燕山大学

Python 机器学习三级项目报告

Python Machine Learning Three-level Project Report

项目题目	基于集成学习的 AMAZON 用户评论质量预测		项目小组	软件工程 18-6 班第 7 组
小组成员	学号	姓名	项目成绩	所做工作
	201811040809	乔翱		AdaBoost 算法的手动实现，特征工程，部分报告书写，PPT 制作
	201811040807	李华宪		Bagging 算法的手动实现，部分报告书写
	201811040810	王紫晔		数据预处理，模型训练及评估，部分报告书写
	201811040808	刘祺		模型训练及评估，部分报告书写，PPT 制作
指导教师	于浩洋 李可 郝晓冰			

教 务 处

2021 年 3 月

目　录

基于集成学习的 AMAZON 用户评论质量预测

1 项目内容

随着电商平台的兴起，以及疫情的持续影响，线上购物在我们的日常生活中扮演着越来越重要的角色。在进行线上商品的挑选时，评论往往是我们十分关注的一个方面。然而目前电商网站的评论参差不齐，甚至有水军刷好评或者恶意差评的情况出现，严重影响了顾客的购物体验。因此，对于评论质量的预测成为电商平台越来越关注的话题，如果能自动对评论质量进行评估，就能根据预测结果避免展现低质量的评论。

本次项目是基于集成学习方法对 Amazon 现实场景中的评论质量进行预测。需要完成两种集成学习算法的实现（Bagging、AdaBoost.M1），其中基分类器使用 SVM 和决策树两种，对结果进行对比分析。

2 项目原理

2.1 项目构思

本项目是一个典型的自然语言处理的问题，以自然语言处理为背景，对评论文本的质量进行分类，这是一个典型的文本分类问题，对于此类问题重点在于对于文本的处理以及模型的构建，文本处理是一个重点也是一个难点，文本的处理对于最后模型的效果可以说是决定性的。

首先先要对数据集进行分析，由于是文本数据，可以对文本长度进行统计，统计单词出现的频率，统计每句话中出现最多的单词，查看标签分类的分布，对于这些数据的分析采用可视化形式（柱状图、直方图等）进行分析。

之后进行预处理，进行数据清洗，首先先要去除文本中的标点符号，其次还应该统一单词中的字母大小写，还需要去除文本中的停用词，这些词会对模型训练造成很大影响。

预处理之后要进行特征工程，把文本数据向量化。

对于模型的选择，由于此次项目要求的是利用集成学习来训练模型，所以采取集成学习的方法来训练模型，采用 Bagging 和 Adaboost 方法，并且改变其中的参数，比较实验结果，分析实验结果选出最优模型。

最后进行模型的评估，对比实验评估两种集成学习算法，采用 auc 作为评价指标，进行对模型的评估。

由于本项目并没有提供 baseline，所以在项目初期快速构建了一个简单的 baseline，baseline 是基于 one-hot+逻辑回归构建，完后不断优化 baseline，提高准确率。

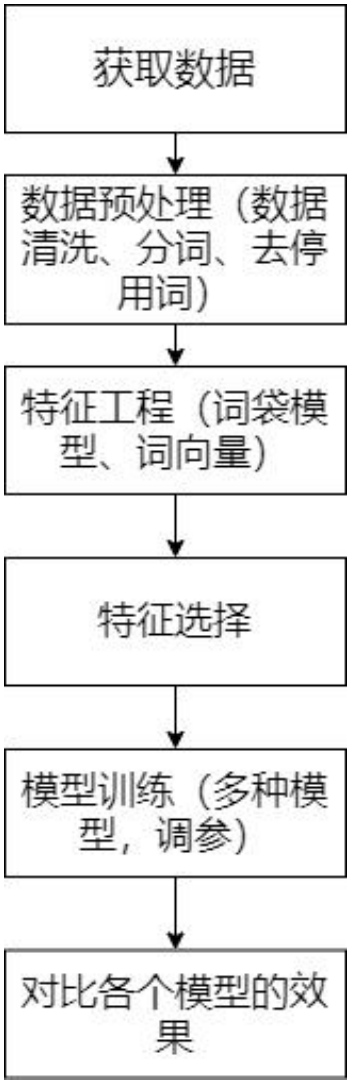

图 2-1 项目流程

2.2 算法模型

1、Bagging

Bagging 算法（英语：Bootstrap aggregating，引导聚集算法），又称装袋算法，是机器学习领域的一种团体学习算法，它的特点是各个弱学习器之间没有依赖关系，可以并行拟合。bagging 的集合策略比较简单，对于分类问题，通常使用简单投票法，得到最多票数的类别或者类别之一为最终的模型输出。

bagging 算法流程：

输入为样本集 D={(x,y1),(x2,y2),…(xm,ym)}，弱学习器算法，弱分类器迭代次数 T。

输出为最终的强分类器 f(x)。

1）对于 t=1,2…,T:

a)对训练集进行第 t 次随机采样，共采集 m 次，得到包含 m 个样本的采样集 Dt

b)用采样集 Dt 训练第 t 个弱学习器 Gt(x)

2) 如果是分类算法预测,则 T 个弱学习器投出最多票数的类别或者类别之一为最终类别。如果是回归算法，T 个弱学习器得到的回归结果进行算术平均得到的值为最终的模型输出。（实验要求分类模型，故只讨论分类算法）

2、AdaBoost.M1

AdaBoost.M1 算法是基于 AdaBoost 算法的一个改进版本,AdaBoost 算法计算基分类器的分类误差率和权重参数都是基于此类标签的指示函数进行调整的。策略为：依据当前分类器的分类错误率，调整样本权值分布，保证错误分类样本的权重增大，正确分类样本的权重减小；且调整当前分类器在最终决策的权重。而 AdaBoost.M1 将其改进为多分类的算法，策略为：依据当前分类器的分类错误率，调整样本权值分布，保证错误分类样本的权重不变（与 AdaBoost 不同），正确分类样本的权重减小；且调整当前分类器在最终决策的权重（与 AdaBoost 的调整公式相同）。

