# 环境要求

本次实验中用到的IDE：Dataspell，Pycharm

本次实验的系统：windows11

pandas == 2.1.0

scikit-learn == 1.3.0

numpy == 1.23.5

torch == 2.0.1

keras == 2.12.0

matplotlib == 3.7.2

# 使用方法

pip install pandas

pip install scikit-learn

pip install torch

pip install numpy

pip install keras

pip install matplotlib

python MLP.py

python SVM.py

python TextCNN.py

python 决策树.py

python 逻辑回归.py

# 实验过程

## 实验目的

在本次实验中，我们对给定的数据集利用逻辑回归、SVM、MLP、决策树等模型进行文本分类，将给定对象归入一个或者多个类别，便于后期进行标签标注以及其他应用。

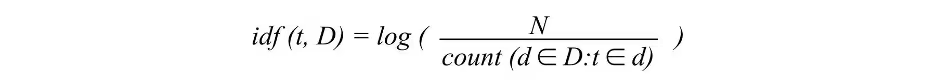
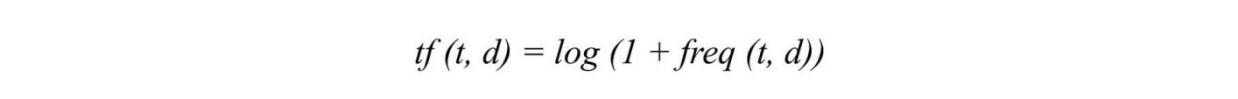
## 实验思路

本次实验的实验思路大致分为七步：

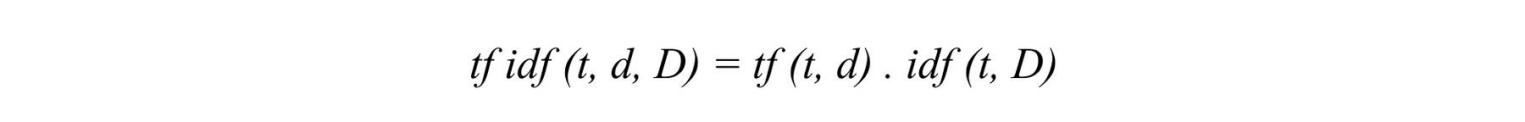
1. 数据集预处理：第一步是对给定的数据集进行预处理，包括读取数据集和对文本进行一些基本处理——包括去除不必要的特殊字符、标点符号等。
2. 数据划分：将数据集划分为训练集和测试集。
3. 特征工程：使用适当的方法将文本转换为可供逻辑回归模型使用的特征向量。将原始数据集被转换为用于训练机器学习模型的平坦特征。本次实验中用到的特征提取方法为TF-IDF。
4. 模型训练：使用逻辑回归/SVM/MLP/决策树/TextCNN算法对训练集进行训练。
5. 模型评估：使用测试集评估训练好的逻辑回归模型的性能。本次实验中的评估指标包括准确率、精确率、召回率和F1分数。
6. 模型应用：在模型经过优化并且达到满意的性能后，将其应用到test.txt中进行分类输出结果。
7. 模型优化：根据评估结果，对模型进行优化。在本次实验中通过调整参数、特征提取方法或模型选择，以提高模型的性能。

## 3.3 实验方法与代码实现

首先介绍一下本次实验中采取的特征工程以及训练集划分方法。本次实验中用到的特征提取方法为***TF-IDF。***

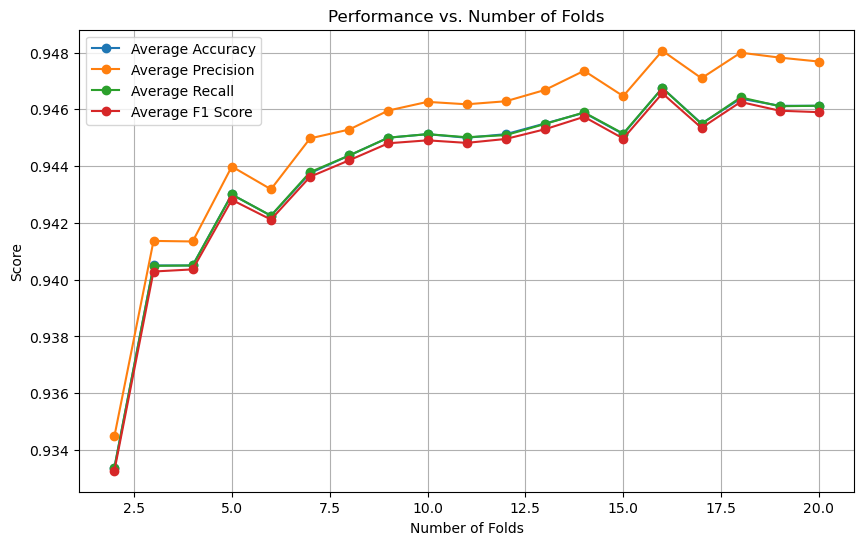


TF 为我们提供了有关术语在文档中出现的频率的信息，而 IDF 为我们提供了有关术语在文档中的相对稀有性的信息。通过将这两个值相乘，我们可以获得最终的 TF-IDF 值。

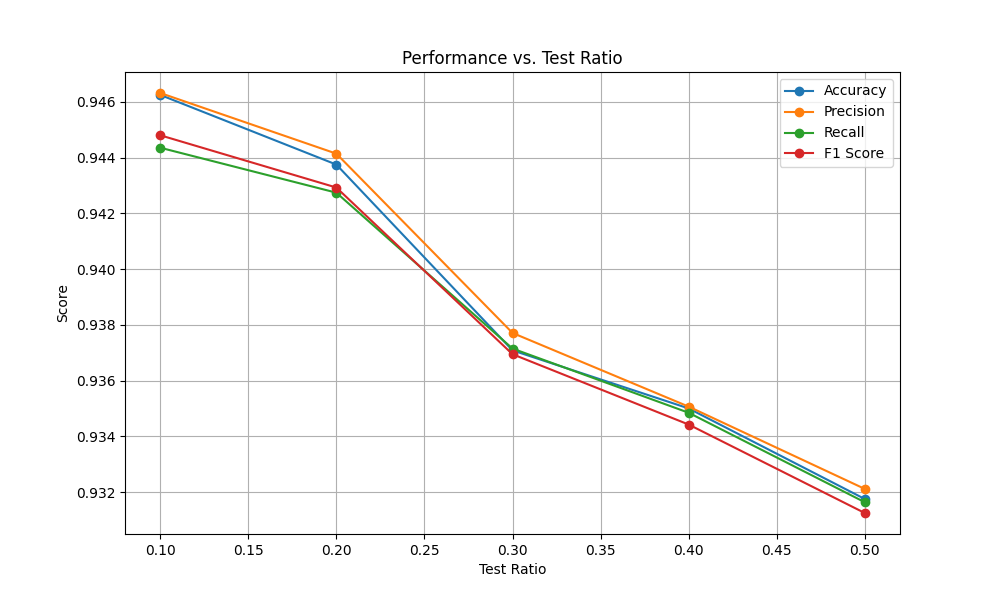


在训练集和测试集的划分上，由于无法确定哪种方式的效果更好，我分别测试了固定划分和多折交叉验证对于模型的准确率、精确率、召回率和F1分数的影响。

在代码***划分比例测试.py***与***折数测试.py***中，我通过循环验证数值列表中的取值，画图显示出模型的四种特征指数随数值变化的曲线，结果如下（以逻辑回归为例）：



多折交叉验证



固定划分

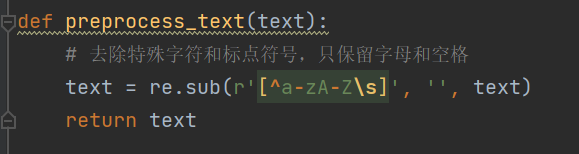
可以看出，多折交叉验证在折数为16的时候效果最好，整体的评价指标都略好于利用固定划分分割训练集和测试集。所以在后续的建模中我都采取了折数为16的多折交叉验证划分训练集和测试集。

