航空领域对话模型开发

1 研究背景

ChatGPT 的成功问世是生成式人工智能技术创新的里程碑，是人类在人工智能领域研发中所取得的一项具有重大意义的历史性突破，也掀起了一股探索自然语言处理技术的热潮。本项目旨在开发适用于航空领域的对话模型，将生成式语言模型应用于飞机客舱环境下与乘客的互动，代替真人空乘与乘客的部分沟通工作，为其减轻工作负担。

2 语料收集

对于生成式语言模型而言，制作语料数据集十分关键。为了使收集到的数据能够有效训练模型，对于语料的收集我们制定了以下规则：

1. 对话场景：在飞机飞行场景下（不包括起飞前、落地之后）；

2. 对话主体：从乘客的角度来向空乘 AI 提出问题，每次对话由乘客发起，由空乘 AI 结束。由于是训练一个多轮对话模型，所以每次对话不少于 2 轮（4 段话）。

3. 对话限制：由于与乘客对话的是 AI，所以应该考虑到乘客提出的一些需求，AI 是没办法自行帮助乘客的，比如乘客需要一杯水，AI 应该说“马上为您准备一杯水”，而不是“马上帮您拿水”，或“给，您的水”这类语句，AI 没办法向真人空乘那样直接给乘客递东西或引导乘客去飞机上的厕所等地，只能告知乘客会为他准备什么东西（实际上是通知后台真人空乘来准备），或者某个地点在飞机上的哪个位置。

4. 数据格式：一句话一行，一次对话结束后空一行，无需标注说话者。例如：

    你好！

    您好！有什么可以帮您的吗？

    能否给我一杯热咖啡？

    当然可以，请问是黑咖啡还是白咖啡？

    黑咖啡，谢谢！

    好的，马上为您准备一杯黑咖啡。

    好的，谢谢！

不客气，很高兴为您服务，还有什么需要吗？

没有了。

飞机怎么不太平稳啊？

不用担心，飞行过程中颠簸是正常现象。

会有什么危险吗？

不用担心，我们的飞机是很安全的。

好吧。

有什么需要请随时提出，祝您旅途愉快。

机上温度有点冷，给我拿一条毛毯吧。

好的请稍等，还有别的需求吗？

再帮我拿个眼罩吧。

好的，马上为您准备毛毯和眼罩。

谢谢。

不客气，很高兴为您服务。

小组中 11 名成员参与了语料收集工作，共收集到 610 次对话数据。

3 模型训练流程

我们在 GPT-2 的基础上使用我们制作的语料数据集对其进行微调，使用了一块 GeForce RTX 3090 显卡进行训练。训练代码和过程主要参考以下链接：

https://huggingface.co/shibing624/gpt2-dialogbot-base-chinese

接下来介绍一下训练流程：

3.1 下载及安装

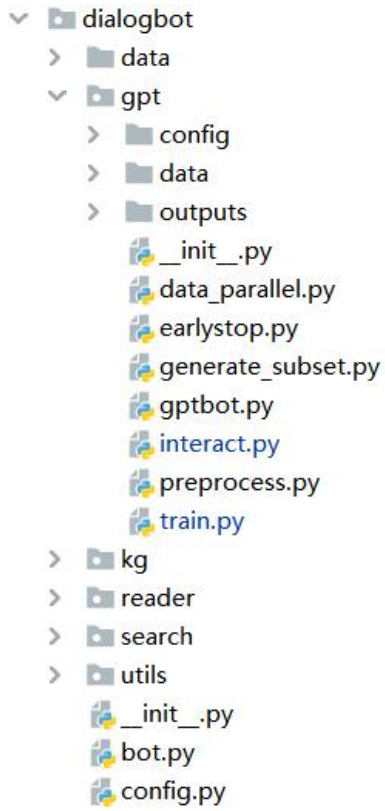
首先从 GitHub 上下载对话模型 dialogbot 的代码，然后根据要求安装 pytorch、transformers、numpy、loguru、jieba 等 python 包，其中 python 的版本需大于等于 3.7。

```
git clone https://github.com/shibing624/dialogbot.git
pip install transformers
pip install numpy
pip install loguru
```

```
pip install jieba
pip insatll gensim
pip install lxml
pip install tqdm
pip install bs4
pip install nltk
```

