

人工智能

项目报告

姓名：舒文桐 学号：2019201418





目录

contents

01

选题背景及意义
(Background &Significance)

02

研究思路与方法
(Research Thoughts and Methods)

03

关键技术与实现难点
(Tackling in Key Technologies)

04

成果展示
(Achievement display)

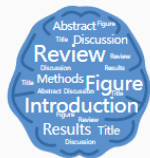
项目内容

01





□ 内容摘要



AI写作助手

(视频教程 1 2 3 4 5 6 7 8 9 10 11)

实时收录全网Online论文，基于BERT自然语言模型，根据关键词或短语（中/英）自动匹配出语义最相近的英文表述方式。非生命科学领域的同学可“自建语料库”。

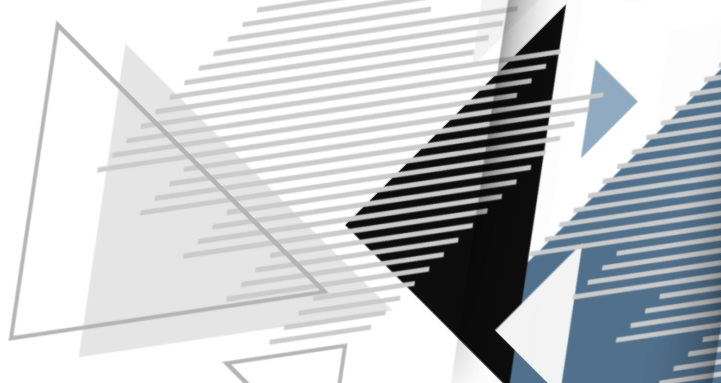
[登录工具](#)