### 3.3.1 逻辑回归模型（Logistic Regression）

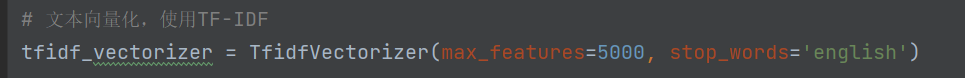
逻辑回归是一种经典的二分类算法，在本次的十分类问题中，可以简单理解为逻辑回归进行了十次二分类，但逻辑回归容易受到异常值的影响，而且只适用于线性可分的问题。

之所以选择逻辑回归模型，是因为逻辑回归是一个相对比较简单的模型，但可解释强，而且非常擅长进行二分类，相对于其它复杂的模型而言计算效率比较高。接下来讲一下重点的代码逻辑（因为代码过长就不再全篇幅复制到实验报告中）

1. 定义函数preprocess\_text()，用于去除特殊字符和标点符号，只保留字母和空格。（在后期检验的过程中发现一些换行符以及其他的符号也被考虑进特征中，为了保证结果的正确性，后期添加了该函数并对数据进行处理）

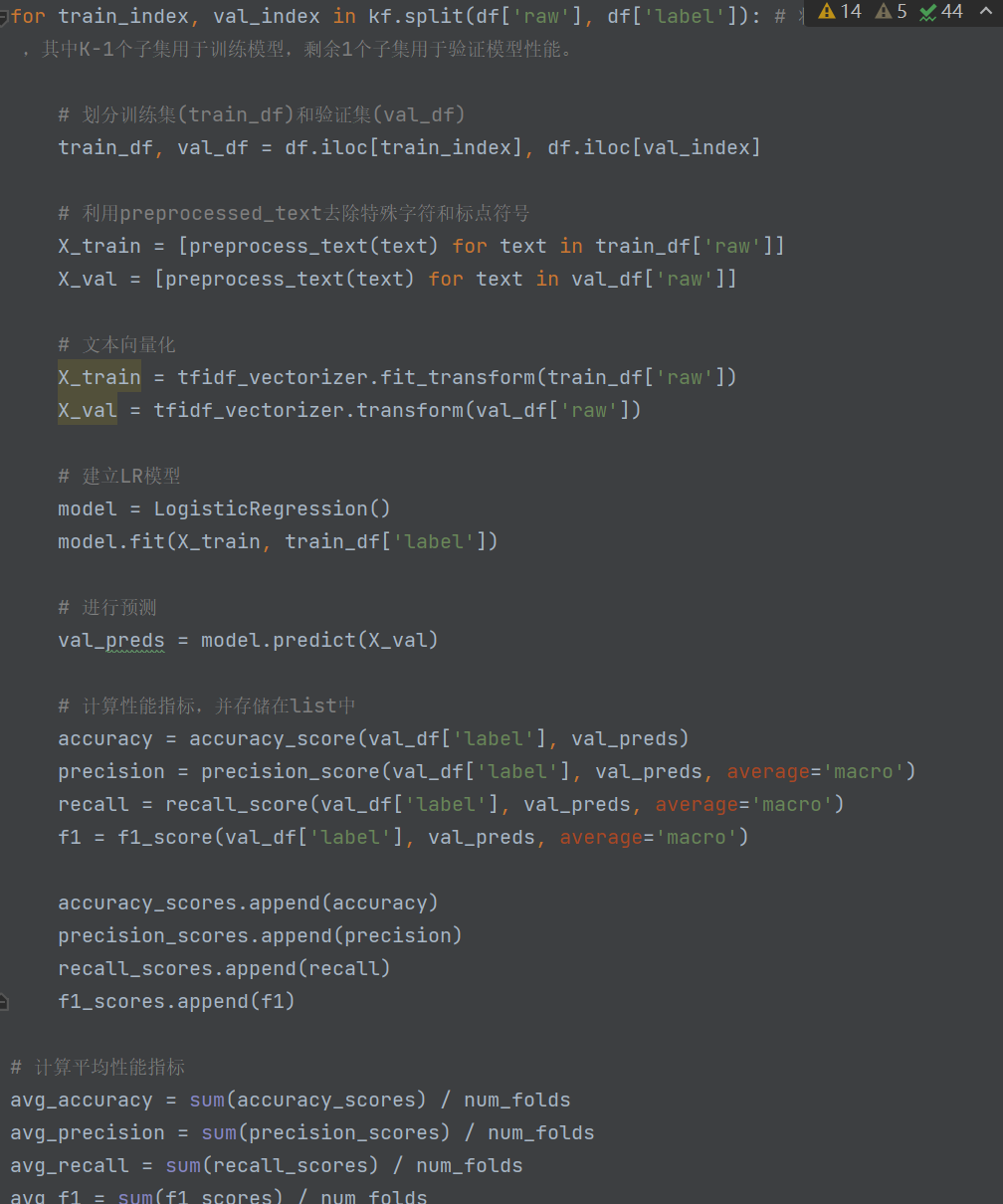
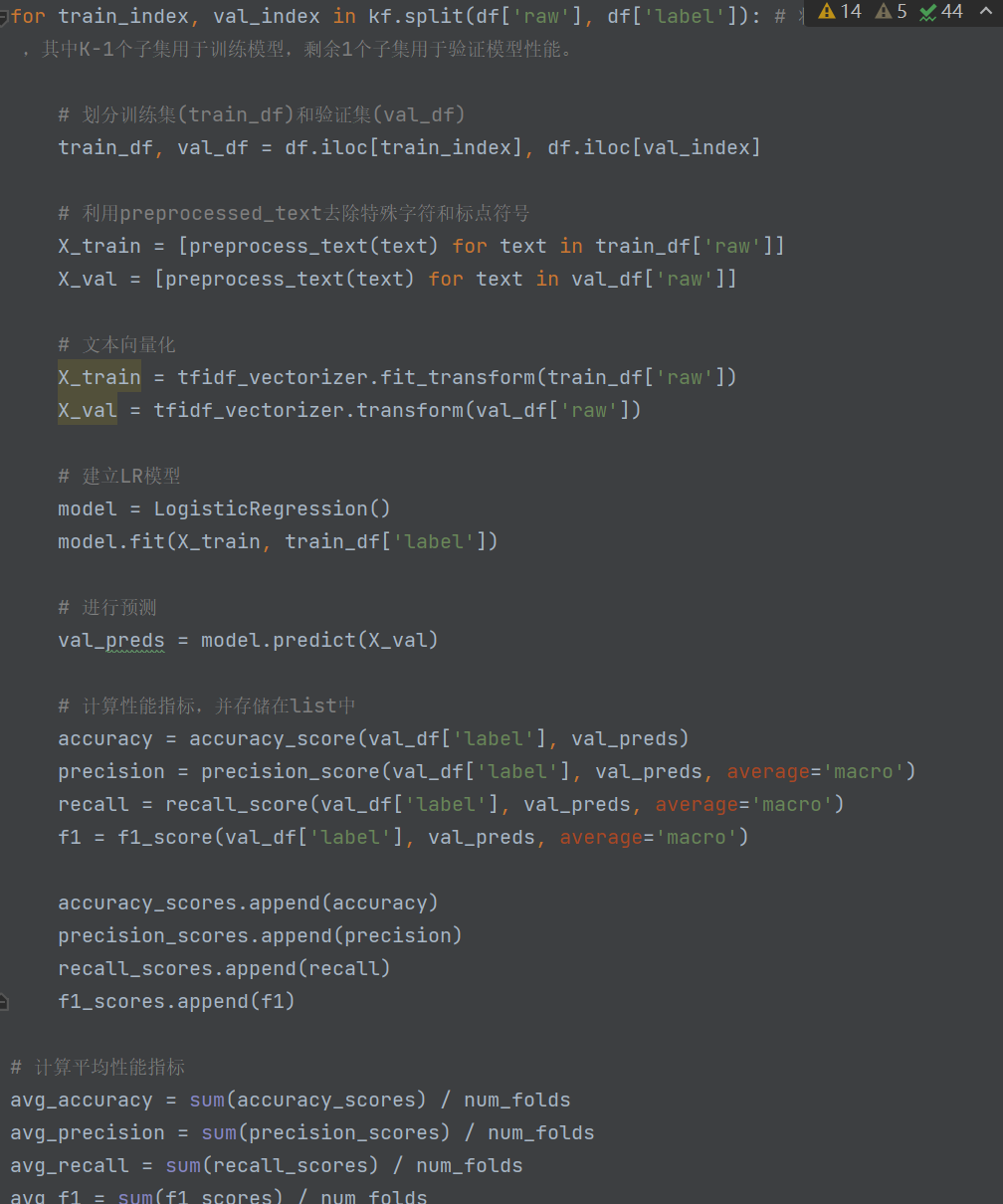


