

总论

2022年11月30日 21:18

基本概念：

语料 (Corpus)：原始文本的集合，用来训练模型的材料

向量 (Vector)：一段文本或一个词在计算机中的表达方式

模型 (Model)：一个抽象的术语，定义了两个向量空间的变换，是需要训练的对象

NLP：Natural Language Processing自然语言处理、AI重要分支之一

主要范畴：

文本朗读 (Text to speech)、语音合成 (Speech synthesis)、语音识别 (Speech recognition)、中文自动分词 (Chinese word segmentation)、词性标注 (Part-of-speech tagging)、句法分析 (Parsing)、自然语言生成 (Natural language generation)、此外还有文本分类、信息抽取、文字校对等

研究难点：

单词的边界界定 (分词)、词义的消歧 (消除歧义)、不规范的输入、句法的模糊性、语言行为与计划 (每种语言的语法很不同)

统计语言模型

N-Gram统计模型

马尔可夫模型：无后效性

隐马尔可夫模型：隐表示隐含的参数，马氏链的状态不能直接观察到

传统NLP和深度学习NLP的区别：主要在模型选择方面

传统NLP：使用各种统计语言模型

深度学习NLP：使用神经网络，如LSTM、GRU等模型

NLTK库：Natural Language ToolKit自然语言工具集，Python上著名的自然语言处理库

语料：语言材料，语料库就是可供训练的语言材料，文本、语音等都是语料

词性标注：给每个词的词性分类 (打标签)，如形容词、动词等

常见中文词性编码

词性编码	词性名称	注 解
<u>Ag</u>	形语素	形容词性语素。形容词代码为 a，语素代码 g 前面置以A。
<u>a</u>	形容词	取英语形容词 adjective 的第1个字母。
<u>ad</u>	副形词	直接作状语的形容词。形容词代码 a 和副词代码 d 并在一起。
<u>an</u>	名形词	具有名词功能的形容词。形容词代码 a 和名词代码 n 并在一起。
<u>b</u>	区别词	取汉字“别”的声母。
<u>c</u>	连词	取英语连词 conjunction 的第1个字母。
<u>dg</u>	副语素	副词性语素。副词代码为 d，语素代码 g 前面置以D。
<u>d</u>	副词	取 adverb 的第2个字母，因其第1个字母已用于形容词。

词性标注的分类

基于规则的词性标注

基于隐马尔可夫模型的HMM词性标注 (HMM即隐马尔可夫模型的缩写)

基于转移的词性标注

基于转移与隐马尔可夫模型相结合的词性标注

词性标注的工具

NLTK：主要处理英文语句

Jieba：主要是中文分词，也可以进行词性标注等其他功能

jieba是github上的开源项目，中文分词最好的库

三种分词模式（支持繁体，支持自定义词典（主要是新词））

精确模式：试图将句子最精确地切开，适合文本分析

全模式：把句子中所有可以成词的词语都扫描出来，速度快但不能解决歧义

搜索引擎模式：再精确模式的基础上，对长词再次进行切分，提高召回率，适合搜索引擎分词

Gensim：开源的第三方Python包，用于自然语言处理中从原始的非结构化的文本中，无监督地学习到文本隐层的主题向量表达，支持TF-IDF、LSA、LDA和Word2Vec等多种模型算法

