# 机器学习工程师纳米学位毕业项目

# 毕设报告

# Jigsaw恶毒评论分类

马 良

2018年11 月31日

## 1. 定义

### 项目概述

Jigsaw(前身为Google ideas）在kaggle平台上举办了一场文本分类比赛，旨在对于网络社区部分恶毒评论进行区分鉴别。在该赛题中，你需要建立一个可以区分不同类型的言语攻击行为的模型，该赛题一共提供了toxic,severe_toxic,obscene,threat,insult,identity_hate这六种分类标签，你需要根据提供的训练数据进行模型训练学习。

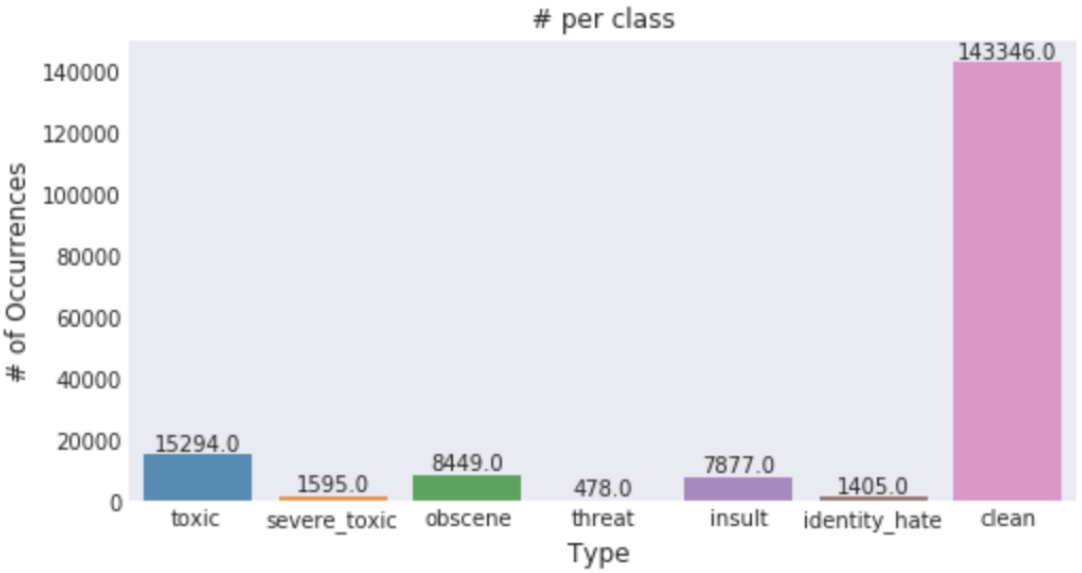
我们这里要求你使用Kaggle端的数据集，其由Train,Test两部分构成，你需要通过在Train数据集上进行验证集划分、建模，在Test数据集上进行测试，并且提交到Kaggle进行测评。

### 问题陈述

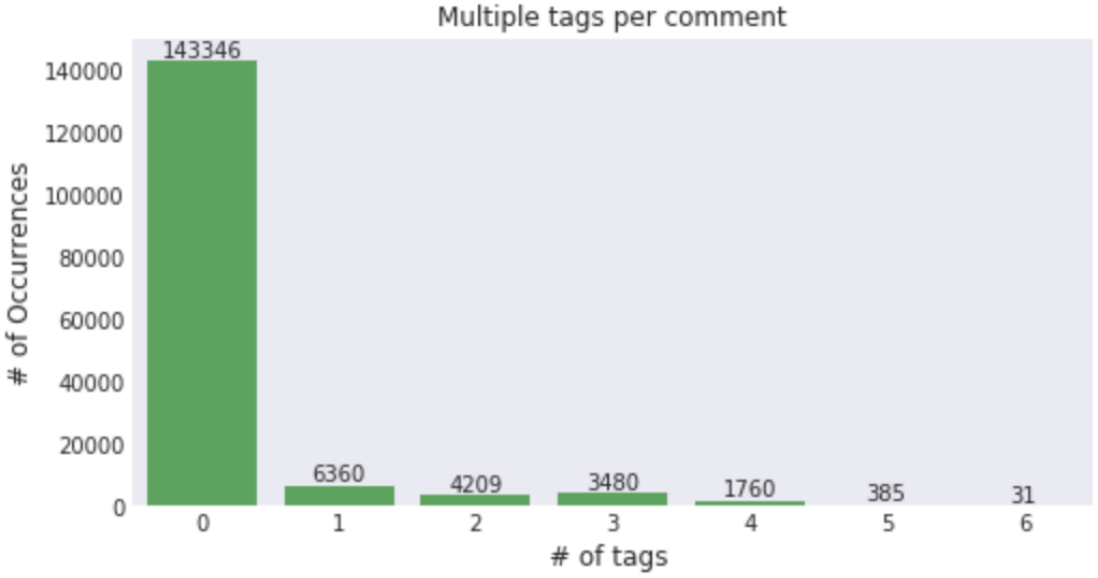
该项目主要关注于NLP领域深度类模型的构建，关注点是深度类模型在NLP领域的应用，且有官方提供的语料数据。这是一个文本多分类问题，并且是极其不平衡的多分类，每一个标签存在不止于1一个的分类标签。

数据有以下两个特点：

1. 非平衡数据集，很显然该题是一个严重不平衡的数据集 。干净数据占据了最大比例，其余不同的类别数据所占的比例也有较大的差距。




2. 多类别标签，也就是一个样本可能被归于多于一个的标签，例如对于某评论，其既涉及言语辱骂，也带有威胁性质，就可以标注为toxic,threat。如下可以看出有相当一部分的数据同时属于多个标签。




**评价指标**

该项目是文本分类的问题，该问题的评价指标为平均列 ROC AUC。换句话说，分数是每个预测列的单个 AUC 的平均值。

ROC（Receiver Operating Characteristic）曲线和AUC常被用来评价一个二值分类器（binary classifier）的优劣。

首先，解释几个二分类问题中常用的概念：True Positive, False Positive, True Negative, False Negative。它们是根据真实类别与预测类别的组合来区分的。

样本中的真实正例类别总数即TP+FN。TPR即True Positive Rate，TPR = TP/(TP+FN)。 同理，样本中的真实反例类别总数为FP+TN。FPR即False Positive Rate，FPR=FP/(TN+FP)。

公式为：

$$TPR = TruePositives/(TruePositives + FalseNegatives)$$

$$FPR = FalsePositives/(FalsePositives + TrueNegatives)$$

AUC（Area Under Curve）被定义为ROC曲线下的面积，显然这个面积的数值不会大于1。又由于ROC曲线一般都处于y=x这条直线的上方，所以AUC的取值范围在0.5和1之间。使用AUC值作为评价标准是因为很多时候ROC曲线并不能清晰的说明哪个分类器的效果更好，而作为一个数值，对应AUC更大的分类器效果更好。

# 2. 分析

## 2.1 数据研究

### 2.1.1 观察数据

Kaggle竞赛的数据一般有train、test和sample_submission，我们用pandas来读取需要的数据。

```
train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')
sample_submission = pd.read_csv('../input/sample_submission.csv')
labels = ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]
```

labels是我们需要将文本分为的六个类别。

### 2.1.2 数据预处理

数据预处理包含以下几个步骤：

1）去掉部分特殊字符和停用词。
2)把& ， @之类的符号换成了and和at
3）找了一些表情符号比如 :) ,:p 之类的替换成smile，把:（ 之类的替换成sad
4）把诸如fuckkkkkkk和fucccccck之类的词语替换成fuck