1. 文本向量化：使用TF-IDF将文本数据转换为数值特征向量。



1. 循环交叉验证：对每个折叠进行循环验证

* 划分训练集和验证集：根据当前折叠的索引，将数据划分为训练集和验证集。
* 文本预处理：对训练集和验证集的文本进行预处理，去除特殊字符和标点符号。
* 文本向量化：使用TF-IDF将训练集和验证集的文本转换为特征向量。
* 模型训练和预测：使用逻辑回归模型对训练集进行训练，并在验证集上进行预测。
* 计算性能指标：计算当前折叠的准确率、精确率、召回率和F1分数，并将其存储到性能指标列表中。
* 计算平均性能指标：循环结束后，计算所有折叠的平均准确率、精确率、召回率和F1分数。



### 3.3.2 支持向量机模型（SVM）

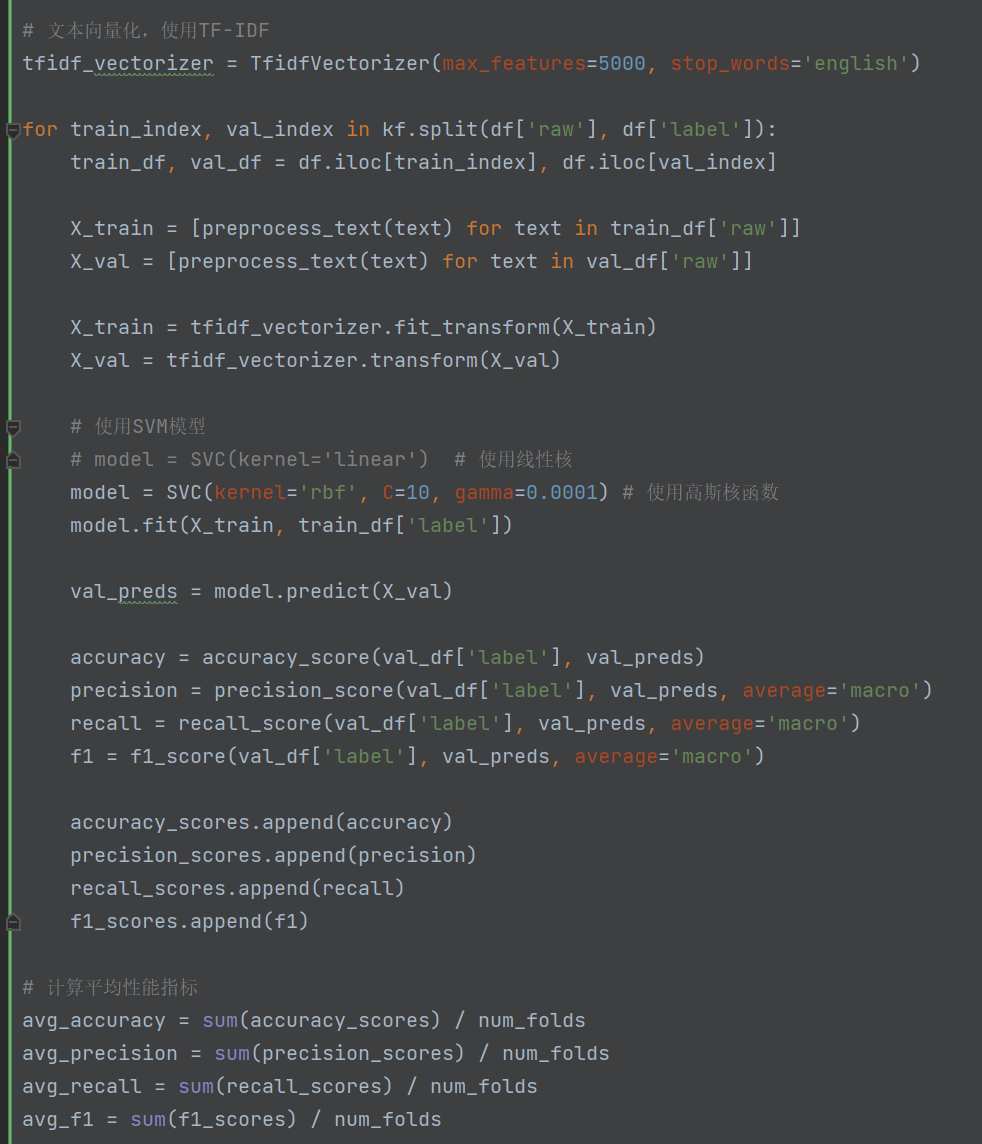
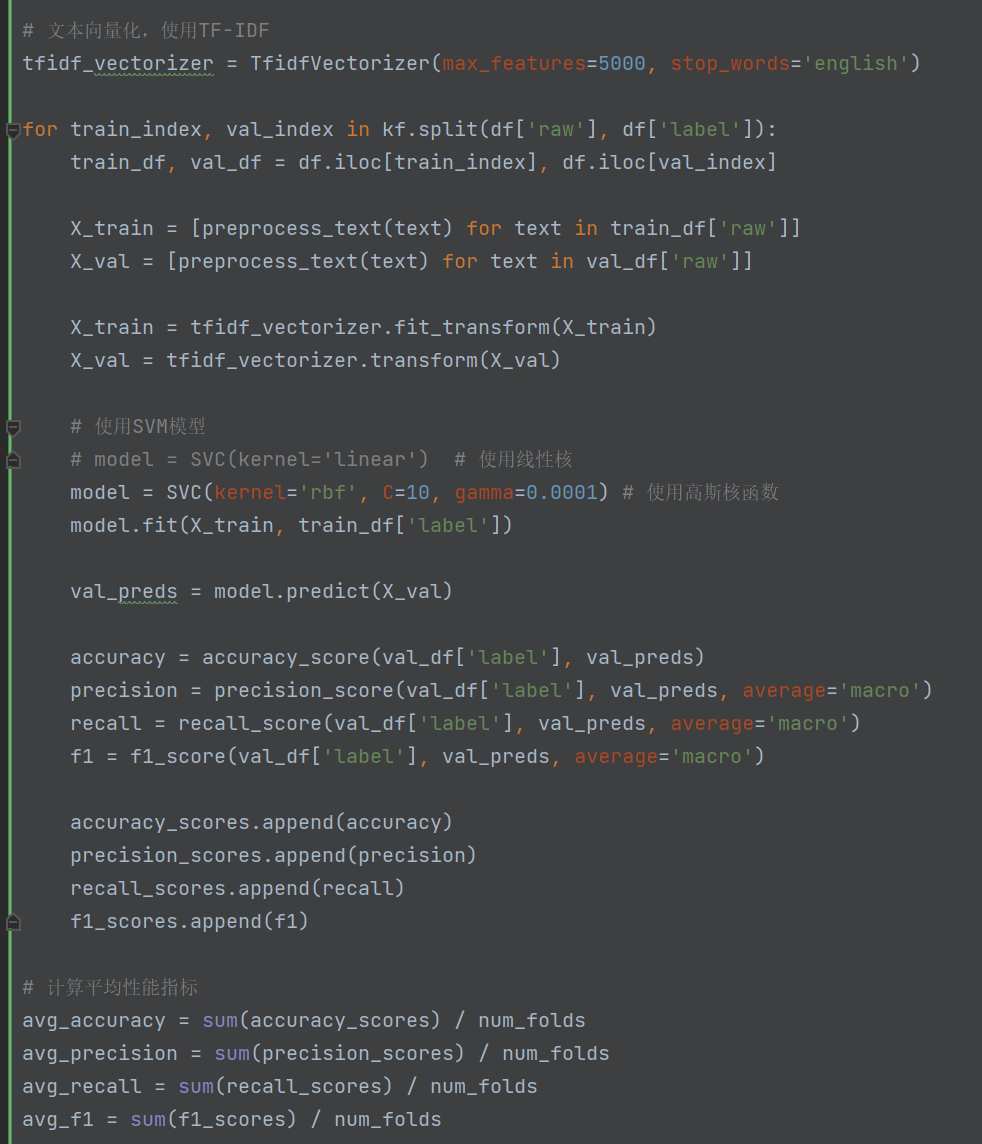
支持向量机（SVM）是一种常用的分类算法，在文本分类任务中也被广泛应用。除了本次实验要求使用SVM以外，以下是我认为应该使用支持向量机进行文本分类的几个原因：