`gensim.utils.simple_preprocess`：分词，且自动去掉标点符号，但只能对英文生效，中文无法分词

NLTK、Sklearn和Gensim之间的关系：

NLTK专门收集和分类非结构化文本，主要用于一般的NLP任务

Sklearn是一种分析工具而不是收集工具，主要用于机器学习（分类、聚类等）问题

Gensim主要用于NLP中Word2Doc领域，即它比较适合Word Embedding，主要用于主题建模和文档相似性研究

三者都有分词等功能，但其每个包的侧重点是不一样的

分词（主要针对中文，英文直接按空格和符号分割就完事了）

分词难点

分词标准：必须动态选择，如“花草”，是不分词还是分成花和草

切分歧义：分词造成的语义分歧

分词细粒度不同造成

本身存在歧义：下雨天留客天天留人不留

交集型的歧义

新词：目前新出现的词语，如鸡你太美

分词的算法

基于词典的分词算法

正向最大匹配

逆向最大匹配

双向匹配分词

全切分路径选择

基于统计的分词算法

HMM隐马尔可夫模型

CRF条件随机场

深度学习

TF-IDF：Term frequency-Inverse document frequency，词频和逆文本频率指数

一种用于信息检索和数据挖掘的常用加权技术

主要思想：

如果某个词在一篇文章（样本）中出现概率很高，并且在其他文章（样本）中很少出现，则认为这个词或者短语具有很好的类别区分能力，是我们需要的关键词

计算方法：

TF（词频）：某文件中该词的出现频率（不是频数）， $TF = \text{某文件中该词的个数} / \text{所有词的个数}$

IDF：表示词语的重要性度量， $IDF = \lg(\text{总文件数} / \text{包含这个词的文件数})$

$TF-IDF = TF * IDF$ ，可理解为重要程度

Tfidf的重要性：分类机器学习算法进行文章分类中前期数据处理方式

文本处理方法：文本是非结构化的数据，不可计算

文本处理方法是现实问题转化为可计算的数学问题，具有决定性意义，主要有以下三种方法

一、OneHot：独热编码：有多少个类别就有多少个状态位来表示他

特征数目很多时，维度将变得非常大

强稀疏性。当词向量很长时，分量绝大多数都是0

不能刻画词与词之间的相似性

二、整数编码：每个单词用一个整数表示，如下；



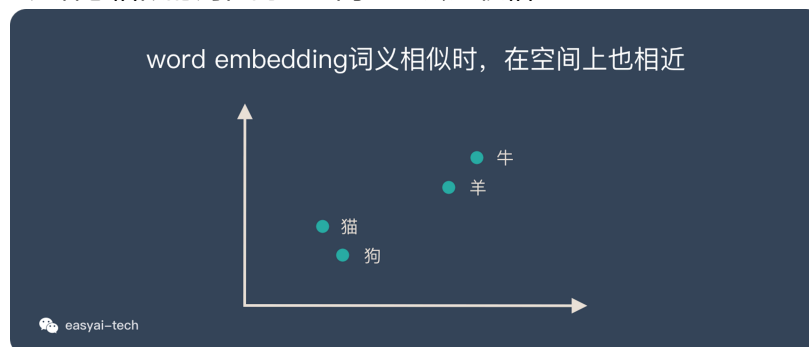
优点是表示简单，不会重复；缺点是无法表达词语之间的关系，且不太好对模型进行解释

三、词嵌入（word embedding）

词嵌入主流有两种方法，一种是word2vec，一种是GloVe

词嵌入不是某种具体的算法。它的优点比独热和整数编码更多：

- 1、可以将文本通过低维向量表达，不像one-hot那么长
- 2、语意相似的词在向量空间上也会比较相近



- 3、通用性强，可以用在不同的任务上

目前词嵌入已不是最好的方法，所以Word2Vec和GloVe也基本不会再用到

Word2Vec：用神经网络（CBOW和Skip-gram）把词转化成向量的模型，NLP前期处理最重要的模型之一。但自2018年开始，该模型已不再是最先进、也不是主流模型了，已被bert和GPT取代

Word2Vec采用Distributed Representation作为词的编码方式，该方法将不同的词转化成向量之后，若词本身之间关联性较强，则转化成的向量的关联性也会越强，即相关系数越接近于1（皮尔逊相关系数）

