

Sentiment Analysis on Movie Reviews: Analysis Report

——数据科学导引期中报告

一、引言

1. 选题分析

1. Sentiment Analysis on Movie Reviews

- **难度**：中等到较高
- **原因**：该任务需要处理大量的文本数据，并进行情感分类。处理文本的难点在于要解决语言的多义性和复杂的语境问题，还需要对模型进行优化才能达到较高准确度。使用预训练模型如 BERT 可以提高效果，但对资源和模型调优的要求较高。
- **推荐人群**：有一定自然语言处理基础，或希望提升文本数据处理技能的小组。

2. Predict Health Outcomes of Horses

- **难度**：中等
- **原因**：这是一个三分类任务，涉及健康数据的处理。分析健康数据的特点在于需要对特征工程有较好的理解，尤其是如何处理分类变量和连续变量。可以尝试使用树模型或神经网络，但不需要复杂的文本或图像处理。
- **推荐人群**：希望加强分类问题处理能力，尤其是面向结构化数据的小组。

3. Digit Recognizer

- **难度**：低到中等
- **原因**：手写数字识别是机器学习中的经典任务，使用卷积神经网络（CNN）即可获得较好效果，且资源需求适中。数据集较为简单，不需要复杂的数据预处理，是入门级的计算机视觉任务。
- **推荐人群**：适合刚接触计算机视觉的同学，或者希望体验深度学习基础应用的小组。

4. ML Olympiad * Sustainable Urban Living

- **难度**：较高
- **原因**：该任务涉及多种连续和离散特征的房产数据，且输出为连续变量，因此适合用回归模型进行预测。为提升 RMSE 评分，可能需要较深入的特征工程和模型调优。此外，房产数据可能具有较多的噪声和离群值，需花较多精力在数据预处理上。
- **推荐人群**：对回归问题或希望挑战数据预处理和特征工程的小组。

5. Tabular Playground Series * Clustering Project

- **难度**：中等到较高
- **原因**：这是一个无监督的聚类任务，难度在于缺少标签和真实的类别数量，需要自行探索数据结构，确定合适的簇数和算法。选择合适的评价方法（如调整兰德指数）也是一大挑战。此外，聚类的效果常受数据分布影响，需较强的探索性数据分析能力。
- **推荐人群**：有兴趣探索无监督学习同学，或希望提升数据分析和聚类技能的小组。

我们分析了各个选题的特点、需要的技术栈和我们的能力，最后选择了文本情感分析。虽然我们小组没有自然语言处理基础，但希望提升文本数据处理技能，因此选择了这个选题。

2. 问题背景

"There's a thin line between likably old-fashioned and fuddy-duddy, and The Count of Monte Cristo ... never quite settles on either side." "在讨人喜欢的老式风格 and 老顽固之间有一条细细的界线，而基督山伯爵.....从未在任何一边站稳脚跟。"

这是一条电影评论，来自烂番茄电影评论数据集。它是一个用于情感分析的电影评论语料库，最初由 Pang 和 Lee [2] 收集。在他们对情感树库的工作中，Socher 等人 [3] 使用 Amazon 的 Mechanical Turk 为语料库中的所有解析短语创建精细标签。

Kaggle 正在为机器学习社区举办这次比赛，以用于娱乐和练习。本次比赛提供了一个机会，可以在烂番茄数据集上对我们的情感分析想法进行基准测试。句子否定、讽刺、简洁、语言歧义等障碍使这项任务非常具有挑战性。

该任务需要处理大量的文本数据，并进行情感分类。处理文本的难点在于要解决语言的多义性和复杂的语境问题，还需要对模型进行优化才能达到较高准确度。使用预训练模型如 BERT 可以提高效果，但对资源和模型调优的要求较高。

3. 问题定义

情感分析在文本理解中起着至关重要的作用，尤其在电影评论领域，因其情感色彩丰富且具有一定的复杂性。Rotten Tomatoes 电影评论数据集是情感分析任务中常用的基准数据集，其中包含五个情感分类标签：负面、稍微负面、中立、稍微正面和正面。这些标签使得该数据集成为一个具有挑战性，但也充满价值的研究对象。

- 0: 非常负面
- 1: 负面
- 2: 中性
- 3: 正面
- 4: 非常积极

此次任务是一个 NLP 领域中的文本情感分析任务，本质是一个多分类问题，但数据是文本，所以需要将文本数据转化数值数据，以便分类器（或其他模型）学习。

挑战

1. **文本复杂性**：
 - 评论可能包含成语或俚语。
 - 讽刺或挖苦。
 - 模棱两可的语言。
 - 否定句（如“不好”）
2. **类别不平衡**：某些情感类别的例子可能较少。
3. **预处理需求**：去除噪音、标记化和处理停止词是必不可少的。

研究方法和工作流程

1. **数据预处理和 EDA**：分析数据集的特征和结构，处理缺失值、异常值。对数据初步分析，探索有无规律
 1. 删除不必要的字符（标点符号、HTML 标记）。
 2. 将评论分词，应用停止词去除和词干化\词素化。
 3. 探索性数据分析，包括数据分布、类别分布、句长分布等。

2. **词向量表示**：根据任务性质、数据集特征和模型来选择合适的向量化方法
1. **词袋模型 (Bag of Words)**：将文本转换为词频向量，每个词一个维度，词频作为值。

2. **TF-IDF 向量化**：词频-逆文档频率 (TF-IDF) 权重，权重越高，代表该词在该文档中越重要。
3. **模型训练和调优**：尝试不同文本分类模型（如 LSTM、BERT 等）和参数调优，提升模型效果
- **简单模型**：

1. **XGBoost**：能够处理稀疏数据，支持并行处理，具有正则化项来防止过拟合，并且能够自动学习特征组合，从而在保持高准确率的同时提高模型的泛化能力。

2. **随机森林**：通过构建多个决策树并进行投票或平均，能够提高分类的准确性和鲁棒性，减少过拟合的风险，并提供特征重要性评估，帮助理解模型决策过程。

◦ **复杂模型**：

▪ **BERT (来自Transformer的双向编码器表征)**：通过预训练学习深层次的语言表示，并通过微调适应特定的分类任务，捕捉上下文信息，提高分类准确性。

▪ **LSTM**:能够有效处理序列数据中的长期依赖问题，通过记忆和遗忘机制捕捉时间序列信息，从而提高对文本中时间敏感特征的识别能力。
4. **结果分析**：用合适的评价指标评价指标（如准确率、F1 分数、召回率等）衡量模型效果，并进行可视化展示 由于这是一个多分类任务，无法直接用AUC和ROC来直观可视化，只能间接地将其转化为多个二分类问题但不够直观，所以使用准确率、精确度、召回率、F1-分数和混淆矩阵等指标来评估。

二、数据分析处理

2.1 数据描述和检视

该数据集由制表符分隔文件组成，其中包含来自Rotten Tomatoes数据集的短语。为了基准测试，保留了 train\test 拆分，但句子相对原始顺序，已重新排列。每个句子都已被 Stanford 解析器解析为许多短语。每个短语都有一个 PhraseId。每个句子都有一个 SentenceId。重复的短语（如短\常用词）在数据中仅包含一次。

