

# 北京交通大学

## 网络内容安全报告 《基于机器学习的中文文本情感 分析系统设计》

学    院：电子信息工程学院

成员 1：张恒瑜 19211090

成员 2：杜飞 19211325

成员 3：李卓音 19211173

指导老师：熊菲

北京交通大学

2022 年 6 月

# 目录

1.研究背景 .....	4
2.相关技术 .....	4
2.1 数据爬虫 .....	5
2.2 中文分词 .....	5
2.2.1 中文分词概念 .....	5
2.2.2 中文分词基本算法原理 .....	6
2.2.3 大规模分词 .....	7
2.2.4 具体算法实现及性能对比 .....	7
2.3 去停用词 .....	10
2.4 生成词向量 .....	11
2.4.1 什么是生成词向量 .....	11
2.4.2 word2vec .....	11
2.4.3 nn.Embedding .....	12
2.5 分类神经网络 .....	13
2.5.1 TextCNN .....	13
2.5.2 AttTextCNN .....	14
2.5.3 损失函数及优化算法 .....	16
3.系统总体设计 .....	17
3.1 后端开发 .....	17
3.2 前端界面 .....	18
4.模块详细设计 .....	18
4.1 数据爬取 .....	18
4.1.1 任务分析 .....	18
4.1.2 具体实现 .....	19
4.1.3 改进措施 .....	22
4.2 中文分词 .....	23
4.2.1 jieba 分词 .....	23
4.2.2 pkuseg 分词 .....	24
4.2.3 SnowNLP 分词 .....	25
4.3 生成词向量 .....	25
4.3.1 Word2Vec .....	25
4.3.2 Embedding .....	27
4.4 分类神经网络 .....	28
4.4.1 TextCNN .....	28
4.4.2 AttTextCNN .....	28
4.4.3 预测方法 .....	29
4.5 系统界面设计 .....	29
5.数据结果分析 .....	32
6.系统使用说明 .....	33
6.1 界面外观介绍 .....	33
6.1.2 主界面介绍 .....	33
6.1.3 子界面介绍 .....	33

6.2 使用方法示例 .....	34
6.2.1 模型选取 .....	34
6.2.2 加载模型 .....	34
6.2.3 人机交互实时文本预测 .....	35
7. 系统评价及改进 .....	35
8. 总结感想与成员分工 .....	36

## 前言

整个工程代码已全部上传至张恒瑜本人的 Github 上供大家参考。

**Github 链接:** <https://github.com/qiku-zhang/Emotion-analysis-of-Chinese-text-based-on-machine-learning>

文件介绍:

**Start.py:** 用来启动系统界面, 实行人机交互;

**Word2vec\_train.py:** 对使用到 word2vec 词向量生成方法的模型进行训练;

**Embedding\_train.py:** 对使用到 embedding 词向量生成方法的模型进行训练;

**Stop\_words.txt:** 停用词表;

文件夹介绍:

**model:** 存放了神经网络的模型代码

--**em\_textcnn.py:** 针对使用到 embedding 词向量生成方法的模型而写

--**textcnn.py:** 针对使用到 word2vec 词向量生成方法的模型而写

**solver:** 存放了对数据的一些操作代码

--**cut.py:** 使用不同的分词手段对文本进行分词

--**dataloader.py:** 生成自己的数据集的类

--**predict.py:** 对文本进行预测的代码函数

**保存模型:** 存放了训练好的各个模型

--**.pt:** 以 pt 结尾的为神经网络的模型

--**.model:** 以 model 结尾的为 word2vec 的词向量模型

**数据集:** 存放了原始的数据集, 以及使用不同分词方法分词后的数据集

**系统界面:** 存放了绘制好的界面 ui 文件以及对应的 py 文件

--**untitled.ui:** 主界面的 ui 文件

--**untitled1.ui:** 子界面的 ui 文件

--**srs.py:** 主界面的代码文件

--**srs1.py:** 子界面的代码文件

# 1.研究背景

科技改变生活。近年来，随着网络的普及以及人们对生活便利性的追求越来越高，越来越多的人选择淘宝、京东等电商进行购物，当然，外卖平台也不断发展与完善。人们在进行网络购物或者点外卖时，往往会根据其他用户的评价对事物进行判断。

在我国电子商务飞快发展的背景下，基本上所有的电子商务网站都支持消费者对产品的相关内容（商品、服务、卖家）等进行打分和发表评论，消费者对商品的评价及打分也在一定程度上影响着其他潜在用户之间的选择。消费者之间通过评论打分等方式进行沟通及交流，在网络电商或外卖平台上发布大量的留言和评论，这已经成为互联网的一种流行形式，而这种流行趋势必然会给平台带来海量的信息。对于卖家而言，可以从评论信息中获取客户的实际需求及用户体验，以改变产品品质，提高自身的竞争力；另一方面，对于一些未知的体验产品，用户为了降低自身的风险更加倾向于得到其他用户的意见与看法，借鉴其他用户的评论信息，以做出更好的购买决策。

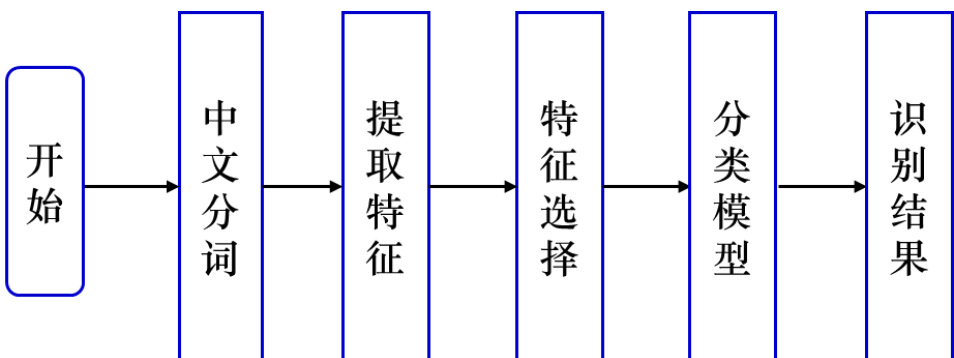
考虑到较为传统的人工分析的低下效率，传统的手段已经没有办法满足现在快速发展的行业需求，所以现在越来越多的人开始使用自然语言处理这一近几年才得以快速发展的技术对互联网上的大规模文本来进行分析其包含的各类情感。目前，由于现在互联网上文本信息数据量极为庞大且句式语法不规范，要冲这些数据中获取其真正的情感是非常困难的，但若仅仅依靠人工方式来判断，这是一件费时费力的事情，那么就需要计算机的帮助。使用自然语言处理这一方式是解决这一问题的有效途径之一，通过这一技术分析文本包含的情感倾向，从而判断信息发布者的主观情感，杂乱无章的海量数据就被归纳整理，成为了可直接使用的有价值的数据。

情感分析是对带有情感色彩的主观性文本进行分析、处理、归纳和推理的过程，我们基于机器学习的方法对获取到的中文文本进行情感分析，以便于更好地对用户评论进行分析，从而帮助商家及顾客做出更好的判断。

我们本次的系统设计以外卖平台上的评论为例，设计了一个可以识别外卖评论情感的系统，节省了人工的分类成本。

# 2.相关技术

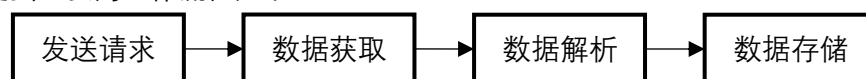
基于机器学习的中文文本情感分析方法的主要流程如下所示：



对于输入进来的文本，我们经过分词，然后提取特征，在进行神经网络分类，最终得到预测结果。基于此，我们先介绍本次系统设计中具体使用到的相关技术。

## 2.1 数据爬虫

爬虫的本质，其实就是让脚本程序模拟人为操作，通过浏览器去访问网页（网站）。对于网站服务器来说，人为的通过浏览器访问和通过脚本程序爬虫本质上没有太大区别（反爬虫机制除外）。不过不同的是，人为访问网页，获取的是一个网页（人眼看到的页面）；而脚本获取的信息，是整个页面的源码（chrom 浏览器按 f12），可以对这些信息进行更加细致的分析。爬虫主要的工作流程如下：



### ①发送请求

通过 HTTP 库向目标站点发送一个 Request，请求包含额外的头部、防盗链，Cookies 等信息，然后等待服务器响应。这个过程其实就相当于程序作为一个浏览的客户端，向服务器端发送了一次请求。

### ②数据获取

发送请求后会得到一个 Response，Response 的内容就是想要获取的页面内容。这个过程就是服务器接收客户端的请求，经过解析后发送给浏览器的网页 HTML 文件。在获取信息过程中可以通过打印 response 的数值来判断是否能够获取到数据，当标号为【403】时代表请求失败，当标号为【200】时，代表请求成功。

### ③数据解析

获取网页源代码后，接下来就是从中提取我们想要的信息，本项目中，采用正则表达式的方法从响应得到的 json 文件中获取想要的信息。通过正则表达式匹配，我们可以从文本字符串中获取我们想要的特定部分，过滤得到有用的数据集信息。

### ④数据存储

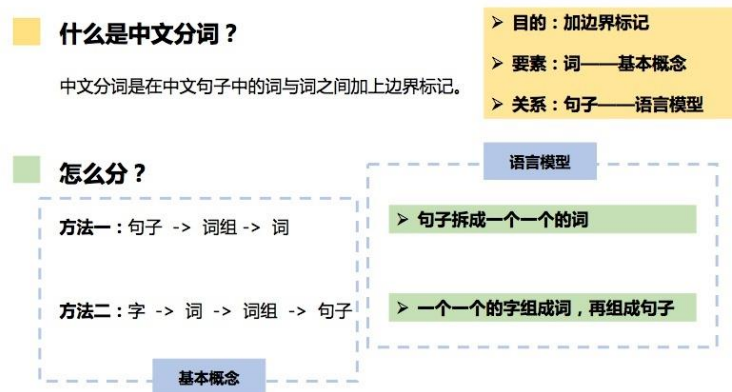
将获取的数据存入.csv 文件，编码方式为 utf-8，后期会转成 txt 文本，便于我们对神经网络进行训练。

## 2.2 中文分词

### 2.2.1 中文分词概念

中文分词，即 Chinese Word Segmentation，即将一个汉字序列进行切分，得到一个个单独的词。

中文分词与英文分词有很大的不同，对英文而言，一个单词就是一个词，而汉语是以字为基本的书写单位，词语之间没有明显的区分标记，需要人为切分。



中文分词的本质是划分词的边界，但同时，也面临着分词规范、歧义切分、新词识别等挑战。现有的分词算法可分为三大类：基于字符串匹配的分词方法、基于理解的分词方法和基于统计的分词方法。按照是否与词性标注过程相结合，又可以分为单纯分词方法和分词与标注相结合的一体化方法。

## 2.2.2 中文分词基本算法原理

我们对上述的三类算法做出大体的介绍：

### ①字符匹配

这种方法又叫做机械分词方法，它是按照一定的策略将待分析的汉字串与一个“充分大的”机器词典中的词条进行匹配，若在词典中找到某个字符串，则匹配成功（识别出一个词）。按照扫描方向的不同，串匹配分词方法可以分为正向匹配和逆向匹配；按照不同长度优先匹配的情况，可以分为最大（最长）匹配和最小（最短）匹配；常用的几种机械分词方法如下：

- 1) 正向最大匹配法（由左到右的方向）；
- 2) 逆向最大匹配法（由右到左的方向）；
- 3) 最少切分（使每一句中切出的词数最小）；
- 4) 双向最大匹配法（进行由左到右、由右到左两次扫描）。

### ②理解法

通过让计算机模拟人对句子的理解，达到识别词的效果。其基本思想就是在分词的同时进行句法、语义分析，利用句法信息和语义信息来处理歧义现象。

它通常包括三个部分：分词子系统、句法语义子系统、总控部分。在总控部分的协调下，分词子系统可以获得有关词、句子等的句法和语义信息来对分词歧义进行判断，即它模拟了人对句子的理解过程。这种分词方法需要使用大量的语言知识和信息。由于汉语语言知识的笼统、复杂性，难以将各种语言信息组织成机器可直接读取的形式，因此目前基于理解的分词系统还处在试验阶段。

### ③统计法

该方法的主要思想：词是稳定的组合，因此在上下文中，相邻的字同时出现的次数越多，就越有可能构成一个词。因此字与字相邻出现的概率或频率能较好地反映成词的可信度。可以对训练文本中相邻出现的各个字的组合的频度进行统计，计算它们之间的互信息。互信息

体现了汉字之间结合关系的紧密程度。当紧密程度高于某一个阈值时，便可以认为此字组可能构成了一个词。该方法又称为无字典分词。

该方法所应用的主要的统计模型有：N 元文法模型 (N-gram)、隐马尔可夫模型 (Hidden Markov Model, HMM)、最大熵模型 (ME)、条件随机场模型 (Conditional Random Fields, CRF) 等。

