2017编程之美挑战赛

NLP,人工智能机器人

### Bot Framework Overview

构建和部署高性能的服务型机器人,该框架包括 Bot构建SDK, Bot连接器,开发者入门,Bot目录,以及一个模拟器.

Bot是指一个可以与用户进行对话式交互的网络服务, 用户可以通过任何方式使用该服务.对话的内容可以是简单的,也可以是复杂的(比如显示卡片选项).

框架提供了 .NET SDK和Node.js SDK来构建机器人, 这些SDK提供了对话框,内置提示等.如果使用其他语言,参考REST APIhttps://docs.botframework.com/en-us/restapi/connector/. 同时,这个SDK本身是开源的,地址: <https://github.com/Microsoft/BotBuilder>

当完成了Bot之后,应当注册并发布它.注册方式: <https://dev.botframework.com/bots/new> 注册完毕之后还应当进行配置,比如在微信或者skype上.

### 使用LUIS训练模型

首先在<https://www.luis.a> 上创建一个账户,然后创建应用, 然后开始定义Intent, Entity, 输入utterance,然后开始训练整个模型.

通过HTTP将这个应用发布出去.

Entity应当是一个具体值的类型

可以添加固定短语

可以添加正则表达式

### 使用LUIS远端模型在本地编程

一个LuisRecognizer(model 🡪 url)即可获取所有的Luis模型.

使用IntentDialog({ recognizers: [recognizer]});来构建一个dialog.

intent.matches( NAME\_OF\_INTENT,…) 来执行匹配的动作.

### 使用Bot Framework Channel Emulator通信

如果使用的nodejs开启的服务器,则需要记录其中的APPID和PASSWORD,在模拟连接的界面输入.

如果使用的是vs的c#, 可以输入微软的账户名和密码连接.

### Bot Nodejs的编程范式

##### 主框架

过程概览:

导入botbuilder包,导入restify包

设置本地监听端口,连接使用的用户名和密码,设置本地服务器的根目录

配置dialog // 调用bot.dialog

bot.dialog(‘/’,…)

第二个参数可以是一个函数, 函数列表,或者dialog类型.

var builder = require('botbuilder');

var restify = require('restify');

//=========================================================

// Bot Setup

//=========================================================

// Setup Restify Server

var server = restify.createServer();

server.listen(process.env.port || process.env.PORT || 3978, function () {

console.log('%s listening to %s', server.name, server.url);

});

// Create chat bot

var connector = new builder.ChatConnector({

appId: process.env.MICROSOFT\_APP\_ID,

appPassword: process.env.MICROSOFT\_APP\_PASSWORD

});

console.log(process.env.MICROSOFT\_APP\_ID);

console.log(process.env.MICROSOFT\_APP\_PASSWORD);

var bot = new builder.UniversalBot(connector);

server.post('/api/messages', connector.listen());

//=========================================================

// Bots Dialogs

//=========================================================

var intents = new builder.IntentDialog();

bot.dialog('/', intents);

intents.matches(/^change name/i, [

function (session) {

session.beginDialog('/profile');

},

function (session, results) {

session.send('Ok... Changed your name to %s', session.userData.name);

}

]);

intents.onDefault([

function (session, args, next) {

if (!session.userData.name) {

session.beginDialog('/profile');

} else {

next();

}

},

function (session, results) {

session.send('Hello %s!', session.userData.name);

}

]);

bot.dialog('/profile', [

function (session) {

builder.Prompts.text(session, 'Hi! What is your name?');

},

function (session, results) {

session.userData.name = results.response;

session.endDialog();

}

]);

##### dialog

* [simple] function or list of functions
* SimpleDialog
* IntentDialog
  + matches 在匹配正则表达式的时候调用
* LuisDialog

##### intent

## 开发Bot应用的关键

1. 交互处理逻辑
2. 数据来源,训练集和增量训练方式
3. 呈现的UI质量
4. 软件架构
5. 逻辑处理的服务

Bot功能性、鲁棒性、UI 设计

### 资格赛,初赛和决赛

资格赛

Document-Based QA

初赛

决赛: Everything@HIT

##### 附录:其他BotFrameWork的函数

builder.Prompts.text(session,”WHAT”);

* session

Reference : <https://docs.botframework.com/en-us/node/builder/chat/session/#navtitle>

session.send(fmt, args) 发送回复,通常发送的回复是以批(batch)的形式发送的, 每次send之后都会计时250ms,如果超时就发送,否则继续等待(参考<https://docs.botframework.com/en-us/node/builder/chat/session/#navtitle>)

send使用sprintf实现

session.beginDialog(dialogName) 开始一个通过bot.dialog定义的对话

session.endDialog() 结束对话, 后面的函数不会被调用

session.sendTyping() 显示正在输入(如果后面有输入,这个显示会被立即覆盖)

## 附录:使用ngrok调试

在命令行调用

ngrok http port

可以将本地的端口暴露出去,在运行之后获得一个ngrok的代理服务器的地址.访问代理服务器即可开始调试.

资格赛

Document-Based Question Answer

## 任务要求

资格赛任务题是基于文档的问答任务（Document-based Question Answering task, DBQA），它是对于给定的一篇文档（Document）和一个从文档中提出的自然语言问题(Question)，参赛队伍需要使用提供的数据集训练模型算法，让模型可以回答问题，回答时仅限于从组成该文档的句子中选出能回答该问题的句子（Answer Selection in Question Answering）。鼓励参赛队伍发挥算法创造力并使用各种资源来训练模型，比如句子匹配模型（Sentence Matching Model），以使模型能准确地回答问题。

要点: 1.使用任何可用的资源来训练模型

2.回答问题的答案只能来自文档本身

## What is Document-based Question Answer

Reference: <https://gallowayapworld.wordpress.com/essay-writing-materials/document-based-questions/>

The document-based question is sometimes better understood as an evidence-based question: you will be given a question on an historical topic, and then provided with a series of documents, charts, maps, photographs, paintings, and other sources as evidence you will use to formulate your answer.

对于DBQA, 技巧比知识本身更有效.

## 想法

基于支持向量机: 将句子分析成树,用向量来表示树的结构; 然后使用支持向量机进行训练, 得到最终的结果.使用未测试的数据进行测试.(SVM with Tree Kernel)

将神经网络应用于句子相似度比较(深度学习的例子):

* 文章相似度(句子较, 基于大型语句)

几个用于表示文章的模型(即向量化的方法)

词袋模型

TF-IDF(主题词在某一文章中的相似度)

LSI模型(潜在主题)

LDI模型

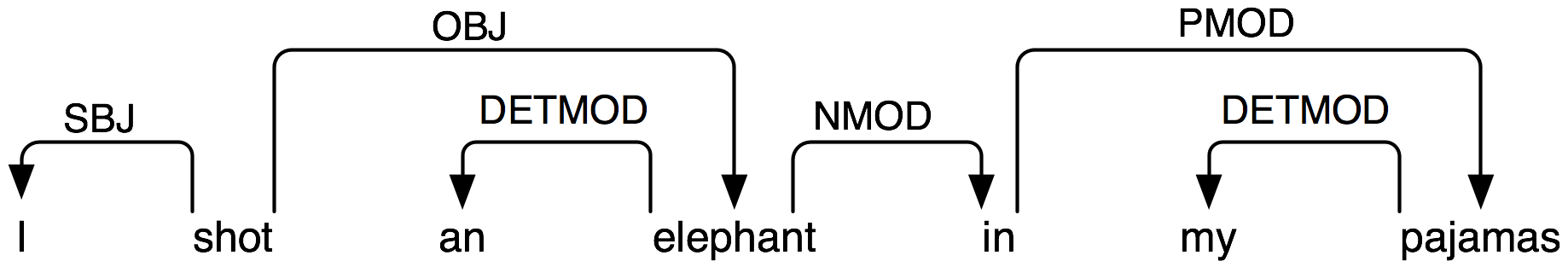
参考此处 <http://www.jianshu.com/p/edf666d3995f> python文本相似度计算

## 参考

一个基于IBM Waston服务的QA系统的例子

<http://fartashh.github.io/post/qa-system-watson/>

## 附录: 依赖文法



使用nltk创建

**>>> groucho\_dep\_grammar = nltk.DependencyGrammar.fromstring("""**

**... 'shot' -> 'I' | 'elephant' | 'in'**

**... 'elephant' -> 'an' | 'in'**

**... 'in' -> 'pajamas'**

**... 'pajamas' -> 'my'**

**... """)**

**>>> print(groucho\_dep\_grammar)**

**Dependency grammar with 7 productions**

**'shot' -> 'I'**

**'shot' -> 'elephant'**

**'shot' -> 'in'**

**'elephant' -> 'an'**

**'elephant' -> 'in'**

**'in' -> 'pajamas'**

**'pajamas' -> 'my'**

如果一个依赖文法是投影的(projective), 即所有的边没有相交的部分,可以以线性的顺序将所有的词语写出.