## 2.2 探索性可视化

查看数据，训练数据159571 条，测试数据153164条，比例大概是在51:49的样子。数据格式如下：
| id | comment_text | toxic | severe_toxic | obscene | threat | insult | identity_hate |

```
#Loading the Data
train = pd.read_csv(train_data_path)
test = pd.read_csv('../input/test.csv')
```

```
#a quick look at our training dataset
train.head()
```

| | id | comment_text | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | Explanation\nWhy the edits made under my usern... | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 000103f0d9cfb60f | D'aww! He matches this background colour I'm s... | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 000113f07ec002fd | Hey man, I'm really not trying to edit war. It... | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0001b41b1c6bb37e | "\nMore\nI can't make any real suggestions on ... | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0001d958c54c6e35 | You, sir, are my hero. Any chance you remember... | 0 | 0 | 0 | 0 | 0 | 0 |

第一列为文本id，第二列为文本内容，后面的6列是需要预测的值，评价指标使用的是6列的AUC值的平均数。

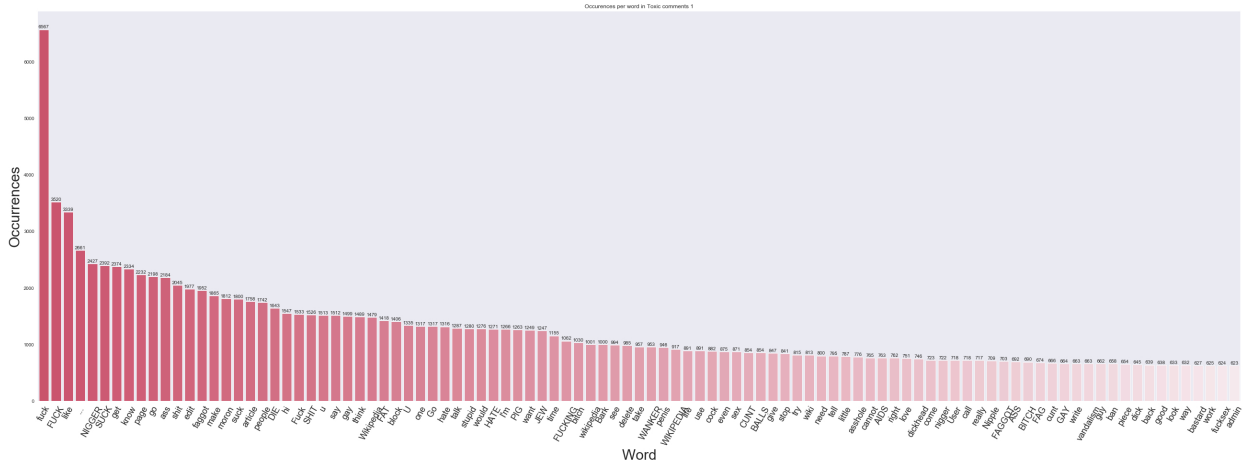
```
# the size of our training dataset
train.shape
```

```
(159571, 8)
```

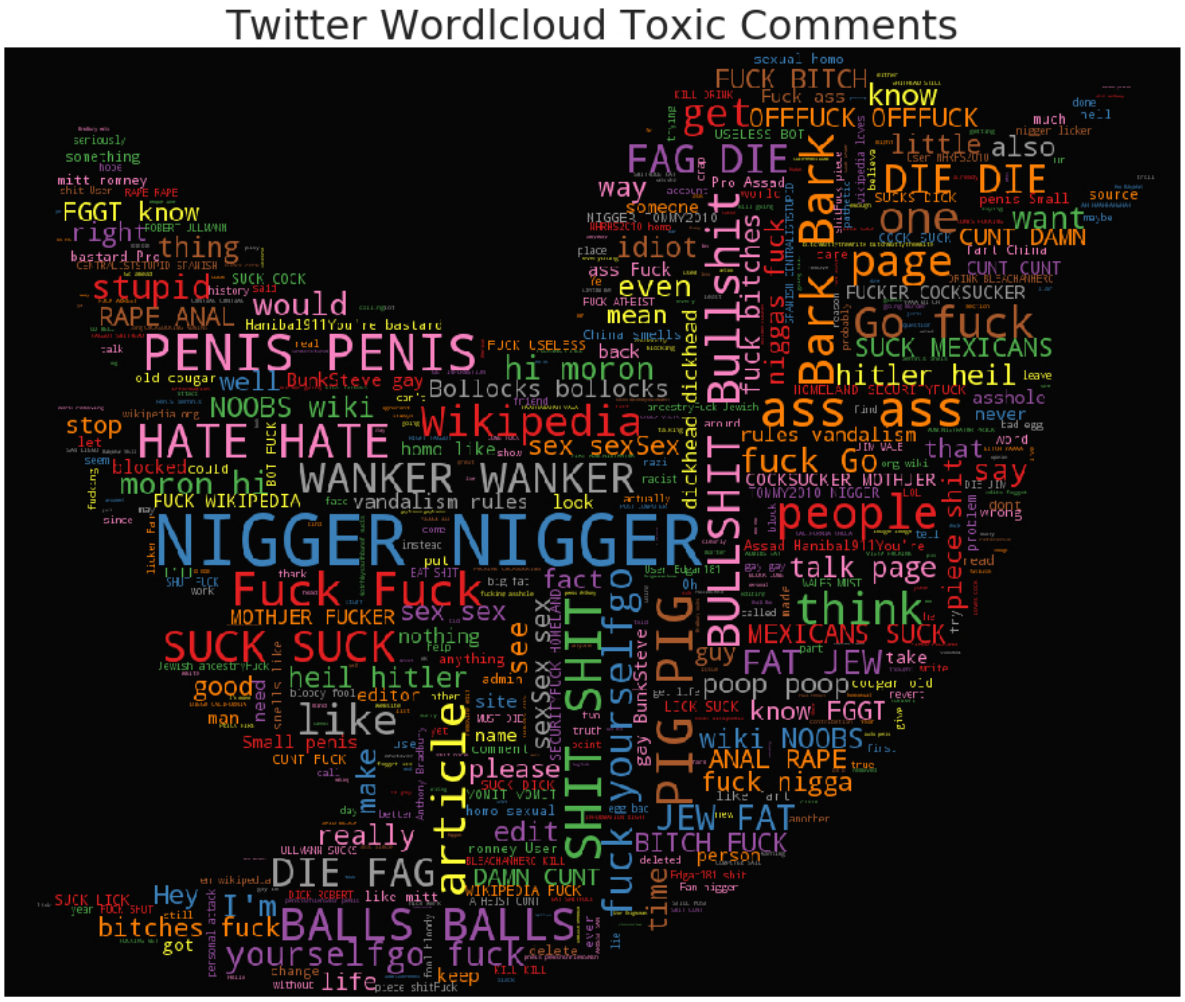
```
rowsums=train.iloc[:,2:].sum(axis=1)
train['clean']=(rowsums==0)
train['clean'].sum()
```

```
143346
```

共159571条数据，143346条含有标签的有效数据。




词表统计图




词云图

此外，为了可以对我们的训练情况进行了解和调整，还需要对训练数据切分为训练集和验证集，比例为7：3.

