# 华中农业大学

## 《医学健康数据分析与挖掘》

## 课程论文

# 基于 BERT 的 20NewsGroups 数据集新闻分类实验

姓名：姚昕智
班级：生物信息学 19 级研究生
专业：生物信息学
学号：2019317110022
日期：2021 年 5 月 25 日

# 基于 BERT 的 20NewsGroups 数据集新闻分类实验

姚昕智 [1]

[1]Hubei Key Lab of Agricultural Bioinformatics, College of Informatics, Huazhong Agricultural University, Wuhan, Hubei Province, P.R. China

**摘要**

在这个信息爆炸的时代，人类如何快速准确的从海量的新闻报道中获取自己最关注的，最具有价值的领域的新闻已然成为了一个不可忽视的重要问题。该实验通过利用主流的自然语言处理模型 BERT[1] 通过在 20NewsGroups 数据集上进行微调，得到了一个具有较好表现的新闻分类器，最好表现的分类器获得 0.98 的 F1 值。并且该模型通过比较不同版本的 BERT 模型，分析了在新闻分类任务上，影响不同 BERT 表现可能的因素以及解决办法。改工作所有的代码已经通过 GitHub 进行公开，`https://github.com/YaoXinZhi/BERT-for-20NewsGroups`。

**Keywords: 新闻分类，自然语言处理，语言模型，BERT**

## 1 论文目的及意义

处于这个信息大爆炸的时代，每分每秒都有海量的信息数据产生，而人类想要实时，精确的获取自己所关心的时事，了解这个时间上正在发生着什么，就无疑需要依赖各类媒体所报道的新闻。而每天在各个领域产生的新闻无疑是海量的，而如何第一时间精准的得到自己想要关注，想要获取的新闻，就成了必须要面对的一个问题。

而准确传递新闻信息的，除了视频影像，其中最重要的就是海量的自然文本，而由于自然文本的特殊性与复杂性，往往又很难快速的通过人工处理对海量的文本新闻信息进行分类。因此，本文旨在用近年来主流的自然语言处理模型，BERT(Bidirectional Encoder Representations from Transformers) 利用 20NewsGroups 训练集对模型进行微调，从而使模型具备能够对新闻进行分类的能力，从而能够快速的提取出海量新闻中自身所需要的新闻文本，对于在这个信息爆炸的时代的人类信息获取具有重要的意义。

## 2 数据来源及结构

**数据来源** 该实验所采用的数据集来源于加州大学欧文分校机器学习数据库（UC Irvine Machine Learning Repository）所收集的 20 NewsGroups 数据集`http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html`，其中包含从 20 个新闻类别准组中包含的两万条消息文本，该数据集的原始收集人为卡耐基梅隆大学计算机科学学院的 Tom Mitchell 教授。其中这 20 个新闻类别来源于 7 个大类别，分别是计算机 (comp)，音乐 (misc)，娱乐 (rec)，科学 (sci)，社会 (soc)，聊天 (talk) 和其他 (alt)，其中每个大类又分别分为了若干小类别，分别是 alt.atheism, comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.windows.x, misc.forsale, rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey, sci.crypt, sci.electronics, sci.med, sci.space, soc.religion.christian, talk.politics.guns, talk.politics.mideast, talk.politics.misc, talk.religion.misc。其中大约 4% 的文章包含多个类别。这些文章是典型的新闻消息，因

此有标题，包括主题行、签名文件和其他文章的引用部分。

**数据结构** 每个新闻组存储在一个子目录中，每一篇文章存储为一个单独的文件。通过预处理，对每个文件中特殊符号进行替换，并删除新闻的来源，署名，邮箱等无用信息，并且按照 7 比 3 的比例对数据集进行分割，最后得到训练文件和测试文件，分别储存在 data/train.txt 和 data/test.txt 文件中，其中每个文件包含两列，第一列是新闻文本本身，第二列是该文本所对于的类别编号。

