# SentimentAnalysis

## 模型概述

**1. 数据加载与预处理**

* **数据加载**：代码首先从Google Drive加载三类情绪标签的数据（积极、中立、消极），这些数据可能包含了文本和对应的情绪标签。
* **标签分配**：数据被加载后，代码为每种情绪文本分配了一个数值标签（积极为1、中立为0、消极为2）。
* **合并数据集**：三个不同情绪的数据集合并为一个数据框架。
* **随机化数据**：为了防止模型在学习时的偏差，数据集被随机化处理。
* **文本预处理**：使用**jieba**进行中文分词，并通过从网络上下载的停用词列表去除停用词。

**2. 模型构建与训练**

* **词向量模型**：代码利用**gensim**库中的**KeyedVectors**加载预训练的词向量模型，将文本转换为数值型数据，以供深度学习模型使用。
* **模型训练**：使用了多种深度学习模型（LSTM、CNN、RNN），对数据进行训练。这些模型通常用于处理序列数据，适合文本分析任务。

**3. 性能评估**

* **性能指标**：使用准确率、召回率和F1得分来评估模型性能。这些指标能有效地衡量模型对于各类情绪标签的分类效能。

**4. 用户交互界面**

* **交互界面**：代码使用**ipywidgets**创建了一个简单的用户交互界面，用户可以选择模型、情感类别和语句数量，通过按钮触发事件处理函数进行查询操作，并显示模型的分类结果和性能指标。

### 可采用的优化方法

**1. 超参数调整**

超参数调整是优化算法中的一个重要部分，它涉及到选择最合适的参数来提高模型的表现。常见的超参数包括学习率、批量大小、训练的轮次等。可以使用以下方法进行自动化的超参数优化：

* **网格搜索（Grid Search）**：系统地遍历多种参数的组合，找到最优的组合。
* **随机搜索（Random Search）**：在参数空间中随机选择参数，这种方法比网格搜索更快，尤其是在参数空间很大时。
* **贝叶斯优化**：使用贝叶斯方法来选择参数，优化过程中考虑了之前的评估结果，通常比网格搜索和随机搜索更有效率。

**2. 特征工程优化**

优化数据的输入，改进特征工程，可以显著提升模型性能。这包括：

* **更精细的文本预处理**：例如更好的分词方法、去除更多的无关噪声等。
* **使用高级的文本表示方法**：例如从TF-IDF转向Word Embeddings或Transformer模型（如BERT）。

**3. 模型架构优化**

改进模型的架构也是优化的一种方式。可能的优化包括：

* **试验不同的模型架构**：比如尝试不同类型的神经网络，如卷积神经网络（CNN）、循环神经网络（RNN）或注意力机制模型。
* **增加或减少网络层数**：改变模型的深度可能会改善或恶化性能，具体需要试验确定。

**4. 训练技巧**

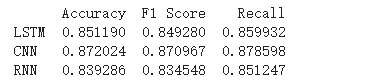
* **正则化方法**：如L1和L2正则化，可以减少模型过拟合。
* **早停（Early Stopping）**：在验证集上的性能不再提升时停止训练，以避免过拟合。
* **学习率调整**：使用学习率衰减或者适应性学习率算法（如Adam）。

