基于深度学习的网络恶意评论识别

项目成员：孔祥毅 2020111498 谢佳殷 2020111108 谢嘉薪 2020111142

1. 项目选题

1.1. 选题背景

如今，随着科技时代的到来，人们在发达的互联网背景下往往倾向于利用方便的电子设备在网络中发表各种各样的言论和表达自身的情感。因此从众多的意见中也产生了海量的数据。但是，其中不乏暗含着具有充满威胁性的甚至报复性质的恶意评论。如果这些评论能够被识别出来并处理，我们就可以拥有一个更安全、干净的互联网环境，因此，本项目采用来自维基百科谈话页面编辑的评论数据集设计了恶意评论的文本分类任务，尝试对恶意文本进行识别。

国内对于恶意文本的研究不是很多，仅有杨金灵（2022）通过建立基于词向量集成与数据增强的恶意评论分类模型识别恶意文本，但有不少基于文本挖掘技术展开的对恶意软件、恶意代码的研究，如张涛等（2019）采用经典的神经概率模型 Doc2Vec 从细粒度层面分析恶意软件家族的行为模式等。

基于以上启发，本小组尝试建立针对恶意评论识别的分类器。

1.2. 数据集与研究问题

本文数据集来自 Kaggle 上的公开竞赛 Jigsaw Multilingual Toxic Comment Classification[1]，该数据集来自维基百科谈话页面编辑的评论数据集，来源可靠且相对权威，包含训练集和测试集，共 226728 条评论作为训练集，评论为全英文文本，共有 6 个标签：toxic、gevere_toxic、obscene、threat、insult、indentity_hate；共 63812 条评论作为测试集，评论为非英文文本，标签为 toxic，本文研究的目标变量也为 toxic。数据集部分内容如下表所示：

表 1 评论数据集示例

id	comment_text	toxic
00040093b2687caa	alignment on this subject and which are contrary to those of DuLithgow	0
003217c3eb469ba9	Hi! I am back again! Last warning! Stop undoing my edits or die!	1
00031b1e95af7921	Your vandalism to the Matt Shirvington article has been reverted. Please don't do it again, or you will be banned.	0

数据集质量较高，但由于计算机算力限制，且希望专注于模型的建立与评估，本文从原始训练集中随机抽取了 5000 条全英文训练数据，并按 8:2 的比例划分训练集和测试集。主要基于深度学习对文本分类，判别是否为恶意文本。

1.3. 程序设计使用的语言和框架

[1] 数据来源：https://www.kaggle.com/competitions/jigsaw-multilingual-toxic-comment-classification/data

本文程序设计中用到的语言为 Python 语言。所用框架为 TensorFlow，最初由 Google 开发，使用 Python 底层 API 封装了 C++源代码，使得用户可以更快地工作。Keras 是 TensorFlow 的一个模块化且可扩展的封装器，作为基于 TensorFlow 和 Theano 的深度学习库，是由纯 python 编写而成的高层神经网络 API，也仅支持 python 开发。它是为了支持快速实践而对 tensorflow 或者 Theano 的再次封装，让我们可以不用关注过多的底层细节，能够把想法快速转换为结果，目前已成为 TensorFlow 官方的高级 API。

具体使用的库以及其所实现的功能分别为：

numpy：科学计算

matplotlib、seaborn、plotly：绘图

pandas：数据整理

sklearn：常用的机器学习功能，如数据预处理、分类性能评估

tenserflow、keras：深度学习环境部署、神经网络搭建

tqdm：进度显示

2. 方法与模型

2.1. 整体框架

传统的文本分类方法，基本都是利用 TFIDF 提取词频以及词语间的 N-gram 信息作为特征，然后通过机器学习方法如逻辑回归、支持向量机等作为分类器，如 TFIDF-LR、TFIDF-NBSVM 都是传统文本分类方法。这些方法特征表达能力差，序列捕捉能力弱，很难深层次的表征文本信息。基于阅读文献和竞赛公开解决方案的启发，本小组聚焦深度学习方法，基于 TensorFlow 框架和 Keras 接口，采用 Simple RNN、基于 GloVe 进行词嵌入的 LSTM 和 GRU 模型进行分类，并通过 AUC、ROC 曲线、混淆矩阵等评价指标进行模型评估。

