

# 中山大学计算机学院

## 人工智能

### 本科生实验报告

(2022 学年春季学期)

课程名称: Artificial Intelligence

教学班级	计科 1 班	专业 (方向)	计算机科学与技术
学号	21307077	姓名	凌国明

# 一、实验题目

## 弱监督文本分类

采用**自训练神经网络**实现弱监督文本分类

使用**同一套参数**, 在两个不同的数据集上评估性能

实现了基于“拒绝低置信度”的**未知类型识别**

实现了高效的数据预处理, 探讨了**训练集初始化**的有效方法

针对神经网络的不确定性, 提出了一种**基于距离的优化方法**

```
13,"nobody is saying that you should n't
    with msg just put the msg on the table
    leave it out you can include a packet
    options treat msg the same way i would
```

### 数据说明

每一行是一个数据项, 第一列是真实的标签, 第二列是文本, 中间以 ‘’ 分隔。第一个数据集有 20 个类别, 第二个数据集有 4 个类别。文本中分词不一定准确, 有许多噪声和干扰项

注意, 整个训练过程不能使用真实的类标签, 只能用生成的伪标签

## 二、实验内容

### 1、算法原理

#### 数据预处理

##### 文本噪声

```
r radiance isy liu se 130 236 1 3 pu  
liu se irisa fr 131 254 2 3 ipsc2 v  
el badouel irisa irisa fr may have d  
s \(\( 640 480 \) rayscene demos ameri
```

观察数据集，发现文本中存在以下形式的噪声：

```
\( qscfphghl \( h hcl kjqtu \) ldh \)  
\) quubut !! s ! l3 !! uu ! qjs ut !  
! j ! b8 9 9l0a i !! g` !! g`b !!
```

1. ' ' 和 ' ' 后面一个字符
2. 标点符号
3. 单个字母
4. 所有数字

认为这些噪声是纯粹的干扰项，对文本分类没有任何帮助，因此删除

##### 停用词

```
know whether it is true or  
d she \) is guiding every  
god is just one aspect of
```

```
ained in the ent files , but the  
ome bio ns ca 142 2 20 2 pub art  
m the 'images from the edge' cd r
```

文本中的停用词包括

1. 冠词：例如 "a"、"an"、"the"。
2. 介词：例如 "in"、"on"、"at"。
3. 连词：例如 "and"、"or"、"but"。
4. 代词：例如 "he"、"she"、"it"。
5. 助词：例如 "is"、"are"、"was"。
6. 情态动词：例如 "can"、"could"、"may"。
7. 副词：例如 "very"、"often"、"always"。
8. 数词：例如 "one"、"two"、"three"。
9. 感叹词：例如 "oh"、"wow"、"alas"。

停用词往往是文本中的常见词汇，它们出现的频率高，但往往**不携带重要的语义信息**。

去除停用词有以下的优缺点：

优点一：提高分类性能：去除停用词可以减少文本中的噪音和冗余信息，使模型更关注那些具有**较高信息量**的词汇，从而提高分类任务的**准确性和泛化能力**。

优点二：提高效率：**减少了特征维度**后，训练和推理的速度都会得到提升，这对于大规模文本分类任务或实时应用是非常有益的。

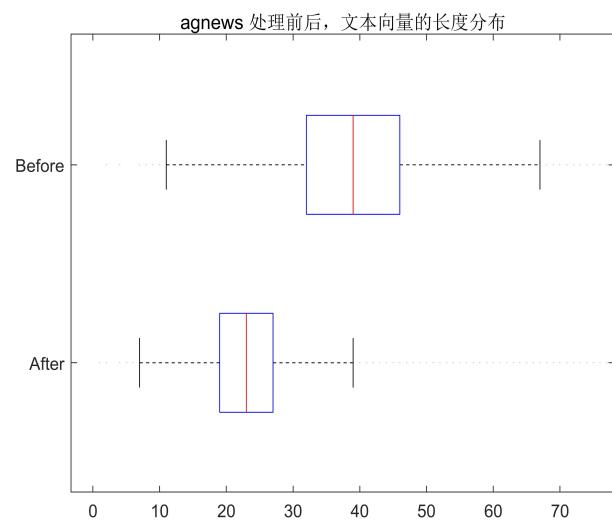
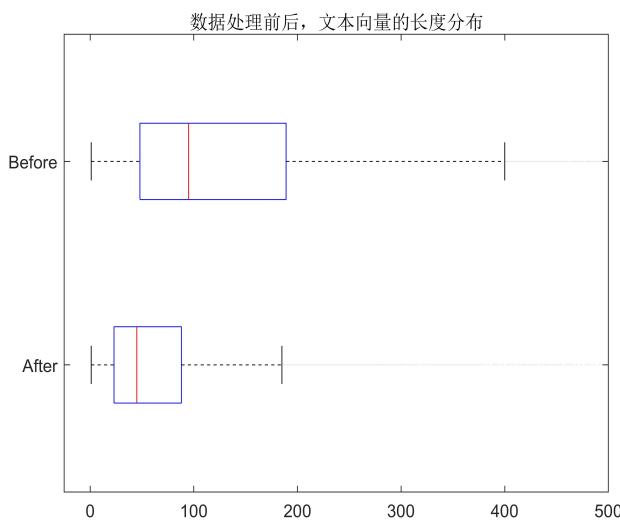
缺点：信息丢失：有时候停用词可能包含一些**上下文相关的信息**，虽然它们在单个文档中不具备显著的意义，但在整个语料库中可能具备一定的信息。去除停用词可能会导致这些**信息的丢失**。

本任务是文本分类，认为停用词对文本分类并没有什么作用，只是一个干扰项，因此去除所有的停用词。

## 数据处理效果

数据处理就是删除文本噪声，删除停用词。数据处理前后，各画了箱线图做对比。

箱线图的数据是数据集的文本长度

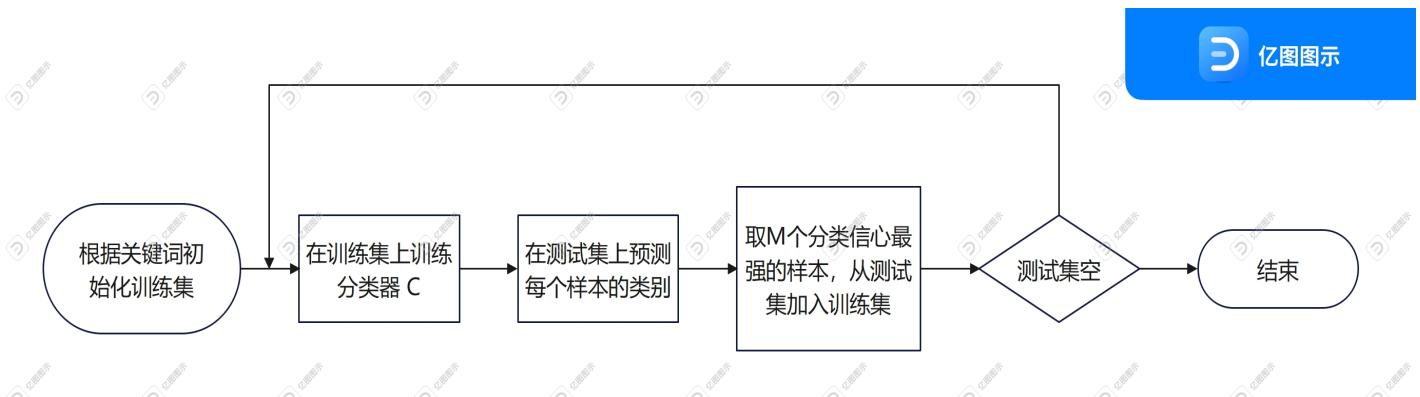


注意，左边的是 20ns，右边的是 agnews。

可见数据处理后，两个数据集的文本的长度近乎**压缩了一半**，这大大降低了特征维度，加快了训练和推理的速度。同时，删除的都是冗余的无用的信息，使得模型更专注于有效的信息，可以有效提升准确性和泛化能力。

