**北京交通大学黄晓雯老师机器学习课程赛题：广告点击率预测结题报告**

**汇报人：程维森（21231264） 韦舒羽（21281202） 于芮（22125535）王若昕（23121096）徐千龙（23121121）**

**汇报学院：詹天佑学院 经济管理学院 土建学院**

# 目录

## 摘要

## 数据集处理

## LG与SVM模型

## Catboost与LightGBM

## Xgboost

## 不足与改进

## 总结与展望

## 贡献说明与代码仓库

# 摘要

点击通过率（CTR）作为广告效果的重要指标，在广告行业中具有重要意义。预测用户对广告的点击率是广告平台追求的目标之一。然而，单个广告平台的用户数据往往存在稀疏性和行为单一性的问题，影响了点击率预测模型的效果。

为解决这一问题，本研究提出了一种联合建模的方法，结合了目标域数据（单个平台下的用户特征数据）和源域数据（相同用户在其他平台下的行为特征数据）。通过整合跨平台的用户行为特征，本研究旨在深度挖掘用户的兴趣偏好，以提高点击率预测模型的性能。

实验中，我们采用了LG、SVM、XGBoost、LightGBM和CatBoost等多种机器学习模型，并对数据集进行了预处理。我们对比分析了这些模型在联合建模下的性能表现，探讨了不同模型在丰富用户行为特征、提高预测性能方面的优劣势。

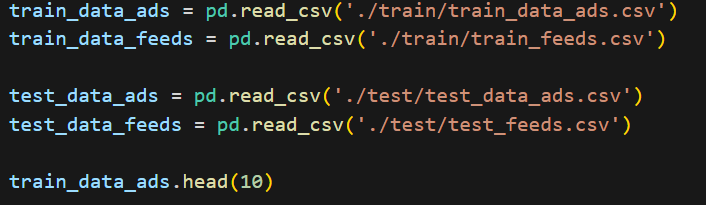
研究结果表明，联合建模方法在整合多平台用户行为特征方面取得了显著的效果，提升了点击率预测的准确性和泛化能力。同时，实验发现在此任务中，某些模型（列举所用的模型名称）相对于其他模型表现出更好的性能。这些发现为广告平台在点击率预测领域提供了有益的参考和启示。

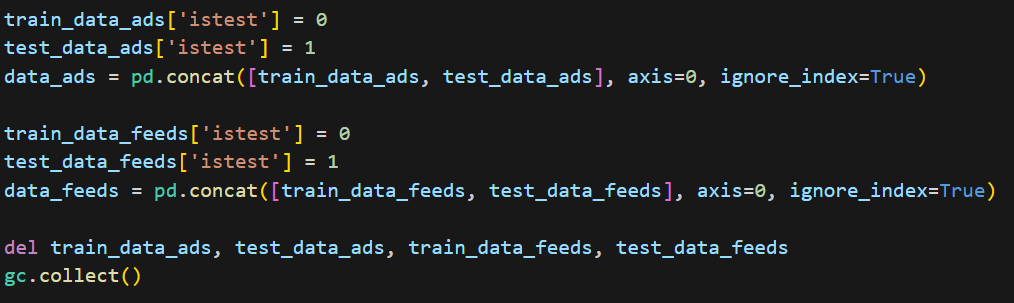
本研究的贡献在于提出了一种有效的方法，以应对单个平台数据稀疏性的挑战，为提升广告点击率预测的效果提供了新的思路和策略。

关键词：点击通过率 (CTR) 联合建模 用户行为特征 广告平台 机器学习模型

# 数据集处理

## 2.1读入数据并预先处理

在本代码中，先读入了测试集与训练集并且输出了测试集的表头来显示测试集的标签。

而后开始合并数据，代码采用了如下内容：  


代码实现了如下功能：  
通过对两个不同数据集（名为train\_data\_ads和test\_data\_ads，以及train\_data\_feeds和test\_data\_feeds）进行处理，将它们合并为新的数据集data\_ads和data\_feeds。

以下是对代码的逐行解释：

train\_data\_ads['istest']=0 和 test\_data\_ads['istest'] =1：在广告数据集中，为了区分训练集和测试集，将训练集中的新列 'istest' 设为 0，而将测试集的相同列设为 1。

data\_ads = pd.concat([train\_data\_ads, test\_data\_ads], axis=0, ignore\_index=True)：使用pd.concat函数将训练集和测试集的广告数据合并为一个新的数据集data\_ads，axis=0表示按行合并，ignore\_index=True表示忽略原始索引并重新索引新的数据集。

train\_data\_feeds['istest']=0和test\_data\_feeds['istest'] = 1：在另一个名为"feeds"的数据集中，同样将训练集和测试集进行标记，训练集列 'istest' 设为 0，测试集列设为 1。