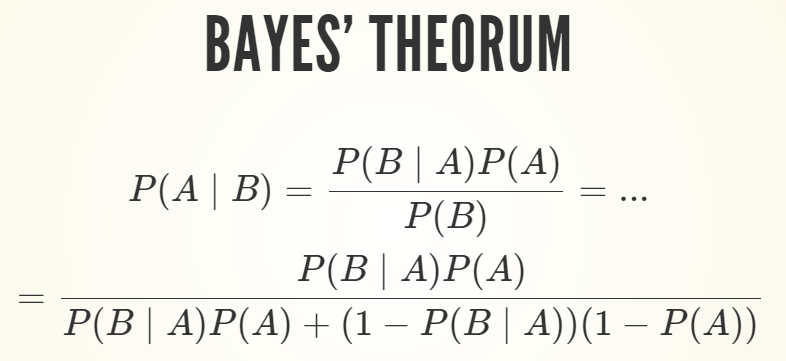
## 附录:汉语分词和标注

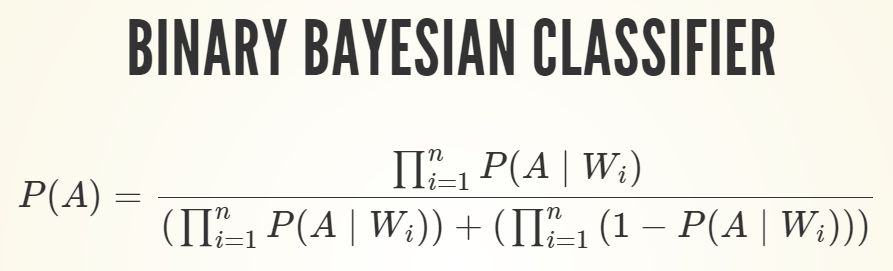
参考库pynlpir

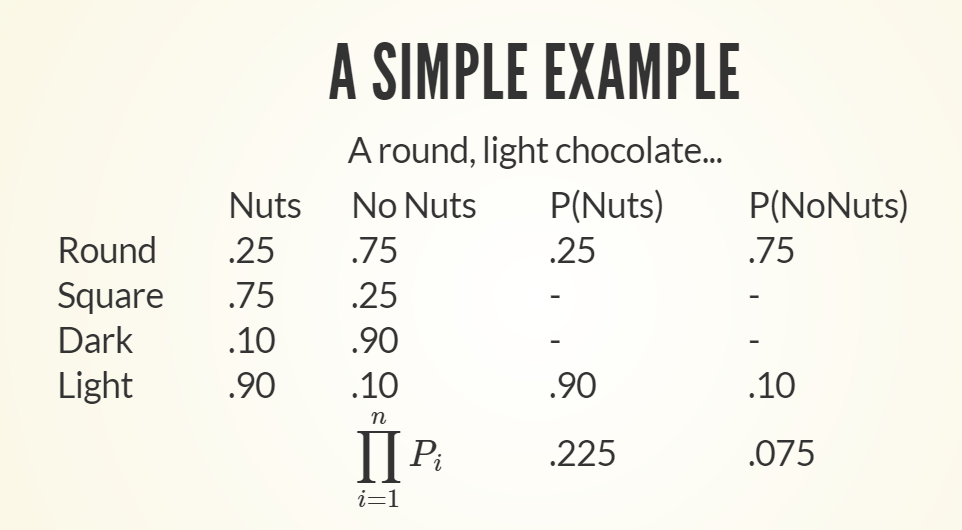
## 附录:Naïve Bayesian Classification

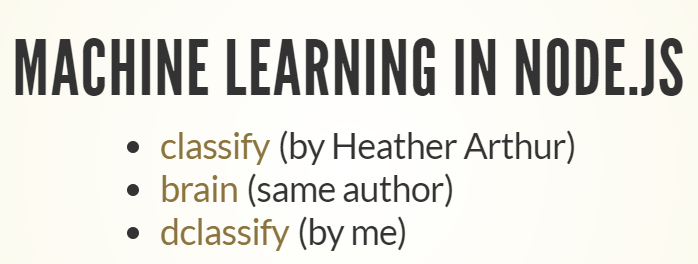
simple, yet surprisingly effective

参考: <http://73rhodes.github.io/talks/MachineLearning/#/4>









## 附录:安装错误及解决方法

* MSBUILD : error MSB4132: The tools version "2.0" is unrecognized. Available tools versions are "4.0". #60

open up a new cmd as administrator and run this command:  
npm install --global --production windows-build-tools  
then  
npm config set msvs\_version 2015 --global  
close all instances of shell/cmd, reopen a cmd (regular this time, non-administrator) return to your directory where you are trying to run npm install and run it again

refer: <https://github.com/chjj/pty.js/issues/60>

* 安装scipy的错误

在cygwin目录下搜索 cyglto\_plugin.dll 查看对应的gfortran版本号,然后下载对应的版本即可,

* sklearn ImportError

在某个目录下. from . import \_iterative显示错误.

修复方案: 此目录下有一个文件: **\_iterative.cpython-36m-x86\_64-cygwin.dll**

只需要再增加一个文件: \_iterative.py , 内容如下:

import ctypes

api = ctypes.cdll.LoadLibrary("\_iterative.cpython-36m-x86\_64-cygwin.dll")

LoadLibrary之后\_iterative的命名空间中会多出许多函数.这就正确了.

可以参考: <http://stackoverflow.com/questions/43917793/what-does-ctypes-cdll-loadlibrary-really-do>

* 安装png和freetype

png可以使用pypng, freetype可以上官网搜索下载.

* python-embeded安装get-pip

首先下载get-pip.py

<https://bootstrap.pypa.io/get-pip.py>

然后按照issues: <https://github.com/pypa/get-pip/issues/7>

去掉注释之后成功安装

* python-embeded安装,使用和更新pynlpir

需要更新Data和liscence,然后使用

python -m pynlpir.cli update

* python2坑很多

如果决定自己编译python2, 那么就有很多后续工作要做.

典型的问题有以下两类:

1. 内置模块缺失(如bz2,sqlite3)

先在系统级别安装bz2,sqlite3的库, 一般是 libxxx-devel之类的,比如libbzip2-devel

然后重新编译python,prefix选择相同的路径即可.

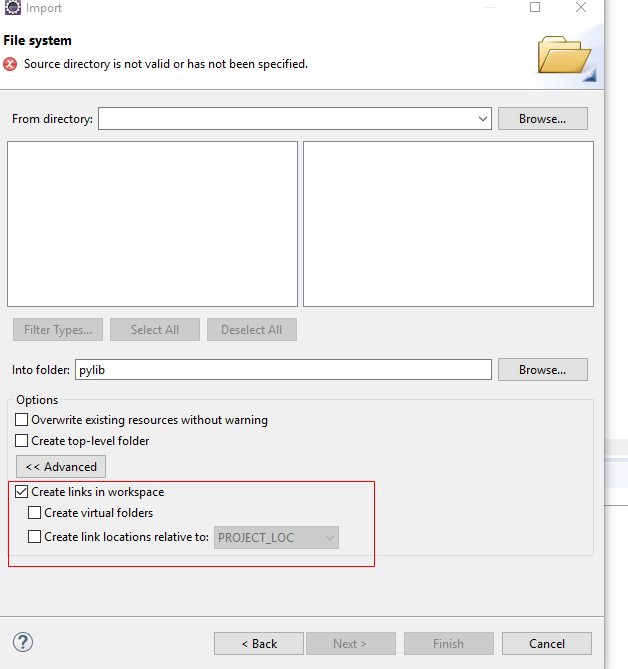
1. 外置模块无法通过pip安装

下载源码, 一般直接通过python setup.py – 参数可以安装.

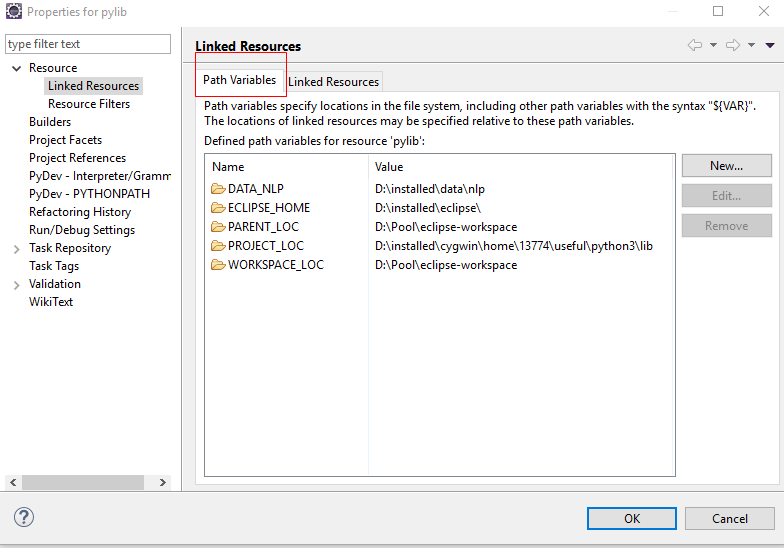
第三类问题,如果修改了python2导致处于不一致的状态, 这时需要做的就是重装一个稳定的(低版本)的python,覆盖原来的安装.

## 附录:eclipse import和建立符号链接

添加常规link



添加PATH



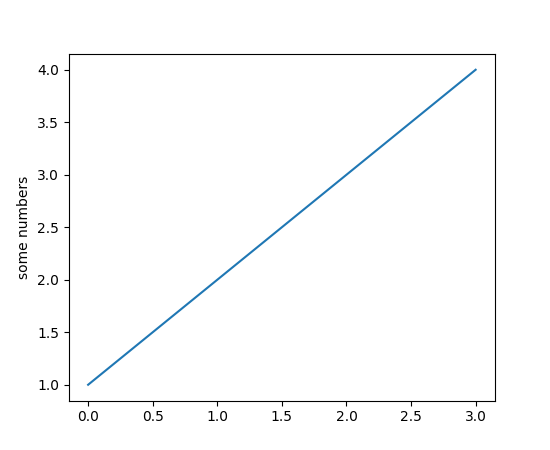
## 附录:matplotlib数据可视化

**import** **matplotlib.pyplot** **as** **plt**

plt.plot([1,2,3,4])

plt.ylabel('some numbers')

plt.show()



* axis

The [axis()](https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.axis) command in the example above takes a list of [xmin, xmax, ymin, ymax] and specifies the viewport of the axes.