Algorithm 4 AdaBoost.M1

Input: Training set $S = \{\mathbf{x}_i, y_i\}$, $i = 1, \ldots, N$; and $y_i \in \mathbb{C}, \mathbb{C} = \{c_1, \ldots, c_m\}$; T: Number of iterations; I: Weak learner

Output: Boosted classifier:

$$H(x) = \arg\max_{y \in \mathbb{C}} \sum_{t=1}^{T} \ln\left(\frac{1}{\beta_t}\right) [h_t(x) = y] \text{ where } h_t, \beta_t$$

are the induced classifiers (with $h_t(x) \in \mathbb{C}$) and their assigned weights, respectively

1: $D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
2: **for** $t = 1$ to T **do**
3: $\quad h_t \leftarrow I(S, D_t)$
4: $\quad \varepsilon_t \leftarrow \sum_{i=1}^{N} D_t(i)[h_t(\mathbf{x}_i) \neq y_i]$
5: $\quad$ **if** $\varepsilon_t > 0.5$ **then**
6: $\quad\quad T \leftarrow t - 1$
7: $\quad\quad$ **return**
8: $\quad$ **end if**
9: $\quad \beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
10: $\quad D_{t+1}(i) = D_t(i) \cdot \beta_t^{1-[h_t(\mathbf{x}_i) \neq y_i]}$ for $i = 1, \ldots, N$
11: $\quad$ Normalize D_{t+1} to be a proper distribution
12: **end for**

图 2-2 AdaBoost.M1 算法流程

3、其他算法模型

在初期快速构建一个 baseline 的时候采取了 sklearn 中的逻辑回归模型,在后期进行改进的时候使用了 xgboost 模型和 lightgbm 模型

逻辑回归是用来做分类算法的，线性回归的一般形式是 Y=aX+b，把 Y 的结果带入一个非线性变换的 Sigmoid 函数中，即可得到[0,1]之间取值范围的数 S，S 可以把它看成是一个概率值，如果设置概率阈值为 0.5，那么 S 大于 0.5 可以看成是正样本，小于 0.5 看成是负样本，就可以进行分类了。

GBDT(Gradient Boosting Decision Tree)，全名叫梯度提升决策树，使用的是 Boosting 的思想。GBDT 的原理就是所有弱分类器的结果相加等于预测值，然后下一个弱分类器去拟合误差函数对预测值的残差(这个残差就是预测值与真实值之间的误差)。LightGBM 就是以 GBDT 为基础的。GBDT 在每一次迭代的时候，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间。尤其面对工业级海量的数据，普通的 GBDT 算法是不能满足其需求的。LightGBM 提出的主要原因就是为了解决 GBDT 在海量数据遇到的问题，让 GBDT 可以更好更快地用于工业实践。

XGBoost 的核心算法思想基本就是：不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数 f(x)，去拟合上次预测的残差。当训练完成得到 k 棵树，要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中会落到对应的一个叶子节点，每个叶子节点就对应一个分数。最后只需要将每棵树对应的分数加起来就是该样本的预测值。显然，目标是要使得树群的预测值$\hat{y_i}$尽量接近真实值y_i，而且有尽量大的泛化能力。类似 GBDT，XGBoost 也是需要将多棵树的得分累加得到最终的预测得分（每一次迭代，都在现有树的基础上，增加一棵树去拟合前面树的预测结果与真实值之间的残差）。

2.3 数据集

数据集中的数据来源于 Amazon 电商平台中的商品评论信息，其中包括用户 ID、商品 ID、英文评论、商品打分、认为评论有用的点赞数、该评论得到的总评价数以及评论的质量。训练数据总共有五万多条，可以看到进行训练的数据量还是很大的，其中用户 ID、商品 ID 在此场景下是无关属性。在进行预处理的时候，数据里面最主要用到的就是评论文本信息，需要进行评论的分词、去除停用词。之后进行特征工程，选取特征并对文本进行向量转换，为之后的模型训练做准备。

2.4 词嵌入（word embedding）

自然语言是一套用来表达含义的复杂系统。在这套系统中，词是表义的基本单元。顾名思义，词向量是用来表示词的向量，也可被认为是词的特征向量或表征。把词映射

为实数域向量的技术也叫词嵌入（word embedding）。

在 NLP(自然语言处理)领域，文本表示是第一步，也是很重要的一步，通俗来说就是把人类的语言符号转化为机器能够进行计算的数字，因为普通的文本语言机器是看不懂的，必须通过转化来表征对应文本。早期是基于规则的方法进行转化，而现代的方法是基于统计机器学习的方法。

数据决定了机器学习的上限,而算法只是尽可能逼近这个上限,在本文中数据指的就是文本表示，所以，弄懂文本表示的发展历程，对于 NLP 学习者来说是必不可少的。接下来开始我们的发展历程。文本表示分为离散表示和分布式表示。

One-hot 简称读热向量编码，也是特征工程中最常用的方法。首先构造文本分词后的字典，每个分词是一个比特值，比特值为 0 或者 1。每个分词的文本表示为该分词的比特位为 1，其余位为 0 的矩阵表示。

TF-IDF 是一种用于信息检索与数据挖掘的常用加权技术。TF 意思是词频，IDF 意思是逆文本频率指数。字词的重要性伴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。一个词语在一篇文章中出现的次数越多，同时在所有文档中出现次数越少，越能代表该文章。

谷歌 2013 年提出的 Word2Vec 是目前最常用的词嵌入模型之一。Word2Vec 是一种浅层的神经网络模型，它有两种网络结构，分别是 CBOW（Continues Bag of Words）连续词袋和 Skip-gram。

CBOW 获得中间词两边的的上下文，然后用周围的词去预测中间的词，把中间词当做 y，把窗口中的其它词当做 x 输入，x 输入是经过 one-hot 编码过的，然后通过一个隐层进行求和操作，最后通过激活函数 softmax，可以计算出每个单词的生成概率，接下来的任务就是训练神经网络的权重，使得语料库中所有单词的整体生成概率最大化，而求得的权重矩阵就是文本表示词向量的结果。

