Transformer下含谐音的恶意评论识别

摘要

含谐音的恶意评论识别网络ark采用$Transformer$的多头注意力机制和$encoder$, $decoder$架构，通过将一个评论的文本分为`[原始文本,拼音文本，拼音首字母文本]`三通道，来应对互联网评论下`谐音恶意评论`的情况，通过对不同的通道做注意力机制在当前文本中提取每个通道最重要的信息，再将三个通道融合后进一步提取信息，最后得到识别结果。

ark神经网络采用纯注意力机制，相对于传统的NLP工作采用的循环神经网络，注意力机制不仅可以并行计算，还可以更好的注意长文本的前后文关系，且深层的注意力机制网络解决了循环神经网络过深后梯度消失导致训练困难的问题。

关键字：恶意评论识别，多模式识别，多头注意力机制，Transformer，残差连接

第一章 绪论

1.1 研究背景

随着网络技术的飞速发展和网络文化的蓬勃生长，网民接触到的网络生活越来越多样化，光怪陆离的信息带来的是思考上的缺失。或许在某一时刻，对于某些刻意引导的恶意观点，在不假思索的认同后会对某个个体带来毁天灭地的伤害。

尤其对于现在层出不穷大大小小的网暴事件，背后的导火线往往是一句脱口而出的恶语相向，一句恶意是小，拥护的人多了便成了暴乱。如果网络规章可以再严格点，如果审核机制再智能点，如果恶意的传播再慢一点，或许我们就可以拯救一面真相，拯救一颗被伤害的心灵，甚至拯救一个家庭。

1.2 研究意义

如今网暴是最难定罪的一种犯罪，如果可以通过实现对网络交流时对其中恶意的识别，我相信网络乃至整个社会会更加健康和谐。

1.3 国内研究现状

当前不同平台如抖音，B站，贴吧等均采用的构建恶意关键词词表和人工举报监督机制对恶意评论进行处理。

构建词表的缺陷在于，只有一个评论里的词出现在词表里才可以识别，且词的出现与前后文无关。由于互联网的不断发展，这样的机制需要不断更新词表，且容易产生误判, 如当词表里存有`妈的`关键词，就会导致`妈的身体越来越差了`被错误识别，且当恶意词用谐音字代替时，词表完全无法识别，如用户用`挠摊`代替`脑瘫`，这样的情况只能举报后人工识别

人工举报监督机制需要用户积极维护平台的环境，且举报后后台的工作人员需要快速的审理解决，这样的方法虽然提高了识别恶意评论的准确率，但是其中耗费的人力是巨大的。

第二章 数据集收集与处理

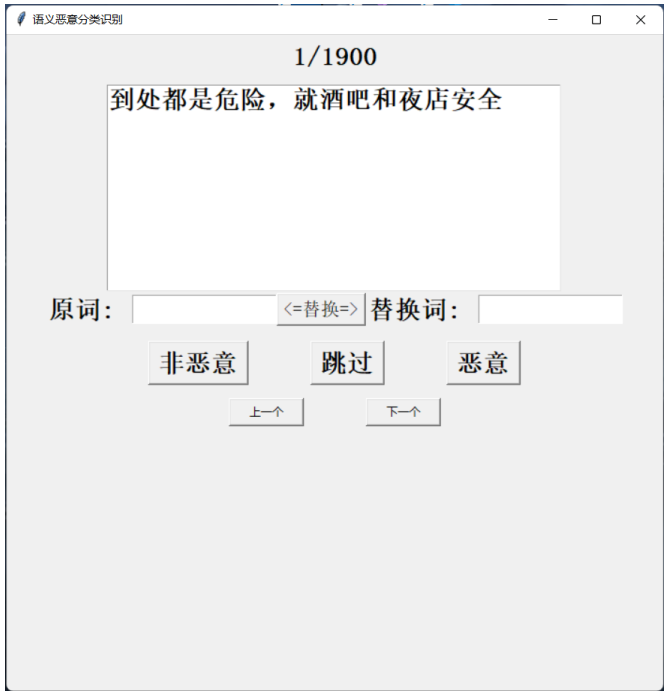
2.1 数据集爬取与收集

当前开源的数据集中，情感分析方面的数据集包括酒店好评差评，网络购物好评差评，中文冒犯性语言数据集等，这些数据集的正负面情绪不足以表现出当前网络环境的恶意或非恶意的评论，譬如一个差评可能是商品本身的质量差导致，并不代表这个评论是一个恶意抹黑的评论。