3.2 项目目录

项目目录如下，主要关注的是 dialogbot 文件夹下 gpt 中的内容，其中 config 文件夹存放配置文件，data 文件夹存放训练数据以及经过序列化处理后的数据，ouputs 文件夹存放训练过程中每个 epoch 得到的模型（格式为.bin）以及训练结束之后获得的模型。



训练过程中涉及到的文件主要是 preprocess.py、train.py、interact.py 这三个。其中 preprocess.py 用于将收集的语料数据集进行序列化处理，生成能够被模型读取的文件格式，train.py 用于训练模型，控制模型训练时迭代次数（epochs）、批次大小（batch_size）、学习率（lr）等参数，interact.py 用于模型的预测，用户可以与训练得到的对话 AI 进行交互，检验模型的训练效果。

```
parser.add_argument('--device', default='0', type=str, help='设置使用哪些显卡')
parser.add_argument('--no_cuda', action='store_true', help='不使用GPU进行训练')
parser.add_argument('--model_config', default='config/config.json', type=str, help='设置模型参数')
parser.add_argument('--train_path', default='data/train.pkl', type=str, help='训练集路径')
parser.add_argument('--max_len', default=150, type=int, help='训练时，输入数据的最大长度')
parser.add_argument('--log_path', default='data/train.log', type=str, help='训练日志存放位置')
parser.add_argument('--log', default=True, help="是否记录日志")
parser.add_argument('--ignore_index', default=-100, type=int, help='对于ignore_index的label token不计算梯度')
parser.add_argument('--epochs', default=100, type=int, help='训练的最大轮次')
parser.add_argument('--batch_size', default=16, type=int, help='训练的batch size')
parser.add_argument('--gpu0_bsz', default=10, type=int, help='0号卡的batch size')
parser.add_argument('--lr', default=2.6e-5, type=float, help='学习率')
parser.add_argument('--eps', default=1.0e-09, type=float, help='衰减率')
parser.add_argument('--log_step', default=10, type=int, help='多少步汇报一次loss')
parser.add_argument('--gradient_accumulation_steps', default=4, type=int, help='梯度积累')
parser.add_argument('--max_grad_norm', default=2.0, type=float)
parser.add_argument('--save_model_path', default='./outputs/', type=str, help='模型输出路径')
parser.add_argument('--pretrained_model', default='uer/gpt2-distil-chinese-cluecorpussmall', type=str,
                    help='预训练的模型的路径')
parser.add_argument('--num_workers', type=int, default=0, help="dataloader加载数据时使用的线程数量")
parser.add_argument('--patience', type=int, default=0, help="用于early stopping,设为0时,不进行early stopping")
parser.add_argument('--warmup_steps_rate', type=float, default=0.05, help='warm up步数')
parser.add_argument('--val_rate', type=float, default=0.1, help='验证集大小')
```

3.3 数据预处理

首先需要对收集到的语料数据进行预处理，预处理过程包括 tokenize 和序列化。

通过 tokenize 完成两件事：

1. 切分语料，以[CLS]标志语料开头，以[SEP]划分说话对象
2. 完成字到字典 id（tokenize_id）的转变

然后进行序列化处理，保存为 train.pkl。

preprocess.py 部分代码如下：

```
    # 初始化tokenizer
    tokenizer = BertTokenizerFast.from_pretrained(args.pretrained_model)
    sep_id = tokenizer.sep_token_id
    cls_id = tokenizer.cls_token_id
    logger.info("preprocessing data,data path:{}, save path:{}".format(args.train_path, ar
gs.save_path))
    with open(args.train_path, 'rb') as f:  # 读取训练数据集
        data = f.read().decode("utf-8")
    # 需要区分linux和windows环境下的换行符
    if "\r\n" in data:
        train_data = data.split("\r\n\r\n")
    else:
        train_data = data.split("\n\n")
    # 开始进行tokenize,保存所有的对话数据,每条数据的格式为:
"[CLS]utterance1[SEP]utterance2[SEP]utterance3[SEP]"
    dialogue_len = []  # 记录所有对话tokenize之后的长度，用于统计中位数与均值
    dialogue_list = []
    for index, dialogue in enumerate(tqdm(train_data)):
```

```
        if "\r\n" in data:
            utterances = dialogue.split("\r\n")
        else:
            utterances = dialogue.split("\n")
        input_ids = [cls_id]  # 每个dialogue 以[CLS]开头
        for utterance in utterances:
            input_ids += tokenizer.encode(utterance, add_special_tokens=False)
            input_ids.append(sep_id)  # 每个utterance 之后添加[SEP]，表示utterance 结束
        dialogue_len.append(len(input_ids))
        dialogue_list.append(input_ids)
    len_mean = np.mean(dialogue_len)
    len_median = np.median(dialogue_len)
    len_max = np.max(dialogue_len)
    with open(args.save_path, "wb") as f:
        pickle.dump(dialogue_list, f)
```