流程图如图 1 所示。

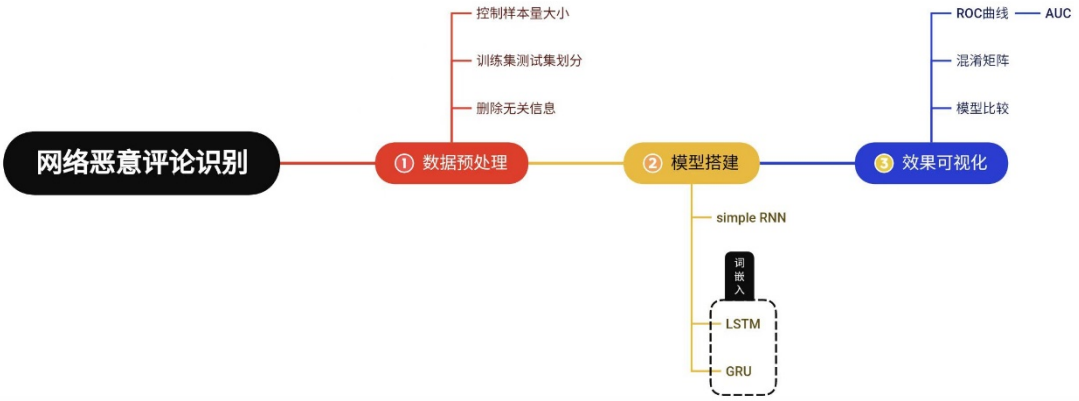

图 1 研究整体框架

2.2. 所用到的方法与模型的理论背景

1. Simple RNN 简单循环神经网络

在利用 N-gram 建立语言模型时，假设一个词出现的概率只取决于前面 N-1 个词，为了更好的捕捉句子信息，只能增加 N-gram 的长度，但当想处理任意长度的句子时，N 设为多少都不合适，而且模型的大小和 N 的关系是指数级的，N=4 时就会占用海量的存储空间了。

RNN 是传统前馈神经网络的扩展，是一类通过使用带有自反馈功能的神经元，处理任意长度时序数据的神经网络，相比于前馈神经网络，RNN 的输出不仅依赖于当前的输入，还与其过去一段时间的输出有关。因此 RNN 理论上可以往前（后）看任意多个词，且不论 RNN 接受多少个时刻的输入，或者输入序列有多长，不同时刻的计算都是用同一套参数，因此参数数量不会因序列长度的增加而增加。

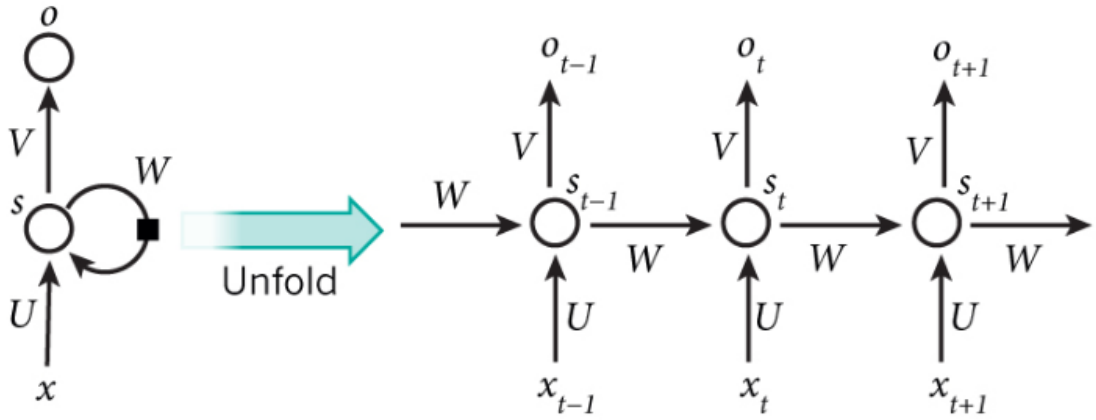

图 2RNN 结构示意图

如图 2 所示，这个网络在 t 时刻接收到输入 x_t 之后，隐藏层的值是 s_t，输出值是 o_t。关键一点是，s_t 的值不仅仅取决于 x_t，还取决于 s_{t-1}。可以用下面的公式来表示循环神经网络的计算方法：

$$o_t = g(Vs_t)$$
$$s_t = f(Ux_t + Ws_{t-1})$$

其中输出层是一个全连接层，也就是它的每个节点都和隐藏层的每个节点相连。V 是输出层的权重矩阵，g 是激活函数。隐藏层是循环层。U 是输入 x 的权重矩阵，W 是上一次的值s_{t-1}作为这一次的输入的权重矩阵，f 是激活函数。从上面的公式我们可以看出，循环层和全连接层的区别就是循环层多了一个权重矩阵 W，因此循环神经网络的输出值，是受前面历次输入值 x_t、x_{t-1}、x_{t-2}...影响的，这就是为什么循环神经网络可以往前看任意多个输入值的原因。