对于类别文件，储存在 data/label.txt 文件中，文件包含一列，对应该数据集合中的 20 个类别，其中每一个类别所在行数，对应该类别的编号，例如第一行的类别 alt.atheism，对应编号 0。

**数据预处理** 因为受制于 BERT 模型 512 的序列输入最大长度，所以对原始数据的长度进行了统计，并且根据统计分析发现大部分句子长度小于 500，而只有少量的句子长度过长，由图1和图2可以看出，因此我们对长度大于 500 的句子进行了删除，从而确保模型的学习质量。

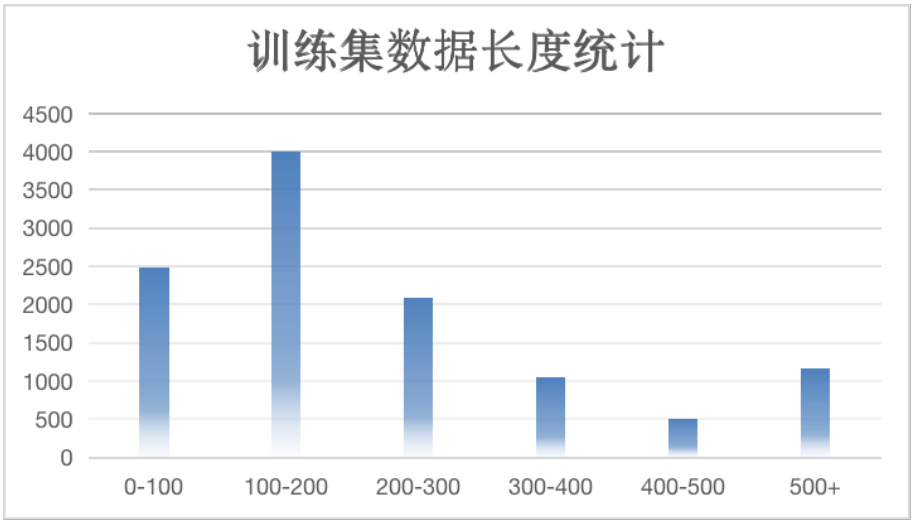


图 1: 训练集数据长度统计

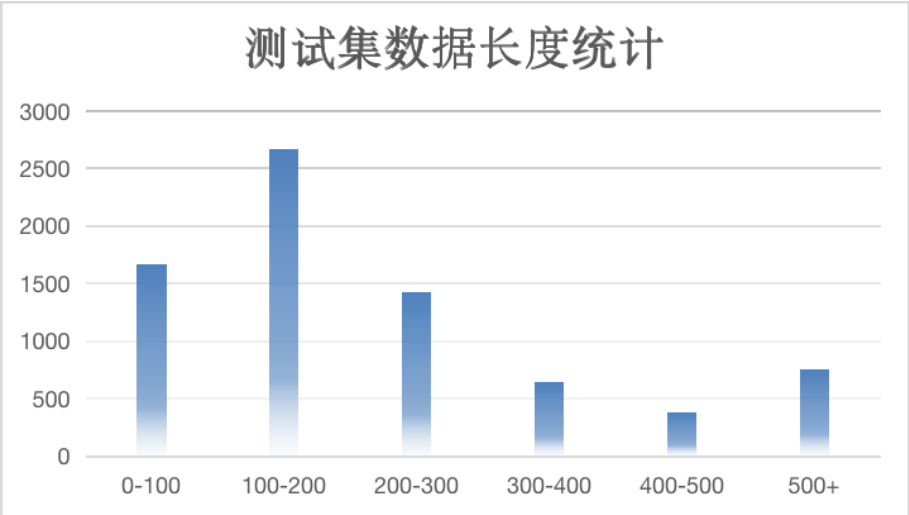


图 2: 测试集数据长度统计

# 3 方法

该实验所采用的方法是 Google 团队在 2019 年提出的 BERT 模型 [1]，该模型基于 Transformers 模型 [2] 的编码器部分进行调整。由于模型参数量巨大，一般 GPU 难以对整个模型从头进行训练，所以在 Google 团队 2019 年公布的 BERT-base-cased 版本中，该模型已经在 BooksCorpus 和 English Wikipedia 两个数据集上进行了预训练，而预训练的任务分为两个，一个是通过对句子中的单词进行随机遮掩，再让模型对所遮掩的单词进行预测。另一个预训练任务是，同时将两个句子作为模型的输入，让模型判断所输入的两个句子是否是前后句关系。同个两个预训练任务，BERT 模型已经能够对常见词语，句式能够有很好的编码能力，并且能够充分学习语义信息。而训练好的 BERT 模型，很方便能够通过在自己的数据集上进行微调，从而让 BERT 模型更好的适应自己的任务。

我们的模型正是使用 Google 所提供的 BERT-base-cased 和 BERT-large-cased 版本，两个模型基本结构相同，但是 large 版本相对于 bert 版本有更大的词汇表，同时具有更深的网络结构，通常可以对语义信息进行更好的学习，但是也会面临更难训练的问题。我们在模型最后加上一个全链接层，让模型能够进行 20NewsGroups 数据集上的分类任务。同时为了更好的让模型学习文本中关键单词的特征，该模型在加载数据的时候对数据中的特殊字符进行了替换，同时在训练时采取 Dropout，Larning Rate Warm-up 等多种训练技巧。而模型损失，采用最常用的多分类损失，及交叉熵损失。

通过参数寻优，获得最优表现的模型参数如下，其中随机种子为 26，最大序列长度为 512，Dropout 概率为 0.3，使用优化器为 Adam，最佳学习率为 2e-5，学习率衰减率为 1e-8。