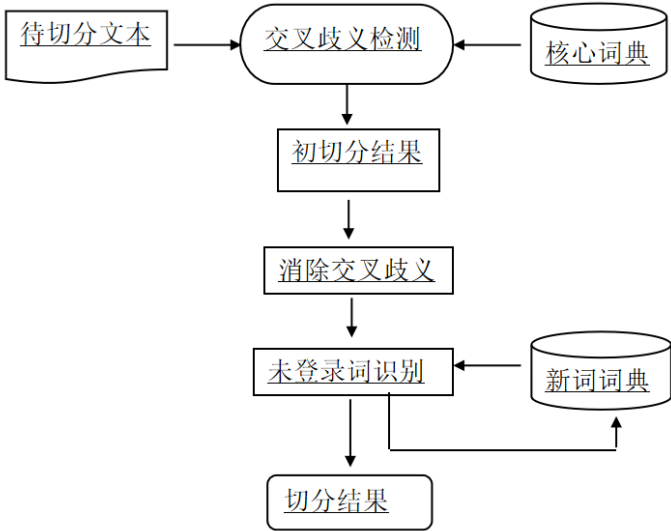
在实际应用中此类分词算法一般是将其与基于词典的分词方法结合起来，既发挥匹配分词切分速度快、效率高的特点，又利用了无词典分词结合上下文识别生词、自动消除歧义的优点。

### 2.2.3 大规模分词

在当前的信息检索技术中，中文切分是很有必要的，但当分词用于与大规模信息检索时，往往存在一下问题：1.是否需要按语言学意义上的词进行切分；2.文档和查询二者的切分方法是否需要一致；3.是否检索系统使用的分词算法切分精度越高其检索效果就越好。在基于字的切分中，有单字切分、二元切分和交叉二元切分；在基于词的切分中，常使用基于词典的匹配和基于统计的方法。在大规模中文信息检索中，对分词算法就有如下几个方面的要求：

- 1.分词算法的时间性能要比较高。
- 2.分词正确率的提高并不一定带来检索性能的提高。
- 3.切分的颗粒度仍然可以依照长词优先准则，但是需要在查询扩展层面进行相关后续处理。
- 4.未登录词识别的准确率要比召回率更加重要。

于是，我们得到对大规模文档分词的基本算法流程如下：



### 2.2.4 具体算法实现及性能对比

在了解算法的基本原理之后，我们做出几种具体中文分词算法的比较，在我们的系统中，有 jiaba 分词、pkuseg 和 snowNLP 三种算法的实现，但常用的算法还有 HanLP、FoolNLTK、LTP、THULAC 等算法，对他们做出一定的概述，如下图所示：

	全称	研发	时间	Star
jieba	结巴分词	fxsjy	2012.09	22.8k
HanLP	汉语言处理包	hankcs	2014.10	19.3k
snownlp	中文类库	isnowfy	2013.11	4.7k
FoolNLTK	中文处理工具包	rockyzhengwu	2017.12	1.5k
LTP	哈工大语言云	哈工大	2015.03 (pyltp)	1.3k
THULAC	清华中文分词	清华	2016.05 (THULAC-Python)	1.3k

GitHub数据来源截止：2020年5月13日

然后，我们对我们的系统中采用的三种算法做出具体的比较，先列出大体的比较结果如下表所示：

分词方法	基本原理	适用	优点	准确度	用时
Jieba	基于词频查找最大概率路径实现	支持繁体分词，支持自定义词典	容易上手操作，原理简单	较高	比 pkuseg 短
Pkuseg	通过预训练和数据库模型进行分词	支持细分领域分词	支持用户自训练模型，简单易用	高	长
SnowNLP	结合了语言模型和统计机器学习模型的序列标注	支持词性标注、情感分析、拼音转换、关键词和摘要生成	适用范围大	较低	位于三者之间

下面再对这三种算法做出具体的介绍：

### ①jieba 分词

jieba 分词器提供 4 种分词模式，并且支持简体/繁体分词、自定义词典、关键词提取、词性标注。

**1.精确模式：**该模式会将句子最精确地切分开，适合在文本分析时使用。

**2.全模式：**该模式会将句子中所有成词的词语都扫描出来，速度也非常快，缺点是不能解决歧义问题，有歧义的词语也会被扫描出来。

**3.搜索引擎模式：**该模式会在精确模式的基础上对长词再进行切分，将更短的词语切分出来。在搜索引擎中，要求输入词语的一部分也能检索到整个词语相关的文档，所以该模式适用于搜索引擎分词。

**4.Paddle 模式：**模式利用 PaddlePaddle 深度学习框架，训练序列标注网络模型实现分词，

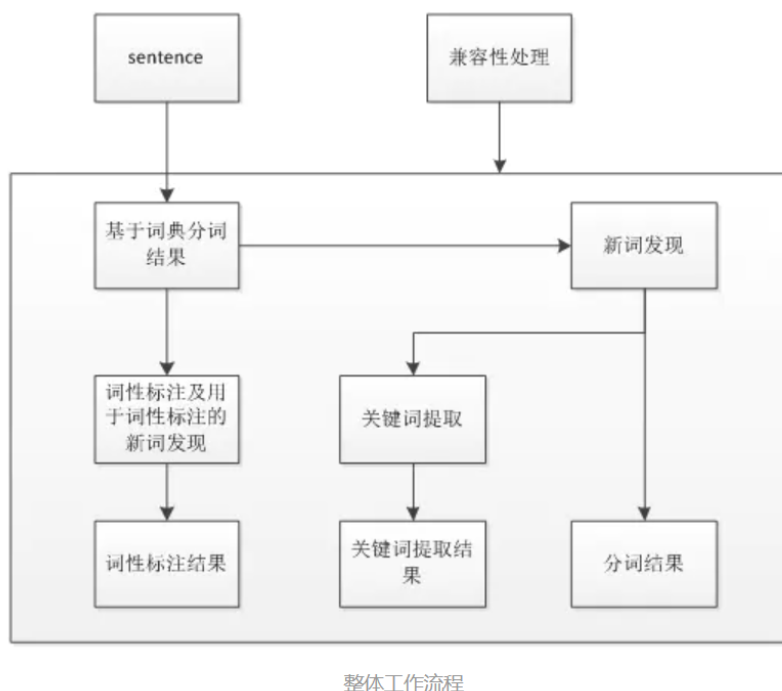


同时支持词性标注。

jieba 分词中，首先通过对照典生成句子的有向无环图，再根据选择的模式不同，根据词典寻找最短路径后对句子进行截取或直接对句子进行截取。对于未登录词（不在词典中的词）使用 HMM 方法新词发现。

jieba 分词主要通过词典来进行分词及词性标注，两者使用了一个相同的词典。正因如此，分词的结果优劣将很大程度上取决于词典，虽然使用了 HMM 来进行新词发现。

jieba 分词包整体的工作流程如下图所示：



## ②Pkuseg 分词

Pkuseg 是北大开源的一种分词方法，主要基于经典的 CRF 模型，辅以 ADF 训练方法 (Sun, et al., 2012) 和精调的特征，实现更快的训练速度、更高的测试效果和更好的泛化能力，具有以下的特点：

**多领域分词：**不同于以往的通用中文分词工具，此工具包同时致力于为不同领域的数据提供个性化的预训练模型。根据待分词文本的领域特点，用户可以自由地选择不同的模型。目前支持了新闻领域，网络文本领域和混合领域的分词预训练模型，同时也拟在近期推出更多的细领域预训练模型，比如医药、旅游、专利、小说等等。

**更高的分词准确率：**相比于其他的分词工具包，当使用相同的训练数据和测试数据，pkuseg 可以取得更高的分词准确率。

**支持用户自训练模型：**支持用户使用全新的标注数据进行训练。

## ③SnowNLP 分词

SnowNLP 是一个常用的 Python 文本分析库，是受到 TextBlob 启发而发明的。由于当前自然语言处理库基本都是针对英文的，而中文没有空格分割特征词，Python 做中文文本挖掘较难，后续开发了一些针对中文处理的库，snowNLP 由此诞生。

它的主要功能包括：中文分词、词性标注、情感分析、文本分类、转换拼音、转繁体字、提取文本关键词和摘要等。

我们来看 jieba 分词和 SnowNLP 分词的一个具体比较示例，如下所示：

```
1 from snownlp import SnowNLP
2 import jieba
3
4 s1 = SnowNLP(u"这本书的质量真不太好！")
5 print("SnowNLP:")
6 print(" ".join(s1.words))
7
8 s2 = jieba.cut(u"这本书的质量真不太好！", cut_all=False)
9 print("jieba:")
10 print(" ".join(s2))
```

分词例子

```
SnowNLP:
这 本 书 的 质 量 真 不 太 好 ！
jieba:
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\msi\AppData\Local\Temp\jieba.cache
这 本 书 的 质 量 真 不 太 好 ！
Loading model cost 0.816 seconds.
Prefix dict has been built successfully.
```

实现结果

我们发现，在上面具体的例子实现过程中，SnowNLP 的分词结果似乎过于僵硬，将很多词语分成了单个的字，而相比较而言 jieba 分词的准确率是优于 SnowNLP 的，分出来的语句更加的符合我们的语言习惯。

三者的更多比较我们放在后续模块具体设计中给大家呈现。

## 2.3 去停用词

在进行分词之后，常常会产生一些对结果没有影响的停用词，这时我们就需要将其去除，以便于后续的处理。

停用词 (StopWords) 是指没有意义的词，比如“啊”、“哎”、“唉”、“吧”、“这”、“那”、“的”、“了”、“在”等语气词、介词、助词、代词等，去除之后对情感分析没有实际影响，甚至还可以降低数据噪声、特征向量维度、提高分类准确性等效果。

特殊字符 (Specialsymbols) 是指数学符号、单位符号、货币符号、日文片假名、汉语注音符号、制表符等使用频率较少且难以输入的一类字符，比如“勺”、“≠”、“ă”等。这一些特殊字符在自然语言处理中无实际意义，去除亦不影响分类效果，同时提升了数据的有效词占比。此时我们就需要将此类词语去除，具体的步骤大致是：

导入停用词表->在分词之后，对分词的结果进行循环判断是否分词结果在停用词表内->如果在，则将其去除；否则留用，继续循环。

## 2.4 生成词向量

### 2.4.1 什么是生成词向量

首先要明确的是，为什么要生成词向量，以及词向量有什么用。

我们现在得到了中文分词后的序列，即每个句子由几个单词组成，但是显然计算机还是不太处理好这些词语，我们需要将这些单词转化成计算机可以处理的数值，即词向量。

词向量就是用来将语言中的词进行数学化的一种方式，顾名思义，词向量就是把一个词表示成一个向量。一个词怎么表示成一个向量是要经过一番训练的，训练方法较多，word2vec 就是其中一种。要注意的是每个词在不同的语料库和不同的训练方法下，得到的词向量可能是不一样的。

由于是用向量表示，而且用较好的训练算法得到的词向量的向量一般是有空间上的意义的，也就是说，将所有这些向量放在一起形成一个词向量空间，而每一向量则为该空间中的一个点，在这个空间上的词向量之间的距离度量也可以表示对应的两个词之间的“距离”。所谓两个词之间的“距离”，就是这两个词之间的语法，语义之间的相似性。

我们这里介绍两类不同的生成词向量的方式，第一种是单独的对语料库进行训练的模型，典型的代表有 word2vec，第二种是将词向量生成模型和后面的分类网络一起进行训练，目的是为了整个训练网络的损失函数达到最小，典型的方法有 nn.embedding 函数。

下面我们对这两类方法分别进行介绍。

### 2.4.2 word2vec

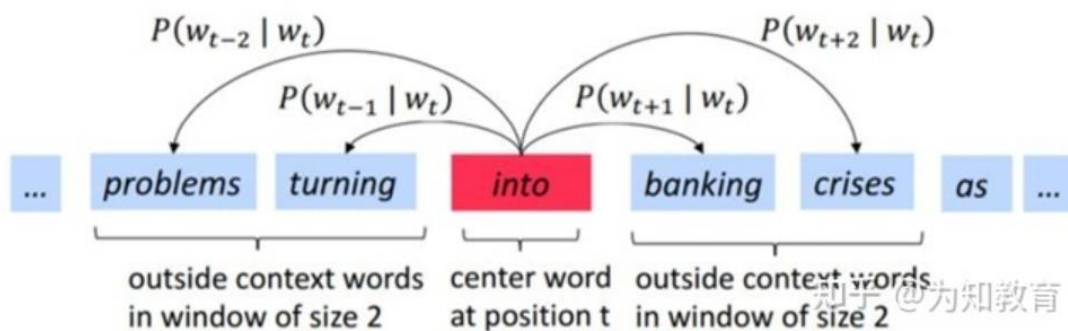
Word2Vec 是一种运用极其广泛的词向量生成模型，它是自主进行训练，单独进行拟合的模型。

由于句子之中相近的词之间是有联系的，比如今天后面经常出现上午、下午。所以它的基本思想就是用词来预测词。准确的说，word2vec 仍然是一种编码方式，将一个个的词给编码成向量，但是被他编码而成的向量并不是随便生成的，而是能够体现这些单词之间的关系，如相似性等。

Word2vec 主要包含两个模型：①跳字模型(skip-gram)：用当前词来预测上下文。②连续词袋模型 (CBOW)：通过上下文来预测当前值。这两种模型是一个相反的逻辑关系，我们最终系统中选取的是比较常见的跳字模型，下面对跳字模型进行讲解。

