？哪些是黑你？

应用：情感分析

最简单的 sentiment dictionary

like 1
good 2
bad -2
terrible -3

类似于关键词打分机制

比如：AFINN-111
http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010

NLTK完成简单的情感分析

```
sentiment_dictionary = {}
for line in open('data/AFINN-111.txt')
    word, score = line.split('\t')
    sentiment_dictionary[word] = int(score)

# 把这个打分表记录在一个Dict上以后
# 跑一遍整个句子，把对应的值相加
total_score = sum(sentiment_dictionary.get(word, 0) for word in words)
# 有值就是Dict中的值，没有就是0

# 于是你就得到了一个 sentiment score
```

Too Young Too Simple

显然这个方法太Naive
新词怎么办?
特殊词汇怎么办?
更深层次的玩意儿怎么办?

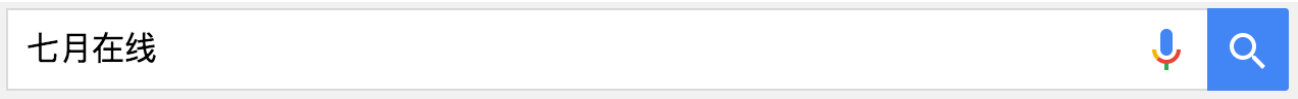
配上ML的情感分析

```
from nltk.classify import NaiveBayesClassifier

# 随手造点训练集
s1 = 'this is a good book'
s2 = 'this is a awesome book'
s3 = 'this is a bad book'
s4 = 'this is a terrible book'


def preprocess(s):
    # Func: 句子处理
    # 这里简单的用了split(), 把句子中每个单词分开
    # 显然 还有更多的processing method可以用
    return {word: True for word in s.lower().split()}
    # return长这样:
    # {'this': True, 'is':True, 'a':True, 'good':True, 'book':True}
    # 其中, 前一个叫fname, 对应每个出现的文本单词;
    # 后一个叫fval, 指的是每个文本单词对应的值。
    # 这里我们用最简单的True,来表示,这个词『出现在当前的句子中』的意义。
    # 当然啦, 我们以后可以升级这个方程, 让它带有更加牛逼的fval, 比如 word2vec
```

配上ML的情感分析

```
# 把训练集给做成标准形式
training_data = [[preprocess(s1), 'pos'],
                 [preprocess(s2), 'pos'],
                 [preprocess(s3), 'neg'],
                 [preprocess(s4), 'neg']]

# 喂给model吃
model = NaiveBayesClassifier.train(training_data)

# 打出结果
print(model.classify(preprocess('this is a good book')))
```

应用：文本相似度



北京七月在线科技的微博_微博
weibo.com/julyedu ▾ Translate this page
北京七月在线科技，七月算法www.julyedu.com官方微博。北京七月在线科技的微博主页、个人资料、相册。新浪微博，随时随地分享身边的新鲜事儿。

七月在线问答的微博_微博
weibo.com/askjulyedu ▾ Translate this page
玩个游戏，转发本微博一句话证明你是七月在线的学员，后天晚上我选一人送《机器学习与量化交易项目班》：O网页链接 100元优惠券一张，或者你任选一本100元以内 ...

今15年创业，享受改变的过程- 结构之法算法之道- 博客频道- CSDN.NET
blog.csdn.net/v_july_v/article/details/47808607 ▾ Translate this page
20 Aug 2015 - 很快，1月27日，我们上线了自己的在线教育网站：七月算法在线学院（后更名为：七月在线） http://www.julyedu.com/。目前专注5类在线课程：面试、 ...

七月题库 - 笔试面试刷题神器
www.julyapp.com/ ▾ Translate this page
七月题库APP，专为IT人打造的笔试面试刷题神器。每天10分钟10道选择 ... 详尽的错题解析。七月题库APP在手，offer从此不再愁。 ... 七月算法官网. © 七月在线科技.