基于上述数据集的不足，该模型的大部分数据集通过网络爬虫对百度贴吧的评论回复爬取获取，少部分数据集采用网络脏话收集。

2.2 对爬取到本地的数据集进行分类

收集好数据集后，所有的数据集处于未分类的状态，这里所有的数据集采用纯人工分类的方法，对每一个评论浏览后标签恶意或非恶意。



其中在一个评论中会存在谐音字代替原字从而避开平台的恶意词表的情况，在人工分类期间会对谐音字手动修正，且标注修正点。当一个评论有n个修正点时，当前评论会被分为2^n个新评论，其中对于第i个新评论，它的第j个修正点是否修正取决于$(i >> j \& 1) == 1$条件是否成立。

2.3 汉字转拼音

采用开源python库：pypinyin实现将汉字转换为拼音

2.3.1 特性

- 根据词组智能匹配最正确的拼音。
- 支持多音字。
- 简单的繁体支持，注音支持，威妥玛拼音支持。
- 支持多种不同拼音/注音风格。

2.3.2 安装

```
pip install pypinyin
```

2.4 数据集增强

文本数据增强通常采用：1. 谐音字替换，2. 近义词替换，3. 翻译为不同语种再翻译回来。这里主要采用方法1选择一些字，替换为它的谐音字。

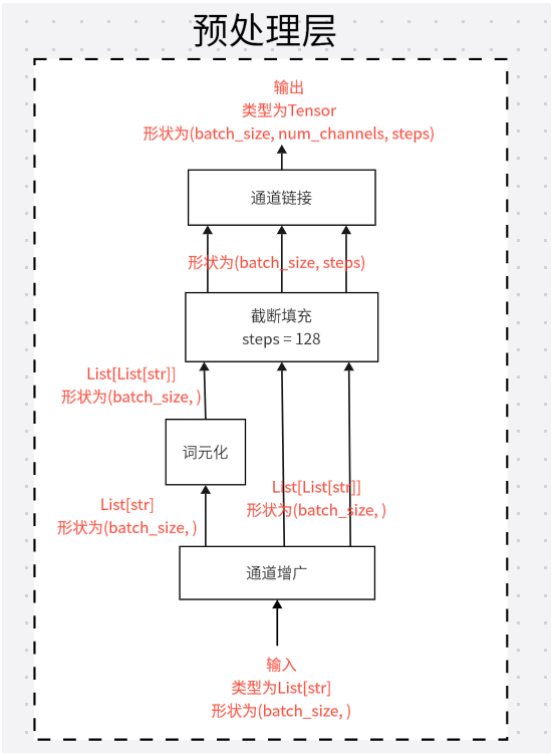
对于所有分类完成的数据集，随机选择40%的数据进行数据增强，对于每个被选中的数据，它的每个字有30%的概率被替换为它的谐音字。根据实际实验得出，每个字被修改的`概率不宜过大`，否则最后生成的句子缺乏前后文有具体意义的字词的依赖，导致整个句子缺乏实际含义。

2.5 完整数据集

	恶意	非恶意
训练集	5000	3500
测试集	600	500

第三章 搭建神经网络结构

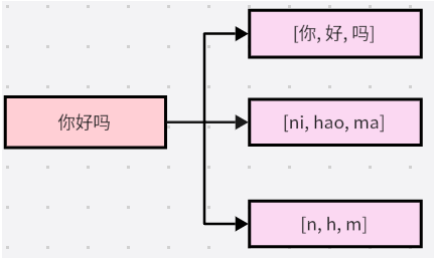
3.1 预处理层



在将输入传入模型前需要先将文本进行预处理，由于模型采用的是小批量训练的方案，预处理层期望传入一个`list`，其中`list`里是用于训练或预测类型为`str`的文本，`list`的长度为`batch_size`。

3.1.1 通道增广

进入预处理层后会先通过$pypinyin$得到拼音文本和拼音首字母文本， 如输入`["你好吗", "你好"]`得到`[["ni", "hao", "ma"], ["ni", "hao"]]`和`[['n', 'h', 'm'], ['n', 'h']]`，则此时我们得到了三个通道，分别是`[原文本，拼音文本，拼音首字母文本]`。