## 附录:ML, NLP参考

Nodejs机器学习 <https://www.npmjs.com/package/machine_learning>

<http://joonku.com/project/machine_learning/apidoc>

Nodejs 自然语言处理 <https://github.com/NaturalNode/natural#installation>

Python3自然语言处理 <http://www.nltk.org/book/>

使用库的基础步骤 Import library, object creation, fitting model and prediction

Python3 scikit-learn手写数字识别的案例 <http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#example-classification-plot-digits-classification-py>

第二个案例 <http://www.jianshu.com/p/d8664bf9adf9>

SVM的讲解 <https://www.analyticsvidhya.com/blog/2015/10/understaing-support-vector-machine-example-code/>

## 附录:SVM使用及其案例

在nodejs的machine\_learning库中,新建一个svm代码如下:

var ml **=** require('machine\_learning');

var x **=** [[0.4, 0.5, 0.5, 0.,  0.,  0.],

         [0.5, 0.3,  0.5, 0.,  0.,  0.01],

         [0.4, 0.8, 0.5, 0.,  0.1,  0.2],

         [1.4, 0.5, 0.5, 0.,  0.,  0.],

         [1.5, 0.3,  0.5, 0.,  0.,  0.],

         [0., 0.9, 1.5, 0.,  0.,  0.],

         [0., 0.7, 1.5, 0.,  0.,  0.],

         [0.5, 0.1,  0.9, 0.,  **-**1.8,  0.],

         [0.8, 0.8, 0.5, 0.,  0.,  0.],

         [0.,  0.9,  0.5, 0.3, 0.5, 0.2],

         [0.,  0.,  0.5, 0.4, 0.5, 0.],

         [0.,  0.,  0.5, 0.5, 0.5, 0.],

         [0.3, 0.6, 0.7, 1.7,  1.3, **-**0.7],

         [0.,  0.,  0.5, 0.3, 0.5, 0.2],

         [0.,  0.,  0.5, 0.4, 0.5, 0.1],

         [0.,  0.,  0.5, 0.5, 0.5, 0.01],

         [0.2, 0.01, 0.5, 0.,  0.,  0.9],

         [0.,  0.,  0.5, 0.3, 0.5, **-**2.3],

         [0.,  0.,  0.5, 0.4, 0.5, 4],

         [0.,  0.,  0.5, 0.5, 0.5, **-**2]];

var y **=**  [**-**1,**-**1,**-**1,**-**1,**-**1,**-**1,**-**1,**-**1,**-**1,**-**1,1,1,1,1,1,1,1,1,1,1];

var svm **=** **new** ml.SVM({

    x **:** x,

    y **:** y

});

svm.train({

    C **:** 1.1, *// default : 1.0. C in SVM.*

    tol **:** 1e-5, *// default : 1e-4. Higher tolerance --> Higher precision*

    max\_passes **:** 20, *// default : 20. Higher max\_passes --> Higher precision*

    alpha\_tol **:** 1e-5, *// default : 1e-5. Higher alpha\_tolerance --> Higher precision*

    kernel **:** { type**:** "polynomial", c**:** 1, d**:** 5}

*// default : {type : "gaussian", sigma : 1.0}*

*// {type : "gaussian", sigma : 0.5}*

*// {type : "linear"} // x\*y*

*// {type : "polynomial", c : 1, d : 8} // (x\*y + c)^d*

*// Or you can use your own kernel.*

*// kernel : function(vecx,vecy) { return dot(vecx,vecy);}*

});

console.log("Predict : ",svm.predict([1.3,  1.7,  0.5, 0.5, 1.5, 0.4]

其中x是二维数据,每一维都是一个向量,而y则是相应标签(-1,+1)

* fit和predict

Estimators predict a value based on the observed data. In scikit-learn, all estimators implement the fit() and predict() methods.

* 参数c的函数

参考 <https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel>

作为惩罚系数,当C越大时,越倾向于选择完美分类的超平面;当C越小时,越倾向于选择具有最佳分类距离的超平面.

* 案例:小型图像识别
* 案例:直线和圆的识别

圆方程:x^2 + y^2 = 1

直线: y=2x+1

利用方程生成随机数据集:

只要随机生成一个合法的x即可, 随机选择一个方程生成它的y值(如果方程不满足y值就抛弃),对选中的方程记录其标签号

f1, f2 may return None

**import** random

**def** **getRandomSet**(n, f1,f2,x0,x1):**#n number of datasets**

**'''n: size**

**f1,f2**

**x0,x2**

**'''**

f=[f1,f2]

data=[ [],[] ]

i=**0**

**while** i<n:

x=random.uniform(x0,x1)

choose=**int**(random.choice(**'01'**))

y=f[**int**(random.choice(**'01'**))](x)

**if** y **is** **not** **None**:

data[**0**].append([x,y])

data[**1**].append(**2**\*choose-**1**)

i=i+**1**

**return** data

## 附录:关于pynlpir的使用

userdict的格式:

山东科技大学有 noun

北京花了 noun

何时需要Init和Exit?

不需要手动Init和Exit.

## 附录:genism

参考 <http://radimrehurek.com/gensim/tut1.html>

关于doc2vec <https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-lee.ipynb>

关于: Doc2Vec is a great way to explore relationships between documents.

但是doc2vec对于较小的输入其结果并不理想.

关于Python计算文本相似度 <http://www.jianshu.com/p/edf666d3995f>

## 附录gensim加载Wiki数据

首先准备的wiki数据是压缩形式的,形如 xxwiki-date.xml.bz2

使用下面的代码解压数据(输出到文件,将xml转换成普通的句子文件,句子按行分, 词按空格分)

from gensim.corpora import WikiCorpus

infile=’data/….xml.bz2’

output=open(‘data/….extract.data’,’wb’)

wiki=WikiCorpus(infile, lemmatize=False,dictionary={})

for text in wiki.get\_texts():

output.write(b’ ‘.join(text) + b’\n’)

output.close()

注意: get\_texts()迭代器返回的是句子的集合.

预处理:

简繁转换

大量的无关的英文词汇去除

中文正则表达式: [\u4e00-\u9fa5]

去除不含中文的词语

>>> regex=re.compile(u'.\*[\u4e00-\u9fa5].\*')

>>> for i in jieba.cut(u'what is 通常被认为是数学的诺贝尔奖 the hell'):

... m=regex.match(i)

... print('if m is None?',m is None)

... print(i)

...

('if m is None?', True)

what

('if m is None?', True)

('if m is None?', True)

is

('if m is None?', True)

('if m is None?', False)

通常

('if m is None?', False)

被

('if m is None?', False)

认为

('if m is None?', False)

是

('if m is None?', False)

数学

('if m is None?', False)

的

('if m is None?', False)

诺贝尔奖

('if m is None?', True)

('if m is None?', True)

the

('if m is None?', True)

('if m is None?', True)

hell

>>> myf=lambda x:regex.match(x) is not None

>>> filter(myf,jieba.cut(u'what is 通常被认为是数学的诺贝尔奖 the hell'))

[u'\u901a\u5e38', u'\u88ab', u'\u8ba4\u4e3a', u'\u662f', u'\u6570\u5b66', u'\u7684', u'\u8bfa\u8d1d\u5c14\u5956']

>>> print(' '.join(filter(myf,jieba.cut(u'what is 通常被认为是数学的诺贝尔奖 the hell'))))

通常 被 认为 是 数学 的 诺贝尔奖

* LineSentece

classgensim.models.word2vec.LineSentence(source, max\_sentence\_length=10000, limit=None)

Bases: object

Simple format: one sentence = one line; words already preprocessed and separated by whitespace.

source can be either a string or a file object. Clip the file to the first limit lines (or no clipped if limit is None, the default).

Example:

sentences = LineSentence('myfile.txt')

Or for compressed files:

sentences = LineSentence('compressed\_text.txt.bz2')

sentences = LineSentence('compressed\_text.txt.gz')

## 附录:jieba附录

import jieba

inStr=’我们一起来打渔吧’

segList=jieba.cut(inStr, cut\_all=False|True) #cut\_all = False表示需要向前搜索, 默认cut\_all=False

print(‘, ‘.join(segList))

* 几个评测:先分词,再繁简转化的对比

>>> g=utils.zhWikiProcess(wiki,cutter=utils.pynlpirCutter,tosimplified=True,stopwords=None,keep\_others=False,join='\t')

>>> print('\n'.join(utils.getFirstN(g,10)))

欧 几 里 得

西 元 前 三 世 纪 的 希 腊 数 学 家

现 在 被 认 为 是 几 何 之 父

此 画 为 拉 斐 尔 的 作品

雅典 学 院

数学

是 利用 符号 语言 研究 数 量

结构

变化 以及 空间 等 概念 的 一 门 学科

从 某种 角度 看 属 于 形式 科 学 的 一 种

>>> g=utils.zhWikiProcess(wiki,tosimplified=True,stopwords=None,keep\_others=False,join='\t')

>>> print('\n'.join(utils.getFirstN(g,10)))

欧几里 得

西元前 三世 纪的 希腊 数学家

现在 被 认为 是 几何 之父

此画 为拉斐尔 的 作品

雅典 学院

数学

是 利用 符号语言 研究 数量

结构

变化 以及 空间 等 概念 的 一门 学科

从 某种 角度看 属 于 形式 科学 的 一种

可以看到jieba表现更好

* 先转换,再分词

>>> g=utils.zhWikiProcess(wiki,cutter=utils.pynlpirCutter,tosimplified=True,stopwords=None,keep\_others=False,join='\t')