Word2Vec是轻量级的神经网络，主要包含CBOW和Skip-gram模型

CBOW（连续词袋）：根据句子前后，预测句子中缺少的某个词

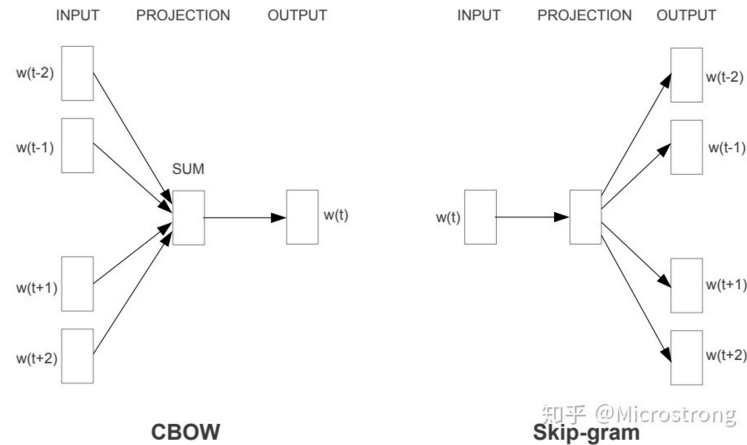
中间层是映射层而不是隐藏层

结合双向上下文，上下文词序无关

输入低维稠密，映射层求和

Skip-gram：跟CBOW完全相反，根据某个词，预测其上下文（即整句）

没有隐层，映射层也可以省略



以上两种模型，实际上生成的不是词，而是直接生成向量，所以叫做用神经网络生成词向量

优化方法：为了提高速度，Word2Vec通常采用两种加速方法

- 1、负采样：Negative Sample
- 2、分层Softmax：Hierarchical Softmax

优缺点：

- 1、考虑上下文（但没有考虑全局）
- 2、生成的词向量维度低，速度更快
- 3、通用性强，可以用在各种NLP任务中
- 4、词和向量是一一对应的关系，多义词问题无法解决
- 5、静态方式，虽然通用性强，但无法针对特定任务做动态优化

GloVe：对Word2Vec方法的扩展，它将全局统计和Word2Vec的基于上下文的学习结合了起来，并且致力于解决词的多义问题

stopwords：停用词，比如介词、标点等不那么重要的信息可以去掉

例如：砍掉the、is、at等来减少维度

聊天机器人的分类

按领域分类

固定领域：专门在某个领域使用，如电商技术支持

开放领域：在各方面都可以使用，比较难实现，代表：微软小冰

按模式分类

规则模式：最简单的模式，只要问题中包含某个词，就回答预先设定的句子

检索模式：一般用在固定领域当中

构建FAQ数据库，然后根据输入问题，搜索数据库的答案然后返回

需要的数据库比较大，回答比较自然

检索匹配的核心是分类，贝叶斯分类是最常见的匹配方法

检索类机器人需要进行分词、语义分析、注意力机制等技术，检索知识库里的相关答案，然后选择最佳的一个答案返回给用户

生成模式：一般用在开放领域当中，实现更加困难

不基于预定义的响应，完全从0开始生成新的响应，通常基于机器翻译技术（生成模型，不是翻译语言的翻译）

通常使用encode-decode编解码框架

按功能分类

问答型聊天机器人

任务型聊天机器人

闲聊型聊天机器人

Chatterbot：一个基于Python、Pytorch机器学习的一个开源检索类机器人项目

直接用pip或conda下载可能会报错

chatterbot的聊天逻辑和输入输出以及存储，都是由各种adapter（适配器）来限定的

常用方法：

chatterbot.trainers.ListTrainer(storage, **kwargs)训练类通过列表数据进行训练

chatterbot.trainers.ChatterBotCorpusTrainer(storage, **kwargs)训练类通过语料库进行训练

chatterbot.trainers.TwitterTrainer(storage, **kwargs)训练类使用Twitter API进行训练

The main class `ChatBot` is a connecting point between each of ChatterBot's `adapters`. In this class, an input statement is processed and stored by the `logic adapter` and `storage adapter`. A response to the input is then generated and returned.

```
class chatterbot.ChatBot(name, **kwargs) \[source\]
```

A conversational dialog chat bot.

Parameters:

- `name (str)` – A name is the only required parameter for the ChatBot class.
- `storage_adapter (str)` – The dot-notated import path to a storage adapter class.
Defaults to `"chatterbot.storage.SQLStorageAdapter"`.
- `logic_adapters (list)` – A list of dot-notated import paths to each logic adapter the bot uses. Defaults to `["chatterbot.logic.BestMatch"]`.
- `logger (logging.Logger)` – A `Logger` object.

