文本分类实验报告

- 10215501403 沈桐乐

实验背景

文本分类(Text Classification)是统计学和自然语言处理领域一个经典任务。

在本次实验当中，我们需要完成的是**序列文本多分类任务**(Sequence Multilabel Text Classification)，其问题描述为给定一段自然语言文本$s \in S$，预测一个对应的文本种类$k \in \{0,1\}^n$，而本次题设当中的文本种类为正交的种类编号，隐去了种类的token信息

该任务现在也是衡量模型语言推理能力的一个重要标准，其准确度的提升衡量人类设计的模型对于整个语言系统的理解程度，其中比较知名的有GLUE Benchmark，和上海AI实验室提出的CLUE Benchmark（主要衡量中文模型）。从经典的Naive Bayes，逻辑回归等统计模型，到Word Embedding与深度学习的出现，再到Pretraining的提出，和先当今的LLM与组合模型，多模态的研究，每一次准确度的提升都意味着NLP领域有了新的爆发式进展。在这些LeaderBoard当中，最近模型的表现已经完全了打败人类自己的文本分类能力。

总而言之，这是一项非常经典且有意义的任务，而我们需要做的是回顾整个文本分类的发展过程，并且测试其中具有代表意义的模型，尝试理解模型背后的设计思路和效果解释。

实验设计

本次实验一共设计了四个板块，按照时间进行排序，已完成前面三种

经典统计方法

计划使用文本的编码为TF-IDF方法，之所以不使用Word2Vec，在后文当中会有解释

- 包含：Naive Bayes，Softmax，决策树，SVM

后三种是题目要求测试的模型，贝叶斯分类器是一种效率极快且相对准确的经典分类器，因此也加入测试

经典深度学习方法

计划使用Word2Vec（GloVe）的静态文本编码

- 包含：MLP，CNN，RNN

MLP是经典的分类器，TextCNN在分类任务上具有良好的效果，而RNN是NLP中的经典网络

基于Pretraining和Finetuning的方法

计划使用各训练模型的动态文本编码

- 包含：BERT，XLNet

选取BERT的原因是其具有划时代意义，借鉴ELMo和双向Transformer的BERT Encoder有非常好的泛用性，包括现在的多模态设计（比如ViBert等）都是在此基础上进行改进

选取XLNet则是因为其在Benchmark上是排名很前的Single Model

LLM-Finetuning

超大参数模型，厂家提供API，输入Token和标签进行文本分类FineTuning，但是由于其价格和使用难度（没有美国信用卡），在本次实验中并没有实现

实验过程

设计了一个简易文本分类测试框架dmac，将测试，超参数，模型，Embedding和io等杂项模块化，便于扩展和调整参数。

验证部分使用的是5-fold Cross Validation，测试验证集的准确率和F1值

模型部分调用scikit-learn, pytorch和HuggingFace封装好的接口进行编写，dmac/model模块只是对于第三方模型接口进行上层封装，主要工作为更改超参数，导入word vector，调整部分模型结构，统一测试接口等

最后测试集使用所有模型中，验证**准确率最高**的模型进行标签预测

模型测试的补充和想法

- 关于经典ML模型

采用Softmax Regression而不是多次二分的Logistic Regression，是因为Softmax函数来源于多标签下logistic函数的扩展形式，具有多分类能力

决策树的损失函数采取的经典交叉熵损失，深度不会设置很高去强行满足准确率

NaiveBayes采用Laplace平滑，防止文本（稀疏矩阵）的后验一直为0的情况

SVM采纳的是RBF Kernel，把文本向量映射到高维空间，rbf的优点是快，效果比较好，同样可以测试经典的Gaussian核，开销变大但是效果差不多

所有ML模型均不能采用单独的Word2Vec Embedding映射到矩阵空间$R^{n\times E}$，只能将句子映射到线性空间R^n上，是因为这些模型都是R^n内的分类模型，而如果映射到矩阵空间，则超出这些方法的使用范围，所以采用出现统计的OneHot编码，或者词频统计的TF-IDF方法。一种可行的Embedding设计是把矩阵空间展开为$R\{n\times E\}\times 1$，但是这样会导致模型过大和稀疏。第二种设计是把所有的Embedding取均值，但是这样就丢失了单个单词的信息。