在训练 RNN 时，需要将 RNN 沿时间轴展开，并将所有 loss 在时刻的梯度反传，来更新网络的参数。这种将网络在时间上展开并进行误差反传的算法叫做 BPTT，基本原理同 BP 算法，包含以下三个步骤：

① 前向计算每个神经元的输出值；

② 反向计算每个神经元的误差项值，它是误差函数 E 对神经元 j 的加权输入的偏导数；

③ 计算每个权重的梯度。

最后再用随机梯度下降算法更新权重。

2. GloVe 词嵌入

词嵌入是将单词映射到低维的连续向量空间中，即单词被表示成实数域上的低维向量。获取词向量基本上有两种思路：

① 利用全局统计信息，进行矩阵分解（如 LSA）来获取词向量，这样获得的词向量往往在词相似性任务上表现不好，表明这是一个次优的向量空间结构；

② 利用局部上下文窗口单独训练，比如 word2vec 算法，但是统计信息作为有用的先验知识，没有被完全利用。

尽管 word2vec 在学习词与词间的关系上有了大进步，它却有很明显的缺点：只能利用一定窗口长度的上下文环境，即利用局部信息，没法利用整个语料库的全局信息。基于此，GloVe 模型诞生了，它的全称是 global vector，其目的就是对 word2vec 的这个缺点进行改进，运用到全局的语料信息。

定义共现矩阵为 X，其元素 x_{ij} 为在整个语料库中，单词 i 和单词 j 共同出现在一个窗口中的次数；

定义共现概率：

$$P(x_{ij}) = P(word_j|word_i) = \frac{x_{ij}}{x_i}$$

即所有以为w_i中心词的窗口中w_j出现的概率。

词的共现次数与其语义的相关性往往不是严格成比例，所以直接用共线性来表征词之间相关性效果不好，因此引入第三个词w_k，通过词之间的差异来刻画相关性。词之间的差异选择用两个词与同一个词的共现概率的比值来判断，即：

$$\frac{P(w_k|w_i)}{P(w_k|w_j)} = \frac{p_{ik}}{p_{jk}} = \frac{x_{ik}}{x_i} \Big/ \frac{x_{jk}}{x_j}$$

则与分母有关的比值比 1 大，与分子更有关的比值比 1 小，与分子分母都有关或都无关的比值维持在 1 附近。

设中心词的词向量为v_i, v_j，关联的背景向量为u_k，基本思想就是找到一个映射关系 g，使得：

$$g(v_i, v_j, u_k) = \frac{p_{ik}}{p_{jk}}$$

这意味着词向量与共现矩阵具有很好的一致性，也就说明词向量中蕴含了共现矩阵中所蕴含的信息。因此容易想到用二者的差方来作为代价函数：

$$J = \sum_{i,j,k}^{N} \left(\frac{p_{ik}}{p_{jk}} - g(v_i, v_j, u_k) \right)^2$$

再对$g(v_i, v_j, u_k)$进行处理，添加偏差项 $b_{i,j}$ 和权重项后，代价函数为：

$$J = \sum_{i,j}^{N} f(x_{i,j}) \left(v_i^T v_j + b_i + b_j - log(x_{i,j}) \right)^2$$

$$f(x) = \begin{cases} (x/x_{max})^{0.75}, & x < x_{max} \\ 1 \quad\quad\quad\quad , & x \geq x_{max} \end{cases}$$

训练后可以得到两个词向量矩阵v_i, v_j，两者之和即为所求词向量。

3. LSTM 长短期记忆递归神经网络

RNN 模型在反向传播过程中，当激活函数的梯度非常小时，权重因子会被

多次乘以这些小的梯度，因而越变越小，随着递归的深入趋于“消失”，小的梯度阻止早期层通过之前的梯度信息来进行权重调整。梯度消失问题这就是 RNN 模型只具备短期记忆的原因，当早期层因为梯度消失问题而停止学习的时候，在分析长句段时，之前读入的信息就可能不被纳入最后的考虑，因此我们有必要引入 LSTM 和 GRU 来解决短期记忆问题。