语言模型：一串词序列的概率分布，就是生成的一串词中，组成合适的句子的概率高不高

基于统计学习：

最常用的就是N-gram语言模型，N可以是不同的数字

以2-gram为例，就是从句首开始，把相邻的两个词拿出来，求概率分布（条件分布）

基于机器学习和深度学习：RNN、LSTM、GRU等

Jupyter Notebook

Jupyter本质为基于网页的用于交互计算的应用程序，类似于iPython以及Python的交互模式（控制台），其可被应用于全过程计算：开发、文档编写、运行代码和展示结果

简而言之，Jupyter Notebook是以网页形式打开，可以在网页页面中直接编写代码和运行代码，代码的运行结果也会直接在代码块下显示的一个程序。如果在编程过程中需要编写说明文档，可以在同一个页面中直

接编写，便于作及时的说明和解释

使用方法：可使用pip install jupyter，更常用的方法为安装Anaconda，之后会自动安装该程序，点击运行后，会自动跳转到Jupyter的网页

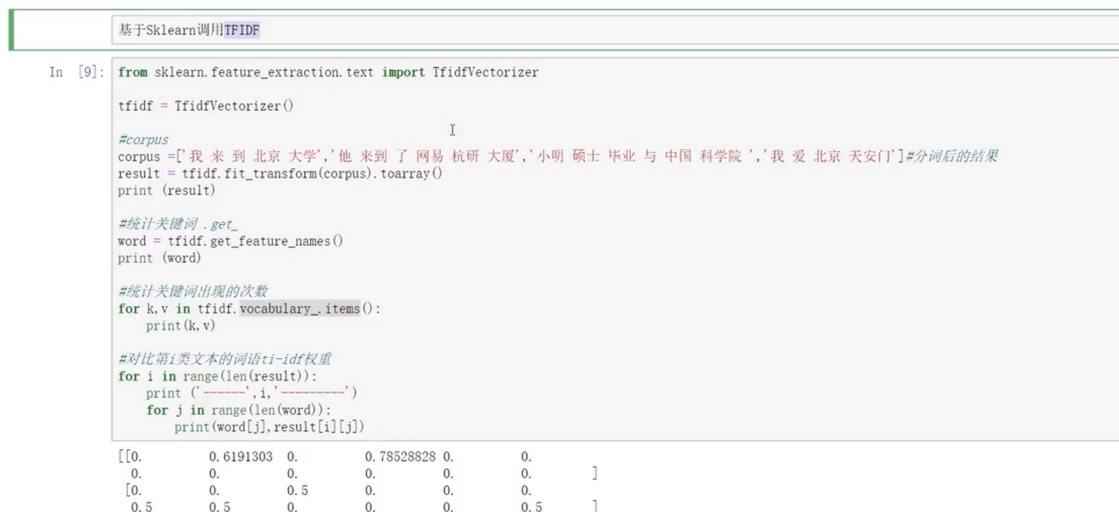
注意木星英文名是Jupiter，有点小区别

使用方法：

首先需要加载或打开文件：.ipynb扩展名

每一行都是独立的区域，可选择其为代码区或文本区域等

每一行都可以敲一段代码，然后运行



The screenshot shows a Jupyter Notebook interface with a title bar '基于Sklearn调用TFIDF'. The code cell contains the following Python code:

```
In [9]: from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()

#corpus
corpus = ['我 来 到 北 京 大 学', '他 来 到 了 网 易 杭 研 大 厦', '小 明 硕 士 毕 业 与 中 国 科 学 院 ', '我 爱 北 京 天 安 门'] #分词后的结果
result = tfidf.fit_transform(corpus).toarray()
print (result)

#统计关键词 . get_
word = tfidf.get_feature_names()
print (word)

#统计关键词出现的次数
for k,v in tfidf.vocabulary_.items():
    print(k,v)

#对比第i类文本的词语ti-idf权重
for i in range(len(result)):
    print ('-----',i,'-----')
    for j in range(len(word)):
        print(word[j],result[i][j])
```

The output of the code is displayed below the code cell:

```
[[0.         0.6191303 0.         0.78528828 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.5         0.         0.         0.
  0.5         0.5         0.         0.         0.         0.5         1.
]]
```

