# 结题报告

### 1. 课题背景

随着信息化发展，通知更多借助网络渠道。学校目前使用的飞书 App 仍存在重要通知被淹没、通知对象针对性不强、通知本身信息冗杂等问题。大多数同学需要在群消息中反复寻找、查看同一条通知，时间利用效率低。项目设计并实现了一个通知内容管理 App，实现学校通知分类、要点捕捉与简化（文本摘要）、日程安排表生成四项功能，希望服务于学院通知发布工作。

### 2. 课题研究内容与方法

#### 研究内容

本项目通过对 “文本挖掘”的研究，利用相关算法将学院以大段文本形式呈现、信息糅合一体的通知抽象成一个个简单标签，可以实现对标签的分类，并且查询到相关的日期。此外，我们还实现了“文本摘要”，可以将大段冗余的通知总结成一段精简的通知

#### 实施方案

（其实我们是先有了后端，再有的前端的，但是报告顺序以及答辩顺序，我们认为可以先展示前端）

##### 第一阶段：基础学习

学习了相关知识：python 的基本用法，学习了基于 pytorch 的模型搭建与调试，学习了 transformers 的调试方法，学习了 “迁移学习” 的 机器学习方法，学习了 css，javascript 语法，学习了 flask 框架、ajax 开发前端，学习了如何使用 SQLAlchemy。学习了 docker 部署等技巧。我们还学习了组员的协作，通过 git 进行版本的管理。

##### 第三阶段：前端开发

前端开发：我们使用了非常简单易用的 LayUI。Layui 是一套免费的开源 Web UI 组件库，我们在 flask 中的 jinja2 模板中调用了 LayUI。我们通过 SQLAlchemy，创建了通知（notice）的 ORM 模型，这能让我们简单且安全的与本地数据库进行交互。

下面是 server，也就是我们这个项目的主体的目录结构：

(base) ╭─wangfiox@localhost ~/Documents/freshman\_project ‹main\*›  
╰─➤ tree  
.  
├── app.py  
├── blueprints  
│   └── notice.py  
├── config.py  
├── data\_utils.py  
├── exts.py  
├── models.py  
├── model  
│   ├── config.json  
│   ├── pytorch\_model.bin  
│   ├── special\_tokens\_map.json  
│   ├── tokenizer\_config.json  
│   └── vocab.txt  
├── statics  
│   ├── image（文件夹，jinja2 模板中一些图片）  
│   ├── js（文件夹，里面有jQuery，被layui、templates/\*依赖）  
│   └── layui（文件夹，web ui 组建库）  
├── templates  
│   ├── add.html  
│   ├── base.html  
│   ├── calendar.html  
│   ├── search.html  
│   └── categories.html  
├── tokenizers\_pegasus.py

下面我将对这个目录中的每个文件进行介绍（根据拓扑顺序）：

* templates 文件夹里面存放的就是我们的 jinja2 模板，json 数据传递通过 ajax 实现
  + base.html相当于是我们其他 html 的底板，里面是整个网页的主题，可以在其他的 templates html 中看到有{% extends "base.html" %}这句话，也可以理解成是继承的关系
  + add.html就是添加通知，并且能够与我们的 server 进行交互（上图！）
  + calender.html就是用到了 layUI 的日历组件，日期与对应的 通知（上图！）
  + categories.html就是我们可以看到分类与分类标签的地方（上图！）
  + search.html就是搜索页面
* exts.py只进行了一个简单的功能，打开数据库
* config.py是与数据库交流的必要配置，帐号，密码，端口等。
* models.py并不是人工智能的模型，而是与数据库交流的 ORM 模型
* data\_utils.py是 fengshen 模型库处理数据的辅助文件，被tokenizer\_pegasus.py依赖。这里直接放到了项目中，一个是环境的问题，还有一个是网络的问题，即使挂了代理，有时候还是莫名奇妙的会说是下载失败（，尽管我环境中确实有 fengshen。（fengshen 是 idea-ccnl 研究院针对中文训练的一系列大模型）。
* tokenizers\_pegasus.py是 fengshen 模型库的 tokenizers。这个文件被notice.py依赖，在文本摘要的时候，用来将输入的文本编码。pegasus就是一个专门用来处理文本摘要的模型。
* static只是静态文件，简而言之就是：资源包。
* blueprints/notices.py这个就是用来处理通知的，里面有文本摘要，模糊搜索，添加通知，文本分类的函数。
* model这个文件夹里面就存放着：字典vocab.txt，超参数tokenizer\_config.json，special\_tokens\_map.json，config.json，以及训练好的模型。
* app.py是top\_module，能路由网页等，启动网页，依赖config.py，exts.py，notice.py，templates