实现一个简易弱化版的科研者之家的AI写作助手。

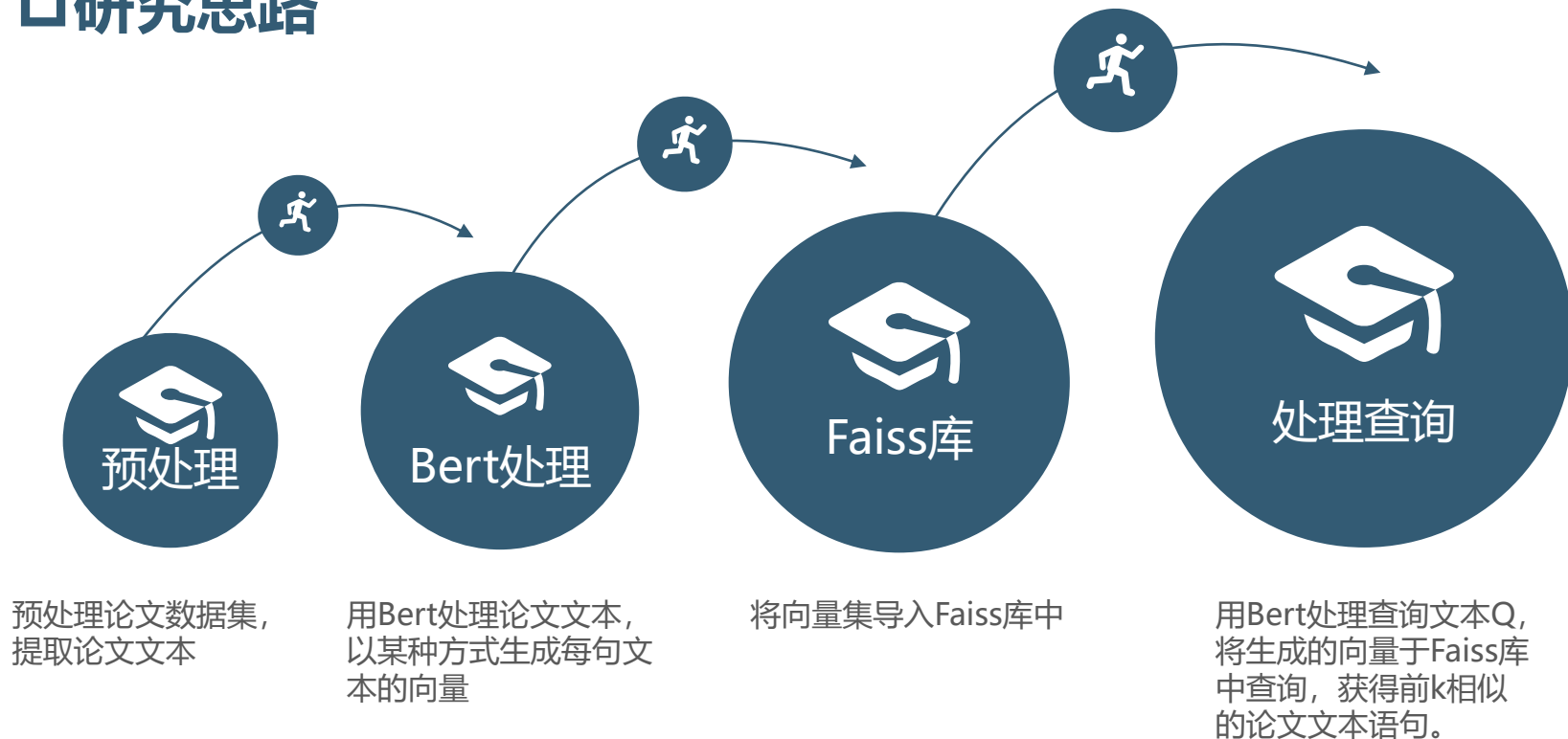
根据用户输出的一条英文语句，优化成与论文写作相关的k条相近语句。

研究思路与方法

02



□研究思路

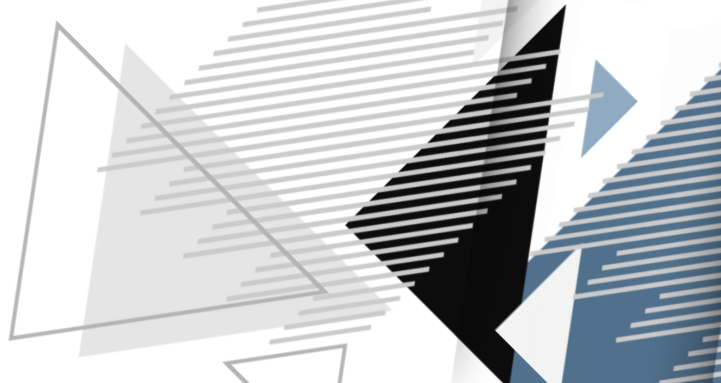


关键技术与实现难点

(Tackling in Key Technologies)

□ 关键技术

□ 实现难点



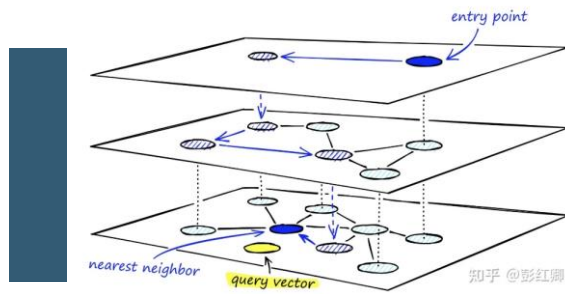


□关键技术

```

1 outdir = os.path.join(outdir, "outdir")
2
3 # Create a directory for the output
4 os.makedirs(outdir, exist_ok=True)
5
6 # Create a file for the output
7 with open(os.path.join(outdir, "output.txt"), "w") as f:
8     f.write("Output file created successfully.\n")
9
10 # Create a file for the output
11 with open(os.path.join(outdir, "output.txt"), "a") as f:
12     f.write("Additional output line.\n")
13
14 # Create a file for the output
15 with open(os.path.join(outdir, "output.txt"), "a") as f:
16     f.write("Final output line.\n")
17
18 # Print the output
19 print(f"Output file created successfully. Path: {os.path.join(outdir, 'output.txt')}")
20
21 # End of script
22

```

The logo for Google BERT, featuring the word "Google" in its multi-colored font and the word "BERT" in a brown, serif font below it.