**跳字模型**的基本思想是在每一次迭代中都取一个词作为中心词汇，尝试去预测它一定范围内的上下文词汇。

所以这个模型定义了一个概率分布：给定一个中心词，某个单词在它上下文中出现的概率。我们会选取词汇的向量表示，从而让概率分布值最大化。重要的是，这个模型对于一个词汇，有且只有一个概率分布，这个概率分布就是输出，也就是出现在中心词周围上下词的一个输出。大致结构如下图所示：



拿到一个文本，遍历文本中所有的位置，对于文本中的每个位置，我们都会定义一个围绕中心词汇大小为  $2m$  的窗口，这样就得到了一个概率分布，可以根据中心词汇给出其上下文词汇出现的概率。

现假设一个已知的句子是我们的一个样本，我们要进行第一次迭代，在迭代的过程中，我们的损失函数（或者说目标函数）为：

$$\prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} P(w^{(t+j)} | w^{(t)})$$

该函数又称似然函数，表示在给定中心词的情况下，在  $2m$  窗口内的所有其他词出现的概率（ $T$  表示词库中的所有词的总数）。我们的目标就是要通过调节参数，从而最大化该似然函数。接下来，我们对该函数取负对数，且除以  $T$ ，得到了新的损失函数（对数似然函数）：

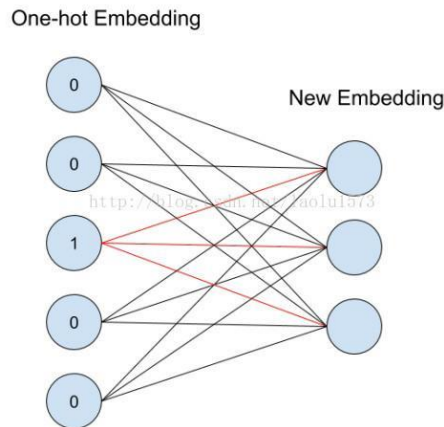
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m} \log P(w_{t+j} | w_t)$$

那么我们的任务就变成了最小化这个损失函数。在进行最小化操作后，就得到了生成词向量。因此，这个模型其实很类似于我们的神经网络，只需要经过多轮的迭代就可以让损失函数最小化，从而训练完成我们的模型。

### 2.4.3 nn.Embedding

Embedding 类是一个非常常见的 pytorch 包，他经常被用来搭建 NLP 网络的结构，因此它虽然是一个生成词向量的工具，但是通常作为网络的一部分，用来和网络一起进行拟合。下面我们对它的实现原理进行阐述。

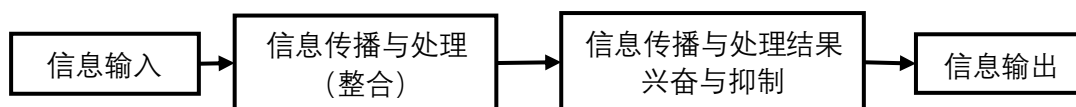
由于 Embedding 是放在网络中进行训练的，因此它的目的其实就是生成一个词语的表示矩阵。首先将输入的词语转换成我们熟悉的 one-hot 编码方式输入，然后再给他随机乘上一个矩阵（这个矩阵的数值是进行随机初始化的），我们就得到了一个词向量矩阵，如下图所示：



其实就是根据一个映射规则，将 one-hot 词向量嵌入，映射成了另一种词向量的嵌入，只不过这种新的词向量的嵌入方式更适用于网络的正确率提高，所以其实 nn.embedding 只是一种随机的映射方式，不过这种映射可以通过神经网络的损失函数进行训练，也许它单独和 word2vec 算法比起来，性能比较差，但是他最终的目的是为了提升分类的准确性，是一种考虑全局的思想，目前使用也比较广泛。

## 2.5 分类神经网络

所谓人工神经网络就是基于模仿生物大脑的结构和功能而构成的一种信息处理系统。粗略地讲，大脑是由大量神经细胞或神经元组成的。每个神经元可看作是一个小的处理单元，这些神经元按某种方式连接起来，形成大脑内部的生理神经网络。这种神经网络中各神经元之间联结的强弱，按外部的激励信号做自适应变化，而每个神经元又随着所接收到的多个接收信号的综合大小而呈现兴奋或抑制状态。大脑处理信息流程如下：

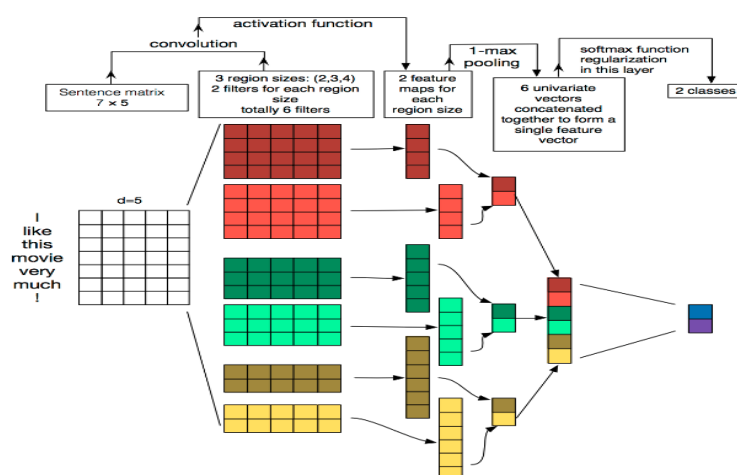


而人工神经网络是根据人的认识过程而开发出的一种算法。把输入与输出之间的未知过程看成是一个“网络”，通过不断地给这个网络输入和相应的输出来“训练”这个网络，网络根据输入和输出不断地调节自己的各节点之间的权值来满足输入和输出。当训练结束后，给定一个输入，网络便会根据自己已调节好的权值计算出一个输出。在本次实验中，我们使用 TextCNN 和 AttTextCNN 作为分类网络模型，下面对这两种方法进行介绍。

### 2.5.1 TextCNN

TextCNN 是一款非常经典的卷积神经网络。CNN 通常被认为是属于 CV 领域，用于计算机视觉方向的工作，但是在 2014 年，Yoon Kim 针对 CNN 的输入层做了一些变形，提出了文本分类模型 TextCNN。

与传统图像的 CNN 网络相比, TextCNN 在网络结构上没有任何变化, 甚至是更加简单了, 它的结构如下所示:



TextCNN 一共分为四个部分, 分别是词嵌入层, 卷积层, 池化层和全连接层。下面将分别介绍每一层的作用及输入举例。

### ① 词嵌入层 (Embedding):

TextCNN 首先将一句话分成多个关键词, 再经过 Embedding 方式将每一个词映射成一个多维词向量输入到网络中, 这样作的好处主要是将天然语言数值化, 方便后续的处理。从这里也能够看出不一样的映射方式对最后的结果是会产生巨大的影响。

图中最左边的 7 乘 5 的句子矩阵, 每行是词向量, 维度=5, 这个可以类比为语句中的原始词结构。

### ②卷积层 (Convolution):

这一层用于提取单词的特征。卷积层的操作类似于数学中的卷积, 但是更加简单, 计算机和我们看到的文字是不一样的, 对于他来说每一句话用一个矩阵表示, 这里的卷积操作是通过卷积核对每个通道的矩阵从左到右, 从上到下进行互相关运算, 就像一个小窗口, 从左上角一步步滑到右下角, 互相关运算的意思就是对应位置相乘再相加, 最后把多个通道的值对应加起来得到一个值。卷积层有多个卷积核, 通过越来越多的卷积, 提取到的文字特征会越来越抽象。嵌入层的向量经过分别为 2,3,4 的一维卷积层, 每个尺寸的卷积层有两个输出通道。

③池化层 (MaxPolling): 将不同长度句子经过 pooling 层之后都能变成定长的表示。通常在连续的卷积层之间会周期性地插入一个池化层。它的作用是逐渐降低数据体的空间尺寸, 这样的话就能减少网络中参数的数量, 使得计算资源耗费变少, 也能有效控制过拟合。从图中可以看出, 不同的长度的句子都变成了二维向量。

④全连接层 (FullConnection): 池化层后面一般接着全连接层, 全连接层将所有的特征矩阵转化为一维特征向量, 用于输出每个类别的概率。

## 2.5.2AttTextCNN

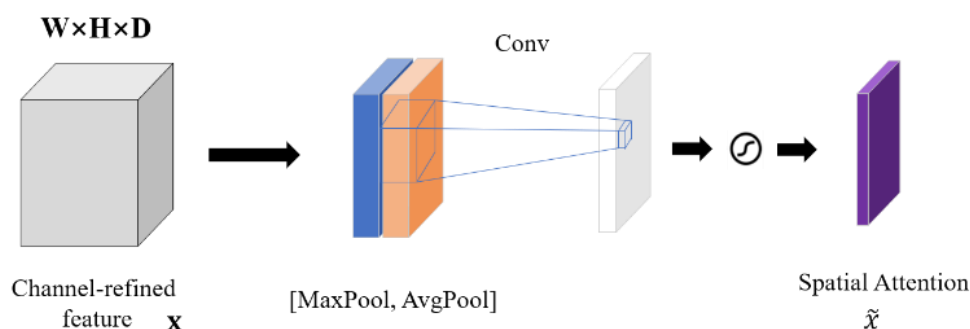
我们小组对现有的 TextCNN 方法进行了改进, 采取的方法是引入 NLP 中比较热门的注意力 Attention 机制, Attention 机制的目的其实就是让网络能够多去关注一些比较重要的信息, 而忽略一些次要的信息。

举个形象的例子, 在我们处理文本情感分析的时候, 如果遇到了下面一句话: “我觉得



这个关东煮很好吃。”那我们判断这个句子是积极的主要依据其实应该是“好吃”这个词，而不是“关东煮”这个词，同样我们也希望网络在自我学习的过程中，能够去注意这些重要的词语。

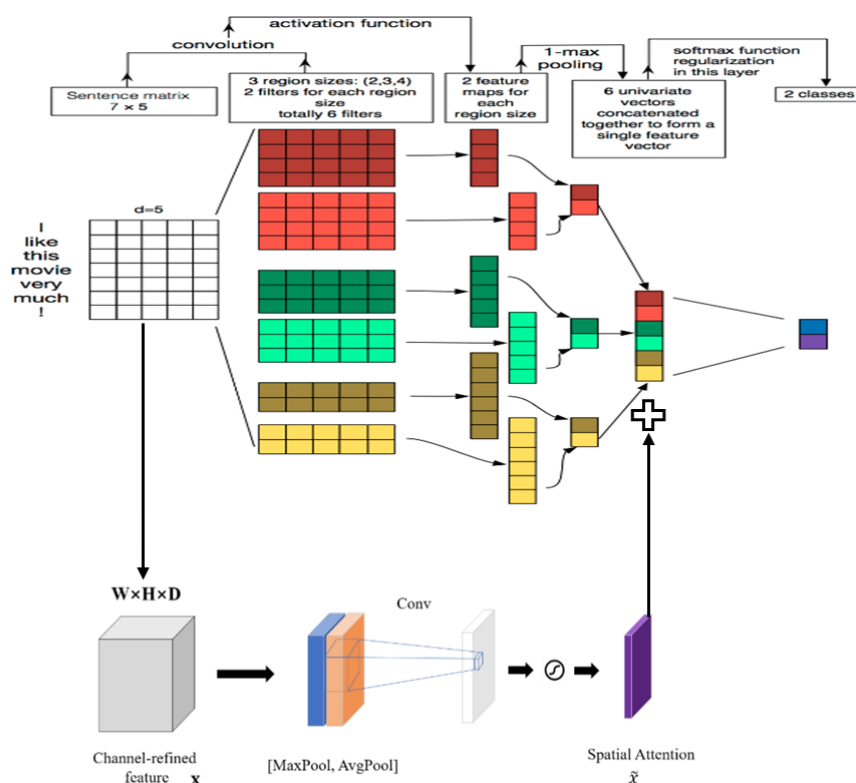
我们选择了使用空间注意力机制 Spatial Attention 模块进行优化，它的结构如下图所示：



它的表达式如下：

$$\tilde{x} = x \cdot \sigma(\text{conv}([\text{Avg Pool}(F), \text{MaxPool}(F)]))$$

上图中通过对输入的矩阵向量，在 channel 维度上进行特征得到融合，分别取每个位置上各通道上的最大值以及平均值，生成了两个 mask，然后通过一个卷积操作，最终合并成了一个 mask，及上图中最后的 Spatial Attention，然后将他和 TextCNN 中的卷积提取的特征向量进行相加，这样就生成了一个带有注意力的网络，我们称它为 AttTextCNN，总体结构如下图所示：



这相当于是有两个通道对网络进行了特征提取，我们觉得这样的好处主要有以下两点：

- ①网络参数量增多，拟合效果更好。增多了网络的参数，可以让函数的拟合效果更好，不过要注意解决过拟合问题。
- ②更关注一些重要的词语信息，注意力有所侧重，可以提高网络的性能。

