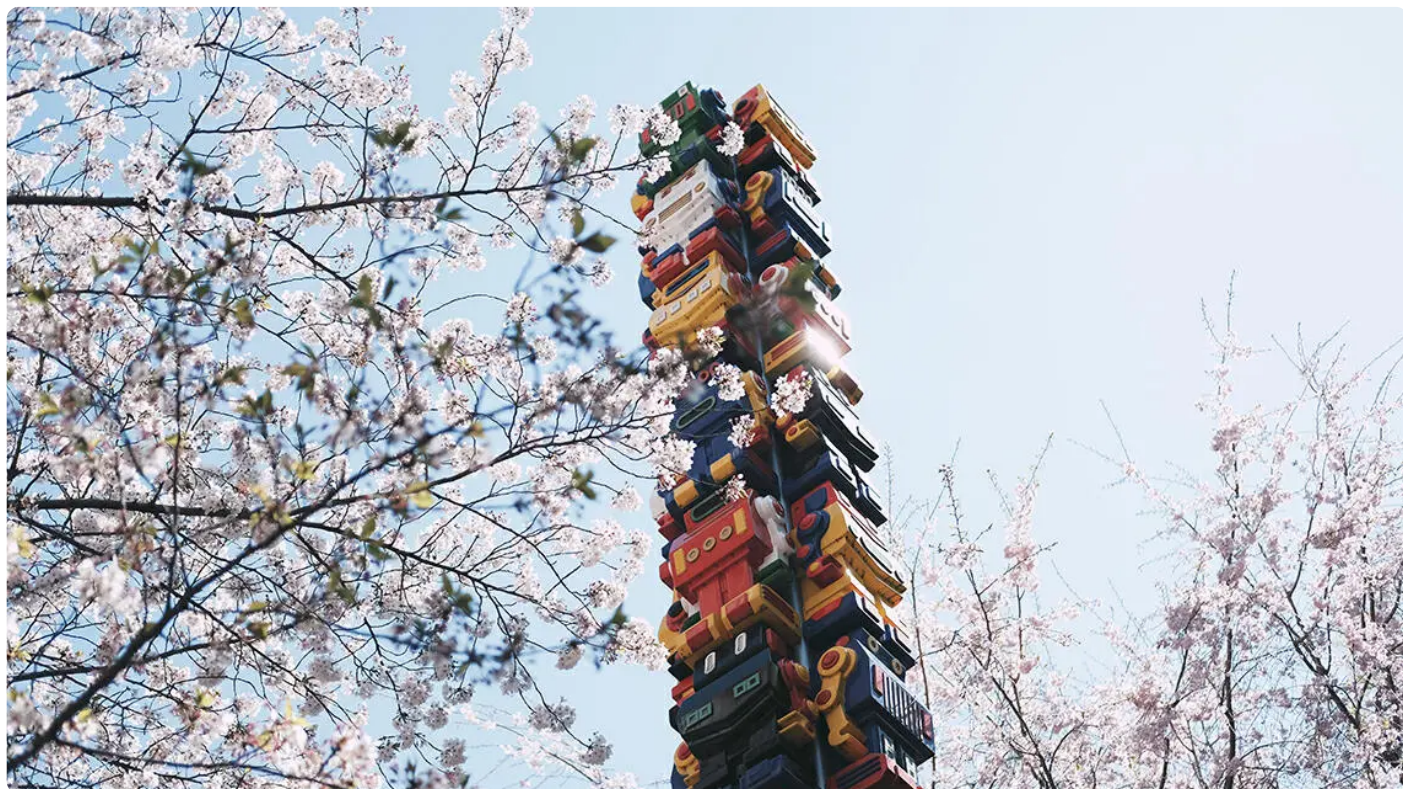


04 | 新时代模型性能大比拼，GPT-3到底胜在哪里？

2023-03-27 徐文浩 来自北京

《AI大模型之美》



你好，我是徐文浩。

前面两讲，我带你体验了 OpenAI 通过 API 提供的 GPT-3.5 系列模型的两个核心接口。一个是获取一段文本的 Embedding 向量，另一个则是根据提示语，直接生成一段补全的文本内容。我们用这两种方法，都可以实现零样本（zero-shot）或者少样本下的情感分析任务。不过，你可能会提出这样两个疑问。

1. Embedding 不就是把文本变成向量吗？我也学过一些自然语言处理，直接用个开源模型，比如 Word2Vec、Bert 之类的就好了呀，何必要去调用 OpenAI 的 API 呢？
2. 我们在这个情感分析里做了很多投机取巧的工作。一方面，我们把 3 分这种相对中性的评价分数排除掉了；另一方面，我们把 1 分 2 分和 4 分 5 分分别合并在了一起，把一个原本需要判断 5 个分类的问题简化了。那如果我们想要准确地预测多个分类，也会这么简单吗？

