舆情文本多分类与敏感性检测技术研究与实现

文本多分类技术研究与实现

Introduction

随着大数据时代的到来，数据的体量越来越大，内容也鱼龙混杂。如何从海量的数据中筛选出有价值的可以满足需求的特定数据，并使其发挥出本身的价值，成为了各行各业，尤其是互联网企业正在努力解决的关键问题。在这样的背景下，文本分类成为了解决上述问题的一项重要技术。

文本分类是指计算机将载有信息的文本映射到预先给定的某一类别或某几类别主题的过程。有效高效的文本分类方法,可以正确归纳文本主题，总结文章内容，充分利用数据的价值，进而开发出有针对性夫的个性化用户推荐系统,使其可以根据用户的个人兴趣来定位并推荐相关的文本资料，从而实现商业需求。这样的优势使得文本分类任务愈发受到各行各业，尤其是掌握大量数据的新闻媒体，电商等行业的青睐。在人工智能浪潮席卷全球的今天，文本分类技术已经被广泛地应用在文本审核、广告过滤、情感分析和反黄识别等NLP领域。

我们可以根据实际应用场景，将文本分类这一课题继续细分。根据类别的数目，可以分为二分类和多类分类；根据每条文本所属的类别个数，可分为单标签文本分类和多标签文本分类。本文主要研究的是基于BERT模型的单标签和多标签的文本分类。

最初的文本分类仅仅通过分析本文内容，抓取文本关键词，通过堆和树等数据结构进行简单的分类操作。比如基于规则的特征匹配，例如篮球映射到体育，电影映射到娱乐等。这样的方法虽然容易理解，但是存在诸多问题。首先，这个方法依赖专家系统，不同类别需要专门构建特征规则，费时费力。其次，运行效率低且准确率低，不能满足实际应用中对于文本分类准确率的要求。

近年来，随着人工智能的蓬勃兴起，文本分类这个课题越来越受到学者，从业人员和企业的关注。最初的解决方案是机器学习方法，如朴素贝叶斯分类，决策树，支持向量机（SVM）等。基本原理是首先，专家们利用领域专业知识，创建合适的输入特征，然后把训练数据输入模型，通过算法拟合出最优的，即错误最小化的模型作为输出，最后把文本作为输入，寻找文本中的某些特征，根据这些特征通过训练好的模型将文本分为不同的类别。这样的机器学习方法，在准确率上远高于人工分类的方法，但是仍有不足之处。首先，如何创建合适的输入特征，即特征工程，是机器学习方法的最大瓶颈。特征工程这一步骤需要大量的专业知识以及不断地尝试，取得突破极为困难。其次，受限于计算能力，训练时使用的数据量有限，导致模型结构比较简单，存在欠拟合的问题。假如使用更大量的数据，又会出现过拟合的问题。

为了解决以上的问题，计算机科学家使用更多层的神经网络，使神经网络自动通过每一层产生适当的特征，从而避免了特征工程的复杂，创造出许多深度学习方法如卷积神经网络（CNN），递归神经网络（RNN），长短期记忆模型（LSTM）等。这些方法主要利用了文本，图像，语音等数据的连续，稠密的特点，利用其局部相关性，自动获取文本的特征。本文采用的BERT（Bidirectional Encoder Representation from Transformers）模型，就是基于深度学习方法，利用多层的双向transformer的网络结构实现了自动获取文本特征的能力。

BERT框架把原来的机器学习过程分成两个阶段：预训练（pre-training）和微调（fine-tuning）。首先，利用Google拥有的强大数据计算能力和海量的训练数据，训练出一个预训练模型，然后使用者可以根据自身的需求对预训练模型进行微调。这样既保证了模型具有极高的准确率，又对于使用者的计算能力没有太高的要求，从而在自然语言处理的许多任务中表现出色。而这种提供一个供其它任务迁移学习的模型，根据任务微调或者固定之后作为特征提取器的二阶段训练模式也成为深度学习的潮流。

本文就是使用预训练+微调的模式，把BERT模型应用在文本分类任务中，在实现BERT模型具有的高准确率的基础上，把模型运用到实际的应用中，使模型发挥自身的价值，并结合第二部分介绍的敏感词检测功能，实现一个完整的高效的业务流程。

Related work

ELMo

2018年3月份之前，在自然语言理解领域，词表征模型的结构是每个词对应一个vector，对于多义词无能为力。2018年3月份提出的ELMo(Embeddings from Language Models)对此提出了一个较好的解决方案。不同于以往的一个词对应一个向量，预训练好的模型不再只是向量对应关系，而是一个训练好的模型。使用时，将一句话或一段话输入模型，模型会根据上下文来推断每个词对应的词向量。这样做之后明显的好处之一就是对于多义词，可以结合前后语境对多义词进行理解。比如appele，可以根据前后文语境理解为公司或水果。作者认为好的词表征模型应该同时兼顾两个问题：一是词语用法在语义和语法上的复杂特点；二是随着语言环境的改变，这些用法也应该随之改变。

ELMo是一种新型深度语境化词表征，可对词进行复杂特征(如句法和语义)和词在语言语境中的变化进行建模(即对多义词进行建模)。ELMo使用的是一个双向的LSTM语言模型，由一个前向和一个后向语言模型构成，目标函数就是取这两个方向语言模型的最大似然。众所周知，LSTM的工作原理为：将单词转换成word embedding，然后将上一时刻的输出/隐状态及第一步中的word embedding一并送入lstm，并得到输出及隐状态，最后将lstm的输出与上下文矩阵相乘，得到一个列向量，再将该列向量经过softmax归一化。这就是一个基本的前向LSTM模型。后向语言模型，跟前向语言模型类似，给定后文来预测前文。ELMo使用的双向lstm语言模型，论文中简称biLM，就是要最大化前向和后向的概率的似然概率的对数。

对于每个单词，对于L层的双向lstm语言模型，一共有2L+1个表征（前向L层的输出，后向L层的输出以及本身）。每层lstm输出是不一样的，针对每个任务每层的向量重要性也不一样，所以对于每层向量，我们加一个权重，将每层的向量与权重相乘之后累加，然后再乘以一个权重γ。这样就得到了ELMo向量。针对具体的自然语言理解任务，我们需要再次训练去得到的这个权重γ。

所以对于ELMo的运行流程，我们可以这样归纳：1产生预训练biLM模型。模型由两层bi-LSTM组成，之间用残差连接的方式连接起来。2在任务语料上微调上一步得到的biLM模型。3利用ELMo的word embedding来对任务进行训练。把它们作为输入加到已有的模型中，可以明显的提高原模型的表现。这种预训练+微调的方式也被证明是有效的。而最终模型的表现也证明了ELMo和双向LSTM模型的成功：ELMo能够同时区分语法和语义，能够在表示词语嵌入时考虑到上下文的信息，既消除了歧义，还能准确把握词性。

但是ELMo的局限性在于，LSTM模型提取特征的能力不如BERT使用的Transformer结构，这影响了双向语言模型的表现。

OpenAI GPT

一个对文本有效的抽象方法可以减轻NLP对有监督学习的依赖。目前大多数深度学习方法依靠大量的人工标注信息，这限制了在很多领域的应用。此外，即使在可获得相当大的监督语料情况下，以无监督学习的方式学到的表示也可以提供显着的性能提升。

GPT的核心思想就是先通过无标签的文本去训练生成语言模型，再根据具体的NLP任务（如文本蕴涵、QA、文本分类等），来通过有标签的数据对模型进行微调。在这篇论文中提出了半监督的方法，即结合了无监督的预训练和有监督的微调。首先，在未标记数据集上训练语言模型来学习神经网络模型的初始参数。随后，使用相应NLP任务中的有标签的数据地将这些参数微调，来适应当前任务。

GPT也是预训练+微调的模式，只不过又LSTM结构进化为Transformer结构，提取文本特征的能力得到增强，但是GPT是一个单向的语言模型，没能同时把握上下文之间的联系，所以在自然语言理解任务中有局限性，但是单向的模型在自然语言生成领域比较有优势，因为符合人类生成语言的逻辑。

GPT模型使用的是多层Transformer的decoder的语言模型，是由12层transformer模块组成的，使用最后的隐藏层来做不同的任务。训练的两个阶段分为无监督的预训练（通过前k-1个词预测第k个词，从前往后，不断前进，属于单向的预测）和有监督的微调（在对模型预训练之后，采用有监督的目标任务对模型参数微调）。这与BERT模型的训练过程相似。