## 2.3 算法与方法

**TF-IDF模型：**

TF-IDF（term frequency–inverse document frequency）是一种用于资讯检索与资讯探勘的常用加权技术。TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一份文件对于所在的一个语料库中的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。

TF(词频)

在一份给定的文件里，词频（term frequency，TF）指的是某一个给定的词语在该文件中出现的频率。这个数字是对词数(term count)的归一化，以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词数，而不管该词语重要与否。）对于在某一特定文件里的词语ti来说，它的重要性可表示为：

以上式子中ni,j是该词在文件dj中的出现次数，而分母则是在文件dj中所有字词的出现次数之和。

IDF(逆向文件频率)

逆向文件频率 (inverse document frequency, IDF) 是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。

其中：

|D|：语料库中的文件总数；

|{j:ti∈dj}|:包含词语ti的文件数目，如果该词语不在语料库中，就会导致被除数为零，因此一般情况下使用1+|{j:ti∈dj}|

所以公式-1可以表示为：

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

**Logistic Regression：**

逻辑斯蒂回归模型是传统机器学习中一种非常高效的分类模型（虽然它的名字叫做回归）。因为算法简单，高效，所以基本上很多分类问题我们都可以直接使用该模型进行简单的了解。

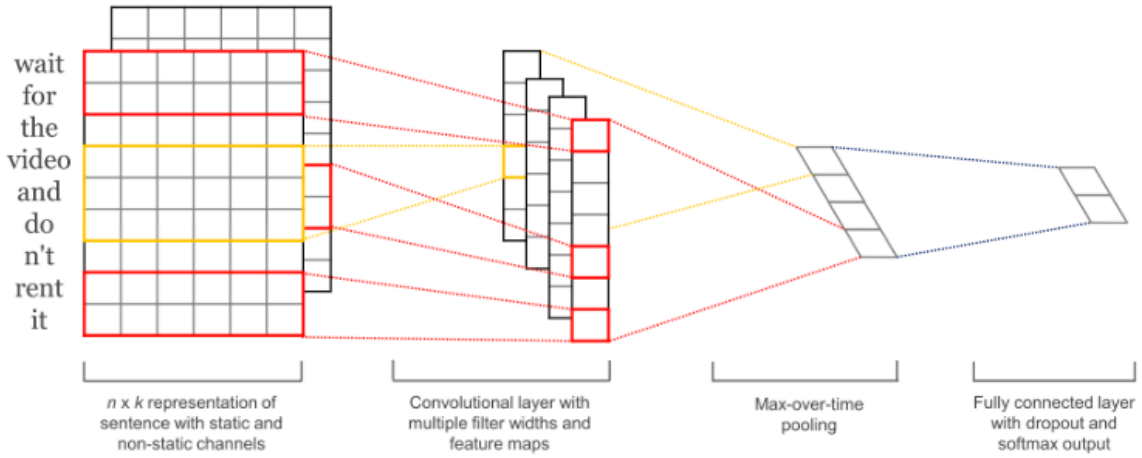
它其实可以简单地理解为y=f(x)，其中X是自变量，Y是因变量，这里我们可以认为X是我们的输入评论，Y是这个评论的标签。我们通过sigmod函数，将f(x)的值转换为一个接近于0或者1的值，来进行分类。即

$$P(y=1|x;\theta)=g(\theta^T x)=\frac{1}{1+e^{-\theta^T * x}}$$

求导我们可以利用梯度下降，随机梯度下降，等很多方式，我选择了随即平均梯度下降。这是梯度下降法的变种，和普通梯度下降法的区别是每次迭代仅仅用一部分的样本来计算梯度，所以拥有更快的收敛速度。机器学习算法选择后还需要对算法进行优化，得到最终的模型，这就涉及到许多超参数的选择，所以直接使用交叉验证来得到一组相对较好的参数，之后再用逻辑斯蒂回归进行训练得到最优模型。

**TEXTCNN：**

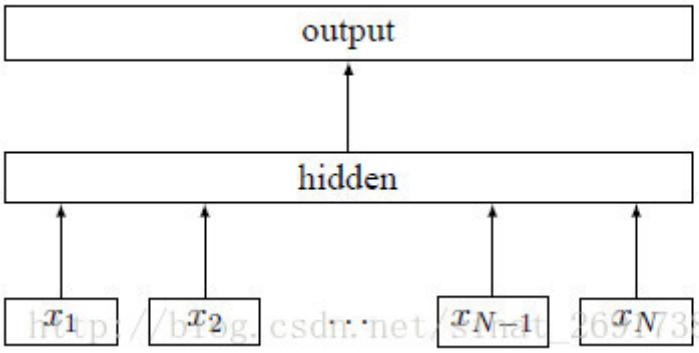
这是一种利用卷积神经网络对文本进行分类的算法




如上图，其实就是将各个词向量拼接起来，他的卷积核是一个个长方形，宽为单词向量的维度，长则是超参数，可以根据情况选择2，3，4，5等不同的数据，每次卷积核的滑动都是多个词向量。一般选择多个不同的卷积核进行滑窗，然后将卷积层的结果进行最大池化并进行拼接变成一个长特征的向量，最后使用softmax进行分类。

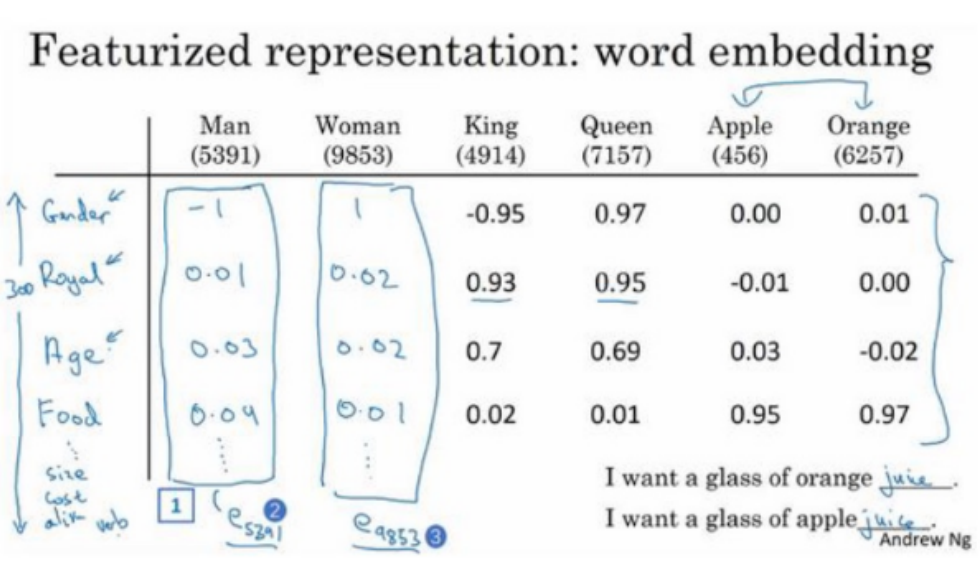
深度机器学习需要计算的参数量是非常大的，一般对于个人学习和了解来说，都会使用预训练的模型来对我们自己的数据集进行训练，也就是迁移学习。我们通过已经别人已经训练好的模型权重（在这里就是预训练词向量），来加速我们自己的训练。比如我们预计使用的“crawl-300d-2M.vec”。这是通过fasttext训练的词向量,这里的300D表示每一个词都是通过300维的特征来表示的，2M表示在两百万个单词上的训练结果。

fasttext：可以简单地理解为如下的一个结构




想要理解fasttext，还要向介绍几个概念。

1、word2vec:即一种使用向量来对词语进行表示的方法。如下，对于Man，使用一个300维的向量进行表示，这个向量上的每一个值都代表了Man对于左侧某一种特征的大小，这样，对于不同的词语将会拥有一个可解释性非常强的向量，我们就可以通过这些向量来得到不同词语的关系。