1. 在文本分类中，特征空间往往是高维稀疏的，SVM通过寻找一个最优的超平面来分隔两个类别，能够有效地处理高维特征空间下的分类问题。
2. 第一种方法逻辑回归是一种线性分类，而SVM通过核函数的引入，可以将低维的非线性可分问题映射到高维特征空间中进行线性分类。这使得SVM能够更好地处理非线性文本分类问题。假如LR模型不能很好的进行分类，那么可能的原因就是因为数据集并不适用于线性分类，我们也就需要使用非线性分类进行本次实验。
3. 从我个人观点上来说，我认为本次实验的数据集是比较小的数据集，而SVM其基于最大化间隔的原则，对于样本量少的情况能够提供较好的分类效果。

接下来解释一下SVM代码中的重要部分：

1. 循环交叉验证：对每个折叠进行循环验证

* 划分训练集和验证集：根据当前折叠的索引，将数据划分为训练集和验证集。
* 文本预处理：对训练集和验证集的文本进行预处理，去除特殊字符和标点符号。
* 文本向量化：使用TF-IDF将训练集和验证集的文本转换为特征向量。
* 模型训练：创建SVC对象（SVM模型），使用线性核（不采用核函数）以及高斯核函数，并使用训练集训练模型。
* 预测并计算评价指标：对验证集进行预测，并计算性能指标（准确率、精确率、召回率、F1分数）。
* 循环结束后，计算平均性能指标。



### 3.3.3 多层感知机模型（MLP）

除了本次实验要求使用MLP以外，我认为多层感知机（MLP）模型适用于文本分类任务的原因如下：

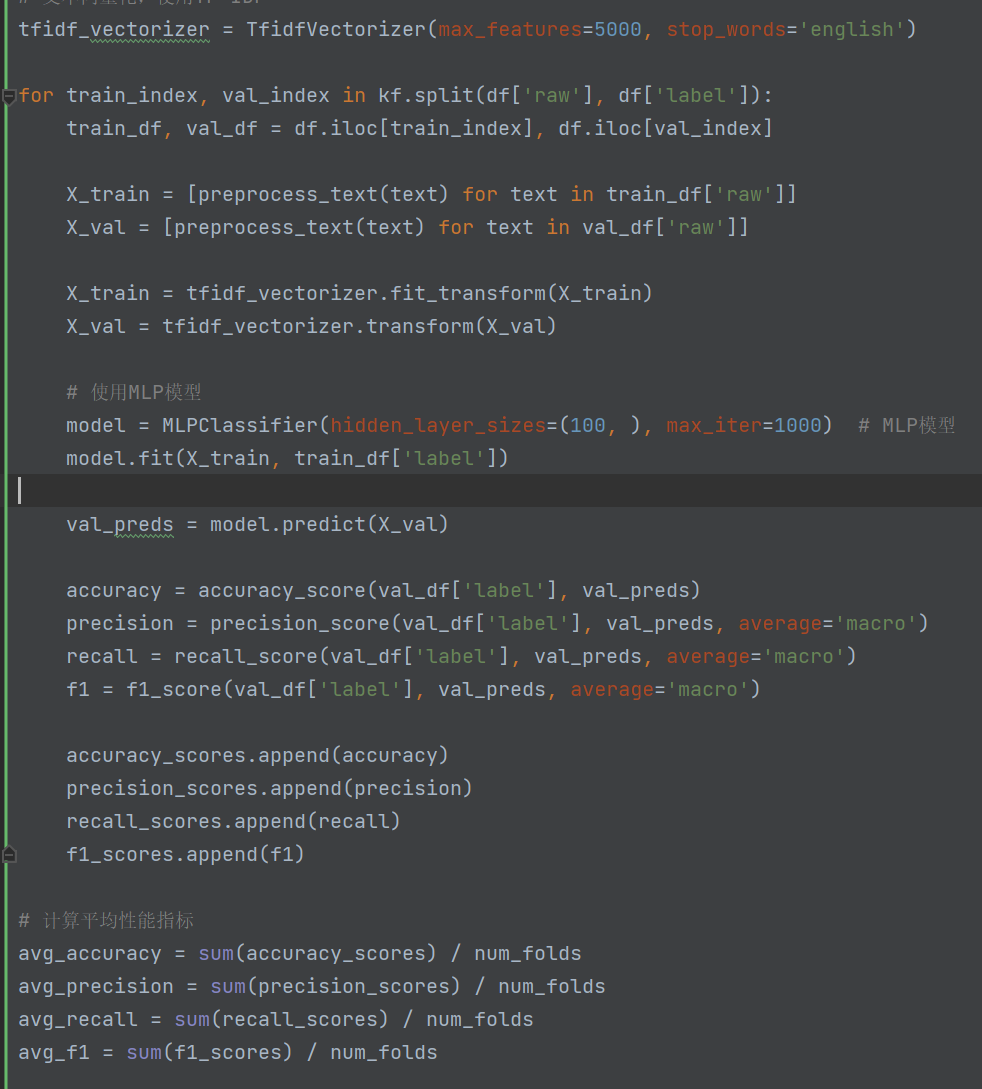
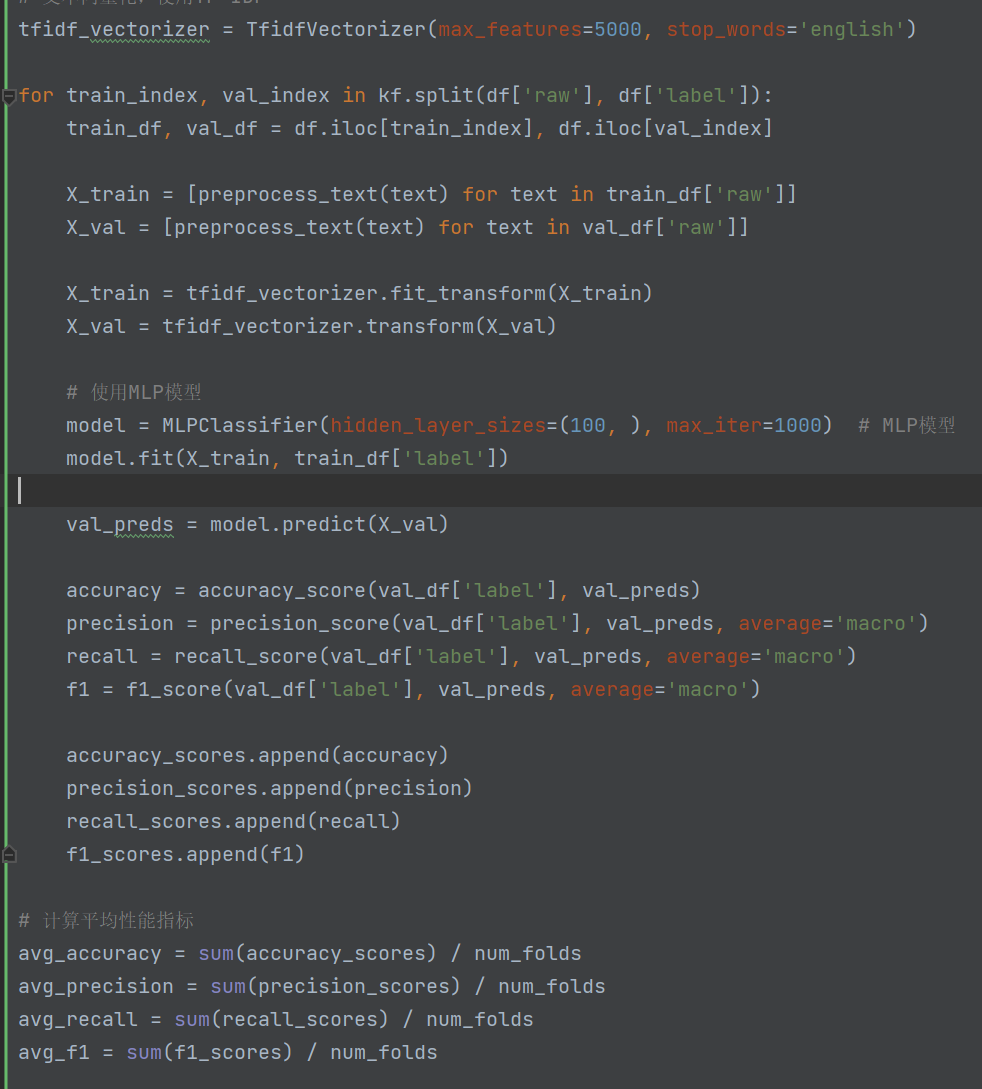
1. 同SVM，MLP可以通过非线性激活函数将输入进行非线性映射，学习和捕捉文本数据中的复杂非线性关系，从而提高文本分类的准确性。