七月在线招聘-北京七月在线科技有限公司招聘-拉勾网
www.lagou.com/gongsi/76553.html ▾ Translate this page
1 七月在线：julyedu.com，专注数据领域的在线教育平台。 2 七月在线APP，配套官网的课程、视频，已经发布. 公司介绍. 专注算法、ml、dl、dm、nlp、cv等领域.

用元素频率表示文本特征

we	you	he	work	happy	are
1	0	3	0	1	1
1	0	2	0	1	1
0	1	0	1	0	0

余弦定理

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Frequency 频率统计

```
import nltk
from nltk import FreqDist

# 做个词库先
corpus = 'this is my sentence ' \
           'this is my life ' \
           'this is the day'

# 随便tokenize一下
# 显然，正如上文提到，
# 这里可以根据需要做任何的preprocessing:
# stopwords, lemma, stemming, etc.
tokens = nltk.word_tokenize(corpus)
print(tokens)
# 得到token好的word list
# ['this', 'is', 'my', 'sentence',
# 'this', 'is', 'my', 'life', 'this',
# 'is', 'the', 'day']

# 借用NLTK的FreqDist统计一下文字出现的频率
fdist = FreqDist(tokens)

# 它就类似于一个Dict
# 带上某个单词，可以看到它在整个文章中出现的次数
print(fdist['is'])
# 3
```

Frequency 频率统计

```
# 好，此刻，我们可以把最常用的50个单词拿出来
standard_freq_vector = fdist.most_common(50)
size = len(standard_freq_vector)
print(standard_freq_vector)
# [('is', 3), ('this', 3), ('my', 2),
# ('the', 1), ('day', 1), ('sentence', 1),
# ('life', 1)
```

Frequency 频率统计

```
# Func: 按照出现频率大小, 记录下每一个单词的位置
def position_lookup(v):
    res = {}
    counter = 0
    for word in v:
        res[word[0]] = counter
        counter += 1
    return res

# 把标准的单词位置记录下来
standard_position_dict = position_lookup(standard_freq_vector)
print(standard_position_dict)
# 得到一个位置对照表
# {'is': 0, 'the': 3, 'day': 4, 'this': 1,
# 'sentence': 5, 'my': 2, 'life': 6}
```

Frequency 频率统计

```
# 这时, 如果我们有个新句子:
sentence = 'this is cool'
# 先新建一个跟我们的标准vector同样大小的向量
freq_vector = [0] * size
# 简单的Preprocessing
tokens = nltk.word_tokenize(sentence)
# 对于这个新句子里的每一个单词
for word in tokens:
    try:
        # 如果在我们的词库里出现过
        # 那么就在"标准位置"上+1
        freq_vector[standard_position_dict[word]] += 1
    except KeyError:
        # 如果是个新词
        # 就pass掉
        continue

print(freq_vector)
# [1, 1, 0, 0, 0, 0, 0]
# 第一个位置代表 is, 出现了一次
# 第二个位置代表 this, 出现了一次
# 后面都木有
```

应用：文本分类


中国联通 16:38 76%
发现 朋友圈
宝马中国 推广
越是期待已久，悦是如期而至。
查看详情
悦
19小时前
来了.....
怎么变成了聊天贴...
微博评论了
早晚要来。。
哎哟...聊天帖
该来的还是来了
6:40 AM
朋友圈
绅宝
SENOVA
绅宝 X65
有种的你，可见！
昨天 删除
一起黯地灰吧!
我靠你改名字了
笑笑笑
昨天接到了宝马广告，今天又接到了绅宝广
告，赞一个！！
重点被骗.....
回复绅宝: 哈哈，流弊
回复绅宝: 哈哈哈哈
可以的啊
这是谁设计的，真心不错，我们大斯堪也
-赵林: 李鬼，有意思吗？
据说这是恶评营销。
嗨，这不会是你家的车吧？
不科学，右上角没有屏蔽功能

TF-IDF

TF: Term Frequency, 衡量一个term在文档中出现得有多频繁。

TF(t) = (t出现在文档中的次数) / (文档中的term总数).