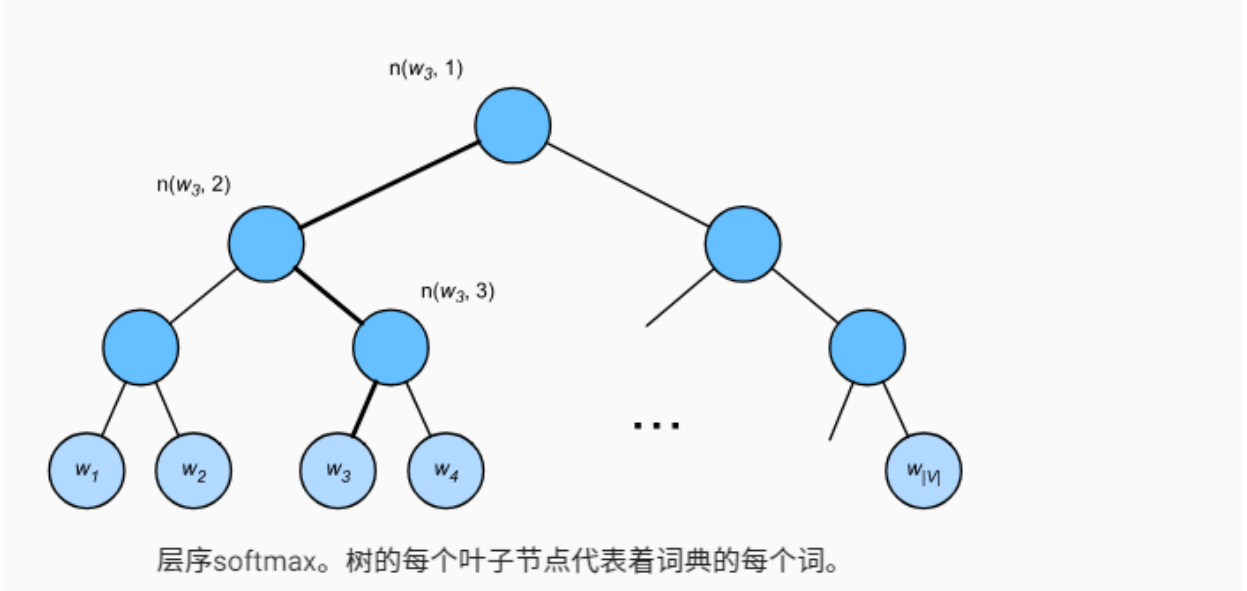

2、n-gram：按照一定的长度N对文本进行滑窗，形成长度为N的片段序列。比如，在英文中，cat,cats。在word2vec中我们使用的是两个不同的向量来表示这两个单词，n-gram则是通过滑窗（假定长度为2）得到了‘<c’ ‘ca’ ,'at','ts','s>'以及完整的词语cat和cats，这样就能够帮助了解捕捉词语的前后缀。

3、CBOW(连续词袋模型):这是一种通过输入前后词语来预测中间词语的模型。例如the gilr loves xiao zhang。滑窗大小为2，那么就是输入the ,girl,xiao,zhang,需要预测的词语就是loves假设词典索引中心词（需要预测的词语，此处是loves）的索引为c，其他输入词语索引为i，那么如下

$$\mathbb{P}(w_c \mid w_{o_1}, \ldots, w_{o_{2m}}) = \frac{\exp\left(\frac{1}{2m}\boldsymbol{u}_c^\top (\boldsymbol{v}_{o_1} + \ldots + \boldsymbol{v}_{o_{2m}})\right)}{\sum_{i \in \mathcal{V}} \exp\left(\frac{1}{2m}\boldsymbol{u}_i^\top (\boldsymbol{v}_{o_1} + \ldots + \boldsymbol{v}_{o_{2m}})\right)}.$$

公式的表示在给出中心词语Wc左右两侧背景词语的情况下，得到中心词语的概率。通过最大似然等价于最小负对数，来对上式进行求解。

4、层序softmax:通过使用 Huffman 算法建立用于表征类别的树形结构




层序softmax。树的每个叶子节点代表着词典的每个词。

假设 L(w) 为从二叉树的根节点到词 w 的叶子节点的路径上的节点数。例如计算给定词语Wc生成W3的概率，那么从跟节点再到叶子节点W3路径是左，右，左，公式就是

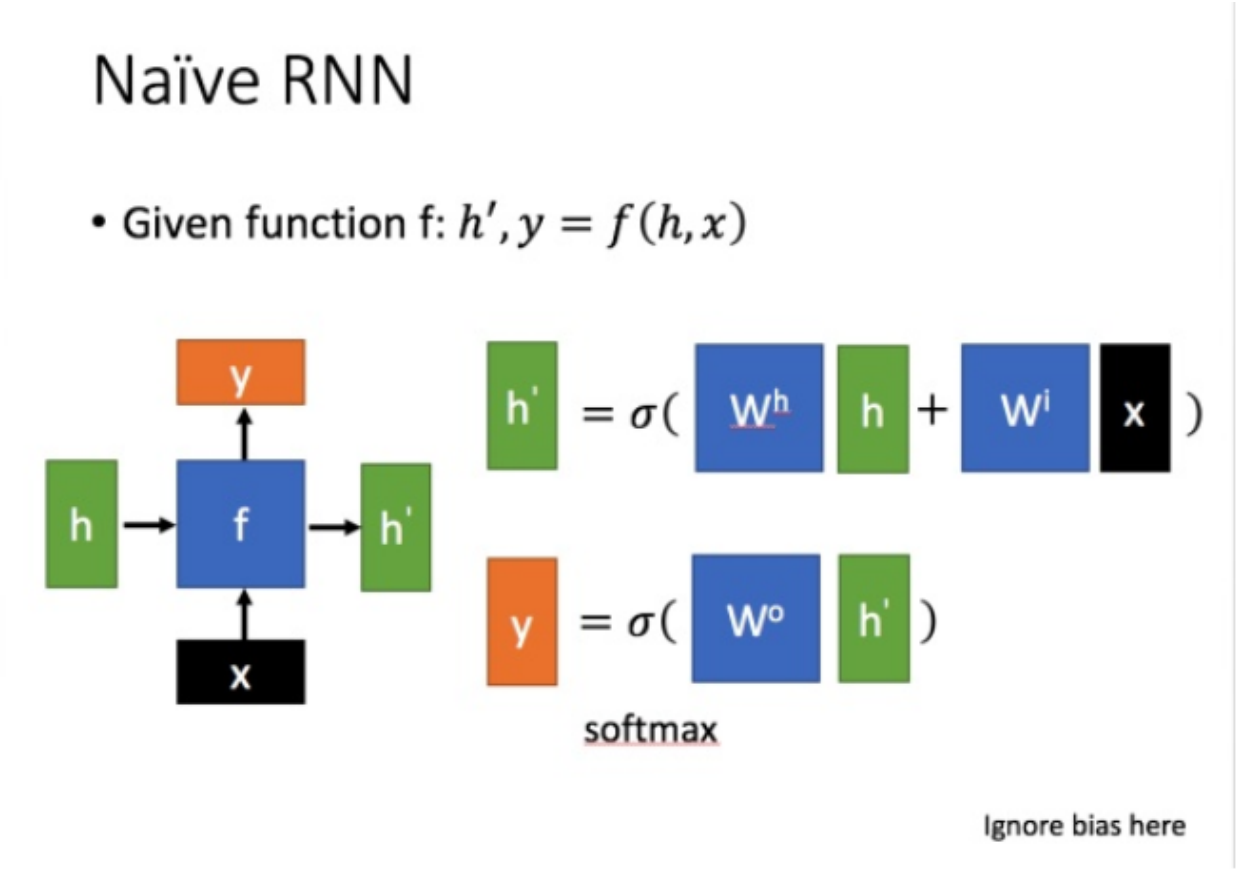
$$P(w_3 \mid w_c)=\sigma(u_{Tn(w3,1)}v_c)\cdot\sigma(-u_{Tn(w3,2)}v_c)\cdot\sigma(u_{Tn(w3,3)}v_c).$$