看了上面的介绍，可以看出，我们的最核心的文件是notice.py，下面一览notice.py的全貌

###### 1. 输入文本，输出标签

def predict\_label(text):  
 # 对文本进行编码  
 inputs = tokenizer\_classify(text, return\_tensors="pt", truncation=True, padding='max\_length', max\_length=128)  
  
 # 将输入移到模型所在的设备上  
 inputs = {key: val.to(model\_classify.device) for key, val in inputs.items()}  
  
 # 使用模型进行预测，这里不会进行训练  
 with torch.no\_grad():  
 outputs = model\_classify(\*\*inputs)  
 logits = outputs.logits  
 predicted\_label\_id = logits.argmax(-1).item()  
  
 # 获取预测的类别名  
 predicted\_label = id\_to\_label[predicted\_label\_id]  
  
 return predicted\_label

###### 2. 文本摘要

def summarize\_text(text, max\_length=128, num\_beams=4, length\_penalty=2.0, max\_length\_output=150):  
 """  
 用输入的模型和分词器生成输入文本的摘要。  
 参数：  
 - text (str): 要生成摘要的输入文本。  
 """  
  
 inputs = tokenizer\_summarize(text, max\_length=max\_length,  
 truncation=True, return\_tensors="pt")  
  
 # beam search 生成摘要  
 summary\_ids = model\_summarize.generate(  
 inputs["input\_ids"],  
 num\_beams=num\_beams,  
 length\_penalty=length\_penalty, # 惩罚系数是2.0  
 max\_length=max\_length\_output,  
 no\_repeat\_ngram\_size=3 # 生成文本时避免重复的n元组的大小  
 )  
  
 return tokenizer\_summarize.batch\_decode(summary\_ids, skip\_special\_tokens=True, clean\_up\_tokenization\_spaces=False)[0]

###### 3. 正则搜索日期

def find\_date(text, patterns):  
 """  
 使用正则表达式从文本中查找日期  
  
 Args:  
 text (str): 输入文本  
 patterns (list of str): 日期的正则表达式列表  
  
 Returns:  
 str: 找到的第一个日期字符串；如果未找到，则为None  
 """  
 for pattern in patterns:  
 # 通过正则表达式找日期  
 match = re.search(pattern, text)  
 if match:  
 return match.group(0)  
 return None

###### 4. 添加新通知（手动指定）

# 添加新通知  
@bp.route("/Add", methods=["POST"])  
def AddNotice():  
 data = request.json  
  
 # 分类  
 new\_notice = NotificationModel(  
 title=data.get("title"),  
 content=data.get("content"),  
 date=datetime.datetime.strptime(  
 data.get("date"), "%Y年%m月%d日") if data.get("date") else None,  
 summary=data.get("summary"),  
 creator\_id=data.get("creator\_id")  
 )  
  
  
 db.session.add(new\_notice)  
 db.session.commit()  
  
 data = {  
 "title": data.get("title"),  
 "content": data.get("content"),  
 "date": new\_notice.date,  
 "summary": data.get("summary"),  
 "creator\_id": data.get("creator\_id")  
 }  
  
 return jsonify({"code": 200, "data":data , "msg": "Notice added successfully"})

###### 5. 模糊搜索

@bp.route("/Search", methods=["GET"])  
def SearchNotice():  
 keyword = request.args.get("keyword")  
 title = request.args.get("title")  
 date = request.args.get("date")  
  
 query = NotificationModel.query  
  
 # 模糊查询  
 if keyword:  
 keyword\_search = or\_(  
 NotificationModel.content.like(f"%{keyword}%"),  
 NotificationModel.summary.like(f"%{keyword}%")  
 )  
 query = query.filter(keyword\_search)  
  
 # 根据类别查询  
 if title:  
 query = query.filter\_by(title=title)  
  
 # 根据日期查询  
 if date:  
 target\_date = datetime.datetime.strptime(date, "%Y-%m-%d")  
 query = query.filter\_by(date=target\_date)  
 else:  
 # 按created\_date降序排列  
 query = query.order\_by(NotificationModel.created\_date.desc())  
  
 results = query.all()  
  
 # 搜索到的结果  
 notices = [  
 {  
 "id": notice.id,  
 "title": notice.title,  
 "content": notice.content,  
 "date": notice.date.strftime("%Y-%m-%d %H:%M:%S") if notice.date else None,  
 "summary": notice.summary,  
 "creator\_id": notice.creator\_id,  
 "created\_date": notice.created\_date.strftime("%Y-%m-%d %H:%M:%S")  
 }  
 for notice in results  
 ]  
  
 return jsonify({"code": 200, "data": notices})

###### 6. 自动生成标签，日期，摘要

@bp.route("/AutoAdd", methods=["POST"])  
def AutoAddNotice():  
 data = request.json  
  
 content = data.get("content")  
 # 去除前后空格  
 content = content.strip()  
  
