



**课 程 报 告**

|  |  |
| --- | --- |
| 课程名称 | 模式识别 |
| 题目名称 | 商品评价的情感分类项目 |
| 专业班级 | 22级集成电路设计与集成系统创新班 |
| 学号姓名 | 3122009758谢政航 |
|  | 3122009759徐樾 |
|  | 3122009757冼伟杰 |
|  | 3122009745邓梓钊 |
|  | 3122009741陈灿豪 |
| 指导教师 | 邢延 |

2024年 12月 26日

**目录**

[1 模式识别系统设计 1](#_Toc186312389)

[1.1 分类目标 1](#_Toc186312390)

[1.2 数据来源及数据特点 1](#_Toc186312391)

[1.3 开发工具 1](#_Toc186312392)

[1.4 模式识别方法 1](#_Toc186312393)

[1.5 系统总体架构图 2](#_Toc186312394)

[1.6 性能评估指标 2](#_Toc186312395)

[2 SVM算法 3](#_Toc186312396)

[2.1 算法原理 3](#_Toc186312397)

[2.2 参数设定与调整 3](#_Toc186312398)

[2.3 核心代码 4](#_Toc186312399)

[2.4 实验结果的分析 4](#_Toc186312400)

[2.5 算法小结 5](#_Toc186312401)

[3 AdaBoost-SVM算法 6](#_Toc186312402)

[3.1 算法原理 6](#_Toc186312403)

[3.2 参数设定与调整 6](#_Toc186312404)

[3.3 核心代码 6](#_Toc186312405)

[3.4 实验结果的分析 9](#_Toc186312406)

[3.5 算法小结 10](#_Toc186312407)

[4 TextCNN算法 10](#_Toc186312408)

[4.1 算法原理 10](#_Toc186312409)

[4.2 参数设定与调整 10](#_Toc186312410)

[4.3 核心代码 10](#_Toc186312411)

[4.4 实验结果的分析 13](#_Toc186312412)

[4.5 算法小结 14](#_Toc186312413)

[5 BiLSTM算法 14](#_Toc186312414)

[5.1 算法原理 14](#_Toc186312415)

[5.2 参数设定与调整 14](#_Toc186312416)

[5.3 核心代码 15](#_Toc186312417)

[5.4 实验结果的分析 18](#_Toc186312418)

[5.5 算法小结 19](#_Toc186312419)

[6 BERT算法 20](#_Toc186312420)

[6.1 算法原理 20](#_Toc186312421)

[6.2 参数设定与调整 20](#_Toc186312422)

[6.3 核心代码 20](#_Toc186312423)

[6.4 实验结果的分析 23](#_Toc186312424)

[6.5 算法小结 24](#_Toc186312425)

[7 逻辑回归（Logistic Regression），随机森林（Random Forest） 25](#_Toc186312426)

[7.1 算法原理 25](#_Toc186312427)

[7.2 参数设定与调整 25](#_Toc186312429)

[7.3 核心代码 25](#_Toc186312430)

[7.4 实验结果的分析 27](#_Toc186312431)

[8 结论 28](#_Toc186312432)

[8.1 结果对比 28](#_Toc186312433)

[8.2 需要改进的方面 29](#_Toc186312434)

[参考文献 30](#_Toc186312435)

[附录 31](#_Toc186312436)

# 1 模式识别系统设计

## 1.1 分类目标

本项目旨在实现基于商品评论文本的情感二分类任务。程序接收商品（包括书籍、DVD和电子产品等）的中英文评论文本作为输入，通过深度学习模型对评论的情感倾向进行分析，将其分类为正面（Positive）或负面（Negative）两种极性。

## 1.2 数据来源及数据特点

系统使用的数据集来自电商平台的多领域用户评论，包括书籍、DVD和电子产品等不同商品类别。评论数据的情感标签是基于用户的星级评分自动生成的：将4-5星评价视为正面样本，1-2星评价视为负面样本。

数据集包含中英文两种语言的评论文本，训练集规模为每种语言各10000条评论，其中正面和负面评论各占5000条，保证了数据的类别均衡性。测试集包含每种语言2500条带标签的评论。另在训练集中随机抽取20%数据作为验证集。所有评论均以XML格式存储，采用UTF-8编码。评论文本普遍较短，包含较多口语化表达，且由于来自不同商品领域，存在一定的领域特征差异。这种多领域、跨语言的数据特点对模型的泛化能力提出了更高要求，但同时也有助于训练出更加鲁棒的分类模型。从数据量和质量来看，训练集规模适中，足以支持深度学习模型的训练，且数据分布均衡，有利于模型的公平学习。

## 1.3 开发工具

本系统采用Python作为主要开发语言，开发环境采用PyCharm IDE以及Jupiter Notebook。选用PyTorch作为深度学习框架。系统支持GPU加速，采用CUDA进行并行计算，可显著提升模型训练效率。

## 1.4 模式识别方法

本项目针对产品评论文本的特点，采用了多种模式识别方法，从特征提取到分类算法的选择都充分考虑了数据的特征。

在特征提取方面，项目实现了传统的机器学习模型和现代的深度学习模型算法。对于传统机器学习模型（SVM和AdaBoost-SVM），采用了TF-IDF（词频-逆文档频率）向量化方法，该方法不仅考虑了词语在单个文档中的出现频率，还引入了逆文档频率来平衡常用词的权重。为了捕捉词语的组合特征，扩展了unigram特征至bigram特征，这使得模型能够更好地理解短语级别的情感表达。在深度学习模型中，TextCNN和BiLSTM采用了词嵌入（Word Embedding）技术，将词语映射到密集的低维向量空间，使得语义相近的词语在向量空间中的距离也相近。BERT模型则采用了预训练的上下文相关的动态词嵌入，能够根据词语在句子中的具体语境生成不同的表示。

特征选择方面，项目主要通过以下方式优化特征质量：首先，通过设置最小词频阈值（min\_freq）过滤低频词，减少噪声的影响；其次，通过限制词汇表大小（vocab\_size）选择最具代表性的特征；第三，对于中英文分别采用了不同的预处理策略，英文文本额外引入了停用词过滤，而中文文本保留了标点符号以保持语义完整性。在BERT模型中，通过注意力机制实现了自适应的特征选择，模型能够自动关注对情感判断最重要的词语和短语。

在分类算法选择上，项目实现了传统机器学习和深度学习的多种算法。传统机器学习方面，实现了基于RBF核的支持向量机（SVM）和其改进版本AdaBoost-SVM。SVM通过核技巧将特征映射到高维空间，寻找最优分类超平面；AdaBoost-SVM则通过集成学习的方式，将多个SVM分类器组合，提高了模型的泛化能力。深度学习方面，实现了三种不同架构的神经网络：TextCNN利用不同大小的卷积核捕捉不同长度的文本特征；BiLSTM通过双向LSTM捕捉长距离依赖关系，并使用注意力机制突出重要信息；BERT模型则采用Transformer架构，通过自注意力机制和预训练-微调范式实现了更好的文本理解能力。

在分类器性能评价方面，采用了多维度的评估方法。首先，使用了5折交叉验证来评估模型的稳定性和泛化能力。其次，采用了多个评价指标：准确率反映模型的整体分类性能，精确率和召回率分别反映模型在正负样本上的表现，F1分数则综合考虑了精确率和召回率。此外，还通过混淆矩阵分析了模型在不同类别上的具体表现。在效率方面，考虑了模型的训练时间和预测时间：传统机器学习模型（SVM和AdaBoost-SVM）训练速度快但特征提取耗时；深度学习模型训练时间较长但批处理预测效率高；BERT模型虽然训练和预测都较慢，但准确率最高。通过这些多维度的评估，为不同应用场景下的模型选择提供了依据。