那么，这一讲我们就先来回答第一个问题。我们还是拿代码和数据来说话，就拿常见的开源模型来试一试，看看能否通过零样本学习的方式来取得比较好的效果。第二个问题，我们下一讲再来探讨，看看能不能利用 Embedding 进一步通过一些机器学习的算法，来更好地处理情感分析问题。

什么是预训练模型？

给出一段文本，OpenAI 就能返回给你一个 Embedding 向量，这是因为它的背后是 GPT-3 这个超大规模的预训练模型（Pre-trained Model）。事实上，GPT 的英文全称翻译过来就是“生成式预训练 Transformer（Generative Pre-trained Transformer）”。

所谓预训练模型，就是虽然我们没有看过你想要解决的问题，比如这里我们在情感分析里看到的用户评论和评分。但是，我可以拿很多我能找到的文本，比如网页文章、维基百科里的文章，各种书籍的电子版等等，**作为理解文本内容的一个学习资料。**

我们不需要对这些数据进行人工标注，只根据这些文本前后的内容，来习得文本之间内在的关联。比如，网上的资料里，会有很多“小猫很可爱”、“小狗很可爱”这样的文本。小猫和小狗后面都会跟着“很可爱”，那么我们就知道小猫和小狗应该是相似的词，都是宠物。同时，一般我们对于它们的情感也是正面的。这些隐含的内在信息，在我们做情感分析的时候，就带来了少量用户评论和评分数据里缺少的“常识”，这些“常识”也有助于我们更好地预测。

比如，文本里有“白日依山尽”，那么模型就知道后面应该跟“黄河入海流”。文本前面是“今天天气真”，后面跟着的大概率是“不错”，小概率是“糟糕”。这些文本关系，最后以一堆参数的形式体现出来。对于你输入的文本，它可以根据这些参数计算出一个向量，然后根据这个向量，来推算这个文本后面的内容。

可以这样来理解：**用来训练的语料文本越丰富，模型中可以放的参数越多，那模型能够学到的关系也就越多。类似的情况在文本里出现得越多，那么将来模型猜得也就越准。**

预训练模型在自然语言处理领域并不是 OpenAI 的专利。早在 2013 年，就有一篇叫做 Word2Vec 的经典论文谈到过。它能够通过预训练，根据同一个句子里一个单词前后出现的单词，来得到每个单词的向量。而在 2018 年，Google 关于 BERT 的论文发表之后，整个业

界也都会使用 BERT 这样的预训练模型，把一段文本变成向量用来解决自己的自然语言处理任务。在 GPT-3 论文发表之前，大家普遍的结论是，BERT 作为预训练的模型效果也是优于 GPT 的。


Fasttext、T5、GPT-3 模型效果大比拼

今天我们就拿两个开源的预训练模型，来看看直接用它们对文本进行向量化，是不是也能取得和 OpenAI 的 API 一样好的效果。

第一个是来自 Facebook 的 Fasttext，它继承了 Word2Vec 的思路，能够把一个个单词表示成向量。第二个是来自 Google 的 T5，T5 的全称是 Text-to-Text Transfer Transformer，是适合做迁移学习的一个模型。所谓迁移学习，也就是它推理出来向量的结果，常常被拿来再进行机器学习，去解决其他自然语言处理问题。通常很多新发表的论文，会把 T5 作为预训练模型进行微调和训练，或者把它当作 Benchmark 来对比、评估。

Fasttext 效果测试

我们先来试一下 Fasttext，在实际运行代码之前，我们需要先安装 Fasttext 和 Gensim 这两个 Python 包。我在下面列出了通过 Conda 安装对应 Python 包的代码，如果你使用的是 PIP 或者其他 Python 包管理工具，你就换成对应的 PIP 命令就好了。

 复制代码

```
1 conda install gensim
2 conda install fasttext
```

然后，我们要把 Fasttext 对应的模型下载到本地。因为这些开源库和对应的论文都是 Facebook 和 Google 这样的海外公司发布的，效果自然是在英语上比较好，所以我们就下载对应的英语模型，名字叫做 “cc.en.300.bin”。同样的，对应模型的下载链接，我也放在 [这里](#)了。

下载之后解压，然后把文件放在和 Notebook 相同的目录下，方便我们接下来运行代码。

这里我们拿来测试效果的数据集还是和 [🔗第 02 讲](#) 一样，用的是 2.5w 条亚马逊食物评论的数据。

[illegible]