运行 preprocess.py 命令如下：

```
cd dialogbot/gpt
python preprocess.py --train_path data/train.txt --save_path data/train.pkl
```

最终获得序列化的数据集，其中包含了 610 次对话。

```
2023-04-10 08:14:49.166 | INFO     | __main__:preprocess:31 - preprocessing data,data path:data/train.txt, save path:data/train.pkl
2023-04-10 08:14:49.171 | INFO     | __main__:preprocess:42 - there are 610 dialogue in dataset
100%|██████████| 610/610 [00:00<00:00, 1657.27it/s]
2023-04-10 08:14:49.548 | INFO     | __main__:preprocess:65 - finish preprocessing data,the result is stored in data/train.pkl
2023-04-10 08:14:49.549 | INFO     | __main__:preprocess:66 - mean of dialogue len:128.02622950819674,median of dialogue len:100.0,max len:459
```

3.4 模型训练

使用预处理后的数据，对模型进行自回归训练，最佳模型保存 outputs 文件夹中，train.py 部分代码如下：

```
    train_dataloader = DataLoader(train_dataset, batch_size=args.batch_size, shuffle=True, num_workers=args.num_workers,
                                  collate_fn=collate_fn, drop_last=True)
    validate_dataloader = DataLoader(validate_dataset, batch_size=args.batch_size, shuffle=True, num_workers=args.num_workers, collate_fn=collate_fn, drop_last=True)
    early_stopping = EarlyStopping(args.patience, verbose=True, save_path=args.save_model_path)
    t_total = len(train_dataloader) // args.gradient_accumulation_steps * args.epochs
    optimizer = transformers.AdamW(model.parameters(), lr=args.lr, eps=args.eps)
    num_warmup_steps = int(t_total * args.warmup_steps_rate)
    scheduler = transformers.get_linear_schedule_with_warmup(optimizer, num_warmup_steps=num_warmup_steps, num_training_steps=t_total)
    logger.info('starting training')
    # 用于记录每个epoch 训练和验证的loss
    train_losses, validate_losses = [], []
    # 记录验证集的最小loss
```

```
    best_val_loss = 10000
    # 开始训练
    train_iterator = trange(int(args.epochs), desc='Epoch', mininterval=0)
    for epoch in train_iterator:
        # ========== train ========== #
        train_loss = train_epoch( model=model, train_dataloader=train_dataloader, optimizer=optimizer, scheduler=scheduler, epoch=epoch, args=args)
        train_losses.append(train_loss)
        # ========== validate ========== #
        validate_loss = validate_epoch( model=model, validate_dataloader=validate_dataloader, epoch=epoch, args=args)
        validate_losses.append(validate_loss)
        if validate_loss < best_val_loss: # 保存当前困惑度最低的模型
            best_val_loss = validate_loss
            model_path = join(args.save_model_path, 'min_ppl_model'.format(epoch + 1))
            if not os.path.exists(model_path):
                os.mkdir(model_path)
            model_to_save = model.module if hasattr(model, 'module') else model
            model_to_save.save_pretrained(model_path)
            tokenizer.save_pretrained(model_path)
```

将迭代次数设为 50，批次大小设为 8，以 9：1 的比例划分训练集和验证集，使用单卡进行训练。运行 train.py 命令如下：

```
python train.py --epochs 50 --batch_size 8
```