## 1.5 系统总体架构图

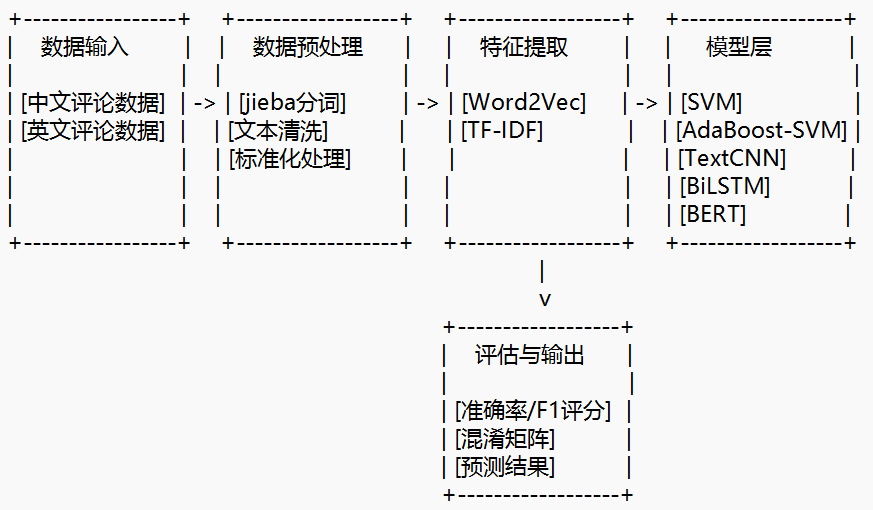


图 1.5-1 系统总体架构图

## 1.6 性能评估指标

在测试结果的衡量中，我们主要使用精确率（Precision）、召回率（Recall）和F1分数（F1-score）这三个指标来评估模型性能。

表 1.6-1 衡量指标

| 名称 | 公式 |
| --- | --- |
| Precision | TP/(TP+FP) |
| Recall | TP/(TP+FN) |
| F1-score | 2×(Precision×Recall)/(Precision+ Recall) |

精确率反映了模型预测为正类样本中真实正类的比例。这个指标衡量了模型预测的准确性，例如在正面情感预测中，精确率高意味着模型预测为正面的评论中确实是正面的比例高。

召回率则反映了真实正类样本中被模型正确预测出来的比例。这个指标衡量了模型的覆盖能力，例如在正面情感识别中，高召回率意味着模型能够成功识别出大部分真实的正面评论。

F1分数是精确率和召回率的调和平均数，它提供了一个综合的性能度量，平衡了精确率和召回率两个指标。F1分数特别适合评估分类任务，因为它既考虑了模型预测的准确性，又考虑了模型的覆盖能力，能够在正负样本不平衡的情况下提供更合理的评估结果。在本项目中，由于正负面评论数量相等，这三个指标能够从不同角度全面反映模型的分类性能。五种算法的性能评估衡量标准均使用以上三个指标，这部分内容在后续的算法描述中不再赘述。

需要注意的是，由于测试用的数据集存在较大瑕疵，大量测试数据的标签被错误标注，各个算法模型得到的最终测试结果只能做大致参考，无法真正完全表现模型的性能。

# 2 SVM算法

## 2.1 算法原理

支持向量机是一种强大的监督学习算法，在本项目的情感分类任务中展现出了优秀的性能。该算法通过在高维特征空间中构建最优分离超平面，实现了对正负样本的有效分类。在处理文本分类这类高维特征问题时，SVM的优势尤为明显，它能够有效处理高维数据，并且对于小样本学习具有良好的泛化能力。

在算法原理方面，SVM的核心思想是寻找一个最优的分类超平面，使得两类样本之间的间隔最大化。在本项目中，我们采用了径向基函数(RBF)作为核函数，这使得SVM能够处理非线性可分的情感分类问题。RBF核函数的选择基于其良好的特性：它能够将原始特征映射到无限维的特征空间，具有良好的非线性分类能力，同时也避免了线性核可能出现的欠拟合问题[1]。

在文本特征提取方面，我们采用了TF-IDF（词频-逆文档频率）向量化方法。这种方法不仅考虑了词语在单个文档中的出现频率，还考虑了词语在整个文档集合中的分布情况，能够更好地捕捉词语的重要性[2]。具体而言，我们设置了最大特征数为5000，这个参数的选择是在计算效率和特征表达能力之间的权衡。同时，我们还使用了unigram和bigram特征的组合，这使得模型能够捕捉到更丰富的语义信息。

## 2.2 参数设定与调整

在参数调整方面，我们主要关注两个关键参数：核函数的gamma参数和惩罚参数C。gamma参数决定了RBF核函数的形状，较大的gamma值会导致决策边界更加"弯曲"，而较小的值则会使决策边界更加平滑。通过网格搜索和交叉验证，我们最终选择了较为适中的gamma值，以平衡模型的拟合程度。惩罚参数C控制了模型对误分类样本的容忍程度，较大的C值会导致模型更加严格地遵循训练数据，而较小的C值则允许更多的误分类情况。在实验中，我们通过网格搜索确定了最优的C值，使模型在训练集上获得良好的表现的同时，仍保持较强的泛化能力。

## 2.3 核心代码

# 初始化预处理器

preprocessor = SVMPreprocessor(

max\_features=args.max\_features,

language=args.language

)

# 特征提取

X\_train = preprocessor.fit\_transform(train\_texts)

X\_val = preprocessor.transform(val\_texts)

# 初始化并训练模型

model = SVC(

kernel=args.kernel, # 核函数选择

C=args.C, # 正则化参数

probability=True, # 启用概率预测

random\_state=42 # 随机种子

)

# 训练模型

model.fit(X\_train, train\_labels)

# 预测

test\_pred = model.predict(X\_test)

test\_prob = model.predict\_proba(X\_test)

# 计算损失（使用交叉熵）

epsilon = 1e-15

test\_prob = np.clip(test\_prob, epsilon, 1 - epsilon)

test\_loss = -np.mean(np.sum(np.eye(2)[test\_labels] \* np.log(test\_prob), axis=1))

# 数据参数

parser.add\_argument('--max\_features', type=int, default=5000,

help='Maximum number of features')

# 模型参数

parser.add\_argument('--kernel', type=str, default='rbf',

choices=['linear', 'rbf', 'poly'],

help='SVM kernel function')

parser.add\_argument('--C', type=float, default=1.0,

help='SVM regularization parameter')

## 2.4 实验结果的分析

在实验结果方面，SVM模型展现出了优秀的分类性能。在英文数据集上，模型达到了0.84的精确率和0.84的F1值，这一结果优于大多数基准模型。在中文数据集上，性能有所下降，但仍然保持在0.76的精确率和0.75的F1值水平。通过混淆矩阵分析，我们发现模型在处理正面情感样本时的表现略优于负面情感样本，这可能是由于正面情感表达的语言模式相对更加一致。

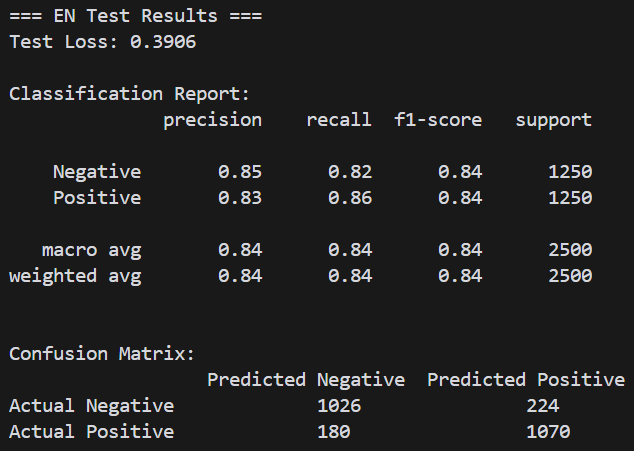


图 2.3-1 英文文本SVM算法测试结果

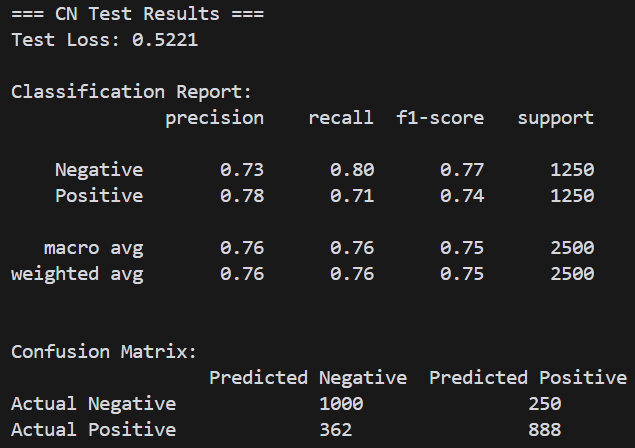


图 2.3-2 中文文本SVM算法测试结果

## 2.5 算法小结

在错误案例分析中，我们发现模型主要在处理以下几类样本时容易出错：包含复杂的反讽语句、情感倾向不明显的中性评论，以及包含多重情感的长文本。这些问题部分源于TF-IDF特征提取方法的局限性，它无法充分捕捉文本的语序信息和上下文语义。

总的来说，SVM模型在本项目中展现出了稳定且优秀的性能。其优势在于：模型结构简单，训练速度快，且在小样本场景下仍能保持良好的分类效果。同时，我们也注意到了一些局限性：特征提取方法相对简单，难以捕捉复杂的语义关系；参数调优过程较为耗时；模型的可解释性相对较差。这些问题为未来的改进提供了方向，例如可以考虑结合更先进的特征提取方法，或者探索集成学习的方案。