>>> print('\n'.join(utils.getFirstN(g,10)))

欧 几 里 得

西 元 前 三 世纪 的 希腊 数学家

现在 被 认为 是 几何 之 父

此 画 为 拉斐尔 的 作品

雅典 学院

数学

是 利用 符号 语言 研究 数量

结构

变化 以及 空间 等 概念 的 一 门 学科

从 某种 角度 看 属于 形式 科学 的 一 种

>>> g=utils.zhWikiProcess(wiki,tosimplified=True,stopwords=None,keep\_others=False,join='\t')

>>> print('\n'.join(utils.getFirstN(g,10)))

欧几里得

西元前 三 世纪 的 希腊 数学家

现在 被 认为 是 几何 之 父

此画 为 拉斐尔 的 作品

雅典 学院

数学

是 利用 符号语言 研究 数量

结构

变化 以及 空间 等 概念 的 一门 学科

从 某种 角度看 属于 形式 科学 的 一种

综上来看,最好的效果是先转换,再分词

## 附录: zhconv简繁转换的使用

import zhconv

for text in wiki.get\_texts():

for sent in text:

simplifiedSent=zhconv.convert(sent..decode(‘u8’),’zh-cn’)

out.write(simplifiedSent + ‘\n’) #convert to LineSentence

out 可以存成bz2格式

## 附录:数个github库的使用

* 中文词向量

Googlenews: <https://code.google.com/archive/p/word2vec/>

github地址: <https://github.com/Kyubyong/wordvectors>

~~可能由于版本的原因, 直接下载的中文词向量不能使用,所以最好是基于他的代码训练出中文的词向量.~~

经过测试是能用的,先前只是加载函数用错了.加载函数为 gensim.models.Word2Vec.load(file)

~~首先是将其改成python3的脚本, 需要修改几个地方.~~

~~然后是下载对应的wiki数据,更改build\_corpus中的fname~~

~~使用python build\_corpus.py –lcode=zh开始构建.~~

python build\_corpus.py –lcode=zh 构建的输出是data/zh.txt

但是这还不是最终的词向量模型.这一步可以理解为将xml转化成了标准的文本集合(包含上下文信息的顺序),最终获得的结果就是分词获得的结果.

还需要使用get\_vectors.py来进一步训练.

当然,可以直接一步使用make\_wordvectors.h来获得词向量,最终生成的结果是

1. 可以直接载入的模型 data/zh.bin
2. data/zh.tsv 对第i个词的向量进行打印

* WMD 词移动距离

github地址: <https://github.com/mkusner/wmd>

参考: <http://vene.ro/blog/word-movers-distance-in-python.html>

先将脚本改成python3,然后更改google对应的词向量文件,按照指示运行即可.

* SWMD 监督词移动距离

github地址: <https://github.com/ByronPan/sWMD>

下载数据之后按照提示运行即可, ipython可以用Python替代.

## 附录: Word2Vec

参考 <http://www.52nlp.cn/tag/gensim> 一位同学基于wiki word2vec的实验

word embeding也称为词向量

* 基于word2vec训练中文词语

首先,这里使用gensim.word2vec

1. 训练文本集

import gensim

from gensim.models import word2vec

sentences=word2vec.Text8Corpus(路径) #文本集不必去除­­­停用词,但最好去除吧

# sentences=word2vec.LineSentence(路径)

model=gensim.models.Word2Vec(sentences, size=200, min\_count=3)

#min\_count对相关性计算有影响,很多词仅仅出现了一次

model.save(存储路径)

print(model.similarity(‘北京’, ‘中国’))

注意: sentences是一个gensim的对象,所以最好从文件读取.

训练参数参考:

Hyper-parameters:

word vector dimension: 300

window: 5

min\_count: 5

epochs: 3

Training algorithm: skip-gram (sg = 1)

Model training: hierarchical sampling (hs = 1)

Note: syn0 and syn1 files are produced by saving model using model.save(), which allows to continue training the model when loaded. The Text file is produced by saving model using model.save\_word2vec\_format(), which allows to view the vectors by a sublime-like software.

1. 加载训练好的模型

model1=gensim.models.Word2Vec.load(…)

y=model1.most\_similarity(‘厉害’,topn=20)

import gensim

# Load Google's pre-trained Word2Vec model.

model = gensim.models.KeyedVectors.load\_word2vec\_format('./GoogleNews-vectors-negative300.bin', binary=True)

###### 关于dim的评测

对于从wiki训练出来的向量模型, 300和50在效果上没有明显的差别.维数低的更适于计算用.

## 附录:BOW (Bag Of Words)

翻译称词袋模型

假定对于一个文本，忽略其词序和语法，句法，将其仅仅看做是一个词集合.

表示文档时, 使用次序固定的词表,仅仅记录出现次数即可.

例子

文档一：Bob likes to play basketball, Jim likes too.  
  
文档二：Bob also likes to play football games.

Dictionary = {1:”Bob”, 2. “like”, 3. “to”, 4. “play”, 5. “basketball”, 6. “also”, 7. “football”，8. “games”, 9. “Jim”, 10. “too”}

表示:

1：[1, 2, 1, 1, 1, 0, 0, 0, 1, 1]  
2：[1, 1, 1, 1 ,0, 1, 1, 1, 0, 0]

丢弃次序,只要统计.

## 附录:独热编码

独热编码基于n个数据集,每个数据集有mi个维度;

独热编码将一个n维的原始数据转化为 m1+m2…mn的向量.

比如:

性别:m,f

地区: us, en, cn

浏览器 : firefox, chrome,ie

表明一个来自us的使用ie的f记为:

[1,0,2] 🡪 [ [1 0] [0 0 1] [ 1 0 0]]🡪[ 1 0 0 0 1 1 0 0]

即将第i位置为1,其他置为0.

python代码:

from sklearn import preprocessing

enc=preprocessing.OneHotEncoder()

enc.fit([ [ 1 0 2] ] )

array=enc.transform( [[0 1 1]]).toarray()

print(array)

关于fit, 需要最多的不同个数的值

**>>> from** **sklearn.preprocessing** **import** OneHotEncoder

**>>>** enc = OneHotEncoder()

**>>>** enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])

OneHotEncoder(categorical\_features='all', dtype=<... 'numpy.float64'>,

handle\_unknown='error', n\_values='auto', sparse=True)

**>>>** enc.n\_values\_

array([2, 3, 4])

**>>>** enc.feature\_indices\_

array([0, 2, 5, 9])

**>>>** enc.transform([[0, 1, 1]]).toarray()

array([[ 1., 0., 0., 1., 0., 0., 1., 0., 0.]])

## 附录:Logistics回归

参考 <http://www.2cto.com/database/201610/553143.html>

* 回归的基本概念

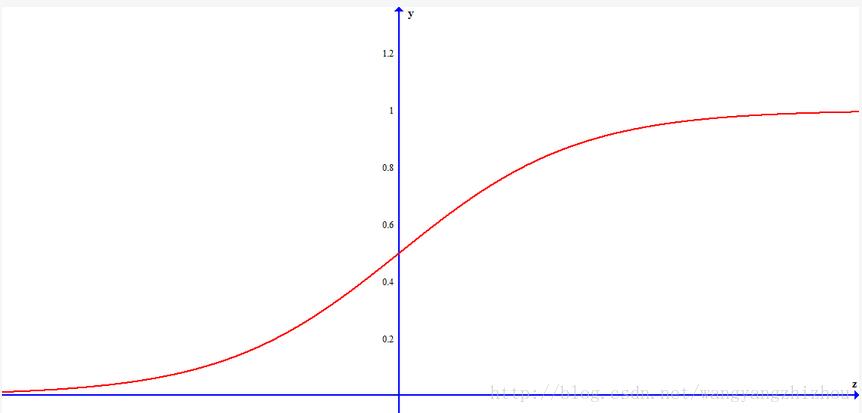
是给定一个点集，能够用一条曲线去拟合之，如果这个曲线是一条直线，那就被称为线性回归，如果曲线是一条二次曲线，就被称为二次回归，回归还有很多的变种，如locally weighted回归，logistic回归，

对于Logistics来说, 回归不是寻找一条曲线使得所有的点都落在这条曲线上,而是寻找一条曲线恰好能够分类不同的点.

更广义的回归可以理解为, 对于点集V, 满足f(V)=0, 回归即求f’, 使得f(V’)=0时f’(V’)=0.

sigmoid函数(二值,可取中连续函数)

 值域: 0~1



对于任意的x,sigmoid函数都能返回一个介于0~1之间的实值.

对于输入I(多维向量), 将其通过theta变换成theta(I), 然后通过sigmoid函数取theta(I)的值获得分类.(theta是多项式叠加的形式,形如 )

问题是:如何确定theta的n+1个参数?

首先, sigmoid函数表明的是概率P(x属于1类|x)=sigmoid(theta(x)), 即一组数据的分类只与sigmoid函数相关.

则有P(x属于0类|x)=1-sigmoid(theta(x))= , 概率比为e^x:1.