## 2.5.3 损失函数及优化算法

对于网络的训练目的，最终是需要将损失函数最小化，达到网络的拟合，因此选择损失函数是每个神经网络的必备过程。

而神经网络的训练往往采用梯度下降的方法，去找到损失函数的最小值，如今有很多种优化算法对梯度下降进行设计，下面我们主要介绍本系统中使用到的损失函数和优化算法。

### ① 交叉熵损失函数

我们的系统中对于训练集采用的是交叉熵损失函数作为 loss 函数。交叉熵刻画的是实际输出（概率）与期望输出（概率）的距离，也就是交叉熵的值越小，两个概率分布就越近。

假设概率分布  $p$  为期望输出，概率分布  $q$  为实际输出， $H(p,q)$  为交叉熵，则：

$$H(p, q) = - \sum p(x) \log q(x)$$

举例说明：假设  $N=3$ ，期望输出  $p(1,0,0)$ ，实际输出  $q_1 = (0.5, 0.2, 0.3)$ ， $q_2 = (0.8, 0.1, 0.1)$  那么根据交叉熵公式计算得

$$H(p, q_1) = 0.3$$

$$H(p, q_2) = 0.1$$

显然， $q_2$  与  $p$  更接近，它的交叉熵也更小。这个例子是单个样例的情况，而在实际使用训练过程中，数据往往是组合成为一个向量来使用，所以对用的神经网络的输出应该是矩阵形式。

### ② 自适应运动估计算法（ADAM 算法）

ADAM 是一种可以替代传统随机梯度下降过程的一阶优化算法，它能基于训练数据迭代地更新神经网络权重，伪代码如下：

**Algorithm 1:** Adam, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation.  $g_t^2$  indicates the elementwise square  $g_t \odot g_t$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

**Require:**  $\alpha$ : Stepsize

**Require:**  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates for the moment estimates

**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$

**Require:**  $\theta_0$ : Initial parameter vector

$m_0 \leftarrow 0$  (Initialize 1<sup>st</sup> moment vector)

$v_0 \leftarrow 0$  (Initialize 2<sup>nd</sup> moment vector)

$t \leftarrow 0$  (Initialize timestep)

**while**  $\theta_t$  not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)

**end while**

**return**  $\theta_t$  (Resulting parameters)

如上算法所述，在确定了参数  $\alpha$ 、 $\beta_1$ 、 $\beta_2$  和随机目标函数  $f(\theta)$  之后，我们需要初始化参数向量、一阶矩向量、二阶矩向量和时间步。然后当参数  $\theta$  没有收敛时，循环迭代地更新



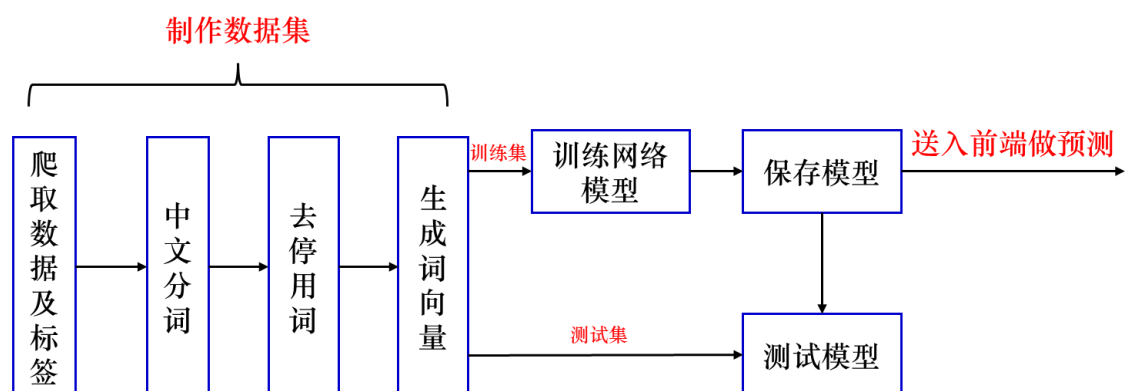
各个部分。即时间步  $t$  加 1、更新目标函数在该时间步上对参数  $\theta$  所求的梯度、更新偏差的一阶矩估计和二阶原始矩估计，再计算偏差修正的一阶矩估计和偏差修正的二阶矩估计，然后再用以上计算出来的值更新模型的参数  $\theta$ 。

## 3.系统总体设计

整个系统可以分为前端和后端两个部分，后端代码负责爬取数据、中文分词、去停用词、生成词向量等操作来制作数据集，同时训练网络模型；前端界面主要负责内容的呈现，以及利用训练好的网络模型进行预测。

### 3.1 后端开发

整个后端的设计思路如下图所示：



主要分为以下几个部分和思路：

#### ① 爬取数据及标签：

利用 python 爬虫爬取美团外卖平台的评论数据，其中将五星评论作为我们的积极数据，而将一星的评论作为我们的消极数据，最终我们共爬取了 8000 条评论（3000 条正面，5000 条负面）作为训练集数据，以及 3000 条（1000 条正面，2000 条负面）作为测试集数据。

#### ② 中文分词：

对收集到的所有数据，进行分词处理，将所有的句子，转换成多个词组组成的序列。

#### ③ 去停用词：

每个句子经过分词后，会有很多的停用词以及标点符号等没有什么实际意义的内容，因此我们需要通过去停用词表将这些部分给去除掉，只留下一些具有实际意义的词语，组成我们最终的序列。

#### ④ 生成词向量模型：

我们得到了每个句子的词语组成之后，需要将这些词语转换成计算机能够处理的矩阵数据，即词向量，使得后续可以进行处理。我们把每个词语都用一个行向量来表示，这样每个句子就可以用一个由多个行向量组成的矩阵来表示。为了保证每个句子的矩阵大小一样，我们最终选定一个句子由 30 个单词组成，如果不足 30 个，那就填充全 0 的行向量；而每个单词我们可以设定它由一个 100 维的行向量组成（当然可以设定的更大，设定的越大则参数

会越多，可能效果会越好)。因此我们以训练集为例，我们训练集的 8000 条评论就可以转换为一个  $8000 \times 30 \times 100$  的矩阵。

#### ⑤ 训练以及保存模型：

我们使用训练集的 8000 条数据对分类网络进行训练，选用交叉熵函数作为 loss 函数，选用 Adam 优化算法来进行梯度的更新，使用 CPU 进行训练（电脑没有 GPU）500 轮，最终将得到的模型全部保存，然后送入我们的前端界面。

#### ⑥ 测试模型：

我们使用测试集中的 3000 条数据对训练好的网络模型进行正确率的测试，这里需要注意的是我们的测试集的词向量生成，使用的是训练集训练好的词向量生成模型，如果有些词语在模型的词语库中并没有出现，我们就把它当作一个 0 向量来处理。

以上是整个后端开发的思路设计情况。

## 3.2 前端界面

我们的设计思路是，让前端和用户进行交互，主要完成以下几个功能。

#### ① 用户选择不同的模型并实现界面显示：

我们计划使用多种不同的分词、词向量生成以及分类网络方案来实现多种模型，因此在前端界面中，希望用户可以去根据自己的想法选择不同的模型；

同时，我们想让选择的模型能够被显示在界面上，并且将这些模型在训练集中的表现告诉我们的用户，给用户一个直观的感受。

#### ② 实时预测输入文本的情感检测：

除了体现模型在训练集上的效果之外，我们还希望能够做成一个实时检测的文本情感分析系统，即我们可以输入想要检测的一句话，然后由系统模型做预测，将对这句话的预测结果输出出来。

整体的前端界面设计思路就是这样，主要实现的方法是使用后端的处理代码，将输入的文本进行分词处理、去停用词、词向量生成，然后跟我们选择的模型，从库中导入之前训练好的模型进行结果预测并输出。

## 4.模块详细设计

我们将在这个部分详细讲解上一部分中提到的各个模块设计的实现，以及具体的代码，来向大家展示我们的成果和工作。

### 4.1 数据爬取

#### 4.1.1 任务分析

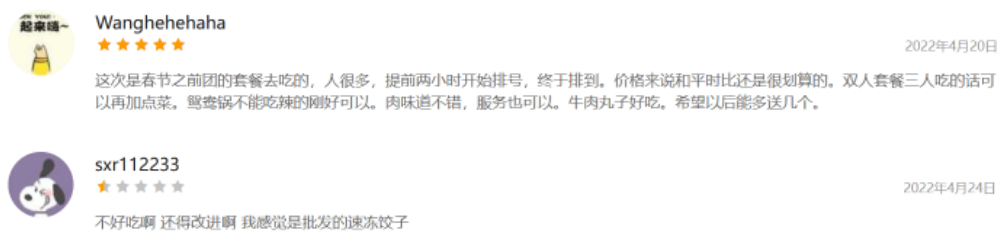
本次实验任务中需要大量的正面（positive）和负面（negative）的数据对神经网络进行训练，我们选择从网页上抓取数据的方式构成我们的数据集。然而，逐句手动判断情感的方式大大增加了我们的工作量，所以我们可以收集带有评分的评论，通过评分对语句的正负面

做出判断。

根据以上构想，我们选择了几个可以尝试的网站，分别是美团外卖评论，淘宝评论和豆瓣影评。但是综合获取数据的难易程度和我们后期训练的可实现性，最终确定美团网作为数据来源。如下图所示：



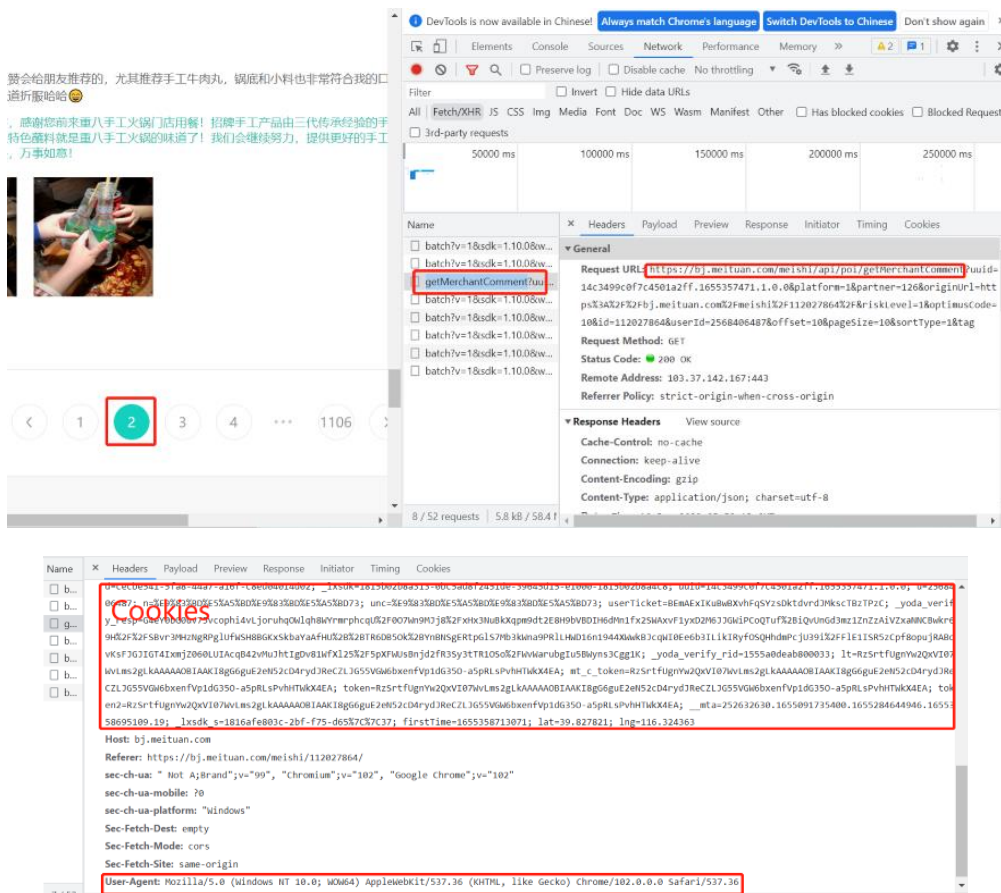
在美食类别中有多个店家，每个店家下有个评论数，完全可以满足我们对训练集量的需求，在下图中第一个是五星级评价，所以我们将它归到 positive 一类，第二个评分较低，最终归结为 negative。



## 4.1.2 具体实现

确定了抓取美团网的数据后，为了满足数据量的需求，即训练集数据 8000 条，测试集数据 3000 条，我们需要爬取多个店铺的评论作为我们的数据来源。我们以其中一家店铺为例，具体介绍我们的实现流程。

首先是获取网页的请求头和防盗链等信息，打开控制台后点击评论下方的翻页按钮，相当于做了一次刷新页面的操作。刷新页面后，在 network 下找到关于第二页评论的包信息，包信息中有网页的 URL 等参数，在本次实验中我们需要这一界面的 URL, params, refer, Cookie, 和 User-Agent 信息，如下所示。



其中，Cookies 包含了我们的登录信息，这是在我们通过账号登陆之后，短期内可以通过这个 Cookies 实现代码的自动登录。有了这些信息后，数据请求部分代码如下：

```
time.sleep(2)
url = 'https://bj.meituan.com/meishi/api/poi/getMerchantComment'
data = {
    'uid': '517381a5b8f54149842c.1655123448.1.0.0',
    'platform': '1',
    'partner': '126',
    'originUrl': 'https://bj.meituan.com/meishi/276310/',
    'riskLevel': '1',
    'optimusCode': '10',
    'id': '276310',
    'userId': '2568406487',
    'offset': '0',
    'pageSize': '10',
    'sortType': '1',
    'tag': '',
}
headers = {
    'Referer': 'https://bj.meituan.com/meishi/42715139/',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36',
    'Cookie': '_lxsdk_cuid=1815b02b8a313-0bc5ad8f2451de-39645d15-e1000-1815b02b8a4c8; mtcdn=K; _hc.v=528c413b-65'
}
response = requests.get(url = url, params = data, headers = headers)#发送请求
```