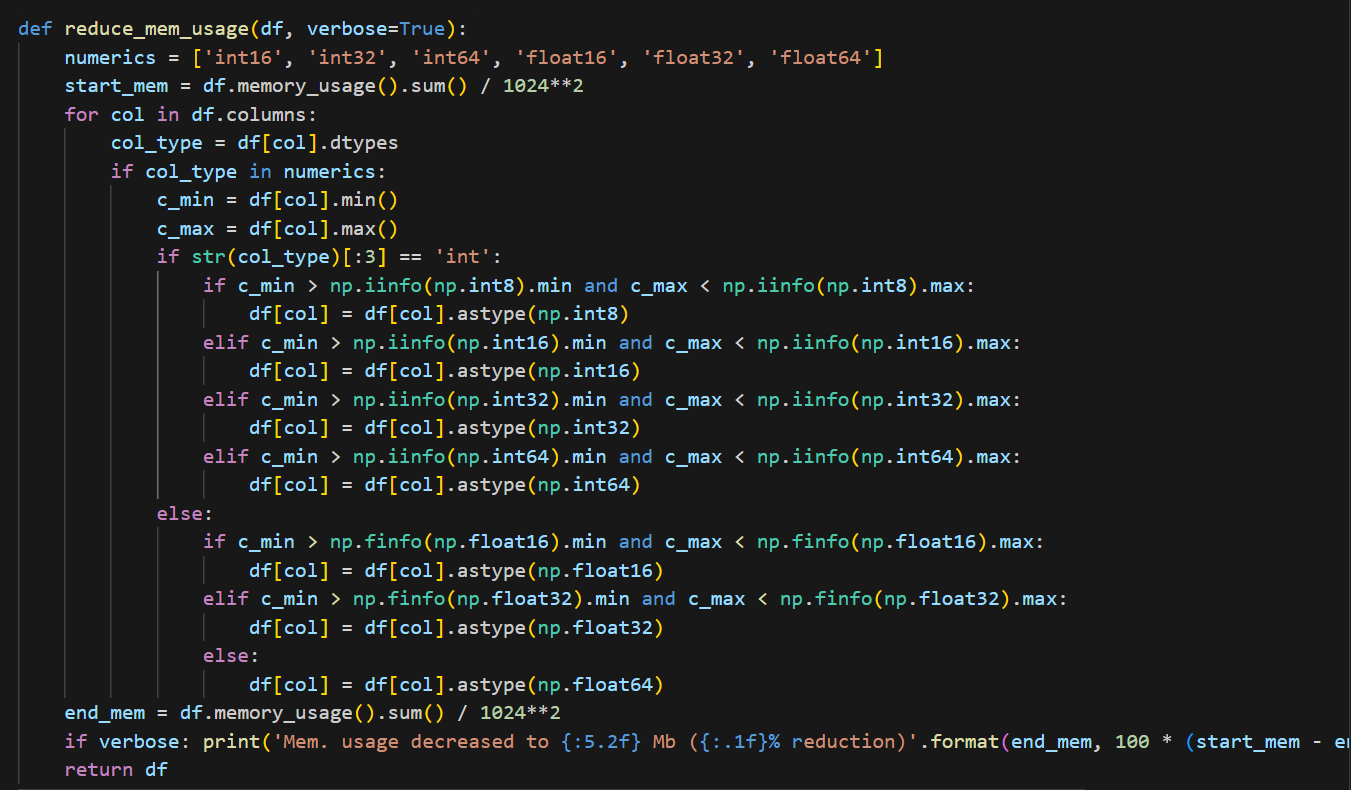
data\_feeds = pd.concat([train\_data\_feeds, test\_data\_feeds], axis=0, ignore\_index=True)：同样使用pd.concat函数将训练集和测试集的"feeds"数据合并为一个新的数据集data\_feeds，合并方式与上述的data\_ads相同。

del train\_data\_ads, test\_data\_ads, train\_data\_feeds, test\_data\_feeds：删除原始的训练集和测试集数据，以释放内存空间。

gc.collect()：手动触发Python的垃圾回收机制，清理不再使用的内存，以提高内存利用效率。

而代码的目的是将广告和"feeds"两个数据集的训练集和测试集进行标记并合并，最终形成名为data\_ads和data\_feeds的新数据集，方便后续的数据处理和分析。

## 2.2数据的内存空间处理与释放



这段代码实现了内存空间的处理，其作用是尝试减少DataFrame中数值类型列的内存使用，以节省内存空间。

以下是代码的具体解释：

函数接受一个DataFrame df 作为输入参数，并有一个布尔参数verbose（默认为True），用于控制是否输出内存优化的详细信息。

函数首先定义了一个列表numerics，其中包含了数值类型的数据类型（整数和浮点数）。

start\_mem = df.memory\_usage().sum() / 1024\*\*2：计算初始时DataFrame所占用的内存空间。

迭代DataFrame的每一列，对于数值类型列进行内存优化处理。对于每一列：

获取列的数据类型col\_type。

检查列的数据类型是否为数值类型，如果是则进行下一步处理。

获取该列的最小值c\_min和最大值c\_max。

根据列的数据类型和数值范围，尝试将列的数据类型转换为占用内存更小的数据类型，例如将int64转换为int32等，以减少内存使用。

根据列的数据范围，选择相应的 np.int8, np.int16, np.int32, np.int64, np.float16, np.float32, np.float64 数据类型来转换列的数据类型。