LSTM 与传统 RNN 的区别主要在于神经元内部结构不同，其神经元结构如图 3 所示。

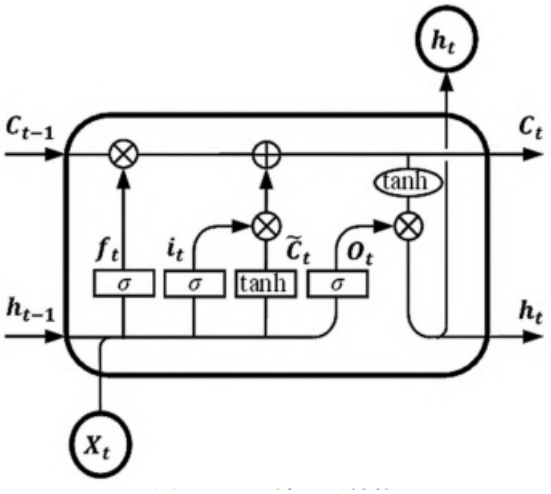

图 3 LSTM 神经元结构

（1） 细胞状态

上图中穿过神经元顶部的水平线即代表细胞状态，细胞状态穿过了 LSTM 链上的所有神经元，并且只有一些小的线性操作作用其上，因此整个链的信息在传递过程中可以保持相对稳定。

（2） 遗忘门

遗忘门决定细胞状态在该神经元中丢弃的信息。遗忘门的输入为上一时刻的隐层状态h_{t-1}和当前时刻的输入x_t，遗忘门的输出用于决定上一时刻的隐层状态中需要被遗忘的信息。遗忘门门控状态f_t的表达式为：

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

式中：σ为 Sigmoid 激活函数，W_f为f_t的权重矩阵；b_f为f_t的偏置。

（3） 输入门

在本时刻需要在细胞状态中存入的信息是由输入门决定的，其中包括输入门的门控状态i_t和候选值向量$\widetilde{C_t}$，具体的表达式为：

$$i_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

式中：W_t为i_t的权重矩阵，b_i为i_t的偏置；W_C为$\widetilde{C_t}$的权重矩阵；b_C为$\widetilde{C_t}$的偏置。

在信息经过输入门后，细胞状态在本时刻的神经元中也随之更新完毕：

$$C_t = f_t \cdot C_{t-1} + i \cdot \widetilde{C_t}$$

式中：C_t为本时刻的细胞状态；C_{t-1}为上一时刻的细胞状态。

（4） 输出门

输出门决定细胞的输出。输出门的输入也是上一时刻的隐层状态h_{t-1}和当前时刻的输入x_t，在通过 sigmoid 层后得到输出门的门控状态O_t，O_t与经过 tanh 函数处理更新后的细胞状态C_t相乘结果即为本时刻神经元的输出，同时该输出又会作为隐层状态输入下一时刻的神经元中，具体的表达式为：

$$O_t = \sigma(W_O \cdot [h_{t-1}, x_t] + b_O)$$
$$h_t = O_t \cdot tanh(C_t)$$

式中：W_O为O_t的权重矩阵；b_O为O_t的偏置；h_t为隐层状态。

4. GRU 门控循环单元

虽然 LSTM 利用门控单元在一定程度上解决了梯度消失和梯度爆炸问题，但是每个 cell 中都存在复杂的门控结构，使其收敛效率和训练效率低下。为解决这一问题，GRU 作为 LSTM 的改进模型被提出，单元结构如图 4 所示。

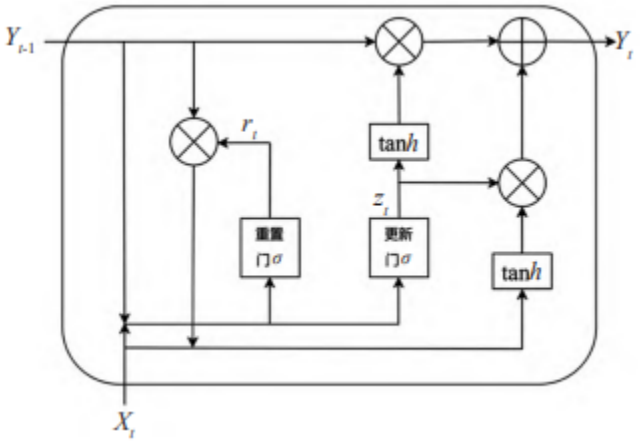

图 4 GRU 单元结构

由图 4 可以看出，GRU 将 LSTM 的遗忘门、输入门、输出门改进为重置门和更新门，实质上是将遗忘门和输入门改进为更新门。当前 cell 的输入受当前时刻的输入X_t和上一 cell 的输出Y_{t-1}控制。