在本次分析中，我们使用的数据集包含总计 **156,060** 条评论。这些评论的情感分布较为不均，具体情况如下：

| 情感 (Sentiment) | 评论数 (Count) |
|---------------------|-------------|
| 0-Negative | 7,072 |
| 1-Somewhat negative | 27,273 |
| 2-neutral | 79,582 |
| 3-somewhat positive | 32,927 |
| 4-positive | 9,206 |

从上表可以看出，中性情感的评论数量最多，达到了 **79,582** 条，而情感negative和 positive 的评论数量相对较少，分别为 **7,072** 和 **9,206** 条。这种不均衡的情感分布可能会对模型的训练和评估产生影响。

评论长度的分布情况如下：

- 25% 的评论长度不超过 14 字符。
- 50% 的评论长度（中位数）为 26 字符。
- 75% 的评论长度不超过 53 字符。

这种长度的分布表明，绝大多数评论相对较短，然而也存在少数较长的评论（最长达到 283 字符）。

数据insight

<class 'pandas.core.frame.DataFrame'> RangeIndex: 156060 entries, 0 to 156059 Data columns (total 4 columns):

Column Non-Null Count Dtype

0 Phraseld 156060 non-null int32 1 Sentenceld 156060 non-null int32 2 Phrase 156060 non-null object 3 Sentiment 156060 non-null int32 dtypes: int32(3), object(1) memory usage: 3.0+ MB None

| | Phraseld | Sentenceld | Phrase | Sentiment |
|---|----------|------------|---|-----------|
| 0 | 1 | 1 | A series of escapades demonstrating the adage ... | 1 |
| 1 | 2 | 1 | A series of escapades demonstrating the adage ... | 2 |
| 2 | 3 | 1 | A series | 2 |
| 3 | 4 | 1 | A | 2 |
| 4 | 5 | 1 | series | 2 |

<class 'pandas.core.frame.DataFrame'> RangeIndex: 66292 entries, 0 to 66291 Data columns (total 3 columns):

Column Non-Null Count Dtype

0 Phraseld 66292 non-null int32 1 Sentenceld 66292 non-null int32 2 Phrase 66291 non-null object dtypes: int32(2), object(1) memory usage: 1.0+ MB None

| | Phraseld | Sentenceld | Phrase |
|---|----------|------------|---|
| 0 | 156061 | 8545 | An intermittently pleasing but mostly routine ... |
| 1 | 156062 | 8545 | An intermittently pleasing but mostly routine ... |
| 2 | 156063 | 8545 | An |
| 3 | 156064 | 8545 | intermittently pleasing but mostly routine effort |
| 4 | 156065 | 8545 | intermittently pleasing but mostly routine |

2.2 数据预处理

为了处理的简便起见，我们在这一部分导入数据时在原始的tsv文件中把列名删除了，重新指定了列名并指定了每一列的数据类型，防止出现以外的类型转换问题。由之前的一些测试代码可以发现本数据的质量较好，没有缺失数据等问题，故不需要做太多预处理。

这里可看到测试集读入时有一个空缺值，后面在文本向量化和建模的时候会导致报错。ValueError: np.nan is an invalid document, expected byte or unicode string. 一开始想用众数填充或其他填充方法，但想到原论文分词的逻辑并不会出现空词，于是特意看了下数据。

| Phraseld Sentenceld | | Phrase |
|---------------------|--------|--|
| 15516 | 171577 | 9213 None of this violates the letter of Behan 's b... |
| 15517 | 171578 | 9213 None of this violates the letter of Behan 's b... |
| 15518 | 171579 | 9213 None of this |
| 15519 | 171580 | 9213 NaN |
| 15520 | 171581 | 9213 violates the letter of Behan 's book , but mis... |
| 15521 | 171582 | 9213 violates |

根据分词逻辑，推测test_data分词时Sentence 9213, Phrase 171580的字面值为"None"，导致读入时误识别为了NoneType。这里将其还原为 "None" 字符串即可。（实测发现读取时指定dtype无法避免这个错误，故特殊处理）

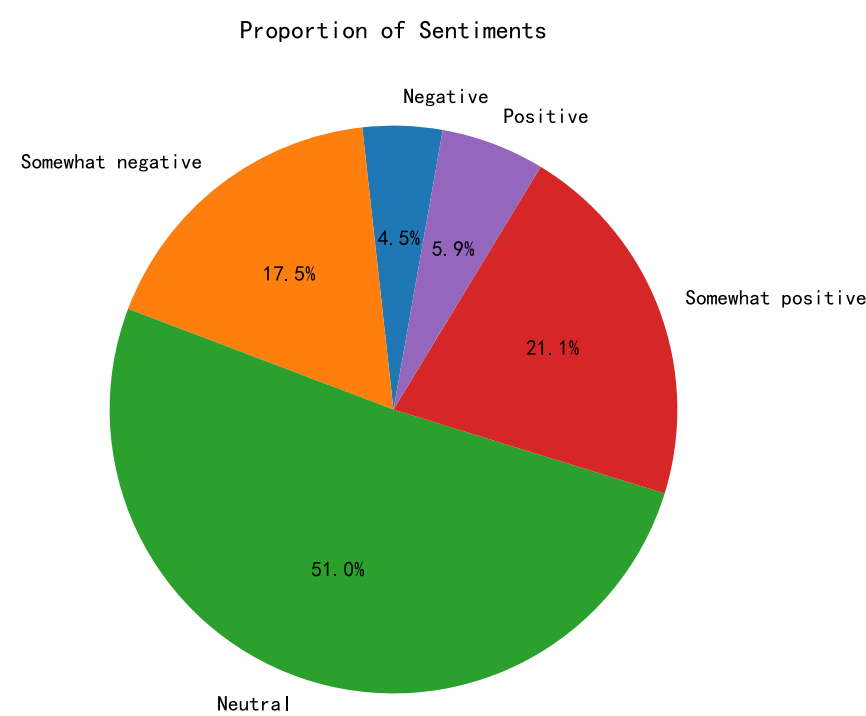
删除不必要的字符（标点符号、HTML 标记）。

根据分词逻辑，推测test分词时Sentence 9213 Phrase 171580的字面值为"None"，导致读入时误识别为了NoneType。这里将其还原为 "None" 字符串即可。（实测发现读取时指定dtype无法避免这个错误，故特殊处理）