然后我们就可以开始爬取对应部分的数据。以下是获取到的 json 格式数据，可以看到 response 返回的 json 格式并不全是我们所需要的，所以我们最终要从其中提取有用的信息，在这里我们采用正则表达式法，对获取到的全部信息进行提取。

```
[{'userName': '晨光彩色', 'userUrl': '', 'avgPrice': 99, 'comment': '味道不错，价格实惠，会经常去，那会排队就是麻烦', 'merchantComment': '亲爱的顾客，感谢您前来重八手工火锅店用餐！招牌手工产品由三代传承经验的手工师傅把关制作，“顺顺手工，口口真传”。搭配独有的手工特色蘸料就是重八手工火锅的味道了！我们会继续努力，提供更好的手工产品和优质的服务，同时也期待您的再次光临，祝您生活愉快，万事如意！', 'picUrls': [], 'commentTime': '1652108494676', 'replyCnt': 1, 'zanCnt': 0, 'readCnt': 282, 'highlight': '', 'userLevel': 0, 'userId': '65341871', 'uType': 2, 'star': 45, 'quality': False, 'alreadyZzz': False, 'reviewId': '4373136759', 'menu': None, 'did': 0, 'dealEndTime': None, 'anonymous': False, 'poiId': None, 'poiIdL': {'buffer': {'type': 'Buffer', 'data': [0, 0, 0, 0, 0, 0, 0, 0]}, 'offset': 0}}, {'userName': '顺功能霸', 'userUrl': '', 'avgPrice': 99, 'comment': '味道好，最喜欢他家麻酱，服务特别好，价格也便宜，牛肉丸也是一绝，如果我吃火锅，最爱没有之一', 'merchantComment': '亲爱的顾客，感谢您前来重八手工火锅店用餐！招牌手工产品由三代传承经验的手工师傅把关制作，“顺顺手工，口口真传”。搭配独有的手工特色蘸料就是重八手工火锅的味道了！我们会继续努力，提供更好的手工产品和优质的服务，同时也期待您的再次光临，祝您生活愉快，万事如意！', 'picUrls': [], 'commentTime': '1650896429576', 'replyCnt': 1, 'zanCnt': 0, 'readCnt': 691, 'highlight': '', 'userLevel': 0, 'userId': '120079447', 'uType': 2, 'star': 50, 'quality': False, 'alreadyZzz': False, 'reviewId': '4325264879', 'menu': '100元代金券1张，可叠加', 'did': 28653354, 'dealEndTime': '1658332799', 'anonymous': False, 'poiId': None, 'poiIdL': {'buffer': {'type': 'Buffer', 'data': [0, 0, 0, 0, 0, 0, 0, 0]}, 'offset': 0}}, {'userName': '采
```

数据提取部分代码如下，score 对应 json 数据中 star 标签，comment 对应 json 数据中 comment 标签。代码中的 for 循环遍历 searchResult 中的所有数据。

```
searchResult = response.json()['data']['comments']
print(searchResult)
for index in searchResult:
    dit = {
        'score': index['star'],
        'comment': index['comment'],
    }
    csv_writer.writerow(dit)
    print(dit)
```

最终，得到结果如下，整个结果包含了评论和得分两个方面，和我们之前的设计思路相符合。

```
{'score': 45, 'comment': '味道不错，价格实惠，会经常去，那会排队就是麻烦'}
{'score': 50, 'comment': '味道好，最喜欢他家麻酱，服务特别好，价格也便宜，牛肉丸也是一绝，如果我吃火锅，最爱没有之一'}
```

我们刚才介绍的是获取网页中一页评论的办法，但是我们得实现多条评论的获取，就得去不同页数的评论网页上去爬取数据。

由于每一家门店评论翻页网站的 URL 是不变的，所以我们没办法通过变动 URL 的方式切换界面。但是我们观察到，美团网站中，每个店家下对应十条评论，每个页面 param 选项中 offset 的值不同，如下所示，offset 的变化代表了页数的变化情况。

Query String Parameter	Query String Parameter	Query String Parameter
uuid: 14c3499c0f7c	uuid: 14c3499c0f7c4	uuid: 14c3499c0f7c
platform: 1	platform: 1	platform: 1
partner: 126	partner: 126	partner: 126
originUrl: https://	originUrl: https://b	originUrl: https://
riskLevel: 1	riskLevel: 1	riskLevel: 1
optimusCode: 10	optimusCode: 10	optimusCode: 10
id: 112027864	id: 112027864	id: 112027864
userId: 2568406487	userId: 2568406487	userId: 2568406487
offset: 20	offset: 30	offset: 40
pageSize: 10	pageSize: 10	pageSize: 10
sortType: 1	sortType: 1	sortType: 1
tag:	tag:	tag:

因此，我们可以用 for 循环并将 offset 改为变量模式达到多页评论获取的目标。如下所示：



```

for page in range(0,2000,10):
    time.sleep(100)
    url ='https://bj.meituan.com/meishi/api/poi/getMerchantComment'
    data = {
        'uuid': '1f54b35c4eb24f48ba38.1655363394.1.0.0',
        'platform': '1',
        'partner': '126',
        'originUrl': 'https://bj.meituan.com/meishi/112027864/',
        'riskLevel': '1',
        'optimusCode': '10',
        'id': '112027864',
        'userId': '2568406487',
        'offset': page,
        'pageSize': '10',
        'sortType': '1',
        'tag': '',
    }
    headers = {
        'Referer': 'https://bj.meituan.com/meishi/112027864/',
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0
        'Cookie': '_lxsdk_cuid=1815b02b8a313-0bc5ad8f2451de-39645d15-e1000-1815b02b8a4c8; mtcidn=K; _hc.v=528c413b-65
    }

```

红框的部分标出了我们的 offset 值，是由 for 循环中的 page 值所改变的，这样一来，我们每次抓取的评论页数就不一样了，可以让抓取的评论数量变得很多。

到目前为止，我们已经爬取到了所有的评论数据，下面需要进行数据的筛选。因为有些评论的星级是三星或者四星，一般来说这样的评论的情感色彩很有可能是积极和消极掺杂，不利于我们的网络训练和预测。因此我们需要将打分为 45-50（满分为 50）的评论放入 positive 文件中，打分 0-10 的评论放入 negtive 文件，而中间打分的评论给删除掉。然后将他们存入.csv 文件中，最后直接将.csv 文件修改后缀可就可转换为 txt 文本，得到我们最终的数据集。最后的训练集如下所示：

```

train_data.txt
1 很快，好吃，味道足，量大
2 没有送水没有送水没有送水
3 非常快，态度好。
4 方便，快捷，味道可口，快递给力
5 菜味道很棒！送餐很及时！
6 今天师傅是不是手抖了，微辣格外辣！
7 "送餐快,态度也特别好,辛苦啦谢谢"
8 超级快就送到了，这么冷的天气骑士们辛苦了。谢谢你们。麻辣香锅依然很好吃。
9 经过上次晚了2小时，这次超级快，20分钟就送到了.....
10 最后五分钟订的，卖家特别好接单了，谢谢。
11 量大，好吃，每次点的都够吃两次
12 挺辣的，吃着还可以吧
13 味道好，送餐快，分量足
14 量足，好吃，送餐也快
15 特别好吃，量特大，而且送餐特别快，特别特别棒
16 口感好的很，速度快！
17 相当好吃的香锅，分量够足，味道也没的说。
18 好吃！速度！包装也有品质，不出家门就能吃到餐厅的味道！

```

```

train_label.txt
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
13 1
14 1
15 1
16 1
17 1
18 1

```

左上图我们截取了多个爬取的评论情况，而右上图则代表了他们的标签，其中 1 代表积极，而 0 代表消极。

### 4.1.3 改进措施

虽然 for 循环我们设置的是 2000 条评论一起获取，但是由于美团的防爬机制非常的强，每次爬取 200 条之后就会强制退出系统登录，然后通过验证码的方式才能登入账号，非常的麻烦，所以我们后来采取延时等待的方式。当系统检测到我们频繁抓取的时候，我们可以让代码自动休息一段时间，然后再次进行爬取的尝试，这可以有效地规避一些反爬机制。

代码如下所示：

```
for page in range(0,2000,10):
    time.sleep(100)
    url = 'https://bj.meituan.com/meishi/api/poi/getMerchantComment'
```

每次爬取的时候，我们通过红框中的办法，让代码休息 100s，这样就可以规避一些反爬机制的搜捕。但如果要完全规避这种反爬机制，那会大大影响我们的爬取数据效率，因此后续还需要在效率和有效性方面做一些权衡。

## 4.2 中文分词

有了所有的评论数据之后，我们需要将评论进行中文分词以及数据清理工作，即将句子划分成多个词语，并去除一些标点杂乱的东西。

我们在整个的方案设计中，选用了三种中文分词的方式，包括 jieba, pkuseg 和 SnowNLP，三种方案的原理比较以及各自的特点我们在前面的相关技术部分已经讲解完成，下面我们讲一下具体的实现方法。

### 4.2.1jieba 分词

下图是我们 jieba 分词函数的代码实现，我们首先需要设定一个停用词表，方便我们后续的停用词的去除工作。

```
def jieba_split_sentence(input_file, output_file):
    # 把停用词做成列表
    stopwords = [line.strip() for line in open('stop_words.txt', 'r', encoding='utf-8').readlines()]

    fin = open(input_file, 'r', encoding='utf-8') # 以读的方式打开文件
    fout = open(output_file, 'w', encoding='utf-8') # 以写的方式打开文件

    for eachline in fin:
        line = eachline.strip() # 去除每行首尾可能出现的空格
        line1 = re.sub(r"[0-9\s+\.\!\\/_,%^*()?;,:-【】+\\\"']+|[\+! , ; : . ? , ~@#¥%_&*()]+", "", line)
        wordlist = list(jieba.cut(line1)) # 用结巴分词，对每行内容进行分词
        outstr = ''
        for word in wordlist:
            if word not in stopwords: # 判断是否在停用词表中
                outstr += word
                outstr += ' '
        fout.write(outstr.strip() + '\n') # 将分词好的结果写入到输出文件
    fin.close()
    fout.close()
```

然后，上图中红框框出了代码的核心部分，即通过一个 for 循环，去对每一条评论进行处理。

首先我们通过正则化 re 库，去除一些句子中的标点符号；

然后我们通过 jieba.cut 这一个函数操作，将所有的句子进行分词；

最后我们需要将分词后的结果，进行再一次的筛选，根据我们的停用词表，去除掉一些连接词，让整个句子的数据量进一步简洁化，方便后续处理，我们的停用词表部分内容如下所示：

≡ stop_words.txt	≡ stop_words.txt	≡ stop_words.txt	≡ stop_words.txt
1 啊	11 吧	31 不成	21 鄙人
2 阿	12 吧哒	32 不单	22 彼
3 哎	13 把	33 不但	23 彼此
4 哎呀	14 罢了	34 不独	24 边
5 哎哟	15 被	35 不管	25 别
6 唉	16 本	36 不光	26 别的
7 俺	17 本着	37 不过	27 别说
8 俺们	18 比	38 不仅	28 并
9 按	19 比方	39 不拘	29 并且
10 按照	20 比如	40 不论	30 不比

整个停用词表共有 1598 个停用词，很多的连接词都被我们去除了。

去除了停用词之后，我们将最终的结果重新进行存储，存储到一个新的文本当中，最后的部分结果如下所示：

```
≡ jieba_cut_train_data.txt
1 很快 好吃 味道 足量
2 送水 送水 送水
3 态度
4 方便快捷 味道 可口 递给 力
5 菜 味道 很棒 送餐
6 师傅 手抖 微辣 辣
7 送餐 态度 特别 辛苦 谢谢
8 超级 送到 冷 天气 骑士 辛苦 谢谢你们 麻辣 香锅 依然 好吃
9 上次 晚 小时 超级 分钟 送到
```

## 4.2.2pkuseg 分词

Pkuseg 分词是北京大学研发的一款中文分词包。整个代码的实现过程和 jieba 分词类似，我们也是写了另外一个函数方便调用，如下所示：

```
seg = pkuseg.pkuseg()
for eachline in fin:
    line = eachline.strip() # 去除每行首尾可能出现的空格
    line1 = re.sub(r"[0-9\s+\.\!\\/_,$%^*()~;:;-【】+\"'\"]+|[+—! , ;:。? 、~@#¥%.....&*()]+", "", line)
    wordlist = list(seg.cut(line1)) # 用北大分词，对每行内容进行分词
```