**5. 使用集成学习**

通过结合多个模型的预测来改进性能，常见的集成学习方法包括Bagging、Boosting和Stacking。

（已使用的： 特征工程优化（jieba分词，去除停用词，采用腾讯预训练模型等预处理方法）、Adam优化算法、 ）

## 结果评估

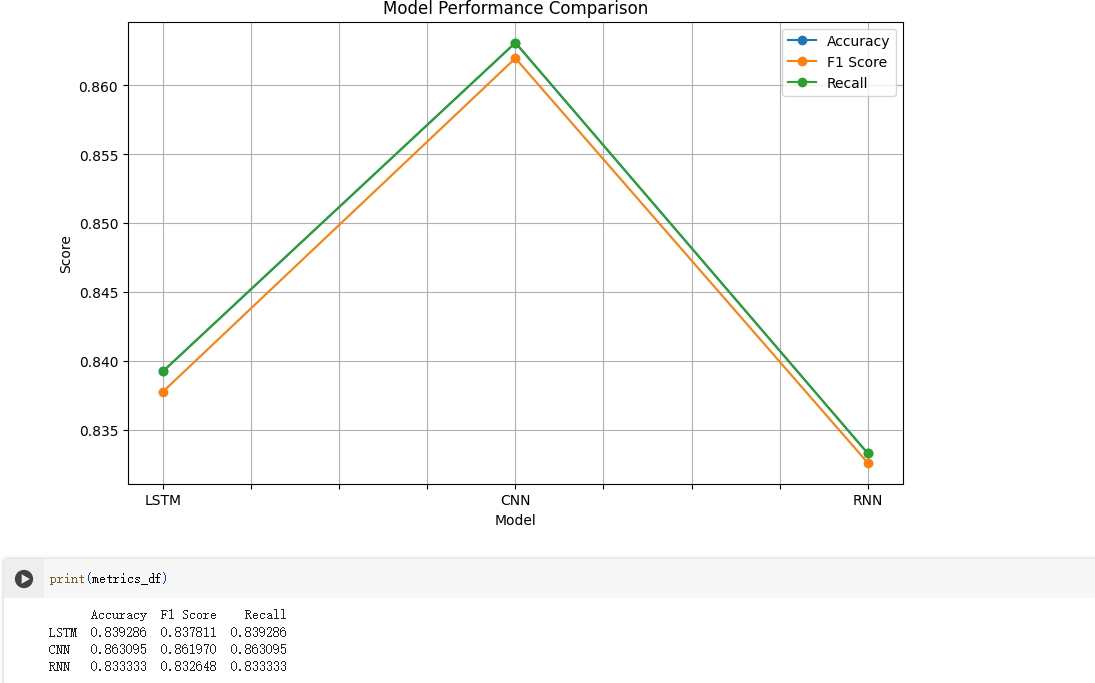


* **LSTM (长短期记忆网络)**：适合处理和预测时间序列中间隔和延迟很长的重要事件。在结果中，LSTM的表现较好，但不如CNN，这可能是因为情感分析任务中相关上下文间的依赖并不需要非常长远的记忆。
* **CNN (卷积神经网络)**：通常在图像识别领域表现突出，但也被用于文本处理，尤其是在捕捉局部相关性方面。CNN在这次的情感分析任务中表现最佳，可能是因为它能有效地捕捉文本中的局部特征（如短语或关键词）。
* **RNN (循环神经网络)**：虽然理论上适合处理序列数据，但在实际应用中，由于梯度消失或爆炸的问题，性能可能不如LSTM和CNN。

**分析结果**

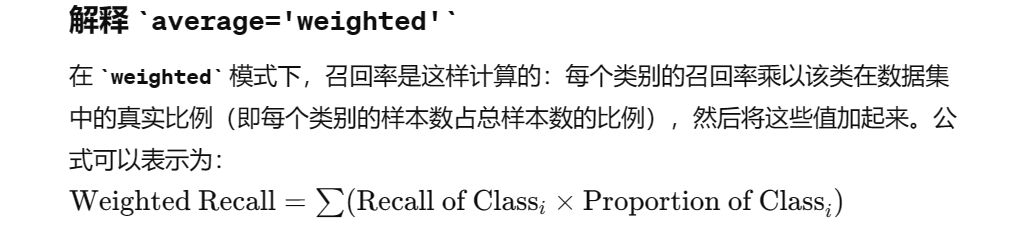
* **CNN (卷积神经网络)**
  + **准确率 (Accuracy)**：87.2024%，这是三种模型中最高的，表明CNN在正确分类情感的能力上最为出色。
  + **F1得分 (F1 Score)**：87.0967%，这反映了精确率和召回率的平衡，同样是三种模型中最高。
  + **召回率 (Recall)**：87.8598%，这表明CNN模型在识别正类（即正确情感）的能力上非常强。
* **LSTM (长短期记忆网络)**
  + **准确率**: 85.1190%，表现良好，但低于CNN。
  + **F1得分**: 84.9280%，较高，显示了对不同类别的良好平衡。
  + **召回率**: 85.9932%，较CNN低，表明其在避免遗漏正类方面略逊一筹。
* **RNN (循环神经网络)**
  + **准确率**: 83.9286%，为三者中最低，但仍表现不错。
  + **F1得分**: 83.4548%，表明精确率与召回率之间的平衡略低。
  + **召回率**: 85.1247%，尽管RNN的整体准确率较低，其召回率还是相对较高