注：各种语言的 Fasttext 模型。


代码的逻辑也不复杂，我们先利用 Gensim 这个库，把 Facebook 预训练好的模型加载进来。然后，我们定义一个获取文本向量的函数。因为 Fasttext 学到的是单词的向量，而不是句子的向量。同时，因为我们想要测试一下零样本学习的效果，不能再根据拿到的评论数据进一步训练模型了。所以我们把一句话里每个单词的向量，加在一起平均一下，把得到的向量作为整段评论的向量。这个方法也是当年常用的一种将一句话变成向量的办法。我们把这个操作定义成了 get_fasttext_vector 这个函数，供后面的程序使用。

 复制代码

```
1 import gensim
2 import numpy as np
3 # Load the FastText pre-trained model
4 model = gensim.models.fasttext.load_facebook_model('cc.en.300.bin')
5
6 def get_fasttext_vector(line):
7     vec = np.zeros(300) # Initialize an empty 300-dimensional vector
8     for word in line.split():
9         vec += model.wv[word]
10    vec /= len(line.split()) # Take the average over all words in the line
11    return vec
```


而对应的零样本学习，我们还是和🔗第 02 讲一样，将需要进行情感判断的评论分别与 “An Amazon review with a positive sentiment.” 以及 “An Amazon review with a negative sentiment.” 这两句话进行向量计算，算出它们之间的余弦距离。

离前一个近，我们就认为是正面情感，离后一个近就是负面情感。

 复制代码

```
1 positive_text = """Wanted to save some to bring to my Chicago family but my North
2 negative_text = """First, these should be called Mac - Coconut bars, as Coconut i
3
4 positive_example_in_fasttext = get_fasttext_vector(positive_text)
5 negative_example_in_fasttext = get_fasttext_vector(negative_text)
6
7 positive_review_in_fasttext = get_fasttext_vector("An Amazon review with a positi
8 negative_review_in_fasttext = get_fasttext_vector('An Amazon review with a negati
9
10 from openai.embeddings_utils import cosine_similarity
11
12 def get_fasttext_score(sample_embedding):
13     return cosine_similarity(sample_embedding, positive_review_in_fasttext) - cosin
14
15 positive_score = get_fasttext_score(positive_example_in_fasttext)
16 negative_score = get_fasttext_score(negative_example_in_fasttext)
17
18 print("Fasttext好评例子的评分 : %f" % (positive_score))
19 print("Fasttext差评例子的评分 : %f" % (negative_score))
```

输出结果：

 复制代码

```
1 Fasttext好评例子的评分 : -0.000544
2 Fasttext差评例子的评分 : 0.000369
```

我们从亚马逊食物评论的数据集里，选取了一个用户打 5 分的正面例子和一个用户打 1 分的例子试了一下。结果非常不幸，通过这个零样本学习的方式，这两个例子，程序都判断错了。

不过，仔细想一下，这样的结果也正常。因为这里的整句向量就是把所有单词的向量平均了一下。这意味着，**可能会出现我们之前说过的单词相同顺序不同的问题。**

“not good, really bad” 和 “not bad, really good” ，在这个情况下，意思完全不同，但是向量完全相同。更何况，我们拿来做对比的正面情感和负面情感的两句话，只差了 positive/negative 这样一个单词。不考虑单词的顺序，而只考虑出现了哪些单词，并且不同单词之间还平均一下。这种策略要是真的有很好的效果，你反而要担心是不是哪里有 Bug。


T5 效果测试

Fasttext 出师不利，毕竟 Word2Vec 已经是 10 年前的技术了，可以理解。那么，我们来看看和 GPT 一样使用了现在最流行的 Transformer 结构的 T5 模型效果怎么样。

T5 模型的全称是 Text-to-Text Transfer Transformer，翻译成中文就是“文本到文本的迁移 Transformer”，也就是说，这个模型就是为了方便预训练之后拿去“迁移”到别的任务上而创造出来的。当时发表的时候，它就在各种数据集的评测上高居榜首。

T5 最大的模型也有 110 亿个参数，也是基于 Transformer，虽然比起 GPT-3 的 1750 亿小了不少，但是对硬件的性能要求也不低。所以，我们先测试一下 T5-Small 这个小模型看看效果。

同样的，在实际运行代码之前，我们也需要安装对应的 Python 包。这里我们分别安装了 SentencePiece 和 PyTorch。在安装 PyTorch 的时候，我一并安装了 Torchvision，后面课程会用到。

 复制代码

```
1 conda install transformers -c conda-forge
2 conda install pytorch torchvision -c pytorch
3 conda install sentencepiece
```

代码也不复杂，我们先加载预训练好的 T5 模型的分词器（Tokenizer），还有对应的模型。然后，我们定义了一个 get_t5_vector 函数，它会接收一段你的文本输入，然后用分词器来分

词把结果变成一个序列，然后让模型的编码器部分对其进行编码。编码后的结果，仍然是分词后的一个词一个向量，我们还是把这些向量平均一下，作为整段文本的向量。


不过要注意，虽然同样是平均，但是和前面 Fasttext 不一样的是，这里每个词的向量，随着位置以及前后词的不同，编码出来的结果是不一样的。所以**这个平均值里，仍然包含了顺序带来的语义信息。**