从上公式很容易看出来，计算量从原本的O（n）降级到了O（logn），这对于计算速度的提升是非常多的，原本可能需要几天，几个小时计算的资源，现在可能几分钟就足够了。

fasttext在一定程度上融合了上面几个技巧的优势，它拥高效的训练速度，理解词汇的构造特征，考虑到词汇的相似性。

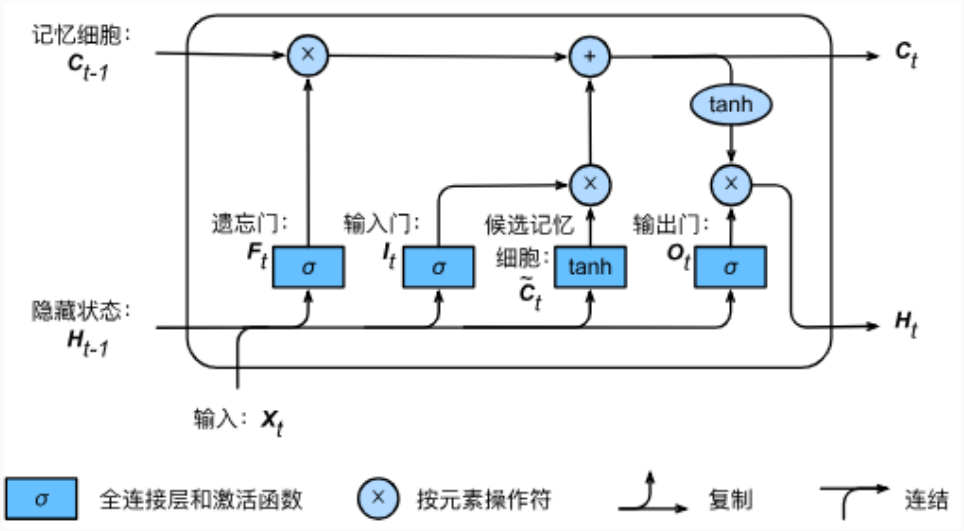
**Bi-LSTM：**

深度机器学习模型出了CNN还有RNN，意思是循环神经网络，它的特点是可以可以依据前一刻的输入会对后面的预测产生影响，也就是具有时效性之，这对于某些传统机器学习模型和CNN模型来说在某些方面会有更好效果。而bi-lstm则是RNN的一个变种，它属于双向循环神经网络，因此想要理解LSTM首先需要理解RNN。




如图，这是RNN一个非常好理解的图，X是输入，Y是输出，F是其中的映射，为了便于理解，假设只有一层隐含层，从上图便可以看出，X的输入经过F的映射，除了得到当前的输出Y，还得到了h，从h的计算公式很容易看出对h'当前输入X的映射以及上一个时刻的输入X映射出来的h，这就是RNN的简述。

LSTM则稍微复杂了一些，它的H包含了四个部分，输入门，输出门，遗忘门以及记忆细胞。

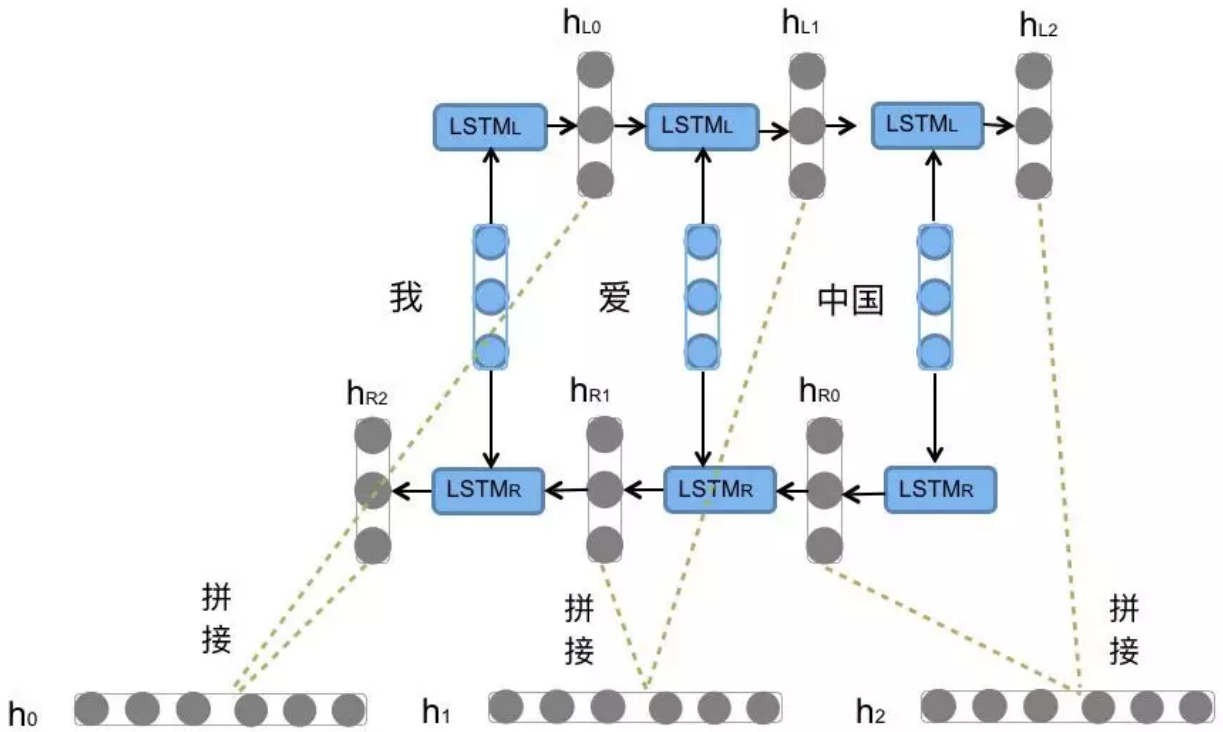



长短期记忆中隐藏状态的计算。这里的乘号是按元素乘法。

$$I_t=\sigma(X_tW_{xi}+H_{t-1}W_{hi}+b_i),$$
$$F_t=\sigma(X_tW_{xf}+H_{t-1}W_{hf}+b_f)$$
$$O_t=\sigma(X_tW_{xo}+H_{t-1}W_{ho}+b_o)$$
$$C_t=F_t\odot C_{t-1}+I_t\odot C\sim t.$$