主要的区别在于调包的情况，我们看一下最终的分词效果和 jieba 有什么不一样：

```
≡ pkuseg_cut_train_data.txt
1 很快 好吃 味道 足量
2 送水 送水 送水
3 态度
4 快捷 味道 可口 快递 给力
5 菜 味道 棒 送餐
6 师傅 手抖 微辣 辣
7 送餐 态度 特别 辛苦 谢谢
8 超级 送 冷 天气 骑士们 辛苦 谢谢 麻辣 香锅 依然 好吃
9 上次 晚 小时 超级 分钟 送到
10 最后五分钟 订 卖家 特别 接单 谢谢
```

我们发现，pkuseg 的分词结果总体来说比 jieba 分词速度更慢，而结果差别并不大，不过我们人为的来看可能不能看出太多的差别，我们后续会比较几种分词的正确率情况。



## 4.2.3 SnowNLP 分词

SnowNLP 对于中文文本的处理函数其实很多，不过这里我们只是利用了分词包进行分词的使用，代码的使用方法如下：

```
line1 = re.sub(r"[0-9\.\!\@\#\$%^&*()?;,:-【】+\\"'+|! , ; : . ? , ~ @ # % ^ & * ( ) ]+", "", line)
wordlist = SnowNLP(line1) # 用snownlp分词，对每行内容进行分词
```

最终的分词结果如下所示：

```
≡ snownlp_cut_train_data.txt
1 很快 好吃 味道 足 量
2 送 水 送 水 送 水
3 态度
4 快捷 味道 可口 快递 给力
5 菜 味道 很棒 送餐
6 师傅 手 抖 微 辣 辣
7 送 餐快 态度 特别 辛苦 谢谢
8 超级快 送 冷 天气 骑士 辛苦 谢谢 麻 辣 香 锅 依然 好吃
9 上次 晚 小时 超级快 分钟 送
10 五 分钟 订 卖家 特别 接 谢谢
```

它的分词速度介于 pkuseg 和 jieba 之间，正确率的话在我们的任务中没有太大的差别。

## 4.3 生成词向量

我们在前文介绍过，生成词向量的方法大体可以分成两类：一类是单独对语料库进行训练，对词语生成词向量的模型方法，常见的有 word2vec, globe 等方法；另外一类是对生成的词向量和后面的分类网络使用一个系统，即使用总体的一个损失函数进行训练拟合，达到整个系统的准确率最高的生成词向量方法。

下面我们对此系统中使用到的方法的具体实现做讲解。

### 4.3.1 Word2Vec

Word2Vec 是一个非常热门的单独生成词向量的方法，它的主要思想是每次迭代，都会选定一个中心词，然后去预测中心词附近的其他词语的概率分布情况。

在 python 中我们可以直接调用 word2vec 的函数库，来实现词向量的生成：

```
#模型生成
model1 = Word2Vec(LineSentence('cut_train_data.txt'),vector_size=100, min_count=10,sg=1)
#模型保存
model1.save('snownlp+word2vec.model')
model1.wv.save_word2vec_format('word2vec.txt',binary=False)
#循环遍历生成序列
model1 = gensim.models.Word2Vec.load('snownlp+word2vec.model')
```

我们将分词好的数据文本送入 word2vec 中进行处理，其中需要设定两个比较重要的超参数：

#### ① Vector\_size:

即每个单词用一个多少维的行向量来表示，我们这里设定为 100，因为数据不算特别多，同时为了兼顾我们的训练拟合速度，这里选择了 100。

## ② Min\_count:

这里是设定最小词频。即如果在我们的训练集文本中，出现词语数量比较少的话，我们就没有必要给他一个单独的向量，而是可以直接当作全 0 向量来处理，这是为了节省我们的空间和参数，我们在这里设定这个值为 10。

然后我们对模型进行保存，我们可以看一下具体的生成情况，如下所示：

```
= word2vec.txt
1 811 100
2 送 0.113900624 0.1619148 -0.111846164 -0.14781663 -0.22152585 -0.3686189 0.12106637 0.13940366 -0.12431784 -0.17901227 -0.018517368 0.04795216 0.085
3 吃 -0.07521554 0.113403484 -0.22651885 -0.082930624 0.15142685 -0.40746823 -0.09588846 0.3251217 0.03322461 -0.23743795 -0.14524955 -0.2258424 -0.01
4 太 0.03128034 0.20143393 -0.20556134 -0.014864382 -0.017652715 -0.23102957 -0.045757543 0.2919791 -0.012125722 -0.21633656 -0.11624102 0.006227037 0
5 餐 -0.089556835 0.23271333 -0.1887452 -0.15511206 -0.17782028 -0.33277446 0.053733304 0.065247655 -0.08415929 -0.18011026 -0.020415336 0.12781636 0.1
6 味道 -0.02656524 0.09341144 -0.2542154 -0.15170704 -0.030863423 -0.26668924 -0.13256514 0.14959694 0.0002263274 -0.21279314 -0.08069189 0.048769254 0
7 好吃 -0.033327695 0.04903499 -0.2998566 -0.19868596 -0.055228535 -0.26821575 -0.17906044 0.10430404 0.04034261 -0.22633971 -0.05605864 0.08154842 -0
8 不错 -0.005483197 0.046422113 -0.30141717 -0.18062498 -0.022110343 -0.26614964 -0.16416286 0.12612212 0.032149322 -0.22223558 0.004890685 0.088660311
9 小时 0.22907382 0.25402254 0.014743476 -0.059437 -0.117056005 -0.42883375 0.25693908 0.18386473 -0.17402571 -0.20306578 -0.09856554 -0.13823415 0.06
10 点 0.045397554 0.06420493 -0.11682444 -0.1286491 0.0008186583 -0.33035102 0.042855285 0.13271073 -0.10915187 -0.17526115 -0.10125463 -0.13457547 -0.0
11 菜 -0.07348332 0.045108765 -0.3046603 -0.09779251 0.04458007 -0.27947542 -0.17735562 0.1867001 -0.06101597 -0.17330028 -0.1390849 -0.07044032 -0.059
12 说 0.12844953 0.006793112 -0.05003302 -0.2368633 0.019124554 -0.45969993 0.18032032 0.19366737 -0.21583594 -0.1698665 -0.081430726 -0.2106592 -0.007
13 慢 0.1744914 0.31183222 -0.14094643 -0.027812883 -0.13628924 -0.3469864 0.13387494 0.24419972 -0.022451999 -0.25994608 -0.009595162 -0.0019131677 0.0
14 速度 0.081433535 0.1427895 -0.22037354 -0.20459108 -0.11479427 -0.28801396 -0.010507623 0.07163039 0.055003844 -0.21078563 0.001747498 0.14195994 0.0
15 两 0.026280891 0.16976361 -0.18267168 -0.09379831 0.011315137 -0.36279288 0.07341429 0.1581139 -0.2019208 -0.1441614 -0.16352113 -0.17585316 -0.0524
16 次 0.24323586 0.17454873 -0.12494281 -0.21988118 0.22551484 -0.54329956 0.17563184 0.308728 0.057695772 -0.2626616 0.0048354045 -0.36985728 -0.06665
17 差 0.11344425 0.21983811 -0.09704603 -0.10055715 -0.04541471 -0.5372918 -0.009437094 0.24696322 -0.121779695 -0.117335156 -0.08384454 -0.1235016 -0
18 饭 -0.040957503 0.057673585 -0.21037546 -0.09184768 0.030056408 -0.2897126 -0.05276919 0.19641994 -0.14545365 -0.10290356 -0.16037701 -0.1498002 -0.0
19 粥 0.026389 0.16295432 -0.14912637 -0.07361279 0.097959645 -0.48816726 0.1739328 0.39318228 -0.21013175 -0.23107497 -0.06699293 -0.2861147 0.0825386
20 外卖 0.25783008 0.10828139 -0.28148246 -0.3245396 0.21717559 -0.5589173 0.20264545 0.23564464 0.12539712 -0.31480318 0.083846696 0.005369343 -0.01861
21 少 -0.061702114 0.07261862 -0.2564523 -0.056249663 0.052840818 -0.27649572 -0.12221032 0.2543739 -0.077254705 -0.18263346 -0.18146485 -0.13296793 -0
22 难 -0.032506283 0.007992666 -0.34758824 -0.022080125 0.1488408 0.34971163 -0.0096143 0.36805743 0.041393522 -0.27508184 -0.14741701 -0.25629592 0.0
23 时间 0.17852314 0.25591633 -0.07406905 -0.117686436 -0.2023733 -0.39827123 0.10259491 0.1430152 -0.09190852 -0.17224857 -0.056520376 0.02074982 0.134
```

我们发现，一共对 811 个词语，生成了具体的数值向量，即我们现在对这 811 个词语都是用他们对应的 100 维的行向量进行了表示；现在我们可以把每一句话，根据它的单词的行向量，组成一个矩阵；这里需要注意的是，如果有些词语不在我们这个生成的词语向量库中，我们就人为将它设定为全 0 向量即可。

我们下面要做的事情就是，将训练集和测试集中的每句话都转换成一个对应的行向量，具体的代码如下所示，我们以训练集的处理为例：

```
#训练集数据
data = [line.strip() for line in open('cut_train_data.txt', 'r', encoding='utf-8').readlines()]
for i,line in enumerate(data):
    words = np.array(line.split())
    for index,word in enumerate(words):
        try:
            datas = torch.unsqueeze(torch.tensor(model1.wv['%s'%word]),dim=0)
        except:
            datas = torch.zeros([1,100],dtype=torch.float)
        if index==0:
            raw_datas = datas
        else:
            raw_datas = torch.cat((raw_datas,datas),dim=0)
        if index>=29:
            break
    if raw_datas.shape[0]<30:
        raw_datas = torch.cat((raw_datas,torch.zeros([30-raw_datas.shape[0],100],dtype=torch.float)))
    if i==0:
        data_raw = torch.unsqueeze(raw_datas,dim=0)
    else:
        data_raw = torch.cat((data_raw,torch.unsqueeze(raw_datas,dim=0)),dim=0)

labels = [line.strip() for line in open('train_label.txt', 'r', encoding='utf-8').readlines()]
for i,line in enumerate(labels):
    if i==0:
        label_raw = torch.unsqueeze(torch.tensor(int(line)),dim=0)
    else:
        label_raw = torch.cat((label_raw,torch.unsqueeze(torch.tensor(int(line)),dim=0)),dim=0)
data_train=data_raw
label_train=label_raw
print(data_train.shape)
print(label_train.shape)
train_data = MyDataset(data_train, label_train)
train_loader = DataLoader(dataset=train_data, batch_size=200, shuffle=True)
```

这里我们主要是使用了两个 for 循环，第一个 for 循环是对训练集中的每一句话进行循环；第二个 for 循环是对每句话中的每个单词进行循环，我们先把每个单词通过刚才训练好的 word2vec 模型进行生成行向量，然后在第二个 for 循环内，将一句话中的每个单词的行

向量进行拼接；然后再在第一个 for 循环内将训练集中的各个句子的矩阵进行拼接，最终得到一个 8000\*30\*100 的向量。其中这个 30 是指每个句子由 30 个单词组成，如上图红框所示，如果不足 30 个单词，我们直接给他添加全零的行向量。

## 4.3.2 Embedding

我们也可以将词向量的生成模型，和后续的分类神经网络放到一起进行训练拟合。比较常见的实现方法是使用 nn.embedding 类，这个类的使用方法如下所示：

```
self.W = nn.Embedding(100, 100)
```

前面一个 100 指的是我们一共想对 100 个词语进行词向量的生成，而后面的 100 指的是，我们想让每个单词的词向量都是 100 维的。定义好了类之后，我们使用的时候，需要输入的是一个句子中词语的序号，比如[1, 3, 19, 40]，这就代表我们想要把这个由四个单词组成的句子，转换成一个向量矩阵。具体如下：

```
embedding_X = self.W(X)
```

其中 X 是我们处理过的训练集或者测试集数据，具体的处理方式我们现在介绍。

由于我们设定了词库中只出现 100 个单词，因此我们需要对训练集中目前分词和去停用词后出现的词语频率进行统计，统计结果如下所示：

```
PS D:\网络内容安全大作业> cd 'd:\网络内容安全大作业'; & 'D:\anaconda\python.exe' 'c:\Users\张恒瑞\.vscode\extensions\ms-python.python-2022.8.0\pythonFiles\lib\python\debugpy\launcher' '57238' '--' 'd:\网络内容安全大作业\onehot_main.py'
{'很快': 336, '好吃': 1188, '味道': 1080, '足量': 1, '送水': 3, '态度': 236, '方便快捷': 3, '可口': 7, '递给': 4, '力': 32, '菜': 373, '很棒': 51, '送餐': 1003, '师傅': 139, '手抖': 1, '微辣': 11, '辣': 104, '特别': 257, '辛苦': 197, '谢谢': 103, '超级': 190, '送到': 548, '冷': 41, '天气': 29, '骑士': 120, '谢谢你们': 2, '麻辣': 17, '香锅': 17, '依然': 8, '上次': 36, '晚': 126, '小时': 781, '分钟': 196, '五分钟': 5, '订': 139, '卖家': 24, '接单': 23, '量': 188, '每次': 71, '点': 509, '够吃': 3, '两次': 38, '挺辣': 2, '吃': 990, '送': 922, '餐快': 4, '分量': 57, '足': 66, '量足': 10, '特大': 4, '棒': 48, '口感': 44, '速度': 480, '够': 40, '说': 534, '包装': 225, '品质': 17, '家门': 4, '餐厅': 51, '好极': 2, '送货': 88, '这家': 103, '赞': 123, '总点': 1, '多够': 2, '五个': 4, '很香': 13, '美味': 24, '下次': 151, '还会': 28, '光顾': 16, '服务': 203, '不错': 899, '半小时': 78, '不到': 83, '送来': 335, '大雾': 5, '霍': 9, '天': 31, '外卖': 406, '赞赞赞': 6, '正宗': 16, '内容': 10, '又快又好': 10, '好大': 9, '一盆': 2, '份量': 27, '配送': 288, '人员': 67, '里': 193, '感谢': 46, '餐员': 170, '非常感谢': 14,
```