如上图所示，说明性文本、代码块、运行结果井然有序，直观且方便

代码

2023年1月9日

13:53

基于规则

2022年12月1日 21:36

```
'''
利用nltk的分词模块
构建简单的基于规则的回复
'''

from nltk import word_tokenize      # 分词模块
import random

# define greets and random_greets
greet = ['hi', 'hello', 'halo', 'hey']
random_greet = random.choice(greet)

# keywords about holiday and response
question = ['break', 'holiday', 'vacation', 'weekend']
responses = ['It was nice', 'I went to Paris', 'I just stay at home']
random_response = random.choice(responses)

while True:
    user_inputs = input('>>>')
    cleaned_inputs = word_tokenize(user_inputs)      # 对输入进行分词，注意这个函数只能处理英文
    if not set(cleaned_inputs).isdisjoint(greet):    # 集合函数isdisjoint判断两个集合是否有相同元素，不包含返回True
        print(random_greet)                        # 即：若输入的句子中，含有关键词，则触发相关回复
    elif not set(cleaned_inputs).isdisjoint(question):
        print(random_response)
    elif user_inputs == 'bye':
        break
    else:
        print('I do not understand what you say')
```


分词与词频统计

2022年12月1日 21:37

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer() # 实例化转换器
# 英文分词的依据就是空格，这里手动打上了空格，也能利用英文分词工具正常分词
CORPUS = ['我 来 到 北 京 大 学', '他 来 到 了 网 易 杭 研 大 厦', '小 明 硕 士 毕 业 于 中 国 科 学 院', '我 爱 北 京 天 安 门']
result = tfidf.fit_transform(CORPUS).toarray()
print(result)

# 统计关键词
word = tfidf.get_feature_names()
print(word)
# 统计关键词出现的次数
for k,v in tfidf.vocabulary_.items():
    print(k,v)
# 对比第i类文本的词语tf-idf权重
for i in range(len(result)):
    print('-----',i,'-----')
    for j in range(len(word)):
        print(word[j],result[i][j])
```

使用gensim训练Word2Vec

2023年1月9日 22:16

```
'''
使用ginsim进行Word2Vec
'''

import gensim # 自然语言处理的库
with open('data/Engtxt.txt','r',encoding='utf-8') as f:
    lines = f.readlines()

def inputs():
    for line in lines:
        yield gensim.utils.simple_preprocess(line) # 使用gensim进行分词，以句子为单位得到列表
document = list(inputs()) # 现在的inputs不只是函数，也是一个迭代器了