Skip-gram 是通过当前词来预测窗口中上下文词出现的概率模型，把当前词当做 x，把窗口中其它词当做 y，依然是通过一个隐层接一个 Softmax 激活函数来预测其它词的概率。

3 项目实现

3.1 实践流程

1、数据读取与数据分析

对于数据的读取，直接读取 csv 文件即可，读取后对数据进行分析，可以看出数据中没有空值。

对文本长度进行了统计，可以看出文本的平均长度为 255，最长的文本有四千多个词，最短的文本只有 1 个词。对 label 类别进行统计，可以看出在训练集五万多条数据

中，有四万多条都是低质量评论。最后统计了单词出现的次数，总共有六十多万个不同的词，其中 the 出现的次数最多，出现了七十多万次，而在出现次数最多的几个词中，大部分都是停用词或语气词，这需要在之后对这些词进行处理。

通过对数据分析可以得出以下结论：

1、每个评论的平均单词数为 255，并不是特别多，但是有些单词的长度过多，可能需要截断，而某些文本的单词数较少这些文本很可能是低质量文本，所以在之后可能根据文本的长度构造特征。

2、类别很不均衡，可能会严重影响模型的精度。

2、数据预处理

对于 NLP 任务，首先需要对文本进行分词，但是由于本次项目中的评论是英文文本，所以只需要依据英文之间的空格分隔进行分词即可，分词较为简单。

分析数据后可以看出首先数据集没有空数据的情况，所以并不需要对空值进行处理。在之前对数据的分析分析中可以得到，文本的长度的平均值为 255，最长的文本有四千多个词，而最短的只有一个词。而分析了单词出现次数后，可以看出出现次数最多的几个词都是停用词或语气词（the,a 等），这类词会对文本分类产生不良影响，所以需要去掉，另外文本中的标点符号也应去掉，还有就是对于文本应统一大小写。

3、特征工程

首先对于特征选择，在本次项目中只是选择了 reviewText 这一个文本特征，对于其他的特征，例如 overall、votes_up（只在训练集出现）、votes_all（只在训练集出现）并没有选取。试验过选取 overall，但是对于整个模型并没有太大的影响。

由于机器学习中需要的数据必须是数值型，对于字符型数据不能直接进行放进机器学习模型直接训练，需要把分词转换为计算机能够计算的类型，一般为向量。常用的方法为词袋模型和词向量。

在本次项目的特征工程中试验了两种办法，在初期使用了 one-hot 编码，但是效果不是很好，因为文本中不同的单词数量很大，所以数据特征的维度就会很大，产生一个维度很高，又很稀疏的矩阵。并且这种表示方法的分词顺序和在句子中的顺序无关，不能保留词与词之间的关系信息。

在之后采取了词袋模型，主要采取了 Count Vectors 和 TF-IDF，相对 one-hot 编码来说，模型的效果有了显著提升。

4、模型训练

本项目要求手动实现 AdaBoost 和 Bagging 算法，由于手动实现集成学习算法较为困难，所以在项目初期，是利用 one-hot+逻辑回归快速构建了一个 baseline，之后在 baseline 基础上不断提高。

在手动实现了 Adaboost 和 Bagging 后,则是对比实验对这两种集成学习算法进行了一个比较，并且采用不同的分类器不同的参数进行对比实验比较。

在后期为了提高预测准确率，本小组使用了 XGBoost 和 LightGBM 模型，进行训练，经过不断的调参，最终模型的效果好于 Bagging 和 AdaBoost。

5、结果评估

对于结果评估，一方面是在比赛网站上提交模型预测结果，另外在比较各个模型的时候，采用 AUC 作为评价指标，并且画出 ROC 曲线进行对比。

AUC 是一种衡量机器学习模型分类性能的重要且非常常用的指标，其只能用于二分类的情况。AUC 的本质含义反映的是对于任意一对正负例样本，模型将正样本预测为正例的可能性大于将负例预测为正例的可能性的概率。AUC 更直观反应了 ROC 曲线表达的模型分类能力。其数值大小代表了模型的性能优劣。

6、结果分析及模型对比

对两种集成学习算法进行对比，采用 AUC 作为评价指标，对比实验结果如下：

集成学习算法	基分类器	AUC
AdaBoost	决策树	0.729
	SVM	0.761
	朴素贝叶斯	0.530
Bagging	决策树	0.714
	SVM	0.731
	Knn	0.667
	朴素贝叶斯	0.724

分析：经过对比实验可以看出，AdaBoost+SVM 的组合 AUC 最高。

bagging 的训练样本集的选择是随机的。各轮训练样本集之间是独立的。 bagging 的 m 个分类器的分类函数是并行生成的。进行测试的时候采用的是投票机制。boosting 的各轮样本集的选择和前面各轮的学习结果有关， boosting 的各个预测函数只能顺序生成的，所以训练比较耗时间。预测的时候采用的是 weighted sum。bagging 可通过并行训练节省训练时间，bagging 和 boosting 都可以提高分类的准确性。 大多数数据集中， boosting 的准确率是比 bagging 高。

而使用 XGBoost 和 LightGBM 后，经过调参后，LightGBM 模型预测准确率最高，最终本小组选取的最优模型就是 TD-IDF+LGB，提交到 paddle 上，分数可以达到 0.79。

3.2 程序源码及说明

1、数据分析

在对数据进行初步分析的时候,主要是采用 dataframe 类型的 info 函数和 describe 方法，核心代码如下：

```
train_data.info()
train_data.describe()
```

在对数据分析的时候，统计了文本的长度，并且将句子长度绘制了直方图，相关核心代码如下：

```
##统计文本长度
train_data['text_len'] = train_data['reviewText'].apply(lambda x: len(
x.split(' ')))
print(train_data['text_len'].describe())
_ = plt.hist(train_data['text_len'], bins=200)
plt.xlabel('Text char count')
plt.title("Histogram of char count")
```

对数据集的类别分布统计，统计每类的评论的个数，相关核心代码如下：

```
##统计 label 种类
train_data['label'].value_counts().plot(kind='bar')
plt.title(comment class count')
plt.xlabel("category")
```