将列的数据类型转换为所选的新数据类型。

end\_mem = df.memory\_usage().sum() / 1024\*\*2：计算优化后DataFrame所占用的内存空间。

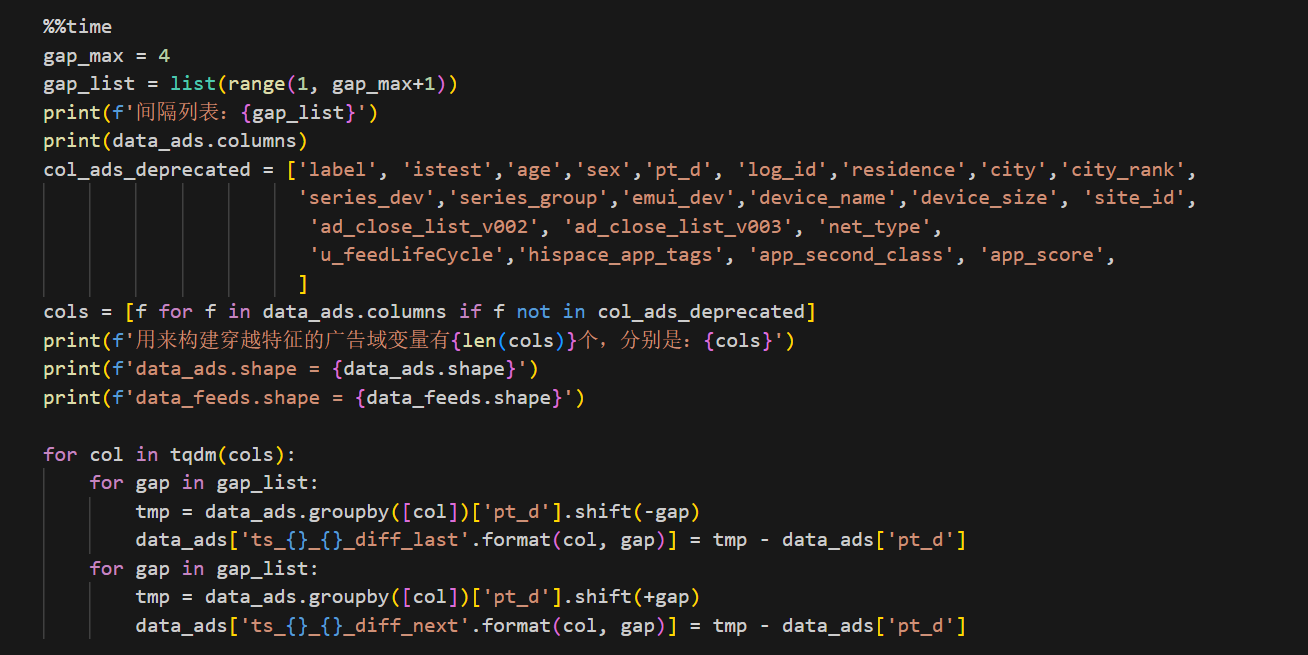
如果verbose为True，则输出优化后的内存使用信息，显示内存使用减少的百分比。

函数返回经过内存优化处理后的DataFrame df。

总的来说，这个函数的作用是尝试将DataFrame中数值类型列的数据类型转换为占用更少内存的类型，从而减少DataFrame整体的内存占用，提高内存使用效率。

## 2.3构建穿越特征

在释放内存前应该先构建穿越特征，如下是构建穿越特征的代码：



而这段代码主要是针对数据集data\_ads 中的特征进行时间穿越（Time Series Cross Feature）的处理。

如下则是代码中各部分的功能：

gap\_max 和 gap\_list：定义了一个时间穿越的间隔列表，从 1 到 gap\_max（此处为 4）。这个列表表示了要构建时间穿越特征所使用的不同时间跨度。

col\_ads\_deprecated：指定了一些不适合用于构建时间穿越特征的列，例如标签列、用于标识测试/训练集的列、和一些与时间无关或不适合用于时间穿越的列。

cols：选取了除去不适合的列之外的其他特征列，用于构建时间穿越特征。

for col in tqdm(cols):：对于被选定的可以构建时间穿越特征的列，循环进行下列操作：

for gap in gap\_list:：对于每个选定的列和不同的时间间隔，进行以下操作：

tmp = data\_ads.groupby([col])['pt\_d'].shift(-gap) 和 tmp = data\_ads.groupby([col])['pt\_d'].shift(+gap)：利用 groupby 和 shift 函数，根据选定列的不同取值，将时间往前或往后移动 gap 个单位，从而获得在不同时间点的数据。

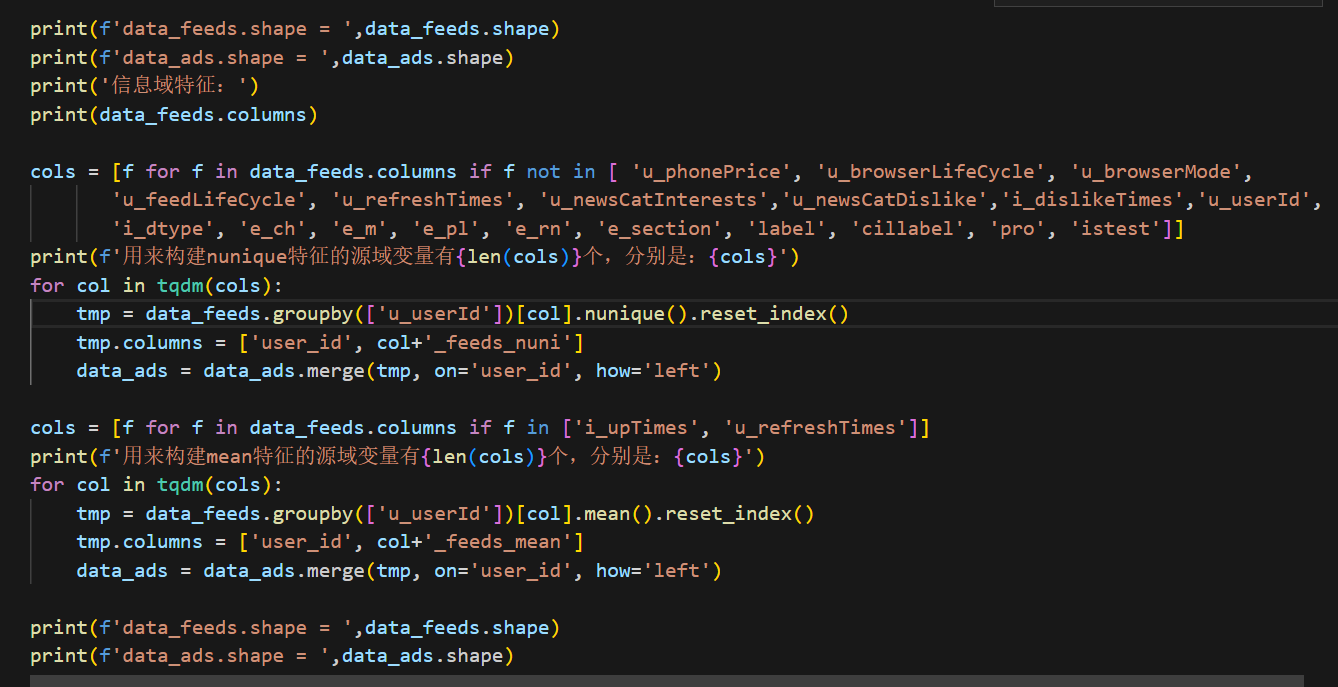
data\_ads['ts\_{}\_{}\_diff\_last'.format(col, gap)] = tmp - data\_ads['pt\_d'] 和 data\_ads['ts\_{}\_{}\_diff\_next'.format(col, gap)] = tmp - data\_ads['pt\_d']：计算了时间间隔内两个时间点之间的时间差，分别创建了两列特征来表示当前时间点与前后时间点之间的时间差。