 # 自动生成摘要  
 summary = summarize\_text(content)  
  
 # 自动生成类别标签  
 title = predict\_label(content)  
  
 # 自动提取日期  
 patterns = [  
 r"\d{4}年\d{1,2}月\d{1,2}日", # YYYY年MM月DD日  
 ]  
 date\_str = find\_date(content, patterns)  
 date = datetime.datetime.strptime(  
 date\_str, "%Y年%m月%d日") if date\_str else None  
  
 # new\_notice = NotificationModel(  
 # title=title,  
 # content=content,  
 # date=date,  
 # summary=summary,  
 # creator\_id=data.get("creator\_id")  
 # )  
  
 data = {  
 "title": title,  
 "content": content,  
 "date": date\_str,  
 "summary": summary,  
 "creator\_id": data.get("creator\_id")  
 }  
  
 # db.session.add(new\_notice)  
 # db.session.commit()  
  
 return jsonify({"code": 200, "data":data, "msg": "Notice added successfully with auto features"})

##### 第二阶段：后端开发

这就要看我们server之外的文件们了

(base) ╭─wangfiox@localhost ~/Documents/freshman\_project/utils ‹main\*›  
╰─➤ tree  
.  
├── classification  
│   ├── classification.ipynb  
│   └── logs（文件夹，训练的日志）  
├── data  
│   ├── x\_通知搜集.md （人工搜集的400条通知，有几个文件）  
│   ├── combined\_data.csv  
│   ├── csv\_into\_db.ipynb  
│   ├── dates\_combined\_data.csv  
│   ├── dates\_combined\_data.ipynb  
│   ├── dates\_combined\_data\_summarized.csv  
│   ├── extract\_content\_to\_csv.ipynb  
│   └── test.csv  
└── summary  
 ├── data\_utils.py  
 ├── summary.ipynb  
 └── tokenizers\_pegasus.py

* summary，我们这里是直接 “拿来主义”，直接调用了IDEA-CCNL/Randeng-Pegasus-523M-Summary-Chinese-V1，并在 jupyter 中测试了效果
  + summary.ipynb，测试 “拿来主义” 的模型的效果
  + tokenizers\_pegasus.py 与 data\_utils.py 在上文中介绍了令人尴尬的效果（
* data里面有一些处理的脚本，我们是搜集了几个文件，一个是将文件合并，一个是将文本总结，一个是将文件导入数据库
* classification才是重头戏！！！下面介绍！！！

###### 1. 使用预训练模型

# 使用预训练模型  
model\_name = 'IDEA-CCNL/Erlangshen-Roberta-110M-Sentiment'  
tokenizer = BertTokenizer.from\_pretrained(model\_name)  
model = BertForSequenceClassification.from\_pretrained(model\_name) # 预训练模型

###### 2. 测试 二分类 结果

texta = '鲸鱼是哺乳动物，所有哺乳动物都是恒温动物'  
textb = '鲸鱼也是恒温动物'  
output = model(torch.tensor([tokenizer.encode(texta, textb)]))  
print(torch.nn.functional.softmax(output.logits, dim=-1)) # 测试一下

OUTPUT:

tensor([[0.0645, 0.9355]], grad\_fn=<SoftmaxBackward0>)

可以看到，第一句话模型认为的并不是很正确（但是实际上应该是对的）；第二句话模型认为是对的。

###### 3. 准备数据

加载数据，将文本 tokenize，将数据集划分为：训练集，测试集。因为预训练模型本身就比较大，就没有进行网格搜索，交叉验证，因此没有验证集。

from datasets import load\_dataset, Features, Value  
  
label\_to\_id = { # 分类  
 "升学": 0,  
 "志愿": 1,  
 "教务": 2,  
 "思政": 3,  
 "心理": 4,  
 "灾害": 5,  
 "作业与考试": 6,  
 "竞赛与机会": 7,  
 "企业参观与就业": 8,  
 "生活": 9,  
 "重要通知": 10,  
 "垃圾与乐子": 11,  
}  
  
# 将label\_to\_id进行反转  
id\_to\_label = {value: key for key, value in label\_to\_id.items()}  
print(id\_to\_label) # 测试一下  
  
# 明确地定义CSV数据的特征描述  
features = Features({  
 '类别': Value('string'),  
 '通知内容': Value('string')  
})  
  
# 使用提供的特征描述加载数据集  
dataset = load\_dataset('csv', data\_files='../data/combined\_data.csv', features=features)  
print(dataset) # 预览数据集  
  
# 数据处理  
def preprocess\_function(batch):  
 # 对通知内容进行分词，并返回结果  
 encoding = tokenizer(batch['通知内容'], truncation=True, padding='max\_length', max\_length=128) # 分词，截断，填充  
 encoding["labels"] = [label\_to\_id[label] for label in batch["类别"]] # 使用label\_to\_id将类别名转换为ID  
 return encoding  
  