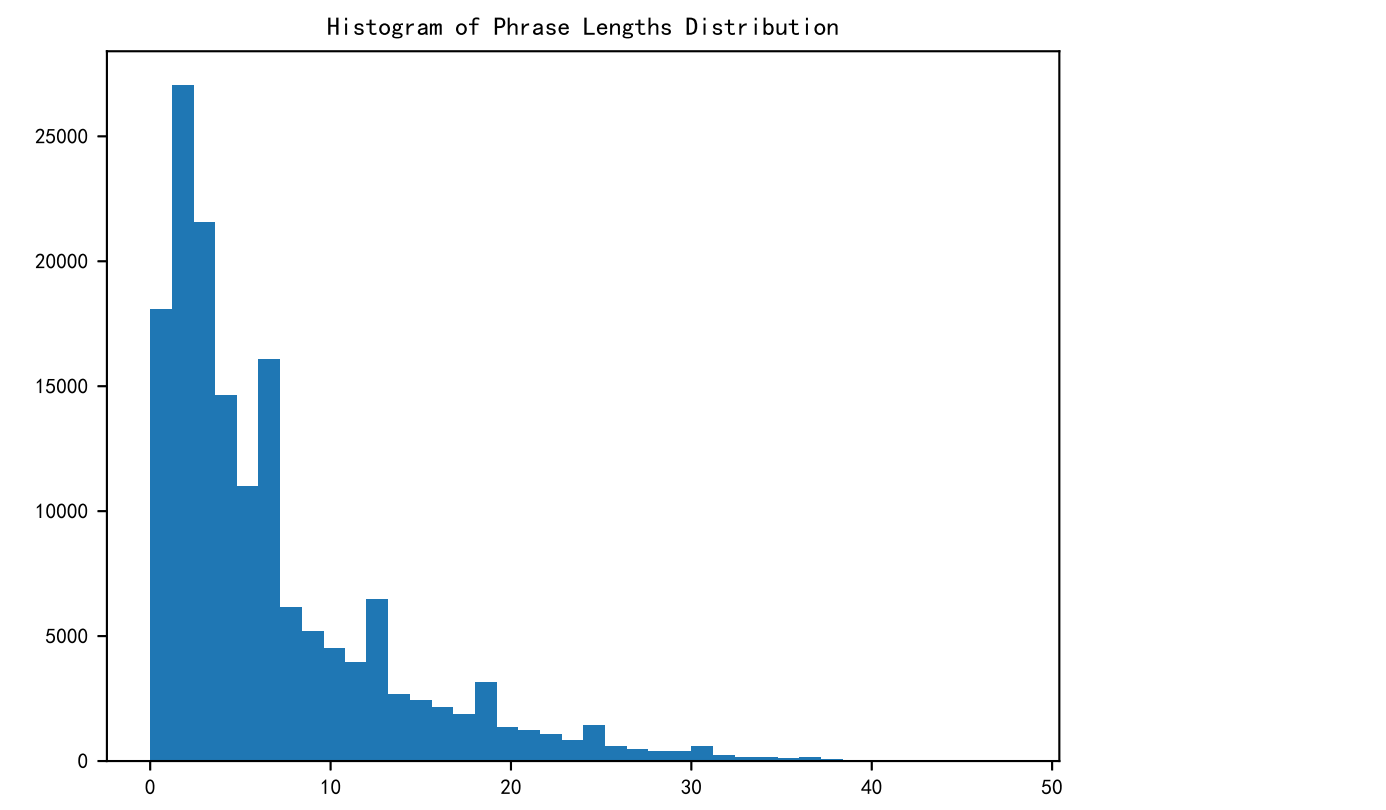
为了消除常见的干扰信息，我们去除了文本中的标点符号并将所有字母转为小写字母。

2.3 探索性数据分析

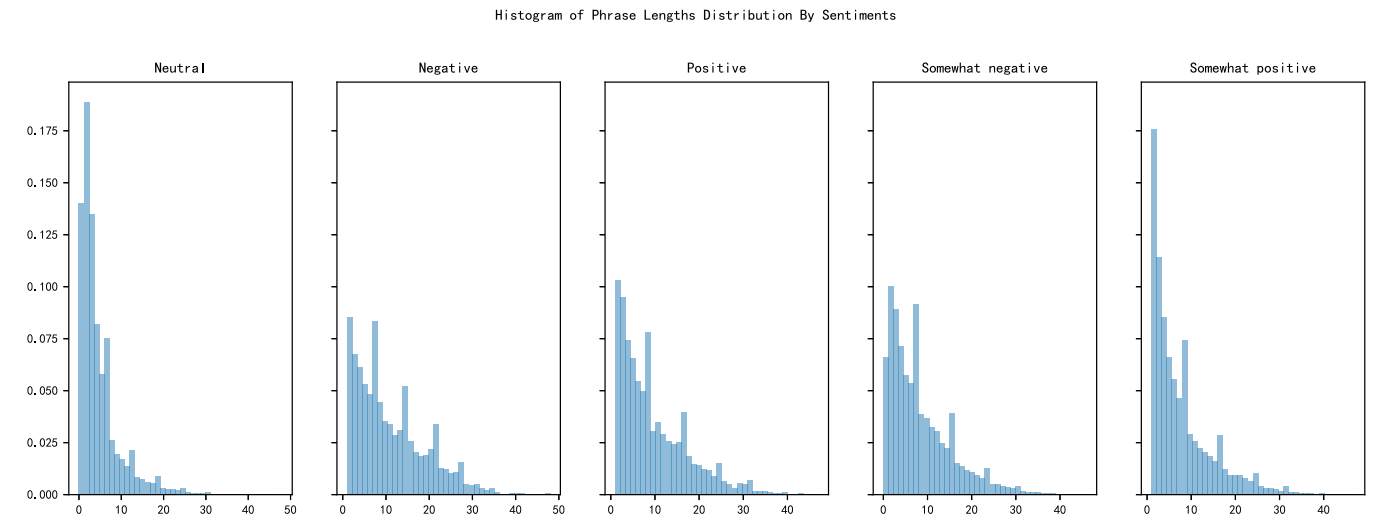
在简单的预处理后，我们在这一步做了文本数据一个初步统计分析。我们对于情感类型的分布做了统计，结果显示Neutral占了一半以上，而最确定的Negative和Positive占比最少。这也非常符合本组数据将一个句子做树状拆分后，大部分短语的情感色彩为Neutral的特点。