IDF: Inverse Document Frequency, 衡量一个term有多重要。
有些词出现的很多，但是明显不是很有卵用。比如’is'，’the‘，’and‘之类的。
为了平衡，我们把罕见的词的重要性（weight）搞高，
把常见词的重要性搞低。

IDF(t) = log_e(文档总数 / 含有t的文档总数).

TF-IDF = TF * IDF

TF-IDF

举个栗子  :

一个文档有100个单词，其中单词baby出现了3次。
那么，TF(baby) = (3/100) = 0.03.

好，现在我们如果有10M的文档， baby出现在其中的1000个文档中。
那么，IDF(baby) = log(10,000,000 / 1,000) = 4

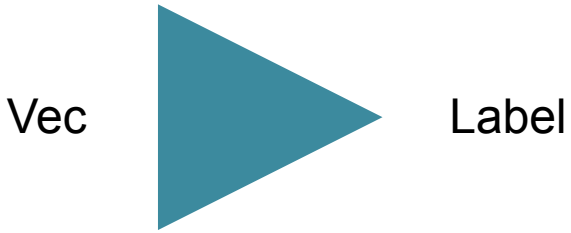
所以， TF-IDF(baby) = TF(baby) * IDF(baby) = 0.03 * 4 = 0.12

NLTK实现TF-IDF

```
from nltk.text import TextCollection

# 首先，把所有的文档放到TextCollection类中。
# 这个类会自动帮你断句，做统计，做计算
corpus = TextCollection(['this is sentence one',
                         'this is sentence two',
                         'this is sentence three'])

# 直接就能算出tfidf
# (term: 一句话中的某个term, text: 这句话)
print(corpus.tf_idf('this', 'this is sentence four'))
# 0.444342

# 同理，怎么得到一个标准大小的vector来表示所有的句子？

# 对于每个新句子
new_sentence = 'this is sentence five'
# 遍历一遍所有的vocabulary中的词：
for word in standard_vocab:
    print(corpus.tf_idf(word, new_sentence))
    # 我们会得到一个巨长(=所有vocab长度)的向量
```

接下来?

ML



可能的ML模型:
SVM
LR
RF
MLP
LSTM
RNN
....

深度学习的加持

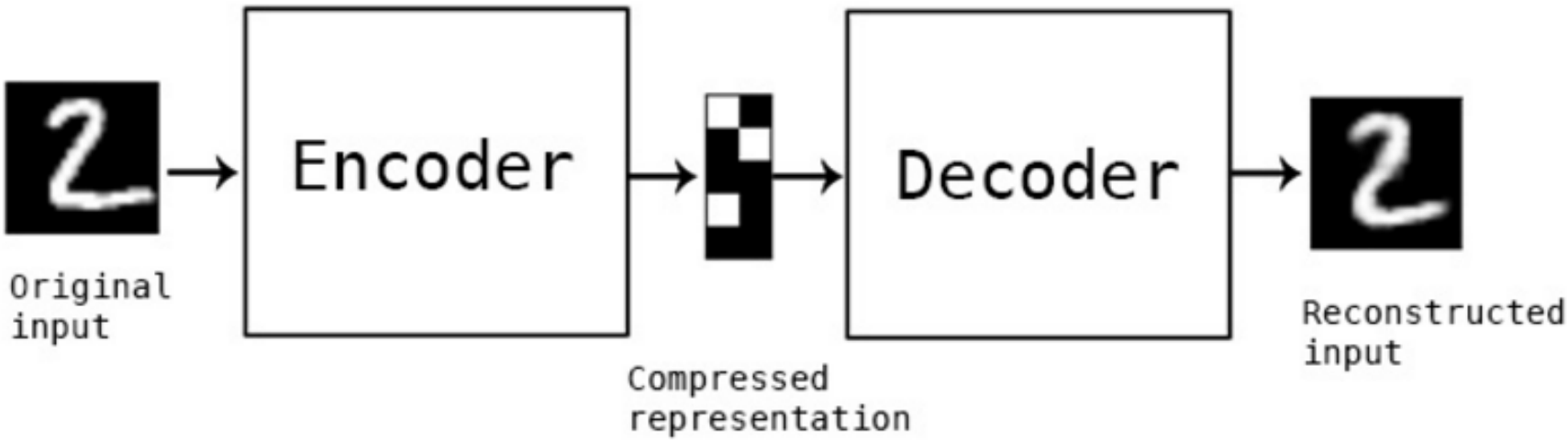
Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations.
(Goodfellow, Bengio, and Courville, 2016)