3.1.2 词元化

在NLP的任务里需要在预处理截断将句子词元化，传统的词元化是对句子进行分词，但是这样的方式强烈依赖于分词器的准确程度，如果分词器没有很好的将句子分词则会导致训练的词元无法良好的表达本意，且对于谐音任务下，分词器很可能因为句子里使用了谐音字导致分词错误，如`修勾可爱`会被分词为`修 勾 可爱`，而原意想表达的`小狗 可爱`很可能因为分词错误而增加训练成本。

并且由于词的数量远多于字的数量，采用分词会导致词表巨大，同时在后面训练的时候$Embedding$层巨大。

于是模型采用的是将每个字词元化，这样不仅减小了词表的大小，而且通过每个字附近的字判断它的意思使得上下文预测更加准确强键。

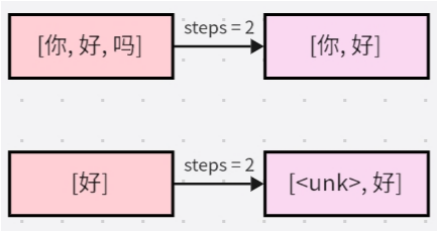
在通道增广的操作中，拼音文本和拼音首字母文本已经实现了词元化，则只需要对原文本词元化即可。如`["你好吗", "你好"]`词元化后的结果为`[["你", "好", "吗"], ["你", "好"]]`

3.1.3 截断填充

由于需要小批量的训练，每个小批量里每个句子的长度需要相同，即在预处理阶段需要定义一个时间步`steps`，表示一个句子的长度若超过`steps`，超过的部分需要截断，若一个句子的长度小于`steps`，缺少的部分需要填充。

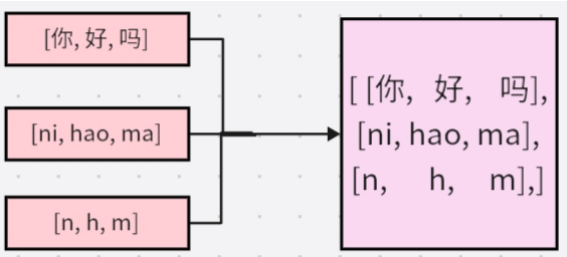
- 对于截断，丢弃在句子中下标$>=$ $steps$的字符
- 对于填充，使用特殊词元`<unk>`前向填充

如对`[["你", "好", "吗"], ["好"]]`执行$steps = 2$的填充截断，结果为`[["你", "好"], ["<unk>", "好"]]`，不难看出截断会丢失部分信息，为此可以将$steps$设置大一些，尽可能的保证信息不会丢失，同样若 $setps$过大会导致一句话过长，增加训练时间，在当前模型中$steps = 128$。

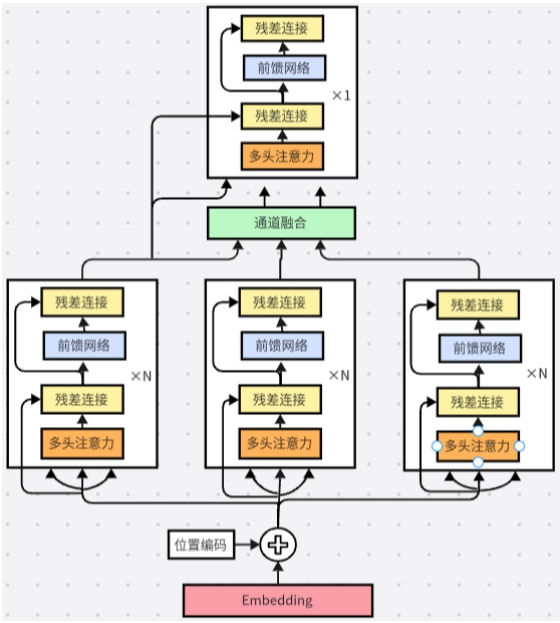


3.1.4 通道链接

经过上述操作后会得到三个形状为`(batch_size, steps)`的`List[List[str]]`， 通过预先得到的字表，将每个字映射到数字，并构造为`Tensor`类型的变量，将三个`Tensor`在`batch_size`和`steps`之间的维度链接在一起，构建形状为`(batch_size, num_channels, steps)`的`Tensor`



3.2 encoder架构



3.2.1 Embedding嵌入

在预处理层的输出里，最后得到一个形状为`(batch_size, num_channels, steps)`的Tensor，Tensor的每个值表示一个词元到数字的映射，现在需要将数字向量化，采用$Embedding$层。$Embedding$是一个简单的查找表，用于存储固定字典和大小的嵌入。此模块通常用于存储词嵌入并使用索引检索它们。 模块的输入是索引列表，输出是对应的词嵌入向量。

