Capstone Proposal

齐梓辰VIP2

问题描述

在这个项目中，需要解决的问题是文本多分类，文本来自于Wikipedia's talk page edits的评论，并且该任务是一个标签不平衡问题，每条数据可能对应不止一个标签。主要的处理方法有以下几种，下面提到的三种分类引用自 [1]:

- **MULTI-LABEL APPROACHES**：首先简单说一下multi-class和multi-label的区别，原文查看[2]。multi-class是每个class互斥，而multi-label则是每个label可能互相包含。而对于该问题，则是一个multi-label的问题，也就是每个sample可能复合多个label的标准。但是在[1]的文献中，发现大多数的研究都是集中在multi-class的研究，如果multi-label可以转化为multi-class的话那么将大大提高模型的准确率。
- **SHALLOW CLASSIFICATION AND NEURAL NETWORDS**: 恶毒评论识别的实现的方式是两种：一种是当作监督学习问题配合手动特征工程，另一种是深度神经网络。使用监督学习的方式需要加入手动定义的特征，而深度学习的方法则会自动学习必要的特征。在[1]的文献综述中可以看到，深度学习方法（如CNN和BiLSTM）的效果比较好，而监督学习的方法解释性更强。
- **ENSEMBEL LEARNING**：融合多种分类器的方法也是比较流行的做法。

解决方法总结

在文本分类问题中，主要使用的机器学习和深度学习方法。机器学习的方法主要是应用线性学习器，对处理好的文本进行学习和预测。为什么是线性模型？因为在处理文本时，文本会以稀疏矩阵的形式存储，而线性模型比较擅长处理稀疏模型。深度学习方法主要是CNN和LSTM。深度学习和机器学习都需要对文本进行处理，一般的处理方式包括词袋模型，N-gram，word-embedding等等。在这里，我将会对机器学习方法和深度学习方法做一个初步的总结。

- 文本预处理
  - Tokenization

```
import nltk
text = "this is Andrew's text, isn't it?"
# whitespace tokenizer对于it 和 it? 区别对待，实际上意思一样
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tokenizer.tokenize(text)
# Treebank会把一些不常见的组合忽视掉
tokenizer = nltk.tokenize.TreebankTokenizer()
tokenizer.tokenize(text)
# WordPunctuation 有时候会把没有意义的词分组，比如s，isn，t
tokenizer = nltk.tokenize.WordPunctTokenizer()
tokenizer.tokenize(text)
```

- Token normalization
  - Stemming (A process of removing and replacing suffixes to get to the root form of the word)

```
# 对于irregular form 无法处理
import nltk
text = "feet cats wolves talked"
tokenizer = nltk.tokenize.TreebankWordTokenizer()
tokens = tokenizer.tokenize(text)
stemmer = nltk.stem.PorterStemmer()
" ".join(stemmer.stem(token) for token in tokens)
 #不是词的所有形式都能处理，动词和名词是不一样的
lemmer = nltk.stem.WordNetLemmatizer()
" ".join(stemmer.stem(token) for token in tokens)
```

  - Lemmatization (Return the base or dictionary form of a word)
  - Normalizing capital letters
  - Acronyms

- 特征提取
  - 词的表示类型
    - 定义：传统的独热表示（ one-hot representation）仅仅将词符号化，不包含任何语义信息。如何将语义融入到词表示中？Harris 在 1954 年提出的分布假说（ distributional hypothesis）为这一设想提供了理论基础：上下文相似的词，其语义也相似。Firth 在 1957 年对分布假说进行了进一步阐述和明确：词的语义由其上下文决定（ a word is characterized by thecompany it keeps）。到目前为止，基于分布假说的词表示方法，根据建模的不同，主要可以分为三类：基于矩阵的分布表示、基于聚类的分布表示和基于神经网络的分布表示。尽管这些不同的分布表示方法使用了不同的技术手段获取词表示，但由于这些方法均基于分布假说，它们的核心思想也都由两部分组成：一、选择一种方式描述上下文；二、选择一种模型刻画某个词（下文称“目标词”）与其上下文之间的关系。
    - 基于矩阵分布：基于矩阵的分布表示通常又称为分布语义模型，在这种表示下，矩阵中的一行，就成为了对应词的表示，这种表示描述了该词的上下文的分布。由于分布假说认为上下文相似的词，其语义也相似，因此在这种表示下，两个词的语义相似度可以直接转化为两个向量的空间距离。常见到的Global Vector 模型（ GloVe模型）是一种对“词-词”矩阵进行分解从而得到词表示的方法，属于基于矩阵的分布表示。
    - 基于神经网络的分布表示，词嵌入（ word embedding）：基于神经网络的分布表示一般称为词向量、词嵌入（word embedding）或分布式表示（distributed representation）。神经网络词向量表示通过神经网络技术对上下文，以及上下文与目标词之间的关系进行建模。由于神经网络较为灵活，这类方法的最大优势在于可以表示复杂的上下文。在前面基于矩阵的分布表示方法中，最常用的上下文是词。如果使用包含词序信息的 n-gram 作为上下文，当 n 增加时， n-gram 的总数会呈指数级增长，此时会遇到维数灾难问题。而神经网络在表示 n-gram 时，可以通过一些组合方式对 n 个词进行组合，参数个数仅以线性速度增长。有了这一优势，神经网络模型可以对更复杂的上下文进行建模，在词向量中包含更丰富的语义信息。
  - Bag of words (词袋模型)