注意，这里箱线图是带离群点检测的，大于  $1.5 \cdot IQR$  的没有画出来（大概有 1000 个）

# 弱监督文本分类任务流程



这引申出来四个问题

1. 如何根据弱监督信息**初始化**文本的“伪标签”?
2. 采用什么**分类器**?
3. 如何**衡量分类信心**?
4. 如何**扩展训练集**?

接下来我将一一回答这些问题，关于分类器的原理较多，会放在这部分的最后

**特别注意：**我将整个数据集分为了训练集和测试集。在训练集上训练，在测试集上测试，并选出  $M$  个加入训练集。最后测试集里没有样本，训练集涵盖整个数据集（这只是我的一种叫法）

## 根据关键词初始化训练集

弱监督文本分类任务中，往往缺少（甚至没有）标签数据，这个时候我们就无法进行有效的训练。所以我们要根据一些**先验知识**来为一部分样本贴上**伪标签**，以此来形成初始的训练集。

在这里，采用**关键词**的方法来初始化训练集。每个类别有对应的关键词，比如 sport 类别的关键词有 basketball, football 等等

```
0:government,military,war  
1:basketball,football,athletes  
2:stocks,markets,industries  
3:computer,telescope,software
```

利用关键词为一部分样本贴上伪标签后，我们就可以以这部分样本为初始训练集，对整个数据集进行弱监督学习任务

注意：这里利用关键词生成的伪标签并不全是正确的，而弱监督学习任务中，**初始训练集的准确率是非常重要的**（最终准确率往往难以超过初始训练集的准确率）。因此，如何合理地利用关键词生成伪标签十分重要。

## 关键词生成的伪标签的三种方法

1. 朴素方法：遇到关键词就贴相应标签
2. 众数表决：一个句子可能有多个类的关键词，选择出现关键词最多的类
3. 占优众数表决：只有当这个句子中的关键词大多属于同一类时才贴标签

朴素方法：遍历每个句子，对每个句子遍历每个单词，一旦遇到关键词，就贴上关键词对应的标签，然后转到下个句子

众数表决：遍历每个句子，对每个句子遍历所有单词，记录句子中所有关键词及其对应的类。遍历完所有单词后，会知道这个句子中有每个类的多少关键词，选择出现关键词数量最多的类。比如有四个类， $[1, 2, 3, 3]$  表示有第 0 个类的 1 个关键词，有第 1 个类的 2 个关键词……这个时候选择关键词数量最多的类，在这个例子中选择类 2 或者 类 3

占优众数表决：对上述方法的改进版，只有当这个句子中的关键词大多属于同一类时才贴标签，否则略过。比如有四个类， $[1, 2, 3, 3]$  表示有第 0 个类的 1 个关键词，有第 1 个类的 2 个关键词……这个例子中，类 2 和类 3 的关键词数量最多，数量为 3，但是这个句子中的关键词并不大多属于同一类，这意味着这个句子的归属还是比较模糊的，于是我们不贴这个标签

具体的算法会在下面用伪代码展示，以下是三种方法的效果

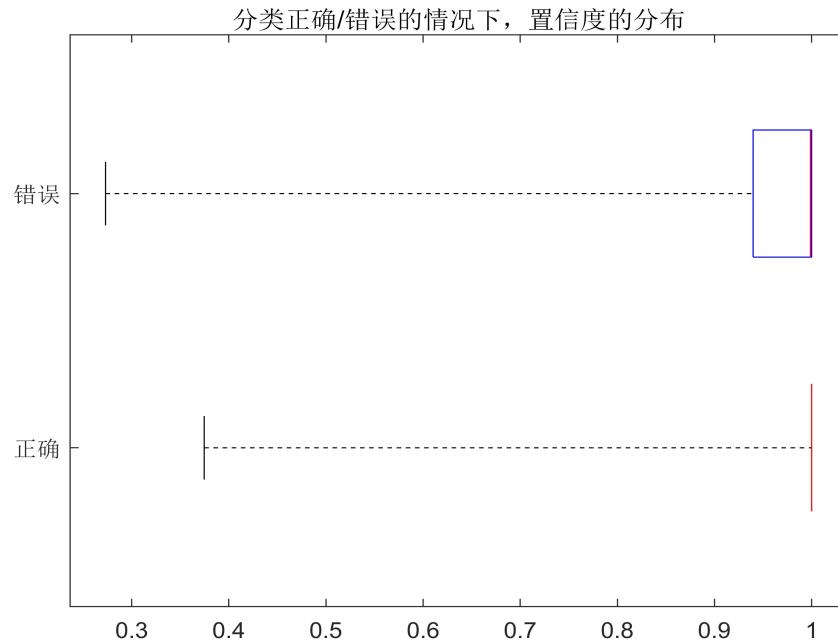
	朴素方法	众数表决	占优众数表决
20ns	11253 58.58%	11253 59.79%	8117 67.70%
agnews	6294 78.92%	6294 78.20%	6162 79.42%

可见，朴素方法和众数表决两种方法，生成的初始训练集样本数量是一样的，这是因为一个数组必定有众数。但是众数表决方法在 20ns 上提升了一点点准确率，在 agnews 上却降低了一点点准确率。

占优众数表决方法虽然生成的初始训练集样本数会稍微少一点，但是能有效地提高初始训练集的准确率，这对于整个学习任务的准确率很有帮助。因此采用占优众数表决方法是十分重要的。