总体而言：这段代码通过循环迭代选定的特征列，并对每个特征列结合不同的时间间隔进行处理，创建了多个时间穿越特征。这些特征反映了不同时间间隔下，当前样本与过去或未来样本之间的时间差信息。

在穿越特征构建结束后为节约时间此时进行第一次内存压缩。

## 2.4处理数据集中的不合适特征

这个代码处理了数据集中一些不适合构建穿越特征的数据，处理后保留了一些标签来进行训练。



以下是对于该代码的详细解释：

对于数据集 data\_feeds 中的一部分特征列，进行了特征工程处理：

去除了一些不适合用于构建计数特征和均值特征的列。

对剩余的特征列进行了如下处理：

构建了基于每个用户的不同特征值数量的计数特征（nunique特征），对应于 cols 中不在排除列表中的列。对于每个特征列，计算了每个用户的独立值数量，并将结果存储在 data\_ads 中的新列中。

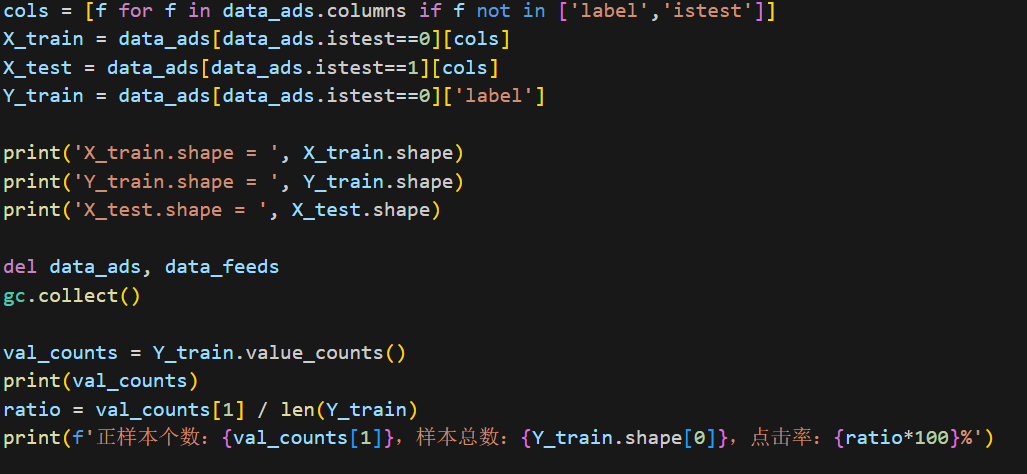
构建了基于均值的特征（mean特征），对应于 cols 中的某些列。对于每个特征列，计算了每个用户对应特征的平均值，并将结果存储在 data\_ads 中的新列中。

该代码的核心是通过对 data\_feeds 数据集中的特定列进行处理，从而为 data\_ads 数据集构建了新的特征。这些新的特征包括每个用户不同特征值数量的计数特征（nunique特征）和某些列的均值特征（mean特征）。这些新特征的添加有助于提高模型对用户行为的理解和预测能力。

处理结束之后又一次压缩了内存，执行了第二次内存压缩

此时数据处理完毕。

## **2.5 划分数据集和测试集**

处理完毕数据后应该进行数据集和测试集的划分：  


此代码进行了包括了特征选择、数据拆分、以及对训练集的目标标签进行统计分析等操作。

具体操作如下：

cols=[f for f in data\_ads.columns if f not in ['label','istest']]：定义了一个特征列表，包含了除了标签列和测试标识列之外的所有特征列。

X\_train = data\_ads[data\_ads.istest==0][cols] 和 X\_test = data\_ads[data\_ads.istest==1][cols]：将训练集和测试集的特征数据分别提取出来。X\_train 包含了训练集的特征数据，X\_test 包含了测试集的特征数据。

Y\_train = data\_ads[data\_ads.istest==0]['label']：提取出训练集的目标标签数据，即广告是否被点击的标签列数据。

打印并显示了训练集和测试集特征数据的形状（行数和列数），以及训练集目标标签数据的形状（行数）。

del data\_ads, data\_feeds 和 gc.collect()：删除了已经提取特征和标签后的数据集，并手动触发了 Python 的垃圾回收机制以释放内存空间。

val\_counts = Y\_train.value\_counts()：统计了训练集目标标签的取值数量，得到了每个类别的样本数量。

ratio = val\_counts[1] / len(Y\_train)：计算了训练集中标签为1（点击）的样本数量与总样本数量之间的比例，从而得到了点击率的百分比表示。