然后我们可以选择其中词频最高的 99 个单词，将他们的序号分别排列为 1-99，这 99 个词语如下所示：

```
['好吃', '味道', '送餐', '吃', '送', '不错', '小时', '送到', '说', '点', '速度', '太', '外卖', '难吃', '菜', '粥', '很快', '送来', '太慢', '时间', '凉', '~', '饼', '配送', '百度', '饭', '慢', '肉', '两个', '特别', '卷饼', '态度', '包装', '小哥', '快递', '卷', '电话', '服务', '东西', '少', '打电话', '感觉', '辛苦', '分钟', '里', '超级', '挺', '量', '差评', '肘子', '真的', '餐员', '差', '一点', '买', '订单', '做', '下次', '米饭', '酒', '师傅', '订', '喜欢', '员', '晚', '第一次', '赞', '实在', '骑士', '不好', '菜品', '商家', '一份', '煎饼', '牛肉', '口味', '鸡肉', '好评', '希望', '送达', '餐', '辣', '谢谢', '这家', '送过来', '套餐', '咖啡', '肥肉', '满意', '订餐', '店里', '好喝', '皮蛋', '送错', '备注', '一个半', '下单', '发票', '送货', '汤']
```

为什么我们不是找词频最高的 100 个单词，而是找了 99 个呢？因为我们在映射的时候，对于其他单词我们不能直接忽略不管，我们将其他的单词统一认为是编号 0，这样就凑齐了词向量空间中的 100 个词语。

我们训练集中的 8000 个句子，最终会被我们先转换为 8000\*30 的矩阵，然后送入 nn.embedding 类中进行词向量的生成，然后最后经过分类网络的处理，和分类网络的损失函数一起处理，更新词向量的生成，最终达到拟合。

## 4.4分类神经网络

经过了前面部分的处理, 我们终于得到了可以供神经网络训练和测试的训练集和测试集样本。我们选用了之前在相关技术中介绍的 TextCNN 和 AttTextCNN 进行分类网络的训练和预测, 下面进行具体讲解。

### 4.4.1TextCNN

TextCNN 是比较经典的 NLP 处理网络, 我们之前已经介绍过, 这里主要讲他的实现。我们一开始按照 TextCNN 的框架, 采用了三种不同的卷积核分别进行提取, 但是由于我们的训练集数量较少, 且受限于 CPU 资源, 发现训练的效果不佳, 不如我们只采用了一种卷积核的情况。

于是, 我们最终只采用了卷积核大小为 (2\*100) 的特征提取办法, 然后对其进行 Maxpool 和全连接, 实现二分类, 具体网络代码如下所示:

```
class TextCNN(nn.Module):
    def __init__(self):
        super(TextCNN, self).__init__()
        output_channel = 5
        self.W = nn.Embedding(100, 100)
        self.conv1 = nn.Sequential(nn.Conv2d(1, output_channel, (2,100)), # inpu_channel, output_channel, 卷积核高和宽 n-gram 和 embedding_size
                                   nn.BatchNorm2d(output_channel),
                                   nn.ReLU(),
                                   nn.MaxPool2d((2,1)))
        self.fc = nn.Linear(70,2)
        self.sf = nn.Softmax(dim=1)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

    def forward(self, X):
        """
        X: [batch_size, sequence_length]
        """
        batch_size = X.shape[0]
        embedding_X = self.W(X)
        embedding_X = embedding_X.unsqueeze(1) # add channel(=1) [batch, channel(=1), sequence_length, embedding_size]
        convded = self.conv1(embedding_X) # [batch_size, output_channel,1,1]
        flatten = convded.view(batch_size, -1) # [batch_size, output_channel*1*1]
        output = self.fc(flatten)
        return output
```

这里我们看到了 nn.embedding 类, 说明这是采用词向量模型和网络模型共同训练的方式完成的, 如果想使用 word2vec 模型, 那我们只需要不经过 nn.embedding 操作, 直接将输入送到 conv 里面进行卷积操作即可。

这里我们对卷积和 BN 的初始参数进行了初始化设计, 对于二维卷积核来说, 我们让他的 weight 满足 kaiming 初始化方法, 而 bias 直接设置为常数 0; 对于 BN 操作, 我们将 weight 设置为 1, bias 设置为 0, 这算是一种现在比较流行的初始化方法。

### 4.4.2AttTextCNN

我们在前面介绍了这个网络, 就是在 TextCNN 中加入了空间注意力机制模块, 让网络的注意力能够有更好的分配, 我们在这里展示一下空间注意力的代码如下所示:

```

class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=3, padding=1, stride=1, dilation=1, groups=1, bias=False, act_func='relu', **kwargs):
        super().__init__()

        self.spatial = nn.Conv2d(2, 1, kernel_size=kernel_size, stride=stride, padding=padding, dilation=dilation, groups=groups, bias=bias)
        self.act = nn.PReLU()
        self.relu = nn.ReLU()

    def forward(self, x):
        x_compress = torch.cat((torch.max(x, dim=1, keepdim=True)[0], torch.mean(x, dim=1, keepdim=True)), dim=1)
        x_out = self.spatial(x_compress)
        scale = torch.sigmoid(x_out)
        return self.relu(x + x * scale)

```

空间注意力机制是为了给每层中的像素点值（这里用了图像处理中的说法，放在 NLP 中，其实相当于是每个单词的行向量中的每一个值）分配一个网络可以进行自我学习的权重，这样他就可以学习出来哪个单词可能会更加的需要注意力。

整个的实现过程就是对空间中的每个点，在 channel 维度上求一次最大值，再求一次平均值，这样我们就可以得到两层 mask，然后将这两层 mask 通过一个卷积操作，将他转换成一个 mask 输出，并把他作为权重和我们的原始数据相乘即可。

### 4.4.3 预测方法

最后，讲完了网络的训练和所有的操作之后，我们需要讲解一下如何去对测试集中的数据或者我们人为输入的数据，利用训练好的模型进行预测。首先，我们需要同训练集的处理一样，根据我们选用的分词方式以及词向量生成方式，对待预测的数据进行处理，将他们生成可以送入网络预测的矩阵，具体过程和训练集的生成方式一样，前面已经介绍过，这里就不再赘述。

然后我们需要将这些矩阵送入网络中去，具体代码如下：

```

model_pre = model.eval()
predict = model_pre(raw_datas).data.max(1, keepdim=True)[1]
if predict[0][0] == 1:
    result='Positive'
else:
    result='Negative'
return result

```

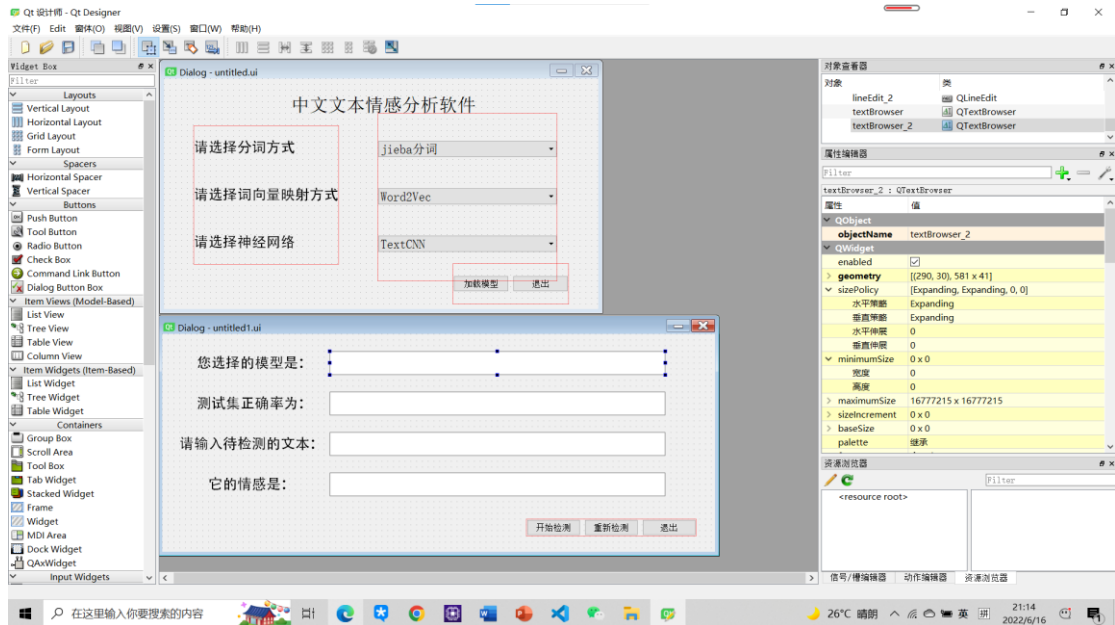
我们需要将网络设置为 eval 模式，即测试模式，然后对数据进行预测。

最终将预测二分类结果中比较大的值的索引，作为最终的标签输出；即如果这个标签为 1 的话，我们认为它是积极的；如果标签为 0 的话，则我们认为它是消极的。

## 4.5 系统界面设计

为了更好地展示我们的模型效果，以及提供人机交互的可操作空间，我们使用了 QT Designer 配合 python，设计了一个简单的人机交互界面。使用 QT designer 绘制界面的情况如下：





一共有两个界面组成，第一个界面负责选择不同的模型方式，而第二个界面负责展示模型的成果以及形成人机交互的情况。

使用 QT designer 生成的窗口文件为 ui 格式，我们需要将 ui 格式的文件先转化为 py 文件，方便我们后续的处理，在 cmd 中 cd 到对应文件所在的地址，然后使用如下面命令即可实现文件格式的转化：

```
d:\网络内容安全大作业\系统界面>pyuic5 -o srs.py untitled.ui_
```

我们就将 Untitled.ui 文件转换为了 srs.py 文件。

然后我们需要重新创建一个 py 文件，然后分别从两个界面生成的 py 文件中导入对应的窗口。我们首先要实现的一个操作是，当点击第一个窗口时的按键时，能够跳转到第二个窗口，具体的实现代码如下所示：

```
class mywindow(QMainWindow, Ui_Dialog):
    def __init__(self):
        super(mywindow, self).__init__()
        self.setupUi(self)
        self.pushButton_start.clicked.connect(self.process)
    def process(self):
        a=self.comboBox_fenci.currentText()
        b=self.comboBox_cixiangliang.currentText()
        c=self.comboBox_net.currentText()
        print(a,b,c)
        self.child_window = secwindow(a,b,c)
        self.child_window.show()
```

在我们的主窗口类的初始化中，我们将按键和process函数进行捆绑连接，如红框所示。

只要我们点击了对应的按键，系统就会立马去执行 process 函数。

而在 process 函数中，我们首先通过 abc 读取此时窗口 1 中对应的方式选择当中的文本，即 currentText，这一步骤的主要目的是为了让我们能够通过用户的不同方式选择，去后台中调用我们训练好的模型进行预测。然后我们定义了一个子窗口，这个子窗口就是需要弹出的第二个窗口，然后将他通过 show 函数给弹出来即可。

而子窗口类中，接收到了刚才传过来的 abc 三个参数，这样就可以根据这三个参数选择具体的模型了。子窗口类的初始化函数如下所示：

```
class secwindow(QWidget, Ui_Dialog1):
    def __init__(self,a,b,c):
        super(secwindow, self).__init__()
        self.setupUi(self)
        self.a=a
        self.b=b
        self.c=c
        if self.a=="jieba分词":
            if self.b=="Word2Vec":
                if self.c=="TextCNN":
                    self.textBrowser.setText('2345/3000')
                    self.textBrowser_2.setText('jieba + word2vec + TextCNN')
                elif self.c=="AttTextCNN":
                    self.textBrowser.setText('2496/3000')
                    self.textBrowser_2.setText('jieba + word2vec + AttTextCNN')
            elif self.b=="embedding":
                if self.c=="TextCNN":
                    self.textBrowser.setText('2395/3000')
                    self.textBrowser_2.setText('jieba + embedding + TextCNN')
                elif self.c=="AttTextCNN":
                    self.textBrowser.setText('2502/3000')
                    self.textBrowser_2.setText('jieba + embedding + AttTextCNN')
        if self.a=="pkuseg分词":
            if self.b=="Word2Vec":
```

我们发现，在子窗口中，我们首先读取了从主窗口中传递过来的用户选择的不同分词方式，词向量选择方式以及分类网络模型，然后通过多个 if 和 else 函数进行判断和选择，最终将用户选择的模型显示在界面上，并且将这个模型的正确率一并显示在界面上。