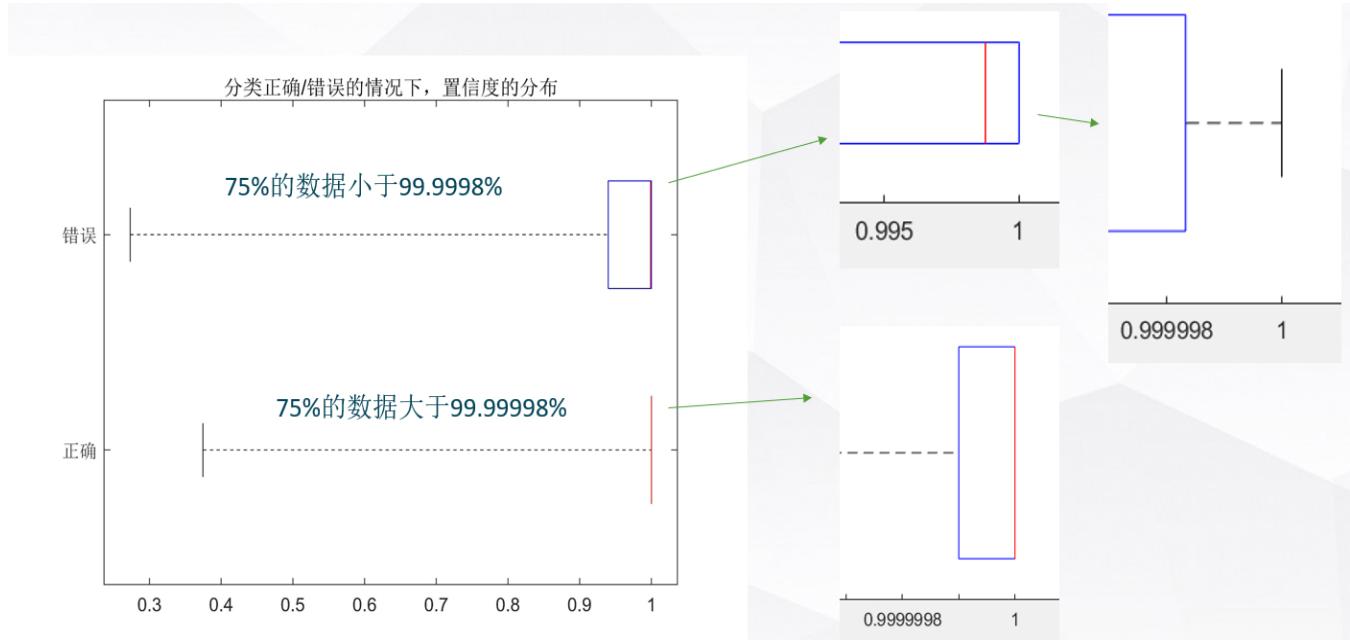
# 用置信度衡量分类信心

在这个任务中，我们用神经网络输出的类别置信度来表示分类信心，这是合理的吗？



在 20ns 数据集上作了以下的小实验：采用占优众数表决的方法生成了有 8117 个样本的初始训练集，将初始训练集中的伪标签全部换成真实标签。

在训练集上作训练直到收敛，在整个数据集上作测试，准确率为 60%（小数点后面忘了），将分类正确和分类错误的数据分为两个类别，记录两个类别的置信度信息，对两个类别的置信度信息作箱线图。



从以上箱线图可以看出，分类错误的样本中，75%的置信度数据小于 99.9998%；分类正确的样本中，75%的置信度数据大于 99.99998%。

因此，由贝叶斯公式，我们可以得出，当我们取得一个分类置信度很大的样本（假设为 1.0），认为这个样本大概率是分类正确的。因此认为用置信度衡量分类信心是合理的。

# 基于距离优化信心衡量方法

上面我们作了分析，认为从贝叶斯的视角，用置信度衡量分类信心是合理的。这是因为取得一个置信度非常大的数据时，认为它分类正确的概率较大。认为置信度和分类正确概率存在正相关的关系。

但是经过思考，认为这种正相关关系具有一定的不确定性，这是因为取得一个置信度非常大的数据时，它仍有可能是分类错误的。于是思考能不能加入一些别的信息来衡量分类信心，而且这个信息是不依赖于我们的神经网络的。

原来的方法，每个样本的评分就是置信度。现在给这个评分增加一项内容——到类别中心的距离，评分计算公式如下

$$Score = \alpha \cdot confidence + \beta \cdot distance\_score$$

这意味着每个样本的评分是置信度和距离评分的加权和。

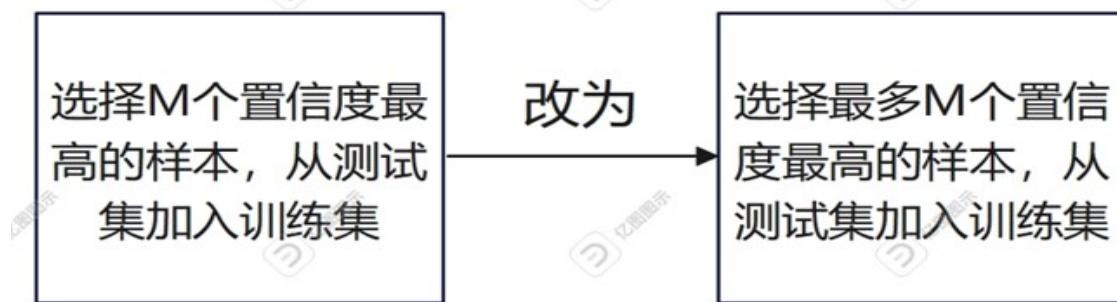
距离评分计算公式如下：

$$distance\_score = \frac{\text{到所有类别中心的距离总和}}{\text{到该类别中心的距离}}$$

基本思想是：距离该类中心近，距离其他类中心远的， $distance\_score$  的分值较高

这里采用 Bag\_Of\_Words 向量来表示每个样本，维护训练集中每个类别的中心，采用欧式距离来衡量样本之间的距离（缺点后续会提到）

## 扩展训练集的方法



采用**拒绝低置信度**的方法来实现**未知类型识别**。

每次迭代，测试时，从置信度高于一定阈值的样本中，挑选最多M个加入训练集。每轮迭代，加入训练集的样本有可能不足M个。当某轮迭代，加入训练集的样本小于N个时，停止迭代。剩下的样本贴上未知类型的标签，加入训练集再训练。这就完成了未知类型识别的任务。

考虑到代码统一的问题，不管有没有未知类型，统一使用图片右边的方法。

# LSTM 分类器

## 概念

长短期记忆 (Long short-term memory, LSTM) 是一种特殊的RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，相比普通的RNN，LSTM能够在更长的序列中有更好的表现。

LSTM从被设计之初就被用于解决一般递归神经网络中普遍存在的**长期依赖**问题，使用LSTM可以有效的传递和表达长时间序列中的信息并且不会导致长时间前的有用信息被忽略（遗忘）。与此同时，LSTM还可以解决RNN中的**梯度消失/爆炸问题**。

LSTM的设计借鉴了人类对于自然语言处理的直觉性经验。一个直观例子如下：

“这个笔记本非常棒，纸很厚，料很足，用笔写起来手感非常舒服，而且没有一股刺鼻的油墨味；  
更加好的是这个笔记本不但便宜还做工优良，我上次在别家买的笔记本裁纸都裁不好，还会割伤手.....”