全部模型框架通过 pyTorch 进行实现，其中利用 Transformers，一个封装了多个预训练模型的 python 包进行 BERT 模型及预训练参数的调用。

# 4 实验结果及分析

通过在 20NewsGroups 数据集上进行模型微调，模型随时正常收敛，通过图3可以发现，模型在训练到第 20 个循环的时候，模型以及基本收敛，并且以及非常小。最优表现如下图所示。


图 3: 训练损失收敛图

| 方法 | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| BERT-base-cased | 0.9791 | 0.9892 | 0.9795 | 0.9843 |
| BERT-Large-cased | 0.9397 | 0.9938 | 0.9221 | 0.9566 |

表 1: BERT 模型在 20NewsGroups 数据集上的表现

### 4.1 总结与讨论

通过对1中结果进行分析，发现 BERT-base 版本的模型，相对于 BERT-Large 版本的模型在 Accuracy，Recall，F1 上获得了更好的表现，而 Large 版本的模型，在 Precision 上则得到了更好的表现。这一结果也较为符合实验预期结果，因为 base 版本，因为具有较少的参数量，往往更容易微调，从而让模型适应自己的数据集，但是 Large 版本参数量过大，反而较难拟合自己的数据集 [3]，但是由于较大的词汇表，往往更容易学习到一种基于模板的匹配规则，从而在见过的模板上预测得到正确的结果，而对于新闻数据，同一个领域的新闻文本通常在用词，或者句式上较为相同，这也就一定程度上解释了为什么 Large 版本的 BERT 模型会在准确性上得到更好的效果。

虽然简单的微调 BERT 模型，在 20NewsGroup 数据集上已经取得了较好的效果，但是依旧还有进一步提升的空间，在之后的实验上，我们计划通过 1. 加大训练的轮次，减少每个 Batch 的数据量，同时减小初始学习率，从而让 Large 版本的 BERT 更好的拟合我们的数据集。2. 因为 20NewsGroups 的数据集本身比较小，难以让模型学习到充足的语义信息，所以我们计划通过将模型在更大的新闻分类数据集上进行微调，之后再通过迁移学习的方法运用到 20NewsGroups 数据集上，从而得到更好的效果 [**?**]。

## 参考文献


[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. arXiv: 1706.03762.