深度学习的加持

https://radimrehurek.com/gensim/

keras.io

Auto-Encoder



- Data-specific
- Lossy
- Learn from examples

Auto-Encoder

```
from keras.layers import Input, Dense
from keras.models import Model
from sklearn.cluster import KMeans


class ASCIIAutoencoder():
    """基于字符的Autoencoder."""

    def __init__(self, sen_len=512, encoding_dim=32, epoch=50, val_ratio=0.3):
        """
        Init.

        :param sen_len: 把sentences pad成相同的长度
        :param encoding_dim: 压缩后的维度dim
        :param epoch: 要跑多少epoch
        :param kmeanmodel: 简单的KNN clustering模型
        """
        self.sen_len = sen_len
        self.encoding_dim = encoding_dim
        self.autoencoder = None
        self.encoder = None
        self.kmeanmodel = KMeans(n_clusters=2)
        self.epoch = epoch
```

Auto-Encoder

```
def fit(self, x):
    """
    模型构建。

    :param x: input text
    """

    # 把所有的trainset都搞成同一个size，并把每一个字符都换成ascii码
    x_train = self.preprocess(x, length=self.sen_len)
    # 然后给input预留好位置
    input_text = Input(shape=(self.sen_len,))
    # "encoded" 每一经过一层，都被刷新成小一点的"压缩后表达式"
    encoded = Dense(1024, activation='tanh')(input_text)
    encoded = Dense(512, activation='tanh')(encoded)
    encoded = Dense(128, activation='tanh')(encoded)
    encoded = Dense(self.encoding_dim, activation='tanh')(encoded)

    # "decoded" 就是把刚刚压缩完的东西，给反过来还原成input_text
    decoded = Dense(128, activation='tanh')(encoded)
    decoded = Dense(512, activation='tanh')(decoded)
    decoded = Dense(1024, activation='tanh')(decoded)
    decoded = Dense(self.sen_len, activation='sigmoid')(decoded)

    # 整个从大到小再到大的model，叫 autoencoder
    self.autoencoder = Model(input=input_text, output=decoded)

    # 那么 只从大到小（也就是一半的model）就叫 encoder
    self.encoder = Model(input=input_text, output=encoded)
```

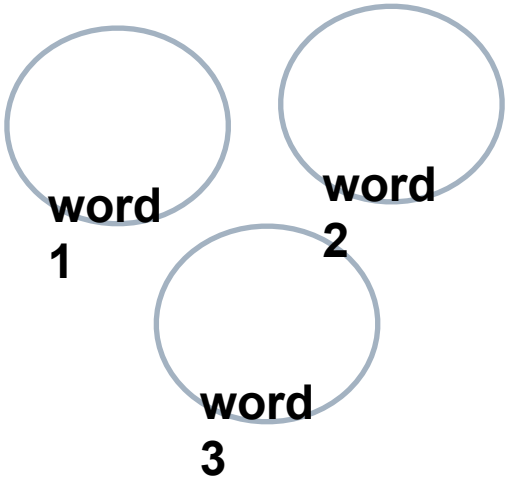
Auto-Encoder

```
# 同理，我们接下来搞一个decoder出来，也就是从小到大的model
    # 来，首先encoded的input size给预留好
    encoded_input = Input(shape=(1024,))
    # autoencoder的最后一层，就应该是decoder的第一层
    decoder_layer = self.autoencoder.layers[-1]
    # 然后我们从头到尾连起来，就是一个decoder了！
    decoder = Model(input=encoded_input, output=decoder_layer(encoded_input))

    # compile
    self.autoencoder.compile(optimizer='adam', loss='mse')

    # 跑起来
    self.autoencoder.fit(x_train, x_train,
                         nb_epoch=self.epoch,
                         batch_size=1000,
                         shuffle=True,
                         )

    # 这一部分是自己拿自己train一下KNN，一件简单的基于距离的分类器
    x_train = self.encoder.predict(x_train)
    self.kmeanmodel.fit(x_train)
```