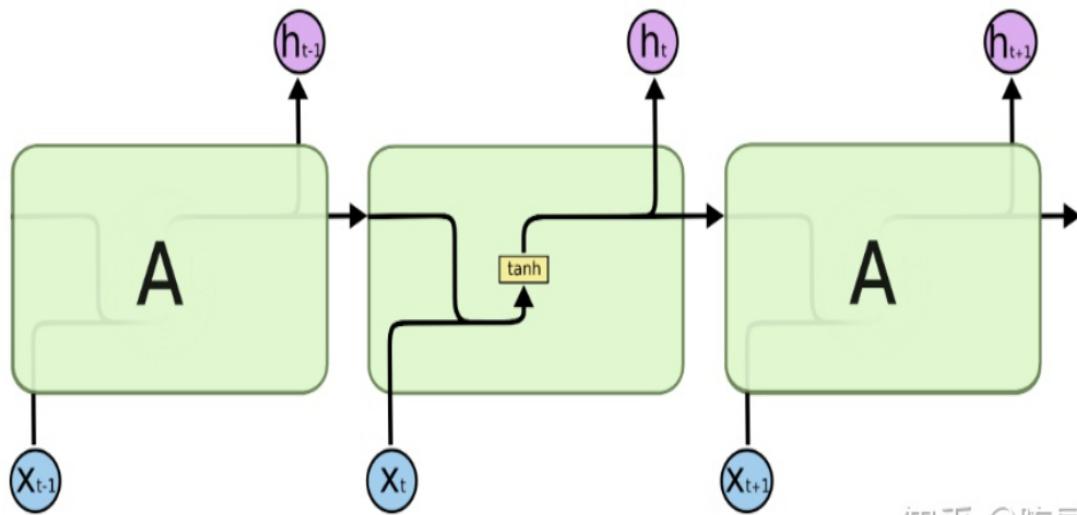
这段话中有几个重要的关键词“纸好”，“没味道”，“便宜”和“做工好”。从这个例子得出：

1. 在一个时间序列中，不是所有信息都是同等有效的，大多数情况存在“**关键词**”或者“**关键帧**”
2. 我们会在从头到尾阅读的时候“自动”**概括**已阅部分的内容并且用之前的内容帮助理解后文

基于以上这两点，LSTM的设计者提出了“长短期记忆”的概念——只有一部分的信息需要长期的记忆，而有的信息可以不记下来。同时，我们还需要一套机制可以动态的处理神经网络的“记忆”，因为有的信息可能一开始价值很高，后面价值逐渐衰减，这时候我们也需要让神经网络学会“遗忘”特定的信息

## 架构

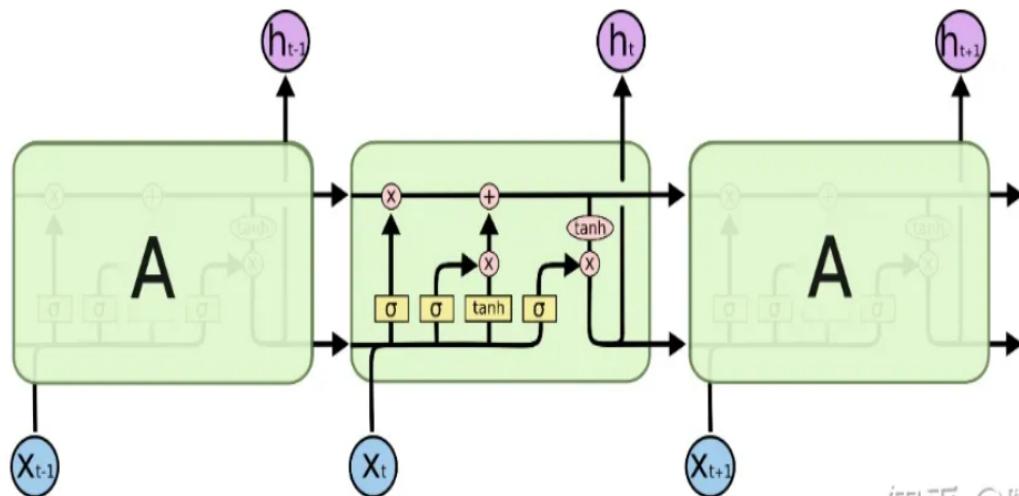
传统RNN的架构如下：



标准 RNN 中的重复模块包含单个层。

知乎 @陈昱

LSTM架构图如下：

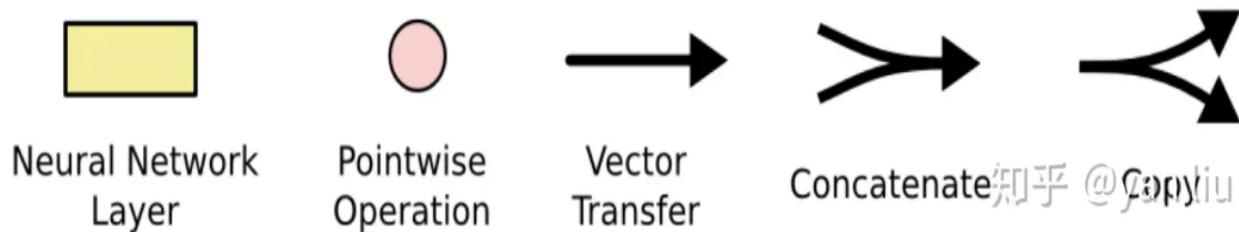


LSTM 中的重复模块包含四个交互层。

知乎 @陈昱

## 数据流向符号及含义

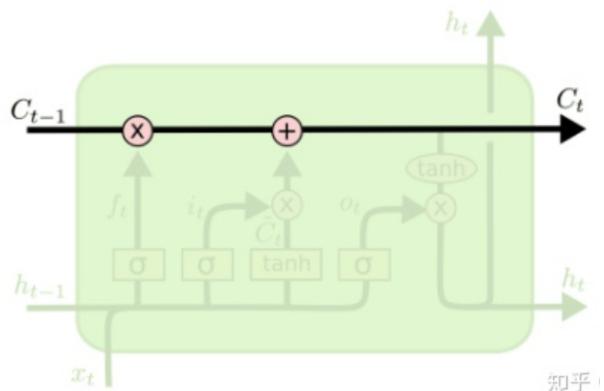
每个黄色方框表示一个神经网络层，由权值，偏置以及激活函数组成；每个粉色圆圈表示元素级别操作；箭头表示向量流向；相交的箭头表示向量的拼接；分叉的箭头表示向量的复制。



知乎 @Copyliu

## 单元状态

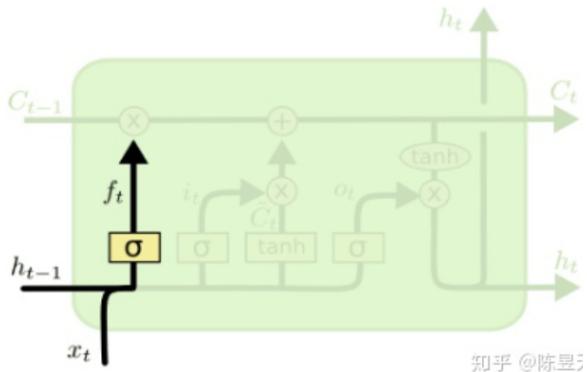
LSTM能够从RNN中脱颖而出的关键就在于上图中从单元中贯穿而过的线——神经元的隐藏态（单元状态），我们可以将神经元的隐藏态简单的理解成递归神经网络对输入数据的“记忆”，用  $C_t$  表示神经元在  $t$  时刻过后的“记忆”，这个向量涵盖了在  $t + 1$  时刻前神经网络对于所有输入信息的“**概括总结**”



$$\text{公式: } C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (\text{都是点乘})$$