# 训练模型
model = gensim.models.Word2Vec(sentences=document,vector_size=150,
                               window=10,min_count=2,workers=10)
model.train(corpus_iterable=document,total_examples=len(document),epochs=10)
```

更多函数可从网上或源码中找到，关键词参数不懂的，源码中有说明，如：

 window : int, optional

 Maximum distance between the current and predicted word within a sentence.

ChatGPT解析

2023年1月8日 21:11

GPT: Generative Pre-training Transformer生成式预训练transformer模型

ChatGPT是OpenAI公司在GPT3.0基础上开发的聊天机器人框架，有1750亿个参数

Instruct GPT是ChatGPT的框架，它有两个特点：

用人类更喜欢的数据来做训练并生成答案

引入强化学习提升性能天花板，不仅告诉模型回答得好不好还告诉怎样才能变好，突破监督学习的桎梏

GPT三大核心技术

1、预训练

预训练是指在大规模语料库上对模型进行训练，使其能够自动学习语言的规律和规则。在预训练过程中，ChatGPT使用了海量的无标签文本数据，比如维基百科和新闻文章等。通过这些数据的训练，ChatGPT可以学习到自然语言的语法、句法和语义等信息，从而能够生成自然流畅的语言表达。

2、Transformer

Transformer网络是一种基于自注意力机制的神经网络，是seq2seq的一种，能够有效地处理长文本序列，并且能够捕捉到序列中的上下文信息。相较于传统的循环神经网络（RNN）和卷积神经网络（CNN），Transformer网络具有更好的并行性和更高的计算效率，能够处理更长的序列，使得ChatGPT能够生成更长、更复杂的文本内容

3、自回归

自回归模型是ChatGPT的核心生成模型。自回归模型是指在生成文本时，模型会根据前面已经生成的文本内容来预测下一个单词或符号。ChatGPT使用了基于循环神经网络的自回归模型，每次生成一个单词或符号时，模型会根据上下文信息和历史生成结果进行预测。通过不断迭代生成，ChatGPT可以生成连贯自然的文本内容。

ChatGPT训练流程：

- 1、监督学习：在互联网语料库里搜集问题，并人工编写答案，用它们来进行训练，以规范回答
- 2、强化学习：建立一个奖励模型，对于多个输出，人工进行打分，并以此训练奖励模型，以预测用户更喜欢哪个输出，从而得到更令人喜欢的回答
- 3、基于强化学习loss持续迭代生成模型，使用奖励模型作为奖励函数，以PPO的方式，微调监督学习训练出来的生成模型

而ChatGPT是由GPT3.5演变而来的，在此之前还经历了无监督学习，即以海量互联网语料，扩充GPT的知识，以达到良好的词嵌入（Word embedding）能力

ChatGPT乃至以后大规模语言模型的意义：

1、基本意义，也就是它能存储人类历史上涌现的所有知识，加快某个人类个体学习知识的速率，达到解放生产力的作用（比如不用花大量时间在规范PPT的排版上，而是节约出时间来思考真正有效的问题）。但这不是革命性的，因为现在的互联网也能存储所有的知识，只是需要手动搜索，自己理解，耗费时间。

2、长远意义，语言模型参数，量变产生质变。如果语言模型产生了逻辑和创造性，那么它就可以利用人类全部的知识、强大的算力，去发现人类没有能力探索的领域。它可能会创造出新技术，创造新发明。当它能创造知识的时候，人类就能在它的帮助下迈向下一个时代

Openai调用API

2023年5月11日 16:44

api和key：拥有openai账号后，可以生成key，在其他项目里，如自己写的程序中调用

ChatGPT。但需注意：

- 1、免费额度有限，要想继续用，就要刷爆信用卡
- 2、api版没有记忆能力，可以在问的时候讲上下文存储起来一起发送过去（网页版也是类似的解决办法）
- 3、在Python等环境中，首先需要下载openai的包，`pip install openai`

首先下载包：

- 1、`pip install openai`（Python环境）
- 2、`npm install openai`（Node.js）

验证（Authentication）：API使用API keys for authentication

所有API请求都需要在http的请求头中包含API key：

Authorization: Bearer OPENAI_API_KEY

最基本的请求（用于首次的测试）：不要使用curl方式，总是会报错：json格式的问题，这个问题在python代码中比较好解决，即直接使用json库把字典转化成json格式即可。所以直接使用requests发送post请求即可

代码示例：

```
import requests
import json

url = 'https://api.openai.com/v1/chat/completions'
headers = {'Authorization': 'Bearer sk-9YW1pfFijCTzLwRtduowT3B1bkFJqE0x8PY1sYLdNyko00qN',
           'Content-Type': 'application/json'}
data = {"model": "gpt-3.5-turbo", "messages": [{"role": "user", "content": "Say this is a test!"}], "temperature": 0.7}
data = json.dumps(data)
response = requests.post(url=url, data=data, headers=headers)
print(response.text)
```

Completions：给定提示词或句子，将返回完整的句子或添加了细节的句子
实际用处应该不是很大

创建聊天

请求结构体：

- 1、model：必填，选择所使用的model的ID，可在[Models - OpenAI API](#)查看模型账号的订阅不同，可以使用的模型也不同
- 2、messages：必填，列表，A list of messages describing the conversation so far。用列表把字典括起来。
注意是messages!!! 有一个s
 - role：必填，这个message的作者，是system、user或assistant（助理）中的一个
 - user：提交的一方
 - assistant：响应的一方，即ChatGPT本身
 - system：让ChatGPT在对话过程中设定自己的行为，但目前还没有太大的实际作用
 - content：必填，message的主体。（请求的content和返回的content即对话内容）
 - name：可选，这个message作者的名字，可以包含a-z、A-Z、0-9和underscores（下划线），最长64个字符
- 3、temperature：数字，可选，默认为1。这是一个基于温度系数的采样，选择什么样的采样温度，值在0-2之间。值越大则结果随机性越高，越低则更精确
- 4、top_p：数字，可选，默认为1。“它是一种称为核采样的温度采样的替代方法，其中模型考虑具有 top_p 概率质量的令牌的结果。因此，0.1 表示仅考虑占前 10% 概率质量的令牌。通常建议更改此参数或 temperature，但不要同时更改两者。”
- 5、n：整数，可选，默认为1。每次对话会生成多少次回应。一般不用设置
- 6、stream：布尔，可选。如果设置，将发送消息增量，就像在chatgpt中一样
- 7、stop：字符串或列表，即停用词，可选，默认为null。最高4个序列可供使用，使其停止生成
- 8、max_tokens：整数，可选，默认为inf（无限）。即一个回答中文本数量
- 9、presence_penalty（存在惩罚）：数字，可选，默认为0，介于-2到2之间。正值会根据新词是否出现在文本中进行惩罚，从而增加模型讨论新主体的可能性
- 10、frequency_penalty（频繁惩罚）：可选，数字，默认为0，介于-2到2之间。正值会根据至今文本中新词的出现频数进行惩罚，降低模型逐字城府同一行的可能性
- 11、logit_bias：map（字典），可选，默认为null。修改生成的回答中特定词出现的可能性
- 12、user：字符串，可选。代表最终用户的唯一标识符，可帮助openai检测滥用行为

代码示例：requests库