2. MLP 可以自动学习文本数据的高级特征表示。MLP 可以逐渐提取出不同抽象层次上的特征，从局部特征到更抽象的语义特征。这些特征表示有助于模型更好地理解文本内容并进行分类。

接下来解释一下MLP代码中的重要部分：

1. 循环交叉验证：对每个折叠进行循环验证

* 划分训练集和验证集：根据当前折叠的索引，将数据划分为训练集和验证集。
* 文本预处理：对训练集和验证集的文本进行预处理，去除特殊字符和标点符号。
* 文本向量化：使用TF-IDF将训练集和验证集的文本转换为特征向量。
* 模型训练：创建一个MLPClassifier对象model，并使用训练集进行训练
* 预测并计算评价指标：对验证集进行预测，并计算性能指标（准确率、精确率、召回率、F1分数）。



### 3.3.4 决策树模型

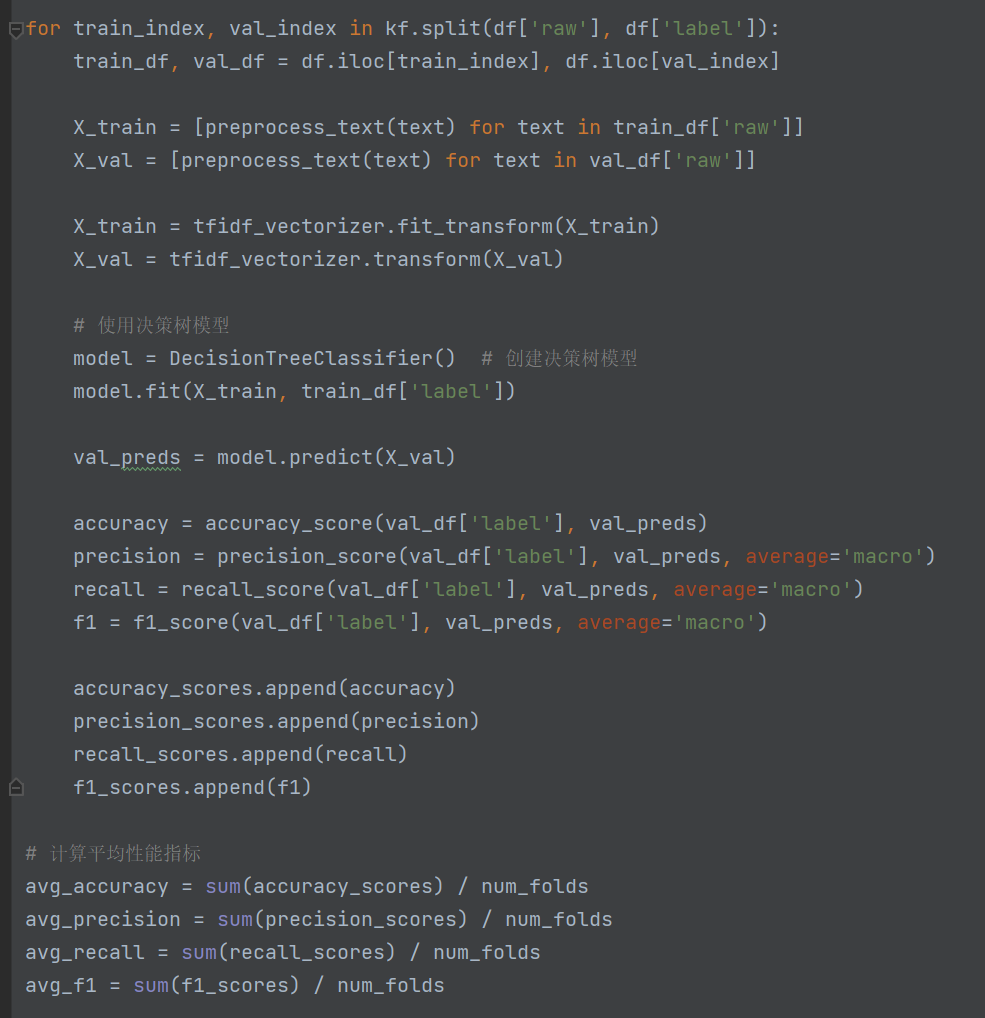
决策树是一种基于树状结构进行决策的机器学习模型，我认为决策树应用在文本分类任务中被广泛应用的原因有如下几种：

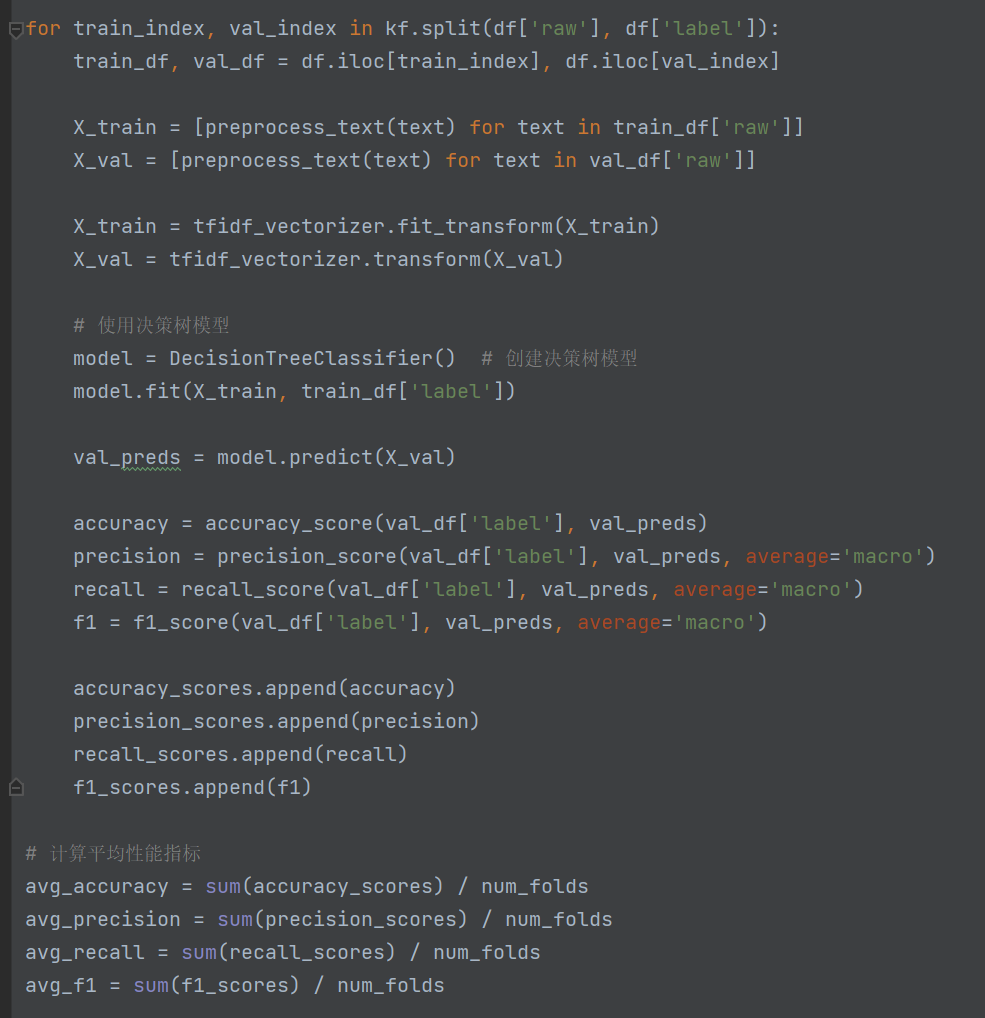
1. 决策树模型具有很强的可解释性。决策树模型的决策可以轻松地转化为可理解的条件语句，便于观察和理解（由于生成过程过长，而且文件过大，电脑吃不消，所以没有生成可视化的决策树）