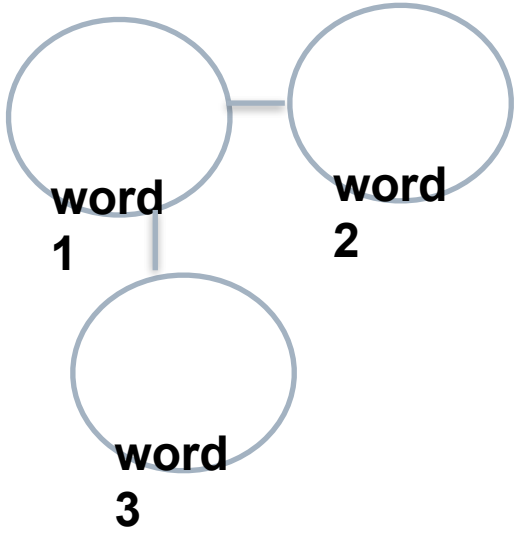
Auto-Encoder

```
def predict(self, x):
        """
        做预测。

        :param x: input text
        :return: predictions
        """
        # 同理，第一步 把来的 都给搞成ASCII化，并且长度相同
        x_test = self.preprocess(x, length=self.sen_len)
        # 然后用encoder把test集给压缩
        x_test = self.encoder.predict(x_test)
        # KNN给分类出来
        preds = self.kmeanmodel.predict(x_test)

        return preds


def preprocess(self, s_list, length=256):
    ...
```

Word2Vec (Mikolov, 2013)



Atomic Units



Relationships

Word2Vec

Lexical Taxonomy 词汇分类: WordNet (Miller, 1990)

Symbolic Representation 符号表示: One-Hot (Turian et al., 2010)

Distributional Similarity Based Representation相似度表示:
Full document: TF-IDF (Joachims, 1996)
Window: co-occurrence matrix + SVD (Bullinaria & Levy, 2012)

Word2Vec

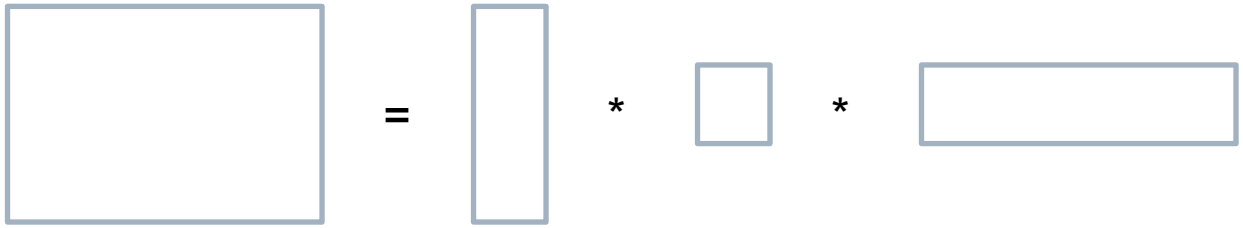
Are you kidding?
No, I am serious?
I am kidding.
You are serious.
Are you serious?
Am I kidding?

window=1	Are	Am	I	You	Kidding	Serious
Are	0	0	0	3	0	1
Am	0	0	3	0	1	1
I	0	3	0	0	0	0
You	3	0	0	0	1	1
Kidding	0	1	0	1	0	0
Serious	1	1	0	1	0	0