```
import requests
import json

url = 'https://api.openai.com/v1/chat/completions'
headers = {'Authorization': 'Bearer sk-9YW1pfFijCTzLwRtduowT3B1bkFJqE0x8PY1sYLdNyko00qN'}
```

```
,
        'Content-Type': 'application/json'}
data = {"model": "gpt-3.5-turbo", "messages":
[{"role": "user", "content": "你好，我能为你做些什么?" }]}, "temperature": 0.7}
data = json.dumps(data)
response =
requests.post(url=url, data=data, headers=headers)
print(response.text)
```

代码示例：openai库，使用ChatCompletion模块

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
completion = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "user", "content": "Hello!"}])
print(completion.choices[0].message)
```

创建edit：

post请求，给定提示和指令，模型将根据指令修改提示词或句子

官方示例：

```
{
  "model": "text-davinci-edit-001",
  "input": "What day of the wek is it?",
  "instruction": "Fix the spelling mistakes",}
```

请求结构体：

model、temperature、top_p，见上面

input：可选，默认为“”，作为编辑起点的输入文本

instruction：必选，告诉模型如何去编辑提示词

n：可选，默认为1，产生多少个编辑后的结果

示例代码：注意chat和edit两个的url是不一样的

```
import requests
import json

url = 'https://api.openai.com/v1/edits'
headers = {'Authorization': 'Bearer
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'
},
        'Content-Type': 'application/json'}
data = {"model": "text-davinci-edit-001",
        "input": "What day of the wek is it?",
        "instruction": "Fix the spelling mistakes"}
data = json.dumps(data)
response =
requests.post(url=url, data=data, headers=headers)
print(response.text)
```

示例代码：openai版本，使用Edit模块：

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.Edit.create(
    model="text-davinci-edit-001",
    input="What day of the wek is it?",
    instruction="Fix the spelling mistakes")
```

Images：目前可根据给定的提示词或一幅输入的图像，生成一幅全新的图像

Image edit：对于给定的图像和提示词，编辑或扩展图像

Image variation：对于给定的图像，创建该图像的变种

Embeddings：嵌入，对于给定的文本，生成嵌入向量

Audio 和 transcription：将音频转化为文本

Translation：翻译文本

Moderation：将其分类，看是不是违反了openai的政策

Files：文件用于上传文档，并可以和微调等功能一起使用，使用File模块

list files：使用get方法，返回所有属于使用者的文件

一般包含两个预训练的文件：train.jsonl和puppy.jsonl

```
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.File.list()
```

upload file：上传文件，post方法。文件必须是json格式的

url为：https://api.openai.com/v1/files

参数：

file：字符串，必填。是jsonlines文件的文件名（含后缀）

purpose：字符串，必填。是该文件的使用目的。若期望用于微调，则使用'fine-tune'

注意：purpose也需使用-F命令，即作为文件上传

官方代码示例：

```
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.File.create(
    file=open("mydata.jsonl", "rb"),
    purpose='fine-tune')
```

delete file：删除一个文件

delete函数，字符串，必填

官方代码示例：