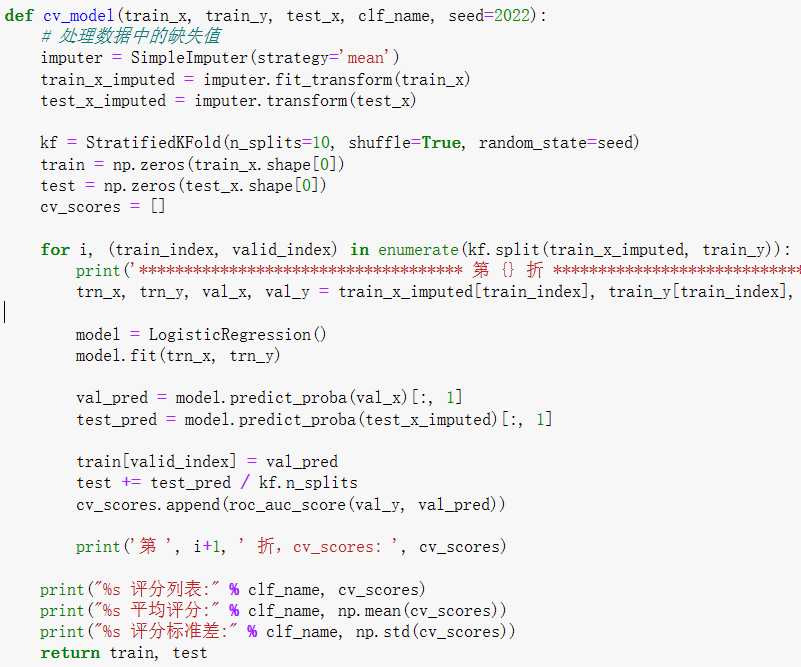
总的来说，这段代码完成了数据集的准备工作，将特征数据和标签数据分别提取出来，并进行了数据形状的确认和内存释放操作。同时，还对训练集的目标标签进行了点击率的统计分析。

此时数据集完全处理完毕，下文将对不同的模型进行分析并且给出测试结果。

# LG与SVM模型

## **3.1 逻辑回归模型(LG)代码实现**

我们首先尝试采用逻辑回归模型进行广告点击率的预测。虽然逻辑回归在许多二分类问题中表现出色，但我们在实际应用中发现其在这个特定任务上的效果并不理想。在这一节中，我们将详细探讨该模型的实现过程和遇到的挑战。



在开始模型训练之前，我们首先关注于数据的预处理，特别是缺失值的处理。使用SimpleImputer来填补缺失值，这个步骤虽然简化了数据，但也可能对模型的最终表现产生了影响。我们采用了特征的平均值来替换缺失值，这种处理方式在某些情况下可能不足以捕捉数据的全部复杂性。

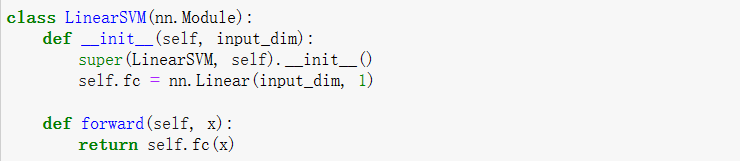
考虑到数据集的复杂性和多样性，我们采用10折StratifiedKFold交叉验证。这种方法可以确保每一折中各类别的比例一致，但即便如此，线性模型在处理高度非线性的数据时仍显示出了局限性。

在实施交叉验证过程中，我们训练了逻辑回归模型并在不同的子集上进行了验证。尽管我们关注于预测概率值以计算AUC分数，但最终的性能表现并未达到预期。这可能是由于模型的简单性和数据特征之间复杂关系的不匹配。

通过这个实验，我们意识到，虽然逻辑回归模型在理论上适用于二分类问题，但在处理实际的、复杂的广告点击率预测任务时，其性能可能会受限。这一发现促使我们考虑更复杂的模型或更精细的特征工程，以提高预测的准确性。此次尝试虽然未能达到预期效果，但它提供了宝贵的经验，为我们后续选择和优化模型提供了重要的参考。

## **3.2 支持向量机(SVM)代码实现**

在本次作业的第二个实验中，我们决定采用支持向量机（SVM）完成广告点击率的预测。SVM是一种强大的机器学习算法，经常用于分类任务。这里，我们实现了一个基于PyTorch的简单线性SVM模型。以下是关键的类定义：



这个类继承自nn.Module，是PyTorch中模型的基础。我们定义了一个线性层self.fc，它将输入数据映射到一个输出节点上，这对应于二分类任务中的一个决策边界。



和逻辑回归模型一样，我们首先处理了数据中的缺失值，并使用了10折StratifiedKFold交叉验证。这些步骤确保了我们的模型能在均衡和完整的数据上进行训练和评估。

在每一轮交叉验证中，我们首先将数据转换为PyTorch张量，并将模型及数据移至GPU以加速训练过程。我们采用了随机梯度下降（SGD）优化器和二元交叉熵损失函数，这是二分类任务中的标准选择。在100个训练周期内，模型在训练集上进行学习。每个周期中，模型的参数通过反向传播算法进行更新，以最小化损失函数。模型在验证集上的预测结果被用来计算AUC分数，以评估模型性能。在每一折结束时，我们记录了当前折的分数，并在所有折完成后，计算出平均AUC分数及其标准差。

通过这一节的内容，我们展示了线性SVM模型在广告点击率预测任务中的实现和性能。尽管线性SVM在某些情况下可能比逻辑回归模型表现更好，但其在处理复杂和高度非线性数据时仍有局限。因此，我们在后续的实验中可能需要考虑更复杂的模型或更细致的特征工程策略，以进一步提高预测的准确性。这一实验过程不仅提供了技术上的洞察，也为我们未来的模型选择和优化提供了重要的经验和参考。