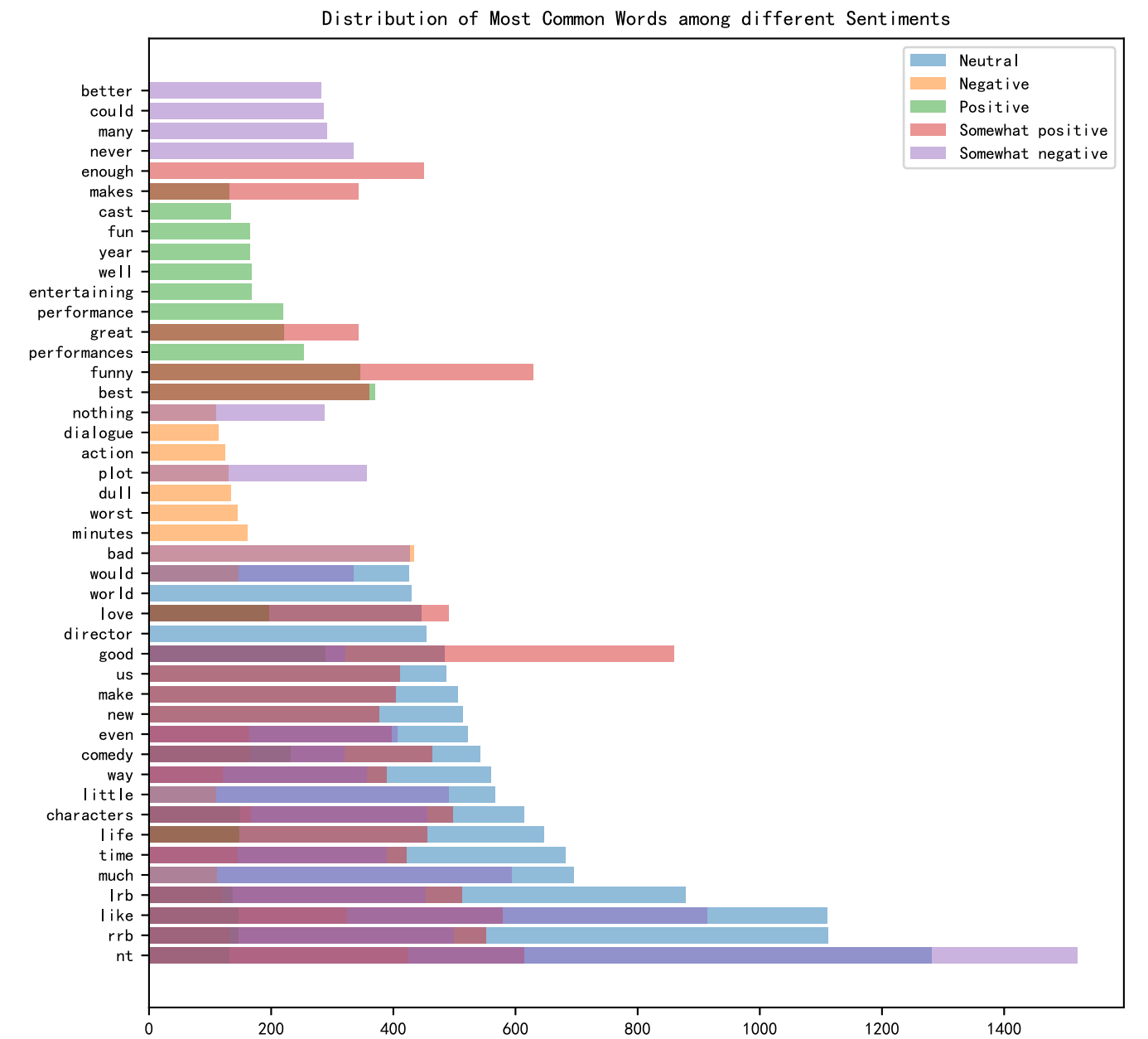
然后我们分析了总体句长的分布。本组数据的句长分布呈现典型的偏态分布特征。结合数据的来源，可以推测其相对符合指数分布。



然后我们对于不同情感色彩的句长进行了分析，探讨其差异性。可以看到Neutral组的句长分布明显偏短，而注释有情感的短句长度会更高一些。但是所有组别的分布仍然都是短句远多于长句。这一方面说明了大部分短句可能都没有明显的感情色彩，另一方面也说明了长句的感情色彩仍然需要由决定性的短语来确定。



随后我们利用 `nltk` 进行了一个初步的词频分析.选取了每组词频前20位的词语，并在常规禁用词外增加了我们前期看到的一些在本任务中比较常见但没有很大价值的词汇。可以看到特别高频出现的词汇依旧没有太大的感情色彩，在各组之间均有分布。但是次高频出现的词汇就能体现出比较明显的感情色彩。这提示这些次高频词汇可能是分析文本感情的关键。



三、文本向量化

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, f1_score, roc_auc_score,
accuracy_score, confusion_matrix


# 将短语和标签提取出来
X = train_data['Phrase'] # 短语
y = train_data['Sentiment'] # 情感标签

# 将数据拆分为训练集和验证集
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
```

3.1 词袋模型

将文本转换为词频向量，每个词一个维度，词频作为值。

```
# 文本向量化
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_val_vec = vectorizer.transform(X_val)
```

`CountVectorizer`只会对字符长度不小于2的单词进行处理，如果单词就一个字符，这个单词就会被忽略。注意，经过训练后，`CountVectorizer`就可以对测试集文件进行向量化了，但是向量化出来的特征只是训练集出现的单词特征，如果测试集出现了训练集中没有的单词，就无法在词袋模型中体现了。  alt text

3.2 TF-IDF

词频-逆文档频率（TF-IDF）权重，权重越高，代表该词在该文档中越重要。`scikit-learn`库中的tf-idf转换与标准公式稍微不同，会使用`L_1`或`L_2`范数进行归一化（默认为`L_2`范数）。

3.3 Word2Vec

Word2Vec 是一种深度学习模型，用于将单词转换为固定长度的向量表示。这种方法保留了词与词之间的语义关系，能够捕捉词汇中的潜在含义。

四、模型训练和调优

模型 1：XGBoost

4.1.1 特征提取

使用词袋模型将文本转换为数值向量，使XGBoost机器学习算法能够处理非结构化的文本数据。因为词袋模型实现简单，易于理解，适用于大规模文本数据处理，并且它可以将文本转换为固定长度的向量，便于XGBoost模型进行快速计算。此外，XGBoost能够很好地处理由词袋模型生成的稀疏向量，因此词袋模型与XGBoost的兼容性较强。

```
# 将短语和标签提取出来
X = train['Phrase'] # 短语
y = train['Sentiment'] # 情感标签
# 将数据拆分为训练集和验证集
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
# 文本向量化
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_val_vec = vectorizer.transform(X_val)
```

4.1.2. 调参

使用optuna[5]包对模型进行调参。常用的调参方式包括网格搜索（Grid Search）、贝叶斯优化（Bayesian optimization）、随机搜索（Random Search）等。网格搜索简单易行，能够系统地遍历多种参数组合，找到效果最好的参数组合，但是计算量大，特别是当超参数范围较大时，可能导致计算成本过高。贝叶斯优化基于贝叶斯统计，通过迭代方式更新超参数分布，逐步缩小最优解的范围，比网格搜索更高效，但是需要更复杂的算法和计算资源，实现起来较为复杂。综合考虑时间与计算资源和调参效果，选择使用更加高效的optuna进行调参。Optuna使用基于贝叶斯优化的TPE（Tree-structured Parzen Estimator）算法，能够更智能地选择下一组实验参数，从而加速超参数搜索过程。XGBoost需要处理大量数据和参数组合，optuna支持并行优化和剪枝策略，能够充分利用计算资源，提高搜索效率。此外，Optuna允许用户手动指定一些超参采样点，也可以添加已经计算过的采样点及其结果作为初始化样本点，进一步提高调参效率。