（1） 重置门

重置门r_t负责将上个 cell 信息的权重比保存到当前 cell 状态，决定当前 cell 含有多少个 cell 信息。重置门的计算公式为：

$$r_t = \sigma\left(W_{xr}X_t + W_{hr}Y_{t-1} + b_r\right)$$

（2） 更新门

更新门z_t负责决定当前 cell 有多少信息的权重比能保持到下个 cell 中，决定下个 cell 含有多少当前 cell 信息。更新门的计算公式为：

$$z_t = \sigma\left(W_{xz}X_t + W_{hz}Y_{t-1} + b_z\right)$$

对于 LSTM，GRU 减少了单元结构中的门控数量，使得 GRU 网络收敛效率更快，在很好地捕捉到长距离上下文的同时大大节约了训练时间。

2.3. 编程实现

1. 预处理

包括数据的导入、筛选，训练与预测集的划分，以及一些自定义函数，用来方便之后的模型训练与评价。

```
strategy = tf.distribute.get_strategy() #分布式训练接口，本次不使用
train = pd.read_csv('jigsaw-toxic-comment-train.csv') #导入数据
train.drop(['severe_toxic','obscene','threat','insult','identity_hate'],axis=1,inplace=True) #删除无关信息
train = train.loc[:5000,:] #取前 5000 行做训练
train['comment_text'].apply(lambda x:len(str(x).split())).max() #获取最大评论字数
# AUC
def roc_auc(predictions,target):
    fpr, tpr, thresholds = metrics.roc_curve(target, predictions)
    roc_auc = metrics.auc(fpr, tpr)
    return roc_auc

# ROC 绘制函数
def acu_curve(predictions,target,title):
    fpr,tpr,threshold = metrics.roc_curve(target,predictions)
    roc_auc = metrics.auc(fpr,tpr)

    plt.figure()
    lw = 2
    plt.figure(figsize=(10,10))
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.3f)' % roc_auc) ###假正率为横坐标，真正率为纵坐标做曲线
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title+" ROC Curve")
    plt.legend(loc="lower right")

    plt.show()

# 混淆矩阵绘制函数
def cmplot(predictions,target):
    C = metrics.confusion_matrix(target, predictions)
    plt.matshow(C, cmap=plt.cm.Reds)
```

```
    for i in range(len(C)):
        for j in range(len(C)):
            plt.annotate(C[j, i], xy=(i, j), horizontalalignment='center',
verticalalignment='center')

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

#划分训练集与测试集
xtrain, xvalid, ytrain, yvalid = train_test_split(train.comment_text.values, train.toxic.values,
                                                  stratify=train.toxic.values,
                                                  random_state=42,
                                                  test_size=0.2, shuffle=True)
```

2. Simple RNN 模型

首先调用 keras 的接口完成分词和 padding，然后创建简单的循环神经网络模型，损失函数为二元交叉熵，激活函数为 sigmoid，优化器为 Adam。Adam 优化器是一种结合 AdaGrad 和 RMSProp 两种优化算法的优化，对梯度的一阶矩估计和二阶矩估计综合考虑来计算步长。调整合适参数训练后，可视化学习性能并记录 AUC。预测阈值设为 0.5，根据预测结果绘制混淆矩阵。

```
# 分词和 padding
from keras.utils import pad_sequences
token = text.Tokenizer(num_words=None)
max_len = 1500

token.fit_on_texts(list(xtrain) + list(xvalid))
xtrain_seq = token.texts_to_sequences(xtrain)
xvalid_seq = token.texts_to_sequences(xvalid)

xtrain_pad = pad_sequences(xtrain_seq, maxlen=max_len)
xvalid_pad = pad_sequences(xvalid_seq, maxlen=max_len)

word_index = token.word_index

%%time
with strategy.scope():
    # 没有预训练嵌入的单全连接层 RNN
    model = Sequential()
    model.add(Embedding(len(word_index) + 1,
                        300,
                        input_length=max_len))
    model.add(SimpleRNN(100))
```

```
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()

#训练
model.fit(xtrain_pad, ytrain, epochs=5, batch_size=64*strategy.num_replicas_in_sync)

#测试
scores = model.predict(xvalid_pad)
scores = [x[0] for x in scores]
print("Auc: %.2f" % (roc_auc(scores,yvalid)))

#模型效果评估
acu_curve(scores,yvalid,"SimpleRNN")

prediction = [round(x,0) for x in scores] #预测阈值为 0.5
cmplot(prediction,yvalid)

scores_model = []
scores_model.append({'Model': 'SimpleRNN','AUC_Score': roc_auc(scores,yvalid)})
```