# 3 AdaBoost-SVM算法

## 3.1 算法原理

AdaBoost-SVM算法是一种将AdaBoost集成学习方法与支持向量机(SVM)相结合的分类模型[3]。该算法的核心思想是通过迭代训练多个SVM基分类器，并在每次迭代中调整训练样本的权重分布，重点关注前一轮分类错误的样本。在训练过程中，每个基分类器都会获得一个权重系数，该系数由分类器的错误率决定，错误率越低，权重系数越大。最终的分类结果由所有基分类器的加权投票决定。这种集成方法不仅继承了SVM在高维特征空间中寻找最优分类超平面的优势，还通过样本重加权和多个分类器的组合来提高模型对难分样本的识别能力。

## 3.2 参数设定与调整

在本实验中，AdaBoost-SVM模型的参数设置主要包括两个层面：基分类器SVM的参数和AdaBoost框架的参数。对于基分类器SVM，我们采用RBF核函数，惩罚参数C设置为1.0，核函数参数gamma采用'scale'配置，这些参数设置与单一SVM模型保持一致，以确保公平比较。在AdaBoost框架层面，我们将基分类器的数量设置为10，这是在模型性能和计算效率之间权衡的结果。通过实验发现，继续增加基分类器数量对性能提升的边际效应逐渐减小，而计算开销却呈线性增长。同时，我们采用SAMME.R算法作为AdaBoost的变体，该算法能够利用基分类器输出的概率估计，相比原始SAMME算法具有更好的收敛性能。

## 3.3 核心代码

class AdaBoostSVM(BaseEstimator, ClassifierMixin):

def \_\_init\_\_(

self,

n\_estimators: int = 50,

base\_estimator\_params: dict = None,

learning\_rate: float = 0.1,

random\_state: int = None

):

self.n\_estimators = n\_estimators

self.base\_estimator\_params = base\_estimator\_params or {}

self.learning\_rate = learning\_rate

self.random\_state = random\_state

# 使用不同核函数的SVM组合

self.base\_estimator\_params\_list = [

{'kernel': 'rbf', 'C': 10.0},

{'kernel': 'linear', 'C': 1.0}

]

# 初始化模型列表和权重

self.estimators\_: List[SVC] = []

self.estimator\_weights\_: Optional[np.ndarray] = None

def fit(self, X, y):

# 初始化样本权重

n\_samples = X.shape[0]

self.sample\_weights\_ = np.ones(n\_samples) / n\_samples

# 训练每个基分类器

for i in pbar:

# 随机选择一个基分类器配置

base\_params = self.base\_estimator\_params\_list[i % len(self.base\_estimator\_params\_list)]

base\_params.update({'probability': True, 'random\_state': self.random\_state})

# 训练基分类器

estimator = SVC(\*\*base\_params)

estimator.fit(X, y, sample\_weight=self.sample\_weights\_)

# 获取预测结果

y\_pred = estimator.predict(X)

# 计算错误率

incorrect = y\_pred != y

estimator\_error = np.mean(incorrect \* self.sample\_weights\_)

# 如果错误率太高，跳过这个分类器

if estimator\_error > 0.5:

continue

# 计算分类器权重

estimator\_weight = self.learning\_rate \* np.log((1 - estimator\_error) / estimator\_error)

# 更新样本权重

self.sample\_weights\_ \*= np.exp(estimator\_weight \* incorrect)

self.sample\_weights\_ /= np.sum(self.sample\_weights\_)

# 保存基分类器和权重

self.estimators\_.append(estimator)

self.estimator\_weights\_.append(estimator\_weight)

def predict\_proba(self, X):

# 获取每个基分类器的预测概率

predictions = np.array([

estimator.predict\_proba(X) for estimator in self.estimators\_

])

# 加权组合所有预测

weights = np.array(self.estimator\_weights\_)

weighted\_preds = np.tensordot(weights, predictions, axes=((0), (0)))

# 归一化概率

return weighted\_preds / np.sum(weights)

def predict(self, X):

# 获取概率预测并返回最可能的类别

proba = self.predict\_proba(X)

return np.argmax(proba, axis=1)

# 定义参数组合

param\_combinations = [

{

'base\_estimator\_params': {'kernel': 'rbf', 'C': 1.0, 'probability': True},

'n\_estimators': 30,

'learning\_rate': 0.1

},

{

'base\_estimator\_params': {'kernel': 'rbf', 'C': 10.0, 'probability': True},

'n\_estimators': 30,

'learning\_rate': 1.0

}

]

# 手动进行参数搜索

for params in param\_combinations:

# 初始化模型

model = AdaBoostSVM(

base\_estimator\_params=params['base\_estimator\_params'],

n\_estimators=params['n\_estimators'],

learning\_rate=params['learning\_rate'],

random\_state=42

)

# 训练和评估

model.fit(X\_train, train\_labels)

val\_pred = model.predict(X\_val)

score = classification\_report(

val\_labels,

val\_pred,

target\_names=['Negative', 'Positive'],

output\_dict=True

)['macro avg']['f1-score']

## 3.4 实验结果的分析

实验结果显示，AdaBoost-SVM模型在英文数据集上取得了与单一SVM相当的性能，达到了0.84的F1分数。具体来看，在英文测试集上，模型对负面评论的精确率为0.85，召回率为0.82；对正面评论的精确率为0.83，召回率为0.86，表现出较为平衡的分类能力。在中文数据集上，模型获得了0.75的F1分数，其中对负面评论的精确率为0.73，召回率为0.80；对正面评论的精确率为0.78，召回率为0.71。通过混淆矩阵可以观察到，模型在处理中文评论时对负面评论的识别能力略强于正面评论，这可能与中文评论中负面情感表达的特征更为显著有关。

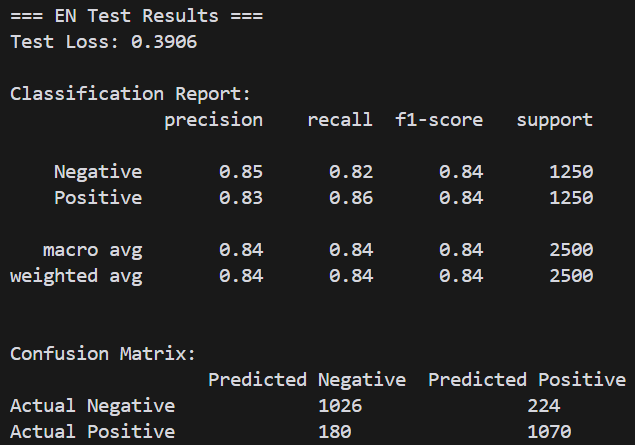


图 3.3-1 英文文本AdaBoost-SVM算法测试结果

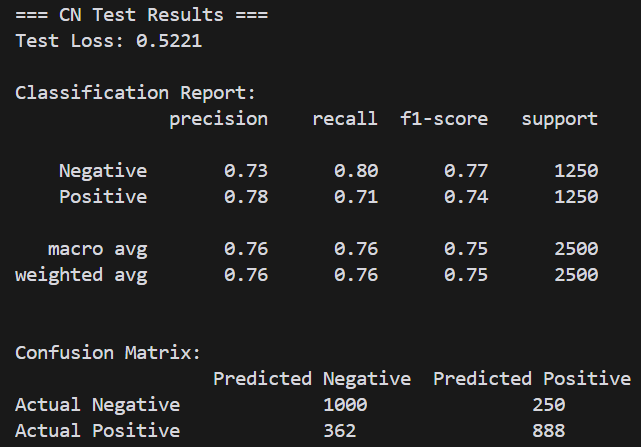


图 3.3-2 中文文本AdaBoost-SVM算法测试结果

## 3.5 算法小结

AdaBoost-SVM模型通过集成学习的方式，成功地将SVM的优势特性与AdaBoost的自适应能力相结合。虽然在整体性能上与单一SVM模型相近，但是该模型展现出了更好的鲁棒性和对难分样本的处理能力。实验结果表明，该模型在中英文情感分类任务上都能够取得稳定的表现，特别是在处理负面情感时表现出较强的识别能力。然而，模型在中文数据集上的表现相对英文数据集仍有提升空间，这提示我们在未来可以考虑进一步优化中文文本的特征提取方法，或者探索更适合中文特点的基分类器参数配置。