将每个词向量化后输出的形状为`(batch_size, num_channels, steps, hidden_size)`，其中$hidden_size$为向量的长度

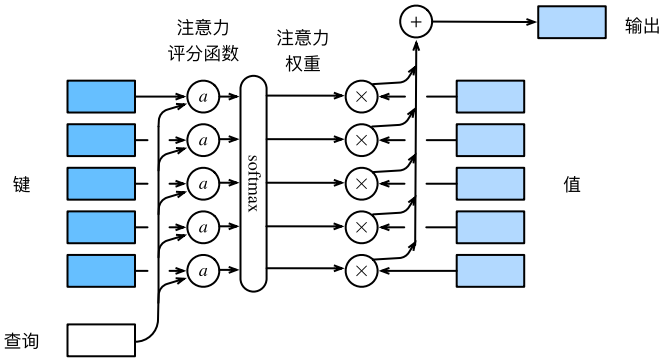
3.2.2 位置编码

在处理词元序列时，循环神经网络是依次循环处理每个词元，注意力机制为了并行计算放弃了顺序操作，这导致注意力机制不会关注一个词元出现的位置以及它附件的词元的分布。为了保留序列的位置信息，需要给注意力机制的输入添加位置编码来注入相对或绝对位置信息。为了保证位置的相对性，这里采用周期函数$sin\ cos$注入位置编码。

对于每个句子

- 第 i 个词元的第 $2\times j$ 个特征的位置编码为：$p_{i,2j}=\sin(\frac{i}{10000^{\frac{2j}{d}}})$
- 第 i 个词元的第 $2\times j+1$ 个特征的位置编码为：$p_{i,2j+1}=\cos(\frac{i}{10000^{\frac{2j}{d}}})$

3.2.3 注意力机制



注意力机制需要传入一个$query, key, value$三个Tensor，其中key和$value$一一对应

- $query$形状为`(batch_size, num_query, query_size)`
- key形状为`(batch_size, num_key_value, key_size)`
- $value$形状为`(batch_size, num_key_value, value_size)`

注意力评分函数：

$$\alpha(query, key_i) = softmax(a(query, key_i)) = \frac{\exp(a(query, key_i))}{\sum_{j=1}^{num_{key_value}} a(query, key_j)}$$

对于每个$query$，计算其与每个key_i的分数$score_i = \alpha(query, key_i)$，通过`sotfmax`使得
$\forall score_i \quad 0 \leq score_i \leq 1 \cup \sum_{i=1}^{n} score = 1$

最后的输出为每个query与value的加权和，即

$$f(query, (key_1, value_1) \ldots, (key_m, value_m)) = \sum_{i=1}^{m} \alpha(query, key_i) \times value_i$$

3.2.4 多头注意力机制

在实践中，当给定相同的$query, key, value$的集合时，我们希望模型可以基于相同的注意力机制学到不同的模式，为此与其使用单一的注意力机制，不如使用h个不同的注意力机制学习不同的模式，最后将学习到的h个模式融合在一起作为输出。

$$h_i = f(W_{iq} \times query, Wik \times keys, Wiv \times value)$$

$$output = W_o \times [h_1 \ldots, h_h]$$

3.2.5 多通道多头注意力机制与通道融合

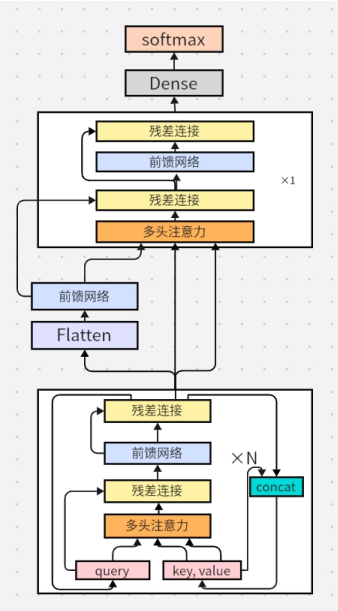
对于每个通道，通过多头注意力提取重要信息，在n层的多头注意力机制后三个通道的输出形状不变，均为`(batch_size, steps, hidden_size)`，最后三个输出在$steps$维度融合，得到形状为`(batch_size, steps * 3, hidden_size)`的Tensor。