统计单词出现次数，相关核心代码如下：

```
##统计单词出现次数
from collections import Counter
all_lines = ' '.join(list(train_data['reviewText']))
word_count = Counter(all_lines.split(" "))
word_count = sorted(word_count.items(), key=lambda d:d[1], reverse = T
rue)

print(len(word_count))
print(word_count[0])
print(word_count[-1])
```

2、数据预处理

大小写转换、去停用词、去除标点符号，核心代码如下：

```
stopwords={}.fromkeys([line.rstrip() for line in
open('./stopwords.txt',encoding='utf-8')])
eng_stopwords=set(stopwords)
```

```
def clean_text(text):
    text=re.sub(r'[^a-zA-Z]',' ',text)
    words=text.lower().split()
    words=[w for w in words if w not in eng_stopwords]
    return ' '.join(words)
```

3、特征工程

特征工程主要采用了 one-hot，Count Vectors 和 TF-IDF，相关核心代码如下所示：

```
train_text = train_df['reviewText']
test_text = test_data['reviewText']
all_text = pd.concat([train_text, test_text])
tfidf = TfidfVectorizer(ngram_range=(1,10), max_features=1000000)
tfidf.fit(train_text)
train_test = tfidf.transform(train_df['reviewText'])
test_data=tfidf.transform(test_data['reviewText'])

vectorizer = CountVectorizer(max_features=3000)
train_test = vectorizer.fit_transform(train_df['reviewText'])
```

4、模型训练

Bagging 算法手动实现，相关核心代码如下所示：

```
class Bagging:
    def __init__(self,n_estimators,estimator,number):
        self.estimator = estimator
        self.n_estimators = n_estimators
        self.number=number
        self.test_data=list()

    def RepetitionRandomSampling(self,data,number):
        print(len(data))
        data = np.array(data)
        sample=[]
        for i in range(int(self.number)):
             sample.append(data[random.randint(0,len(data)-1)])
        return sample

    def fits(self,data):
        x_train,x_test,y_train,y_test=train_test_split(data['reviewTex
t'],data['label'],test_size=0.2)
```

```
        test = tfidf.transform(x_test)
        test.toarray()
        self.test_data=np.array(y_test)
        print("y_test[0]:")
        print(self.test_data[0])
        print(self.test_data)
        for i in range (self.n_estimators):
            train_data=data.sample(frac=0.1,replace=True,axis=0)
            train= tfidf.transform(train_data['reviewText'])
            train.toarray()
            clf=self.estimator
            clf.fit(train,train_data['label'])
            m.append(clf.predict(test))
        return m

    def Score(self):
        global result
        term = np.transpose(m)
        #print(len(term))
        for i in range (len(term)):
            if(term[i].mean()>=0.5):
                result.append(1)
            if(term[i].mean()<0.5):
                result.append(0)
        result=np.array(result)
        count=0
        for i in range(len(term)):
            if self.test_data[i]==result[i]:
                count +=1
        print(count/np.float(len(term)))
        return count/np.float(len(term))
```

AdaBoost.M1 算法手动实现，相关核心代码如下所示：

```
class AdaBoost:
    # 基分类器默认为多分类的逻辑回归
    def __init__(self,m,\
        clf=LogisticRegression()):
        # 基分类器数量
        self.m = m
        # 基分类器模型
```

```
        self.clf = clf
        # 缓存基分类器和权重参数
        self.clf_arr = []
        self.alpha_arr = []
    # 指定训练数据集、基分类器、迭代次数

    def fit(self,X,Y):
        num = X.shape[0]
        # 初始化样本权重
        W = np.ones(num) / num
        # 迭代
        for i in range(self.m):
            # 基分类器预测
            self.clf.fit(X, Y, sample_weight=W)
            self.clf_arr.extend([self.clf])
            Y_pred = self.clf.predict(X)
            # 分类误差率
            indic_arr = [1 if Y_pred[j]!= Y[j] else 0 for j in range(n
um)]
            err = np.dot(W, np.array(indic_arr))
            print("classify error rate is ",err)
            if err>0.5:
              self.m=i-1
              return
            # 分类器系数
            alpha = err/(1-err)
            self.alpha_arr.extend([1-alpha])
            # 更新权重
            temp = W * (alpha**[1-i for i in indic_arr])
            W = temp / np.sum(temp)
        return self

    def predict(self, X,Y):
        num = X.shape[0]
        mulit_idx_pred = []
        mulit_Y_pred = []
        for i in range(self.m):
            Y_pred = self.clf_arr[i].predict(X)
            mulit_Y_pred.append(Y_pred.tolist())
            indic_arr=[1 if Y_pred[i]=Y[i] else 0 for i in range(num)]
```

```
            temp = [np.log(self.alpha_arr[i])*k for k in indic_arr]
            mulit_idx_pred.append(temp)
        # 保存各个基分类器预测的最大索引
        max_idx_pred = np.array(mulit_idx_pred).argmax(axis=0)
        # 保存各个基分类器的预测值
        mulit_Y_pred = np.array(mulit_Y_pred).T
        # 获取最大索引对应的预测标签
        result = np.array([x[max_idx_pred[i]] for i,x in enumerate(mul
it_Y_pred)],dtype=np.float64)
        return result

    def score(self,X,Y):
        Y_pred = self.predict(X,Y)
        count = 0.
        for i in range(len(Y)):
            if Y_pred[i]==Y[i]:
                count +=1
        return count/np.float(len(Y))
```