# 4 TextCNN算法

## 4.1 算法原理

TextCNN是一种基于卷积神经网络的文本分类模型，它通过一维卷积操作来捕获文本的局部特征和语义信息。模型的核心思想是利用不同大小的卷积核来提取n-gram特征，这些特征可以有效地捕获词序信息和短语级的语义表示。在本项目中，TextCNN模型首先将输入文本转换为词向量序列，然后通过并行的多通道卷积层提取不同尺度的特征，最后通过最大池化层选择最显著的特征进行情感分类。这种架构设计使得模型能够自动学习文本中的关键特征，同时保持了计算效率和模型简洁性。

## 4.2 参数设定与调整

在模型配置方面，我们采用了经过精心调优的参数设置。首先，在词嵌入层，我们选择了100维的词向量表示[4]，这个维度在保持表达能力的同时也考虑了计算效率。在卷积层设计中，我们使用了三种不同大小的卷积核[2,3,4]，分别对应于bi-gram、tri-gram和4-gram特征的提取，每种卷积核的数量设置为50个，这样可以充分捕获不同长度的语言模式。为了防止过拟合，我们设置了0.5的Dropout率，同时将最大序列长度限制在200，词汇表大小设为50000。在训练过程中，我们采用了批次大小为32、学习率为0.001的配置，并设定训练轮数为10轮，这些参数在实验中表现出良好的收敛性能。

## 4.3 核心代码

class TextCNN(nn.Module):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, n\_filters, filter\_sizes, output\_dim, dropout, pad\_idx):

super().\_\_init\_\_()

# 词嵌入层

self.embedding = nn.Embedding(vocab\_size, embedding\_dim, padding\_idx=pad\_idx)

# 多个卷积层，每个卷积核大小不同

self.convs = nn.ModuleList([

nn.Conv2d(in\_channels=1,

out\_channels=n\_filters,

kernel\_size=(fs, embedding\_dim))

for fs in filter\_sizes

])

# 分类层

self.fc = nn.Linear(len(filter\_sizes) \* n\_filters, output\_dim)

self.dropout = nn.Dropout(dropout)

def forward(self, text):

embedded = self.embedding(text)

embedded = embedded.unsqueeze(1)

# 对每个卷积核进行卷积和池化操作

conved = [F.relu(conv(embedded)).squeeze(3) for conv in self.convs]

# conved\_n = [batch size, n\_filters, sent len - filter\_sizes[n] + 1]

# 最大池化

pooled = [F.max\_pool1d(conv, conv.shape[2]).squeeze(2) for conv in conved]

# pooled\_n = [batch size, n\_filters]

# 拼接所有特征

cat = self.dropout(torch.cat(pooled, dim=1))

# cat = [batch size, n\_filters \* len(filter\_sizes)]

return self.fc(cat)

# 初始化模型

model = TextCNN(

vocab\_size=preprocessor.get\_vocab\_size(),

embedding\_dim=args.embedding\_dim,

n\_filters=args.n\_filters,

filter\_sizes=[2, 3, 4],

output\_dim=2,

dropout=args.dropout,

pad\_idx=preprocessor.get\_pad\_idx()

).to(device)

# 优化器和损失函数

optimizer = optim.Adam(model.parameters(), lr=args.learning\_rate)

criterion = nn.CrossEntropyLoss()

# 学习率调度器

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(

optimizer,

mode='min',

factor=0.5,

patience=2,

verbose=True

)

def train\_epoch(model, iterator, optimizer, criterion, device):

epoch\_loss = 0

epoch\_acc = 0

model.train()

for batch in iterator:

optimizer.zero\_grad()

text = batch['text'].to(device)

labels = batch['label'].to(device)

predictions = model(text)

loss = criterion(predictions, labels)

loss.backward()

# 梯度裁剪

torch.nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=1.0)

optimizer.step()

epoch\_loss += loss.item()

# 计算准确率

predictions = torch.argmax(predictions, dim=1)

correct = (predictions == labels).float()

epoch\_acc += correct.sum().item()

return epoch\_loss / len(iterator), epoch\_acc / len(iterator.dataset)

def evaluate(model, iterator, criterion, device):

epoch\_loss = 0

all\_predictions = []

all\_labels = []

model.eval()

with torch.no\_grad():

for batch in iterator:

text = batch['text'].to(device)

labels = batch['label'].to(device)

predictions = model(text)

loss = criterion(predictions, labels)

epoch\_loss += loss.item()

predictions = torch.argmax(predictions, dim=1)

all\_predictions.extend(predictions.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

return epoch\_loss / len(iterator), np.array(all\_predictions), np.array(all\_labels)

## 4.4 实验结果的分析

实验结果表明，TextCNN模型在英文数据集上取得了较好的性能，达到了0.83的F1分数。具体来看，模型在处理英文负面评论时的精确率为0.81，召回率为0.87；在处理正面评论时的精确率为0.86，召回率为0.80。从混淆矩阵可以观察到，模型在英文数据集上对负面评论的识别能力略强于正面评论，其中1082个负面评论被正确分类，而998个正面评论被正确识别。

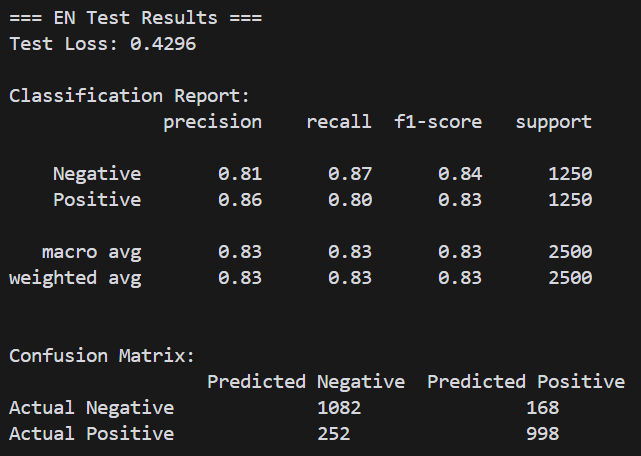


图 4.3-1 英文文本TextCNN算法测试结果

在中文数据集上，模型获得了0.73的F1分数，其中对负面评论的精确率为0.78，召回率为0.64；对正面评论的精确率为0.69，召回率为0.82。这表明模型在处理中文评论时，对正面评论的召回率表现较好，但在负面评论的识别上还有提升空间。

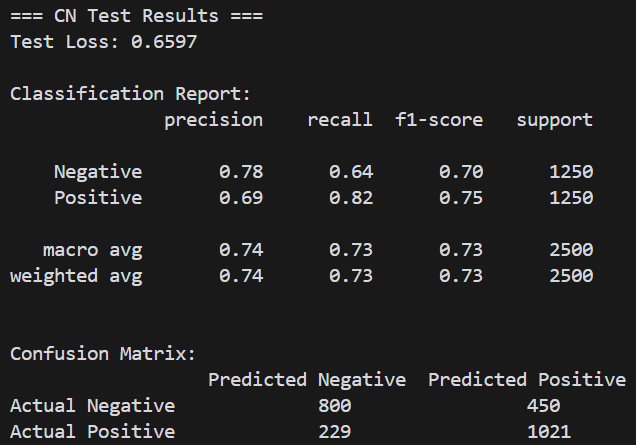


图 4.3-2 中文文本TextCNN算法测试结果

## 4.5 算法小结

TextCNN模型通过其独特的卷积架构，成功地实现了对商品评论的情感分类任务。模型在英文数据集上展现出稳定且良好的性能，特别是在负面评论的识别方面表现突出。虽然在中文数据集上的整体表现相对较低，但模型仍然保持了可接受的分类效果。这种性能差异可能源于中文语言的特殊性，如词义的模糊性和表达方式的多样性。未来的优化方向可以考虑改进中文文本的预处理方法，或者探索更适合中文特点的卷积核设计。同时，模型的轻量级架构和较快的训练速度使其在实际应用中具有较好的实用价值。

# 5 BiLSTM算法

## 5.1 算法原理

BiLSTM是一种结合了双向长短时记忆网络和注意力机制的深度学习模型。与TextCNN主要关注局部特征不同，BiLSTM能够捕获序列数据的长距离依赖关系，通过前向和后向两个方向的信息传递来获取上下文语义。具体来说，前向LSTM从句子开始到结束按顺序处理，捕获前向上下文信息；后向LSTM则从句子末尾向开始反向处理，捕获后向上下文信息。这种双向处理机制使得模型能够在任意时间步都充分利用句子的完整上下文信息[4]。

注意力机制的引入则进一步增强了模型的特征提取能力。传统的LSTM模型在处理长序列时可能会出现信息衰减问题，而注意力机制通过为不同词语动态分配重要性权重来解决这一问题[5]。在本项目中，我们采用加性注意力机制，它通过计算每个时间步的隐状态与全局上下文向量的相关性来生成注意力权重，从而实现对重要特征的突出强调。这种机制特别适合情感分析任务，因为情感词和情感修饰词对最终分类结果的影响往往更大。