下面就属于求w0,…wn这n+1个参数的过程, 使用的是最大似然估计:

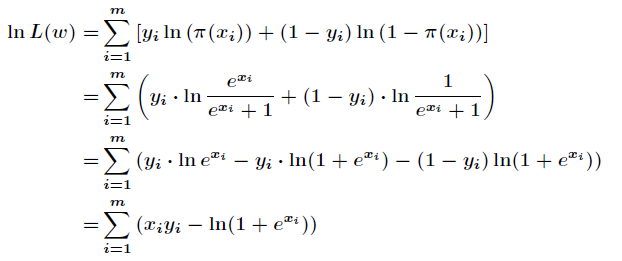
因为p(yi=1|xi)=pi, p(yi=0|xi)=1-pi

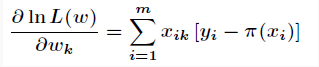
p(yi|xi)=pi^yi \* (1-pi)^(1-yi) 这是统一表达式

对于m个数据,有似然函数:

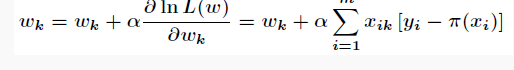
L(w)=  (m个数据的联合分布)

当L(w)具有最大值时的概率分布的w取值即为所求参数.





根据梯度上升法



如果用tensorflow写出来,表达式就是:

输入为x,输出为y

tf.reduce\_sum(tf.multiply(x,tf.subtract(y, tf.sigmoid(x)))

但是请注意, tf.train.GradientDescentOptimizer在训练时是以函数作为输入参数,而不是以求得的梯度作为参数.

Logistics以最大似然函数作为估计.(而同样可以以它的对数函数为输入)

更一般来说, 梯度下降算法以一个待求极大值(或者极小值)的函数作为输入.

注意一点:tf.log以e为底.

注意:tf的标量乘法,即使施加于向量上(rank=1),比如

[1.0,2.0] \* [4.0,5.0] 结果 是 [4.0,10.0]

[1.0,2.0] \* [4.0,5.0]T 结果 是 [

[4.0, 8.0]

[5.0, 10.0]

]

而matmul只能施加于[ [] ]的形式,不能施加于[ ]的形式.

注意:reduce\_mean其实只是reduce\_sum的一个变种.

tf.reduce\_mean(x) ==> 2.5 #如果不指定第二个参数，那么就在所有的元素中取平均值

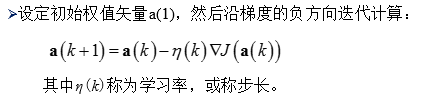
tf.reduce\_mean(x, 0) ==> [2., 3.] #指定第二个参数为0，则第一维的元素取平均值，即每一列求平均值

tf.reduce\_mean(x, 1) ==> [1.5, 3.5] #指定第二个参数为1，则第二维的元素取平均值，即每一行求平均值

* 梯度下降法

假设有向量a, 需要求a的一个取值,使得性质J(a)取极小值, 这个就是最优化过程.

梯度下降的思想是每一次迭代求解, 都沿着梯度下降的方向

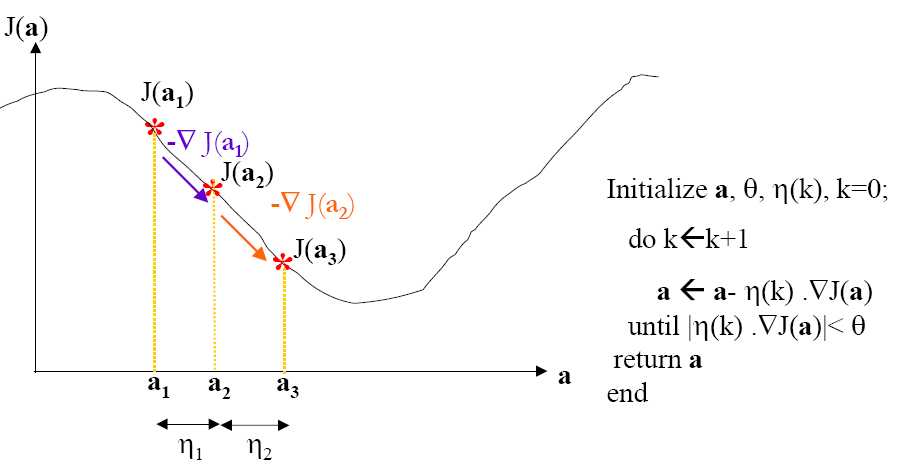


(注:梯度是标量值下降最快的方向

)

因为梯度下降最快, 所以每一次a的减少也最大,能尽快逼近局部极小值.

梯度上升可以用来求极大值.



* 最大似然估计

如果在theta条件下x的出现为x1,x2,…,xn,认为theta有利于xi的出现.求这种出现的概率最大值.

则认为p(x1,x2,…,xn|theta)

* 一个简单的实例

import tensorflow as tf

from numpy import \*

x\_train = [[1.0, 2.0], [2.0, 1.0], [2.0, 3.0], [3.0, 5.0], [1.0, 3.0], [4.0, 2.0], [7.0, 3.0], [4.0, 5.0], [11.0, 3.0],

[8.0, 7.0]]

y\_train = [1, 1, 0, 1, 0, 1, 0, 1, 0, 1]

y\_train = mat(y\_train) #把向量放到矩阵中才能施加向量运算

theta = tf.Variable(tf.zeros([2, 1]))

theta0 = tf.Variable(tf.zeros([1, 1]))

y = 1 / (1 + tf.exp(-tf.matmul(x\_train, theta) + theta0))

loss = tf.reduce\_mean(- y\_train.reshape(-1, 1) \* tf.log(y) - (1 - y\_train.reshape(-1, 1)) \* tf.log(1 - y))

train = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

init = tf.initialize\_all\_variables()

sess = tf.Session()

sess.run(init)

for step in range(1000):

sess.run(train)

print(step, sess.run(theta).flatten(), sess.run(theta0).flatten()

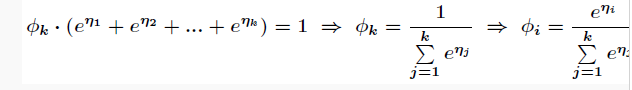
## 附录:softmax回归

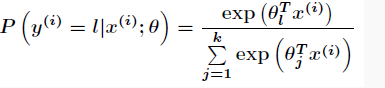
softmax是Logistics回归的扩展, 当面对一个多分类问题时常用softmax回归.

但是注意, 除非分类的问题之间是正交的,softmax能够取得较好效果,其他情况下,如人,科学家,狗的分类,因为人和科学家是有交叉到,更适合单独与狗进行二值回归.

在Logistics回归中,y的取值是{0,1},而在softmax中扩展到{1,2,…,k},k是类别数目.

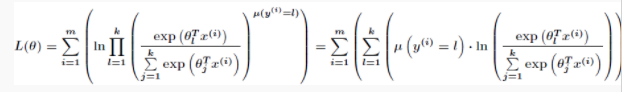
softmax的概率估计函数为:





ni为待训练的参数.

对数似然函数为



* 一个识别手写数字的例子

#!/usr/bin/env python

import tensorflow as tf

import numpy as np

import tensorflow.examples.tutorials.mnist.input\_data as input\_data

# read MNIST data set

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

#trX, trY, teX, teY = mnist.train.images, mnist.train.labels, mnist.test.images, mnist.test.labels

# create symbolic variables

X = tf.placeholder(tf.float32, [None, 784])

Y = tf.placeholder(tf.float32, [None, 10])

# create variables: weights and biases

W = tf.Variable(tf.zeros([784, 10]))

b = tf.Variable(tf.zeros([10]))

# define model

y = tf.nn.softmax(tf.matmul(X, W) + b)

# cross entropy

cross\_entropy = -tf.reduce\_sum(Y \* tf.log(y))

# train step

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)

# init step

init = tf.initialize\_all\_variables()

with tf.Session() as sess:

    # run the init op

    sess.run(init)

    # then train

    for i in range(1000):

        batch\_trX, batch\_trY = mnist.train.next\_batch(128)

        sess.run(train\_step, feed\_dict={X: batch\_trX, Y: batch\_trY})

    # test and evaluate our model

    correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(Y, 1))

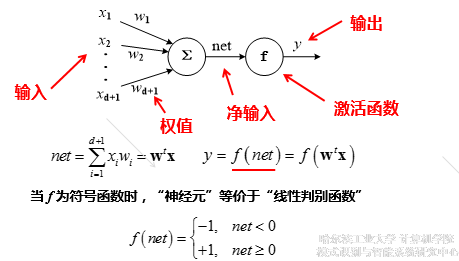
    accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

    print sess.run(accuracy, feed\_dict={X: mnist.test.images, Y: mnist.test.labels})

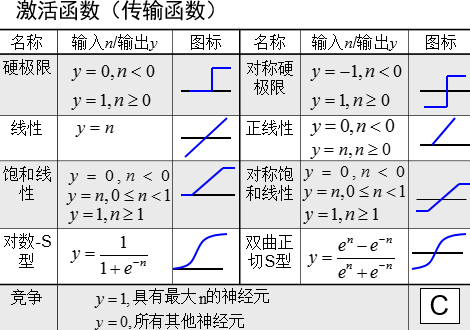
## 附录:CNN

* 预备知识:神经元和感知器网络

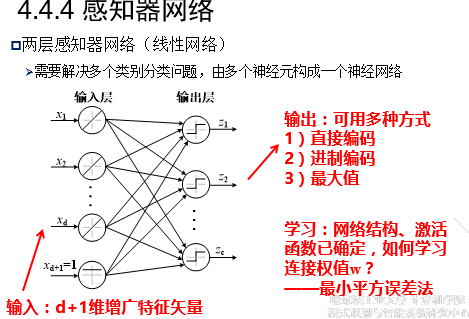
人工神经元模型:



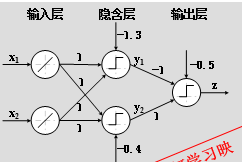
激活函数的类型:

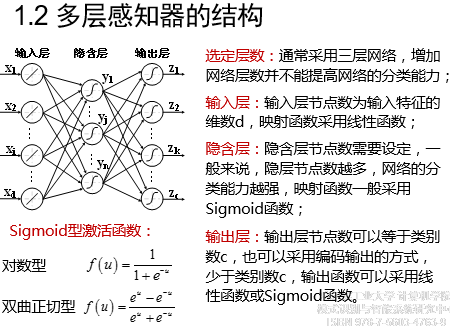


感知器网络:神经元之间构成网络,输出表征分类, 多个z之间竞争.