运行截图如下：

```
2023-04-10 10:15:53.868 | INFO     | __main__:load_dataset:60 - loading training dataset and validating dataset
2023-04-10 10:15:53.873 | INFO     | __main__:load_dataset:68 - data size: 610, val_num: 61
/home/nianwuluo@corp.sse.tongji.edu.cn/.conda/envs/pytorch/lib/python3.7/site-packages/transformers/optimization.py:395: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this warning
  FutureWarning,
2023-04-10 10:15:53.891 | INFO     | __main__:train:215 - starting training
Epochs 1/50, Batchs 60/68, Training Loss:    2.4139, Training Acc    0.4760: 100%|| 68/68 [00:02<00:00, 25.14it/s]
2023-04-10 10:15:56.600 | INFO     | __main__:train_epoch:151 - epoch 1: loss 2.5784706052611854, predict_acc 0.4581425363766553| 68/68 [00:02<00:00, 26.82it/s]
2023-04-10 10:15:56.600 | INFO     | __main__:train_epoch:154 - saving model for epoch 1
2023-04-10 10:15:57.498 | INFO     | __main__:train_epoch:160 - epoch 1 finished
2023-04-10 10:15:57.499 | INFO     | __main__:train_epoch:162 - time for one epoch: 0:00:03.604441
2023-04-10 10:15:57.499 | INFO     | __main__:validate_epoch:168 - start validating
2023-04-10 10:15:57.580 | INFO     | __main__:validate_epoch:190 - validate epoch 1: loss 2.296402488436018
2023-04-10 10:15:57.580 | INFO     | __main__:validate_epoch:192 - time for validating one epoch: 0:00:00.080619
2023-04-10 10:15:57.580 | INFO     | __main__:train:236 - saving current best model for epoch 1
Epochs 2/50, Batchs 60/68, Training Loss:    1.8242, Training Acc    0.5889: 100%|| 68/68 [00:02<00:00, 26.44it/s]
2023-04-10 10:16:00.981 | INFO     | __main__:train_epoch:151 - epoch 2: loss 2.1225541356731865, predict_acc 0.5365118752215526| 68/68 [00:02<00:00, 24.91it/s]
2023-04-10 10:16:00.981 | INFO     | __main__:train_epoch:154 - saving model for epoch 2
2023-04-10 10:16:01.881 | INFO     | __main__:train_epoch:160 - epoch 2 finished
2023-04-10 10:16:01.881 | INFO     | __main__:train_epoch:162 - time for one epoch: 0:00:03.472583
2023-04-10 10:16:01.882 | INFO     | __main__:validate_epoch:168 - start validating
2023-04-10 10:16:01.962 | INFO     | __main__:validate_epoch:190 - validate epoch 2: loss 1.897267971720014 3
2023-04-10 10:16:01.963 | INFO     | __main__:validate_epoch:192 - time for validating one epoch: 0:00:00.080444
2023-04-10 10:16:01.963 | INFO     | __main__:train:236 - saving current best model for epoch 2
Epochs 3/50, Batchs 60/68, Training Loss:    1.7605, Training Acc    0.5958: 100%|| 68/68 [00:02<00:00, 26.89it/s]
2023-04-10 10:16:05.339 | INFO     | __main__:train_epoch:151 - epoch 3: loss 1.8016921124037575, predict_acc 0.5818470775207487| 68/68 [00:02<00:00, 26.01it/s]
2023-04-10 10:16:05.340 | INFO     | __main__:train_epoch:154 - saving model for epoch 3
2023-04-10 10:16:06.232 | INFO     | __main__:train_epoch:160 - epoch 3 finished
```