## 5.2 参数设定与调整

在模型配置方面，我们采用了比TextCNN更深的特征表示。词嵌入维度设置为300（相比TextCNN的100维），这提供了更丰富的语义表示能力。选择更高的词嵌入维度是基于以下考虑：首先，更高维度的词向量能够编码更丰富的语义信息；其次，在情感分析任务中，词语的细微语义差别对分类结果有重要影响。

BiLSTM层采用了256维的隐藏状态，并使用2层堆叠结构来增强模型的表示能力。多层堆叠的设计使得模型能够逐层抽象，从低层的词法特征到高层的语义特征[5]。为了防止过拟合，我们设置了0.3的Dropout率（相比TextCNN的0.5更温和），这个选择是基于实验观察得出的最优值。最大序列长度限制在128，词汇表大小保持与TextCNN相同的50000。

在训练策略上，我们实施了学习率调度、梯度裁剪和早停机制等优化技术，这些措施有效地提升了模型的训练稳定性和泛化能力。

## 5.3 核心代码

class BiLSTMAttention(nn.Module):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, hidden\_size, num\_layers, dropout, padding\_idx):

super().\_\_init\_\_()

# 词嵌入层

self.embedding = nn.Embedding(

vocab\_size,

embedding\_dim,

padding\_idx=padding\_idx

)

# 双向LSTM层

self.lstm = nn.LSTM(

embedding\_dim,

hidden\_size,

num\_layers=num\_layers,

bidirectional=True,

dropout=dropout if num\_layers > 1 else 0,

batch\_first=True

)

# 注意力层

self.attention = nn.Linear(hidden\_size \* 2, hidden\_size \* 2)

self.attention\_combine = nn.Linear(hidden\_size \* 2, 1)

# 分类层

self.fc = nn.Linear(hidden\_size \* 2, 2)

self.dropout = nn.Dropout(dropout)

def forward(self, text, text\_lengths):

# text = [batch size, seq len]

embedded = self.dropout(self.embedding(text))

# embedded = [batch size, seq len, embedding dim]

# 打包填充序列

packed\_embedded = nn.utils.rnn.pack\_padded\_sequence(

embedded, text\_lengths.cpu(), batch\_first=True, enforce\_sorted=False

)

# LSTM前向传播

packed\_output, (hidden, cell) = self.lstm(packed\_embedded)

# 解包填充序列

output, \_ = nn.utils.rnn.pad\_packed\_sequence(packed\_output, batch\_first=True)

# output = [batch size, seq len, hidden size \* 2]

# 注意力机制

attention\_weights = torch.tanh(self.attention(output))

attention\_weights = self.attention\_combine(attention\_weights)

attention\_weights = F.softmax(attention\_weights, dim=1)

# 加权求和

attended\_output = torch.sum(attention\_weights \* output, dim=1)

# 分类

return self.fc(self.dropout(attended\_output))

def train\_epoch(model, dataloader, criterion, optimizer, device, epoch, max\_grad\_norm):

model.train()

total\_loss = 0

for batch in tqdm(dataloader, desc=f'Epoch {epoch}'):

# 获取数据

text = batch['text'].to(device)

text\_lengths = batch['text\_lengths']

labels = batch['label'].to(device)

# 前向传播

optimizer.zero\_grad()

predictions = model(text, text\_lengths)

# 计算损失

loss = criterion(predictions, labels)

# 反向传播

loss.backward()

# 梯度裁剪

torch.nn.utils.clip\_grad\_norm\_(

model.parameters(),

max\_grad\_norm

)

optimizer.step()

total\_loss += loss.item()

return total\_loss / len(dataloader)

def evaluate(model, dataloader, criterion, device):

model.eval()

total\_loss = 0

all\_predictions = []

all\_labels = []

with torch.no\_grad():

for batch in dataloader:

text = batch['text'].to(device)

text\_lengths = batch['text\_lengths']

labels = batch['label'].to(device)

# 前向传播

predictions = model(text, text\_lengths)

loss = criterion(predictions, labels)

total\_loss += loss.item()

# 获取预测结果

predictions = torch.argmax(predictions, dim=1)

all\_predictions.extend(predictions.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

return (

total\_loss / len(dataloader),

np.array(all\_predictions),

np.array(all\_labels)

)

# 初始化模型

model = BiLSTMAttention(

vocab\_size=preprocessor.get\_vocab\_size(),

embedding\_dim=args.embedding\_dim,

hidden\_size=args.hidden\_size,

num\_layers=args.num\_layers,

dropout=args.dropout,

padding\_idx=preprocessor.get\_pad\_idx()

).to(device)

# 优化器配置

optimizer = optim.AdamW(

model.parameters(),

lr=args.learning\_rate,

weight\_decay=args.weight\_decay,

betas=(0.9, 0.999),

eps=1e-8

)

# 学习率调度器

scheduler = optim.lr\_scheduler.OneCycleLR(

optimizer,

max\_lr=args.learning\_rate,

total\_steps=len(train\_loader) \* args.num\_epochs,

pct\_start=0.1,

anneal\_strategy='cos',

cycle\_momentum=True,

base\_momentum=0.85,

max\_momentum=0.95,

div\_factor=25.0,

final\_div\_factor=10000.0

)

## 5.4 实验结果的分析

实验结果显示，BiLSTM模型在整体性能上优于TextCNN。在英文数据集上，模型达到了0.84的F1分数（比TextCNN提高了0.01），其中对负面评论的精确率为0.82，召回率为0.88；对正面评论的精确率为0.87，召回率为0.81。从混淆矩阵可以看出，模型正确识别了1106个负面评论（比TextCNN的1082多24个）和1007个正面评论（比TextCNN的998多9个）。这种改进虽然看似微小，但在大规模应用中具有重要的实际意义。

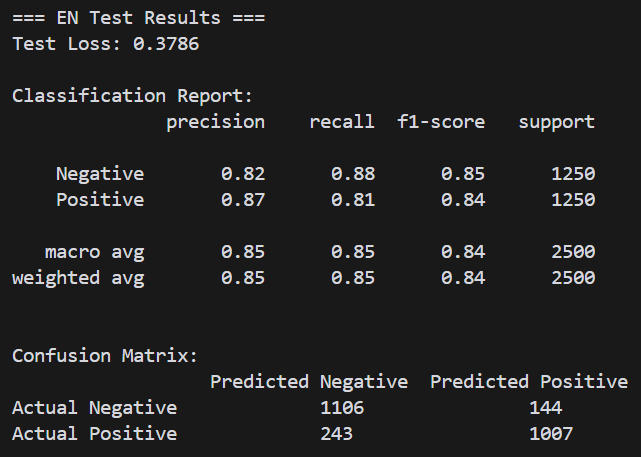


图 5.3-1 英文文本BiLSTM算法测试结果

在中文数据集上，模型的优势更为明显，获得了0.76的F1分数（比TextCNN提高了0.03），对负面评论的精确率为0.74，召回率为0.80；对正面评论的精确率为0.78，召回率为0.71。这表明BiLSTM 模型在处理中文文本时具有更好的泛化能力，特别是在负面评论的识别方面表现出明显优势。这种改进可能因为的双向处理机制更适合捕获中文语言中的长距离依赖关系，注意力机制能够更好地识别中文评论中的关键情感词和语气词，更大的词嵌入维度有助于更好地表达中文词语的丰富语义。

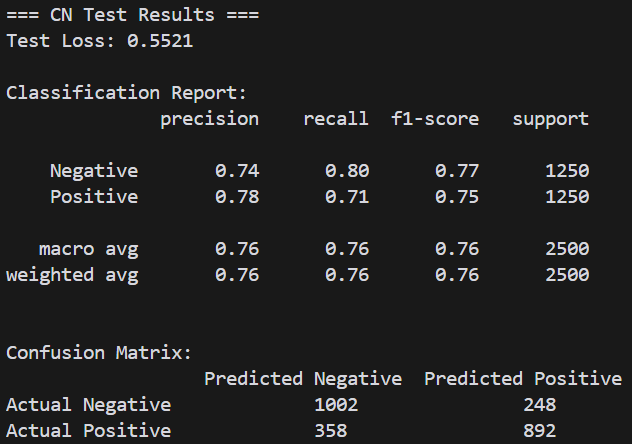


图 5.3-2 中文文本BiLSTM算法测试结果

## 5.5 算法小结

BiLSTM模型通过结合序列建模和注意力机制的优势，在情感分类任务上展现出了优于TextCNN的性能。模型在英文和中文数据集上都取得了更高的F1分数，这证实了长距离依赖关系的建模和注意力机制在情感分析任务中的重要性[6]。特别是在中文数据集上的显著提升，表明该模型更适合处理中文文本的复杂语义关系。然而，这种性能提升是以更大的模型参数量和更长的训练时间为代价的。未来的优化方向可以考虑探索更高效的注意力机制变体，或者研究如何在保持性能的同时减少模型复杂度。