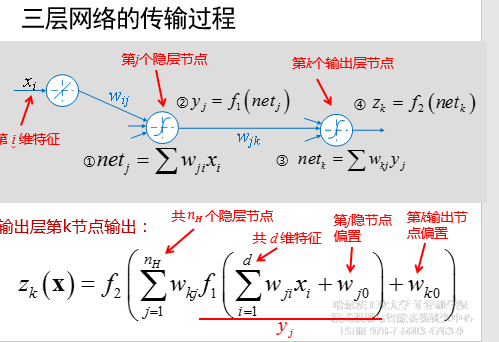


输入层,隐含层和输出层:

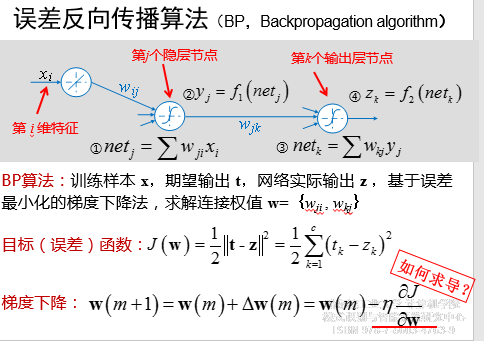


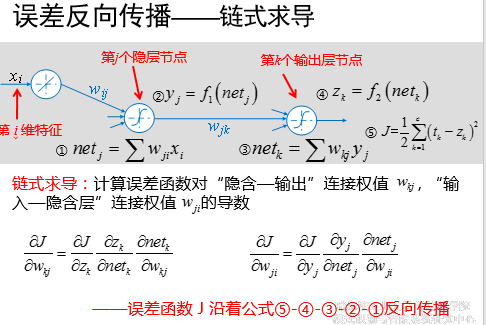


网络传输过程:上一个层的输出作为下一个层的输入



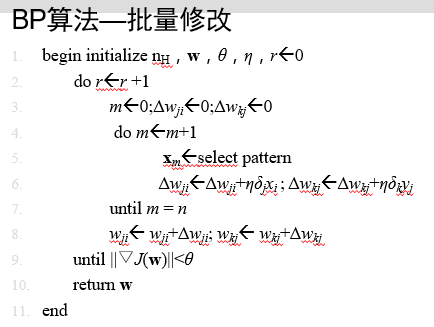
误差反向传播算法:





注意:BP算法用于适配期望的输出和实际的输出之间做出调整.

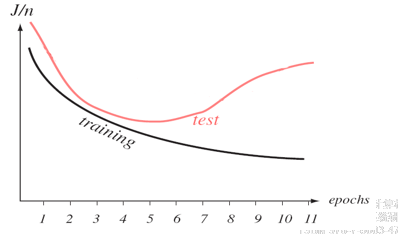
是神经网络中调整学习行为的重要方式.

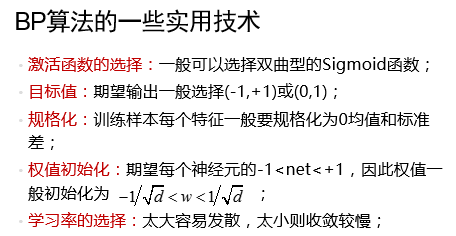


对感知器网络进行训练的过程就是BP算法运行的过程.

BP算法存在的问题:

1.收敛速度慢 2.收敛于局部最优解而不是全局最优解 3.当隐含层的节点较多时,能识别训练样本,难识别测试样本

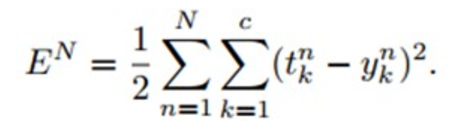




* CNN的组成

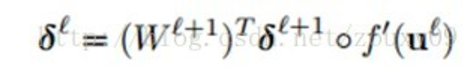
主要组成: 卷积层和下采样层, 通常两个层并不像传统的感知网络具有全连接关系.

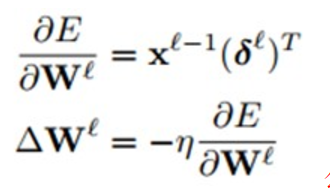
平方代价函数:

, t的形式是一个n维二值向量, 属于某个类时将其置1.

对于sigmoid(S形状的)函数,二值为{0,1},对于双曲的,二值为{-1,1}

下面是bias的变化公式:

 圆圈o表示按元素相乘

权值更新公式  
, 其中E是最小平方差准则函数, 对不同的Wij, 学习率都不同.

CNN训练的过程也可以看成BP算法的运行过程, 先正向计算,然后反向传播, 直到最后误差准则函数收敛.(收敛不是趋于0)

CNN的目标就是求出最终W矩阵和偏置向量b.

CNN的训练主要是在卷积层和子采样层的交互上，其主要的计算瓶颈是：

1）前向传播过程：下采样每个卷积层的maps；

2）反向传播过程：上采样高层子采样层的灵敏度map，以匹配底层的卷积层输出maps的大小；

3）sigmoid的运用和求导。

* CNN的典型特征

典型的CNN中，开始几层都是卷积和下采样的交替，然后在最后一些层（靠近输出层的），都是全连接的一维网络。

* dropout

Dropout是指在模型训练时随机让网络某些隐含层节点的权重不工作，不工作的那些节点可以暂时认为不是网络结构的一部分，但是它的权重得保留下来（只是暂时不更新而已），因为下次样本输入时它可能又得工作了。训练神经网络模型时，如果训练样本较少，为了防止模型过拟合，Dropout可以作为一种trick供选择。

我们知道如果要训练一个大型的网络，训练数据很少的话，那么很容易引起过拟合(也就是在测试集上的精度很低)，可能我们会想到用L2正则化、或者减小网络规模。然而深度学习领域大神Hinton，在2012年文献：《Improving neural networks by preventing co-adaptation of feature detectors》提出了，在每次训练的时候，让一半的特征检测器停过工作，这样可以提高网络的泛化能力，Hinton又把它称之为dropout。

* 过拟合

对于训练的数据都具有很好的分辨率,但是对于测试数据很差.

* cnn的训练结果

如果有n层网络, 就是训练n层网络的权值Wn, 也就是一个矩阵和偏置.

可以分多次训练,也可以一次训练完毕.

注意:flatten函数

>>> sess.run(tf.contrib.layers.flatten([ [[1.0,2.0], [3.0,4.0]] , [[5.0,6.0],[7.0,8.0]] ] ))

array([[ 1., 2., 3., 4.],

[ 5., 6., 7., 8.]], dtype=float32)

就是将每一行进行flatten之后再添加到新矩阵中.得到的仍然是一个可以使用的矩阵.

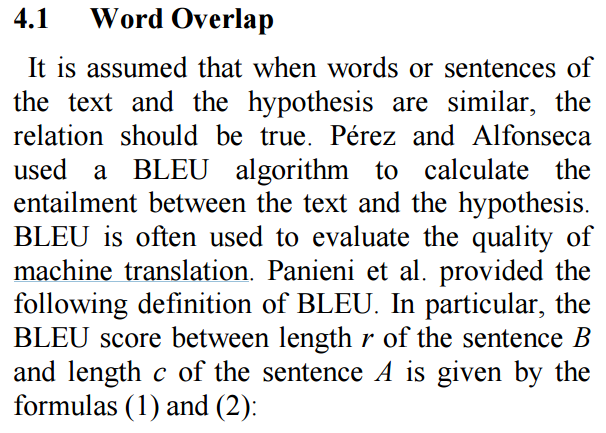
* 激活函数relu

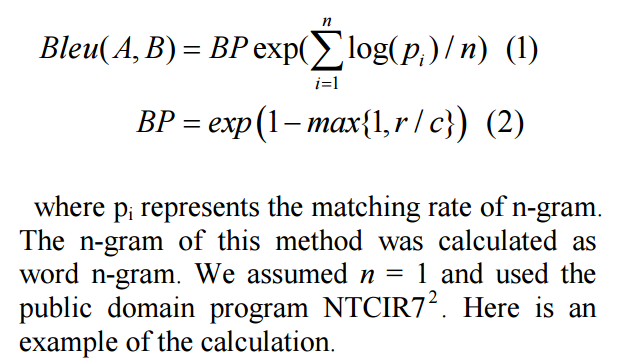
 ReLU（Rectified Linear unit）激活函数最近变成了神经网络中隐藏层的默认激活函数。这个简单的函数包含了返回max(0,x).