这段代码执行的过程可能会有点慢。因为第一次加载模型的时候，Transformer 库会把模型下载到本地并缓存起来，整个下载过程会花一些时间。

 复制代码

```
1 from transformers import T5Tokenizer, T5Model
2 import torch
3
4 # load the T5 tokenizer and model
5 tokenizer = T5Tokenizer.from_pretrained('t5-small', model_max_length=512)
6 model = T5Model.from_pretrained('t5-small')
7
8 # set the model to evaluation mode
9 model.eval()
10
11 # encode the input sentence
12 def get_t5_vector(line):
13     input_ids = tokenizer.encode(line, return_tensors='pt', max_length=512, trunc
14     # generate the vector representation
15     with torch.no_grad():
16         outputs = model.encoder(input_ids=input_ids)
17         vector = outputs.last_hidden_state.mean(dim=1)
18     return vector[0]
```

有了模型和通过模型获取的向量数据，我们就可以再试一试前面的零样本学习的方式，来看看效果怎么样了。我们简单地把之前获取向量和计算向量的函数调用，都换成新的 `get_t5_vector`，运行一下就能看到结果了。

 复制代码

```
1 positive_review_in_t5 = get_t5_vector("An Amazon review with a positive sentiment
2 negative_review_in_t5 = get_t5_vector('An Amazon review with a negative sentiment
3
4 def test_t5():
```

```

5 positive_example_in_t5 = get_t5_vector(positive_text)
6 negative_example_in_t5 = get_t5_vector(negative_text)
7
8 def get_t5_score(sample_embedding):
9     return cosine_similarity(sample_embedding, positive_review_in_t5) - cosine_si
10
11 positive_score = get_t5_score(positive_example_in_t5)
12 negative_score = get_t5_score(negative_example_in_t5)
13
14 print("T5好评例子的评分 : %f" % (positive_score))
15 print("T5差评例子的评分 : %f" % (negative_score))
16
17 test_t5()

```

输出结果：

 复制代码


```

1 T5好评例子的评分 : -0.010294
2 T5差评例子的评分 : -0.008990

```

不幸的是，结果还是不太好，两个例子都被判断成了负面情绪，而且好评的分数还更低一点。不过别着急，会不会是我们用的模型太小了呢？毕竟 T5 论文里霸占各个排行榜的是 110 亿个参数的大模型，我们这里用的是 T5-Small 这个同样架构下的小模型，参数数量只有 6000 万个。

110 亿个参数要花太多时间了，我们不妨把模型放大一下，试试有 2.2 亿个参数的 T5-Base 这个模型？试用起来也很简单，我们就直接把上面模型的名字从 T5-small 改成 T5-base 就好了，其他代码不需要动，重新运行一遍。

 复制代码

```

1 tokenizer = T5Tokenizer.from_pretrained('t5-base', model_max_length=512)
2 model = T5Model.from_pretrained('t5-base')
3
4 # set the model to evaluation mode
5 model.eval()
6
7 # encode the input sentence
8 def get_t5_vector(line):

```



```
9     input_ids = tokenizer.encode(line, return_tensors='pt', max_length=512, trunc
10     # generate the vector representation
11     with torch.no_grad():
12         outputs = model.encoder(input_ids=input_ids)
13         vector = outputs.last_hidden_state.mean(dim=1)
14     return vector[0]
15
16 positive_review_in_t5 = get_t5_vector("An Amazon review with a positive sentiment
17 negative_review_in_t5 = get_t5_vector('An Amazon review with a negative sentiment
18
19 test t5()
```

输出结果：

 复制代码

```
1 T5好评例子的评分 : 0.010347
2 T5差评例子的评分 : -0.023935
```

这一次，结果似乎是我们想要的了，好评被判定为正面情感，而差评被判定为负面情感。不过，也许我们只是运气好，在这一两个例子上有效果呢？所以，接下来让我们把整个数据集里，1分2分的差评和4分5分的好评都拿出来看一看。在 OpenAI 的 API 拿到的 Embedding 里，它的准确率能够达到 95% 以上，我们看看用这个有 2.2 亿个参数的 T5-base 模型能有什么样的结果。

对应的代码也不复杂，基本上和 [第 02 讲](#) 里 OpenAI 给到的 Embedding 代码是类似的。无非是通过 pandas，根据评论的 Text 字段，全部计算一遍 T5 下的 Embedding，然后存到 DataFrame 的 t5_embedding 里去。

同样的，我们还是要通过 T5 的模型，来获得 “An Amazon review with a positive sentiment.” 以及 “An Amazon review with a negative sentiment.” 这两句话的 Embedding。然后，我们用刚刚计算的用户评论的 Embedding 和这两句话计算一下余弦距离，来判断这些评论是正面还是负面的。