## 相关问题

**1. 为什么accuracy和recall相同**  


可能导致这种结果的原因：

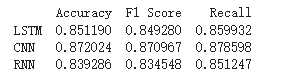
使用了 **average='weighted'** 参数来计算 **recall\_score**，这会使得召回率是根据每个类别的实际样本数量（支持度）加权的。这种计算方式可以为类别不平衡的数据集提供一个公平的召回率评估，因为它反映了每个类别在数据集中的重要性。



但是实际样本中，三种样本的数量很接近，所以使用加权计算导致accuracy和weight相同

更改为后macro后，用于类别平衡的情况

* **average='macro'**：计算每个类别的召回率，然后求其未加权平均（即各类别召回率的算术平均，不考虑各类的支持度或实例数）。这通常用于类别平衡的情况。



**2.训练集和验证集相关问题**

使用了 **train\_test\_split** 方法来划分训练集和测试集（通常作为验证集使用）。在代码中，设置了 **test\_size=0.2** 和 **random\_state=42**，这意味着数据的 20% 被保留作为验证集（测试集），80% 的数据用于模型的训练。**random\_state** 确保了结果的可重复性。

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

构建了 LSTM, CNN, 和 RNN 三种不同的模型，并为每个模型进行了训练。在模型训练过程中，使用了验证数据 **X\_test, y\_test** 来监控训练过程并评估模型在未见过数据上的性能。

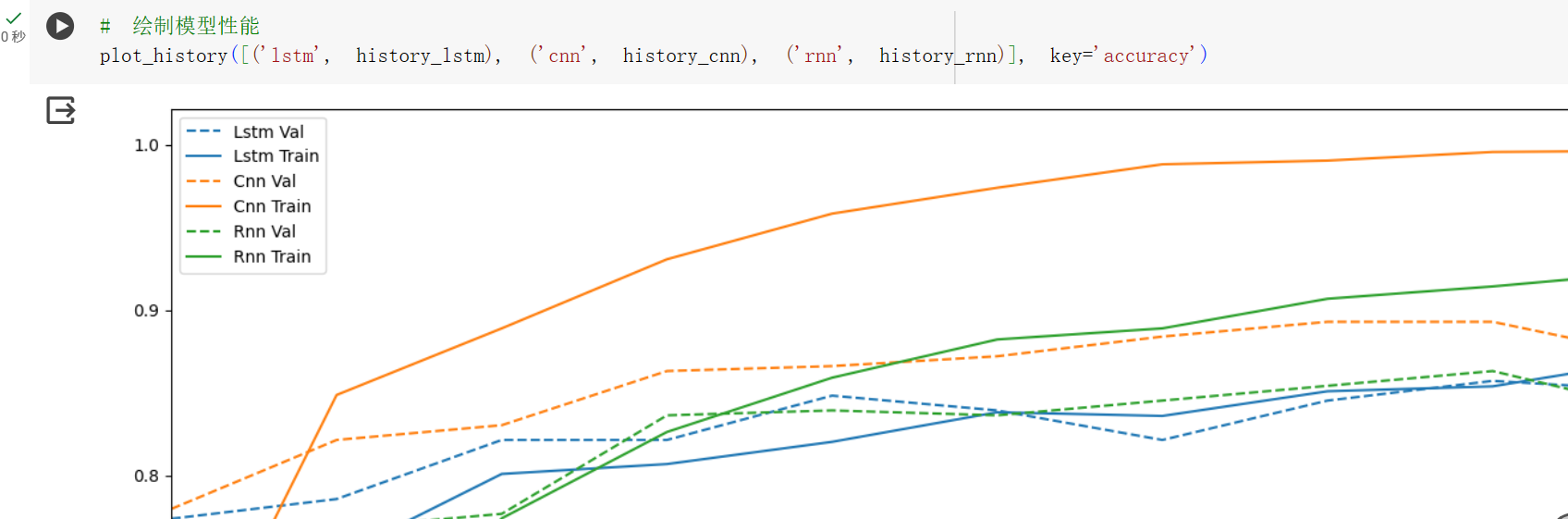
history\_lstm = lstm\_model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test), batch\_size=64)

history\_cnn = cnn\_model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test), batch\_size=64)

history\_rnn = rnn\_model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test), batch\_size=64)

（validation\_data为验证集）

这个图是模型在训练过程中，精确度accuracy的在训练集和验证集的变化 val是验证集，train是训练集



这个图同上，只不过变化的不是体现的不是精确度，而且损失函数（损失函数的图像通常是通过在训练过程中记录每个训练批次（batch）或每个训练周期（epoch）的损失值来生成的）