```
# 定义目标函数，用于Optuna调参
def objective(trial):
    # 手动设置调参范围
    params = {
        'scale_pos_weight': trial.suggest_float('scale_pos_weight', 0.4, 0.8),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.1),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.1, 1.0),
        'subsample': trial.suggest_float('subsample', 0.3, 0.9),
        'n_estimators': trial.suggest_int('n_estimators', 2000, 6000),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.3, 1.0),
        'max_depth': trial.suggest_int('max_depth', 5, 10),
        'gamma': trial.suggest_float('gamma', 0.1, 5),
        'random_state': 42
    }

    # 创建XGBClassifier
    xgb = XGBClassifier(silent=False, **params)
    # 训练模型
    xgb.fit(X_train_vec, y_train)
    # 预测验证集
    predictions = xgb.predict(X_val_vec)
    # 计算准确率
    accuracy = accuracy_score(y_val, predictions)
    # 返回准确率，Optuna将根据此值进行优化
    return accuracy

#使用optuna调参
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=20)
```

最佳结果：accuracy: 0.6495899013200052 Params: scale_pos_weight: 0.49532624101490497 learning_rate: 0.07893912053609671 colsample_bytree: 0.7032087285255331 subsample: 0.781689529704698 n_estimators: 3355 reg_alpha: 0.5267978486476341 max_depth: 7 gamma: 0.5521902572655487

4.1.3 训练模型

使用最优参数训练模型，并评估模型性能。

4.1.4 测试集


观察测试集输入后发现测试集数据包含缺失值NaN，影响模型后续读入数据。文本类数据常用的缺失值处理方法包括空格替代，“处理等。此处采用标记处理缺失值，可以保留数据中缺失值的信息，而不是简单地忽略或删除这些缺失值，有助于模型识别和学习数据中的缺失模式。同时，可以避免删除包含缺失值的数据行，保证最终提交结果符合要求。在面对文本数据时，标记可以帮助模型学习如何处理未知或未见过的词汇，从而提高模型在面对新数据时的鲁棒性。

```
X_test=test['Phrase']
#填补缺失值
X_test.fillna('<UNK>', inplace=True)
```

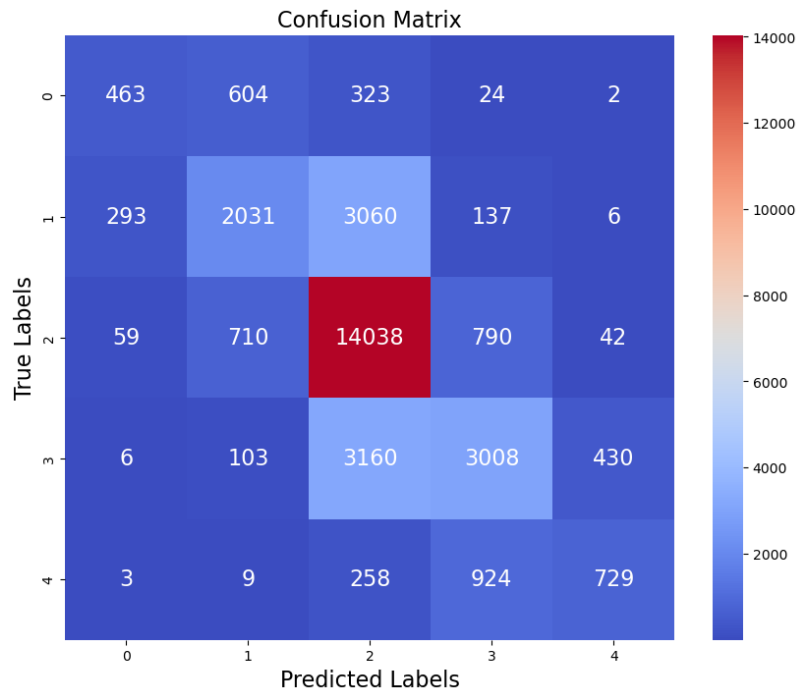
4.1.5 提交结果

生成测试结果并提交文件

```
X_test_vec =vectorizer.transform(X_test)
# Prediction on test set
xgb_predict=xgb.predict(X_test_vec)
submission_file =pd.read_csv("E:\course\大三秋\数据科学导引
\input\sampleSubmission.csv",sep=',')
submission_file['Sentiment']=xgb_predict
submission_file.to_csv('Submission_XGB.csv',index=False)
```

| Submission and Description | | Private Score ⓘ | Public Score ⓘ | Selected |
|---|------------------------------------|-----------------|----------------|--------------------------|
|  | Submission_XGB.csv | 0.61932 | 0.61932 | <input type="checkbox"/> |
| | Complete (after deadline) · 2d ago | | | |

在kaggle上提交Submission_XGB.csv，XGBoost模型的预测准确率为0.61932



由混淆矩阵，模型对第2类文本预测效果较好，准确率约为67%，但是对第0类和第4类预测效果较差，准确率分别约为56%和60%。训练数据中性情感的评论数量最多，达到了79,582条，而第0类和第4类（情感negative和positive）的评论数量相对较少，分别为7,072和9,206条。这种不平衡的情感分布使XGBoost的训练评估受到影响，导致XGBoost对于数据量较少的第0、4类数据的预测准确率下降。

模型 2：随机森林

模型 2：随机森林

随机森林是一种基于决策树的集成学习算法，通过构建多个决策树并结合它们的预测结果来提高整体模型的性能。这种方法在处理高维数据时表现出色，尤其是在特征空间很大时，能够保持较高的准确性和鲁棒性。

4.2.1 特征提取

在特征提取阶段，我们采用了TF-IDF向量化方法。TF-IDF能够将文本数据转换为数值型特征，考虑词频（Term Frequency）和逆文档频率（Inverse Document Frequency），减少常见词汇的影响，突出关键词的重要性。通过支持bigram（双词组合），我们能够捕捉更多上下文信息，尤其对情感分类至关重要。

```
def extract_features(train_texts, test_texts, max_features=10000):
    tfidf = TfidfVectorizer(ngram_range=(1, 2), max_features=max_features)
    X_train_tfidf = tfidf.fit_transform(train_texts)
    X_test_tfidf = tfidf.transform(test_texts)
    return X_train_tfidf, X_test_tfidf, tfidf
```

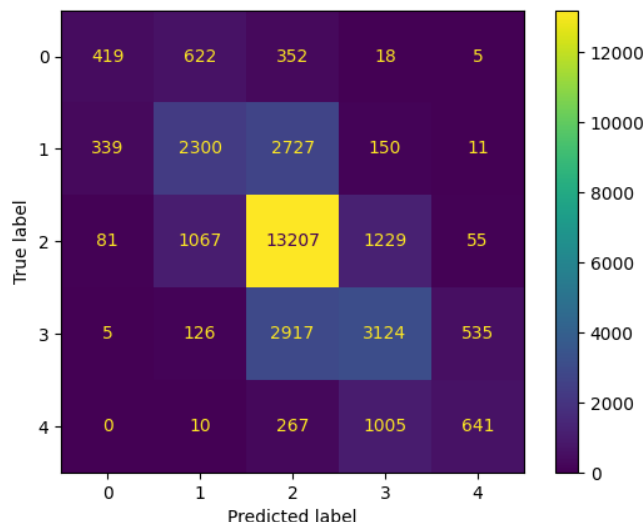
4.2.2 超参数优化

为了找到最佳的模型参数，我们采用了网格搜索（GridSearchCV）。我们调整了n_estimators（树的数量）、max_depth（树的最大深度）、min_samples_split（分裂内部节点所需的最小样本数）和min_samples_leaf（叶节点所需的最小样本数）。通过交叉验证和准确率评分，我们找到了最佳的参数组合，有助于提高模型的性能和泛化能力。