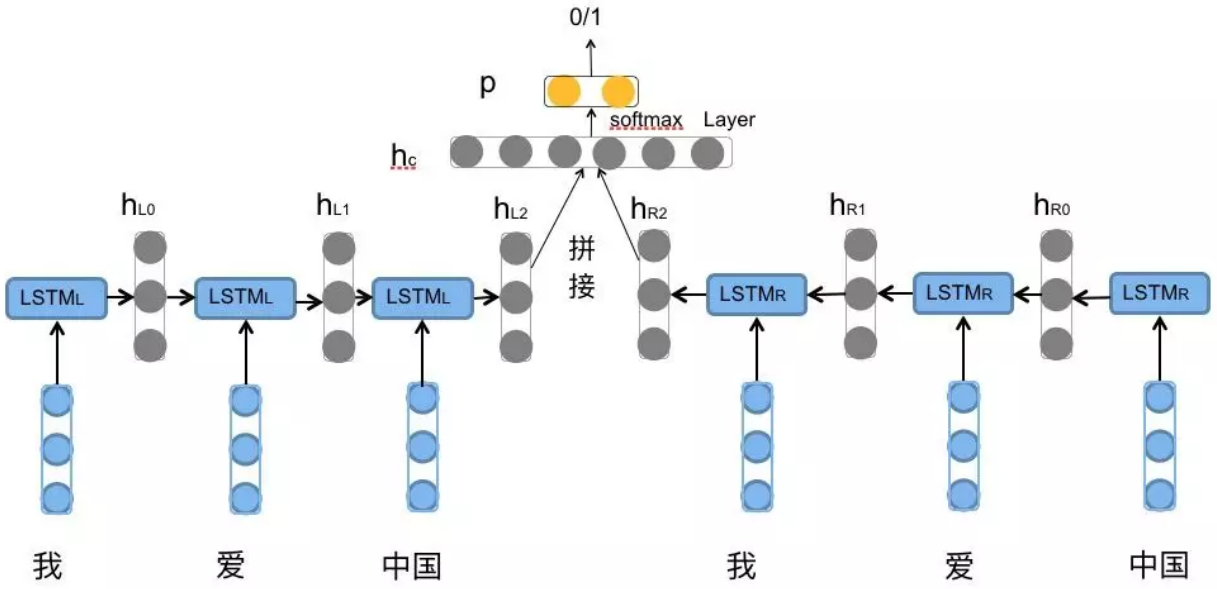
利用LSTM对句子进行建模还存在一个问题：无法编码从后到前的信息。在更细粒度的分类时，如对于强程度的褒义、弱程度的褒义、中性、弱程度的贬义、强程度的贬义的五分类任务需要注意情感词、程度词、否定词之间的交互。举一个例子，“这个餐厅脏得不行，没有隔壁好”，这里的“不行”是对“脏”的程度的一种修饰，通过BiLSTM可以更好的捕捉双向的语义依赖。

前向的LSTM与后向的LSTM结合成BiLSTM。比如，我们对“我爱中国”这句话进行编码，模型如图所示。




前向的LSTMr 依次输入“我”，“爱”，“中国”得到三个向量{hL0, hL1, hL2 }。后向的 LSTMr 依次输入“中国”，“爱”，“我”得到三个向量{ hR0, hR1, hR2 }。最后将前向和后向的隐向量进行拼接得到{[ hL0, hR2 ], [hL1, hR1 ], [hL2, hR0 ]}，即{ h0, h1, h2 }。

对于情感分类任务来说，我们采用的句子的表示往往是[hL2, hR2 ]。因为其包含了前向与后向的所有信息，如图所示。




基于word和char特征的逻辑斯蒂回归模型得分0.979。
基于glove和fasttext预训练词向量Bi-LSTM模型得分0.982。
基于glove和fasttext的textcnn模型：9782
要求：9863

### 2.4 基准模型

本次要求成绩达到 private leaderboard前20%，查询了排名也就是0.9862的得分，所以以此为基准

# 3. 实现

### 3.1 数据预处理

和前面2.1.2说的差不多
1.做了一个词典，替换数据集中可能的颜文字，替换一些特殊字符、缩写等等，如下。

```
[78]: repl = {
          "&lt;3": " good ",
          ":d": " good ",
          ":dd": " good ",
          ":p": " good ",
          "8)": " good ",
          ":-)": " good ",
          ":)": " good ",
          ";)": " good ",
          "(-:": " good ",
          "(:": " good ",
          "yay!": " good ",
          "yay": " good ",
          "yaay": " good ",
          "yaaay": " good ",
          "yaaaay": " good ",
          ":/": " bad ",
          ":')": " sad ",
          ":-(": " bad ",
          ":(": " bad ",
          ":s": " bad ",
          ":-s": " bad ",
          ":d": " smile ",
          ":p": " smile ",
          ":dd": " smile ",
```

2.利用keras自带的预处理函数 text.text_to_word_sequence进行分词，并去掉特殊字符，统一大小写等

```
138]: def preproces(data):
          data_pre = data
          for i in range(len(data)):
              data_pre[i] = text.text_to_word_sequence(data[i],filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',lower = True,split = ' ')
          for i in range(len(data_pre)):
              for j in range(len(data_pre[i])):
                  if data_pre[i][j] in keys:
                      data_pre[i][j] = repl[data_pre[i][j]]
              data_pre[i] = ','.join(data_pre[i])
          return data_pre
```

## 3.2 实施

3.2.1首先尝试了几个kernel，比如lr,textcnn,lstm，并进行一些基础的调参工作

```
In [ ]:

In [43]: batch_size = 128
         epochs = 3
         drop_list = [0.3,0.5,0.7]
         ls_list = []
         ac_list = []

In [44]: X_tra, X_val, y_tra, y_val = train_test_split(x_train, y_train, test_size=0.25, random_state=18)
         RocAuc = RocAucEvaluation(validation_data=(X_val, y_val), interval=1)
         for drop in drop_list:
             model = get_model(drop)
             print('drop:',drop)
             hist = model.fit(X_tra, y_tra, batch_size=batch_size, epochs=epochs, validation_data=(X_val, y_val),callbacks=[RocAuc],verbose=2)
             ls_list.append(hist.history['val_loss'])
             ac_list.append(hist.history['val_acc'])

         drop: 0.3
         Train on 119678 samples, validate on 39893 samples
         Epoch 1/3
          - 74s - loss: 0.0694 - acc: 0.9771 - val_loss: 0.0462 - val_acc: 0.9831

          ROC-AUC - epoch: 1 - score: 0.9826

         Epoch 2/3
          - 71s - loss: 0.0460 - acc: 0.9830 - val_loss: 0.0443 - val_acc: 0.9836

          ROC-AUC - epoch: 2 - score: 0.9864
```

优化器选择了adam，adam利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率，是非常优秀的一种优化器。

使用一个循环对drop值进行简单的尝试，可以看到，当drop值较小的时候，效果是比较好的。

```
ROC-AUC - epoch: 1 - score: 0.9817

Epoch 2/10
119678/119678 [==============================] - 73s 609us/step - loss: 0.0463 - acc: 0.9831 - val_loss: 0.0441 - val_acc: 0.9837

Epoch 00002: val_loss improved from 0.04754 to 0.04409, saving model to textcnn_weights_base2.best.hdf5

 ROC-AUC - epoch: 2 - score: 0.9855

Epoch 3/10
119678/119678 [==============================] - 73s 610us/step - loss: 0.0393 - acc: 0.9852 - val_loss: 0.0449 - val_acc: 0.9834

Epoch 00003: val_loss did not improve from 0.04409

 ROC-AUC - epoch: 3 - score: 0.9860

Epoch 4/10
119678/119678 [==============================] - 73s 610us/step - loss: 0.0339 - acc: 0.9871 - val_loss: 0.0457 - val_acc: 0.9836

Epoch 00004: val_loss did not improve from 0.04409

 ROC-AUC - epoch: 4 - score: 0.9851

Epoch 5/10
```

在第三个epoch的时候roc-auc就不再更新了，于是进行了提交，结果都还可以，平均得分0.98左右。但是距离要求的0.9863还有一段距离，于是进行了最简单的依据得分进行加权平均

```
submission_1 = pd.read_csv("submission-textcnn2.csv")                    #0.979
submission_2 = pd.read_csv("submission_lr.csv")                  #0.9782
submission_3 = pd.read_csv("submission_lstm.csv")                      #0.982

blend = submission_3.copy()
col = blend.columns
```