对于有些任务，像文本分类，我们能够直接用上文的模型进行微调。另外的任务，比如问答系统，需要构造输入的句子对。又比如文本相似度任务，由于没有文本内部的先后顺序，可以链接前后两个文本作为输入等。

在BERT之后，GPT2推出，使用40GB的高质量语料进行训练。同时针对文本输入的表示进行了改进，在生成类文本任务中有着很优秀的表现。另外,GPT2提出了一种类似于“NLP通用模型”的概念，作者认为语言模型=无监督多任务学习。相比于有监督的多任务学习，语言模型只是不需要显示地定义哪些字段是要预测的输出，所以，实际上有监督的输出只是语言模型序列中的一个子集。GPT2使用的是带有任务信息的训练数据，以这种数据形式可以有监督地训练一个single model。

自2018年BERT模型提出以来，BERT模型被广泛应用在不同的自然语言处理任务中，均取得了出色的成绩。但是学者没有停止对BERT的研究，提出了许多针对不同自然语言处理任务的改进。其中比较出色的包括MT-DNN，XLNet，RoBERTa和SpanBERT。

MT-DNN（Multi-Task Deep Neural Networks for Natural Language Understanding）模型是一种多任务深度神经网络，用于跨多种自然语言理解任务的学习表示。多任务学习( MTL )的灵感来自于人的学习活动。在人类的学习活动中，人们经常应用从以前的任务中学到的知识来帮助学习新的任务。多任务学习的优点有两个，一是弥补了有些任务的数据不足问题；二是得益于正则化效果，这种效果使得学习到的表示具有更强的泛化能力，防止模型过拟合。

MT-DNN模型的思想是，多任务训练和预训练语言模型是互补的技术，可以结合起来提高文本特征表示的学习效果，进而提高各种自然语言处理任务性能的表现。因此，MT-DNN模型的训练也分为两个阶段，即预训练和微调。但是，MT-DNN在微调阶段用多个特定于任务的层进行多任务学习。主要有四种类型，其具体任务全部出自GLUE（General Language Understanding Evaluation，通用语言理解评估）：（一）单句分类：CoLA（Corpus of Linguistic Acceptability）任务是预测英语句子是否合乎语法，SST-2（Stanford Sentiment Treebank）任务预测电影评论是正向还是负向。（二）文本相似度：这是一个回归任务。对于给定的句子对，模型计算二者之间的相似度。在GLUE中即为STS-B（Semantic Textual Similarity Bench-mark）这个任务。（三）成对文本分类（文本蕴含）：对于给定的句子对，推理两个句子之间的关系。RTE（Recognizing Textual Entailment）和MNLI（Multi-Genre Natural Language Inference）是语言推理任务，推理句子之间是否存在蕴含关系、矛盾的关系或者中立关系。QQP（Quora Question Pairs）和MRPC（Microsoft Research Paraphrase Corpus）是预测句子是否语义等价。（四）相关性排序：给定一个问题和一系列候选答案，模型根据问题对所有候选答案进行排序。QNLI（Stanford Question Answering）任务是预测候选答案中是否包含对问题的正确答案。这个任务重排了候选答案，将正确答案排在更前。

MT-DNN的体系结构主要可以概括为：底层（即BERT的预训练阶段）在所有任务中共享，顶层（即上述四种类型的任务）代表特定任务的输出。输入是一个句子或一对组合在一起的句子对。训练模型时把所有数据合并在一起，每个batch只有单一任务的数据，同时会带有一个task-type的标志，这样模型就可以分辨不同的任务，进行不同的loss计算。相比于交替训练（先训练任务A再训练任务B），这个训练方法可以有效避免偏向某个任务，近似地优化所有多任务的和。

从训练结果上来看，MT-DNN模型在GLUE中除了WNLI之外的全部任务都超越了已有的模型，尤其是在数据集较少的时候，MT-DNN相较于BERT有着极大的提升。这样的结果可以证明，预训练模型+多任务学习的方式，确实可以提升了模型的表现，而且可以有效解决数据集不足的问题，使得模型具有更好的泛化能力。但是MT-DNN模型也有有待提高的地方，比如作者提到的更有效的训练方法，更深度的权重共享等。

XLNet也是学者对BERT模型的一种改进。XLNet首先提出了自回归语言模型（Autoregressive LM）以及自编码语言模型（Autoencoder LM）的分析策略。自回归语言模型就是指根据上文内容预测下一个可能跟随的单词，就是常说的自左向右（或从右向左）的语言模型任务训练出的语言模型，自回归语言模型的优势是擅长生成式自然语言处理任务。 因为在生成上下文时，通常是单向的，自回归语言模型很自然地适用于此类 NLP 任务。但是，它只能使用前向上下文或后向上下文，这意味着它不能同时使用前向和后向上下文。这样的缺点使得预测的准确率仍有提高的空间。而自编码语言模型可以同时关注上下文，因为自编码语言模型的目的是从损坏的输入重建原始数据，方法是在输入中随机用伪码标记代替一部分单词，然后预根据上下文单词来预测这些被替换掉的单词，从而得到原句。BERT模型就是一个很典型的自编码语言模型，因为BERT模型预训练时的任务之一就是Masked Language Model。这样的训练方法可以兼顾前文和后文，能比较自然地融入双向语言模型，相较于自回归语言模型是一个进步。但是自编码语言模型也有缺点。此训练方法在输入侧引入伪码标记，但是Fine-tuning阶段是看不到伪码标记的，导致预训练阶段和微调阶段不一致的问题，从而引入了一些人为误差。而且该方法假设被伪码代替的词之间彼此独立。而在实际数据中，被代替的词之间可能会存在相关性，我们想要语言模型学习这种相关性来对这些被代替的词进行更准确的预测。

XLNet试图找到一种方法，融合自回归和自编码两种语言模型，集合两者的优点，避免两者的缺陷。XLNet采用的是自回归语言模型，为了解决上述的缺点，作者提出了排列语言模型（Permutation Language Modeling）。传统的自回归模型是把序列的值按顺序进行建模，所以是单向的。XLNet采用的方法是最大化所有可能的序列的因式分解顺序的期望对数似然，即随机排列组合句子中的单词，在随机排列组合后的各种可能里，再选择一部分作为模型预训练的输入。假如我们有一个序列abcd，如果我们要预测字母c，先对该序列进行因式分解，最终会有24种排列方式，假如抽取cabd的排列方式，因为c的左边没有其他的值，所以该情况无需做对应的计算；假如抽取adcb的排列方式，模型就会保留a，d的信息，从而使得上下文信息都能被保留，解决了传统自回归语言模型的缺点。而且这种方法避免了采用伪码标记，也解决了BERT模型的缺点。

但是这种方法在实现的过程中也有困难。虽然排列语言模型能满足目前的目标，但是对于普通的transformer结构来说存在问题。传统的Transformer结构中，某个单词的内容和位置向量在输入到模型前就已经加在一起了，后续的隐向量同时具有内容和位置的信息。但是，我们希望在预测下一个词时只能提供位置信息，不能提供内容相关的信息。因此模型需要同时做两件事，首先它需要预测自己到底是哪个字符，其次还要能预测后面的字符是哪个。为了解决这个问题，论文中提出来新的机制，来实现目标位置感知——双流自注意力机制（Two-Stream Self-Attention）。该机制包括两部分，一个是内容流自注意力，就是标准的Transformer的计算过程，同时编码了上下文和自身的内容和位置信息；二是查询流自注意力，包含上下文的内容信息和目标的位置信息，但是不包括目标的内容信息。两个流共同计算，进行模型的训练。

同时，XLNet模型还集成了transformer-xl的两个最重要的技术点，即片段循环机制和相对位置编码。片段循环机制主要是为了解决超长序列的依赖问题，首先取第一个段进行计算，然后把得到的结果的隐藏层的值进行缓存，第二个段计算的过程中，把缓存的值拼接起来再进行计算。该机制不但能保留长依赖关系还能加快训练，因为每一个前置片段不需要再重新计算。但是BERT模型中采用的绝对位置编码，没法区分之前存储的信息到底是哪一个片段里的，这就导致了一些位置信息的损失。因此XLNet采用相对位置编码，在计算attention的时候根据当前的位置和要参考的位置的相对距离来”实时”体现在attention的计算中。

在实验中，XLNet的表现也相较BERT有明显提升，尤其是在较长的文本上的表现。排列语言模型的机制对于自回归和自编码两种模式的融合，片段循环机制的引入，更大的数据规模，这些因素共同提升了XLNet在自然语言处理领域的表现。