知乎

## 遗忘门



知乎 @陈昱

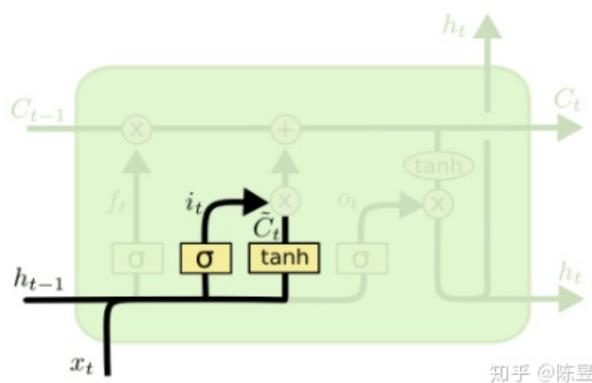
对于上一时刻LSTM中的单元状态来说，一些“信息”可能会随着时间的流逝而“过时”。为了不让过多记忆影响神经网络对现在输入的处理，我们应该选择性遗忘一些在之前单元状态中的分量——这个工作就交给了“遗忘门”

每一次输入一个新的输入，LSTM会先根据新的输入和上一刻的输出**决定遗忘掉之前的哪些记忆**——输入和上一步的输出会整合为一个单独的向量，然后通过  $sigmoid$  神经层，最后点对点的乘在单元状态上。因为  $sigmoid$  函数会将任意输入压缩到  $(0, 1)$  的区间上，我们可以非常直觉的得出这个门的工作原理——如果整合后的向量某个分量在通过  $sigmoid$  层后变为0，那么显然单元状态在对位相乘后对应的分量也会变成0，换句话说，“遗忘”了这个分量上的信息；如果某个分量通过  $sigmoid$  层后为1，单元状态会“保持完整记忆”。不同的  $sigmoid$  输出会带来不同信息的记忆与遗忘。通过这种方式，LSTM可以长期记忆重要信息，并且记忆可以随着输入进行**动态调整**

遗忘门的计算公式如下，其中  $f_t$  表示  $sigmoid$  神经层的输出向量

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (\text{得到的向量的每个元素在 } [0, 1] \text{ 之间})$$

## 记忆门



知乎 @陈昱

记忆门是用来控制是否将在  $t$  时刻（现在）的数据并入单元状态中的控制单位。首先，用  $tanh$  函数层将现在的向量中的**有效信息提取**出来，然后使用（图上  $tanh$  函数层左侧）的  $sigmoid$  函数来控制这些记忆要放“多少”进入单元状态。这两者结合起来就可以做到：

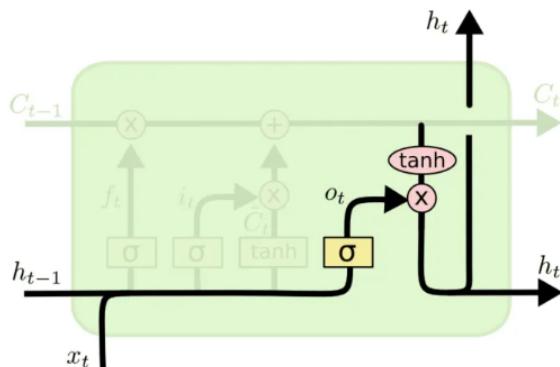
1. 从当前输入中**提取有效信息**
2. **对提取的有效信息做出筛选**，为每个分量做出评级 (0 ~ 1)，评级越高的最后会有越多的记忆进入单元状态

记忆门中的计算公式如下：

$$C'_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (\text{提取信息})$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (\text{筛选信息})$$

## 输出门



输出门，顾名思义，就是LSTM单元用于计算当前时刻的输出值的神经层。输出层会先将当前输入值与上一时刻输出值整合后的向量（也就是公式中的  $[h_{t-1}, x_t]$ ）用  $sigmoid$  函数提取其中的信息，接着，会将当前的单元状态通过  $tanh$  函数压缩映射到区间  $(-1, 1)$  中，最后两者点乘得到当前时刻的输出

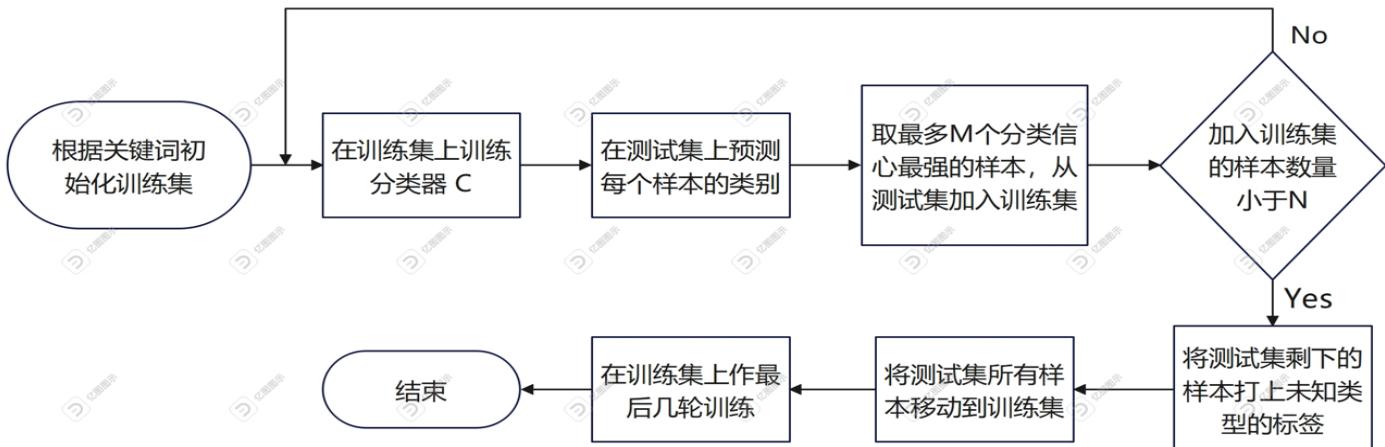
输出门中的计算公式如下：

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (\text{提取当前信息})$$

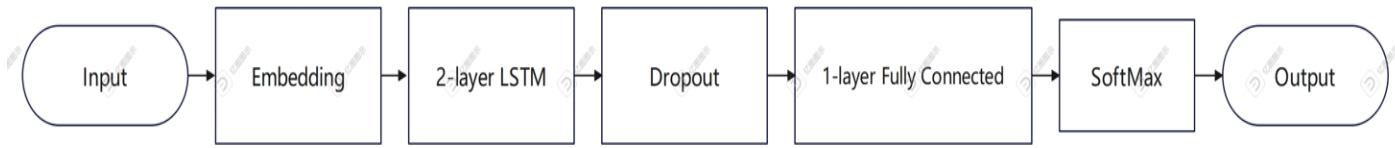
$$h_t = o_t \cdot \tanh(C_t) \quad (\text{点乘})$$