3. 词嵌入

使用预训练的 GloVe 词向量数据创建我们的嵌入矩阵。这里使用了之前运行的结果。

```
## load the GloVe vectors in a dictionary:
# embeddings_index = {}
# f = open('D:\EDGEdownload\glove.840B.300d.txt\glove.840B.300d.txt','r',encoding='utf-8')
# for line in tqdm(f):
#         values = line.split(' ')
#         word = values[0]
#         coefs = np.asarray([float(val) for val in values[1:]])
#         embeddings_index[word] = coefs
# f.close()
# print('Found %s word vectors.' % len(embeddings_index))

#np.save('embeddings_index.npy', embeddings_index)

# Load
embeddings_index = np.load('embeddings_index.npy',allow_pickle=True).item()

#  为数据集中的词创建嵌入矩阵
embedding_matrix = np.zeros((len(word_index) + 1, 300))
for word, i in tqdm(word_index.items()):
```

```
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

4. LSTM 模型

使用 glove 嵌入，单全连接层的 LSTM 模型。建立嵌入层时，将嵌入矩阵作为权重传递给该层，而不是通过词汇进行训练，因此设置 trainable=False。除了用 LSTM 单元取代 SimpleRNN 外，模型其余部分与之前相同。最后绘制相应的 ROC 曲线和混淆矩阵。

```
%%time
with strategy.scope():

    # 使用 glove 嵌入的单全连接层 LSTM 模型
    model = Sequential()
    model.add(Embedding(len(word_index) + 1,
                        300,
                        weights=[embedding_matrix],
                        input_length=max_len,
                        trainable=False))

    model.add(LSTM(100, dropout=0.3, recurrent_dropout=0.3))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])

model.summary()

#训练
model.fit(xtrain_pad, ytrain, epochs=5, batch_size=64*strategy.num_replicas_in_sync)

#模型效果评估
scores = model.predict(xvalid_pad)
scores = [x[0] for x in scores]
print("Auc: %.2f" % (roc_auc(scores,yvalid)))

acu_curve(scores,yvalid,"LSTM")

prediction = [round(x,0) for x in scores]
cmplot(prediction,yvalid)

scores_model.append({'Model': 'LSTM','AUC_Score': roc_auc(scores,yvalid)})
```

5. GRU 模型

使用 glove 嵌入，双全连接层的 GRU 模型。最后绘制相应的 ROC 曲线和混淆矩阵。

```
%%time
with strategy.scope():
    # 使用 glove 嵌入的双全连接层 GRU 模型
     model = Sequential()
     model.add(Embedding(len(word_index) + 1,
                         300,
                         weights=[embedding_matrix],
                         input_length=max_len,
                         trainable=False))
     model.add(SpatialDropout1D(0.3))
     model.add(GRU(300))
     model.add(Dense(1, activation='sigmoid'))
     model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])

model.summary()

#训练
model.fit(xtrain_pad, ytrain, epochs=5, batch_size=64*strategy.num_replicas_in_sync)

#模型效果评估
scores = model.predict(xvalid_pad)
scores = [x[0] for x in scores]
print("Auc: %.2f" % (roc_auc(scores,yvalid)))

acu_curve(scores,yvalid,"GRU")

prediction = [round(x,0) for x in scores]
cmplot(prediction,yvalid)

scores_model.append({'Model': 'GRU','AUC_Score': roc_auc(scores,yvalid)})

scores_model
```

6. 可视化效果对比

使用表格和漏斗图对比三个模型效果的 AUC 得分。

```
# 可视化
results = pd.DataFrame(scores_model).sort_values(by='AUC_Score',ascending=False)
results.style.background_gradient(cmap='Blues')