XGBoost 模型相关核心代码如下所示：

```
class XGB():
    def __init__(self, X_df, y_df):
        self.X = X_df
        self.y = y_df

    def train(self, param):
        self.model = XGBClassifier(**param)
        self.model.fit(self.X, self.y,
                       eval_set=[(self.X, self.y)],
                      eval_metric=['mlogloss'],
                      early_stopping_rounds=10
                     )
           train_result, train_proba = self.model.predict(self.X),
self.model.predict_proba(self.X)
        train_acc = accuracy_score(self.y, train_result)
        train_auc = f1_score(self.y, train_proba, average='macro')
        print("Train acc: %.2f%% Train auc: %.2f" % (train_acc*100.0,
train_auc))

    def test(self, X_test, y_test):
```

```
        result, proba = self.model.predict(X_test),
self.model.predict_proba(X_test)
        acc = accuracy_score(y_test, result)
        f1 = f1_score(y_test, proba, average='macro')
        print("acc: %.2f%% F1_score: %.2f%%" % (acc*100.0, f1))

    def grid(self, param_grid):
        self.param_grid = param_grid
        xgb_model = XGBClassifier(nthread=20)
        clf = GridSearchCV(xgb_model, self.param_grid,
scoring='f1_macro', cv=2, verbose=1)
        clf.fit(self.X, self.y)
        print("Best score: %f using parms: %s" % (clf.best_score_,
clf.best_params_))
        return clf.best_params_, clf.best_score_
```

LightGBM 模型相关核心代码如下所示：

```
import lightgbm as lgb

clf = lgb.LGBMClassifier(
        boosting_type='gbdt', num_leaves=55, reg_alpha=0.0, reg_lambda=1,
        max_depth=15, n_estimators=6000, objective='binary',
        subsample=0.8, colsample_bytree=0.8, subsample_freq=1,
        learning_rate=0.06, min_child_weight=1, random_state=20, n_jobs=4
    )
```

3.3 调试过程

1、预处理调试

（1）在预处理分词的时候，在得到每个词之后，将字母转换为小写的过程中发现运行时出现错误：列表中不包含转换为小写的操作。

```
----------------------------------------------------------------------------AttributeError
st)<ipython-input-4-fae137818521> in <module>
      8 print(stop_words)
      9 for i in range(len(train_s)):
---> 10     train_s[i]=train_s[i].lower()#转换为小写
     11     train_s[i]=re.sub(',',' ',train_s[i])
     12     train_s[i]=re.sub('\.',' ',train_s[i])
AttributeError: 'list' object has no attribute 'lower'
```

图 3-1 list 没有 lower 方法

之后将列表转化为数组试试，但还是在调试的过程中遇到了错误，数组中也不包含转化为小写的操作。

```
---------------------------------------------------------------------------AttributeError
st)<ipython-input-7-6db7d68b0d35> in <module>
      9 for i in range(len(train_s)):
     10     train_s[i]=np.array(train_s[i])
---> 11     train_s[i]=train_s[i].lower()#转换为小写
     12     train_s[i]=re.sub(',',' ',train_s[i])
     13     train_s[i]=re.sub('\.',' ',train_s[i])
AttributeError: 'numpy.ndarray' object has no attribute 'lower'
```

图 3-2 ndarray 没有 lower 方法

之后在不断的搜索资料，发现字符串是可以进行转换为小写的，于是将其转换为 str 类型，再进行调试就正确了。

（2）在进行去除符号‘\\’的时候，调试过程也出现了错误。

```
     22     train_s[i]=re.sub('/',' ',train_s[i])
     23     train_s[i]=re.sub('"',' ',train_s[i])
---> 24     train_s[i]=re.sub('\\','',train_s[i])
     25     #将每个词得到
     26     word_tokens=word_tokenize(train_s[i])
```

图 3-3 调试中没加转义字符

在问同学之后，知道了\转义字符，如果需要去除它本身的话，需要每个前面都加上，于是修改其为‘\\\\’，调试成功。

2、Bagging 调试过程

（1）在进行循环训练时中进行了数据划分，导致最后预测结果与真实结果无法比较

```
for i in range (self.n_estimators):
    train_data=data.sample(frac=0.1,replace=True,axis=0)
    x_train,x_test,y_train,y_test=train_test_split(train_data['reviewText'],train_data['label'],test_size=0.2)
    #train_data=self.RepetitionRandomSampling(data,self.number)
```

图 3-3 数据集划分不当

后改为进行模型训练之前进行数据划分，结果正确。

（2）将 train['reviewText']转化为 array 形式，就进行训练，导致训练类型不匹配。

```
ValueError: setting an array element with a sequence.
```

图 3-4 数据类型不匹配错误

后将 train['reviewText']改为 np.array(train['reviewText'])解决了这一问题。

（3）由于 y_test 是在 fit()函数中划分出来的，故在 score()函数中无法引用，后将 y_test 改为 bagging 类的变量（self.y_test）解决这一问题。

3、尝试深度学习模型

采用 fastText 来训练模型，但是每次在预测的时候都会提示会话奔溃，最终也没有找到解决办法，猜想可能的原因是数据量太大了。

```
#  转换为FastText需要的格式
train_df  =  pd.read_csv('/content/ctrain.csv',nrows=15000)
train_df=train_df[['reviewText','label']]
train_df=train_df.dropna()
train_df['label']  =  '__label__'  +  train_df['label'].astype(str)
train_df[['reviewText','label']].iloc[:-5000].to_csv('train.csv',  index=None,  header=None)

model  =  fasttext.train_supervised('train.csv',lr=1.0,wordNgrams=2,verbose=2,minCount=1,epoch=25)

val_pred  =  [model.predict(x)[0][0].split('__')[-1]  for  x  in  train_df.iloc[-5000:]['reviewText']]
print(f1_score(train_df['label'].values[-5000:].astype(str),  val_pred,  average='macro'))
```

图 3-4 fasttext 调试出错

在尝试 LSTM 过程中也遇到了很多的问题，开始调试过程中是由于文本格式不正确导致出错，后期建立字典，把文本转换为数字，放入模型训练，虽然没有报错，但是最后预测准确率只有百分之六十左右，可能是模型的配置出了问题，最后此问题也没有解决。

```
from  keras.layers  import    LSTM
from  keras.models  import  Sequential
from  keras.layers  import  Flatten,Dense,Embedding
model=Sequential()
model.add(Embedding(10000,32))
model.add(LSTM(32))
model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
history=model.fit(x_train,y_train,epochs=10,batch_size=32,validation_split=0.2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-3292b797b26d> in <module>()
      7 model.add(Dense(1,activation='sigmoid'))
      8 model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
----> 9 history=model.fit(x_train,y_train,epochs=10,batch_size=32,validation_split=0.2)
```