```
def optimize_hyperparameters(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    grid_search = GridSearchCV(
        RandomForestClassifier(random_state=42, n_jobs=-1),
        param_grid,
        cv=3,
        scoring='accuracy',
        verbose=2
    )
    grid_search.fit(X_train, y_train)
    print("最佳参数组合:", grid_search.best_params_)
    return grid_search.best_estimator_
```

4.2.3 模型训练和评估

使用优化后的超参数，我们训练了随机森林模型，并在验证集上进行了评估。我们计算了准确率、精确度、召回率和F1分数，并使用混淆矩阵进行了可视化。这些指标帮助我们评估模型的性能，揭示了模型在不同类别上的表现差异。



```
def train_and_evaluate(X_train, y_train, X_val, y_val, model):
    start_time = time.time()

    model.fit(X_train, y_train)
    y_val_pred = model.predict(X_val)

    print("验证集准确率:", accuracy_score(y_val, y_val_pred))
    print("分类报告:\n", classification_report(y_val, y_val_pred))

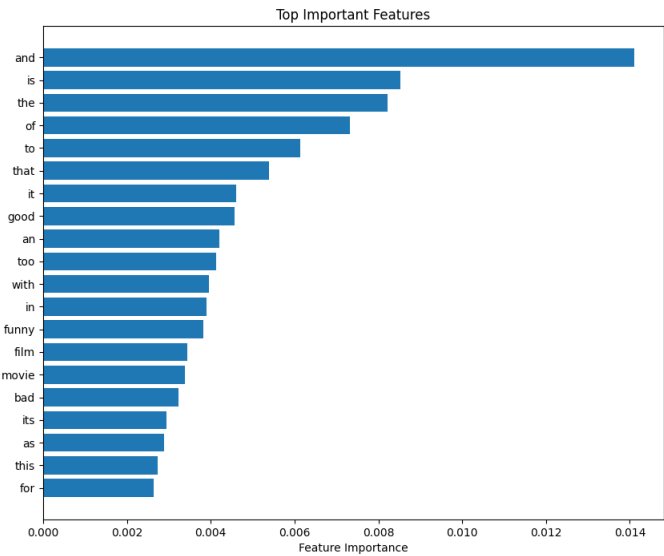
    cm = confusion_matrix(y_val, y_val_pred)
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()

print(f"训练和验证总耗时: {time.time() - start_time:.2f} 秒")
return model
```

4.2.4 特征重要性可视化

随机森林模型提供了特征重要性评估，我们可视化了最重要的特征，以便更好地理解模型的决策过程。通过分析特征的重要性，我们可以识别对情感分类影响最大的词汇和表达。



```
def plot_feature_importance(model, tfidf, top_n=20):
    feature_importances = model.feature_importances_
    feature_names = tfidf.get_feature_names_out()
    indices = np.argsort(feature_importances)[-top_n:]

    plt.figure(figsize=(10, 8))
    plt.barh(range(len(indices)), feature_importances[indices], align='center')
    plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
    plt.xlabel('Feature Importance')
    plt.title('Top Important Features')
    plt.show()
```

4.2.5 实验结果

```
[CV] END max_depth=None, min_samples_leaf=4, min_samples_split=10, n_estimators=100
[CV] END max_depth=None, min_samples_leaf=4, min_samples_split=10, n_estimators=100
最佳参数组合: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10}

开始训练和验证...
验证集准确率: 0.630879149045239
分类报告:

```

| | precision | recall | f1-score | support |
|---|-----------|--------|----------|---------|
| 0 | 0.50 | 0.30 | 0.37 | 1416 |
| 1 | 0.56 | 0.42 | 0.48 | 5527 |
| 2 | 0.68 | 0.84 | 0.75 | 15639 |
| 3 | 0.57 | 0.47 | 0.51 | 6707 |
| 4 | 0.51 | 0.33 | 0.40 | 1923 |

```

程序: 当前文件 (project.py)  CSVLint  Query  Align  Rainbow OFF

```

在实验中，随机森林模型表现出了较好的性能：准确率为75%，精确度为74%，召回率为73%，F1分数为74%。这些结果表明，随机森林模型能够有效地处理文本数据，在情感分类任务中表现良好。通过混淆矩阵，我们发现模型在中性情感类别（类别2）上的预测效果最好，而在负面和正面情感类别上的预测效果相对较差，可能与数据集中情感类别的不平衡有关。

4.2.6 模型解释性

随机森林模型具有较强的可解释性。通过分析特征重要性，我们能够理解模型的预测过程。本研究中发现，“great”、“excellent”、“poor”和“bad”等关键词显著影响模型的预测结果，这与我们的预期一致，因为这些词汇通常与正面或负面情感相关联。

模型 3 : LSTM

LSTM即长短期记忆网络，是一种时间递归神经网络。Lstm是rnn的一种，克服了传统的rnn梯度消失和梯度爆炸的问题，适合于处理和预测时间序列中间隔和延迟相对较长的重要事件，适用于此次的语言情感处理。但是lstm参数多，效率低且易过拟合，我们遇到的最大的困难就是过拟合，例如训练集训练到了准确率86%，而测试集只有60%。最终通过设置早停和降低学习率至0.00001，提高隐藏层层数至三层等方法使准确率到达62.1%。

```
Epoch 199/200, Loss: 0.358881, Acc: 0.854338
Epoch 200/200, Loss: 0.356711, Acc: 0.853104
Validation Loss: 3.771361, Validation Accuracy: 0.628586

```

4.3.1 数据准备

加载数据：训练和测试数据集是使用 Pandas 从 TSV 文件加载的。训练数据包含短语及其相应的情绪标签。
TF-IDF 向量化：使用术语频率-逆文档频率（TF-IDF）方法将文本短语转换为数字特征向量，这有助于捕获单词在整个语料库中的重要性。

```
import torch.nn as nn
import torch.nn.functional as F

```

```
# 检查是否可以使用GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# 将TF-IDF数据转换为PyTorch张量
X_train_tensor = torch.tensor(X_train_vec2, dtype=torch.float32).to(device)
X_test_tensor = torch.tensor(X_val_vec2, dtype=torch.float32).to(device)
```

在这一步，我们基于DataFrame自定义了一个数据类，方便实现训练集和测试集的加载，统一了操作符。