  
# 使用map函数进行预处理  
encoded\_dataset = dataset['train'].map(preprocess\_function, batched=True).train\_test\_split(test\_size=0.05)  
  
train\_dataset = encoded\_dataset['train']  
test\_dataset = encoded\_dataset['test']  
  
# 输出训练集和测试集的大小  
print(len(train\_dataset))  
print(len(test\_dataset))  
  
# 打印第一个样本的内容，带换行符  
print(train\_dataset[0])  
# 输出：  
# {'类别': '生活', '通知内容': '各位同学@所有人 今天晚上收到多名同学反馈在教学楼、活动中心和宿舍楼附近发现卖笔的人员，请大家不要轻信和购买，保护好自身财产安全', 'input\_ids': [101, 1392, 855, 1398, 2110, 137, 2792, 3300, 782, 791, 1921, 3241, 677, 3119, 1168, 1914, 1399, 1398, 2110, 1353, 7668, 1762, 3136, 2110, 3517, 510, 3833, 1220, 704, 2552, 1469, 2162, 5650, 3517, 7353, 6818, 1355, 4385, 1297, 5011, 4638, 782, 1447, 8024, 6435, 1920, 2157, 679, 6206, 6768, 928, 1469, 6579, 743, 8024, 924, 2844, 1962, 5632, 6716, 6568, 772, 2128, 1059, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'token\_type\_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attention\_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'labels': 9}

###### 4. 将 二分类 转化为 12 分类

# 修改模型输出  
num\_labels = len(label\_to\_id)  
model = BertForSequenceClassification.from\_pretrained(model\_name, num\_labels=num\_labels, ignore\_mismatched\_sizes=True)  
  
# 打印模型的最后一层，验证是12分类  
print(model.classifier)

OUTPUT:

Linear(in\_features=768, out\_features=12, bias=True)

###### 5. 激动人心的 trainer.train()

from transformers import Trainer, TrainingArguments  
  
# 定义训练参数  
training\_args = TrainingArguments(  
 output\_dir='./results',  
 evaluation\_strategy="steps",  
 eval\_steps=10,  
 per\_device\_train\_batch\_size=64,  
 per\_device\_eval\_batch\_size=128,  
 num\_train\_epochs=3,  
 save\_steps=50,  
 logging\_steps=20,  
 learning\_rate=2e-5,  
 weight\_decay=0.01,  
 logging\_dir='./logs',  
 load\_best\_model\_at\_end=True,  
)  
  
# 创建Trainer对象  
trainer = Trainer(  
 model=model,  
 args=training\_args,  
 train\_dataset=train\_dataset,  
 eval\_dataset=test\_dataset,  
 compute\_metrics=None, # 如果你需要在验证时计算评估指标，请提供一个compute\_metrics函数  
)  
  
# 开始训练  
trainer.train()

### 3. 研究结果

小组合作，做出了一个 web app，非常有成就感。

### 4. 创新点

没啥创新的，不过是用了一些很简单的东西，然后把他们缝合在了一起。

### 5. 结束语

人工智能好玩。好好学习，以后还要深入的玩人工智能。

我们将这次大一立项开源到了 github 上：https://github.com/KINGFIOX/freshman\_project。 欢迎在座的各位给个 star

### 6. 参考文献

[1] https://huggingface.co/IDEA-CCNL/Randeng-Pegasus-523M-Summary-Chinese 文本摘要 （pegasus）

[2] https://huggingface.co/IDEA-CCNL/Erlangshen-Roberta-110M-Sentiment 文本分类（二分类）

[3] https://flask.palletsprojects.com/en/3.0.x/ flask 文档

[4] https://arxiv.org/abs/1912.08777 PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization（pegasus 文本摘要预训练模型）

[5] https://arxiv.org/abs/1907.11692 RoBERTa: A Robustly Optimized BERT Pretraining Approach （预训练的 BERT 模型，用于分类）

[6] https://github.com/IDEA-CCNL/Fengshenbang-LM 封神榜大模型

[7] https://en.wikipedia.org/wiki/Ajax\_(programming) ajax wiki

[8] https://api.jquery.com/category/ajax/ ajax jquery

[9] https://revealjs.com reveal.js

[10] https://www.sqlalchemy.org sqlalchemy

[11] https://pytorch.org pytorch

[12] https://layui.dev LayUI

[13] https://www.python.org python

[14] https://developer.mozilla.org/en-US/docs/Web/CSS css

[15] https://developer.mozilla.org/en-US/docs/Web/javascript javascript

[16] https://nodejs.org/en/download node.js（npm）

[17] https://www.opensuse.org opensuse

[18] https://arxiv.org/abs/1706.03762 attention is all you need