- 定义：Bag-of-words model （BoW model）忽略文本的语法和语序，用一组无序的单词（words）来表达一段文字或一个文档。根据下面两句话中出现的单词，构建一个字典（dictionary）。该字典中包含10个词，每个单词有唯一索引，注意它们的顺序和出现在句子中的顺序没有关联，根据这个词典，我们将上述两句话重新表达为下面的两个向量。这两个向量共包含10个元素, 其中第i个元素表示字典中第i个单词在句子中出现的次数. 因此BoW模型可认为是一种统计直方图 (histogram). 在文本检索和处理应用中, 可以通过该模型很方便的计算词频。

```
{"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6,
"football": 7, "games": 8, "Mary": 9, "too": 10}
'John likes to watch movies. Mary likes too.'
'John also likes to watch football games.'
[1, 2, 1, 1, 1, 0, 0, 0, 1, 1]
[1, 1, 1, 1, 0, 1, 1, 1, 0, 0]
```

- 语言模型
  - 定义：N-gram模型是一种语言模型（Language Model，LM），语言模型是一个基于概率的判别模型，它的输入是一句话（单词的顺序序列），输出是这句话的概率，即这些单词的联合概率（joint probability）
  - 公式：
    - $\mathbf{w} = (w_1 w_2 w_2 \dots w_k)$
    - Chain rule: $p(\mathbf{w}) = p(w_1)p(w_2|w_1)\cdots p(w_n|w_1 \dots w_{i-1})$
    - Markov Assumption: $p(w_i|w_1 \dots w_{i-1}) = p(w_i|w_{i-n+1} \dots w_{i-1})$
    - Bi-gram (n = 2): 如果一个词的出现仅依赖于它前面出现的一个词，那么我们就称之为 Bi-gram $p(S) = p(w_1 w_2 \cdots w_i) = p(w_1)p(w_2|w_1)\cdots p(w_k|w_{k-1})$
    - Tri-gram (n = 3): 如果一个词的出现仅依赖于它前面出现的两个词，那么我们就称之为 Tri-gram $p(S) = p(w_1 w_2 \cdots w_k) = p(w_1)p(w_2|w_1)\cdots p(w_k|w_{k-1}w_{k-2})$
    - MLE: 那么，如何计算其中的每一项条件概率 $p(w_k|w_{k-1} \cdots w_2 w_1)$ 呢？答案是极大似然估计（Maximum Likelihood Estimation，MLE），说人话就是数频数：$p(w_k|w_{k-1}) = \frac{C(w_{k-1}w_k)}{C(w_{k-1})}$
    - > 困惑度
      >
      > This is an optional reading about perplexity computation to make sure you remember the material of the videos and you are in a good shape for the quiz of this week.
      >
      > Perplexity is a popular quality measure of language models. We can calculate it using the formula:
      >
      > $$\mathcal{P} = p(w_{test})^{-\frac{1}{N}}, \text{ where } p(w_{test}) = \prod_{i=1}^{N+1} p(w_i|w_{i-n+1}^{i-1})$$
      >
      > Recall that all words of the corpus are concatenated and indexed in the range from 1 to NN. So NN here is ***the length of the test corpus**.** Also recall that the tokens out of the range are **fake start/end tokens** to make the model correct.