[3] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A Primer in BERTology: What we know about how BERT works. *arXiv:2002.12327 [cs]*, November 2020. arXiv: 2002.12327.


## A 附录

### A.1 模型结构代码

```

import torch
import torch.nn as nn
import torch.nn.functional as f
from transformers import BertConfig
from transformers import BertPreTrainedModel

class BertClassifier(BertPreTrainedModel):

    def __init__(self, bert_config: BertConfig, bert, hidden_size, label_number, dropout):
        super(BertClassifier, self).__init__(bert_config)
        self.bert = bert
```

```
        self.fc = nn.Linear(hidden_size, label_number)
        self.dropout = nn.Dropout(dropout)

        self.CrossEntropyLoss = nn.CrossEntropyLoss()
        self.Softmax = nn.Softmax(dim=1)


    def forward(self, encoded_input, labels=None):

        bert_output = self.bert(**encoded_input)

        cls_state = bert_output['last_hidden_state'][:, 0, :]

        predicted = self.dropout(cls_state)

        predicted = self.fc(predicted)
        if labels is not None:
            loss = self.CrossEntropyLoss(predicted.to('cpu'), labels)
            return loss
        else:
            predicted = self.Softmax(predicted)
            predicted_label = torch.argmax(predicted, dim=1)
            return predicted_label
```