- 关于经典DL模型

三个模型都采用了Dropout方法防止过拟合

CNN借鉴的是TextCNN的设计，包含一层卷积层和全连接层

RNN模型为2 Layer Stacked RNN/LSTM/GRU Encoder+Attention Output的设计，是一种经典的设计模式

- 关于Finetuning

所有微调的实验有单块显卡的显存的上线（24G），因此选取的是参数量较小的bert-base和xlnet-base

实验结果和分析

- 实验配置

ML模型在CPU上运行，DL模型在GPU上运行

Cpu :MAC m1（本机）

GPU: NVIDIA GTX 4090 24G(AutoDL) / Tesla T4(colab)

对于模型本身不会在报告中讲解，分析的部分主要关注于如何解释测试出来的结果。选取的都是常见模型，其实现原理可以参考相应论文

统计学模型

- 使用句子编码为：TF-IDF

模型名称	计算单元	验证准确率	验证F1值	训练时间	预测时间
Random	CPU	0.0994	0.0996	0.00s	0.00s
SVM	CPU	0.9460	0.9459	20.93s	2.71s
Softmax	CPU	0.9445	0.9444	1.64s	0.00s
DecisionTree	CPU	0.7377	0.7392	1.65s	0.00s
NaiveBayes	CPU	0.9460	0.9459	0.00s	0.00s

- 使用句子编码为：OneHot

模型名称	计算单元	验证准确率	验证F1值	训练时间	预测时间
NaiveBayes	CPU	0.9140	0.9132	2.09s	0.03s

我们对每一个模型对结果进行分析：

- Random

随机生成的标签，作为整个测试工作的BaseLine，防止在测试的时候其他模型出错

- SVM

分类任务当中效果极好的分类模型，在多分类问题中，SVM的优化问题描述为K个自己Label和其他Label的支持向量机。训练时间相对更长，是因为优化目标是其对偶问题，而且使用RBF核把对偶问题中的坐标内积映射到Hilbert Space，从而在高维空间内学习分离超平面的参数。而预测时间较长是因为一共要跑10个分类器，获得所有分类器的投票结果，相比其他的单个模型会花更多的预测时间

- Softmax

Softmax作为Logistics Regression的扩展形式，其本身诞生于多分类问题，因此验证效果非常好。和SVM不同的是，逻辑回归对于标签群体的要求是**线性不可分的**，否则模型会在两个标签的"间隔"之间互相震荡导致无法收敛，当然在高维的句子编码中，我们可以假定不会出现这个问题。

- DecisionTree

决策树本身并不适用于多分类问题。用一句比较通俗的话来说，决策树设计的目标是如何区分标签，使得标签之间**区分度最大**，而不是如何区分这些标签，而原版的决策树模型是一种平面分类模型，对于一个特征F来说，如果我们认为F是在加序空间上，则决策树可以获得这个空间的最优分割，取决于损失函数，但是对于句向量来说，显然单个/多个特征的组合不具有狭义的可比性

- NaiveBayes

贝叶斯分类器是先当今仍然非常常用的分类器，常见于垃圾邮件的分类。和测试结果一致的是，Naive Bayes又快又准。其原因是因为我们认为模型的标签和某些特定的单词是有先验/后验的关系的。在这个数据集中，同一标签大概率会出现某些比较类似的单词，最经典的例子是认为垃圾邮件中大概率出现“SALE”的单词，因此标签和该单词的出现有很强的关联性。在这个数据集中也是类似的

然而，这些统计模型都是有准确率瓶颈的，其主要有以下几个原因：

1. 自然语言本身是很复杂的，而使用TF-IDF或者OneHot只能把这些语言压缩到概率空间$s \in [0,1]^n$，这个压缩的操作本身会损失大量信息，而且使结果具有局限性，比如TF-IDF在转换新句子的过程中，不会统计没有出现过的单词，而张量空间的表示方法不在这些模型的表示范围之内
2. 模型架构过于简单。所有的统计学模型都是有这样的工作形式：假定数据集满足某种条件，在这个条件下满足某种形式，以统计学和数学运算取得结果，并且证明得到某种性质，最后通过学习和预测的结果来判断这些假定是否是正确的。

   在NLP领域中，这样的假设往往和现实世界相比是相差甚远的。比如在NER，Machine Translation，Text Classification等任务中，统计学家的研究方法是找到现实世界的某种假定，但是先前一百多条假定的复杂模型仍然不能在这些下游任务中取得良好的结果。所以想要取得文本分类的更好结果，我们需要对这样的“假定-验证”的思想推翻重来

深度学习模型

- 传统深度学习模型，静态Word Embedding（glove-twitter-100），（MLP使用TF-IDF加速）

模型名称	计算单元	验证准确率	验证F1值	训练时间	预测时间
MLP(1 hidden layer)	CPU	0.9460	0.9459	27.10s	0.04s
CNN(1-layer Conv2d)	GPU(4090)	0.9243	0.9240	45.34s	0.11s
RNN(2-layer LSTM + Attention Output)	GPU(4090)	0.9116	0.9113	14.18s	0.12s