图 3-5 LSTM 调试出错

3.4 项目结果

1、数据分析

文本长度统计：

```
count    57039.000000
mean       255.126335
std        242.025413
min          1.000000
25%        100.000000
50%        187.000000
75%        325.000000
max       4322.000000
Name: text_len, dtype: float64
Text(0.5,1,'Histogram of char count')
```

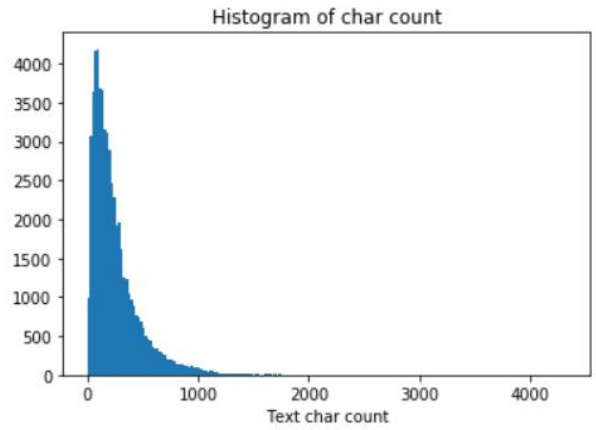

图 3-6 文本长度统计

Label 分布：

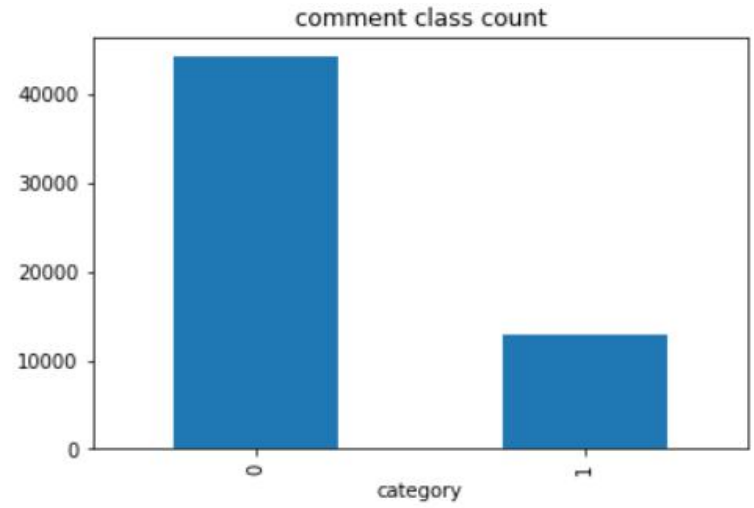

图 3-7 label 分布

单词出现次数统计：

```
[('the', 739355),
 ('and', 391103),
 ('of', 361795),
 ('a', 354147),
 ('to', 322494),
 ('is', 256211),
 ('', 250515),
 ('in', 206754),
 ('I', 164299),
 ('that', 157172),
 ('this', 140488),
 ('it', 122320),
 ('for', 108880),
 ('with', 106773),
 ('as', 104036),
 ('was', 102829),
 ('on', 85198),
 ('The', 84878),
 ('are', 77620),
 ('but', 77299),
 ('you', 68734),
 ('have', 67179),
 ('not', 66506),
 ('his', 64217),
 ('be', 61644),
 ('movie', 60203),
 ('film', 51392),
 ('by', 51001),
 ('one', 49246),
 ('from', 49159),
 ('an', 48203),
```