Check yourself: how many start and end tokens do we have in a trigram model?

Now, if just one probability in the formula above is equal to zero, the whole probability of the test corpus is zero as well, so the perplexity is infinite. To avoid this problem, we can use different methods of smoothing. One of them is Laplacian smoothing (add-1 smoothing), which estimates probabilities with the formula:

$$p(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i})+1}{c(w_{i-n+1}^{i-1})+V}$$

Note, that VV here is the number of possible continuations of the sequence w_{i-n+1}^{i-1}, so V is **the number of unique unigrams in the train corpus plus 1**. Do you see why? Well, we include the fake end token to this number, because the model tries to predict it each time, just us as any other word. And we do not include the start tokens, because they serve only as a prefix for the first probabilities.

Now, let's review the following task together.

Task:

Apply add-one smoothing to the trigram language model trained on the sentence:

"This is the cat that killed the rat that ate the malt that lay in the house that Jack built."

Find the perplexity of this smoothed model on the test sentence:

"This is the house that Jack built."

Solution:

We have n=3, so we will add two start tokens , and one end token .

Note, that we add **(n-1) start tokens,** since the start tokens are needed to condition the probability of the first word on them. The role of the end token is different and we always add ***just one end token**.** It's needed to be able to finish the sentence in the generative process at some point.

So, what we have is:

train: *This is the cat that killed the rat that ate the malt that lay in the house that Jack built*

test: *This is the house that Jack built*

Number of unique unigrams in train is 14, so V = 14 + 1 = 15.

Number of words in the test sentence is 7, so N = 7.

$$\mathcal{P} = p(w_{test})^{-\frac{1}{N}}, \text{ where } p(w_{test}) = \prod_{i=1}^{8} p(w_i|w_{i-2}w_{i-1}) = \prod_{i=1}^{8} \frac{c(w_{i-2}w_{i-1}w_i)+1}{c(w_{i-2}w_{i-1})+15}$$

All right, now we need to compute 8 conditional probabilities. We can do it straightforwardly or notice a few things to make our life easier.

> First, note that all bigrams from the test sentence occur in the train sentence **exactly once,** which means we have (1 + 15) in all denominators.
>
> Also note, that "is the house" is the only trigram from the test sentence that is not present in the train sentence. The corresponding probability is p(house | is the) = (0 + 1) / (1 + 15) = 0.0625.
>
> All other trigrams from the test sentence occur in the train sentence exactly once. So their conditional probabilities will be equal to (1 + 1) / (1 + 15) = 0.125.
>
> In this way, perplexity is (0.0625 * 0.125 ^ 7) ^ (-1/7) = 11.89.
>
> The quiz of this week might seem to involve heavy computations, but actually it does not, if you think a bit more :) Good luck!

        - 改进：
            - 开始和结束词：加上开始和结束
              $p(\mathbf{w}) = p(w_1|start)p(w_2|w_1)\cdots p(w_k|w_{k-1})p(end|w_k)$
            - n的选择取决于语料的大小
            - out-of-vocabulary word
                - 把oov词语替换成UNK (unknown)
                - 建立大的词典
            - Smooth
                - Laplacian Smoothing
                - Katz backoff
                - Interpolation Smoothing
                - Absolute dscounting
                - Kneser_Ney Smoothing (most popular)
        - 可以把高频率n-grams (and a等) 和低频率n-grams (罕见的词语) 去除
    - TF-IDF(词频-逆文本频率)：
        - 定义：
            - 词频 (TF)= 某个词在文章中出现的总次数/文章的总词数 或者：词频 = 某个词在文章中出现的总次数/文章中出现次数最多的词的个数。
            - 逆文档频率(IDF) = log(词料库的文档总数/包含该词的文档数+1)，为了避免分母为0，所以在分母上加1。
            - TF-IDF值 = TF * IDF。在此有：TF-IDF值与该词的出现频率成正比，与在整个语料库中的出现次数成反比，符合之前的分析。