我们来对结果进行分析：

本阶段我们使用了静态Word Embedding讲文本映射到张量空间$R^{n \times E}$，而Word2Vec的发明使得单词本身在高维空间R^E中获得线性关系，不再是One-Hot Vector的超高维空间$\{0,1\}^N$上的单纯正交关系。进行了数据压缩的同时保留了主要信息，是Word2Vec的主要好处。

然而，MLP，CNN模型中忽略了一个重要的信息：位置向量。不仅单词之间有线性关系，句子与句子之间单词是有顺序规则的，而这些模型的输入忽略了这个规则，导致分类出现瓶颈。对于这个问题的解决方法有两种：一种是Positional Embedding，在Word Vectors输入模型之前在和位置向量编码结合，第二种是使用有向模型（比如RNN，ELMo等）在训练的时候顺序输入，所以对于这两个模型和有向方法的结合，是传统深度学习模型在文本任务上得高分的提升空间，比如RCNN，CNN+Positional Embedding+RNN等

- MLP，CNN

MLP相当于深层叠加神经元（neuron = Linear + Activation) 拟合。其设计模式能在小样本分类上取得非常好的结果，而CNN则压缩相邻的单词信息，通过Pooling进行采样，是首先在CV领域提出的一种设计。二维的卷积框架在句子分类中可以借用，由于叠加的层数为1，准确率并没有非常高。

- RNN

RNN模型在分类任务中不是一种常用模型，其主要处理的领域是序列化任务，比如Language Model，Machine Translation，时序预测等。尽管在实验中已经尝试了多种RNN的改进策略，比如LSTM Cell，Attention Output等，其效果仍然不够可观。原因在于其输出目标是最后一个Cell的输出结果(Attention是整个一层的加权结果)，为了模型的对齐，在预处理阶段我使用了Max-Length Padding的方法。而且题设当中给出的标签并非单词标签，而是数字标签。结果就是模型很难通过上一轮的输出来获得下一轮的结果（因为文本分类任务是Encoder Only的设计，Decoder相当于只有一个Cell，输出结果，很难通过顺序来获得足够多有效的信息）

RNN第二个瓶颈在于其没有提供足够多的学习样本。RNN模型具有一定程度的泛化能力，但是训练集非常小，导致无论如何调参和改进模型都无法取得更好的结果。

- 预训练模型微调

全部使用base参数量（防止显存溢出）

模型名称	计算单元	验证准确率	验证F1值	训练时间	预测时间
BERT	GPU(4090)	0.9625	0.9625	225.18s	7.47s
XLNet	GPU(T4)	0.9532	0.9541	1152.93s	2.71s

我们来对结果进行分析：

首先有必要解释一下预训练为什么能够取得极好的效果。Pretraining设计的思想是，我们给定一个无监督任务，比如Cloze，假设模型训练的最优解G和实际子任务的最优解G^*是非常接近的，在NLP则认为文本的理解方式都是近似的。而Finetuning在此基础上把G优化到当前任务的最优解G^*，如果假设成立，则Finetuning的结果会非常快。这种做法在大模型上尤其使用，从零开始训练参数的开销非常大，而下载参数微调的代价相对较小。

- BERT

BERT在Transformer Encoder下增加了位置学习方法（设计位置Mask的预训练任务），同时借鉴序列模型的设计方式，把句子开头的CLS作为输出结果，并且把transformer Encoder（多重Self Attention+LayerNorm+FFN）进行叠加，能够在多个任务甚至多模态中取得非常好的成果。在本次实验中，使用的是bert-base-uncased模型，Finetuning的收敛速度非常快，也取得了最好的准确率和F1值

- XLNet

XLNet在本次实验中并没有取得和论文一样比Bert更好的结果，其原因在于实际调优的过程中，模型的内存占用非常大。可能这个模型在更大的参数量下比BERT更优秀，但是由于实验条件的显存限制，只能下载较小的模型进行调优。

结论和展望

本次实验的结果基本和预期是一致的，

在条件允许的情况下，可以进一步测试LLM的微调结果，以及尝试一些经典模型的实现，比如ResNet，多层的CNN等。

在dmac测试框架中，也可以把第三方实现模型转换成自己实现的模型，从而进一步理解模型等结构和效果之间的关系。也可以引入更多，规模更大的测试用例放入测试框架中，使得测试的结果更加具有说服力。