## A.2 训练函数代码

```

def _train(paras):

    logger = logging.getLogger(__name__)
    if paras.save_log_file:
        logging.basicConfig(format='%(asctime)s - %(levelname)s - %(name)s -   %(message)s',
                            datefmt='%m/%d/%Y %H:%M:%S',
                            level=paras.logging_level,
                            filename=f'{paras.log_save_path}/{paras.mode}-{paras.train_log_file}',
                            filemode='w')
    else:
        logging.basicConfig(format='%(asctime)s - %(levelname)s - %(name)s -   %(message)s',
                            datefmt='%m/%d/%Y %H:%M:%S',
                            level=paras.logging_level,)

    device = 'cuda' if torch.cuda.is_available() and (not paras.use_cpu) else 'cpu'
    if device == 'cuda':
        logging.info('----- cuda for training -----')
    else:
        logging.info('----- cpu for training -----')


    logger.info(f'Loading model: {paras.model_name}')

    bert_classifier, tokenizer = _load_pre_trained_model(paras)
    bert_classifier.to(device)

    train_dataset = News_Dataset(paras, 'train')
    train_dataloader = DataLoader(train_dataset, batch_size=paras.batch_size,
                                  shuffle=paras.shuffle, drop_last=paras.drop_last)
    label_to_index = train_dataset.label_to_index
    index_to_label = train_dataset.index_to_label
```

```
33
34
35      test_dataset = News_Dataset(paras, 'test')
36      test_dataloader = DataLoader(test_dataset, batch_size=paras.batch_size,
37                                   shuffle=paras.shuffle, drop_last=paras.drop_last)
38
39      if paras.optimizer.lower() == 'adam':
40          logger.info('Loading Adam optimizer.')
41          optimizer = torch.optim.Adam(bert_classifier.parameters(), lr=paras.learning_rate)
42      elif paras.optimizer.lower() == 'adamw':
43          logger.info('Loading AdamW optimizer.')
44          no_decay = [ 'bias', 'LayerNorm.weight' ]
45          optimizer_grouped_parameters = [
46              {'params': [ p for n, p in bert_classifier.named_parameters() if not any(nd in n for nd in
                                                                                  no_decay) ],
47               'weight_decay': 0.01},
48              {'params': [ p for n, p in bert_classifier.named_parameters() if any(nd in n for nd in
                                                                                  no_decay) ],
49               'weight_decay': 0.0}
50          ]
51          optimizer = AdamW(optimizer_grouped_parameters, lr=paras.learning_rate,
52                            eps=args.adam_epsilon)
53      else:
54          logger.warning(f'optimizer must be "Adam" or "AdamW", but got {paras.optimizer}.')
55          logger.info('Loading Adam optimizer.')
56          optimizer = torch.optim.Adam(bert_classifier.parameters(),
57                                       lr=paras.learning_rate)
58
59      logger.info('Training Start.')
60      best_eval = {'acc': 0, 'precision': 0, 'recall': 0, 'f1': 0, 'loss': 0}
61      for epoch in range(paras.num_train_epochs):
62          epoch_loss = 0
63          bert_classifier.train()
64          for step, batch in enumerate(train_dataloader):
65              optimizer.zero_grad()
66
67              batch_data, batch_label = batch
68              encoded_data = tokenizer(batch_data,
69                                       padding=True,
70                                       truncation=True,
71                                       return_tensors='pt',
72                                       max_length=paras.max_sequence_length)
73              encoded_data.to(device)
74
75              label_tensor = batch_label_to_idx(batch_label, label_to_index)
76              label_tensor.to(device)
77
78              loss = bert_classifier.forward(encoded_data, label_tensor)
79
80              epoch_loss += loss_to_int(loss)
81
82              logging.info(f'epoch: {epoch}, step: {step}, loss: {loss:.4f}')
83
84              loss.backward()
85              optimizer.step()
86
87          epoch_loss = epoch_loss / len(train_dataloader)
88
89          logging.info('Evaluating.')
```

```
        acc, precision, recall, f1 = evaluation(bert_classifier, tokenizer, test_dataloader,
                                                paras.max_sequence_length, label_to_index)

        logging.info(f'Epoch: {epoch}, Epoch-Average Loss: {epoch_loss:.4f}')
        logger.info(f'Accuracy: {acc:.4f}, Precision: {precision:.4f}, '
                    f'Recall: {recall:.4f}, F1-score: {f1:.4f}')

        if best_eval['loss'] == 0 or f1 > best_eval['f1']:
            best_eval['loss'] = epoch_loss
            best_eval['acc'] = acc
            best_eval['precision'] = precision
            best_eval['recall'] = recall
            best_eval['f1'] = f1
            _save_model(paras.model_save_path, bert_classifier, tokenizer, paras.model_save_name,
                        paras.config_save_name)


            with open(f'{paras.log_save_path}/{paras.checkpoint_file}', 'w') as wf:
                wf.write(f'Save time: \{time.strftime("\%Y-\%m-\%d \%H:\%M:\%S", time.localtime())\}\n')
                wf.write(f'Best F1-score: {best_eval["f1"]:.4f}\n')
                wf.write(f'Precision: {best_eval["precision"]:.4f}\n')
                wf.write(f'Recall: {best_eval["recall"]:.4f}\n')
                wf.write(f'Accuracy: {best_eval["acc"]:.4f}\n')
                wf.write(f'Epoch-Average Loss: {best_eval["loss"]:.4f}\n')

            logger.info(f'Updated model, best F1-score: {best_eval["f1"]:.4f}\n')

    logger.info(f'Train complete, Best F1-score: {best_eval["f1"]:.4f}.')
```

## A.3 评估函数代码

```


def evaluation(model: BertModel, tokenizer: BertTokenizer, dataloader: DataLoader,
               max_sequence_length: int, label_to_index: dict, device='cuda'):
    model.eval()
    total_pred_label = []
    total_ture_label = []
    with torch.no_grad():
        for batch in tqdm(dataloader):

            batch_data, batch_label = batch
            label_idx = batch_label_to_idx(batch_label, label_to_index, False)
            encoded_data = tokenizer(batch_data,
                                     padding=True,
                                     truncation=True,
                                     return_tensors='pt',
                                     max_length=max_sequence_length)
            encoded_data.to(device)
            predict_label = model(encoded_data)
            predict_label = tensor_to_list(predict_label.to('cpu'))

            total_ture_label.extend(label_idx)
            total_pred_label.extend(predict_label)

        precision, recall, acc, f1 = fuck_metric(total_ture_label, total_pred_label)

    return acc, precision, recall, f1
```