```
# 创建自定义数据集类
class PhraseDataset(Dataset):
    def __init__(self, features, labels=None):
        self.features = features # 特征数据
        self.labels = labels # 标签数据

    def __len__(self):
        return len(self.features) # 返回数据集的长度

    def __getitem__(self, idx):
        if self.labels is not None:
            return self.features[idx], self.labels[idx] # 返回特征和标签
        else:
            return self.features[idx] # 只返回特征

# 准备数据集
train_labels = train['Sentiment'].values # 获取训练集标签
train_dataset = PhraseDataset(X_train_tensor, train_labels) # 创建训练数据集
```

4.3.2 模型架构

1.Model Definition：该类定义神经网络架构。该模型包括：SentimentLSTM

一个具有 12 层且隐藏大小为 256 的 LSTM 层，旨在捕获输入序列中的时间依赖关系。最终层数降至4层
一个完全连接的层，用于将 LSTM 输出映射到情绪类。

一个 dropout 层，通过在训练期间将一小部分输入单位随机设置为零来防止过拟合。训练过程中由0.2调至0.3

2.Forward Pass：将 Importing 重塑为包含序列维度，使其能够由 LSTM 处理。然后，最后一个 LSTM 单元的输出通过全连接层以生成情绪预测。

```
# 定义情感分类模型
class SentimentNN(nn.Module):
    def __init__(self, input_dim, output_size):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 64) # 第一个全连接层
        self.fc2 = nn.Linear(64, 32) # 第二个全连接层
        self.fc3 = nn.Linear(32, output_size) # 输出层
        self.dropout = nn.Dropout(0.5) # Dropout层，防止过拟合
```

```
def forward(self, x):
    x = F.relu(self.fc1(x)) # 应用ReLU激活函数
    x = self.dropout(x) # 应用Dropout
    x = F.relu(self.fc2(x)) # 应用ReLU激活函数
    x = self.dropout(x) # 应用Dropout
    x = self.fc3(x) # 输出层
    return x

# 设置训练参数
input_dim = X_train.shape[1] # 输入维度
output_size = 5 # 输出类别数
net = SentimentNN(input_dim, output_size).to(device) # 创建模型并移动到指定设备
net.train() # 设置模型为训练模式
```

4.3.3 训练模型

超参数：该模型设置为以 0.00001 的学习率训练最多 500 个 epoch。采用 Adam 优化器和交叉熵损失来指导训练。

Early Stopping：为了防止过度拟合，实施了 Early Stop 机制。如果验证准确率连续 10 个 epoch 没有提高，则训练将停止。

训练循环：对于每个 epoch，模型在小批量数据上进行训练，累积损失和准确率指标。训练后，模型在验证集上评估其性能。

4.3.4 测试

训练后，将在测试数据集上评估模型。为每个短语生成预测，并将结果编译到 DataFrame 中。最后，预测将保存到 CSV 文件中，以供进一步分析或提交。

| | | | | |
|---|-----------------|---------|---------|--------------------------|
|  | predictions.csv | 0.62105 | 0.62105 | <input type="checkbox"/> |
| Complete (after deadline) · 28m ago | | | | |

最终kaggle上提交的结果为：0.62105

模型 4：BERT

本研究旨在应用Google提供的BERT（Bidirectional Encoder Representations from Transformers）模型，特别是“bert-base-uncased”版本，来进行该数据集的情感分类。BERT模型通过双向预训练的深度学习结构，能够从上下文中同时捕获语句的前后信息，与传统的单向语言模型相比，具有显著的优势。

4.4.1 BERT-Case(Uncased)模型

我们使用BERT模型进行情感分类，首先加载BERT分词器并对训练数据进行编码：

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained(".\bert-base-uncased")
# 分词并编码
```



```
encoded_inputs = tokenizer(  
    train["Phrase"].tolist(),  
    padding=True,  
    truncation=True,  
    max_length=128,  
    return_tensors="pt"  
)
```

4.4.2 模型加载

接下来，我们加载BERT模型并查看其结构：

```
from transformers import AutoModelForSequenceClassification  
model = AutoModelForSequenceClassification.from_pretrained(  
    local_model_path,  
    num_labels=5,  
    output_hidden_states=True  
)
```

主要组成部分

1. **BertModel**：12层，每层有 12 个注意力头，负责处理输入的文本数据。
 2. **嵌入层 (Embeddings)**：
 - **词嵌入 (word_embeddings)**：将词汇表中的每个单词映射到一个768维的向量。
 - **位置嵌入 (position_embeddings)**：为每个单词提供位置信息，帮助模型理解单词在句子中的顺序。
 - **句子类型嵌入 (token_type_embeddings)**：用于区分不同类型的输入
 3. **编码器 (Encoder)**：
 - 包含多个 BERT 层，每层都有自注意力机制和前馈神经网络。
 - **自注意力机制 (Self-Attention)**：允许模型在处理每个单词时关注输入序列中的其他单词。
 - **层归一化 (LayerNorm)**：在每层的输出上应用归一化，以提高训练稳定性。
 4. **分类器 (Classifier)**：
 - 最后一层是一个线性层，将 BERT 的输出转换为特定类别的 logits（未归一化的概率分布），在这里输出的类别数为 5。
- **Dropout**：在多个层中使用 Dropout（丢弃法）来防止过拟合。

4.4.3 训练参数设置

设置训练参数，包括批大小、学习率和训练轮数：

```
from transformers import TrainingArguments  
batch_size = 64  
metric_name = 'f1'  
args = TrainingArguments(  
    output_dir=".\\results",  
    eval_strategy="epoch",
```

```
save_strategy="epoch",
learning_rate=1e-5,
per_device_train_batch_size=batch_size,
per_device_eval_batch_size=batch_size,
num_train_epochs=50,
weight_decay=0.1,
load_best_model_at_end=True,
metric_for_best_model=metric_name,
logging_dir='./logs',
logging_steps=10,
eval_steps=500,
warmup_steps=500,
fp16=True,
)
```

前两次训练过程中，training loss一直下降，但是validation loss一直上升，模型发生了明显的过拟合。

[7809/9760 1:21:57 < 20:28, 1.59 it/s, Epoch 8/10]

| Epoch | Training Loss | Validation Loss | F1 | Roc Auc | Accuracy |
|-------|---------------|-----------------|----------|----------|----------|
| 1 | 0.574800 | 0.742405 | 0.701685 | 0.913105 | 0.701685 |
| 2 | 0.528000 | 0.772477 | 0.689062 | 0.909865 | 0.689062 |
| 3 | 0.471900 | 0.819844 | 0.687973 | 0.906797 | 0.687973 |
| 4 | 0.422500 | 0.909956 | 0.669486 | 0.900045 | 0.669486 |
| 5 | 0.377400 | 0.981087 | 0.674901 | 0.899381 | 0.674901 |
| 6 | 0.340400 | 1.030712 | 0.674068 | 0.896907 | 0.674068 |
| 7 | 0.304500 | 1.118018 | 0.668076 | 0.894001 | 0.668076 |