RoBERTa（a Robustly Optimized BERT Pretraining Approach）是BERT 的改进版。与BERT使用相同的网络结构，但是从以下四个方面对BERT进行了提升：

首先是更大的数据规模，更大的batch size，更长的训练时间。RoBERTa 在训练过程中使用了更大的bacth size。尝试过从 256 到 8000 不等的bacth size。训练数据包括160GB 纯文本，相较于最初的BERT使用16GB 的数据集和英语维基百科进行训练，RoBERTa显然会具有更好的性能。

第二，RoBERTa使用动态掩码（dynamic masking）的方式进行预训练。原始Bert模型对每一个输入序列随机选择15%的单词替换成掩码，为了消除与下游任务的不匹配，还对这15%的单词进行分类（1）80%替换成掩码；（2）10%不变；（3）10%替换成其他词。但整个预训练过程，这15%的单词一旦被选择就不再改变，每个epoch都是重复，后续的每个训练步都采用相同的掩码。这就叫做静态掩码。作者为了研究掩码对于模型性能的影响，设计了两种改进方式进行了对照。第一种方式是在预处理的时候将数据集用不同的掩码拷贝 10 次，等价于原始的数据集采用10种静态 mask 来训练。第二种方式是不在预处理的时候执行掩码标记，而是在每次向模型提供输入时动态生成掩码。经过实验发现，第一种方法与原始BERT的静态掩码方式效果相同，而第二种动态掩码的方式可以提升模型的性能。所以在RoBERTa中，作者采用动态掩码的方式进行预训练。

第三，RoBERTa研究了原始BERT预训练过程中，NSP（Next Sentence Prediction，下一句预测）任务的效果。NSP任务的目的是原为了捕捉句子之间的关系，方法是输入连续的多个句子组成的文本A和B，判断这两组句子是否是连续的。在训练集的数据中，50%的B是A的下一组句子，50%的B是随机抽取的。近来有学者对于NSP任务提出质疑，认为这个任务不是必要的。RoBERTa的作者对此设计了不同的训练方法进行了实验：（1）原始BERT的NSP任务（2）用单个句子代替原来的多个句子，且句子长度不超过512。 由于这些输入明显少于512 个单词，因此增加batch size的大小，以使单词总数保持与原始BERT的NSP任务相似。（3）舍弃NSP任务，直接用来自同一个文档或者不同文档的连续多个句子作为输入，单词总数不超过 512 。一次输入可能会跨越文档边界，如果跨文档，则在上一个文档末尾添加文档边界标记。（4）舍弃NSP任务，直接用来自同一个文档的连续多个句子作为输入，单词总数不超过 512 。由于在文档末尾附近采样的输入长度短于 512个单词，因此在此情况下动态增加batch size大小以达到与方法（3）相似的单词总数。经过实验，方法（4）的性能表现最好。但是由于方法（4）需要动态调整batch size，为了避免不必要的麻烦。RoBERTa的作者采用了实验结果最接近方法(4)的方法（3）。这两种方法的表现都明显优于原始BERT的方法。

第四，RoBERTa采用了byte-level的字节对编码（BPE）的方式处理自然语言语料库中常见的大量词汇。字节对编码不依赖于完整的单词，而是依赖于子词(sub-word)单元，这些子词单元是通过对训练语料库进行统计分析而提取的。原始BERT使用的是 character-level 的大小为30K的字节对编码，而RoBERTa使用 bytes 而不是 unicode 字符作为子词的基本单位，可以编码任何输入文本而不会引入未知标记。词表大小也从30K提升到50K。这样的改进为原始BERT模型的预训练增加了1500万以上的参数，有效的提高了模型的表现。

RoBERTa设计实验评估了许多设计训练时的策略，并通过更高级的测试环境实现了对原始BERT模型的改进，而且在测试集上的表现甚至超越了XLNet，证明了设计训练策略的重要性。

SpanBERT也是针对BERT在预训练阶段的训练策略做出的改进。与RoBERTa相似的是，SpanBERT在训练时也舍弃了NSP任务，SpanBERT的作者给出了如下的解释：（1）相比起两句拼接，一句长句，模型可以获得更长上下文（类似 XLNet 的一部分效果）（2）在 NSP 的负例情况下，基于另一个文档的句子来预测词，会给 Masked Language Modeling 任务带来很大噪音。

SpanBERT的创新之处在于：

第一，提出了更好的掩码方案。对于原始BERT模型，训练时会随机选取整句中的最小输入单元来进行遮盖。这样会让本来应该有强相关的一些连在一起的字词，在训练时是割裂开来的。这样会导致训练结果的表现下降。为了解决这个问题，有学者尝试将这样具有较强相关性的单词连接起来一起用掩码标记（BERT WWM 模型）。还有学者更进一步，直接将由几个词组成的实体都遮盖掉（如ERNIE模型）。SpanBERT的做法是，根据几何分布，先随机选择一段（span）的长度，之后再根据均匀分布随机选择这一段的起始位置，最后按照长度遮盖。经过作者的实验，这种random span的方法是优于以上两种改进方法的。

第二，SpanBERT在训练阶段新加入了一项任务：Span Boundary Objective。在训练时取 Span 前后边界但不在Span内的两个词，然后用这两个词向量加上 Span 中被遮盖掉词的位置向量，来预测原词。这项新任务的加入，大大提高了random span的掩码方法的表现。

在以上创新点的基础上，作者主要对抽取式问答，指代消解，关系抽取，还有GLUE 做了实验。发现SpanBERT的表现普遍强于 BERT，而且SpanBERT 尤其在抽取式问答上表现好，这与它新加入的SBO预训练目标关系较大。

除了以上提到的已有模型，还有学者对于BERT模型在不同任务下的表现进行了研究。比较有代表性的如：（1）BERT模型在Argument Reasoning Comprehension 任务中的表现。ARC 任务是在给定的前提下，对于某个陈述，相反的两个依据哪个能支持前面提到陈述的推理。作者进行了实验，得出了结论并提出了合理疑问，认为BERT模型并没有真正学习到做出推测所需要的语义信息，只是利用了某些出现频率很高而且对于推测有很大影响的词，比如not，are等。作者又设计实验，创建了对抗数据集（Adversarial Dataset），重新进行实验，发现BERT模型在ARC任务上的表现并不好。（2）BERT模型在 Natural Language Inference（自然语言推理，NLI） 任务中的表现。作者首先假设在 NLI 中表现好的模型可能利用了三种启发式的特征，在任务的前提中就给了模型一些提示，有如下三种：语义重叠，对应的 推理是前提的子序列，对应的推理是前提的子串。基于这个假设，作者也做了实验并观察到，MNLI 训练集中许多数据点都存在这样的特征，且对应的选项是正确的数量远多于不正确。针对这种情况，作者构造了 HANS 数据集，均衡两种类型样本的分布，并且标记了每条数据的前提是否涉及上述几种特征。实验时模型在 MNLI 数据集微调，在 HANS 数据集评测，结果推理正确的数据点中模型都表现不错，而在推理错误类型中模型表现欠佳。这一实验结果支持了作者的假设：模型过度利用了这些启发式特征信息。因此作者在原来的训练集中加入了一定的 HANS 数据，构造了 MNL + 数据集，让模型在该数据集微调，最终获得了比原来更优秀的结果。

虽然BERT在许多多面仍有提高的空间，但BERT作为这些新模型的基础，仍具有可观的研究价值和提升潜力。接下来我们将通过BERT方法，实现在真实平台上可用，有效，高效的文本多分类模型，并研究BERT模型在文本分类任务中的表现。

BERT

本文研究的是基于原始BERT模型的文本分类任务。BERT的全称是Bidirectional Encoder Representation from Transformers，即Transformer结构中双向的Encoder。

Attention

要介绍Transformer结构，首先要介绍注意力Attention这个机制。注意力的灵感来自于人处理信息时采取的一种手段，具体表现为我们如何对信息的不同区域关注度会有所区别，通常对于感兴趣或有更大价值的部分往往会分配大量的注意力。在深度学习中，注意力可以大致理解为对于某一个向量关注的程度如何，我们使用注意力向量来估计关注的部分和其他元素之间的关系强弱，并将不同部分的值的和用注意力向量加权得到的结果作为目标的近似值。