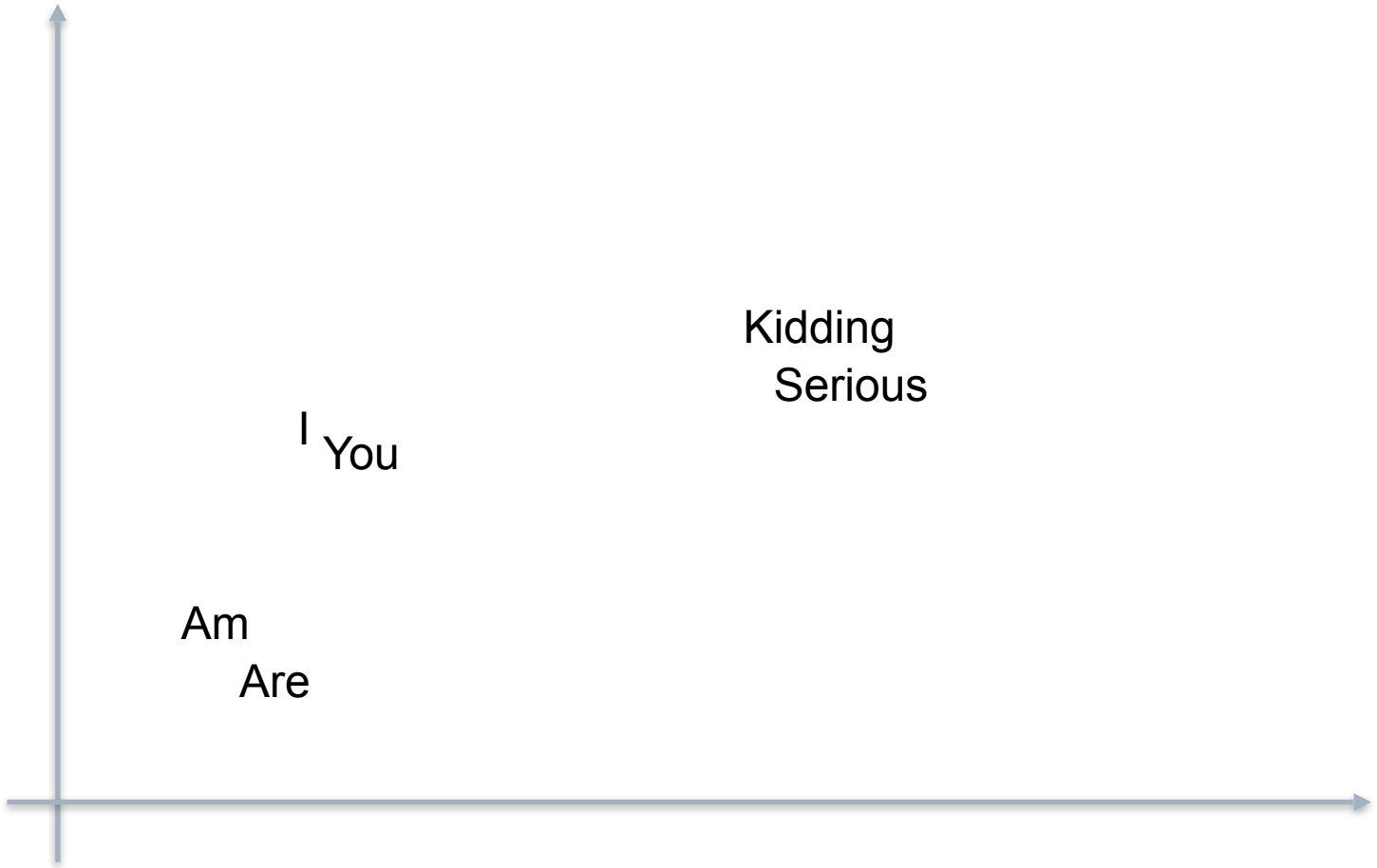
Word2Vec

SVD

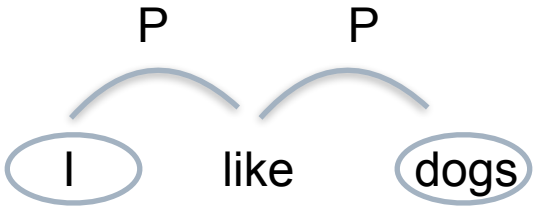
M = UΣV*



Word2Vec



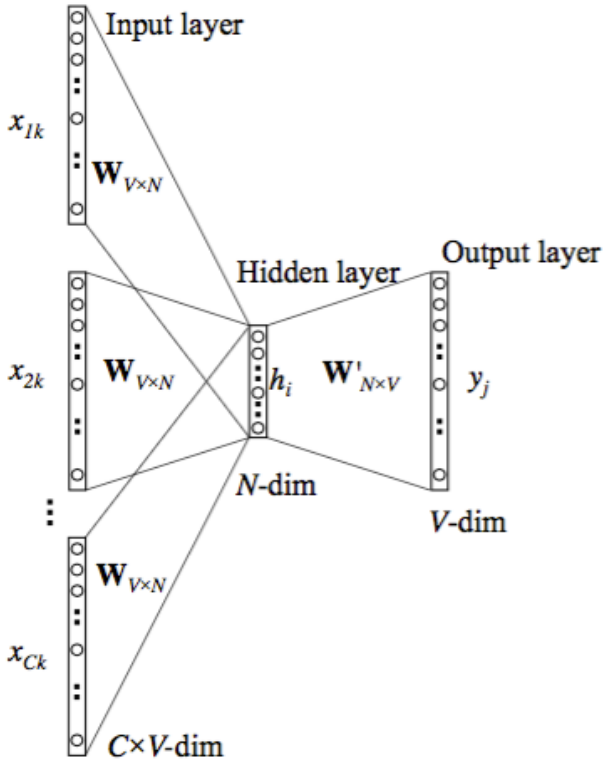
Word2Vec



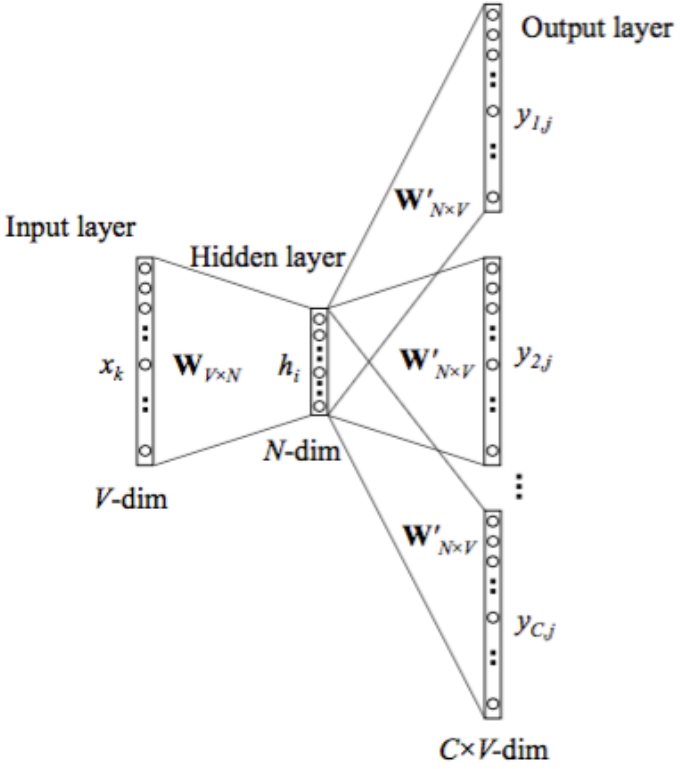
$$J(\theta)=1/T * \Sigma^{T} \log P(w_{t+j} \mid w_t)$$

Word2Vec

CBoW



Skip-Gram



Word2Vec

"**Skip-gram:**
works well with small amount of the training data,
represents well even rare words or phrases.
CBOW:
several times faster to train than the skip-gram,
slightly better accuracy for the frequent words。"
— Mikolov (2013)