[3]:
```
blend.head()
```

ut[3]:

| | id | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|---|
| 0 | 00001cee341fdb12 | 0.999802 | 4.774746e-01 | 9.846950e-01 | 2.600611e-01 | 0.956651 | 2.942494e-01 |
| 1 | 0000247867823ef7 | 0.000008 | 1.001766e-12 | 7.067682e-07 | 5.889907e-09 | 0.000002 | 3.751409e-09 |
| 2 | 00013b17ad220c46 | 0.000029 | 7.640322e-11 | 5.104831e-06 | 1.455284e-07 | 0.000011 | 9.698881e-08 |
| 3 | 00017563c3f7919a | 0.000051 | 2.560379e-11 | 6.031402e-06 | 3.634127e-08 | 0.000017 | 3.932794e-08 |
| 4 | 00017695ad8997eb | 0.000058 | 1.701672e-10 | 7.344144e-06 | 2.711431e-07 | 0.000015 | 1.305792e-07 |

[6]:
```
col = col.tolist()
col.remove('id')

a= 0.25
b = 0.25
c = 0.5

blend[col] = a*minmax_scale(submission_1[col].values) + b*minmax_scale(submission_2[col].values) + c*minmax_scale(submission_3[col].values)
```

按照得分的高低，赋予不同的权重，再一次进行提交，竟然有了不错的提升， private leaderboard得到了0.9842的得分，但是这里我们想要的最低0.9862依然还有不少的差距。

3.2.2 在不改变模型的前提下，为了得到更好的成绩，stacking是个不错的选择。其实很简单，步骤如下：

1、选择基模型。我们可以有xgboost，lightGBM，RandomForest，SVM，ANN，KNN，LR，CNN,RNN,LSTM等等你能想到的各种基本算法模型。

2、把训练集分为不交叉的五份。我们标记为train1到train5。

3、从train1开始作为预测集，使用train2到train5建模，然后预测train1，并保留结果；

然后，以train2作为预测集，使用train1，train3到train5建模，预测train2，并保留结果；如此进行下去，直到把train1到train5各预测一遍；

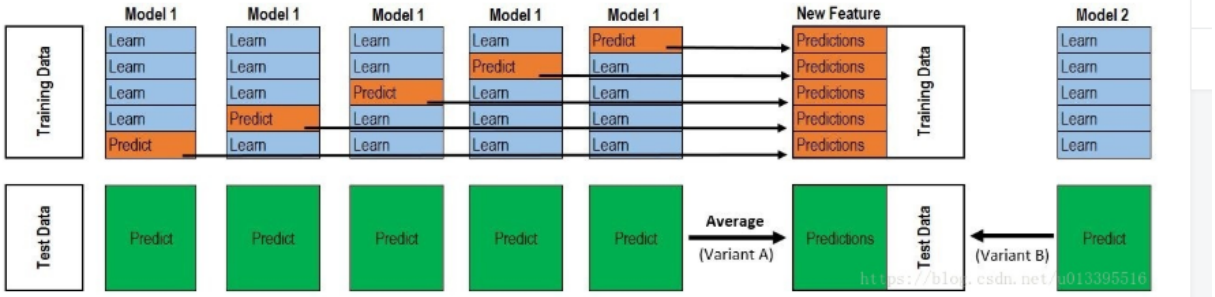
4、把预测的结果按照train1到trian5的位置对应填补上

5、在上述建立的五个模型过程中，每个模型分别对test数据集进行预测，并最终保留这五列结果，然后对这五列取平均，保留该预测结果作为第二层模型的测试集。

6、选择第二个基模型，重复以上2-5操作

7、以此类推。有几个基模型，就会对整个train数据集生成几列新的特征表达。同样，也会对test有几列新的特征表达。

8、选择一个表达能力较强的模型作为第二层，将第一层的几个模型的预测结果作为新的特征输入。




于是，采用了上面表现较好的LSTM和textcnn模型作为基模型进行kfold训练，并且在训练的过程中，同时使用了fasttext和glove的预训练词向量。

```
def get_model():
    inp = Input(shape=(max_len, ))
    #fasttext
    fast_text_embedding = Embedding(max_features, embed_size, weights=[embedding_matrix_fasttext])(inp)
    x = SpatialDropout1D(0.1)(fast_text_embedding)
    x = Reshape((max_len, embed_size, 1))(x)


    conv_0 = Conv2D(num_filters, kernel_size=(filter_sizes[0], embed_size), kernel_initializer='normal',activation='elu')(x)
    conv_1 = Conv2D(num_filters, kernel_size=(filter_sizes[1], embed_size), kernel_initializer='normal',activation='elu')(x)
    conv_2 = Conv2D(num_filters, kernel_size=(filter_sizes[2], embed_size), kernel_initializer='normal',activation='elu')(x)

    maxpool_0 = MaxPool2D(pool_size=(max_len - filter_sizes[0] + 1, 1),strides=(1,1), padding='valid')(conv_0)
    maxpool_1 = MaxPool2D(pool_size=(max_len - filter_sizes[1] + 1, 1),strides=(1,1), padding='valid')(conv_1)
    maxpool_2 = MaxPool2D(pool_size=(max_len - filter_sizes[2] + 1, 1),strides=(1,1), padding='valid')(conv_2)

    concatenated_tensor1  = Concatenate(axis=1)([maxpool_0, maxpool_1, maxpool_2])



    #glove
    glove_embedding = Embedding(max_features, embed_size, weights=[embedding_matrix_glove])(inp)
    y = SpatialDropout1D(0.1)(glove_embedding)
    y = Reshape((max_len, embed_size, 1))(y)


    conv_3 = Conv2D(num_filters, kernel_size=(filter_sizes[0], embed_size), kernel_initializer='normal',activation='elu')(x)
    conv_4 = Conv2D(num_filters, kernel_size=(filter_sizes[1], embed_size), kernel_initializer='normal',activation='elu')(x)
    conv_5 = Conv2D(num_filters, kernel_size=(filter_sizes[2], embed_size), kernel_initializer='normal',activation='elu')(x)
```

在模型的embedding层载入预训练词向量，经过三个不同的卷积核进行滑窗后，对划船结果进行池化，并将池化层进行拼接，无论是fasttext还是glove都是如上的步骤，最后经过flatten层展开，在经过sigmoid激活函数进行分类预测。（因为当前数据集可以看作是六个二分类，当标签之间具有排他性的时候，可以选择softmax）。

```
17]: def get_model():
    inp = Input(shape=(max_len,))
    x1 = Embedding(max_features, embed_size, weights=[embedding_matrix_glove])(inp)
    x1 = SpatialDropout1D(0.1)(x1)
    x1 = Bidirectional(CuDNNLSTM(128, return_sequences=True))(x1)

    x = Embedding(max_features, embed_size, weights=[embedding_matrix_fast_text])(inp)
    x = SpatialDropout1D(0.1)(x)
    x = Bidirectional(CuDNNLSTM(128, return_sequences=True))(x)

    concat = concatenate([x1, x])
    avg_pool = GlobalAveragePooling1D()(concat)
    max_pool = GlobalMaxPooling1D()(concat)
    concat_pool = concatenate([avg_pool, max_pool])


    x = Dense(32, activation="relu")(concat_pool)
    x = Dropout(0.1)(x)
    out = Dense(6, activation='sigmoid')(x)

    model = Model(inp, out)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model
```

对bi-lstm模型也做一样的kfold处理，之后就开始了第一次stacking

```
[35]: sub_cnn = pd.read_csv('0.9869textcnn_submssioncnn-10-fold.csv')
sub_lstm = pd.read_csv('0.98tm_submssionbi-lstm-10-fold.csv')
cv_cnn = pd.read_csv('0.9869textcnn_cv_cnn-10-fold.csv')
cv_lstm = pd.read_csv('0.98tm_cvbi-lstm-10-fold.csv')
```