2. 决策树可以处理高维特征：文本分类通常涉及大量的特征，其中每个特征表示一个词汇或者词组。决策树模型能够有效处理高维特征空间，并自动选择重要的特征。它通过逐步分割训练数据并考虑每个特征对分类的贡献，从而找到最佳的分类规则。

3. 决策树模型可以通过集成方法（如随机森林和梯度提升树）进行扩展，从而提高预测性能。这些方法通过结合多个决策树的预测结果来取得更好的分类效果，适用于大规模的文本分类任务。

决策树模型的逻辑和上面很像，就不多加赘述了。附上关键代码：





### 3.3.5TextCNN模型

TextCNN模型是我在网上进行资料搜索后找到的适用于文本分类的模型。TextCNN是一种基于卷积神经网络的文本分类模型。它可以有效地处理自然语言中的文本数据，并在大规模文本分类任务上取得了优秀的表现。

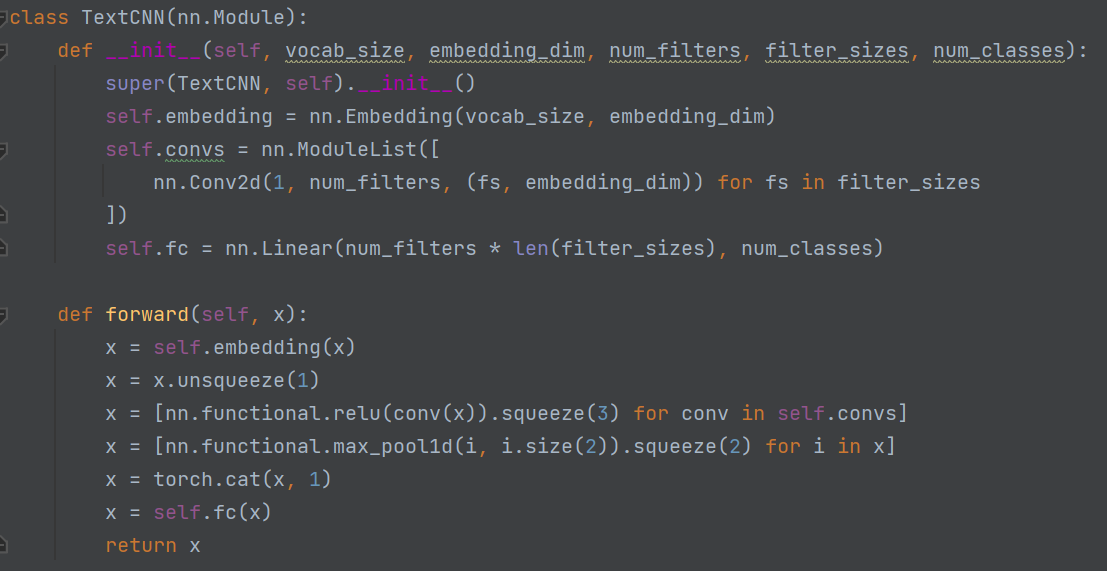
TextCNN模型的输入是一个句子或一个文档，首先将文本数据表示成一个固定长度的词向量序列，然后通过卷积神经网络进行特征提取和分类。具体来说，TextCNN模型中包括以下几个主要组件：嵌入层、卷积层、池化层、全连接层（具体含义由于篇幅限制不在报告中体现）。

以下是代码的详细解释：

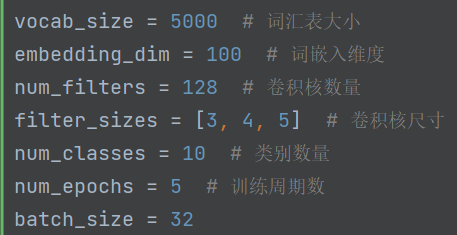
1. MAX\_SEQUENCE\_LENGTH 是定义的一个常量，表示文本序列的最大长度，用于填充和截断文本。



3. 定义了PyTorch模型类。这个模型是一个卷积神经网络，用于文本分类任务。它包含了嵌入层、卷积层和全连接层，其中嵌入层用于将文本序列转换为词嵌入向量，卷积层用于提取文本特征，全连接层用于进行分类预测。



5. 定义了一些超参数，包括词汇表大小、词嵌入维度、卷积核数量、卷积核尺寸、类别数量、训练周期数和批处理大小。



6. 使用交叉验证循环迭代，每次迭代都划分训练集和验证集，并在每个折叠上进行以下操作：

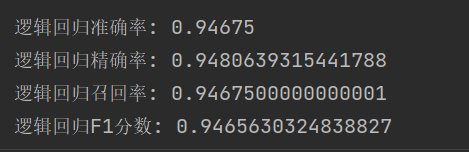
* 创建一个 Tokenizer 对象，用于将文本转换为整数序列，并构建词汇表。
* 将训练文本和验证文本转换为整数序列，并进行填充。
* 创建 PyTorch 数据集和数据加载器，用于训练模型。
* 初始化 TextCNN 模型、优化器和损失函数，然后训练模型。
* 最后评估模型的性能。

（TextCNN代码过长，由于篇幅限制，详见TextCNN.py 71-141行）

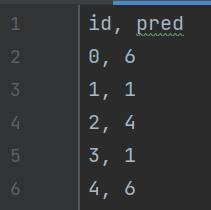
# 结果分析

所有调参过程最后的最优结果都应用在代码中了。

## 逻辑回归模型（Logistic Regression）



评价指标



预测结果

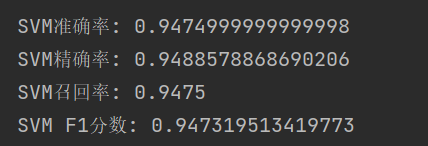
逻辑回归的结果比我预想中要好很多，因为逻辑回归是相对比较简单的模型，也不具有调参的过程。所以这样的结果是相当优秀的，我认为可能的原因有以下几点