最后，通过 Scikit-learn 的分类报告类库把评估的报告结果打印出来。

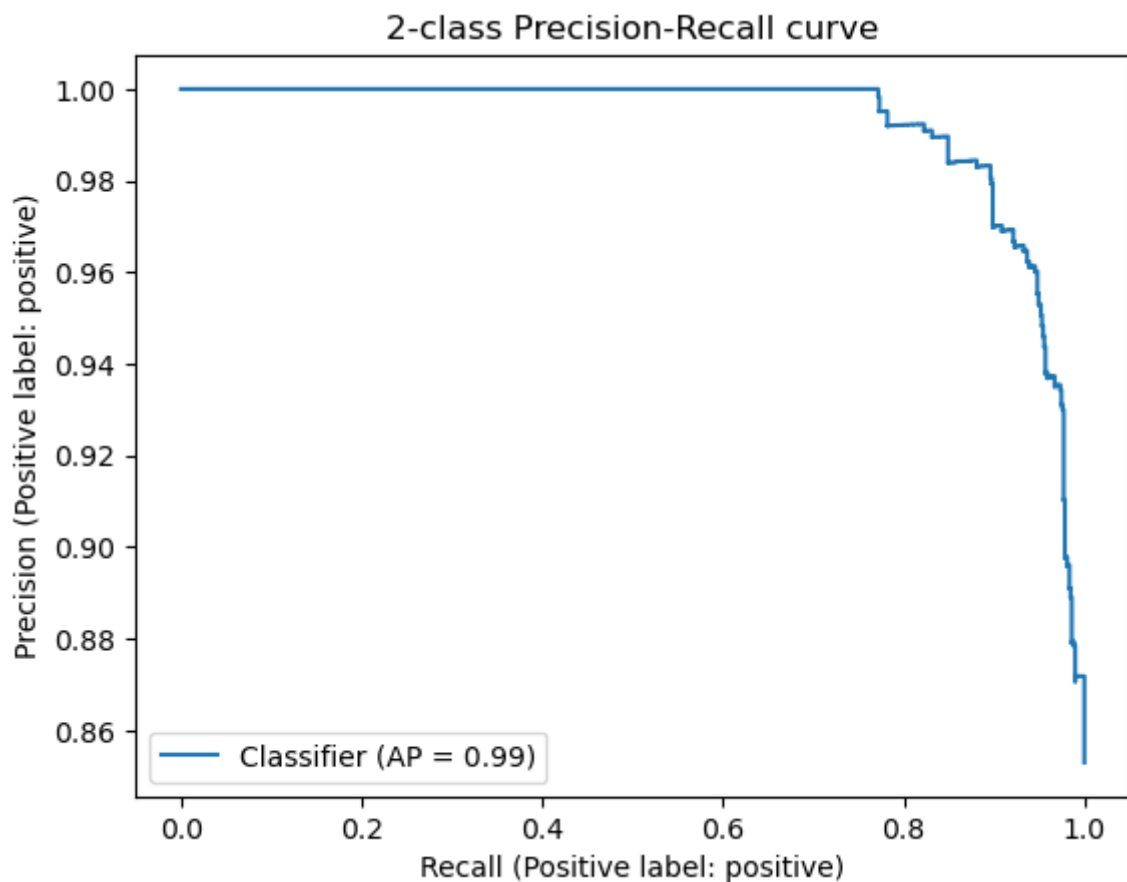
[复制代码](#)

```
1 import pandas as pd
2 from sklearn.metrics import classification_report
3
4 datafile_path = "data/fine_food_reviews_with_embeddings_1k.csv"
5
6 df = pd.read_csv(datafile_path)
7
8 df["t5_embedding"] = df.Text.apply(get_t5_vector)
9 # convert 5-star rating to binary sentiment
10 df = df[df.Score != 3]
11 df["sentiment"] = df.Score.replace({1: "negative", 2: "negative", 4: "positive",
12
13 from sklearn.metrics import PrecisionRecallDisplay
14 def evaluate_embeddings_approach():
15     def label_score(review_embedding):
16         return cosine_similarity(review_embedding, positive_review_in_t5) - cosin
17
18     probas = df["t5_embedding"].apply(lambda x: label_score(x))
19     preds = probas.apply(lambda x: 'positive' if x>0 else 'negative')
20
21     report = classification_report(df.sentiment, preds)
22     print(report)
23
24     display = PrecisionRecallDisplay.from_predictions(df.sentiment, probas, pos_l
25     _ = display.ax_.set_title("2-class Precision-Recall curve")
26
27 evaluate_embeddings_approach()
```

输出结果:

[复制代码](#)

	precision	recall	f1-score	support
negative	0.60	0.90	0.72	136
positive	0.98	0.90	0.94	789
accuracy			0.90	925
macro avg	0.79	0.90	0.83	925
weighted avg	0.93	0.90	0.91	925



结果显示，使用 T5 的效果也还可以，考虑所有样本的准确率也能达到 90%。但是，在比较困难的差评的判断里，它的表现要比直接用 OpenAI 给到的 Embedding 要差很多，整体的精度只有 60%。我们去看整体模型的准确率的话，OpenAI 的 Embedding 能够到达 96%，还是比这里的 90% 要好上一些的。

复制代码

		precision	recall	f1-score	support
1					
2	negative	0.98	0.73	0.84	136
3	positive	0.96	1.00	0.98	789
4	accuracy			0.96	925
5	macro avg	0.97	0.86	0.91	925
6	weighted avg	0.96	0.96	0.96	925

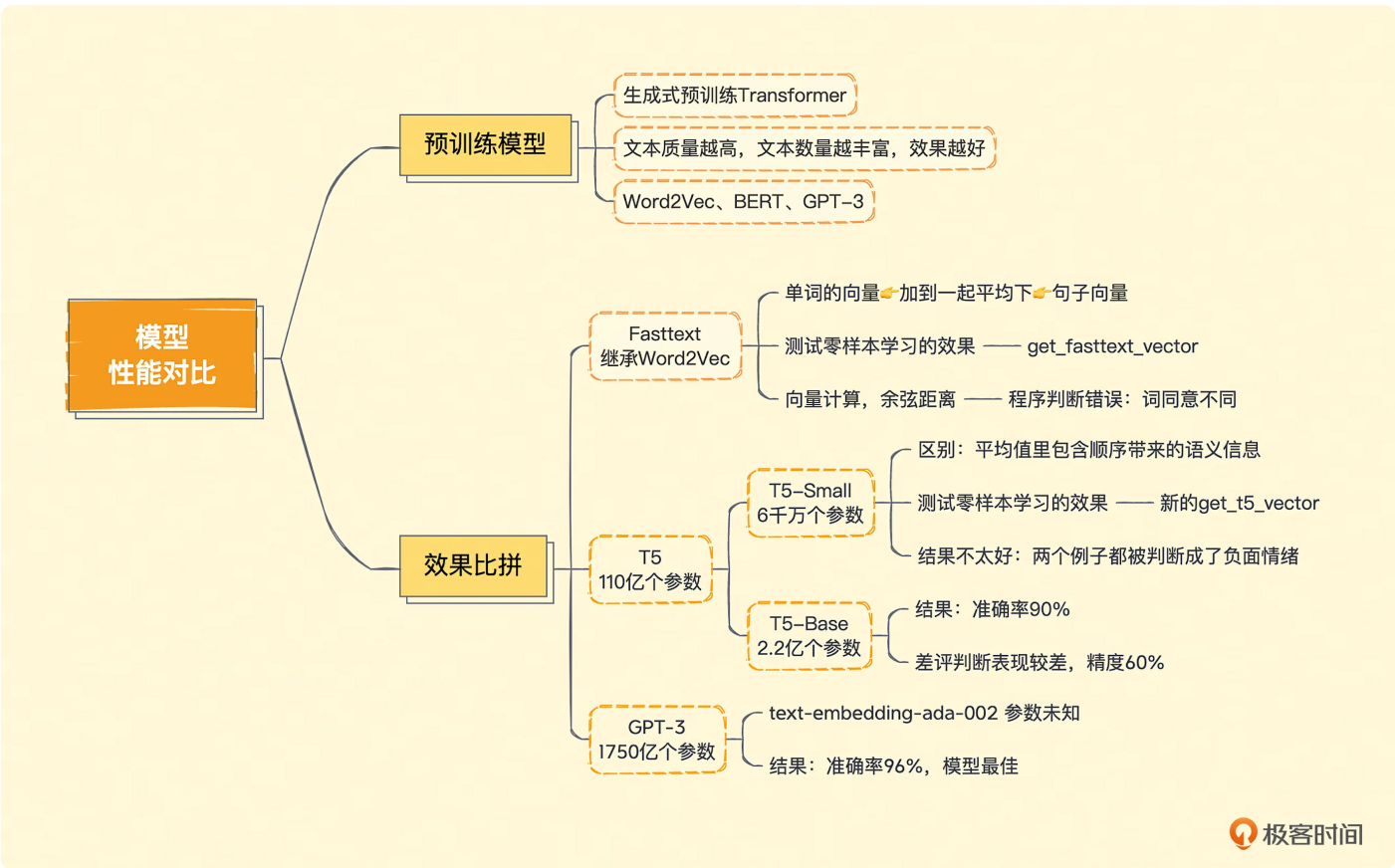
这里我重新贴一下使用 OpenAI 的 Embedding 取得的效果，你可以做个对比。