## **3.3 无CUDA加速的SVM (SVMnocuda)实现与评估**

在前两节中，我们尝试了逻辑回归和支持向量机模型在广告点击率预测中的应用。除此之外，我们也尝试了无CUDA加速的SVM模型。

SVMnocuda与先前的SVM模型有着类似的结构，但在计算资源的使用上有所不同。无CUDA加速的环境意味着所有的计算都在CPU上执行，这可能会对训练时间和效率产生显著影响。

SVMnocuda的实现同样涉及到对数据进行预处理，包括处理缺失值和应用10折 StratifiedKFold 交叉验证。这些步骤保证了实验的一致性，并允许我们评估不同硬件配置下的模型性能。

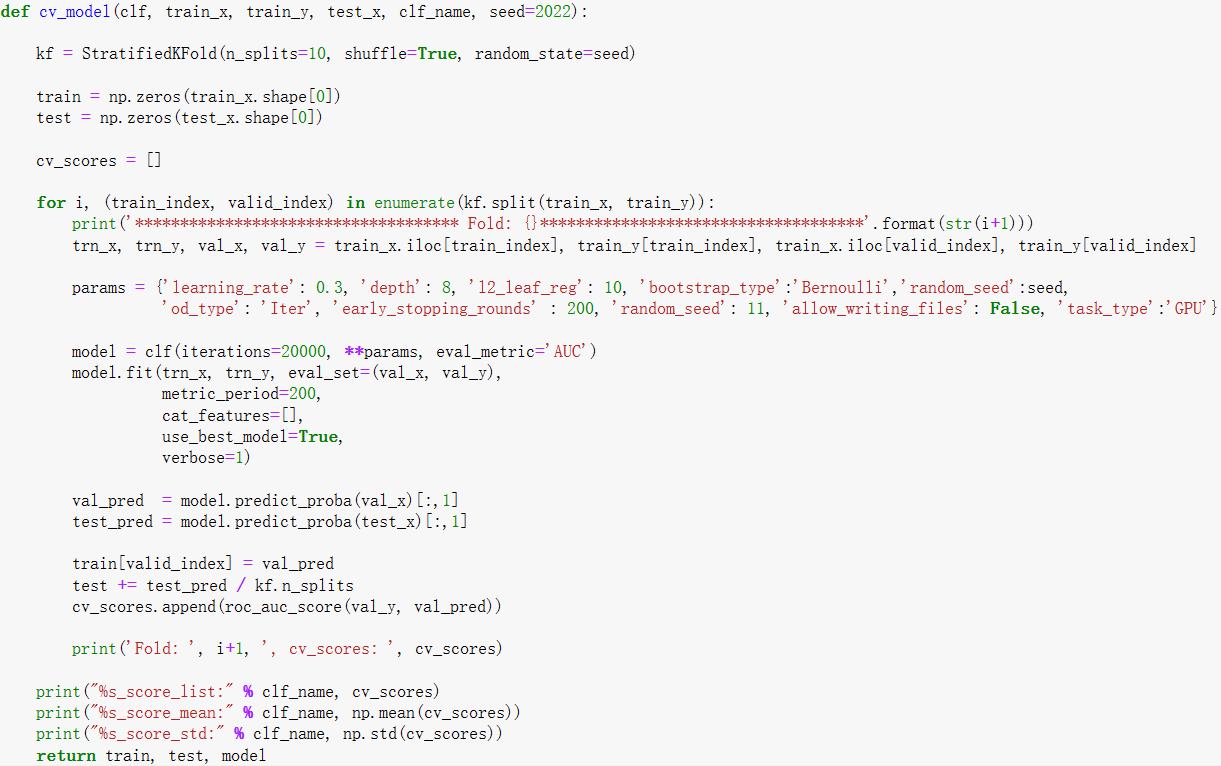
由于缺乏GPU加速，我们注意到SVMnocuda模型的训练过程更加耗时。这在处理大型数据集时尤为明显，其中每次迭代和优化步骤都需要更长的时间来完成。然而，尽管训练效率降低，我们仍然能够评估模型在验证集上的性能，并通过AUC分数进行量化。

此次探索显示：虽然无CUDA加速的SVM模型在训练效率上存在局限，但它仍然能够提供对广告点击率预测任务有价值的见解。这一发现强调了在不同硬件条件下进行机器学习实验的重要性，同时也为资源受限环境中的模型部署提供了实际经验。

# Catboost和LightGBM

## **4.1 Catboost代码实现**

在上一章中，线性模型的表现一般，因此我们将选择更加复杂的模型来协助完成作业，在本节中，我们将介绍CatBoost算法在广告点击率预测任务中的实现。CatBoost是一个高效的、用于分类、回归和其他机器学习任务的梯度提升决策树算法。它在处理类别特征方面特别有效，并且经常被用于复杂的数据集。



与之前的模型类似，我们采用了相同的数据处理方式，并同样通过10折StratifiedKFold交叉验证进行评估。这种方法确保了模型在不同子集上的性能可以被公平地评估。在每一轮交叉验证中，模型使用一组特定的参数进行训练。这些参数包括学习率、树的深度、L2正则化系数等，旨在优化模型的表现和防止过拟合。

模型的训练过程涉及到对训练集和验证集的分别学习和评估。通过设置 early\_stopping\_rounds，我们得以确保模型在达到某个性能阈值后停止训练，从而提高效率。此外，使用GPU加速进一步加快了模型的训练过程。