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
tfidf = TfidfVectorizer(min_df = 2, max_df = 0.5, ngram_range=(1,2))
features = tfidf.fit_transform(texts)
pd.DataFrame(features.todense(),columns=tfidf.get_feature_names())
```

- 线性模型

-
- 深度学习模型
  - CNN
  - RNN(Reccurent neural network)
  - RNN(Recursive neural network)

> Jigsaw(前身为Google ideas）在kaggle平台上举办了一场文本分类比赛 **Toxic Comment Classification Challenge**， 旨在对于网络社区部分恶毒评论进行区分鉴别。
>
> 数据集下载链接：https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

数据或输入

数据包括人为标记的维基百科评论，这些评论按照恶毒程度分为了以下几个级别：

- `toxic`
- `severe_toxic`
- `obscene`
- `threat`
- `insult`
- `identity_hate`

项目提供的文件如下：

- **train.csv** - 训练集包括以二进制形式标记的标签
- **test.csv** -在测试集中，必须预测这些的评论恶毒程度的概率。为了阻止手工标记，测试集包含一些未包含在评分中的注释。
- **sample_submission.csv** - a sample submission file in the correct format
- **test_labels.csv** - 测试数据的标签；值为`-1`表示未用于评分；

数据集的分类标准缺乏一个明确的标准定义，恶毒程度的分类依靠人的喜好标记。

Class	# of occurance
Clean	143346
Toxic	15294
Obscene	8449
Insult	7877
Identity Hate	1405
Severe Toxic	1595
Threat	478

表格1

从上面的表格1可以看出，数据的分布非常不均匀。总共159571中有143345的评论被分为*clean*大约占90%，而*clean*不属于我们预测分类的6个class之一。而恶毒评论仅占10%左右，而在6个恶毒评论分类中*threat*仅占0.3%。

	Train	**Test**
数量	159671	159571
比值	51	49

表格2