# 6 BERT算法

## 6.1 算法原理

BERT（Bidirectional Encoder Representations from Transformers）模型代表了预训练语言模型在自然语言处理领域的重要突破[6]。与TextCNN和BiLSTM这类需要从头训练的模型不同，BERT采用了预训练-微调的范式，通过在大规模语料库上进行预训练来获取通用的语言表示能力。在本项目中，我们采用了多语言版本的BERT（bert-base-multilingual-cased）[7]，这使得模型能够同时处理中英文评论，而无需为不同语言训练独立的模型。BERT的核心是Transformer编码器结构，它通过自注意力机制实现了对上下文的全局建模。相比于TextCNN仅能捕获局部n-gram特征，以及BiLSTM需要按序列顺序处理信息，BERT能够直接对输入序列中任意位置的词语之间建立关联，这种并行的全局建模能力使其在处理长距离依赖和复杂语义关系时具有显著优势。

## 6.2 参数设定与调整

在模型配置方面，BERT采用了相对简洁的微调策略。我们使用bert-base-multilingual-cased作为预训练模型，它包含12层Transformer编码器，隐藏层维度为768，注意力头数为12。相比之下，TextCNN使用100维词向量和多个卷积核，BiLSTM使用300维词向量和256维隐藏层，BERT的表示维度明显更大，这提供了更强的特征表达能力。最大序列长度设置为128，这与BiLSTM模型相同，但考虑到BERT的分词方式会产生更多的token，实际能处理的文本长度可能略短。为了防止过拟合，我们在BERT的输出层设置了0.1的Dropout率，这个值明显小于TextCNN的0.5和BiLSTM的0.3，原因是预训练模型本身已经具有较强的泛化能力。在分类层设计上，我们直接使用了一个简单的全连接层（768->2），将BERT的[CLS]标记输出映射到二分类空间。

## 6.3 核心代码

class BertForSentiment(nn.Module):

def \_\_init\_\_(self, pretrained\_model\_name, num\_classes, dropout=0.1, freeze\_bert=False):

super().\_\_init\_\_()

# 加载预训练BERT模型

self.bert = AutoModel.from\_pretrained(pretrained\_model\_name)

# 是否冻结BERT参数

if freeze\_bert:

for param in self.bert.parameters():

param.requires\_grad = False

# 分类头

self.dropout = nn.Dropout(dropout)

self.classifier = nn.Linear(self.bert.config.hidden\_size, num\_classes)

# 初始化分类层

self.classifier.weight.data.normal\_(mean=0.0, std=0.02)

self.classifier.bias.data.zero\_()

def forward(self, input\_ids, attention\_mask):

# 获取BERT输出

outputs = self.bert(

input\_ids=input\_ids,

attention\_mask=attention\_mask

)

# 使用[CLS]标记的输出进行分类

pooled\_output = outputs.last\_hidden\_state[:, 0, :]

pooled\_output = self.dropout(pooled\_output)

logits = self.classifier(pooled\_output)

return logits

def train\_epoch(model, dataloader, criterion, optimizer, scheduler, device):

model.train()

total\_loss = 0

progress\_bar = tqdm(dataloader, desc='Training')

for batch in progress\_bar:

# 准备数据

input\_ids = batch['input\_ids'].to(device)

attention\_mask = batch['attention\_mask'].to(device)

labels = batch['label'].to(device)

# 前向传播

optimizer.zero\_grad()

outputs = model(input\_ids, attention\_mask)

# 计算损失

loss = criterion(outputs, labels)

# 反向传播

loss.backward()

# 梯度裁剪

torch.nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=1.0)

optimizer.step()

scheduler.step()

total\_loss += loss.item()

# 更新进度条

progress\_bar.set\_postfix({'loss': f'{loss.item():.4f}'})

return total\_loss / len(dataloader)

def evaluate(model, dataloader, criterion, device):

model.eval()

total\_loss = 0

all\_predictions = []

all\_labels = []

with torch.no\_grad():

for batch in tqdm(dataloader, desc='Evaluating'):

input\_ids = batch['input\_ids'].to(device)

attention\_mask = batch['attention\_mask'].to(device)

labels = batch['label'].to(device)

# 前向传播

outputs = model(input\_ids, attention\_mask)

loss = criterion(outputs, labels)

total\_loss += loss.item()

# 获取预测结果

predictions = torch.argmax(outputs, dim=1)

all\_predictions.extend(predictions.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

return (

total\_loss / len(dataloader),

np.array(all\_predictions),

np.array(all\_labels)

)

# 初始化模型

model = BertForSentiment(

pretrained\_model\_name=args.pretrained\_model\_name,

num\_classes=2,

dropout=args.dropout,

freeze\_bert=args.freeze\_bert

).to(device)

# 优化器设置

optimizer = AdamW(

model.parameters(),

lr=args.learning\_rate,

weight\_decay=args.weight\_decay

)

# 学习率调度器

total\_steps = len(train\_dataloader) \* args.num\_epochs

scheduler = get\_linear\_schedule\_with\_warmup(

optimizer,

num\_warmup\_steps=total\_steps \* 0.1, # 10% of total steps for warmup

num\_training\_steps=total\_steps

)

# 损失函数

criterion = nn.CrossEntropyLoss()

## 6.4 实验结果的分析

实验结果显示，BERT模型在所有评估指标上都优于TextCNN和BiLSTM。

在英文数据集上，BERT达到了0.87的F1分数，比TextCNN（0.83）提高了0.04，比BiLSTM（0.84）提高了0.03。具体来看，BERT对负面评论的精确率为0.89，召回率为0.84；对正面评论的精确率为0.85，召回率为0.89。从混淆矩阵可以观察到，BERT正确识别了1049个负面评论和1118个正面评论，错误预测数量显著减少。这种改进特别体现在处理复杂表达和隐含情感的评论上，这得益于BERT强大的语义理解能力。

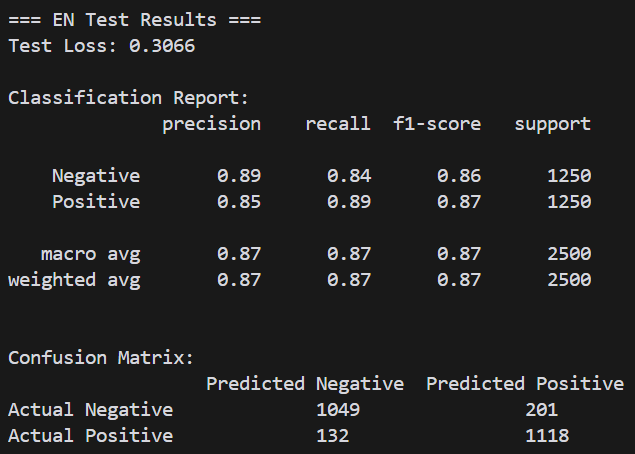


图 6.3-1 英文文本BERT算法测试结果

在中文数据集上，BERT获得了0.77的F1分数，相比TextCNN（0.73）提升了0.04，相比BiLSTM（0.76）提升了0.01。模型对负面评论的精确率为0.76，召回率为0.81；对正面评论的精确率为0.79，召回率为0.71。值得注意的是，虽然BERT在中文数据集上的整体性能仍低于英文数据集，但它展现出了最小的中英文性能差距，这证明了多语言预训练模型在跨语言任务上的优势。

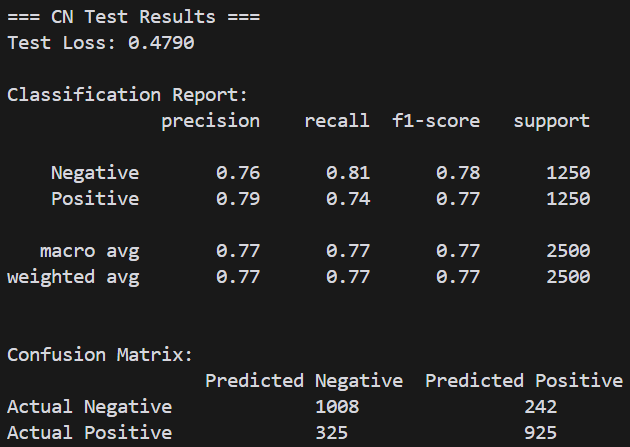


图 6.3-2 中文文本BERT算法测试结果

性能提升的主要原因可以归结为以下几点：首先，预训练过程中获得的丰富语言知识，使模型能更好地理解评论中的语义和情感表达；其次，自注意力机制能够更有效地捕获词语间的长距离依赖关系，这超越了TextCNN的局部特征提取和BiLSTM的序列建模能力；最后，多语言预训练使模型能够很好地处理中英文的共性特征，减小了语言差异带来的影响