```
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.File.delete("file-XjGxS3KTG0uNmNOK362iJua3")
```

Fine-tuning：微调，将预训练的模型微调成定制模型，详见迁移学习

均使用FineTune模块

创建微调：create函数

对于给定的数据集，创建一个微调的特定模型

参数：不完全版本，省略了一些没啥用的

training_file：字符串，必填，值为需要使用的已上传的文件的ID

validation_file：字符串，可选，包含validation data（验证集）的文件ID

model：字符串，可选，默认为curie。可在ada、babbage、curie和

davinci中选择一个

n_epochs: 整数, 可选, 默认为4, 即训练的次数

batch_size: 整数, 可选, 默认为null。默认代表动态选择一个大小的batch数据集进行训练

learning_rate_multiplier: 数字, 可选, 默认为null。即学习率, 默认则根据batchsize来决定

prompt_loss_weight: 数字, 可选, 默认0.01。用于提示词损失的权重。如果提示词非常长(相对于回应来说), 可减少此项防止过拟合

compute_classification_metrics: 布尔, 可选, 默认为false。如果设置, 则会返回计算的分类矩阵

suffix: 字符串, 可选, 最多40个字符可以加入到微调模型名称中

```
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.FineTune.create(training_file="file-XGinujblHPwGLSztz8cPS8XY")
```

返回所有微调模型名称:

```
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.FineTune.list()
```

retrieve fine-tune: 返回某个模型的信息

```
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.FineTune.retrieve(id="ft-AF1WoRqd3aJAHsqc9NY7iL8F")
```

取消微调:

```
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.FineTune.cancel(id="ft-AF1WoRqd3aJAHsqc9NY7iL8F")
```

删除微调模型:

```
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.Model.delete("curie:ft-acmeco-2021-03-03-21-44-20")
```

相关问题:

- 1、若报错: 未提供API_KEY, 则不用os, 而是直接openai.api_key=key就行了, 不需要用os的函数
- 2、messages的进阶使用:
messages是一个列表, 字典为列表的元素, 一个字典即一个单方面的对话。因此可以用messages存储多轮对话, 提问时一起发送过去, 此时ChatGPT即可“有记忆”
- 3、催眠问题: 即使有催眠, 不要一开始就说出过于违禁的句子, 需要经过几轮对话后, 用前几轮的对话给她参考, 进一步催眠她, 此时才能够为所欲为...

API调用代码示例

2023年5月11日 16:44

最基本的对话:

```
import openai

content = '好久不见'
key = "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"
messages = [{'role': 'user', 'content': content}]

openai.api_key =
"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"
completion = openai.ChatCompletion.create(model='gpt-3.5-turbo', messages=messages)
# message中是十六进制的字符串，想要转换成中文，只需先编码成
字节流（即二进制），再解码回字符串
print(completion.choices[0].message['content'].encode().decode('UTF-8'))
```

带有上下文的多轮对话:

```
import openai

prompt = ''
openai.api_key =
"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"
messages = [{'role': 'user', 'content': ''},
             {'role': 'assistant', 'content': ''},
             {'role': 'user', 'content': ''},
             {'role': 'assistant', 'content': ''}]

while True:
    chat = input('chat:')
    if chat == 'quit':
        print('结束...')
        break
    else:
        question = {'role': 'user', 'content': prompt+chat}
        messages.pop(0)      # 第一个去掉，防止上文对话过多
        messages.append(question)  # 把本次的问题加入到
messages中
        completion =
openai.ChatCompletion.create(model='gpt-3.5-
```

```
turbo', messages=messages)      # completion即回应
    messages.pop(-1)             # 把本次的对话换成没有prompt的版本，防止prompt的冗余占用大量tokens
    messages.append({'role': 'user', 'content': chat})
    response =
completion.choices[0].message['content'].encode().decode('UTF-8')    # 回应的转换
    answer = {'role': 'assistant', 'content': response}      # 把
answer也做成message，加入messages
    messages.pop(0)
    messages.append(answer)
    # message中是十六进制的字符串，想要转换成中文，只需先编码成字节流（即二进制），再解码回字符串
    print(response)
```