提取论文文本

Bert处理

Faiss库



□ 实现难点

如何提取论文文本

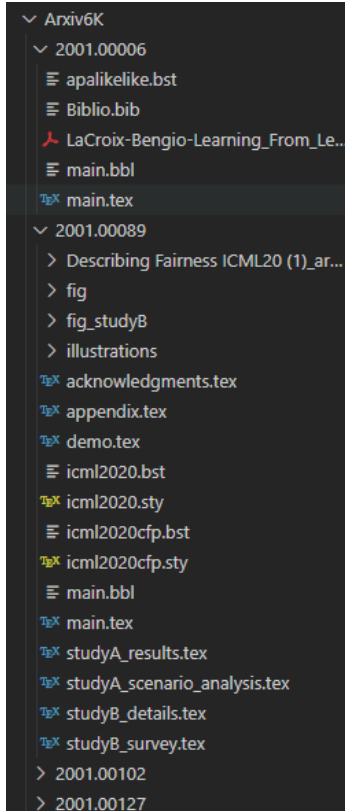
论文数据集文件类型非常杂乱，无法直接读取。

首先处理数据文件，只保留tex类型的latex文件。

接着使用脚本，调用pandoc把tex文件全部转换docx文件

然后再python中使用docx库，即可实现对文本数据的读入

再进行某种划分，即可划分出每个句子





□ 实现难点

如何提取论文文本

在划分的过程中遇到了很多问题

比如在使用脚本调用pandoc时，会出现很多链接文件找不到的问题

但如果将脚本调用前cd入该文档目录下，有些文档在pandoc时又会莫名其妙卡住

最终进行了调整，脚本pandoc转换+手动修正删除某些无法正常pandoc的文档

处理文本数据大概花了十个小时的时间，非常繁琐

```
Arxiv6K
├── 2001.00006
│   ├── apalikelike.bst
│   ├── Biblio.bib
│   ├── LaCroix-Bengio-Learning_From_Le...
│   ├── main.bbl
│   ├── main.tex
│   └── 2001.00089
│       ├── Describing Fairness ICML20 (1)_ar...
│       ├── fig
│       ├── fig_studyB
│       ├── illustrations
│       ├── acknowledgments.tex
│       ├── appendix.tex
│       ├── demo.tex
│       ├── icml2020.bst
│       ├── icml2020.sty
│       ├── icml2020cfp.bst
│       ├── icml2020cfp.sty
│       ├── main.bbl
│       ├── main.tex
│       ├── studyA_results.tex
│       ├── studyA_scenario_analysis.tex
│       ├── studyB_details.tex
│       └── studyB_survey.tex
├── 2001.00102
└── 2001.00127
```



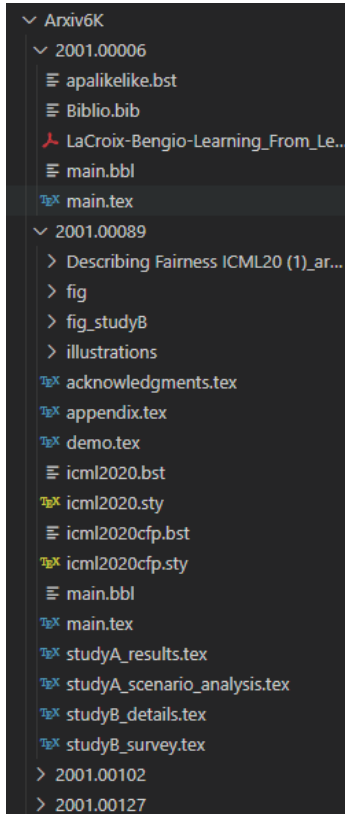
□ 实现难点

如何提取论文文本

最终正确地进行文本的提取和句子的划分，获得了27253个docx文档，

在这些文档中进行文字的筛选去杂，获得了1679622个句子

将这些句子作为论文句子数据集。





□ 实现难点

如何使用bert生成向量

解决聚类 and 语义搜索的一种常见方法是将每个句

子映射到一个向量空间，使得语义相似的句子很接近。

通常获得句子向量的方法有两种：

1. 计算所有Token输出向量的平均值
2. 使用[CLS]位置输出的向量

但这些使用bert和roberta的传统方法都有两个缺

点: 1. 准确性不够 2. 速度过慢

甚至在某些时候都不如w2v

```
bert.py
from transformers import AdamW, get_linear_schedule_with_warmup, BertTokenizer, BertModel
import torch

device = torch.device("cuda:0")

tokenizer = BertTokenizer.from_pretrained("/home/dou/replearn/transformers_models/bert")
query = "i love beijing renmin university of china."
bert_model = BertModel.from_pretrained("/home/dou/replearn/transformers_models/bert")
bert_model.to(device)
bert_model = torch.nn.DataParallel(bert_model)
train_data = tokenizer(query, return_tensors="pt")
result = bert_model.forward(**train_data)
hidden_state = result['last_hidden_state']

print(hidden_state.size())
print(result['last_hidden_state'][0][0].size())
print(result['last_hidden_state'][0][0])
```



□ 实现难点

如何使用bert生成向量

然而，UKP的研究员实验发现，在文本相似度（STS）任务上，使用上述两种方法得到的效果却差强人意，即使是Glove向量也明显优于朴素的BERT句子embeddings。