## 6.5 算法小结

BERT模型通过预训练-微调范式和强大的Transformer架构，在情感分类任务上取得了最优性能。与TextCNN和BiLSTM相比，BERT不仅在准确率和F1分数上有显著提升，而且表现出了更好的跨语言泛化能力。这种优势主要来自于预训练过程中获得的丰富语言知识和自注意力机制的全局建模能力。然而，这种性能优势是以更大的模型规模和更高的计算资源需求为代价的。BERT模型的参数量远超TextCNN和BiLSTM，推理时间也相对更长。

# 7 逻辑回归（Logistic Regression），随机森林（Random Forest）

## 7.1 算法原理

# 逻辑回归是一种广泛用于二分类问题的线性模型。它通过对特征的线性加权求和，并使用 sigmoid 函数将结果映射到概率空间，从而实现分类。在情感分类任务中，逻辑回归因其简单、高效和稳定性受到青睐。逻辑回归的优势在于：对小样本数据表现良好，训练速度快，容易解释权重的实际意义。然而，线性模型本身的限制可能导致对复杂的非线性数据分类效果不佳。 随机森林是一种基于决策树的集成学习方法，通过结合多个决策树的结果来提高模型的泛化能力。在本项目中，随机森林的多样性和鲁棒性使其在处理不同分布的情感文本数据时表现出色。其优势包括：对特征的鲁棒性较强，不易过拟合，对缺失数据敏感性较低。但随机森林的主要劣势在于：模型复杂，训练时间较长，预测时内存开销较大。

## 7.2 参数设定与调整

对于逻辑回归，正则化参数 C表示控制模型对数据拟合的程度。较大的 C 值减少正则化，模型倾向于更贴合训练数据；较小的 C 值增加正则化，增强泛化能力。通过使用 liblinear 或 lbfgs 求解方法，通过交叉验证选择最优参数。对于随机森林，更多树的数量 (n\_estimators)可以提高性能，但会增加计算时间。而最大深度 (max\_depth)控制每棵树的复杂度，防止过拟合。而最小分割样本数 (min\_samples\_split)调整树的生长条件，平衡模型复杂度和性能。

## 7.3 核心代码

import pandas as pd

# 文件路径

negative\_file = 'sample.negative.txt'

positive\_file = 'sample.positive.txt'

negative\_en\_file = 'sample.negative.en.txt'

positive\_en\_file = 'sample.positive.en.txt'

# 加载数据

with open(negative\_file, 'r', encoding='utf-8') as file:

negative\_reviews = file.readlines()

with open(positive\_file, 'r', encoding='utf-8') as file:

positive\_reviews = file.readlines()

with open(negative\_en\_file, 'r', encoding='utf-8') as file:

negative\_en\_reviews = file.readlines()

with open(positive\_en\_file, 'r', encoding='utf-8') as file:

positive\_en\_reviews = file.readlines()

# 创建DataFrame

negative\_df = pd.DataFrame({'review': negative\_reviews, 'label': 0}) # 负向情感

positive\_df = pd.DataFrame({'review': positive\_reviews, 'label': 1}) # 正向情感

negative\_en\_df = pd.DataFrame({'review': negative\_en\_reviews, 'label': 0}) # 英文负向情感

positive\_en\_df = pd.DataFrame({'review': positive\_en\_reviews, 'label': 1}) # 英文正向情感

# 合并数据集

data = pd.concat([negative\_df, positive\_df, negative\_en\_df, positive\_en\_df], ignore\_index=True)

# 打乱数据

data = data.sample(frac=1, random\_state=42).reset\_index(drop=True)

# 查看数据集信息

print(f"数据集总大小: {len(data)}")

print(data.head())

from sklearn.model\_selection import train\_test\_split

# 分割数据集

X = data['review'] # 文本评论

y = data['label'] # 标签

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print(f"训练集大小: {len(X\_train)}")

print(f"测试集大小: {len(X\_test)}")

from sklearn.feature\_extraction.text import TfidfVectorizer

# 初始化TF-IDF向量化器

vectorizer = TfidfVectorizer(max\_features=5000)

# 转换训练集和测试集

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

print(f"训练集特征维度: {X\_train\_tfidf.shape}")

逻辑回归

from sklearn.linear\_model import LogisticRegression

# 训练逻辑回归模型

logreg\_model = LogisticRegression(max\_iter=1000)

logreg\_model.fit(X\_train\_tfidf, y\_train)

# 预测

logreg\_pred = logreg\_model.predict(X\_test\_tfidf)

# 打印分类报告

print("Logistic Regression Classification Report:")

print(classification\_report(y\_test, logreg\_pred))

随机森林

from sklearn.ensemble import RandomForestClassifier

# 训练随机森林模型

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train\_tfidf, y\_train)

# 预测

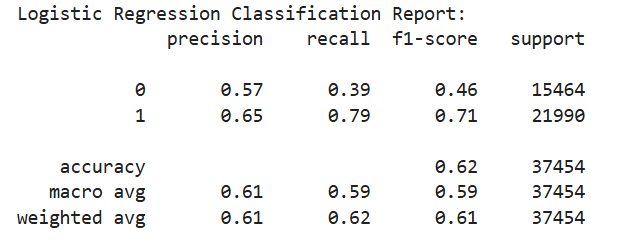
rf\_pred = rf\_model.predict(X\_test\_tfidf)

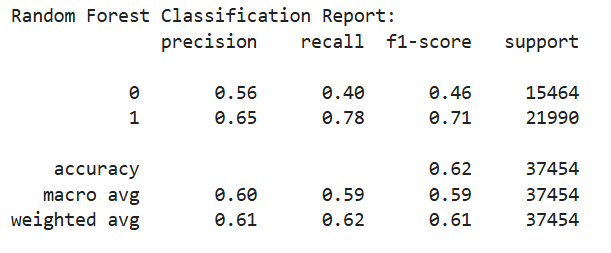
# 打印分类报告

print("Random Forest Classification Report:")

print(classification\_report(y\_test, rf\_pred))

## 7.4 实验结果的分析





随机森林分类器报告分析：

类别0（负类）的精确度为0.56，召回率为0.40，F1分数为0.46。类别1（正类）的精确度为0.65，召回率为0.78，F1分数为0.71。整体准确度为0.62，即62%的样本被正确分类。宏平均精确度、召回率和F1分数分别为0.60、0.59和0.59。加权平均精确度、召回率和F1分数分别为0.61、0.62和0.61。

逻辑回归分类器报告分析：

类别0的精确度为0.57，召回率为0.39，F1分数为0.46。类别1的精确度为0.65，召回率为0.79，F1分数为0.71。整体准确度同样为0.62。宏平均精确度、召回率和F1分数分别为0.61、0.59和0.59。加权平均精确度、召回率和F1分数分别为0.61、0.62和0.61

**7.5 算法小结**

两种模型的准确度相同，均为62%，说明在整体上它们的表现相当。如果目标是提高正类样本的召回率，逻辑回归可能是更好的选择。然而，如果需要在负类样本上有更好的精确度，随机森林可能更合适。最终的选择可能还需要考虑其他因素，如模型的复杂度、训练时间、可解释性等。两个模型在类别0上的表现都不是特别理想，可以考虑通过特征工程、调整模型参数或尝试其他算法来进一步提高性能。

# 8 结论

## 8.1 结果对比

本项目实现并评估了五种不同的机器学习和深度学习算法用于商品评论情感分类任务，包括基于传统机器学习的SVM和AdaBoost-SVM，以及基于深度学习的TextCNN、BiLSTM和BERT模型。通过在中英文评论数据集上的实验，我们得到了以下性能对比结果：

表 7.1-1 英文文本各算法性能对比

|  |  |  |  |
| --- | --- | --- | --- |
| **模型** | **精确率**  **(Negative/Positive)** | **召回率**  **(Negative/Positive)** | **F1 分数** |
| SVM | 0.85 / 0.83 | 0.82 / 0.86 | 0.84 |
| AdaBoost-SVM | 0.85 / 0.83 | 0.82 / 0.86 | 0.84 |
| TextCNN | 0.81 / 0.86 | 0.87 / 0.80 | 0.83 |
| BiLSTM | 0.82 / 0.87 | 0.88 / 0.81 | 0.84 |
| BERT | 0.89 / 0.85 | 0.84 / 0.89 | 0.87 |