在自然语言理解领域内，注意力这一机制大放异彩。最初的自然语言理解任务中，解决方案大多为Sequence To Sequence模型。目的是将一个输入序列转换为一个新的目标序列，并且输入序列和目标序列的长度可以是不固定的。为了实现这一目的，模型主要做两件事情：（1）将输入序列数据的信息压缩到一个固定长度的上下文向量（context vector）中，得到的表示向量可以较好的包含整个输入的信息。这个阶段也叫做Encoder。

（2）使用上下文向量对模型进行初始化，然后输出转换后的向量。这个阶段也叫做Decoder。

这种传统的模型可以完成机器翻译、生成问答对话、句法分析等自然语言处理任务，但这个机制有一个关键的不足：使用固定距离的上下文向量，会导致其不能记住较长的句子。一旦完成了对于某个输入的处理，它就会忘记之前已经学到的部分。而且，上下文向量可能无法完全表示整个序列的信息。

为了解决这个问题，科学家创造出Attention机制。带有 Attention 机制的 Encoder-Decoder 模型中，上下文编码就不在是输入序列的直接编码，而是要从序列中学习到每一个元素的重要程度，然后按重要程度将元素合并。在 Encoder 将输入的序列元素进行编码时，得到的不在是一个固定的上下文编码，而是存在多个隐藏状态，且最后的上下文编码由不同的隐藏状态以不同的权重参数组合而成。所以通过该机制，模型能够选择性地关注输入序列的有用部分，从而学习它们之间的“对齐”，即将原文的片段与其对应的译文片段进行匹配。

在此基础上，又根据不同的输出计算方式，分为Hard Attention和Soft Attention。Soft Attention中又根据加权求和的集合不同，分为Global Attention和Local Attention。

Attention机制使模型能够有效捕捉文本的全局联系和局部联系，模型复杂度小，参数少，而且可以通过并行计算减少模型训练时间。Attention机制作为一种思想，可以根据实际情况和多种模型进行结合。Google就在Attention机制的基础上，设计了Transformer这样一种结构，并以Transformer为基本单元，设计出BERT模型，在自然语言理解领域取得了非凡的成就。

Transformer

我们首先介绍Self-Attention的机制。Self-Attention是Attention的特殊形式。为了研究一个单词在单词所在的句子中受其他单词影响的情况，把要处理的单词作为输入，把单词所在原句的各个部分通过Encoder得到对应的隐藏状态，按照上一部分介绍的计算方法进行计算。Self-attention使我们能够学习到当前单词和句子中前一部分之间的相关性。

在Self-Attention的基础上，我们通过不同的线性变换对输入和隐藏状态进行投影，分别计算注意力，最后将不同的attention结果拼接起来，这种方式被称作Multi-head Attention。Multi-head Attention使得模型能够在不同的位置共同关注来自不同表示子空间的信息，如果只有一个Attention，平均就会抑制这种情况。

下面介绍Transformer结构。Encoder部分用于生成基于注意力的表示。首先，Encoder部分把输入进行Positional Encoding，这是为了明确关于单词在句子中位置的信息。然后将encoding后的数据与embedding数据求和，这样就加入了相对位置信息。将得到的词向量进行Multi-head Self-Attention计算，然后进行残差连接（Residual Connection，通过将前一层的信息传递到下一层，解决多层神经网络训练困难的问题）和层归一化（Layer Normalization，通过对层的激活值的归一化，加速模型的训练过程，使其更快的收敛）。再把得到的结果输入position-wise类型的全连接前馈网络层。然后再经过残差连接和层归一化得到输出。Encoder默认有6层。

Decoder默认也是6层，每次计算之后也都会进行残差连接和层归一化。这一部分的输入有两个，一个是上一级decoder的输出，一个是Encoder阶段的输出。对于上一级Decoder的输出，首先进行一次Masked Self-Attention，其中Mask掉的是当前时刻以后的信息，这样可以保证预测时不会利用被预测单词之后的文本信息。然后把得到的输出作为下一环节的输入，与Encoder部分的输出一起进行Attention的计算（注意这一次不是Self-Attention，而是普通的Attention计算）。最后把得到的结果输入全连接层得到输出。decoder的输出会经过线性层和softmax层得到最终的结果。

BERT

模型结构

BERT就是以Transformer结构为基本元素，通过预训练和微调的方法训练出的模型。BERT模型最大的特色在于，通过对处于所有层中的上下文联合作用，来对深度双向表示的未标记文本进行预先训练。在BERT之前，有两个策略将预训练的语言表征应用到下游任务：基于特性（feature-based)和微调。前者（例如ELMo）采用将预训练表征作为额外特征的具体任务架构。后者（例如OpenAI GPT）引入了最小的随任务变化而变化的参数，可以通过简单地微调所有预训练参数对下游任务进行训练。这两种方法在预训练中使用相同的目标函数，使用单向语言模型来学习通用语言。BERT的作者认为，单向语言模型这种技术限制了预训练表征的能力，特别是对于微调方法。这将会限制对预训练中使用架构的选择。这种限制对句子级别的任务而言影响不大，但对于一些必须双向合并语境，像问答这样的单词级别的任务时这种限制就变得影响很大。为了减轻这种单向模式的限制，BERT采用Masked Language Model作为预训练目标并且采用深层的双向Transformer组件（使用双向self-attention，使得每一个单词可以联系到其之前以及之后的单词）。

BERT模型还根据模型大小分成两种：BERTBASE (L=12, H=768, A=12, Total Param- eters=110M) and BERTLARGE (L=24, H=1024,A=16, Total Parameters=340M)。其中L表示层数，H表示隐藏大小，A表示self-attention heads的数量。

输入输出

由于BERT是一个预训练模型，其必须要适应各种各样的自然语言任务，因此模型所输入的序列必须有能力包含一句话（文本情感分类，序列标注任务）或者两句话以上（文本摘要，自然语言推断，问答任务）。所以输入的Embedding由三种Embedding求和而成：（1）Token Embeddings是词向量，第一个单词是CLS标志，可以用于之后的分类任务。（2）Segment Embeddings用来区别两种句子，因为预训练要做以两个句子为输入的分类任务。（3）Position Embeddings将单词的位置信息编码成特征向量。其中，为了能在单词的有效性和字符的灵活性之间取得一个折中的平衡，作者还使用了WordPiece算法，将单词划分成一组有限的公共子词单元。

预训练

任务一 Masked Language Model

标准条件语言模型只能从左到右或从右到左进行训练，因为双向条件允许每个单词间接"看到自己"，并且模型可以在多层语境中轻微地(trivially)预测目标单词。

为了训练一个深度双向表示，BERT在预训练时随机屏蔽了一部分的输入单词，然后预测那些被屏蔽的单词。此过程即为" Masked Language Model "（MLM），在文献中通常被称为完形填空任务。在这种情况下，与掩码对应的最终隐藏向量会像标准 LM 中一样（输入到全连接，然后用softmax输出每个token的概率，最后用交叉熵计算loss）。在所有实验中，我们随机屏蔽每个序列中15%的经WordPiece嵌入后的单词。

虽然这样的预训练任务可以获得一个双向预训练模型，但缺点是，这项任务没有正确匹配预训练和微调这两个过程，因为掩码这个单词在微调期间不会出现。为了改进这一缺点，BERT并不总是用掩码标记来替换被遮蔽的单词。训练数据生成器会随机选择 15% 的单词位置进行预测。如果第i个单词被选中，我们（1）有80%的可能将其替换为掩码标记（2）10%的可能替换为随机的单词（3）10%的可能保持原单词不变。随后，第i个单词将和交叉熵损失一起被用来预测原始单词。

任务二 Next Sentence Prediction (NSP)

许多重要的下游任务，如问答（QA）和自然语言推理（NLI）都是基于对两个句子之间关系的理解，而这种关系并不是被语言模型直接捕获的。为了培养一个了解句子关系的模型，BERT预先训练了一个二分法语句对预测任务，这项任务可以从任何单一语言的语料库中产生。具体来说，为每个预训练示例选择句子 A 和 B 时，B有50% 的可能是 A 之后的实际下一个句子（标记为 IsNext），50%的可能是来自语料库的随机句子（标记为 NotNext）。 如图 1 所示，C 用于语句对预测（NSP）。尽管这项任务很简单，但这项任务对 QA 和 NLI 都非常有用。

NSP 任务与representation-learning objectives密切相关。但是，在BERT之前，只有句子级别的嵌入被转移到下游任务，但是BERT 把包括句子和单词等所有参数都进行了转化，用来初始化最终任务模型参数。