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	76.69	71.22

通过我自己的测试也发现，若直接使用bert或者roberta，都不能获得较好的结果



□ 实现难点

如何使用bert生成向量

为了获得更好的效果，我决定选择 Sbert，即**Sentence-BERT**

通过阅读论文《Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks》，我了解了一种针对于句向量的bert方法。

sentence-bert预训练的BERT进行修改：使用Siamese and Triplet Network（孪生网络和三胞胎网络）生成具有语义的句子Embedding向量。语义相近的句子，其Embedding向量距离就比较近，从而可以使用余弦相似度、曼哈顿距离、欧氏距离等找出语义相似的句子。SBERT在保证准确性的同时，可将上述提到BERT/RoBERTa的65小时降低到5秒（计算余弦相似度大概0.01秒）。这样SBERT可以完成某些新的特定任务，比如聚类、基于语义的信息检索等



□ 实现难点

Sbert模型简单介绍

SBERT在BERT/Roberta的输出结果上增加了一个Pooling操作(MEAN策略)，从而生成一个固定维度的句子Embedding。

SBERT对于bert进行了fine-tune，它采用了**孪生网络**和**三胞胎网络**来更新参数，以达到生成的句子向量更具语义信息

孪生网络是共享参数的两个神经网络，它衡量两个输入的差异程度。将两个输入分别送入两个神经网络，得到其在新空间的representation，然后通过Loss Function来计算它们的差异程度。

而三胞胎网络的输入是三个：一个正例+两个负例，或一个负例+两个正例。训练的目标仍然是让相同类别间的距离尽可能小，不同类别间的距离尽可能大。

文中实验了下面几种机构和目标函数



□ 实现难点

Sbert模型简单介绍

Classification Objective Function(分类目标函数)

针对分类问题，作者将向量 $u, v, |u - v|$ 三个向量拼接在一起，然后乘以一个权重参数 $W_t \in \mathbb{R}^{(3n \times k)}$ ，其中 n 表示向量的维度， k 表示label的数量

损失函数为CrossEntropyLoss

$$o = \text{softmax}(W_t[u; v; |u - v|])$$

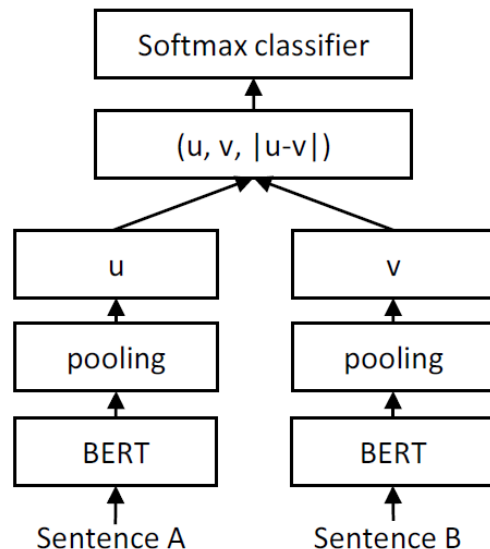


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).



□ 实现难点

Sbert模型简单介绍

Regression Objective Function(回归目标函数)

两个句子embedding向量 u, v 的余弦相似度
计算结构如右所示，损失函数为MAE (mean squared error)

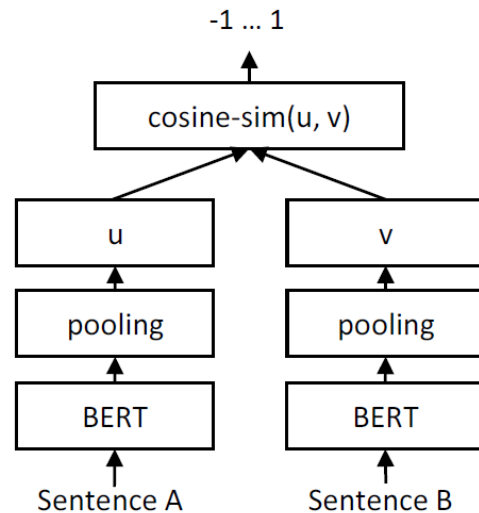


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.