Word2Vec

```
# 简单的文字预处理：

# 1. 去除HTML
# 这里用到BeautifulSoup这个库，
# 当然，这种简单的事情，也可以自己做个字符串运算解决
from bs4 import BeautifulSoup
beautiful_text = BeautifulSoup(raw_text).get_text()
#
# 2. 把非字母的去除掉
# 这里可以用正则表达式解决
import re
letters_only = re.sub("[^a-zA-Z]", " ", beautiful_text)
#
# 3. 全部小写化
words = letters_only.lower().split()
#
# 4. 去除stopwords
# 这里用到NLTK
from nltk.corpus import stopwords
stops = set(stopwords.words("english"))
meaningful_words = [w for w in words if not w in stops]
# 高阶文字处理：
# 5. Lemmatization
#
# 这个比较复杂，下次NLTK的时候讲

# 6. 搞回成一长串string
return( " ".join( meaningful_words ))
```

Word2Vec

```
# tokenizor: 把原来的string训练集，变成 list of lists:
# 这个寒老师上堂课应该讲过:
# 简单点的话，可以用这个:
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
# 达到这样的效果:
>>> print sentences
# 原文: ['Hello, how are you', 'im fine, thank you, and you?']
[['hello', 'how'], ['fine', 'thank']]

# 现在进入正题，w2v。
# 我们用Gensim这个库来做，很方便。
from gensim.models import word2vec
# 先设一下param
num_features = 1000    # 最多多少个不同的features
min_word_count = 10    # 一个word，最少出现多少次 才被计入
num_workers = 4        # 多少thread一起跑（快一点儿）
size = 256             # vec的size
window = 5             # 前后观察多长的"语境"

# 跑起来
model = word2vec.Word2Vec(sentences, size=size, workers=num_workers, \
            size=num_features, min_count = min_word_count, \
            window = window)
# 你可以save下来
model.save('LOL.save')
# 日后再load回来
model = word2vec.Word2Vec.load('LOL.save')
```

Word2Vec

```
# 当然 你们也许会看到谷歌也提供了自己的News包：
# 要load 其他语言train出来的文件（比如C） 的Bin或者text文件
# 那就这样：
model = Word2Vec.load_word2vec_format('google_news.txt', binary=False)  # C text format
model = Word2Vec.load_word2vec_format('google_news.bin', binary=True)  # C binary format

# 几个常用的用法：
# woman + king - man = queen
>>> model.most_similar(positive=['woman', 'king'], negative=['man'])
[('queen', 0.50882536), ...]
# 求两个词的senmatics相似度
>>> model.similarity('woman', 'man')
0.73723527
# 就更dict一样使用你train好的model
>>> model['computer']
array([-0.00449447, -0.00310097,  0.02421786, ...], dtype=float32)

# 现在 你可以把这个model包装起来。把你所有的sentences token 过一遍
def w2vmodel(sentences):
    ...
    return vec
```

Word2Vec

```
# 这个时候你会发现，我们的vec是针对每个word的。而我们的训练集 是sen和label互相对应的，
# 工业上，到了这一步，有三种解决方案：
# 1. 平均化一个句子里所有词的vec。
# sen_vec = [vec, vec, vec, ...] / n
# 2. 排成一个大matrix (M * N)，等着CNN来搞
# [ vec | vec | vec | vec | ... ]
# 3. 用Doc2Vec。这是基于句子的vec，跟word2vec差不多思路，用起来也差不多。
# 只对长篇 大文章效果好。对头条新闻，twitter这种的东西，就不行了。每个"篇"的句子太少。
# 具体可以看gensim。

# Anyway, 这一步完成后，你会对于每个训练集的X，得到一个固定长度的vec或者matrix
# 接下来的事情，大家就可以融会贯通了。
# 比如，可以用前面冯老师讲的RF跑一遍 做classification。
```

以上

有没有发现哪些点可以给第一节课那种最简单的Rule-base机器人做升级?

感谢大家！

恳请大家批评指正！