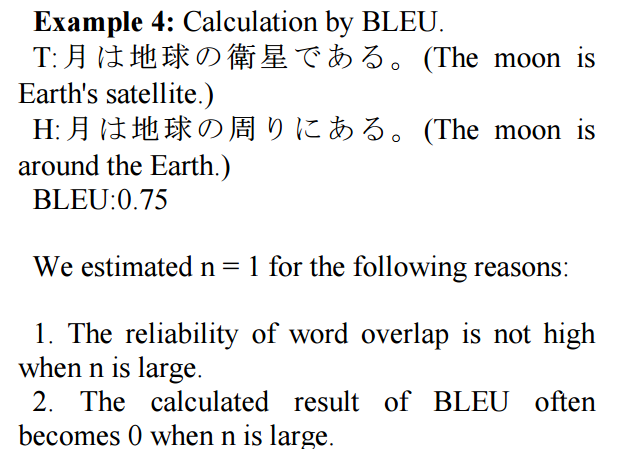
* 将图片转化成矩阵

## 附录:CNN实现的DBQA

###### Word Overlap





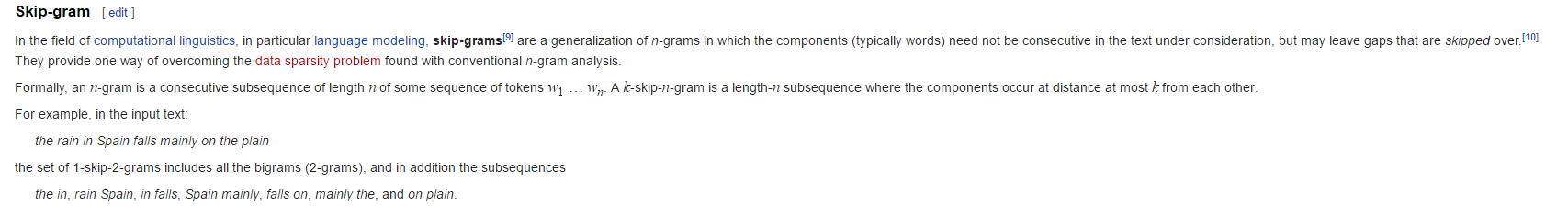


###### n-gram

An *n*-gram of size 1 is referred to as a "unigram"; size 2 is a "[bigram](https://en.wikipedia.org/wiki/Bigram)" (or, less commonly, a "digram"); size 3 is a "[trigram](https://en.wikipedia.org/wiki/Trigram)". Larger sizes are sometimes referred to by the value of *n* in modern language, e.g., "four-gram", "five-gram", and so on.

###### skip-gram

是n-gram的泛化



###### RELU

假设有一个已经翻译好的语句集合,现在评价一个新的翻译和这个翻译的好坏.使用的就是RELU评价方法.

参考: <http://www.aclweb.org/anthology/P02-1040.pdf>

使用: <http://stackoverflow.com/questions/32395880/calculate-bleu-score-in-python>

|  |  |
| --- | --- |
|  | The BLEU score consists of two parts, modified precision and brevity penalty. Details can be seen in the [paper](http://www.aclweb.org/anthology/P02-1040.pdf). You can use the [nltk.align.bleu\_score](http://www.nltk.org/_modules/nltk/align/bleu_score.html) module inside the NLTK. One code example can be seen as below:  import nltk  hypothesis = ['It', 'is', 'a', 'cat', 'at', 'room']  reference = ['It', 'is', 'a', 'cat', 'inside', 'the', 'room']  #there may be several references  BLEUscore = nltk.translate.bleu\_score.sentence\_bleu([reference], hypothesis)  print BLEUscore  Note that the default BLEU score uses n=4 which includes unigrams to 4 grams. If your sentence is smaller than 4, you need to reset the N value, otherwise ZeroDivisionError: Fraction(0, 0) error will be returned. So, you should reset the weight like this:  import nltk  hypothesis = ["open", "the", "file"]  reference = ["open", "file"]  #the maximum is bigram, so assign the weight into 2 half.  BLEUscore = nltk.translate.bleu\_score.sentence\_bleu([reference], hypothesis, weights = (0.5, 0.5))  print BLEUscore |

## 附录:Reccent NN

## 附录:Recursive NN

## 附录:LSTM

## 附录:基于深度学习的语句匹配

参考文档: <https://cs224d.stanford.edu/reports/SanbornAdrian.pdf>

根据这篇文档的说明, 深度学习可以从有限的训练集中找到有效的句子相似度的模型.

词向量算法:GloVe

由于初始的文本较少,所以使用了预先训练好的GloVe向量.(50~100维)

* recurrent NN
* recursive NN

更加适合用于自然语言处理

* tensorflow用法

基本概念 <https://www.tensorflow.org/versions/r0.10/get_started/basic_usage>

基本的概念包括图,图中的节点(就是一个定义的运算过程)

真正执行运算的是tf.Session().run(what).

如果一个operation依赖于其他operation,这是就会形成所谓的图结构.

operations类型:

tensorflow.constant 常实数或者向量

Variable 在运行之前,必须调用v.initializer.run,

如果为了初始化所有的数据,获取下面的初始化首先运行

变量能被复制,变量可以使用name来初始化

initizlize\_all\_variables 一个初始化所有变量操作

placeholer(tf.float32) 一个类型占位符,在run的时候使用feed\_dict来输入实际参数

operation之间的操作:(同样返回operation类型)

tf.add tf.add(o1,o2) 结果返回,o1,o2不变

tf.matmul 矩阵乘法

tf.assign 赋值

operations可以单独调用run方法而不需要Session的参与.

feed的例子:

input1 = tf.placeholder(tf.float32)  
input2 = tf.placeholder(tf.float32)  
output = tf.mul(input1, input2)  
  
with tf.Session() as sess:  
  print(sess.run([output], feed\_dict={input1:[7.], input2:[2.]}))  
  
# output:  
# [array([ 14.], dtype=float32)]

* tensorflow concepts

Rank(秩)

不同于矩阵的Rank,Tensor的Rank是指tensor的维数.譬如:

t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

rank=2, rank用来表示tensor的数据类型.(因为张量本身是量的扩张)

当shape=[2, 2, 2] 表示这是一个2\*2\*2的矩阵.

reshape函数就是将一个shape转换成另一个shape. reshape的shape参数中可以有至多一个-1,表示自己计算.计算方法就是作除法. [-1]表示转换成一个向量

[1,-1]表示转换成向量列表,但是列表中仅仅有一个向量.

实现过程: **reshape（t, shape） => reshape(t, [-1]) => reshape(t, shape)**

Shape

和Rank,dimension number一样,下面的转换关系表

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | [] | 0-D | A 0-D tensor. A scalar. |
| 1 | [D0] | 1-D | A 1-D tensor with shape [5]. |
| 2 | [D0, D1] | 2-D | A 2-D tensor with shape [3, 4]. |
| 3 | [D0, D1, D2] | 3-D | A 3-D tensor with shape [1, 4, 3]. |
| n | [D0, D1, ... Dn-1] | n-D | A tensor with shape [D0, D1, ... Dn-1]. |

Shape的类型:

Python lists/tuples or [TensorShape class](https://www.tensorflow.org/versions/r0.10/api_docs/python/framework/defining_new_operations#TensorShape).

Data Types

| Data type | Python type | Description |
| --- | --- | --- |
| DT\_FLOAT | tf.float32 | 32 bits floating point. |
| DT\_DOUBLE | tf.float64 | 64 bits floating point. |
| DT\_INT8 | tf.int8 | 8 bits signed integer. |
| DT\_INT16 | tf.int16 | 16 bits signed integer. |
| DT\_INT32 | tf.int32 | 32 bits signed integer. |
| DT\_INT64 | tf.int64 | 64 bits signed integer. |
| DT\_UINT8 | tf.uint8 | 8 bits unsigned integer. |
| DT\_STRING | tf.string | Variable length byte arrays. Each element of a Tensor is a byte array. |
| DT\_BOOL | tf.bool | Boolean. |
| DT\_COMPLEX64 | tf.complex64 | Complex number made of two 32 bits floating points: real and imaginary parts. |
| DT\_COMPLEX128 | tf.complex128 | Complex number made of two 64 bits floating points: real and imaginary parts. |
| DT\_QINT8 | tf.qint8 | 8 bits signed integer used in quantized Ops. |
| DT\_QINT32 | tf.qint32 | 32 bits signed integer used in quantized Ops. |
| DT\_QUINT8 | tf.quint8 | 8 bits unsigned integer used in quantized Ops. |

## 附录: 一个使用tensorflow训练的例子

假设有方程(r,cx,cy,a,b):

1: (x-cx)^2 <= r^2 时 (x-cx)^2 + (y-cx)^2 = r^2

2: 其他情况 y=a\*x + b

训练的目标参数就是 方程中的5个参数尽量接近原始的参数.

或者说, 训练的参数的输出与原输出相差较小.

一开始给定一些与原始方程有偏差的数据 dataGen{x,y},

定义一个函数是完全定义函数,当且仅当这个函数在给定输入X上均有取值.

方程的函数取值:

def eq(r,cx,cy,a,b):

def evalEq(x):

z=r\*\*2 – (x-cx)\*\*2

if z>=0:

return random.choice([-1,1])\*math.sqrt(z) + cy

else:

return a\*x+b

return evalEq

随机偏差设置设y的偏差在+/-d内随机变化(d=0.5),获取偏差数据的方法:

def getDiff(y,d=0.5):

return y+random.uniform(-d,d)

trainR,trainCX, trainCY, trainA,trainB

模型的输出函数inference(x):#x的输入可能是向量

z=tf.pow(trainR,2) –tf.pow( tf.subtract(x,trainCX),2)

tf.map\_fn(…

度量函数loss(x,y): #x,y are in the original data

predictedY=inference(x)

return tf.reduce\_sum(tf.square\_difference(y,predictedY))

tensorflow支持同时迭代:

As on today, I see that map\_fn is enhanced to take two tensors as the documentation says that - "elems: A tensor or (possibly nested) sequence of tensors, each of which will be unpacked along their first dimension. The nested sequence of the resulting slices will be applied to fn." The example (though given in numpy form) also shows that it can take two tensors. I'm copying it here.

elems = (np.array([1, 2, 3]), np.array([-1, 1, -1]))

alternate = map\_fn(lambda x: x[0] \* x[1], elems, dtype=tf.int64)

# alternate == [-1, 2, -3]

* 结论

不能使用随机的圆方程

learningRate=0.00001

steps=5500的时候loss

loss: 0.107045

loss: 0.0907904

loss: 0.0773396

loss: 0.0661303

loss: 0.0567337

loss: 0.0488141

## 附录:第一节mooc

问题分类: 事实类 定义 YES-NO 观点 比较(Nokia和IPhone的区别)

* 问答系统分类

基于知识的问答:KQA 知识:以抽象的图结构存在(A->B A有属性B)

结构化知识: 实体 和 实体之间的联系

知识的结构:三元组 (主语, 谓词, 宾语)如 <清华大学, 校长, 邱勇>

知识与上下文有关.