□实现难点

Sbert模型简单介绍

Triplet Objective Function(三重目标函数)

给定一个主句a，一个正面句子p和一个负面句子n，三元组损失调整网络，使得a和p之间的距离尽可能小，a和n之间的距离尽可能大。

数学上，期望最小化以下损失函数：

$$\max(||s_a - s_p|| - ||s_a - s_n|| + \epsilon, 0)$$

其中，s_x表示句子x的embedding，||表示距离，边缘参数表示两者距离差至少为 ϵ 。在实验中，使用欧式距离作为距离度量， ϵ 设置为1



□实现难点

Sbert模型简单介绍

模型训练细节

作者训练时结合了SNLI (Stanford Natural Language Inference) 和Multi-Genre NLI两种数据集。SNLI有570,000个人工标注的句子对，标签分别为矛盾，蕴含，中立三种；MultiNLI是SNLI的升级版，格式和标签都一样，有430,000个句子对，主要是一系列口语和书面语文本

实验时，作者使用类别为3的softmax分类目标函数对SBERT进行fine-tune，batch_size=16，Adam优化器，learning_rate=2e-5



□ 实现难点

Faiss库提速

获得了大量句子的向量后，我们需要支持快速的查询某条query所对应的最相似的k条句子。

这时我们用到Faiss库去提速。

Faiss由Facebook AI Research开发，是一个用于相似性搜索和密集向量聚类的高性能库，支持十亿级别向量的搜索，是目前最为成熟的近似近邻搜索库。它包含多种搜索任意大小向量集的算法，以及用于算法评估和参数调整的支持代码。Faiss用C++编写，并提供与Numpy完美衔接的Python接口。除此以外，对一些核心算法提供了GPU实现。



□ 实现难点

Faiss库提速

单纯地使用faiss.IndexFlatL2的话，速度会太慢。

为了扩展到非常大的数据集，Faiss提供了基于产品量化器的有损压缩来压缩存储的向量的变体。压缩的方法基于乘积量化。即faiss.IndexIVFPQ

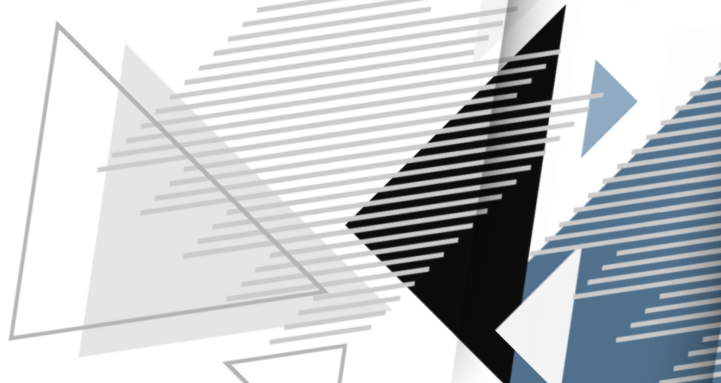
使用faiss.IndexIVFPQ进行加速，可以使得每次查询从10s减少到4s

```
quantizer = faiss.IndexFlatL2(d)
nlist = 100
index = faiss.IndexIVFPQ(quantizer, d, nlist, 4, 8)
index.train(xb)
|
print(index.is_trained)
index.add(xb)
print(index.ntotal)
index.nprobe = 5
```

成果展示

(Achievement display)

04





我使用Python Flask与semantic，实现了一个My AI helperd的网页可视化

用户可以向百度翻译那样输入一条英文语句，并选择需要返回多少条相似的论文文本

在经过4s左右的处理后，用户可以获得其数据

查询结果是按照相似度排名的论文文本，同时还会获得该文本和查询文本的相似程度与论文编号。



AI Helper Search


主页

Welcome to AI HelperSearch

搜索

数量: 3 ▾



 AI Helper Search [主页](#)

Welcome to AI HelperSearch

搜索

数量:

computer.

▼ 文件信息

排名

1

论文编号

848557

相似度

1.3689582653024424

▶ 文件信息



Welcome to AI Helper Search

The result is good.

搜索

数量: 3 ▼

Right: Results.

▸ 文件信息

We show the results in .

▸ 文件信息

This is the extent of the next result.

▸ 文件信息

We show these results in .

▸ 文件信息



后续还想做一个新的功能，即模仿百度的搜索，做一个输入框的实时下拉搜索结果刷新，随着用户的输入。

通过几天的学习，目前实现了两种方法：

1. 全局刷新

使用<meta http-equiv="refresh">进行定时定向url的全局刷新，但这样对用户体验可能不好。这个页面跳转的方法好处在于不需要JS调用，直接在html文件头里加入即可。

2. 局部刷新

使用ajax请求，再专门写一个可局部刷新的(下拉)表单，进行局部刷新。缺点是需要写一个JS调用和局部刷新表单，比较困难。

由于技术和时间问题，这两种方法在项目中的运用还不够成熟，有一些bug，故暂且不附在代码文件中。



谢谢