然后当我们在子窗口的文本框中输入对应的文本时，然后点击子窗口上的对应按钮时，我们会跳转到另一个函数，如下所示：

```
def process(self):
    xxx = self.lineEdit.text()
    if self.a=="jieba分词":
        if self.b=="Word2Vec":
            if self.c=="TextCNN":
                model1 = Word2Vec.load('jieba+word2vec.model')
                model = torch.load('jieba+word2vec+TextCNN1.pt')
                result=predict_jieba(model,model1,xxx)
            elif self.c=="AttTextCNN":
                model1 = Word2Vec.load('jieba+word2vec.model')
                model = torch.load('jieba+word2vec+AttTextCNN1.pt')
                result=predict_jieba(model,model1,xxx)
        elif self.b=="embedding":
            if self.c=="TextCNN":
                model = torch.load('jieba+embedding+TextCNN1.pt')
                result=predict_jieba1(model,xxx)
            elif self.c=="AttTextCNN":
                model = torch.load('jieba+embedding+AttTextCNN1.pt')
                result=predict_jieba1(model,xxx)
```

我们使用 xxx 先读入文本框中输入的 text 信息，然后也是根据用户选择的模型，通过 if

和 else 语句，从后台调取我们训练好的对应模型，然后用此模型对输入的文本做预测，最终将预测结果写入 result 并显示在桌面上。

至此，整个模块的设计介绍完毕。

## 5.数据结果分析

我们在第四部分的模块设计中，一共实现了三种不同的分词方式，两种不同的词向量生成方式，以及两种不同的分类网络，进行组合后，我们就有了 12 种不同的模型。我们对这 12 种模型在网络中分别进行训练，然后用我们的 3000 条测试集评论进行测试，最终输出他们的正确率情况，代码如下：

```
right_count = 0
wrong_count = 0
for batch_x, batch_y in test_loader:
    model = model.eval()
    predict = model(batch_x).data.max(1, keepdim=True)[1]
    print(model(batch_x).data)
    print(model(batch_x).data.max(1, keepdim=True))
    if predict[0][0] == batch_y:
        right_count += 1
    else:
        wrong_count += 1
```

对于预测正确的样本，我们让 right\_count+1，对于错误的样本，我们让 wrong\_count+1，最终我们将  $\frac{right\_count}{right\_count+wrong\_count}$  作为我们模型对于测试集的正确率，最终 12 种模型的正确率如下所示：

序号	分词方式	词向量生成方式	神经网络	正确数量	正确率
1	Jieba	Word2Vec	TextCNN	2345/3000	78.2%
2	Jieba	Word2Vec	AttTextCNN	2496/3000	83.2%
3	Jieba	Embedding	TextCNN	2395/3000	79.8%
4	Jieba	Embedding	AttTextCNN	2502/3000	83.4%
5	Pkuseg	Word2Vec	TextCNN	2302/3000	76.7%
6	Pkuseg	Word2Vec	AttTextCNN	2423/3000	80.8%
7	Pkuseg	Embedding	TextCNN	2314/3000	77.1%
8	Pkuseg	Embedding	AttTextCNN	2444/3000	81.5%
9	SnowNLP	Word2Vec	TextCNN	2299/3000	76.6%
10	SnowNLP	Word2Vec	AttTextCNN	2404/3000	80.1%
11	SnowNLP	Embedding	TextCNN	2393/3000	80.0%
12	SnowNLP	Embedding	AttTextCNN	2504/3000	83.4%

- 通过上述 12 种方式的正确率分析，我们可以得到如下的几条结论：
- ① 以上方法的正确率基本都是在 80%左右，而我们整个网络的训练轮数在 500 轮，速度也不算很慢，基于此，这个预测的结果我觉得是一个比较令人满意的结果。
  - ② 其中，正确率最高的方法组合是“jieba+embedding+AttTextCNN”以及“SnowNLP+Embedding+ AttTextCNN”。



- ③ 通过控制变量的方式，我们大概可以知道在此次任务种，分词方式选择 SnowNLP 的正确率是最高的；词向量生成方式上选择 Embedding 的方式也要略好于 Word2Vec，可能是由于它和网络一起训练拟合的原因；而加入了 Attention 机制的 AttTextCNN 网络的性能明显要优于 TextCNN。

## 6.系统使用说明

我们在这个部分会详细介绍我们的系统的使用方法，以及具体的操作流程实例。

### 6.1 界面外观介绍

我们的界面由一个主界面和一个子界面组成。

#### 6.1.2 主界面介绍

主界面的作用主要是实现用户的模型选择。整个主界面的设计比较清晰简洁，提供了一个供用户可以选择不同模型的途径：



在红框标出的范围中我们可以选择不同的模型，然后加载模型按键，可以让我们跳转到子窗口中去。

#### 6.1.3 子界面介绍

子界面主要的作用是显示用户的方案选择情况，并且提供一个可以实时进行文本预测的人机交互系统，具体界面如下所示：



子界面中，用红框框出的部分是弹窗开始便显示的地方，它是根据主界面中用户的选择而显示的情况，可以看到它能够清晰的显示用户选择的模型，以及这个模型在我们之前测试集当中的正确率情况。

而蓝框框出的部分则是待用户输入的检测文本区域，可以提供给我们实时的对一些文本进行情感的预测。

## 6.2 使用方法示例

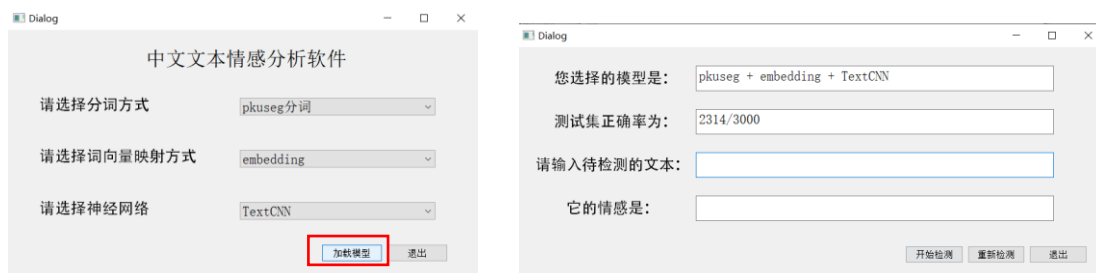
### 6.2.1 模型选取

在主界面中，我们可以分别对分词、词向量生成以及分类网络进行选择，如下所示：



### 6.2.2 加载模型

分别对三个类别的方式进行选取，选择完毕后，点击“加载模型”，跳转到下一子界面窗口：



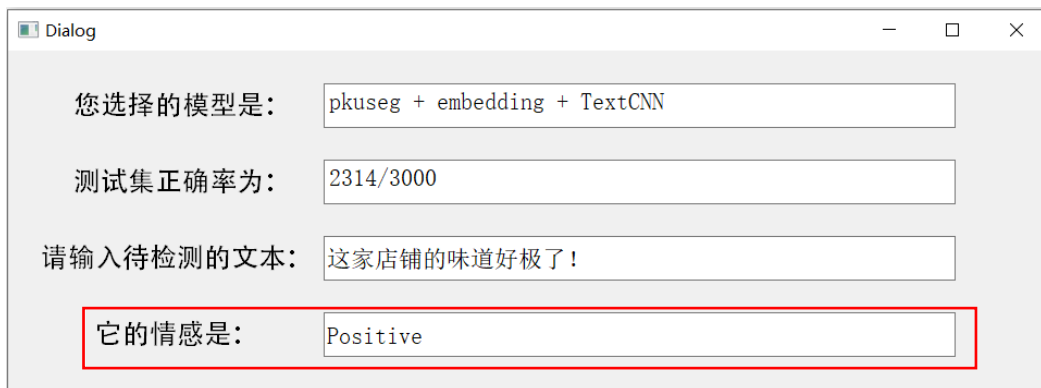
我们可以看到，右边的子窗口界面，显示出了选择的模型情况以及测试集的正确率。

## 6.2.3 人机交互实时文本预测

我们在子界面中输入对应的想要预测的文本，然后点击开始预测按键：



之后，我们就可以看到在“它的情感是”的栏目中，出现了“positive”的字样，表示我们系统预测该文本的结果是正面的：



以上就是我们的界面设计部分的讲解和展示。

## 7.系统评价及改进

整个系统搭建完成之后，我们对系统进行了客观的分析和评价，从以下几个角度说明一下我们这个系统的情况。

### ① 资源占用率：

相比于其他的一些大型网络，我们使用的 TextCNN 网络是一款轻量级的网络模型，整个的层数很少，因此所占用的资源很小，对计算机的性能要求也不是很高，因此使得整个系统只占用了很小一部分的计算资源，就能够达到很高的效果，因此效率是非常高的。

### ② 运行时长：

在训练阶段，由于我们使用的是轻量级的模型，因此训练的速度很快，运行时长较短，不过在训练端的运行时长的意义其实不是很大，因为市场上的模型一定是经过训练完成之后再投入使用的，主要需要看前端预测的时长。

我们从系统的运行来看，当我们输入一段文本的时候，每种方式的时长不等，比较快的方式能够在 0.5 秒之内做出响应，并把结果输出到界面显示上；而比较慢的大约在 2 秒之内能够做出响应。这种处理速度其实感觉不够快，不过也有可能和电脑的性能有关系，在运行的时候，能感觉到我们的电脑存在卡顿的现象。

### ③ 准确率：

我们模型对于测试集的准确率在 80% 上下浮动，算是一个比较不错的性能了，因为我们的词向量生成是基于训练集中的评论生成的，而测试集是根据训练集中生成的词向量进行的预处理，因此其实算是一种放在现实生活中的直接测试了，因此我觉得这样的效果，基于较小的资源消耗的情况下，是一个比较令人满意的结果。

但是，我们要考虑的是如何能够让准确率进一步提升呢？

我觉得影响准确率的一个主要原因，是词向量的生成。当遇到词库中没有出现的词时，我们通常是把这个词作为 0 向量生成了矩阵，这其实会对准确率造成较大的影响，我们通过查阅资料，大概了解了处理未登录词的方法有以下三种：

1. 用 UNK 标签表示所有未登录词，但是 UNK 的 embedding 一般不用零向量。然后将 UNK 词向量随机初始化，在训练的时候，根据词频  $f(w)$ ，当  $f(w) > 2$  时，采用  $z / (z + f(w))$  的概率把词随机变为 UNK，而  $z$  通常设为 0.8375，这一根据词频给定一个未登录的概率的方式用的算是比较多
2. 借鉴 fastText 思想。fastText 使用和 word2vec 几乎一样的算法，但由于 fastText 考虑了每个词的  $n$ -gram，最终可以用一个相似的向量来表示未登录词。
3. 也有的方法将每个未登录词都使用正态分布给他随机一个向量，感觉这个方法也是不算非常靠谱。

后续如果有机会的话，我们组会将这个问题进行继续研究，希望能够进一步提高网络的性能。

## 8. 总结感想与成员分工

### 张恒瑜

主要工作：完成词向量生成和神经网络搭建，以及界面设计的部分，和对应的报告撰写

感想：这次的大作业过程中，让自己接触到了一个全新的 NLP 领域，之前自己比较熟悉 CV 方面的工作，对 NLP 的整个流程不是很熟悉，刚好借此机会，熟悉了 NLP 的处理流程，掌握了一些基础的自然语言处理方法，从分词到词向量嵌入，再到对经典网络 TextCNN 的学习，是一次很棒的体验。同时，能够独立地使用 python 以及 QT 实现一个交互界面的设计，感觉也是非常棒的。我觉得《网络内容安全》这门课是一门非常有意义的课程，教会了我们很多关于机器学习方面的知识，是未来研究生阶段很重要的一项工具和技能。最后，我们组的开源代码上传到了我本人的 Github 账号上，欢迎大家一起讨论交流！

### 李卓音

主要工作：完成数据爬取和数据集的制作，以及对应的报告撰写和 ppt 制作

感想：这次实验过程中，我深入的了解了爬虫程序工作的原理，也更清晰的了解了网页架构和上网过程中服务器发送请求的方式。成功获取到数据的时候很有成就感，同时也意识到网络安全的重要性。在阅读其他同学部分报告时，对神经网络有了更深刻的理解。在我看来神

神经网络就是训练一种模型使得他能像人一样处理问题, 而我们对网络的训练就是让这个模型更加智慧化, 智能化。因为之前从未接触过神经网络方面的知识, 所以通过这门课对这一方向有了基本的了解, 感觉很神奇。

### 杜飞

主要工作: 完成中文分词的实现和比较, 以及对应的报告撰写和 ppt 制作

感想: 在这次的大作业过程中, 主要进行了中文分词部分的工作, 除了课堂上讲的中文分词知识以外, 在以前也是没有接触过此部分知识或者实践操作的, 但通过此次大作业的完成过程中, 对中文分词和对文本情感分析有了更加充分的了解和掌握。完成大作业的过程中, 通过查阅各种资料, 对不同的分词方法进行对比和学习, 一定程度上也开阔了自己的眼界和提升了自己的能力。当然, 大作业的完成也离不开两位队友的完美配合与合作, 虽然小组只有三个人, 但三人通过分工, 也完成了无论是系统的设计, 还是研讨 PPT 的制作或是报告的编写, 是一次非常愉快的合作过程。