当然，这个分数也还不错，也能作为一个合格的情感分析分类器的基准线了。毕竟，我们这里采用的是零样本分类的方法，没有对需要分类的数据做任何训练，使用的完全是预训练模型给

出来的向量，直接根据距离做的判断。所以，看起来大一点的预训练模型的确有用，能够取得更好的效果。而且，当你因为成本或者网络延时的问题，不方便使用 OpenAI 的 API 的时候，如果只是要获取文本的 Embedding 向量，使用 T5 这样的开源模型其实效果也还不错。

小结

最后，我们来复习一下这一讲的内容。



这一讲我们一起使用 Fasttext、T5-small 和 T5-base 这三个预训练模型，做了零样本分类测试。在和之前相同的食物评论的数据集上，使用只学习了单词向量表示的 Fasttext，效果很糟糕。当我们换用同样基于 Transformer 的 T5 模型的时候，T5-small 这个 6000 万参数的小模型其实效果也不好。但是当我们用上 2.2 亿参数的 T5-base 模型的时候，结果还可以。不过，还是远远比不上直接使用 OpenAI 的 API 的效果。可见，模型的大小，即使是对情感分析这样简单的问题，也能产生明显的差距。

课后练习

1. 我们在尝试使用 T5-base 这个模型之后，下了个判断认为大一点的模型效果更好。不过，其实我们并没有在整个数据集上使用 T5-small 这个模型做评测，你能试着修改一下代码，用 T5-small 测试一下整个数据集吗？测试下来的效果又是怎样的呢？
2. 我们使用 Fasttext 的时候，把所有的单词向量平均一下，用来做情感分析效果很糟糕。那么什么样的分类问题，可以使用这样的方式呢？给你一个小提示，你觉得什么样的文本分类，只关心出现的单词是什么，而不关心它们的顺序？

期待你的思考，也欢迎你把这节课分享给感兴趣的朋友，我们下一讲再见。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (24)



Daniel

2023-03-28 来自北京

BERT: BERT 基于 Transformer 的 (Encoder)。BERT 使用双向 (bidirectional) 的自注意力机制，可以同时捕捉文本中的前后上下文信息。

GPT: GPT 基于 Transformer 的 (Decoder)。GPT 使用单向 (unidirectional) 的自注意力机制，只能捕捉文本中的前文 (left context) 信息。

能否请老师详细讲一下，这两者的差别？

作者回复: BERT是

白日依山尽，____，欲穷千里目

GPT是

白日依山尽，____

一个是完形填空，一个是续写。GPT没法看到后面的东西，所以在很多语义理解的指标上不如BERT。

但是很多真实的使用场景你看不到后面的东西，所以从AGI的路线上，很多人觉得GPT才是正确路径。



👍 29



stg609

2023-03-28 来自浙江

老师能说说

1. davinci, ada 等模型与gpt3的关系吗?
2. gpt3有1750亿参数, 那是不是 ada 也有这么大量的参数

作者回复: 1. 都是 GPT 家族的模型, ada, babbage, curie, davinci 模型从小到大
2. ada模型应该要小得多, 所以便宜



6



Roy Liang

2023-03-27 来自广东

1. 小数据集验证结果是这样, 但是门外汉其实不懂表格里什么意思

precision recall f1-score support

negative 0.25 0.99 0.40 136

positive 1.00 0.48 0.65 789

accuracy 0.56 925

macro avg 0.62 0.74 0.52 925

weighted avg 0.89 0.56 0.61 925

2. 可能适合新闻分类、垃圾邮件分类等不关心词语次序的场景吧

作者回复: 1. 往后看一讲, 第5讲里会具体解释这些指标的含义。

2. 对, 主题分类不太关注语序

共 2 条评论 >



5



Geek_61af67

2023-03-27 来自北京

不关心顺序的话, 对tags进行分析会不会比较合适?

作者回复: tags的确不太关注顺序



4



Steven

2023-04-08 来自辽宁

Fasttext 效果测试时碰到异常：

DLL load failed while importing _imaging

更新 pillow 版本，然后重启 Notebook 即可

pip install --upgrade pillow



2



HXL

2023-04-02 来自北京

还是没明白什么是 预训练模型

作者回复：预训练模型，顾名思义，就是“预先训练好的模型”

也就是这个模型，用了别的很多很多数据训练好了。可能和我们现在要解决的问题的数据有关，也有可能没有关系。

但是因为 预训练模型 通常通过海量的数据训练的，它多少对你现在要解决的问题的知识是有了解和帮助的。



2



Geek_93970d

2023-05-12 来自北京

load_facebook_model 这个函数就得卡好半天，模型文件6.7G，加载很慢。另外 fasttext 既然用的是 gensim 里的，那就不用单独安装了吧？安装也安装不上，PackagesNotFoundError。