# 漏斗图
fig = go.Figure(go.Funnelarea(
    text =results.Model,
    values = results.AUC_Score,
    title = {"position": "top center", "text": "Funnel-Chart of Sentiment Distribution"}
    ))
```

fig.show()

3. 实验结果

（1）训练结果

SimpleRNN 模型训练用时约 5 分钟，测试集 AUC 为 0.743。如果原假设是“不是恶意评论”，准确率为 0.908，召回率为 0.988，精确率为 0.916。

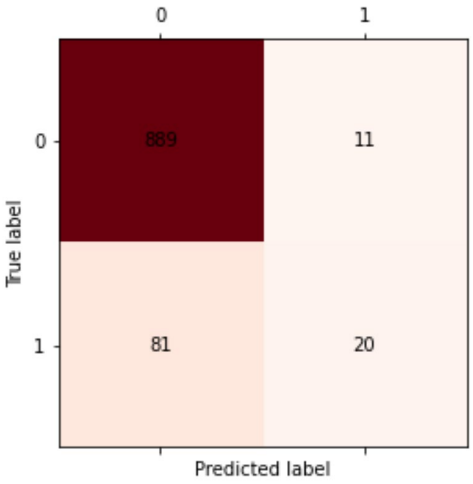

图 5 SimpleRNN 混淆矩阵

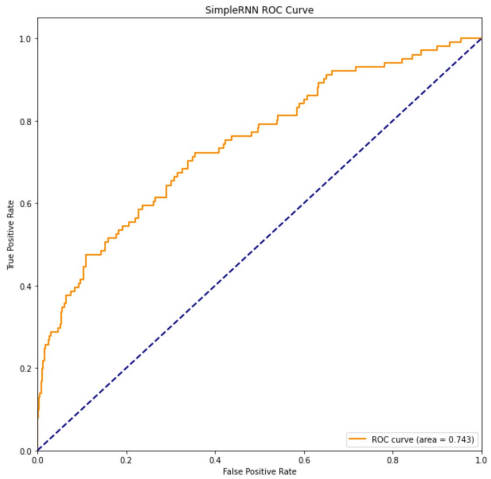

图 6 SimpleRNN ROC 曲线

LSTM 模型训练用时约 3 小时，验证集 AUC 为 0.946，准确率为 0.947，召回率为 0.98，精确率为 0.962。

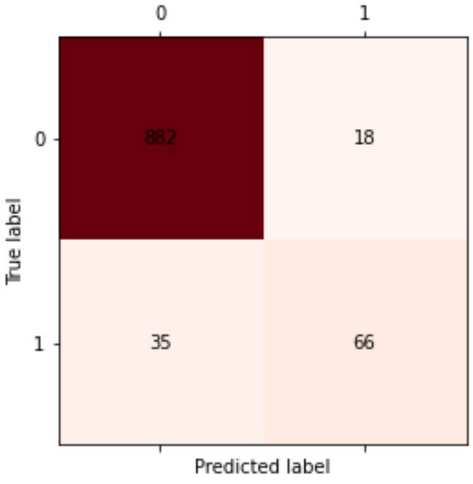

图 7 LSTM 混淆矩阵

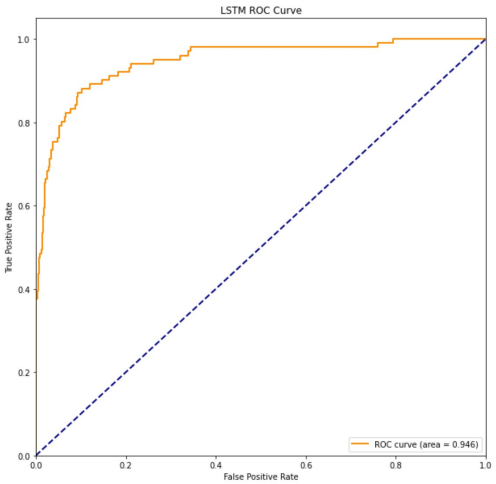

图 8 LSTM ROC 曲线

GRU 模型训练用时约 1.4 小时，验证集 AUC 为 0.962，准确率为 0.945，召回率为 0.982，精确率为 0.958。

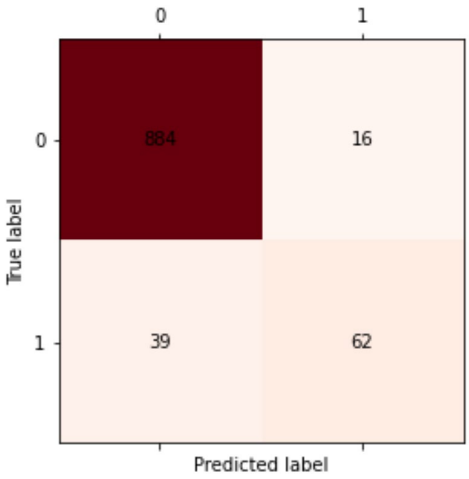

图 9 GRU 混淆矩阵

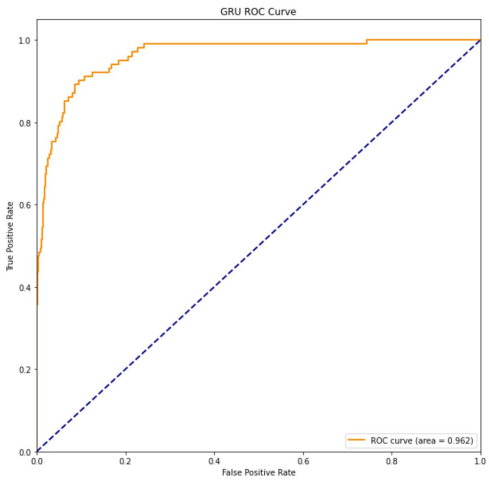

图 10 GRU ROC 曲线

（2）模型对比

从三个模型的 AUC 可以看出，GRU 的效果最好，LSTM 紧跟其后，simple RNN 的效果最差。另外，GRU 的模型训练时间比 LSTM 少了一半以上，这也证明了 GRU 的模型性能比 LSTM 要好很多。

	Model	AUC_Score
2	GRU	0.961694
1	LSTM	0.946172
0	SimpleRNN	0.743399

图 11 AUC 对比

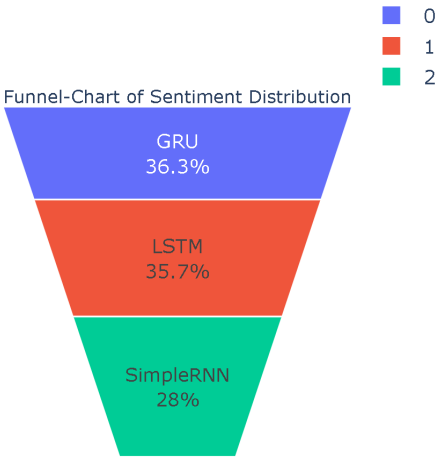

图 12 漏斗图（AUC）