读取存储了的基模型对训练集的预测值和测试集的预测。

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 0001d958c54c6e35 | you,sir,are,my,hero,any,chance,you,remember,wh... | 0 | 0 | 0 | 0 | 0 | 0 | |

```
In [61]: train_df = train.merge(cv_1, on=['id'])
train_df = train_df.merge(cv_2, on=['id'])
```

```
In [62]: train_df.head()
```

Out[62]:

| | id | comment_text | toxic | severe_toxic | obscene | threat | insult | identity_hate | toxic_cvcnn | severe_toxic_cvcnn |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000997932d777bf | explanation,why,the,edits,made,under,my,userna... | 0 | 0 | 0 | 0 | 0 | 0 | 0.002993 | 0.000229 |
| 1 | 000103f0d9cfb60f | d'aww,he,matches,this,background,colour,i,am,s... | 0 | 0 | 0 | 0 | 0 | 0 | 0.001511 | 0.000129 |
| 2 | 000113f07ec002fd | hey,man,i,am,really,not,trying,to,edit,war,it,... | 0 | 0 | 0 | 0 | 0 | 0 | 0.011895 | 0.000190 |
| 3 | 0001b41b1c6bb37e | more,i,can,not,make,any,real,suggestions,on,im... | 0 | 0 | 0 | 0 | 0 | 0 | 0.000109 | 0.000031 |
| 4 | 0001d958c54c6e35 | you,sir,are,my,hero,any,chance,you,remember,wh... | 0 | 0 | 0 | 0 | 0 | 0 | 0.025816 | 0.001058 |

```
In [63]: test_df = test.merge(sub_1, on=['id'])
test_df = test_df.merge(sub_2, on=['id'])
```

因为训练的时候是给每个ID配一个随机产生的1到10的数，所以训练出来的顺序可能是不同的，这两招ID进行拼接。

```
[1]: def get_stacking_model():
    cv1_input = Input(shape=(6,))
    cv2_input = Input(shape=(6,))
    merged = concatenate([cv1_input, cv2_input])
    merged = Dense(1024, activation='relu')(merged)
    dense = Dense(256, activation='relu')(merged)
    drop = Dropout(0.1)(dense)
    output = Dense(6, activation='sigmoid')(drop)
    model = Model(inputs=[ cv1_input, cv2_input],
                  outputs=output)
    model.compile(loss='binary_crossentropy',
                  optimizer=keras.optimizers.Adam(lr=1e-3),
                  metrics=['accuracy'])
    return model
```

一个非常简单的NN模型，且训练的时候依然采取了10折训练，最后对基模型的testdata进行预测

[120]:
```
for i in range(Kfold):
    print("prediction "+ str(i))
    if len(r) == 0:
        r = cv_models[i].predict(test_ips,batch_size=256)
    else:
        r += cv_models[i].predict(test_ips,batch_size=256)
```

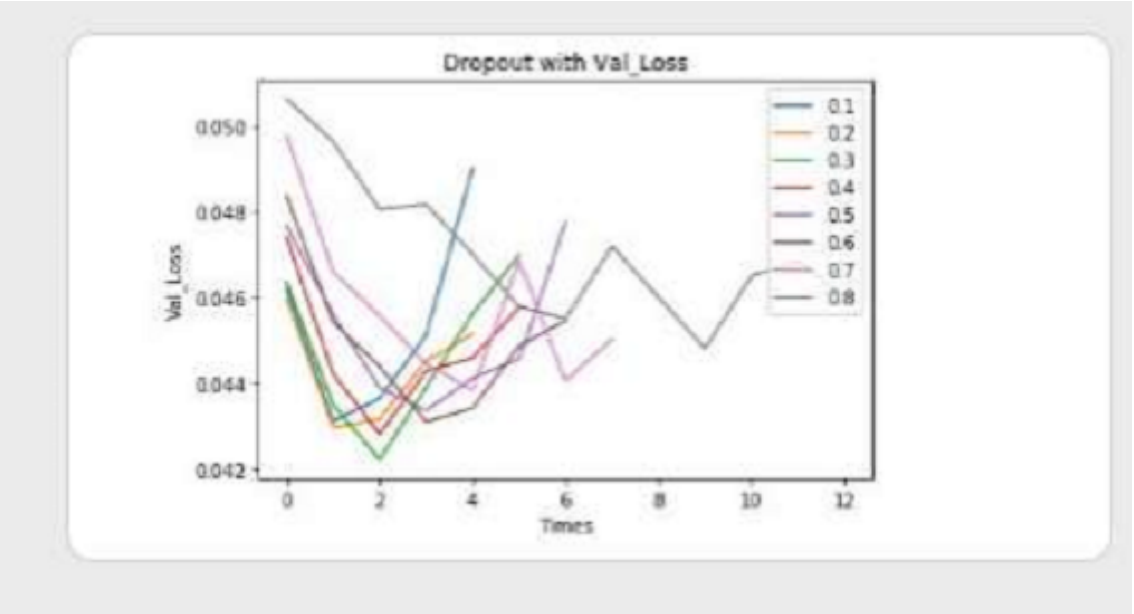
```
prediction 0
prediction 1
prediction 2
prediction 3
prediction 4
prediction 5
prediction 6
prediction 7
prediction 8
prediction 9
```

不知道为什么，提交了成绩反而降低了，Private Score LB得分只有0.9810。
再次尝试了K折训练的textcnn、lstm以及LR，Private Score LB得分0.9842。

在经过了第一次提交之后，依据反馈，修改了自己的模型和参数，同时对原有的lstm模型进行了优化，尝试了更加复杂的bilstm-cnn,gru-cnn等模型，这样的的复杂模型或许拥有更强的解释能力，在划分数据集的时候使用了MultilabelStratifiedKFold，因为当前任务是一个多标签且不平衡数据集，该函数可以在划分数据集时保持数据集的分类比例，这样就保证了再每一折之中都有足够多的不同标签样本作为训练集。使用了不同的预训练词向量（fasttext和glove）进行比较，效果都接近，添加BN层防止过拟合

```
x = BatchNormalization()(cov)
x = Dense(50, activation = 'relu')(x)
x = BatchNormalization()(x)
```

然而，最终结果也无法让人满意，单模型的最好成绩依然只有0.983，训练过程中val_loss依旧无法收敛，因为尝试了太多不同的参数和模型，因此仅保留了最后两个




### 3.3完善

3.3.1 猜测数据的预处理做的不好，需要更多的特征工程来挖掘数据集本身

3.3.2将多个表现能力较好的单模型进行融合

# 4. 结果

### 4.1 模型评估与验证

### 4.2 理由

。

# 5. 结论

### 5.1 自由形态的可视化

### 5.2 思考

对于此项目，仅仅是自然语言处理的开端，想要实际解决项目问题，还需要学习更多的知识。

基础的模型有了一个大致的了解，但是想要做的更好，许多复杂的混合模型以及模型融合的方法还需要了解。

对于机器学习, 目前比较难的地方应该还是在于解决实际问题， 结束此次课程后, 将主要研究如何将机器学习应用道具体项目之中。

## 5.3 改进

1、没有尝试更多的方式提取词特征
2、没有尝试数据增强
3、尝试的模型过少，还有cnn-rnn,rnn-cnn,gru-CNN等许多模型没有尝试

**参考文献：**


**[1] ROC Curves in Python and R**

**[2] A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification**

**[3] Python TF-IDF计算100份文档关键词权重.**

**[4] Library for efficient text classification and representation learning**

**[5] 李牧：动手深度学习**

**[6] BiLSTM介绍及代码实现**