作者回复：pip install fasttext 不行么？



1



qingtama

2023-04-21 来自北京

请问老师，这里的2.2亿参数，可以理解成向量所在的维度是2.2亿个吗？

作者回复：不能，输出的向量维度没有那么大。

是指所谓的 transformer模型里面的各种变量参数有2.2亿个。



👍 1



摩西

2023-04-16 来自广东

刚接触机器学习，基础比较薄弱，请问老师 Transformer 是指什么？这里的transformer 跟 huggingface 中的transformer是相同的内容吗？

作者回复: transformers 是指一种深度学习的基础模型架构，huggingface的transformers库，相当于为这类模型架构的开发、部署、试用定义了一个通用的接口形式。



👍 1



Steven

2023-04-08 来自辽宁

Win10 下 fasttext 安装 (Linux 环境未尝试) :

1, 从 <https://www.lfd.uci.edu/~gohlke/pythonlibs/#fasttext> 下载fasttext-0.9.2-cp310-cp310-win_amd64.whl 文件

2, 执行 pip install fasttext-0.9.2-cp310-cp310-win_amd64.whl 安装

共 1 条评论 >

👍 1



王昊翔Harry

2023-04-04 来自英国

这一套流程有没有在Colab友好的。本身没有编程经验该怎么着手？

作者回复: 我在 <https://github.com/xuwenhao/geektime-ai-course> 放了可以在Colab运行的Notebook 代码，只要通过 pip 安装好依赖的包就能够运行。



👍 1



小神david

2023-03-27 来自北京

鉴黄鉴暴



👍 1



牧海

2023-05-26 来自广东

AttributeError Traceback (most recent call last)

Cell In[11], line 3

```
1 # %load /Users/zongxi.lzx/gensim.py
```

```
2 #!/usr/bin/env python3
```

```
----> 3 import gensim
```

```
4 import numpy as np
```

```
5 # Load the FastText pre-trained model
```

File ~/gensim.py:5

```
3 import numpy as np
```

```
4 # Load the FastText pre-trained model
```

```
----> 5 model = gensim.models.fasttext.load_facebook_model('cc.en.300.bin')
```

```
7 def get_fasttext_vector(line):
```

```
8     vec = np.zeros(300) # Initialize an empty 300-dimensional vector
```

AttributeError: partially initialized module 'gensim' has no attribute 'models' (most likely due to a circular import)

老师，报这个错是什么原因呢？



王石磊

2023-05-13 来自土耳其

参考文中的用例，用T5-base 推理的结果如下，准确度为56%，这大概是什么原因呢？

precision recall f1-score support

negative	0.25	0.99	0.40	136
----------	------	------	------	-----

positive	1.00	0.48	0.65	789
----------	------	------	------	-----

accuracy			0.56	925
----------	--	--	------	-----

macro avg	0.62	0.74	0.52	925
-----------	------	------	------	-----

weighted avg	0.89	0.56	0.61	925
--------------	------	------	------	-----

作者回复：看一下哪里有问题？ T5-Base没有那么差，这个看起来像T5-Small乃至更小的模型的结果。



Geek_b7449f

2023-05-09 来自新加坡

1、课后题的 Embedding

	precision	recall	f1-score	support
negative	0.25	0.99	0.40	136
positive	1.00	0.48	0.65	789
accuracy			0.56	925
macro avg	0.62	0.74	0.52	925
weighted avg	0.89	0.56	0.61	925

2、个人认为应该属于文章分类，词库分类等场景更适合这种只关注单词，而非词序的。



王尼玛

2023-05-04 来自北京

mac 安装pytorch pip3 install torch torchvision torchaudio



Geek_40e894

2023-04-23 来自江苏

colab Load the FastText pre-trained model 内存不够，崩溃

作者回复: 应该是够的呀，Colab的机器有12G内存，看看是否有别的什么已经占了太多内容 restart jupyter runtime一下？



buffett

2023-04-23 来自北京

from transformers import T5Tokenizer, T5Model, 这个代码调用T5模型的也是通过通过api请求吗？是maas的方式吗？还是本地loaded了模型了

作者回复: 不是，本地加载模型了



意无尽

2023-04-21 来自重庆

老师，有点不是很理解的，就是文中讲的向量，可不可以这么理解：Fasttext 和 GPT3 的一个差别是 Fasttext 使用到的上下文比较少，所以很多场景下缺乏语义，导致判断失败，而 GPT3 包含的很多，所以更精准？

作者回复：1. Fasttext没有语序信息，只有单个单词的文本信息

2. 模型规模也有比较大差距



yange2004

2023-04-10 来自上海

如果用完整的110亿参数来测试，准确率应该会更高吧

作者回复：一般来说是这样的。