在每一折的交叉验证结束时，我们使用ROC-AUC分数来评估模型在验证集上的性能。最终，汇总所有折的结果，计算出平均分数和标准差，这为我们提供了模型整体性能的全面视图。

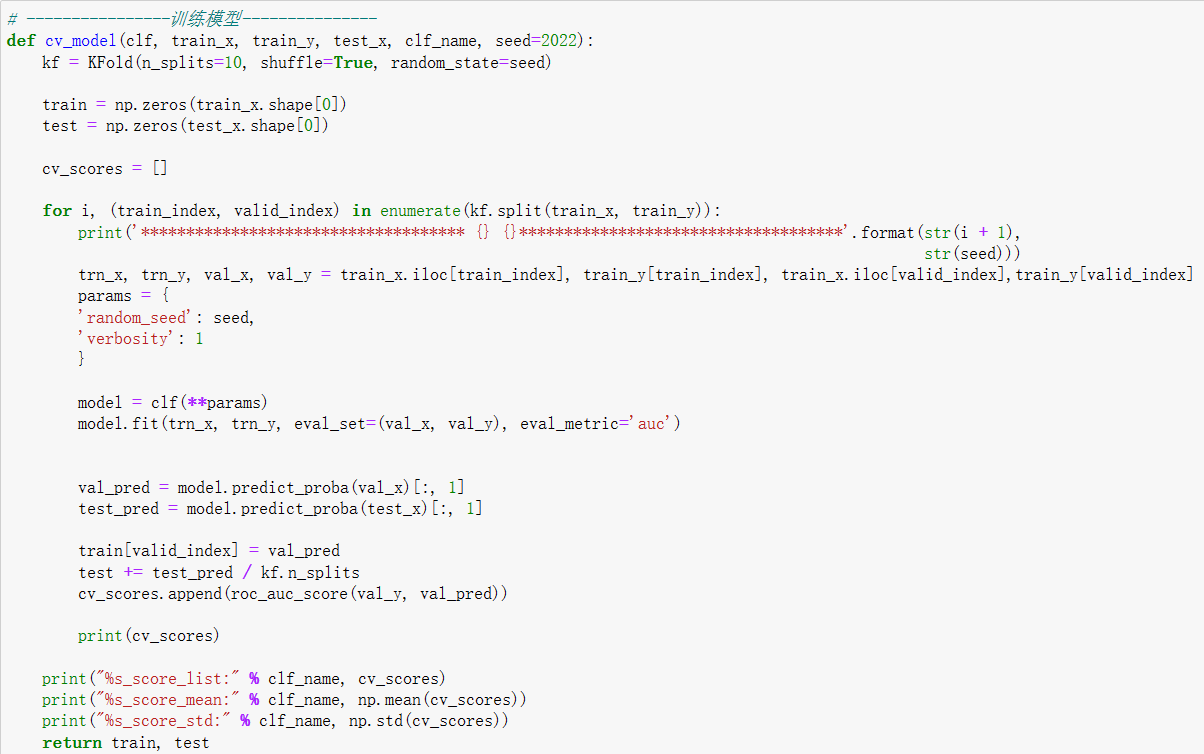
通过详细的数据预处理、精心的参数选择、GPU加速和早停机制，CatBoost模型在广告点击率预测任务中展现了优秀的性能。这一实验不仅证明了CatBoost在处理复杂数据集时的有效性，也为未来更广泛应用提供了坚实的基础。我们的结果强调了合适的预处理和参数调整对于实现高性能预测模型的重要性。

## **4.2 LightGBM代码实现**

继CatBoost之后，我们也在在广告点击率预测作业中尝试了LightGBM模型的应用。LightGBM是一个高效的梯度提升框架，特别适合处理大型数据集，并且在处理类别特征方面表现优异。

与CatBoost类似，我们对LightGBM模型的数据进行了彻底的预处理，包括数据合并、缺失值处理，以及内存优化。这些步骤对于保证模型训练的高效性和数据的可靠性至关重要。

LightGBM模型训练代码核心部分如下：



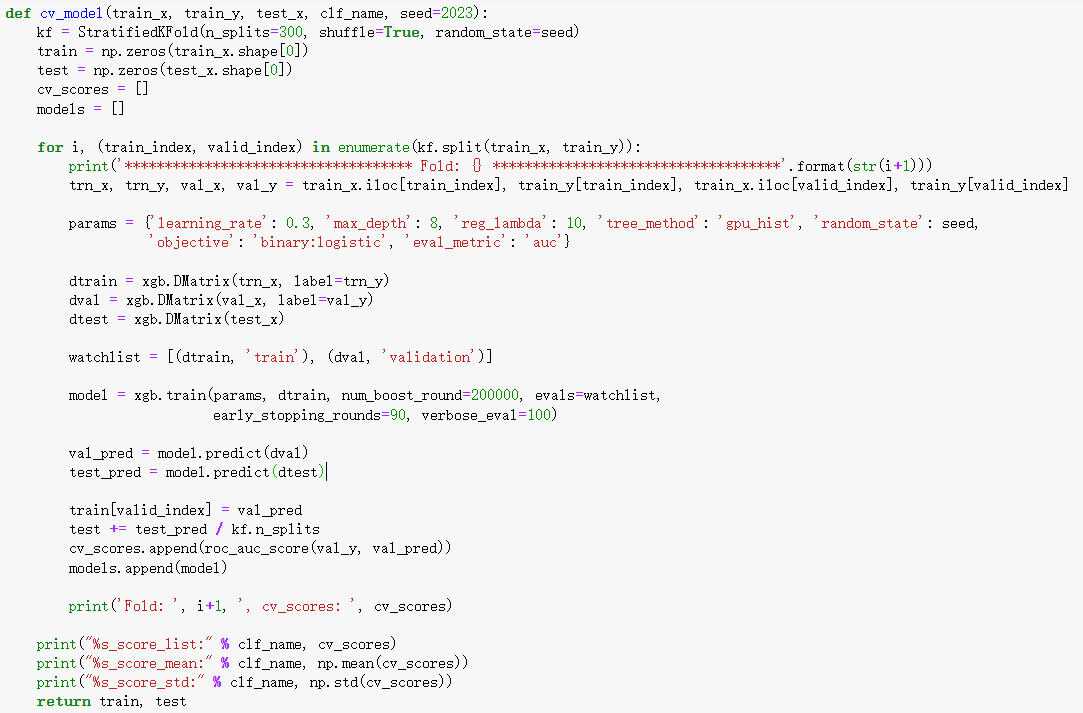
在这个函数中，我们使用了KFold交叉验证而非StratifiedKFold，这是因为LightGBM对数据的分布和结构非常敏感，故采用不同的交叉验证策略来确保模型的泛化能力。