BERT使用的预训练数据

这两个任务所需的数据其实都可以从无标签的文本数据中构建，样例为：

Input1=[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

Label1=IsNext

Input2=[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]

Label2=NotNext

把每一个训练样例输入到BERT中可以相应获得两个任务对应的loss，再把这两个loss加在一起就是整体的预训练loss。（也就是两个任务同时进行训练）

对于预训练语料库，BERT使用 Books语料库（800M 单词）和英语维基百科（2500M 单词）。对于维基百科，BERT只提取文本段落而忽略列表、表格和标题。并且十分重要的一点是BERT使用文档级语料库而不是杂乱的句子级语料库（如Billion Word Benchmark）来提取冗长的连续序列。这有利于模型具备抽象连续长序列特征的能力，更好地学习句子之间的关系。

微调

经过预训练的BERT模型可以通过一个额外输出层进行微调，以创造出适用于广泛任务模型，例如问答任务和语言推理，而不需对具体的任务做大量架构修改。transformer中的self-attention机制允许BERT通过交换适当的输入和输出来建模许多下游任务（无论是涉及单个文本还是文本对），这使得微调非常简单。例如：（1）在分类任务中，例如情感分析等，只需要在 Transformer 的输出之上加一个分类层。（2）在问答任务（例如SQUAD v1.1）中，问答系统需要接收有关文本序列的问题，并且需要在序列中标记答案。 可以使用 BERT 学习两个标记答案开始和结尾的向量来训练Q＆A模型。（3）在命名实体识别（NER）中，系统需要接收文本序列，标记文本中的各种类型的实体（人员，组织，日期等）。 可以用 BERT 将每个 单词的输出向量送到预测 NER 标签的分类层。

在本课题中，我们将根据文本分类任务对已经经过预训练的中文BERT模型进行微调，并进行了如下的实验。

实验：单标签多类分类

实验采用的是BERT-Base的中文版本，链接为<https://storage.googleapis.com/bert_models/2018_11_03/chinese_L-12_H-768_A-12.zip>。

相较于普通的BERT-Base模型，中文版本增加了BasicTokenizer和tokenization.py两个文件使得模型可以读取中文字符，并通过在中文字符两侧增加空格使得文本可以通过WordPiece算法进行embedding。我们只需要在上述预训练模型的基础上进行微调，使得模型可以满足文本多分类任务的需要即可。

微调阶段使用的参数

--task\_name：选择要使用的处理模式（Processor），包括XNLI模式，MNLI模式，MRPC模式，Cola模式和本实验使用的自己编写的Multilabel（多标签）模式。

--do\_train：选择是否是要对模型进行微调。

--do\_eval：选择是否对微调结果定时进行评估。

--do\_predict：选择是否在微调结束后进行测试。

--do\_export：选择是否导出pb模型。

--data\_dir：选择数据集存储的路径。

--vocab\_file：选择进行tokenization的词汇文件。

--bert\_config\_file：选择配置文件路径。

--init\_checkpoint：加载上次的checkpoint。

--train\_batch\_size：设置batch\_size的大小，本课题受限于训练硬件条件，一般选择4或8。

--learning\_rate：设置学习率，一般选择2e-5或3e-5，本实验中采用2e-5。之所以选择2e-5这样比较低的学习率，是为了减轻灾难性遗忘（catastrophic forgetting）现象带来的危害。灾难性遗忘是指在一个顺序无标注的、同种任务可能长时间不复现的任务序列中，AI对当前任务B进行学习时，对先前任务A的知识会突然地丢失的现象。

--num\_train\_epochs：设置epoch数目，一般设置为3.0。

--max\_seq\_length：最大能处理的序列/文本长度，最高设置为512。这是由输入数据处理时需要进行positional embedding导致的。

--output\_dir：选择模型输出路径。

--export\_dir：选择模型导出路径。

数据预处理

本实验使用的是THUCNEWS中文新闻长文本标注数据集（http://106.13.187.75:8003/download/?dataset\_name=thucnews）和今日头条中文新闻短文本分类数据集（http://106.13.187.75:8003/download/?dataset\_name=tnews）这两个数据集。

短文本

今日头条中文新闻短文本分类数据集也是每行为一条数据，以\_!\_分割的个字段，从前往后分别是：新闻ID，分类ID，分类名称，新闻字符串（仅含标题），新闻关键词。在本实验中，我们只需要分类ID和新闻字符串，但是其他数据在预处理时也予以保留。

首先，我们把txt格式的文件转换成csv格式。采用按行读取的方式。由于python中csv模块在生成csv文件时自动用“,”分隔，为了避免引起错误，我们把文本中所有的“,”都替换成“，”（中文逗号）。然后再一行行写入csv文件。

为了让分隔符更独特，防止样例中出现混淆，转化为csv文件后，我们又将“,”分隔符转换成原来的“\_!\_”。在后面进行数据集处理时，我们只需按\_!\_分隔，按照不同部分的index获取数据即可。

长文本

THUCNEWS中文新闻长文本标注数据集中每行为一条数据，以\_!\_分割的个字段，从前往后分别是 类别ID，类别名称，文本ID，文本内容。长文本与短文本在预处理阶段的操作基本相同。唯一的区别在于，由于BERT模型有长度限制，样例最长为512个字，其中还包括一些特殊token，在文本分类中，要包含开头的[CLS]和结尾的[SEP]，因此实际只能最多装510个字。而数据集中有些样例是长于510个字的。为了解决这一问题，我们提出了以下几种思路：

1. 取文本的前510个字（前截断）
2. 取文本的后510个字（后截断）
3. 取文本的前255和后255个字拼接（前后截断）
4. 采用文本提取方法提取出长度适合的文本（文本摘要）
5. 把超过510个字的文本拆分成多个长度合适的文本（拆分法）

如果一个长文本的重要信息是在开头，可能方法（1）效果是比方法（2）要好。同理，方法（2）对信息点在结尾的长文本效果较好。使用方法（3），是综合方法（1）和方法（2），一般而言是好于单一的截断方式。但是这些方法，都会丢失一部分序列信息。在文本不是特别长的场景这三种方法效果可能不错。但如果是更长的文本，文本长度几千字，如果直接使用以上三种截断法，必然会丢失大量信息，造成准确率下降。

在某一些场景，可以尝试对原始文本进行过滤，降低有效文本的大小，即方法（4）。关键点是如何筛选出有效句子和有效信息。这就涉及到自然语言理解任务中的文本摘要任务。

自然语言处理（NLP）中有两种文本摘要生成方法：抽取式和抽象式。抽取式摘要的方法是，抽取一段文本中表示重点内容的单词子集，并结合起来生成摘要。抽取式摘要通常需要衡量基本句子成分的权重，并根据权重结果生成摘要。不同类型的算法和方法均可用于衡量句子的权重，之后根据各成分之间的关联性和相似性进行排序-并进一步将这些成分连接起来以生成摘要。抽象式摘要的方法是，使用高级深度学习方法（advanced deep learning technique）用于解释和缩写原始文档，生成表示源文本中最重要信息的新短语和句子，所以这些抽象式算法有助于克服抽取式摘要中的语法不准确问题。但是在本实验的情景下，由于BERT模型会学习单词与前后单词，句子与句子之间的关系，从定性的角度讲，抽象式摘要方法中生成的新短语和句子可能会为微调过程带来噪音，会比抽取式的摘要方法带来更大的误差，而且BERT模型的训练方法会进一步放大这种误差，因此本实验采用抽取式摘要方法进行过长文本的摘要。

最终本实验采用的是TextRank算法。TextRank 算法是一种抽取式的无监督的文本摘要方法，是一种用于文本的基于图的排序算法，通过把文本分割成若干组成单元（句子），构建节点连接图，用句子之间的相似度作为边的权重，通过循环迭代计算句子的TextRank值，最后抽取排名高的句子组合成文本摘要。具体流程为：1. 第一步是把所有文章整合成文本数据2. 接下来把文本分割成单个句子3. 然后，我们将为每个句子找到向量表示（词向量）。4. 计算句子向量间的相似性并存放在矩阵中5. 然后将相似矩阵转换为以句子为节点、相似性得分为边的图结构，用于句子TextRank计算。6. 最后，一定数量的排名最高的句子构成最后的摘要。在本实验的数据预处理过程中，我们先运行textrank程序把数据集中的长文本进行摘要，然后再进行微调。