图 3-8 单词频率统计

2、集成学习算法 ROC 曲线

对于模型的评估，采取 AUC 作为评估指标，绘制 ROC 曲线进行比较，ROC 曲线下的面积就是 AUC。

AdaBoost+决策树：

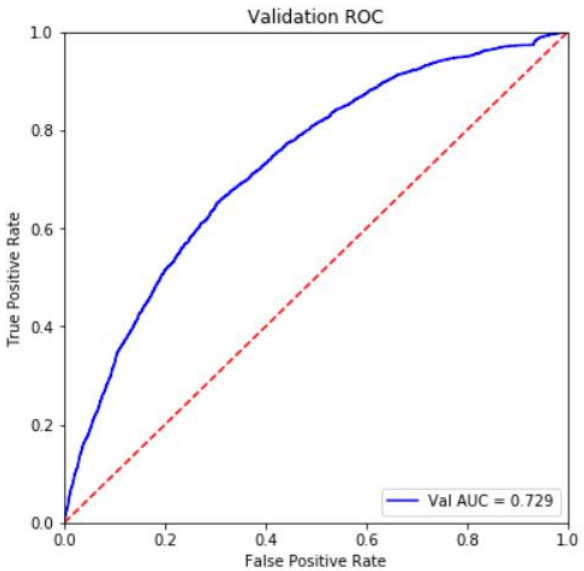

图 3-9 AdaBoost+DecistionTree ROC 曲线

AdaBoost+SVM：

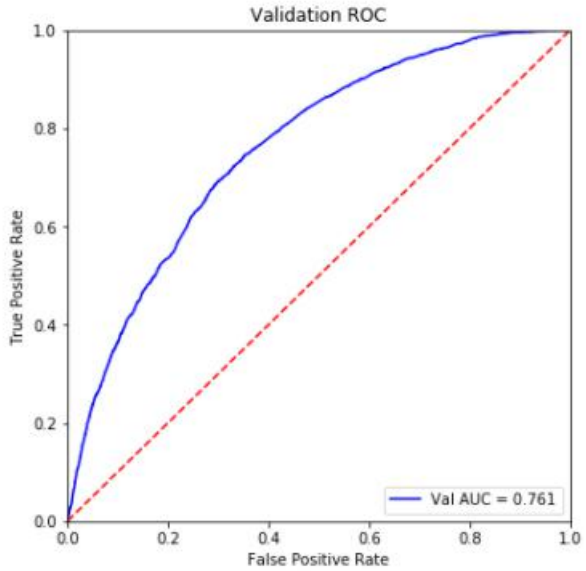

图 3-10 AdaBoost+SVM ROC 曲线

AdaBoost+朴素贝叶斯：

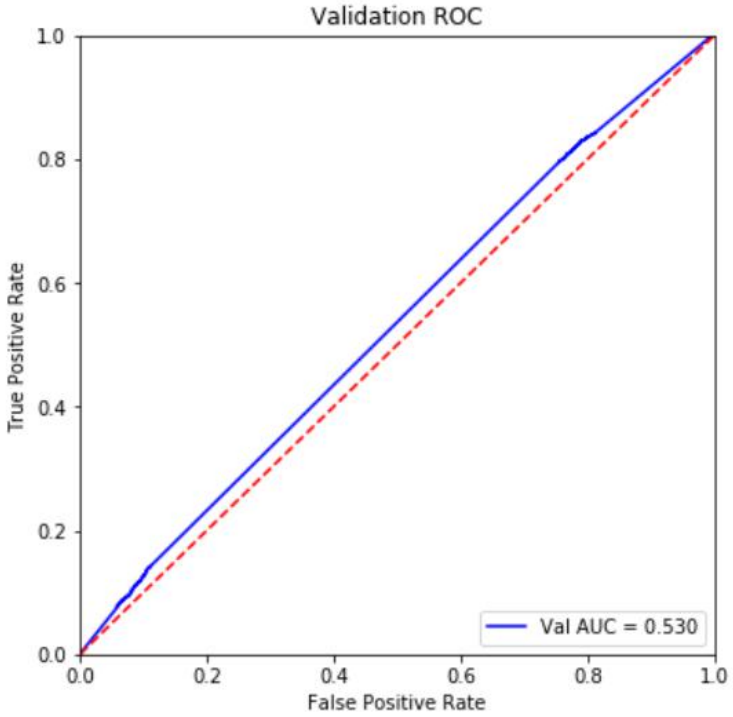

图 3-11 AdaBoost+朴素贝叶斯 ROC 曲线

Bagging+决策树：

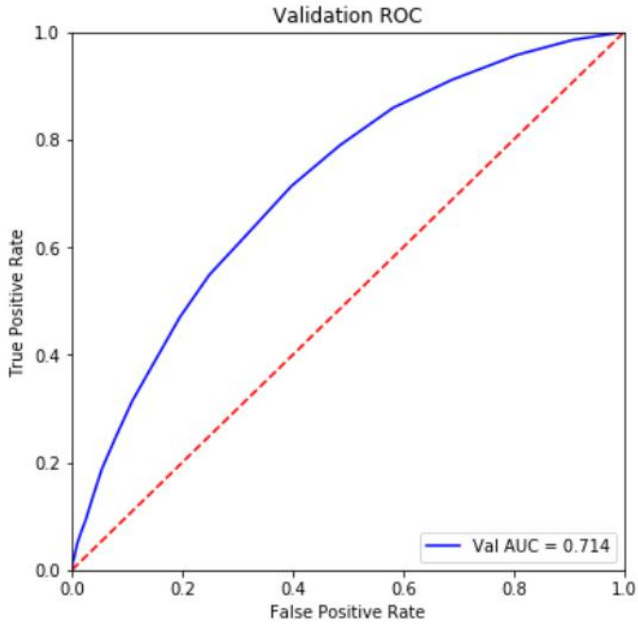

图 3-12 Bagging+DscisitionTree ROC 曲线

Bagging+SVM：

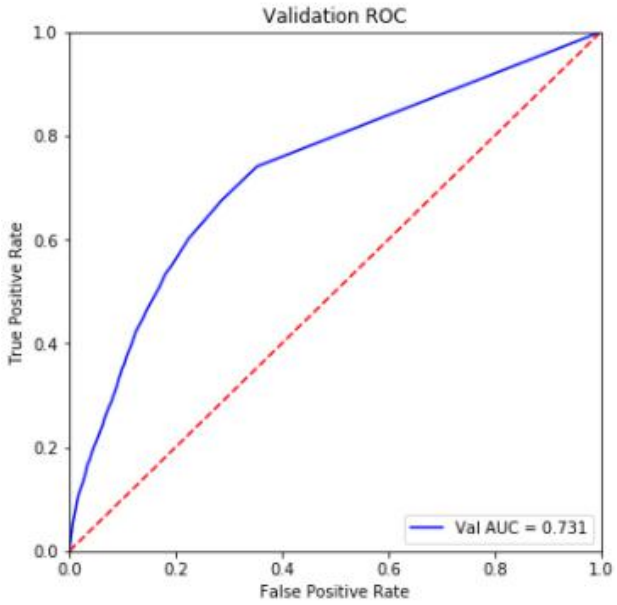

图 3-11 Bagging+SVM ROC 曲线

Bagging+knn：

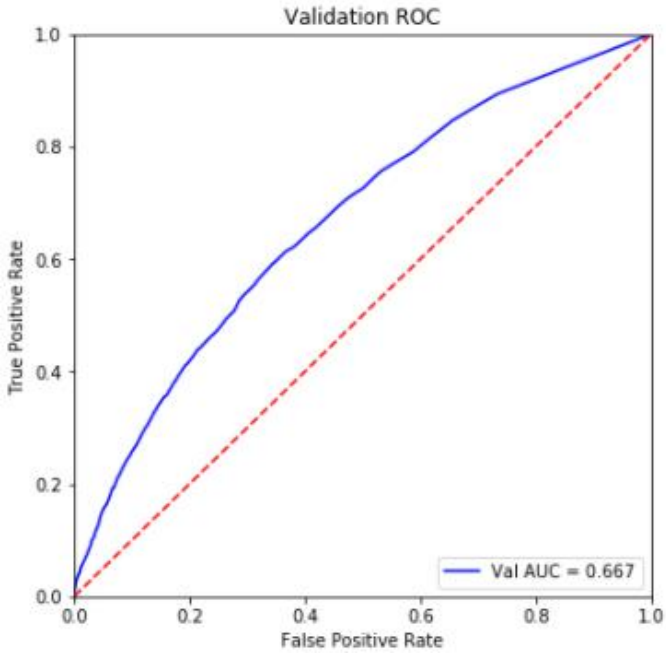

图 3-11 Bagging+SVM ROC 曲线

Bagging+朴素贝叶斯：

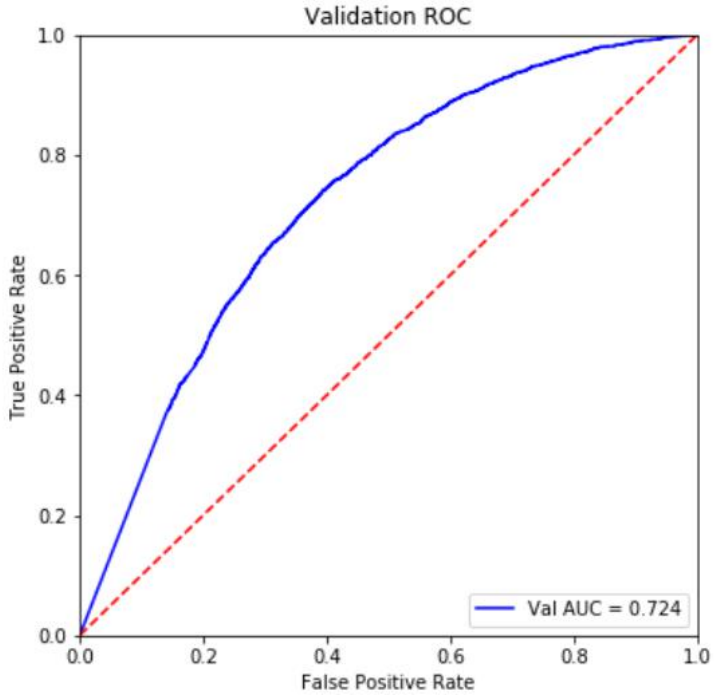

图 3-12 Bagging+朴素贝叶斯 ROC 曲线

4 项目总结

4.1 遇到的问题与解决办法

1、在预处理阶段，由于对 python 语法不熟悉，在去停用词的时候遇到了一些困难，后来查阅资料，发现 python 中有内嵌的停用词库 ntlk，最终利用 ntlk 进行去停用词，并且利用正则表达式去掉了文本中的标点符号，清洗了数据。

2、手动实现了 AdaBoostig 和 Bagging 的时候遇到了很多困难，虽然两种算法的思想并不难理解，但是实际实现起来这两种算法还是遇到了不少的困难，最后在查阅别人的资料、博客，“站在巨人的肩膀上”，自己手动实现了两种集成学习算法。

3、在词嵌入阶段，开始使用 one-hot，但是效果很差，一个是训练时间很长，另外训练出的模型效果也很差，查阅了相关资料，采取了别的方法，才提高了模型的准确率。

4.2 接下来待改进的方面

1、数据预处理

在数据预处理方面尝试其它的处理方法，例如对于词性进行标注，对于文本中的每个词，对其词性进行标注，经过实验判断哪类词对模型影响较大，加大这些词的特征重要性。

进行词形还原，对词进行还原。把各种时态的单词还原成单词的基本形态。词形还原就是去掉单词的词缀，提取单词的主干部分，通常提取后的单词会是字典中的单词。比如，单词“cars”词形还原后的单词为“car”，单词“ate”词形还原后的单词为“eat”。