在LightGBM模型的训练过程中，我们细心设置了一系列基本参数，其中包括随机种子和日志输出级别。由于AUC是这次作业的指定评估指标，我们在训练过程中特别注意了模型在这一指标上的表现。通过在不同的训练和验证集上计算AUC分数，我们能够直观地评估模型在各个方面的性能。

LightGBM模型在广告点击率预测任务中表现出色。通过详细的数据预处理和精心的参数调整，我们能够高效地训练模型，并取得了优秀的性能。LightGBM的实现强调了对数据特征和模型参数深入理解的重要性。此次实验不仅展示了LightGBM在处理复杂数据集时的能力，也为未来的应用提供了有价值的经验和参考。

# XGboost

我们在本次作业中选择XGBoost算法作为基座模型。XGBoost常用于设计和优化大规模、高效的分布式计算。相较于前文的Catboost和LightGBM方法，XGBoost在处理小数据集方面表现更为稳定，也更为容易调参。同时，XGboost在处理二分类问题时表现更好，可解释性好，可以输出各个特征的重要性评分，对于结果的解释更加直观。XGboost的模型训练代码如下所示。



对于XGBoost算法，我们依然采用StratifiedKFold交叉验证作为数据处理方式。需要注意的是，在这种方法下为了提高模型性能评估的可靠性，需要XGboost的模型的K值，这样可以使每个分折包含更多的训练样本，从而提高模型性能评估的可靠性，以及更全面地评估模型在不同数据子集上的一致性和稳定性。这里，我们对K的取值为300。

在核心的训练代码之后，我们将数据集转化为DMatrix格式以供模型使用，并设置了监控列表用于指定需要监控的数据集。xgb.train()函数用于训练XGBoost模型，在训练过程中，模型会根据损失函数和优化器对样本进行迭代优化，逐渐提升性能。此处我们设置模型如果在验证集上连续90轮性能没有提升，则停止训练并返回最后一个训练轮数对应的模型，且模型在每100轮迭代后会输出一次验证集上的性能指标，以供观察和调整参数。训练后，得到的XGBoost模型就可以用于预测新的样本。

与前面的几种方法相比，XGBoost算法具有高效的训练和预测速度，在大规模数据集和高维特征空间下表现优异。其采用正则化技术和特征重要性评估等手段，有多个可以调节的超参数，可以通过交叉验证等方式进行优化选择，提升模型的性能和泛化能力。XGBoost算法同时也具有较好的可解释性，可以通过可视化和特征重要性评估等方法，解释模型的预测过程和结论，为后续的决策提供支持和参考。

# 不足与改进

目前，我们所做的作业的处理数据部分在数据去噪声以及数据深度处理方面上尚有欠缺，如果这两个内容能够实现，可以提升模型的稳定性以及泛化能力，有助于提升模型的效率。

同时，考虑到混合模型将大大增加整体模型的复杂程度，参数调优也更加困难，容易导致过拟合的发生，我们未将CatBoost以及LightGBM等模型这些方法进行组合，形成非线性混合模型。若能够成功地使用混合模型进行计算预测，则有望取得较混合前更高的分数。

# 总结与展望

针对网站的广告点击率预测任务，我们使用了不同的机器学习算法，包括LG、SVM、CatBoost、LightGBM和Xgboost方法。这些算法各自具有独特的优点和适用范围。对于LG和SVM这类线性算法而言，由于模型的简单性和数据特征之间复杂关系的不匹配，无法很好地处理广告点击率这种较为复杂的非线性数据，导致最终的性能表现并未达到预期。即便如此，它们仍然能够为资源受限环境中的模型准备提供实际经验。而CatBoost、LightGBM和Xgboost算法则表现出了较为出色的性能。其都是基于决策树的集成学习算法，具有高效性、可扩展性、稳定性等优点。

在未来，随着数据量不断增大、复杂度不断提高，机器学习算法以及其相应框架和工具会不断更新迭代，更好的应对现实世界中更加复杂的问题。本次作业通过对比使用不同算法处理数据的预测结果，为广告平台在点击率预测领域提供了有益的参考和启示。

# 贡献说明与代码仓库

**贡献度**：**程维森（21231264）20%：  
负责论文主题以及穿越特征调整以及XGboost**

**韦舒羽（21281202） 20%：**

**负责穿越特征以及模型筛查与XGboost**

**于芮（22125535）20%：**

**负责LG回归部分**

**王若昕（23121096）20%：  
负责Xgboost部分的论文**

**徐千龙（23121121）20%：**

**负责catboost部分的论文**

**代码仓库：**<https://github.com/sksx085/CTR_for_bjtu>