从表格2中可以看出Train 和 Test的数据比值为51和49，还是比较合理的分布。

评估标准

评估标准总结

分类问题的粗略评估：准确率([Accuracy] [3])

	Predicted Possitive	**Predicted Negtive**
Positive	True Positives-1000	False Negatives-200
Negative	False Positives-800	True Negatives-8000

$Accuracy = \frac{1000+8000}{10000} = 90\%$

以上是准确率的定义和例子，但是准确率的问题也是非常明显。在一个分类任务中如果正样本占大多数，负样本占少数，那么准确率则很难给出准确的评估。例如，正样本900，负样本100，如果分类器预测全为正，那么准确率为90%。但是显然这个分类器一个负样本也没有检测出来。

分类任务的精确度量：查准率、查全率与 F 1

查准率：

- P=TP/(TP+FP)

查全率：

- R=TP/(TP+FN)

F1-score:

- 其中β>0度量了查全率对查准率的相对重要性。
- β>1的时候查全率有更大影响；
- β<1的时候查准率有更大影响；

ROC曲线（Receiver Operating Characteristic）

ROC曲线主要关注TPrate和FPrate。

- True positive Rate = True Posivives/All Posivives 在所有positive中分对了多少
- False positive Rate = False Posivives/All Posivives 在所有negtive中分错了多少

如果一个学习器的PR曲线被另一个学习器的PR曲线完全包住，则后者的性能优于前者。如果两个学习器的PR曲线相交，就不能着急下结论了，这个时候一般算两条曲线下面包裹的面积，面积大的很明显取得PR双高的概率要更高。但是这个不太容易估算。对此，有人设计了BEP，平衡点（Break-Even Point）。 BEP是指查准率=查全率时候的取值，也就是画一条直线f(x)=x，曲线和这条直线的交点就是BEP。毕竟BEP还是过于简化了些。更常用的方法是F1度量.

- Area under ROC-curve
- Area under precision-recall curve

参赛评估标准

由于参赛方的规定，所有的提交都要用mean ROC AUC（The area computed under the recerer operating characteristic curve from prediction score）。

基准模型

我将使用逻辑回归作为基准模型。因为逻辑回归作为基础的线性模型，计算迅速且善于处理稀疏矩阵。在构建特征时，我将会运用词袋模型和tf-idf，这样就会构建一个稀疏矩阵，所以运用线性模型处理是个很好的选择。具体的模型如下：