2、特征工程

在词嵌入的过程中尝试其他的办法，例如 word2vec、glove 等，尝试不同的词嵌入模型，可以对这些模型训练出的词向量进行选择融合，选出每个词向量模型中的部分词向量。

本项目在特征选择方面，只选了文本特征，并没有选择其他特征，例如点赞数，评分等特征，在之后的学习过程中，可以尝试使用其他的特征，或者来构造新的特征。由于 votes_up 和 votes_all 只在训练集出现，没有在测试集出现，所以可以想办法把这两个特征和商品联系起来，放入模型进行训练。

3、模型选择

在本次项目中尝试了大多数的机器学习模型，除了题目要求的集成学习模型，另外还尝试了其他的深度学习模型，例如：xgb、lgb 等模型。根据一般经验，深度学习模型往往比机器学习模型效果更好，在本次项目的进行过程中，也尝试使用了深度学习模型，尝试使用了 LSTM、FastText，但是都没有成功调试出正确的结果，在接下来的学习过程

中，要继续研究深度学习模型来解决此任务，尝试多种深度学习模型，例如：CNN、RNN、LSTM、Seq2Seq、FastText、TextCNN 等。另外还可以尝试自然语言处理中的经典模型，例如 Bert、XLNet。

4.3 心得体会

在完成这次项目的过程中，我们打开了学习自然语言处理的大门。自然语言处理首先要理解自然语言，之后生成自然语言。我们这次三级项目中主要使用到的就是自然语言理解，理解评论的语意属于哪一类。在这里我们对于 NLP 预处理的分词、去停用词以及将所有词加入词典有了更加理论上以及实践上的认知，首先将所有词提取出来之后去掉对句意来说无关紧要的词并将所有词加入词典。之后我们对于 NLP 模型构建和模型训练等也有了更加深刻的认识，使用词典中的数字表示评论中的单词，之后使用构建的模型来对这些数字进行训练，根据拟定的算法会得到每一个评论的类别。在这次三级项目中，我们感觉收获颇丰！不仅仅了解了自然语言处理的基本过程，同时对于 NLP 预处理的理论及实践提升了认知，也更加锻炼了小组合作能力。

5 小组分工

	学号	姓名	自评成绩	所做工作
小组成员	201811040809	乔翱	A	AdaBoost 算法的手动实现，特征工程，部分报告书写，PPT 制作
	201811040807	李华宪	A	Bagging 算法的手动实现，部分报告书写
	201811040810	王紫晔	B	数据预处理，模型训练及评估，部分报告书写
	201811040808	刘祺	B	模型训练及评估，部分报告书写，PPT 制作

封面设计： 贾丽

地 址：中国河北省秦皇岛市河北大街 438 号
邮 编：066004
电 话：0335-8057068
传 真：0335-8057068
网 址：http://jwc.ysu.edu.cn