## 2、伪代码，流程图

### 全过程流程图



### 模型结构图



### 初始化训练集：占优众数表决

---

#### Algorithm 1 Init

---

```
procedure INIT(Dataset)
    读取关键词文件
    初始化数组 count，长度为类别数量，元素全 0
    for 所有句子 do
        重置 count 数组
        for 句子的所有单词 do
            if 这个单词是关键词 then
                count[关键词对应的类别索引] ++
            end if
        end for
        if Max(count) > 2 * Sum(count) then
            为这个句子贴伪标签，为 count 的最大值处的索引
        end if
    end for
end procedure
```

---

### 3、关键代码展示（带注释）

#### 数据预处理

```
def preprocess_string(text):
    # 删除标点符号和数字
    text = re.sub('[^\w\s]', '', text)
    text = re.sub('\d+', '', text)
    # 删除 '\' 和 '\" 后面的一个字符
    text = re.sub(r'\\(.)', '', text)
    # 去除停用词
    stop_words = set(stopwords.words('english'))
    tokens = text.split()
    # 去除单个字母的 token
    tokens = [token for token in tokens if len(token) > 1]
    # 去除停用词
    tokens = [token for token in tokens if token.lower() not in stop_words]
    text = ' '.join(tokens)
    # 去除两端的空格并将多个空格合并为一个空格
    text = re.sub('\s+', ' ', text).strip()
    return text
```

#### 占优众数表决

```
for line in dataset:
    # 获取每行的 label 和 text, 省略
    break_flag = False
    class_index = [] # 记录这个句子中, 每个关键词对应的类别
    for word in text.split(' '):
        for i in range(len(keyword)):
            if word in keyword[i]: # 当这个单词是关键词时
                class_index.append(i) # append关键词对应的类别
    # 当这个句子有关键词 而且 句子长度不为 0
    if len(class_index) > 0 and len(text) != 0:
        # 占优众数表决
        counts = np.bincount(class_index) # 计算句子中, 每个类别的关键词的数量
        if 2 * max(counts) > sum(counts): # 当众数占优时, 加入初始训练集
            write_file_1.write(label + '\t' + str(int(np.argmax(counts))) + '\t' + text + '\n')
        else: # 众数不占优时, 加入测试集
            write_file_2.write(label + '\t' + str(-1) + '\t' + text + '\n')
    # 没有关键词, 而且句子长度不为 0
    elif len(class_index) == 0 and len(text) != 0:
        write_file_2.write(label + '\t' + str(-1) + '\t' + text + '\n')
```

# 读取数据，并计算初始类别中心

```
def transform_list(input_list, length): # 将词向量转化为词袋表示
    output_list = [0] * length # 创建一个长度为指定长度的初始列表，全部填充为0
    for i in range(len(input_list)):
        if input_list[i] < length:
            output_list[input_list[i]] += 1
    return output_list

def vec_add(v1, v2): # 串行向量加法
    if len(v1) != len(v2):
        return
    ans = [0 for _ in range(len(v1))]
    for i in range(len(v1)):
        ans[i] = v1[i] + v2[i]
    return ans

def load_data(self): # 数据集类中，读取数据
    with open(self.path, 'r', encoding='utf-8') as file:
        for line in file:
            # 读取真实标签，伪标签，文本
            # 特别特别注意，训练过程不能使用真实标签
            real_label, known_label, text = line.strip().split('\t')
            # 将文本转化为词向量表示，每个token对应一个编号，将单词转化为对应的编号
            # 得出的text_indices是一个数组
            text_indices = [self.word2index.get(word, 0) for word in text.split()]
            self.data.append(text_indices)
            self.real_labels.append(int(real_label))
            self.known_labels.append(int(known_label))

            # 特别注意，如果没有采用距离优化的方案，则下面这段完全删除
            # 计算每个类别的数量，中心（伪标签）
            self.count = [0 for j in range(class_num)] # 每个类别的数量（伪标签）
            self.class_center = [[0 for l in range(vocab_size)] for j in range(class_num)] # 每个类
            # 对每个数据
            for i in range(len(self.data)):
                # 记录每个类别的数量
                self.count[int(self.known_labels[i])] += 1
                # 将词向量转化为词袋表示
                tmp_data = transform_list(self.data[int(self.known_labels[i])], vocab_size)
                # 中心 += 新样本的向量
                self.class_center[int(self.known_labels[i])] = vec_add(self.class_center[int(self.known_labels[i])], tmp_data)
            # 平均，计算中心
            for i in range(class_num):
                for j in range(len(self.class_center[i])):
                    if self.count[i] != 0:
                        self.class_center[i][j] /= self.count[i]
```

特别注意，如果没有采用距离优化的方案，则上面有一段完全删除即可

# 将测试集的最多 M 个样本移动到训练集

```
def move_data(labeled_dataset, unlabeled_dataset, rm, num=500):
    # rm 是所有置信度高于一定阈值的样本
    # rm 是一个元组，第一个元素是置信度，第二个元素是一个元组
    # 这个元组是（真实标签，伪标签，文本）
    for i in range(len(rm)):
        tmp_data = transform_list(rm[i][1][0], vocab_size) # 转化为词袋表示
        dist = [0 for _ in range(len(labeled_dataset.class_center))] # 到训练集每个类别中心的距离
        for j in range(len(dist)): # 计算 样本 到 训练集每个类别中心 的距离
            dist[j] = dist_cal(tmp_data, labeled_dataset.class_center[j])
        factor = sum(dist)
        for j in range(len(dist)): # 将 距离 归一化
            dist[j] /= factor
        # 加权求和
        rm[i][0] = (rm[i][0] - 0.95) * 200
        rm[i][0] += (1 / class_num) / dist[rm[i][2]]
    # 特别注意，如果没有采用距离优化，则上面这段完全删除
    # 对 分类信息 从高到低 排序
    rm.sort(key=lambda x: x[0], reverse=True)
    counter = 0
    for item in rm:
        labeled_dataset.add(item[1][0], item[1][1], item[1][2]) # 将 样本 加入 训练集
        unlabeled_dataset.remove(item[1][0]) # 将 样本 从 测试集 删除
        if counter > num:
            return
        counter += 1
```

特别注意，如果没有采用距离优化，则上面有一段完全删除

# 将一个样本加入训练集

```
def add(self, text_, known_label_, real_label_):
    known_label_ = int(known_label_)
    real_label_ = int(real_label_)
    self.data.append(text_)
    self.real_labels.append(real_label_)
    self.known_labels.append(known_label_)
    # 如果不采用距离优化，则删除下面这段。这段是重新计算类别中心的代码
    tmp_data = transform_list(text_, vocab_size)
    for j in range(len(self.class_center[known_label_])):
        # 类别中心 *= 训练集中该类别样本数量
        self.class_center[known_label_][j] *= self.count[known_label_]
    # 类别中心 += 该样本的 bow 向量
    self.class_center[known_label_] = vec_add(self.class_center[known_label_], tmp_data)
    self.count[known_label_] += 1
    for j in range(len(self.class_center[known_label_])):
        self.class_center[known_label_][j] /= self.count[known_label_] # 平均，计算类别中心
```