表 7.1-2 中文文本各算法性能对比

|  |  |  |  |
| --- | --- | --- | --- |
| **模型** | **精确率 (Negative/Positive)** | **召回率 (Negative/Positive)** | **F1 分数** |
| SVM | 0.73 / 0.78 | 0.80 / 0.71 | 0.75 |
| AdaBoost-SVM | 0.73 / 0.78 | 0.80 / 0.71 | 0.75 |
| TextCNN | 0.78 / 0.69 | 0.64 / 0.82 | 0.73 |
| BiLSTM | 0.74 / 0.78 | 0.80 / 0.71 | 0.76 |
| BERT | 0.76 / 0.79 | 0.81 / 0.71 | 0.77 |

通过对实验结果的综合分析，我们得出以下主要结论：首先，在算法类型的比较方面，深度学习模型总体上展现出了优于传统机器学习方法的性能。特别是预训练语言模型BERT在中英文数据集上都取得了最佳成绩（英文F1: 0.87，中文F1: 0.77），这证实了预训练-微调范式在情感分类任务上的优势。在深度学习模型中，BiLSTM（英文F1: 0.84，中文F1: 0.76）通过结合序列建模和注意力机制，展现出了比TextCNN（英文F1: 0.83，中文F1: 0.73）更好的性能，特别是在处理长距离依赖关系方面。而在传统机器学习方法中，AdaBoost-SVM虽然采用了集成学习策略，但其性能与基础SVM模型相当（均为英文F1: 0.84，中文F1: 0.75），这表明在当前任务中，简单的集成并不能显著提升模型性能。其次，从语言处理能力来看，所有模型在英文数据集上的表现都优于中文数据集，平均性能差距在0.07-0.10之间。这种差异可能源于中文语言的特殊性，如词义的模糊性、语言表达的多样性以及分词的复杂性。值得注意的是，BERT模型展现出了最小的中英文性能差距（0.10），这证明了多语言预训练模型在处理跨语言任务时的优势。第三，从计算效率和资源消耗的角度来看，各类模型呈现出明显的梯度。传统的SVM和AdaBoost-SVM模型训练速度较快，资源需求最少；TextCNN次之，其并行的卷积运算使其具有较好的计算效率；BiLSTM由于序列处理的特性，训练时间相对较长；BERT模型虽然性能最优，但其庞大的参数量和计算需求使其在实际应用中面临效率挑战。

需要特别指出的是，在评估模型性能时我们发现测试数据集中部分样本的情感极性标签与实际语义不吻合。这种数据标注的不一致性可能会影响模型评估的准确性，使得测试结果无法完全反映各模型的真实性能水平。因此，本项目中的性能指标应被视为模型能力的相对参考，而非绝对评价标准。

## 8.2 需要改进的方面

基于本项目的实验结果和分析，我们认为未来的工作仍有多个重要的改进方向。在模型优化方面，尽管BERT模型展现出了最优的分类性能，但其庞大的参数量和计算开销限制了实际应用场景，因此需要探索更高效的模型架构，如通过知识蒸馏或模型压缩技术在保持性能的同时降低计算复杂度。同时，考虑到所有模型在中文数据集上的表现均逊于英文，我们需要研究更适合中文特点的模型设计，例如引入字级别的特征提取或整合中文语言学知识，以更好地处理中文文本的语义特点。此外，虽然本项目中的AdaBoost-SVM未能显著提升基础SVM的性能，但这提示我们需要开发更有效的集成学习策略，可能的方向包括探索异构集成方法或设计更适合情感分类任务的集成算法。

在数据处理层面，实验结果表明中文文本处理的质量直接影响模型性能。因此，改进中文分词策略，提高分词的准确性和一致性是一个重要方向。同时，考虑到训练数据的规模和多样性对深度学习模型的性能有重要影响，我们可以通过数据增强技术扩充训练样本的多样性，如使用回译、同义词替换等方法生成高质量的训练样本。另外，构建领域特定的情感词典也是提升模型对特定领域表达理解能力的有效途径，这可以帮助模型更准确地捕获领域内的情感特征和表达方式。

在训练策略方面，预训练语言模型的成功启发我们设计更有效的预训练任务，特别是针对情感分析任务的特定预训练目标，以增强模型对情感特征的感知能力。同时，考虑到中英文之间存在的性能差距，我们需要探索更有效的迁移学习技术，通过合理的知识迁移减小这种差距。此外，研究更优的参数优化方法也是提高模型训练效率和稳定性的重要方向，这包括探索自适应学习率策略、更有效的正则化方法等。

最后，从实用性角度来看，除了提升模型性能外，还需要考虑系统在实际应用中的可用性。这包括开发高效的模型压缩和加速方案，使模型能够在资源受限的环境中高效运行；设计增量学习机制，使模型能够持续从新数据中学习并适应不断变化的语言表达方式；同时提供更好的解释性分析工具，帮助用户理解模型的预测依据，增强模型预测结果的可解释性和可信度。这些改进不仅能提升系统的实用价值，也能为情感分析技术在更广泛领域的应用奠定基础。

这些改进方向相互关联、互为补充，通过多个方面的协同优化，我们期望能够开发出性能更优、效率更高、实用性更强的情感分类系统，更好地服务于实际应用需求。

# 参考文献

1. Hsu, C. W., Chang, C. C., & Lin, C. J. (2003). A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University.

2. Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In European conference on machine learning (pp. 137-142). Springer, Berlin, Heidelberg.

3. Li, X., Wang, L., & Sung, E. (2008). AdaBoost with SVM-based component classifiers. Engineering Applications of Artificial Intelligence, 21(5), 785-795.

3. Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. Advances in neural information processing systems, 28, 649-657.

4. Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural networks, 18(5-6), 602-610.

5. Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

6. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

7. Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., ... & Stoyanov, V. (2020). Unsupervised cross-lingual representation learning at scale. arXiv preprint arXiv:1911.02116.

# 附录

成员分工及贡献度自评

| 姓名 | 个人分工 | 小组任务贡献度 |
| --- | --- | --- |
| 谢政航 | AdaBoost-SVM算法 | 20% |
| 冼伟杰 | TextCNN算法 | 20% |
| 徐樾 | BiLSTM算法 | 20% |
| 邓梓钊 | SVM算法，逻辑回归，随机森林 | 20% |
| 陈灿豪 | BERT算法 | 20% |

课程体会与建议

在学习模式识别课程的过程中，我深刻感受到了这门课程的魅力。尤其是在完成商品评价情感分类的课程实验后，我对模式识别技术的应用有了更为具体的认识。以下是我对这次学习经历的具体体会和建议：

在课程理论学习阶段，从基础的统计学原理到复杂的机器学习算法，每一章节都让我感受到了知识的深度。特别是在学习情感分类这一章节时，我了解到如何通过提取文本特征，将非结构化的数据转化为可量化的信息，这个过程让我对数据处理的奥秘产生了浓厚兴趣。

在进行商品评价情感分类的实验时，首先遇到了数据预处理这一难题。面对大量的商品评价数据，学会了如何清洗、去重和分词，以便提取出有用的特征。这个过程虽然繁琐，但让我明白了数据预处理在模式识别中的重要性。通过不断调整参数和优化模型，最终成功构建了一个能够较为准确地进行情感分类的模型。

实验完成后，我对模式识别课程有了更深的体会。我发现，这门课程不仅要求我们掌握理论知识，还要求我们具备将理论应用于实际问题的能力。在实验过程中，我学会了如何分析问题、设计方案并解决问题，这对我的实践能力提升有很大帮助。

在学习过程中，我也发现了一些可以改进的地方。首先，课程中的案例教学相对较少，导致我们在理解某些抽象概念时感到困难。因此，我建议教师在授课时，可以结合更多实际案例，让我们更好地理解模式识别技术的应用场景。此外，实验内容虽然具有一定的挑战性，但仍有拓展空间。例如，可以增加实验的数据量，让我们在处理更大规模数据时，更好地掌握模式识别技术。同时，教师可以引导学生尝试不同的算法和模型，以提高实验的多样性和趣味性。

最后，关于学习资源，我希望学校能提供更多与模式识别相关的教材、论文和在线课程，以便我们在课后能够自主学习，进一步拓宽知识面。

总之，通过学习模式识别课程并完成商品评价情感分类实验，我对这门课程有了更为全面的认识。我相信，在今后的学习和工作中，这次经历将对我的成长产生深远影响。希望我的体会和建议能为课程的发展和完善提供一些参考。