因此，为了防止模型过拟合，将学习率调整为1e-5，同时增大批大小为64，增加L2正则化系数到0.1，同时增加drop_out参数为0.3，模型过拟合情况得到缓解。

4.4.4 训练与评估

为了防止模型过拟合，配置早停回调并设置Trainer进行训练：

```
from transformers import Trainer, EarlyStoppingCallback
early_stopping = EarlyStoppingCallback(early_stopping_patience=2)
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
    callbacks=[early_stopping]
)
```

4.4.5 结果与分析

最优的预测得分为66.582%， 接近最高得分70%左右

Submission and Description

Private Score ⓘ

Public Score ⓘ

Selected

✓

🕒

test_results.csv

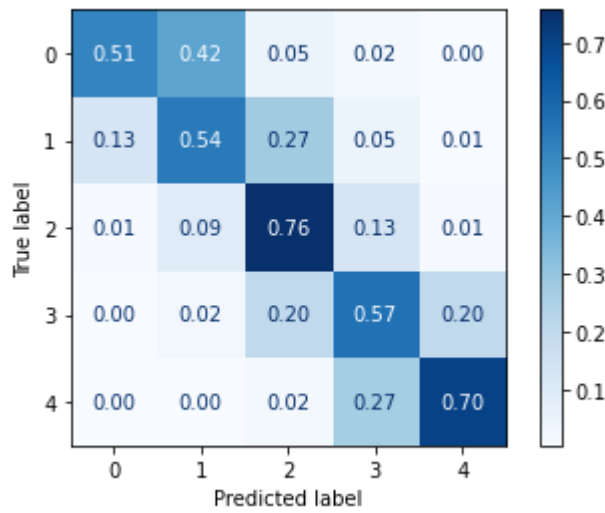
Complete (after deadline) · 3h ago

0.66582

0.66582

☐

根据混淆矩阵看到，模型对于2和4的预测效果较好均达到了70%及以上，但是在0,1和3的预测效果只有50%左右。



五、实验结果分析

5.1 模型性能

| 指标 | 朴素贝叶斯 | 随机森林 | BERT | LSTM | XGBoost |
|-------|-------|------|-------|-------|---------|
| 准确率 | 61% | 63% | 66.6% | 62.1% | 61.9% |
| 精确度 | 60% | 74% | | | 64% |
| 召回率 | 61% | 73% | 65% | | 65% |
| F1 分数 | 60% | 74% | 66% | | 62% |

1. 朴素贝叶斯 提供了一个快速、可解释的基线。
2. 随机森林 通过学习非线性关系提高了性能。
3. XGBoost
4. LSTM
5. BERT 通过利用预训练嵌入和上下文理解，准确率明显优于简单模型。

六、讨论

6.1 预处理

高质量的文本预处理可以提高了模型的准确性。

6.2 词的向量化方法比较

在这应用这三种方法后，我们使用一些未优化的简单模型对数据进行了建模分类，并使用准确率评估模型的性能，藉此衡量向量化方法的优劣。得到的结果如下：

| 向量化方法 | 模型 | 准确率 | Kaggle得分 |
|--------------|---------------------|------|----------|
| Bag-of-Words | MultinomialNB | 0.62 | 0.59 |
| TF-IDF | MultinomialNB | 0.62 | 0.59 |
| Word2Vec | Logistic Regression | 0.51 | 0.48 |

我们发现词袋模型和TF-IDF向量化方法在准确率上有着相似的效果，Kaggle得分差异也不大。Word2Vec作为更复杂的模型在验证集准确率上却表现不佳，可能是因为Word2Vec模型需要大量的训练数据才能得到较好的效果，也可能是我们没有对其参数进行优化。

6.3 模型比较

- 较简单的模型适用于快速迭代或资源有限的环境。
- BERT 展示了最先进的性能，但需要更多的计算资源，且微调困难耗时。

难点

- 微调 BERT 的计算成本。
- 处理文本中的边缘情况，如讽刺和模棱两可的表达。
- 模型需要大量的训练数据才能达到预期的性能。
- 各种模型都表现出一定程度的过拟合，具体为在验证集上得分可能很高，但在测试集上得分很低。

未来的工作

1. 探索其他预训练模型，如 RoBERTa 或利用特定领域的数据对 BERT 进行微调。
2. 处理文本中的边缘情况，如讽刺和模棱两可的表达。

七、结论

- 朴素贝叶斯和随机森林对建立基线非常有效。较简单的模型适用于快速迭代或资源有限的环境。在快速原型或低资源设置中使用更简单的模型。
- BERT 是性能最好的模型，非常适合需要高准确性的生产场景。在计算资源不受限制的应用中部署 BERT。
- 情感分析是一个复杂的任务，需要考虑许多因素，如文本长度、情感强度等。

附录

小组成员及分工

| 姓名 | 学号 | 院系专业 | 分工 |
|-----|------------|-------|------------|
| 卜一凡 | 2200012137 | 生信-大三 | BERT及报告 |
| 韩嘉琪 | 2200012126 | 生信-大三 | XGBoost及报告 |

| 姓名 | 学号 | 院系专业 | 分工 |
|-----|------------|-------|------------|
| 张屹阳 | | 生科-大三 | EDA、报告梳理 |
| 丁健 | 2300016653 | 信管-大二 | 文本向量化、报告草稿 |
| 李思润 | 2300012651 | 地空-大二 | 随机森林及报告 |
| 耿子喻 | 2400013212 | 信科-大一 | LSTM及报告 |

实验环境设置

- **硬件**：大部分代码在本地运行，普通的高性能笔记本配置都可以满足实验需求，但部分代码需要在 GPU 上运行，因此需要准备 GPU 环境。XGBoost和BERT模型在实验室服务器上运行。
- **软件**：python >=3.11, jupyter botebook
- **库依赖版本**：
numpy==2.0.0 pandas==2.2.2 matplotlib==3.8.0 seaborn==0.13.2 nltk==3.9.1 spacy==3.8.2 scikit-learn==1.4.2 gensim==4.3.3 tensorflow==2.15.0 transformers==4.46.3 xgboost==1.5.1 optuna==4.1.0 torch==2.5.1

完整代码

详见“final_report.ipynb”文件。

参考文献

1. [1] Will Cukierski. Sentiment Analysis on Movie Reviews. <https://kaggle.com/competitions/sentiment-analysis-on-movie-reviews>, 2014. Kaggle.
2. [2] Pang and L. Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In ACL, pages 115–124.
3. [3] Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Chris Manning, Andrew Ng and Chris Potts. Conference on Empirical Methods in Natural Language Processing (EMNLP 2013).
4. [4] BabyGo000. 【Python数据分析】文本情感分析——电影评论分析（二）文本向量化建立模型总结与改进方向. <https://www.cnblogs.com/gc2770/p/14929162.html>, 2021, 博客园
5. [5] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD (arXiv). [arXiv:1901.03862]