* 基于文档的问答:DQA 非结构化的知识

不仅能从已经整理好的知识问答,也能从自然语言本身回答.(或者多媒体)

计算问题与文档中的句子的相关性,相关性足够的时候返回这个句子.

本次任务: 计算相似,然后排序(不仅仅是最高的相似度)

问题和答案相关性:

1. 简单统计
2. 通过Word Embeding(词向量)
3. 基于深度学习

* 基于社交的问答:FAQ

一些问答网站就是典型的基于社区网站的问答.

关键: 计算两个问题之间的相似度

计算相似度: 语义相似, 语法相似 (自然语言的核心问题)

模型

1:词对齐模型 (将两个句子之间的成分对应起来, 需要大量数据才能建立起来)

如 Who is the founder of Mi;

Who founded Mi.

2.

3.利用机器翻译,如果翻译结果相同, 源语言可能是相同的

* 知识的实体识别和关系识别

实体名称可能不同(清华,清华大学)

关系识别方法:

1. 基于问题模板的方法(基于完全匹配)

谁是#的创始人 创始人

#的创办人是谁 创始人

…

缺点:覆盖度不强

1. 基于关系关键字的方法

创办人|||创办人

创始人|||创办人

创办者|||缔造者

* 将问题转化为多维的实数

保证相关问题在向量空间上的距离是较小的.

## 附录:第二节MOOC

地址:

## 附录:第三节MOOC

## 附录:第四节MOOC

## 构建测试

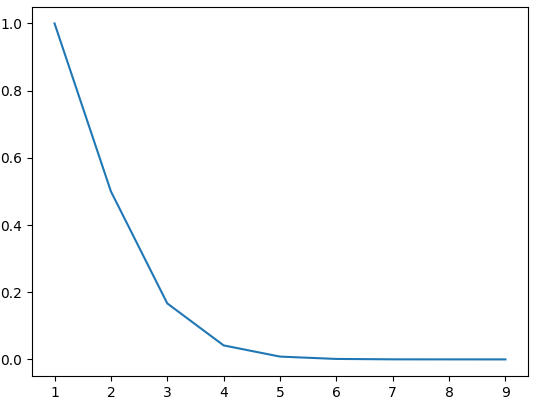
* 测试数据
  + 来源
    - BoP本身的训练集和开发集数据
      * 格式: [0|1] [问题] 候选句
  + 数据量
    - 50个文档
    - 50个问题
  + 预处理
    - 源文档的格式:以段为单位,但是段是句子的集合,所以一开始处理的时候可以先将换行符去掉,或者在处理的过程中忽略.
    - 依照按行的思想,每一行被视作独立的文档
    - 将文档拆分为以句为单位的句子, 句子分割以 逗号和句号,问号,感叹号[等]为分界符,保留分界符,所有句子以行划分
      * 以re来查找(避免出现空串)

r=re.compile(r'[^!,.;?:！，。；？：]\*[!,.;?:！，。；？：]+|[^!,.;?:！，。；？：]+[!,.;?:！，。；？：]\*')

r.findall(r'hello,perp?wher,are you;all')

['hello,', 'perp?', 'wher,', 'are you;', 'all']

* + 问题生成
    - 对于给定的一个句子
    - 通过分词和词性标注得到分词结果
    - 构造等义词集合
      * 手工构造一个小的等义词集合
      * 等义词集合(包括停用,实体名词,动词)
    - 每一行都是一些同义的词语(这就称作等义词词典)
      * 处理方式:最适合用并查集实现
    - 通过下列方式变换成另一一个句子
      * 所有变换随机进行,降比概率发生: 1 1/2 1/3 1/4 …



* + - * 实体如果含有同义词, 变换成同义词或者什么(即问句)
      * 停用词变换成另一个停用词
  + 数据量
    - 数据量使测试结果能够达到稳定状态
  + 格式
    - 训练集(答案已知)

[性质]={是,否} [问题] [句子]

* + - 测试集

[问题] [句子]

两种测试集都以

* 评估方式
  + 正确率
    - 即一个问题Qi的答案个数为ni,比较其前ni个预测答案集合pi与已知答案集合ui的差异,具体来说, Qi的正确率= pi中相同元素个数/ni=即在pi也在ui中的元素个数/ni

可以对每个问题求一个正确率

然后对总问题求一个正确率.

要求的输入形式:

Qi

Qi的所有回答

构造出

Qi所有回答的评分

构造出

Qi的正确答案集合 🡪Qi的所有回答中示性为1的集合

Qi的预测答案集合 🡪 Qi所有回答评分的排序的前ni个

计算出

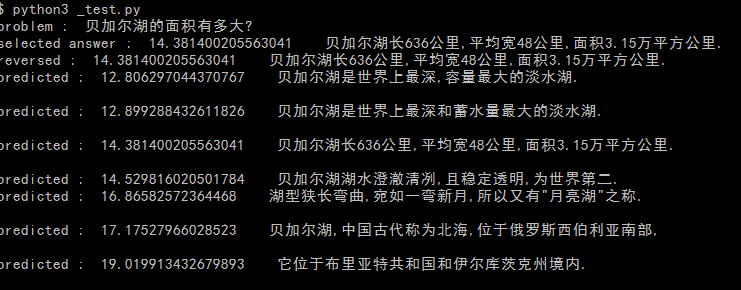
Qi的正确率

更新总的正确率

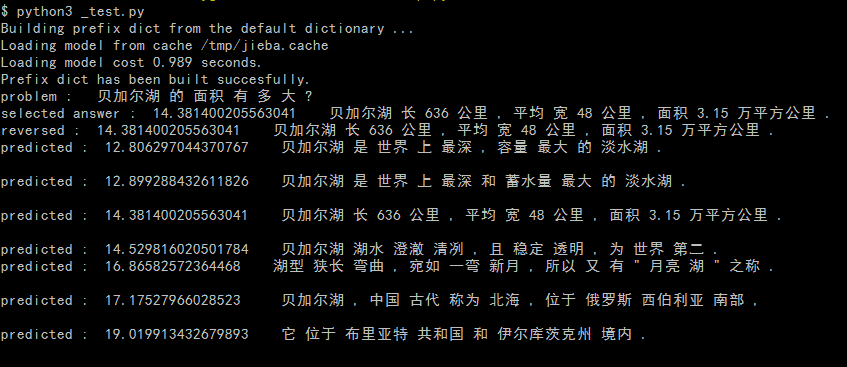


## 测试结果

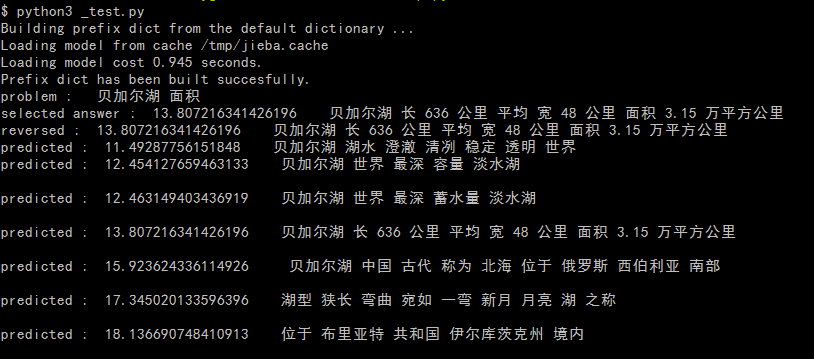
* 不分词,不去停用词



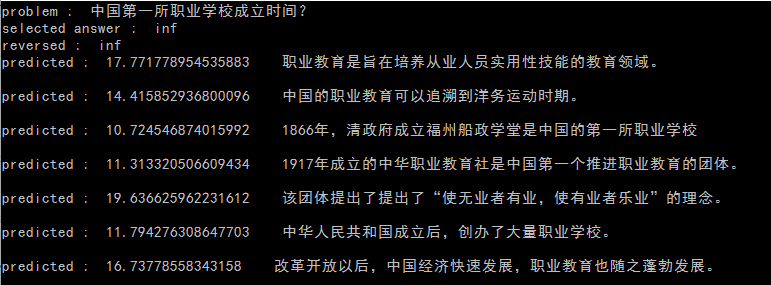
* 分词,不去停用词



* 分词,去停用词



* BOP上的案例



desired predicted

0.2343556 5 6

0.5634232 2 4

1.2384834 1 1

0.2324467 6 2

0.1283447 7 7

0.3434554 4 3

0.4754545 3 5

## 测试总结

微软此次给定的数据集主要是语义分析十分重要.

比如对于问题: problem: 关西大学的简称是什么？ predicted: {'关西大学'} suggested: {'略称为“关大”。'}

预测是由predicted得到的,很显然, 对于特定的文档集合,预先训练好的vector并不一定有效.

简称和略称实际上应当由训练的模型去发现其中的相似语义.