在做注意力评分时，对于每个输入的句子，$step$维度表示num_key_value维，通过将三个通道在$steps$维融合，相当于增加了每个小批量的$key - value$对，当该输入再作为$key - value$传入多头注意力时，$query$可以同时学到三个通道的信息。

3.2.6 多模式学习

当多个通道在$steps$维度融合后，意味着每个小批量的$key - value$对包含了多个通道的信息，以原文本通道的输出作为$query$，融合信息作为$key - value$进行一层多头注意力机制提取关键信息，最后输出为形状为`(batch_size, steps, hidden_size)`

3.3 decoder架构



3.3.1 可记录历史的Transformer

$decoder$的输入来自$encoder$的输出，形状为`(batch_size, steps, hidden_size)`。此时每个小批量里都包含了`原始文本`，`拼音文本`，`拼音首字母文本`的信息，接下来需要在当前信息中继续寻找需要被注意的重要信息。

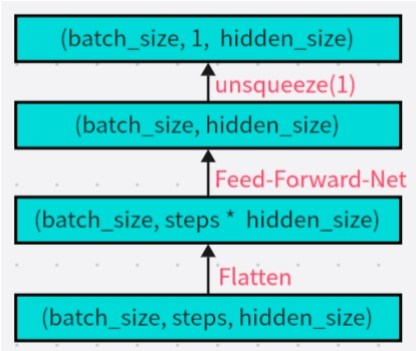
在$decoder$的$Transformer$网络中，下一层网络的$query$来自于当前网络的输出，下一层网络的$key-value$来自于当前网络的输出和当前网络的$key-value$在$steps$维度链接的结果。如此一来，每层的$key-value$为历史每一层的$key-value$在$steps$连接的结果，即每层的$key-value$保留了历史每一层$key-value$的信息。

如此可以保证即使模型训练的不好，由于有历史信息，模型也不会训练的越来越差。

3.3.2 来自全文本的query

在之前的多头注意力机制中，每个小批量的每个$query$代表的是对每个词元信息的查询，对于需要理解一个文本的信息，仅通过一个词元信息判断整个文本是不可取的，于是需要将一个文本的所有词元信息融合到一起。

在收到记录历史的$Transformer$输出后，将其输出的每个小批量拉伸为一维，并进入一个前馈网络融合每个steps的信息。



最后输出表示每个小批量有一个查询，且查询包含了整个文本的信息。

3.3.3 基于全文的注意力

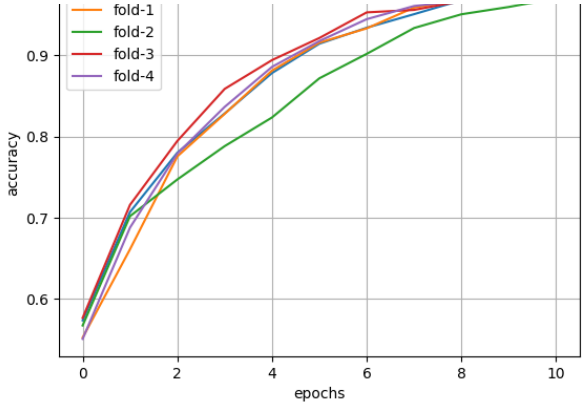
得到来自全文本的$query$后，将其与记录历史的$Transformer$做注意力机制，每个小批量的$query$只有一个查询，且查询的信息来自于整个文本，$key-value$的信息来自于历史上所有被关注到的信息，通过从全文注意所有信息，可以得到每个小批量所包含的信息。

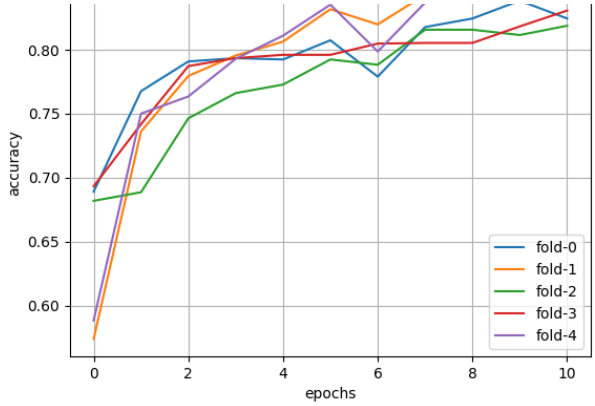
最终采用一个线性层，将每个小批量的包含的信息映射到类别空间，即可完成对每个小批量的语义识别。