4 用户与模型的交互

最后是检验模型训练的效果，运行 interact.py，启动“空乘 AI”，然后乘客就可以和空乘 AI 进行对话了。

interact.py 部分代码如下：

```
    text_ids = self.tokenizer.encode(query, add_special_tokens=False)
    self.history.append(text_ids)
    input_ids = [self.tokenizer.cls_token_id]  # 每个input以[CLS]为开头
    if use_history:
        for history_id, history_utr in enumerate(self.history[-self.max_history_len:]):
            input_ids.extend(history_utr)
            input_ids.append(self.tokenizer.sep_token_id)
    else:
        input_ids.extend(text_ids)
        input_ids.append(self.tokenizer.sep_token_id)
    input_ids = torch.tensor(input_ids).long().to(self.device)
    input_ids = input_ids.unsqueeze(0)
    response = []  # 根据context，生成的response
    for _ in range(self.max_len): # 最多生成max_len个token
        outputs = self.model(input_ids=input_ids)
        logits = outputs.logits
        next_token_logits = logits[0, -1, :]
        # 对于已生成的结果generated中的每个token添加一个重复惩罚项，降低其生成概率
        for id in set(response):
            next_token_logits[id] /= self.repetition_penalty
        next_token_logits = next_token_logits / self.temperature
        # 对于[UNK]的概率设为无穷小，也就是说模型的预测结果不可能是[UNK]这个token
        next_token_logits[self.tokenizer.convert_tokens_to_ids('[UNK]')] = -float('Inf')
        filtered_logits = top_k_top_p_filtering(next_token_logits, top_k=self.topk, top_p=
    self.topp)
        # torch.multinomial表示从候选集合中无放回地进行抽取num_samples个元素，权重越高，抽到的
    几率越高，返回元素的下标
        next_token = torch.multinomial(F.softmax(filtered_logits, dim=-1), num_samples=1)
        if next_token == self.tokenizer.sep_token_id:  # 遇到[SEP]则表明response生成结束
            break
        response.append(next_token.item())
        input_ids = torch.cat((input_ids, next_token.unsqueeze(0)), dim=1)
     self.history.append(response)
     response_tokens = self.tokenizer.convert_ids_to_tokens(response)
```

运行 interact.py 命令如下：

```
    python interact.py
```

以下展示了测试时乘客与空乘 AI 的几次对话：

```
开始和空乘AI对话，输入q以退出
乘客:您好，可以给我一杯橙汁吗？
空乘AI:当然可以。请问您需要哪种橙汁？
乘客:就是鲜榨橙汁。
空乘AI:好的，马上为您准备。
乘客:谢谢！
空乘AI:不客气，很高兴为您服务。
乘客:q

开始和空乘AI对话，输入q以退出
乘客:您好，请问洗手间在哪儿？
空乘AI:您好，洗手间在客舱的前部和尾部。请问你还有其他的问题吗？
乘客:我知道了，谢谢！
空乘AI:不用谢，有什么需要尽管告诉我。
乘客:q

开始和空乘AI对话，输入q以退出
乘客:您好，请问飞机上有哪些饮料？
空乘AI:您好，每种饮料都包括无糖、无糖和无糖等多种饮料。
乘客:我问的是饮料有哪些种类。
空乘AI:我想知道有哪些饮料可以提供给您？
乘客:我在问你有哪些饮料。
空乘AI:我想知道有哪些饮料可以提供？
乘客:我不知道，所以问你。
空乘AI:好的，谢谢。
乘客:给我一杯咖啡吧.
空乘AI:好的，马上为您准备。请问还有其他需要帮忙的服务吗？
乘客:没有了。
空乘AI:好的。
乘客:q

开始和空乘AI对话，输入q以退出
乘客:飞机上的正餐有哪些？
空乘AI:您好，飞行期间，我们航班提供免费的餐食，包括主食和小食，您想要哪一种？
乘客:我想要主食，主食有哪些？
空乘AI:我想要素食餐。
乘客:我不吃素食，我想要牛排。
空乘AI:好的，请您享用。
乘客:q
```

通过这些对话测试可以发现不是所有提问都能得到明确的回答，有时候空乘 AI 会顾左右而言他，胡乱回答，甚至反过来提问乘客，这说明目前训练得到的对话模型没有办法准确理解乘客的提问，造成这种情况的可能有以下原因：

1. 缺少训练数据。目前用于训练的数据集只收集到 610 次对话，要想训练一个效果较好的语言模型，这样的数据量是远远不够的。

2. 数据集中的信息不够可靠。存在矛盾，对于模型有一定的误导，比如有些对话给出

的信息是“飞机上有可乐”，而另外一些对话给出的信息却是“飞机上没有可乐”，这导致在训练时模型可能会混淆这两者。

3. 用于微调的模型不太适合。本项目的对话模型基于 GPT-2，是属于聊天型对话，不以解决实际问题为目标。而在航空领域，一般是乘客向空乘提出问题，空乘给出解决方案，所以该场景可能更适合采用另外两种对话模型，也就是任务型对话和问答型对话，这两类主要是用于解决用户问题的。