1. 当前的数据集能够通过一条直线或超平面进行良好的二分类时，是线性可分的数据。

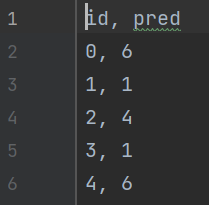
2. 当前数据的特征维度较低，逻辑回归模型可以更有效地进行参数估计和建模。

3. 当前数据的数据量比较少，逻辑回归模型学习起来比较简单。

## 支持向量机模型（SVM）



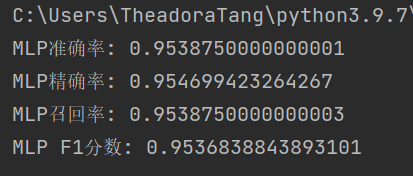
评价指标



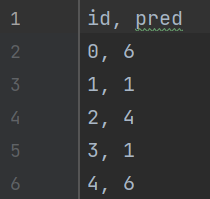
预测结果

SVM的预测效果也比较好，我后续没有对SVM进行进一步的优化以及参数选择，所以这里不多加分析。

## 多层感知机模型（MLP）



评价指标



预测结果

MLP的预测结果很好，是在没调参的情况下效果最好的一个。我认为原因有以下几点：

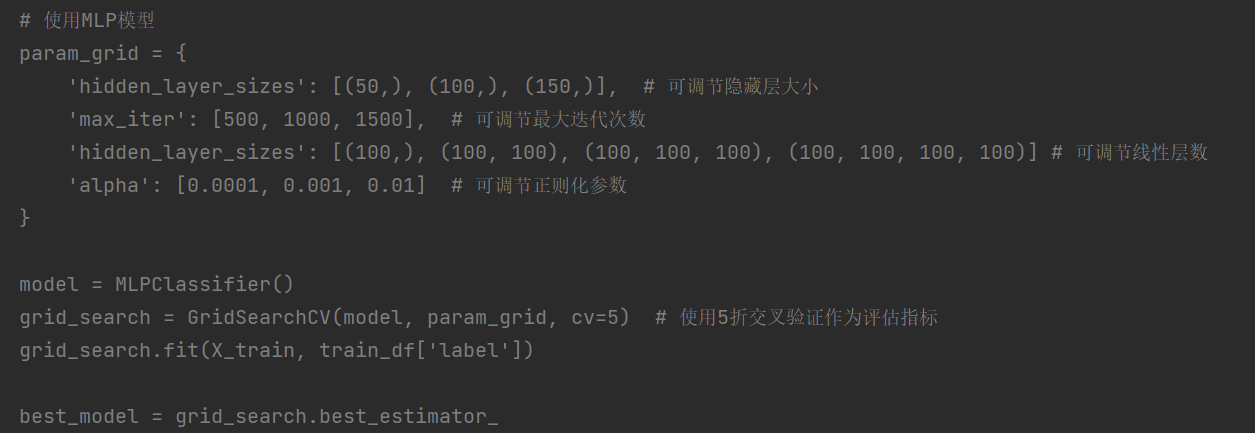
1. 可以学习到复杂的特征表示：MLP可以通过多层神经元的组合来学习到更丰富、更复杂的特征表示。

2. 具有良好的泛化性能：MLP模型具有较强的泛化性能，即对于未见过的数据也可以进行准确的分类。这是由于MLP采用了反向传播算法来训练模型，可以避免过拟合等问题。

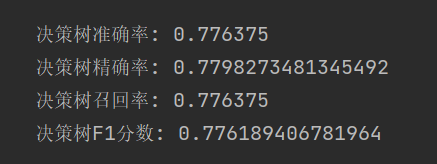
3. 使用MLP模型进行文本分类，通常可以进行端到端的学习。也就是可以直接使用原始的文本数据作为输入，并使用输出层的分类结果直接进行预测，而无需其他的预处理或特征提取步骤。本次实验中我的数据预处理是比较少的，预处理的不足也可能会导致其他模型的效果不佳，但对MLP的影响较小

因为模型表现良好，所以我又使用GridSearchCV（网格搜索）函数在给定的参数范围内搜索最佳组合。我选择了一些参数：**可调节隐藏层大小，epoch，可调节线性层数，可调节正则化参数**，然后让机器选择具有最佳性能的参数组合，并在整个训练集上拟合最终的MLP模型。

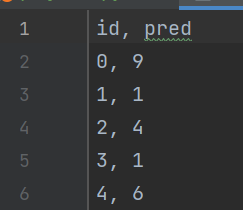
具体代码如下：



## 决策树模型



评价指标



预测结果

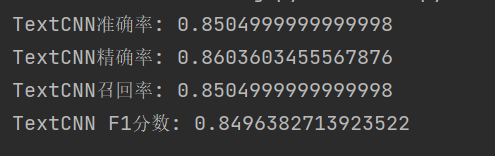
决策树的结果比较差，这个在我的意料之外。我猜想可能的原因有以下几点：

1. 缺乏全局优化能力：决策树是一种贪心算法，它通过局部最优来构建树结构，但这并不一定能够达到全局最优。在文本分类任务中，每个节点的选择只考虑当前节点的最佳特征，而没有全局优化能力。这可能导致决策树模型无法捕捉到特征之间的复杂关联和语义信息。

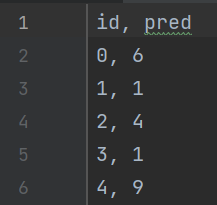
2. 过拟合：决策树模型在处理文本分类问题时容易过拟合，特别是当模型的层数增加时。决策树可能会过度关注训练数据中的噪声或局部特殊情况，而无法泛化到整个数据集。

3. 处理长文本困难：对于长文本数据，决策树模型可能无法充分利用文本中的上下文信息。由于决策树是基于一系列离散判断条件进行分割的，它可能无法捕捉到长文本中的复杂结构和连续性。

## TextCNN模型

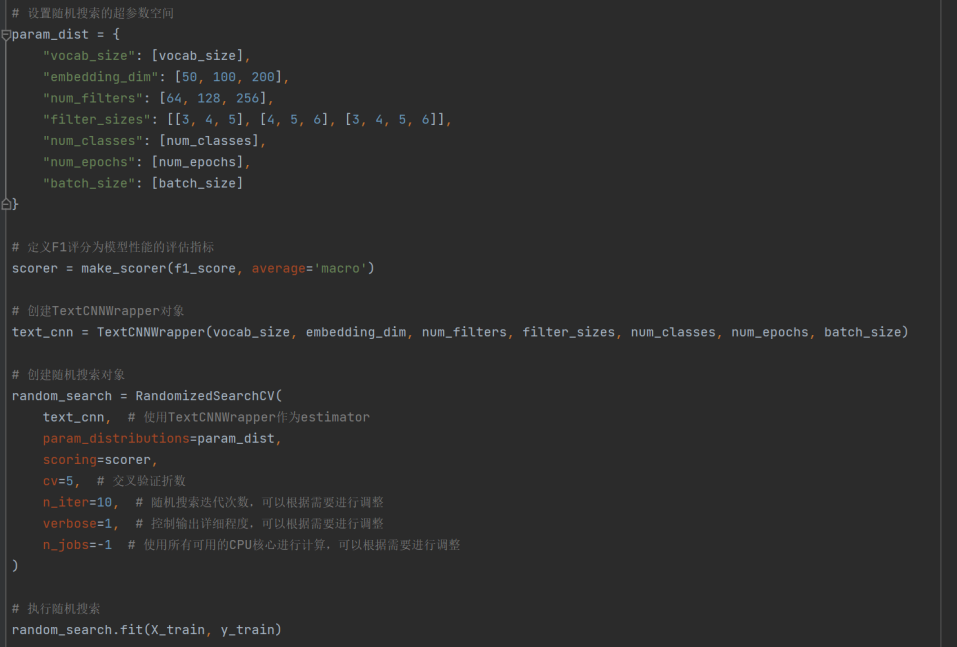


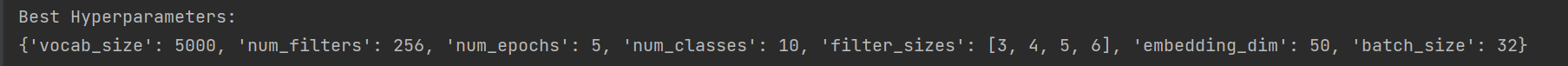
评价指标



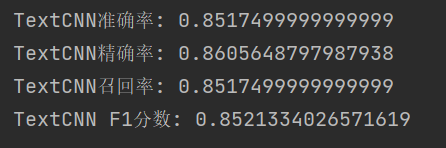
预测结果

TextCNN模型的预测结果虽然不差，但远远没有我预料中好，作为一个优秀的文本分类模型“鼻祖”，我最开始猜测TextCNN的准确率应该达到95%以上，所以我认为是我的参数选择不佳，于是我又利用随机搜索来进行超参数选择。