方法（5）是将一个整段的文本拆分为多个部分，每一个部分的长度小于510。拆分可以暴力地通过510的大小进行划分，或者通过断句的方式，将相邻的句子放入一个部分。这种方法将所有序列都放入模型之中。考虑到了全局的信息，对文本很长且对截断敏感的任务有较好的效果。但有一些缺点：比如性能较差，原来截断法需要encode一次，方法（5）需要encode多次，篇章越长，速度越慢。而且各个部分之间的联系会丢失，可能会出bad case。在本实验情境下，由于BERT模型会学习句子之间的关系，那么暴力的通过510字的大小进行划分就会为模型训练带来不便。因此本实验采用断句的方式，把过长的样例文本分成同类型的几个长度合适的样例。

本实验中，我们使用以上五种方法分别处理相同的长文本数据集，然后分别进行微调评估结果。

上述分析都是定性的分析，真正的实际效果将在后续的实验中进行对照。

训练流程

由于我们已经下载了预训练好的BERT模型，所以只需要import三个模块：modeling，optimization和tokenization。Modeling提供创建模型的接口，optimization优化训练过程，tokenization对输入文本进行单词化处理。

在微调过程开始之前，首先读取输入的所有参数，根据参数使用tokenization.validate\_case\_matches\_checkpoint函数匹配相应的checkpoint，使用modeling.BertConfig.from\_json\_file函数配置BERT模型。同时还要检查max\_seq\_length是否超过了512，并检查do\_train,do\_predict,do\_eval三个参数至少有一个为TRUE。最后创建输出文件夹，根据task\_name参数选择处理方式（本实验中选择multilabel processor）并读取label\_list（所有分类标签的列表）。

Train

如果do\_train参数为TRUE，则说明我们要进行微调训练。首先要把文本转化为BERT模型的输入。我们在Input\_examples函数中定义了BERT模型输入的格式。分为四部分：guid代表输入样例的id, text\_a代表输入样例的标题, text\_b代表输入样例的正文, label代表输入样例的标签。这四部分共同组成object类型的样例。

为了把存储在csv文件中的数据集转化为上述样例的格式，我们首先调用get\_train\_example函数，从输入参数data\_dir的目录下，读取所使用数据集的csv文件（本实验中短文本使用short\_train/test/dev.csv,长文本使用long\_train/test/dev.csv），然后调用\_create\_examples函数，循环读取每一行数据，并根据train，test和dev三个不同的数据集设置id，格式为“数据集类型”-“行数”，这样每条数据就拥有了不会相同的id。由于数据集里的数据中，标题和正文是使用空格隔开的，无法进行分离（因为标题和正文中同样有可能有空格），所以把标题和正文全部放入text\_a，text\_b为空，然后获得label。

然后我们需要把InputExample的集合转化为TFRecord格式的文件（文件名为train.tf\_record）。这一步骤是由convert\_single\_example函数实现的。

首先，调用tokenizer.tokenize函数把text\_a和text\_b转化为tokens\_a和tokens\_b。然后把tokens\_a和tokens\_b长度相加与max\_seq\_length比较，检查输入的有效性。然后给tokens\_a和tokens\_b的第一个单词前添加[CLS]，表示是文本的第一个词，句子和句子之间添加[SEP]，区别一条文本中的两个句子（在本实验中即区别文本的标题和正文，把标题的segment\_ids设为0，把正文的segment\_ids设为1）。

然后设置训练数据的mask，在本实验中，文本中的单词的mask设置为1，而刚才添加的单词（如[CLS]）的mask设置为0，但是由于我们只将文本中的单词进行mask操作，所以mask序列中的元素全为1。

最后把得到的tokens序列，mask序列用0补齐至max\_seq\_length长度。然后调用InputFeatures函数把input\_mask，segment\_ids等训练特征封装成features对象返回。在最终输入模型前，还需要经过切片（tf.data.Dataset.from\_tensor\_slices），这样就把数据集里的文本转化成BERT模型可以读取的样例。

调整完输入数据后，我们根据样例的长度，epoch数目和batch\_size大小计算训练的步长num\_train\_ step。公式为步长=样例长度/batch\_size\*epoch数目。

这些工作完成以后，我们就开始加载BERT模型。由于BERT模型是用TPUEstimater封装的。所以我们首先使用model\_fn\_builder函数，创建一个model\_fn对象。然后用tf.contrib.tpu.TPUEstimater函数创建一个estimater对象，调用estimator.train函数。这样我们就可以利用样例中的features对象的参数创建模型并开始训练。在model\_fn对象中，我们使用create\_model函数创建初始模型，然后用modeling.get\_assignment\_map\_from\_checkpoint函数加载上次的checkpoint。如果检查estimater的模式为TRAIN，则调用optimization.create\_optimizer函数建立一个优化器，然后用tf.contrib.tpu.TPUEstimatorSpec函数输出loss。

EVAL

EVAL阶段就是对训练模型的表现进行评估，包括准确率，loss等指标。EVAL阶段与TRAIN阶段在数据集处理，方法调用上没有区别，只是用estimator.evaluate函数代替了estimator.train函数。因此在加载模型时estimater的模式为EVAL，此时就不再是用TPUEstimatorSpec函数输出loss，而是调用tf.metrics.accuracy和tf.metrics.mean函数输出准确率和total\_loss。

PREDICT

PREDICT阶段和以上连两个阶段都不同，因为数据集是测试集，所以不像训练和评估阶段时文本和所属的分类都已知，而是模型要根据文本判断所属的类型，然后把所属的类别进行输出，与测试集中的标准数据相比较，得出准确率。因此，在数据集处理上，我们把所有样例的label全部设置为0，然后进行处理。加载模型时，使用estimator.predict函数，用TPUEstimatorSpec函数输出模型做出的预测和该预测的概率，输出到名为test\_results.csv的文件中。

在微调过程结束后，我们使用output.py文件手动比较test\_results.csv和测试集中的数据，计算准确率。

EXPORT

EXPORT阶段的作用，是把模型导出为pb格式的文件，方便接下来开发应用时模型的持久化。原始的BERT模型主要设计为单次运行的目的，如果把 do\_predict 参数设置成 True，也可以进行预测，但输入样本是基于文件的，并且不支持将模型持久化在内存里进行多次调用，会造成不必要的麻烦，影响应用的性能。因此需要自己改写代码，达到两个目的：

1.允许将模型加载到内存里。允许一次加载，多次预测。

2.允许读取非文件中的样本进行预测。譬如从标准输入流读取样本输入。

BERT模型使用TPUEstimater封装，但是遗憾的是 estimator 原生并不支持一次加载，多次预测。因此需要使用 estimator.export\_saved\_model() 方法把 estimator 重新导出成 tf.saved\_model。执行后，会有一个时间戳命名的模型目录。这样之后我们就不需要estimator 对象了，可以直接调用tf.contrib.predictor.from\_saved\_model函数自行从刚刚的模型目录加载模型，在应用模型时会方便许多。

实验：多标签多类分类

在多标签分类问题中，训练集由实例组成，每个实例可以被分配有表示为一组目标标签的多个类别，并且任务是预测测试数据的标签集。例如：文本可能同时涉及任何宗教，政治，金融或教育，也可能不属于任何一种。电影可以根据其摘要内容分为动作，喜剧和浪漫类型。有可能电影属于romcoms [浪漫与喜剧]等多种类型。

在多类分类中，每个样本被分配给一个且仅一个标签（即单标签多类分类）：水果可以是苹果或梨，但不能同时是两者。在多标签情况下，每个样本可以属于一个或多个类。

解决多标签分类问题，由两种思路：

1. 把多分类问题拆分成多个二分类问题。这种方法在标签量小，模型不复杂的时候可以使用。但是在实际应用中，标签数量一般会超过5个，这意味着一条数据我们需要进行至少五次输入模型的过程，而每次输入都需要加载模型，这带来了巨大的时间和空间开销，不适合应用在舆情平台这样的应用中。
2. 把单标签分类模型进行改造，使其能够满足多标签分类的需要。这种方法可以有效解决上述的问题，只需一次输入，一次加载模型，就可以直接得到标签数据。但是与单标签分类相比，多标签分类进行预测时会有多个标签概率较高。而原模型的输出层采用的softmax函数的输出值相互关联，不能满足多标签分类的需要。