数据预处理：

- 文本处理：规范大小写，去除停用词和符号等等

特征工程：

- 词袋模型
- Tf-idf

构建分类器：

- 逻辑回归
- OneVsRest 分类器

评估：

- 准确率
- F1-score
- Area under ROC-curve
- Area under PR curve

项目设计

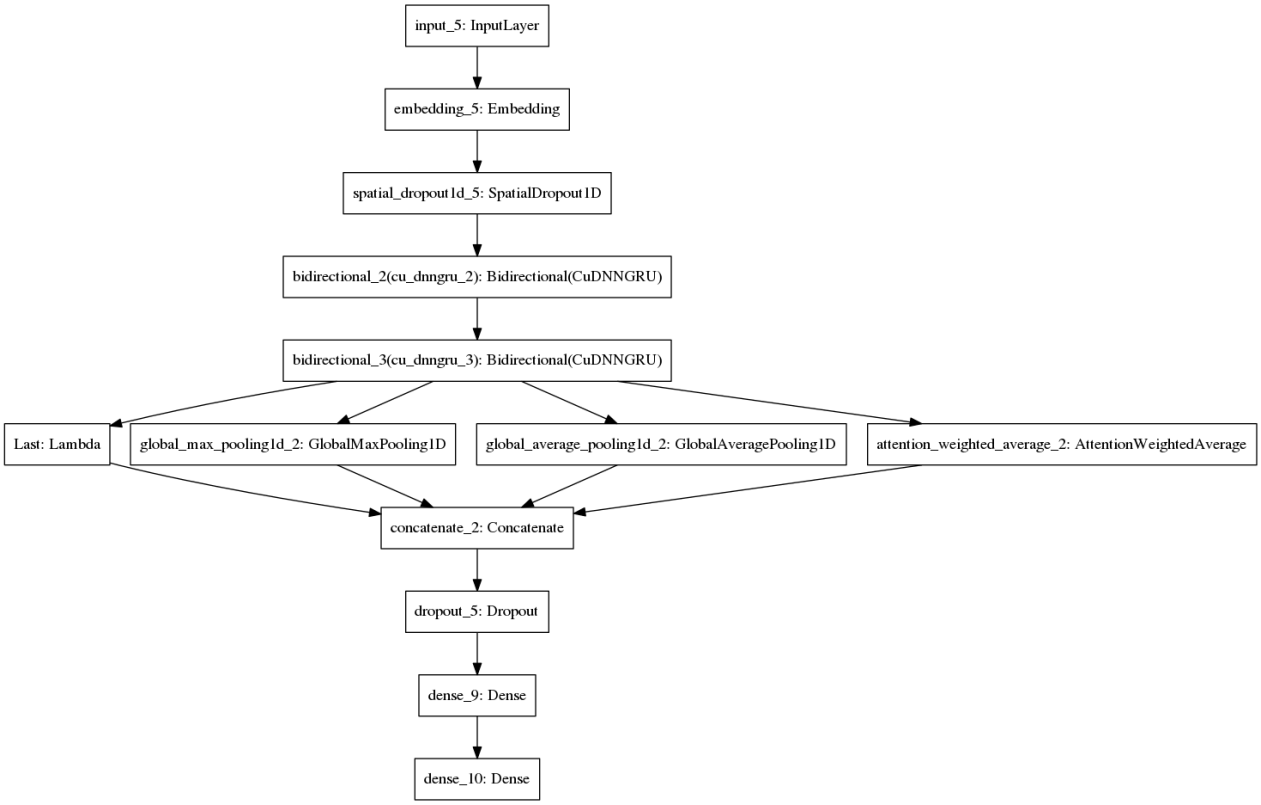
数据预处理：

- 转换大写为小写
- 把/(){}[]|@,;转换为空格
- 把^0-9a-z #+_删除

- 去除stopwords

模型:

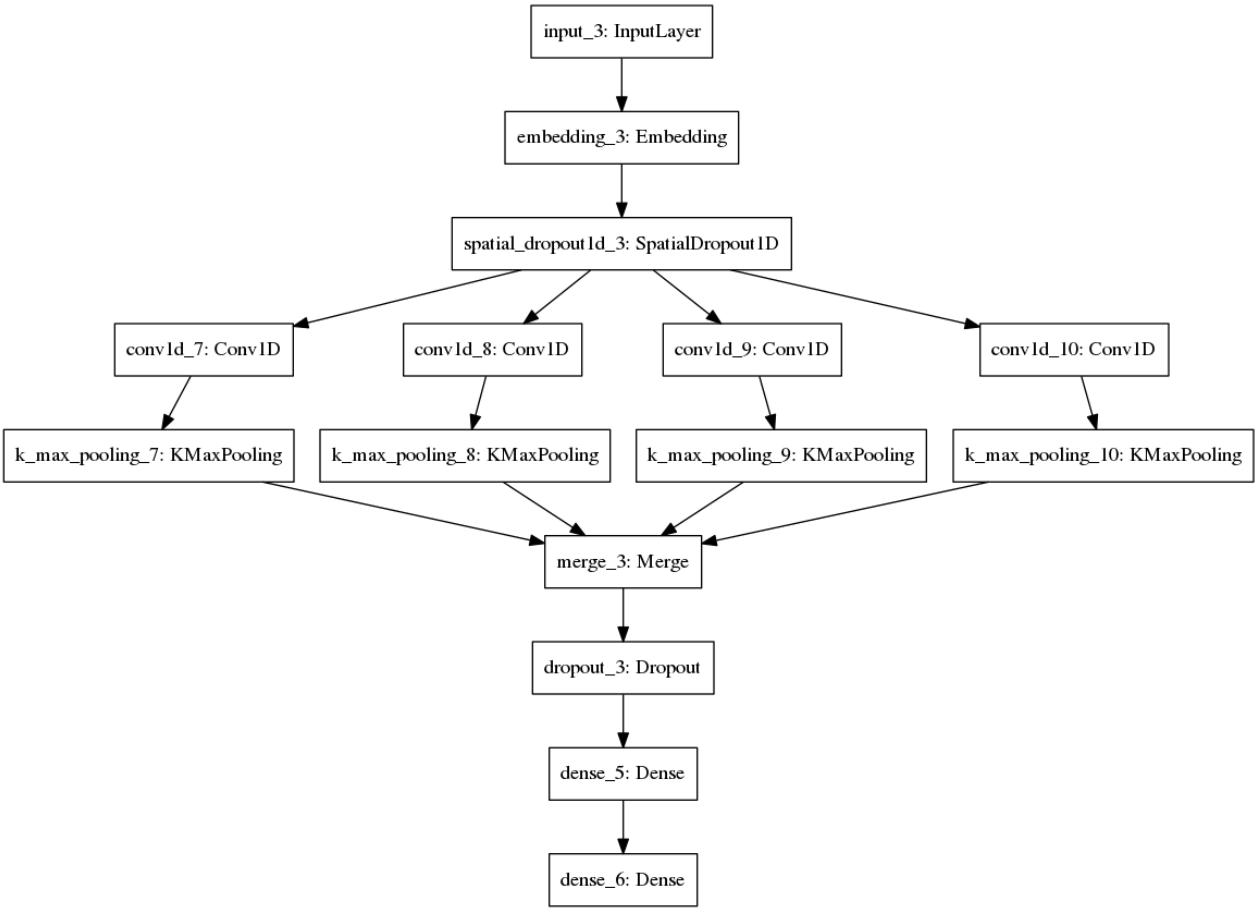
在 [4]中提到，RHN with recurrent dropout, DPCNN, VDCNN, HAN, Convolutional Attention Model 这些比较复杂的模型在test data中表现不错，但是在kaggle平台表现不佳。他推荐了两个比较好的模型一个是Pooled RNN 另一个是Kmax text CNN



Pooled RNN

这是一个比较经典的Text-RNN，我会尝试同[4]同样的思路调参数和设计，使用4个pooling layers。具体如下：

- Lambda layer, return the last hidden state
- Global max pooling
- Global average pooling
- Dot attention



而CNN模型主要参考[5]这篇文章。

- 对于cnn和rnn我会尝试使用word level 和 character level的模型，进行对比。
- 对于词嵌入，同样我会采取和Justin一样的设置Fasttext，Glove，Twitter三个模型会被使用。
    - 分别实验三种不同的embedding
    - 融合embedding

- 模型融合：最后对于两个模型试着采用lightgbm进行融合。

文献引用

[1]: van Aken, B., Risch, J., Krestel, R. and Löser, A., 2018. Challenges for Toxic Comment Classification: An In-Depth Error Analysis. *arXiv preprint arXiv:1809.07572*.

[5]:Young, T., Hazarika, D., Poria, S. and Cambria, E., 2018. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, *13*(3), pp.55-75.