优化后，结果并没有大幅度的提升

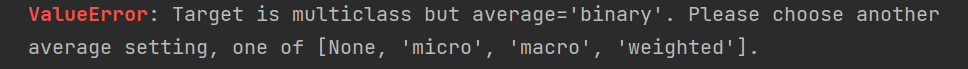


我并不认为TextCNN不适用于本次的预测，我更觉得是参数选择和数据预处理上存在问题。

# 总结与思考

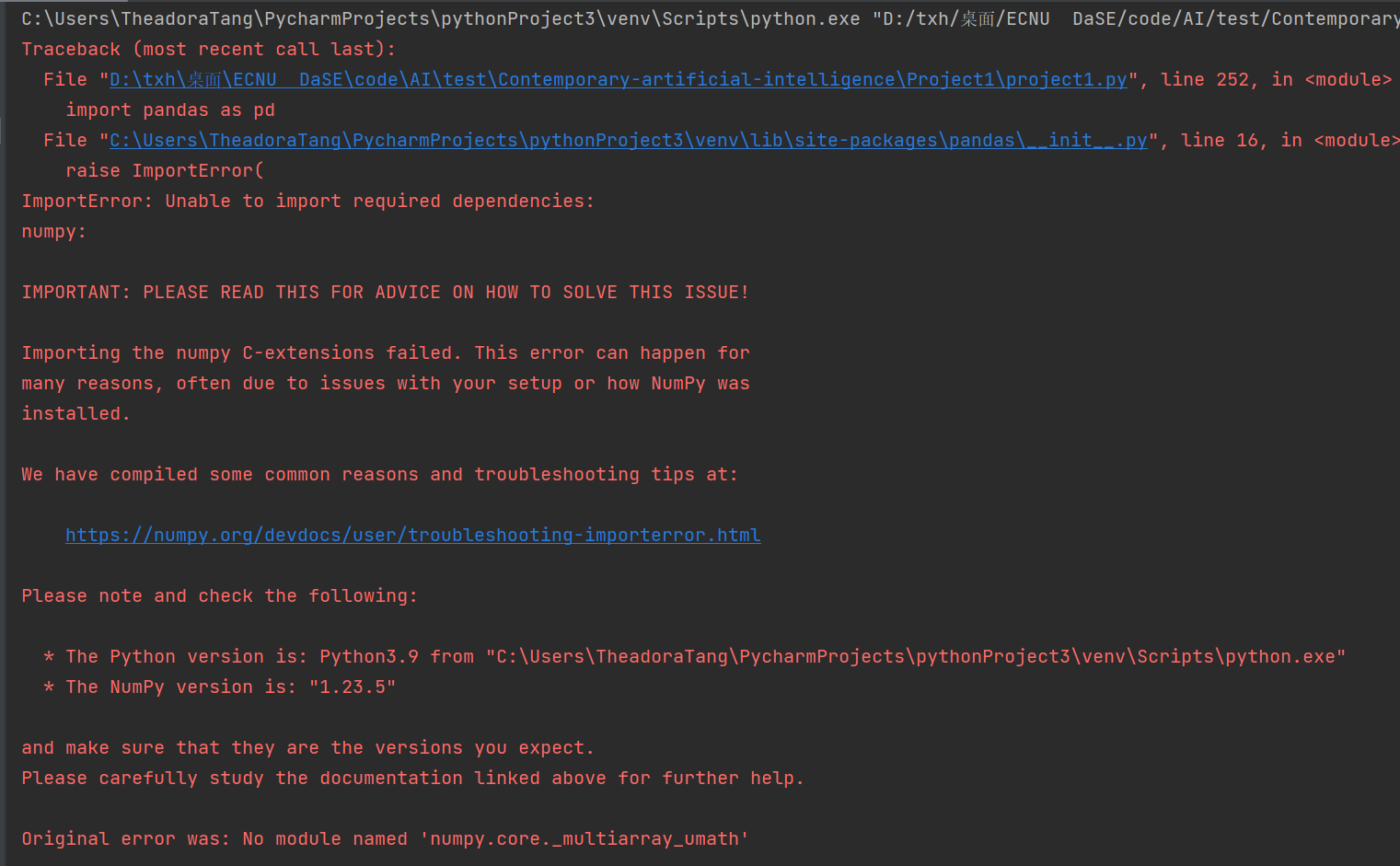
## 遇到的问题

1. 最开始预测精确率，召回率以及f1分数的时候，没有加上average=‘macro’的选项，代码运行报错。



后来查找资料得知，目前默认average值为binary，可是目前是十分类任务，binary参数值与目标数据的多类别属性不匹配，导致无法计算精确率。而调整为macro后，该函数计算每个类别的精确率，并对它们进行平均，这样就达到了我们的目的。

1. 因为运行模型中途强行终止，报错如下：



我将numpy下载、升级了多次，均没有结果。后来看到CSDN上的博客写到模型训练中遭遇强制终止可能会造成模型损坏，导致报错。所以我将该文件删除后重新复制粘贴相同的代码，问题解决。

1. 最开始读取txt文本时，我为了图省事方便直接用了pd.read\_csv（）打开txt文件，结果一直报错显示我有一些行并不只是有两个文本片段。

**ParserError: Error tokenizing data. C error: Expected 2 fields in line 3, saw 11**

在多次检查文本内容之后，我依然坚信我的代码没有问题，我选择用Excel打开txt文件一探究竟，这个时候在Excel的提示下我才想起直接使用readcsv打开txt文件会默认以逗号分割文本片段，而英文段落中有很多逗号，才造成了这场误会，之后我老老实实open txt文件，事情得以解决。

通过这个小错误，我后续为了防止其他识别文本的问题出现，将第一个逗号后的所有句子都拼接在一起（理论上就是一起的，只是防止意外），也算是进行了数据预处理。

1. 我还尝试使用了Bert模型，但是一直显示timeout，应该是我自己连的梯子或者网络有问题，无法下载预训练模型，后来放弃了。
2. 运行时间过长：本次实验除了逻辑回归外的所有模型的运行时间都非常长，这难免与使用了Kfolds之间存在联系，循环的增加让代码的时间复杂度变高，在实验结束后的反思中我也有想过，在实验刚开始时我过分追求了准确率，实际从图上来看，差距微乎其微，只有0.02左右，完全可以舍弃准确率换效率，但是工程比较庞大，最后也没有更改代码。
3. 超参数选择：但是值得反思的事，我给出的超参数候选值的量极其有限，真正的最佳参数可能不包含在这其中，尤其是TextCNN，我坚信我没能找到最好的参数。但如果将许多参数都纳入考量范围内，极大概率运行时间过长，甚至CPU带不动。
4. MLP的优化，SVM的高斯内核代码都因为时间过长没能观察到结果，甚至直接把电脑跑模型跑到蓝屏花屏。也希望老师和助教能够给予一些帮助。

## 对本次实验的总结

首先，我先给出我自己认为最适合运用于本次实验的模型：**逻辑回归模型**。逻辑回归模型作为上述五个模型中逻辑最简单的模型，可解释能力强、运算效率高、预测准确率高（仅比MLP低1%左右）。其余几个模型的训练时间相对来说过长，TextCNN模型甚至需要半个小时左右，而调参过程更是可能需要用上几天几夜。从原理上我相信TextCNN或者Bert会比逻辑回归的效果更好，但如果同时追求效率和准确率的话，我认为还是逻辑回归更好。

在本次的实验中，我用到了许多比较良好的文本分类模型，虽然在优化上依然有所欠缺，但我认为从结果来看已经比较完美了。在搜索资料的时候，我还在Github上找到了一个用于文本分类的模型https://github.com/luopeixiang/textclf，这个模型综合了多种模型，效果应该比较好，我后期也会进行尝试。这次没能用上BERT模型也比较遗憾，而且代码运行时间过长，本机CPU已经有点乏力了，我之后会考虑使用云主机colab跑模型。