本实验在准备阶段存在一个问题，由于没有经过很好地标注的中文多标签文本分类数据集，所以本实验只能采用英文数据集，同样的，BERT模型也不再使用BERT-Base的中文版本，使用BERT-Base（Uncased）预训练模型（https://storage.googleapis.com/bert\_models/2018\_10\_18/cased\_L-24\_H-1024\_A-16.zip），最终本实验数据集采用恶毒评论分类挑战（https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data），除了序号和原始文本以外，每行数据都包含了6个维度的标注，分别是：toxic（恶毒），severe\_toxic（非常恶毒），obscene（污言秽语），threat（威胁），insult（侮辱），identity\_hate（憎恨）。如符合某一项标签，则标注为1，否则标注为0。

多标签分类与单标签分类在代码逻辑上并没有什么不同，主要区别有两个：

1样例数据输入时，单标签只有一个label，故inputexample对象中的label属性为一个字符串，而多标签多类分类中样例数据有多个标签，如：[0,0,0,0,0,0]。因此inputexample对象中的label属性是作为一个字符数组输入的。

2在文本分类中将分类器的原始输出值映射为概率的方式有softmax()，sigmoid()两种。在简单的二进制分类中，两者之间没有太大的区别，但是在多标签分类的情况下，sigmoid允许处理非独占标签（也称为多标签），而softmax处理独占类。Sigmoid函数会分别处理各个原始输出值，因此其结果相互独立，各标签概率总和不一定为1。相反，Softmax函数的输出值相互关联，其概率的总和始终为1，增大某一类别的概率，其他类别的概率必须相应减少。在多标签多类分类实验中，显然Sigmoid函数是更适合本实验的映射方式。因此把原模型中tf.nn.softmax函数，替换为tf.nn.sigmoid函数，同时在计算loss时，为计算每个示例损失，我们使用tf.nn.sigmoid\_cross\_entropy\_with\_logits测量离散分类任务中的概率误差，其中每个类是独立的而不是互斥的。

进行以上更改后，我们就可以开始训练。

训练结果

经过实验，短文本单标签分类准确率为89.78%，达到了目前BERT-Base模型的基准值。长文本单标签分类准确率为96.02%，比基准值高了0.67个百分点。这样的提升来自于前后截断的方法，因为前后截断非常适用于新闻类的文本。

不足之处和改进方法

首先，本实验采用的数据集为新闻类文本。众所周知，新闻类文本往往会在开头或结尾标注上文章作者的姓名或者报纸的名称，还会存在“据……报道”“作者将进一步追踪事件的发展”等无关紧要的句子。这些句子对文本分类没有作用，甚至还会干扰模型的训练。因此应当将这些无关文本去除，保留有用的文本。

其次，对于单标签短文本分类，受限于文本长度，准确率没有长文本那么高，但是由于数据集量大（10万余条），故取得了不错的准确率。但是数据量大，学习率低也导致训练时间过长，模型收敛太慢，对训练硬件的要求比较高。而且，在本实验选择的数据集中，所有数据一共分成17个标签，标签数量较多也是影响学习效率的因素之一。

再次，对于单标签长文本分类，虽然取得了96%的正确率，但是我认为还有提高的空间。关键在于过长文本的处理方法。三种截断法中前255字+后255字的方法虽然取得了最好的表现，但是这样的表现并不具有普遍性。新闻类文本的结构，大多采用总分的结构，即文本的开头几乎都会有导语来总结新闻内容，新闻的最后也会有总结性的文字对新闻内容进行进一步的总结。这样的总分式的结构使得前后截断的方法“渔翁得利”，因为前后截断几乎总会获得总结全文信息的文本。但是对于其他类型的文本，比如游记，一般采用顺序记叙的方法，这样前后截断的方法可想而知就不会有最好的表现，反而是前510字截断或后510字截断甚至中间取510字截断的方法更能够有效地捕获全文信息。这就告诉我们，在截断法的使用上，我们要联系样例文本的结构进行选择，没有一个一劳永逸的解决方案。我们也不能根据模型在某一个数据集上的表现来判断其在另一个数据集上的表现。

对于文本摘要方法，TextRank算法也不是最好的解决方案。在自然语言理解领域，文本摘要与文本分类相同，也存在基于机器学习或基于深度学习的多种摘要方式。效果比较出色的如聚类方法（将文章中的句子视为一个点，按照聚类的方式完成摘要），序列标注方式（建模为序列标注任务进行处理），序列标注结合Seq2Seq方法（在序列标注的基础上结合 Seq2Seq 和强化学习），句子排序方式（建模为句子排序任务）等。

此外对于文本摘要中生成式（抽象式）方法，本实验只是定性猜想其应用在BERT模型的文本分类任务中会对模型造成负面影响，缺少必要的实验进行证明。

对于把长文章分成短文章的方法的评价与截断法类似。在本实验中，由于样例文本长度超过512字但不会达到两三千字的水平，所以这样的拆分法可能并没有发挥出相应的效果。假如样例文本再长一些，截断法所截取的文本片段可能就不能很好地总结文章内容，这时候拆分的方法就会有更好的表现。

最后，对于多标签分类，受限于使用的数据集是英文的，所以可能模型直接迁移到中文上表现会下降。针对这一问题，可以采用网络爬虫的办法进行解决，但是由于网络爬虫运行效率有限，想通过爬虫方法获得数量上万的样例数据，需要耗费大量的时间。而且爬虫得到的数据需要经过进一步的清洗和预处理，可能会存在bad case，故把这种方法放在附录中，可以进行进一步的研究。

另外，对于本实验选用的数据集，各个标签之间存在着包含关系：一条“非常恶毒”的评论，一定是“恶毒”的。这就意味着标签之间不是相互独立的。这样的包含关系对模型的训练和迁移造成了负面的影响。

综上所述，根据以上对本课题的思考，我们提出以下改进措施：

1 对于短文本分类，尽量减少标签个数，增加样例长度。对于硬件条件没有那么高的实验环境，可以适当减少训练集，测试集和开发集的大小以提高训练效率。

2 对于长文本分类，首先应该明确文章长度，文章类型，文章结构，短一些的采用截断法，更长的篇章级的文本采用拆分法。对于有明确总分总结构的文章可以有针对性的截取总结性的文本作为样例。如果采用文本摘要的方法，则应当多测试几种方法以后选择表现最优的方法进行模型训练。

3 本实验的预训练模型是BERT-Base版本。如果有更好的实验条件，可以选择BERT-Large模型，会获得更好的表现。但是BERT-Large暂时还没有针对中文的预训练模型，所以在英文文本上的表现会更好。而且，对于BERT模型，现在已经有许多研究对它做出了不同的改进，可以利用这些改进后的新模型加以训练，提升文本分类任务的表现。

4 本实验采用的数据集样例数量有限，如果采用更大的数据集，训练更长的时间，模型自然会有更好的表现。

5 对数据集的数据进行进一步的清洗，去除文本中不必要的文字或句子，优化模型的训练过程。

敏感词检测技术研究与实现

Introduction & Related Work

随着信息时代的发展，涉及政治，色情，暴力等因素的敏感词汇也层出不穷。在许多对外公共场合下，有些内容是要经过审查才能显示的。理论上讲，只要涉及用户输入的地方，都需要进行文本校验，以保证敏感词汇不被展示在网络上。所以我们需要一个有效高效的敏感词检测手段，在涉及文本输入的时候进行快速的检测。既不影响用户的使用体验，又能准确地过滤非法信息。在网络审查初期，都是通过人工审核，这种审核方式虽然准确，但与网络上文本产生的速度相比，其效率就显得过于低了。因此，自动化的敏感词检测方法的需求越来越强烈。

最普通的敏感词检测，就是用遍历的方法，从敏感词库中逐条读取敏感词，在待检测文本中进行搜索匹配。实现起来非常容易，但是问题很大：当待检测文本比较长，敏感词汇比较多的时候，这种方法效率很低，严重影响用户体验。

现在被广泛应用的敏感词检测方法有两种。第一种是本文采用的敏感词检测方法：DFA，即确定有限状态机，将字符比较转化为了状态转移。第二种方法是Aho-Corasick自动机算法。该算法是DFA的一种实现。这两个算法有两个特点，一个是扫描文本时完全不需要回溯，一次遍历全部检测；另一个是时间复杂度为O(n)，时间复杂度与关键字的数目和长度无关，因此具有很高的效率，不影响用户的使用体验。

在本课题的研究中，曾经尝试过使用上文陈述的深度学习方法——BERT模型进行敏感词的学习，但是学习效果很差，原因在于，敏感词的长度太短，而敏感词检测又是一个对精度要求比较高的任务。BERT模型无法很好地学习敏感词向量的特征，造成误判漏判的概率很高，不适合应用在实际应用中。所以本课题最终放弃了深度学习方法，改用最准确的DFA算法应用在平台上。