## 模型

```
class TextClassifier(nn.Module):
    def __init__(self, num_classes_, vocab_size_, embedding_dim_, hidden_dim_=100, num_layers_=3):
        super(TextClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size_, embedding_dim_)
        self.lstm = nn.LSTM(embedding_dim_, hidden_dim_, num_layers_, batch_first=True, bidirectional=False)
        self.fc = nn.Linear(2*hidden_dim_, num_classes_)
        self.drop = nn.Dropout(p=0.2)
        self.sm = nn.Softmax(dim=1)

    def forward(self, x):
        embedded = self.embedding(x)
        output, _ = self.lstm(embedded)
        last_hidden_state = output[:, -1, :]
        last_hidden_state = self.drop(last_hidden_state)
        output = self.fc(last_hidden_state)
        output = self.sm(output)
        return output
```

## 训练集训练

```
# 有监督学习 (有标签数据)
def supervised_train(model, criterion, optimizer, labeled_loader):
    model.train()
    for i, (texts, real_labels, known_labels) in enumerate(labeled_loader):
        optimizer.zero_grad()
        outputs = model(texts.cuda())
        # 在伪标签上训练 (伪标签不一定是正确的)
        loss = criterion(outputs, known_labels.cuda())
        loss.backward()
        optimizer.step()
```

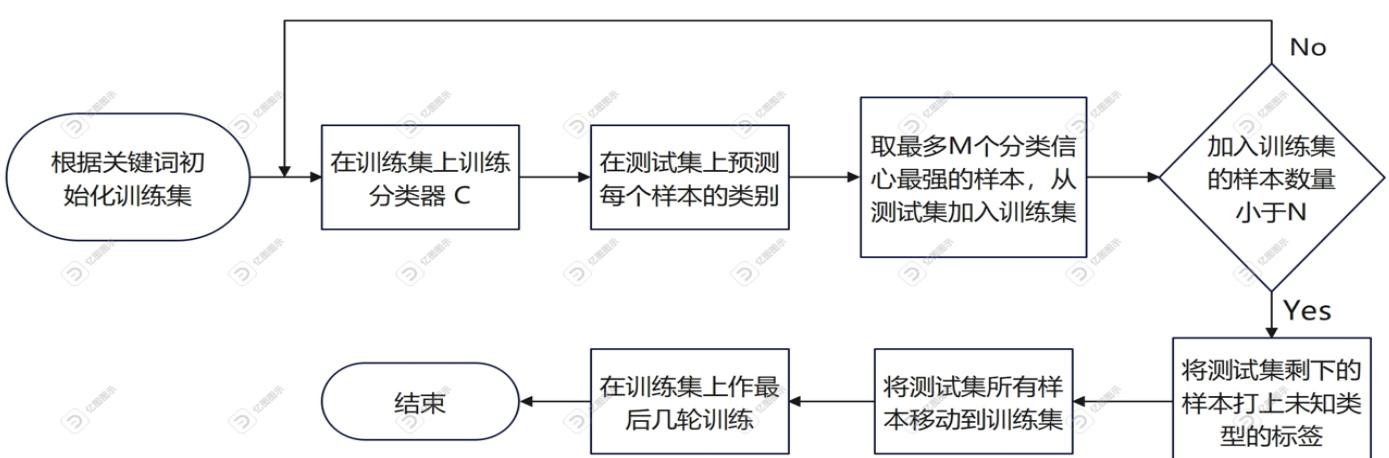
## 测试集测试

```
def unsupervised_train(model, criterion, optimizer, unlabeled_loader):
    to_remove = []
    model.eval() # 关闭 dropout
    with torch.no_grad():
        for i, (texts, real_labels, known_labels) in enumerate(unlabeled_loader):
            outputs = model(texts.cuda())
            # 选择置信度最高的类别, 作为类别伪标签
            pseudo_labels = torch.argmax(outputs, dim=1)
            # 置信度高于阈值, 将样本加入 to_remove
            if outputs[0, int(torch.argmax(outputs, dim=1))].detach().cpu().numpy() > 0.97:
                to_remove.append([outputs[0, int(torch.argmax(outputs, dim=1))].detach().cpu(),
                                  (texts[0], real_labels[0], pseudo_labels.detach().cpu)])
    return to_remove
```

# 主函数

```
previous_len = unlabeled_dataset.__len__()
for epoch in range(num_epochs):
    if epoch == 0: # 第一轮先训练多几次
        for i in range(25):
            res = supervised_train(model, criterion, optimizer, labeled_loader)
    else:
        res = supervised_train(model, criterion, optimizer, labeled_loader)
    # 在测试集上测试, rm 是高于置信度阈值的样本
    rm = unsupervised_train(model, criterion, optimizer, unlabeled_loader)
    # 将 rm 中最多 750 个样本加入训练集, 从测试集删除
    move_data(labeled_dataset, unlabeled_dataset, rm, 750)
    # 重新加载 data_loader
    labeled_loader = DataLoader(labeled_dataset, batch_size=batch_size, shuffle=True)
    unlabeled_loader = DataLoader(unlabeled_dataset, batch_size=batch_size, shuffle=True)
    # 当加入训练集的样本数量小于250, 则退出, 认为测试集剩下的样本属于未知类型
    if previous_len - unlabeled_dataset.__len__() < 250:
        break
    else:
        previous_len = unlabeled_dataset.__len__()

# 认为测试集剩下的样本属于未知类型
for i, (texts, real_labels, known_labels) in enumerate(unlabeled_loader):
    # 将样本打上未知类型的标签, 加入训练集
    outputs = model(texts.cuda())
    to_remove.append([outputs[0, int(torch.argmax(outputs, dim=1))].detach().cpu().numpy(), (tex
# 加入训练集
move_data(labeled_dataset, unlabeled_dataset, to_remove, unlabeled_dataset.__len__())
labeled_loader = DataLoader(labeled_dataset, batch_size=batch_size, shuffle=True)
# 最后进行 10 轮训练
for i in range(10):
    res = supervised_train(model, criterion, optimizer, labeled_loader)
    print(res)
```