第四章 评估指标

4.1 k折交叉验证

为了验证我们模型的训练精度，我们采用了k折交叉验证方法。k折交叉验证是一种常用的验证技术，它将数据集分割成k个相似的子集，然后将模型在这k个子集上进行k次训练和验证。在每一次训练中，其中一个子集被用作验证集，而其余的k-1个子集被用作训练集。通过这种方式，我们可以对模型在不同数据集上的性能进行评估，从而更加客观地评估其泛化能力。最终，我们将k次验证结果的平均值作为我们模型的训练精度。这种方法可以有效地减少由于数据集划分不均匀而引起的偏差，并且能够更好地利用数据集中的信息，从而得到更加可靠的模型评估结果。

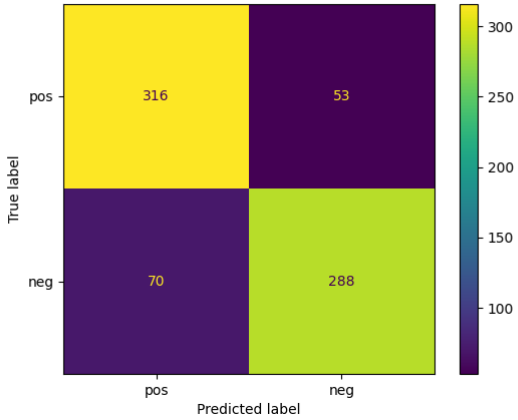




4.2 混淆矩阵验证模型准确率

为了更全面地评估我们模型的性能，我们使用了混淆矩阵来验证其准确率。混淆矩阵是一种用于评估分类模型性能的表格，它将模型的预测结果与真实标签进行对比，从而提供了关于模型分类能力的详细信息。

在混淆矩阵中，行代表真实类别，列代表模型预测的类别。每个单元格的数值表示模型将一个真实类别预测为某个预测类别的次数。基于混淆矩阵，我们可以计算出各种性能指标，包括准确率、精确度、召回率和F1分数等。



在这个示例中，我们有一个二分类问题，从混淆矩阵中我们可以看到，模型将316个真实阳性样本正确分类为阳性，将288个真实阴性样本正确分类为阴性。然而，它误将53个真实阳性样本分类为阴性，以及误将70个真实阴性样本分类为阳性。

4.2.1 准确率 （Accuracy）

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{316 + 288}{316 + 288 + 53 + 70} = 0.831$$

4.2.1 精确度（Precision）

$$Precision = \frac{TP}{TP + FP} = \frac{316}{316 + 53} = 0.856$$

4.2.2 召回率（Recall）

$$Recall = \frac{TP}{TP + FN} = \frac{316}{316 + 70} = 0.819$$

4.2.3 F1 分数（F1 Score）

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times 0.856 \times 0.819}{0.856 + 0.819} = 0.837$$

在上述公式中，(TP)、(TN)、(FP)和(FN)分别代表混淆矩阵中的真阳性、真阴性、假阳性和假阴性的数量。

第五章 结束语

本研究针对互联网环境中日益突出的含谐音恶意评论问题，提出了一种基于Transformer架构的多模式识别方法。通过采用多头注意力机制和三通道输入（原始文本、拼音文本、拼音首字母文本）的方式，本模型能够有效识别并处理网络评论中的谐音恶意内容。相较于传统的循环神经网络，本模型的注意力机制不仅提高了计算效率，还增强了对长文本前后文关系的捕捉能力，同时避免了深层网络训练中的梯度消失问题。

在数据集的收集与处理方面，本研究通过网络爬虫技术从百度贴吧等平台获取了大量实际评论数据，并通过人工分类的方式确保了数据的质量和准确性。通过对数据集进行增强，如谐音字替换等策略，进一步提升了模型的泛化能力和鲁棒性。

在模型评估方面，本研究采用了k折交叉验证和混淆矩阵的方法，全面地验证了模型的性能。评估结果表明，本模型在恶意评论识别任务上取得了令人满意的准确率、精确度、召回率和F1分数，显示出良好的应用前景。

总体而言，本研究的恶意评论识别模型为网络环境的净化提供了一种有效的技术支持。未来，我们将进一步优化模型结构，探索更多的数据增强技术，并尝试将本模型应用于更广泛的自然语言处理任务中，以促进网络空间的健康发展，维护网络环境的和谐与秩序。同时，我们也期待本研究能够为相关领域的研究者提供有益的参考和启示。