DFA

DFA(Deterministic Finite Automaton)，即确定有限状态机，基本工作原理是通过事件和当前的状态得到下一个状态。在敏感词检测的任务中，我们可以把字符看做状态，把字符间的前后联系视作事件，这样就把字符的比较转化为状态的转移。具体的算法原理为：

首先以敏感词库中的词汇建立一个确定性的树形有限状态机。比如，假设敏感词库里有abc，abd，ef三个敏感词，我们就可以建立a->b->c , a->b->d ,e->f的树形结构。然后以待检测文本作为该有限状态机的输入，使状态机进行状态的转换，当到达某些特定的状态时，说明发生文本匹配，即在待检测文本中检测到了敏感词汇。假设待检测文本为abcdefg，首先检查DFA中是否有a这样一个起始状态，若存在，则检查第二个字符b是否是a的下一个状态，若是，再检查下一个字符c是否是b的下一个状态，结果为是。由于c已经是此确定有限状态机这条分支的最后一个状态，所以我们可以判定待检测文本中存在敏感词abc。检测完毕。

具体代码实现

敏感词检测功能的代码分为三个部分：

add()和parse()函数，负责建立状态机，以多维数组的形式储存敏感词库中的敏感词。

Filter()函数，负责根据待检测文本，以循环的方式查询已建立好的多维数组，得出结论是否匹配。假如匹配，则用长度与敏感词长度相同的连续占位符代替原有的敏感词，并把处理好的文本返回。

主函数main()，负责处理输入输出，调用parse()和filter()，并输出整个敏感词检测的流程所用的时间。

在本实验中，我们不止希望能够鉴别并过滤敏感词，而且我们建立了不同种类的敏感词库，希望能够得到敏感词具体的分类。这就需要在原本的代码中加入一个分类的步骤，解决这一问题有两种思路：

1可以给每一个敏感词加上一个标签，标注该敏感词的敏感原因，假如匹配到了该敏感词，则输出该词的标签。这样做比较直接，但是会造成内存空间的占用变大，假如敏感词数目比较大，这种方式会影响应用的性能。

2 把同种类的敏感词分类，分次多次建立状态机。这样的方法可以有效节省内存空间，但代价是处理时间的增加。这种方法在敏感词种类比较多的时候不占优势。但是对于敏感词种类不多但每个类别中的数目较多的敏感词库比较有优势。

由于本次实验采用的敏感词库有五种分类，每个分类下有几百个长度不同的词汇，最终我们选择方法2对敏感词库进行处理。但是要说明的是，两种方法在这样的数据量下，处理时间上的区别不大，在空间上的优势是我们选择方法2的主要原因。

实验结果

不足之处及改进方法

DFA算法在时间和空间复杂度上已经比遍历的方法有了很大提升，但是如果敏感词库过大，敏感词种类过多，在建立状态机的阶段依然会耗费大量的时间和内存空间，这样对之后的舆情监测平台的设计会造成不利的影响。所以如果能把建立好的状态机存储起来，在有需要的时候进行调用，是比目前一次建立一次使用更好的方式。

其次，如今的网友们已经采取了很多方式绕开敏感词检测。例如，用空格隔开敏感词，或者用相同读音的别的文字代替原本的敏感词，或者用符号表达敏感词的含义等方式。如果敏感词算法可以过滤空格等间隔词，或者学习汉字中的拼音，或者赋予符号一些文字含义，就可以更加“智能”地鉴别敏感词，有效防止绕过敏感词检测的小伎俩。

此外，虽然运用了状态机这样的模型，但是本质上依旧是匹配字符串的方法，在网络环境日益发展的今天，敏感词会以更快的速度出现和传播，这就要求敏感词检测功能要具有快速更新，更大容量的能力。因此如果能把机器学习甚至深度学习方法融入进来，比如上一部分介绍的BERT模型，利用神经网络模型学习敏感文本，获得鉴别敏感词的能力，是比字符串匹配更好的选择。但是受限于敏感词长度太短，而且敏感词的格式复杂多变，可能模型训练会有一些难度，这有待于进一步的研究。

系统综合设计与开发

为了使文本多分类和敏感词检测功能能够更好的集成到平台中，实现这两个功能的应用价值，我们将这两个功能进行封装并设计了一个系统来使用这两个功能并使这两个功能可以更方便地迁移到别的平台。

准备工作

对于文本多分类任务，由于我们之前使用estimator.export\_saved\_model() 方法把模型导出为tf.saved\_model，我们可以使用tf.contrib.predictor.from\_saved\_model函数直接加载模型并持久化在内存里，使用在predict阶段定义的的 predict\_fn 变量，就可以直接进行预测。在此系统中，我们从标准输入流读取问题样本，并预测分类。即先调用InputExample("id", question, None, '某固定伪标记')定义变量predict\_example，再调用convert\_single\_example(100, predict\_example, label\_list,FLAGS.max\_seq\_length, tokenizer)获取样例的feature。最后调用predict\_fn函数进行预测，获得预测结果prediction。变量prediction中储存着每个标签对应的概率，我们先获取这些概率probabilities = prediction["probabilities"]，然后寻找概率最大的一项，在标签列表中按相同的index获得标签，即label = label\_list[probabilities.argmax()]。这样就完成了一次完整的预测过程。

因此我们编写test.py文件或改写原来的文件，把文本多分类功能尽可能地从原模型的复杂代码中剥离出来，降低运行成本。首先我们可以减少参数的个数，因为预测过程不涉及训练部分，所以训练相关的参数可以全部去除，包括：--data\_dir，--train\_batch\_size，--learning\_rate，--num\_train\_epochs等。然后，我们也可以去除原代码中创建的model\_fn和Estimator对象，因为我们有新的加载模型的方式。最后，我们去除不必要的参数检查，比如--do\_train，--do\_eval，--do\_predict和--do\_export，这样可以去掉不必要的数据处理步骤。

系统设计

系统采用Flask框架，连接MySQL数据库，搭建了一个从数据库读取/向数据库储存数据，在前端进行信息展示的平台。

本系统为了展示文本多分类和敏感词检测的功能，设计了三个功能：上传文章，按标签搜索文章，新增分类标签。

上传文章并归档

上传文章归档功能模拟的是用户编写一篇文章上传至平台的操作。这一功能主要展示敏感词检测和文本多分类的实际应用。首先用户在输入框中输入文本，编辑标题以后，点击提交。此时后台先进行敏感词检测操作：获得用户的文本信息，调用DFA函数。假如用户的输入信息中存在敏感词，则会重定向到新的页面，页面中展示敏感片段并展示敏感原因。假如用户的输入信息中不存在敏感词，则会调用文本多分类函数，加载模型，对用户的输入文本进行预测，得到预测结果后，把文本按照“类别“—“标题”—“正文”的格式存入数据库。在系统的首页可以进行展示。

按标签搜索文章/标题

按标签搜索文章/标题功能展示的是文本多分类的应用，模拟的是假如管理员只有一些没有标签的文本（假设这些文章不存在敏感词），但是需要获得对应标签下的所有文章的操作。我们把这个功能分为两步：先分类，再搜索。我们假设待分类文本存储在另一张表中。我们按行读取文本，执行文本分类函数，然后把获得对应标签的文本按照“类别“—“标题”—“正文”的格式存入数据库。在搜索时，我们从前端获取要搜索的标签，然后在数据库中按“类别”条目进行搜索，获得对应的结果展示在前端中。而管理员也可以选择搜索文章内容或是搜索标题，相对应的就是长文本和短文本的分类。

新增标签分类

新增标签分类功能是为了应对已有的类别不能满足实际应用的需要的情况，我们需要根据实际情况新增标签。但是由于分类器是一个深度学习模型，不经过训练（即微调）是无法处理新标签的文本的。因此，我们需要在原来的数据集中加入新标签下的文本，对BERT模型进行重新微调。当然由于训练时间长，训练条件要求苛刻，所以我们不展示具体的训练过程，只针对微调阶段的数据预处理过程进行展示。为了保证数据集的质量，获得高准确率的分类模型，我们不能单纯的把新文本插入原数据集中，需要打乱训练集数据的顺序。由于本实验的训练集数据量为短文本10万条，长文本12538条，故选择长文本训练集进行添加新标签的操作，简化系统的流程。在旧模型中无法分类的文本，在训练后的新模型中能够得到准确的分类，就实现了新增分类标签的效果。