## 4、创新点&优化

### 训练集的初始化方法

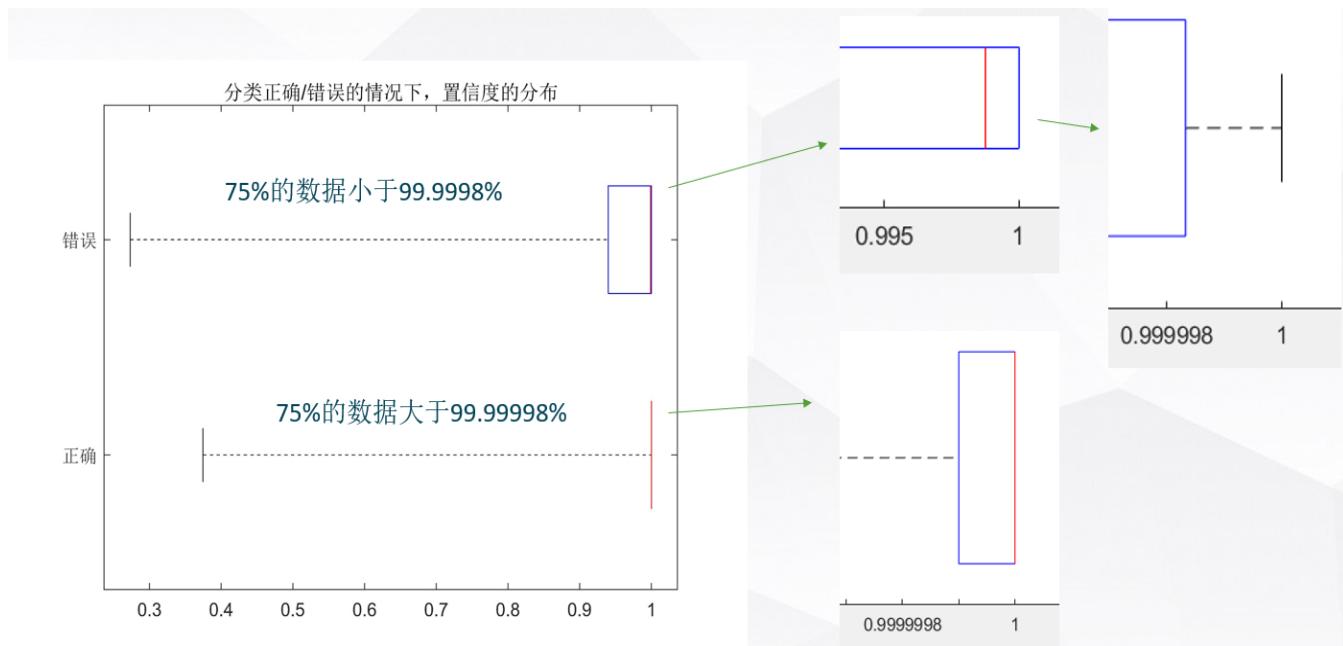
讨论了三种利用关键词初始化训练集的方法，得出了“占优众数表决”这种最好的方法，可以有效提高初始训练集的准确率，从而提高模型在整个数据集上的表现。（具体数据没有记录）

以下是三种方法生成的初始训练集的样本数量和准确率

	朴素方法	众数表决	占优众数表决
20ns	11253 58.58%	11253 59.79%	8117 67.70%
agnews	6294 78.92%	6294 78.20%	6162 79.42%

### 分类信心的探讨

采用小实验的方法，探讨了用置信度表示分类信心的可行性，这一点在原理部分有详细说明。



最终由贝叶斯公式得出，用置信度衡量分类信心是合理的。

认为这一点的探讨是非常必要的，可能很多人直接使用置信度作为分类信心，而不去探讨这种做法的合理性。认为每一种做法，都要搞清楚原理，弄清楚为什么可以这样，要详细分析合理性与可行性。

# 分类信心评分的优化

上面我们作了分析，认为从贝叶斯的视角，用置信度衡量分类信心是合理的。

但是经过思考，认为这种正相关关系具有一定的不确定性，这是因为取得一个置信度非常大的数据时，它仍有可能是分类错误的。于是思考能不能加入一些别的信息来衡量分类信心，而且这个信息是不依赖于我们的神经网络的。

原来的方法，每个样本的评分就是置信度。现在给这个评分增加一项内容——到类别中心的距离，评分计算公式如下

$$Score = \alpha \cdot confidence + \beta \cdot distance\_score$$

这意味着每个样本的评分是置信度和距离评分的加权和。

距离评分计算公式如下：

$$distance\_score = \frac{\text{到所有类别中心的距离总和}}{\text{到该类别中心的距离}}$$

基本思想是：距离该类中心近，距离其他类中心远的， $distance\_score$  的分值较高

这里采用 Bag\_Of\_Words 向量来表示每个样本，维护训练集中每个类别的中心，采用欧式距离来衡量样本之间的距离。

这样的做法效率会比较低，特别是当词表长度很大时。设词袋向量长度（词表长度）为  $L$ ，测试集样本数为  $N$ ，类别数为  $C$ ，则将  $N$  个样本加入训练集的过程，时间复杂度为  $O(N \cdot C \cdot L)$

而且这种方法对数据集的分布较敏感：20ns是大类别套小类别的风格，在20ns上做这个优化反而使准确率降低 3%；但是在 agnews 这种正常风格的数据集上，能提升 2% 的准确率。

以下是无未知类型的结果

20ns M=750 优化前

正确数量： 11854 总样本数： 18299  
准确率： 0.6477949614733045

agnews M=1000 优化前

正确数量： 34286 总样本数： 40000  
准确率： 0.85715

20ns M=750 优化后

正确数量： 11192 总样本数： 18299  
准确率： 0.6116181212088092

agnews M=1000 优化后

正确数量： 35047 总样本数： 40000  
准确率： 0.876175

# 三、实验结果与分析

## 1、实验结果展示示例

### 20ns 优化前后

正确数量： 11854 总样本数： 18299  
准确率： 0.6477949614733045  
精确率： 0.6358605623245239  
召回率： 0.6463921666145325  
F1分数： 0.6334478855133057

正确数量： 11192 总样本数： 18299  
准确率： 0.6116181212088092  
精确率： 0.5969251394271851  
召回率： 0.5995970964431763  
F1分数： 0.5877388715744019

左边是优化前的，右边是优化后的。20ns 数据集是大类包小类的风格，上述距离评分不能很好地反映数据集的特征，因此优化后各种指标反而减低了

### agnews 优化前后

正确数量： 34286 总样本数： 40000  
准确率： 0.85715  
精确率： 0.8571500182151794  
召回率： 0.8568945527076721  
F1分数： 0.8562696576118469

正确数量： 35047 总样本数： 40000  
准确率： 0.876175  
精确率： 0.8761749863624573  
召回率： 0.8763105869293213  
F1分数： 0.8753064274787903

左边是优化前的，右边是优化后的。agnews 数据集正常风格，上述距离评分可以较好地反映数据集的特征，因此优化后各种指标有所提升

### 20ns\_unk 和 agnews\_unk 优化前

正确数量： 12226 总样本数： 18299  
准确率： 0.6681239411989727  
精确率： 0.6565945744514465  
召回率： 0.6649064421653748  
F1分数： 0.6544716358184814

正确数量： 35265 总样本数： 40000  
准确率： 0.881625  
精确率： 0.8816249966621399  
召回率： 0.8811100721359253  
F1分数： 0.8809875845909119

因为时间关系没有跑优化后的结果

可以看到，未知类型识别 的效果，反而比 无未知类型识别 的效果 还要好。这是因为 未知类型识别 的最后，对训练集又做了 10 轮训练；而 无未知类型识别的代码中，最后没有这个训练操作。如果加上这个最后的训练，则无未知类型识别的准确率应该高于 未知类型识别的准确率？

## 四、参考资料

<https://pytorch.org/docs/stable/nn.html> PyTorch 官方文档

<https://zhuanlan.zhihu.com/p/123857569> LSTM 详解

这里参考资料比较少，大部分东西都是自己捣鼓出来的