4. 总结与讨论

4.1. 方法与结果的总结

本文运用简单循环神经网络、GloVe 词嵌入技术、LSTM 长短期记忆递归神经网络和 GRU 门控循环单元对维基百科谈话页面编辑的评论进行分类预测，在词嵌入技术的基础上进行的 LSTM 和 GRU 模型预测效果较好，GRU 的训练速度相较 LSTM 更快，后两种方法最后得到的 AUC 相对 simple RNN 有显著提升。不难理解该结果，解决了短期记忆的问题后，结合长句子序列靠前部分所携带的信息（类似于人类在做恶意评论识别时所提到的“语境”的概念）进行分析得到的结果，比缺失该部分信息得到的预测结果更准确，更不容易犯“断章取义”的错误。

4.2. 未能实现与有待改进的方面

未能实现的技术：

1. 对于 Bi-Directional LSTM 模型的尝试与效果评估。
2. 对于 seq2seq 模型架构的尝试。
3. 采用 BERT 等注意力机制的方法进行文档向量的产生。
4. TensorFlow 的分布式计算。

尚未解决的问题：

我们在训练这三个神经网络模型之后，都没有进行后续 fine-tuning 的工作，实际上无论是从神经网络结构上，还是从激活函数的选择、学习率的确定、是否采用正则化等方面来说都是值得调优的，但由于有限的时间和算力，以及现有足够令人满意的结果，我们没有解决上述问题。

5. 项目分工

孔祥毅：选题、SimpleRNN 模型搭建、GloVe 词嵌入实现、LSTM 模型搭建、GRU 模型搭建、可视化、报告撰写
谢佳殷：查找文献、数据预处理、LSTM 实践、可视化、报告撰写
谢嘉薪：查找文献、数据预处理、SimpleRNN 实践、可视化、报告撰写

6. 参考文献

[1] 杨金灵. 基于词向量集成与数据增强的恶意评论分类模型[J]. 科学技术创新, 2022(22):76-81.
[2] 张涛, 王俊峰. 基于文本嵌入特征表示的恶意软件家族分类[J]. 四川大学学报(自然科学版), 2019, 56(03):441-449.
[3] 黄磊, 杜昌顺. 基于递归神经网络的文本分类研究[J]. 北京化工大学学报(自然科学版), 2017, 44(01):98-104. DOI:10.13543/j.bhxbzr.2017.01.017.
[4] 梁军, 柴玉梅, 原慧斌, 高明磊, 昝红英. 基于极性转移和 LSTM 递归网络的情感分析[J]. 中文信息学报, 2015, 29(05):152-159.