Analysis Report

黄子骕 21307130013

尚　宇 21307130233

游年浩 21302010043

探索性数据分析（EDA）与特征工程

数据探索性分析（EDA）在项目中具有至关重要的作用。通过对数据的初步观察和可视化，EDA有助于揭示数据的分布、特征之间的关系和潜在问题，能够帮助我们更清晰地了解数据的特点，包括每个特征的分布情况、异常值的存在以及数据的整体形态，进一步我们能够发现特征之间的关系，从而为后续的特征工程提供指导。

数据集情况总览

首先我们需要了解数据集的规模以及特征，以下为本数据集的 shape 以及 columns

```
(284807, 31)
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

由此可知，数据集大约有 28w 条样本数据，每条样本除了 label 列 “Class” 之外，有 30 个特征，其中 28 个是经过 PCA 后的结果，其余两个分别为 “Time”、”Amount”。一般情况下，这样数量的特征有些过多了，并且我们不知道其中是否有一些无关特征，留下他们会导致模型性能的降低，因此我们需要进一步筛选特征，以构造更高质量的数据。

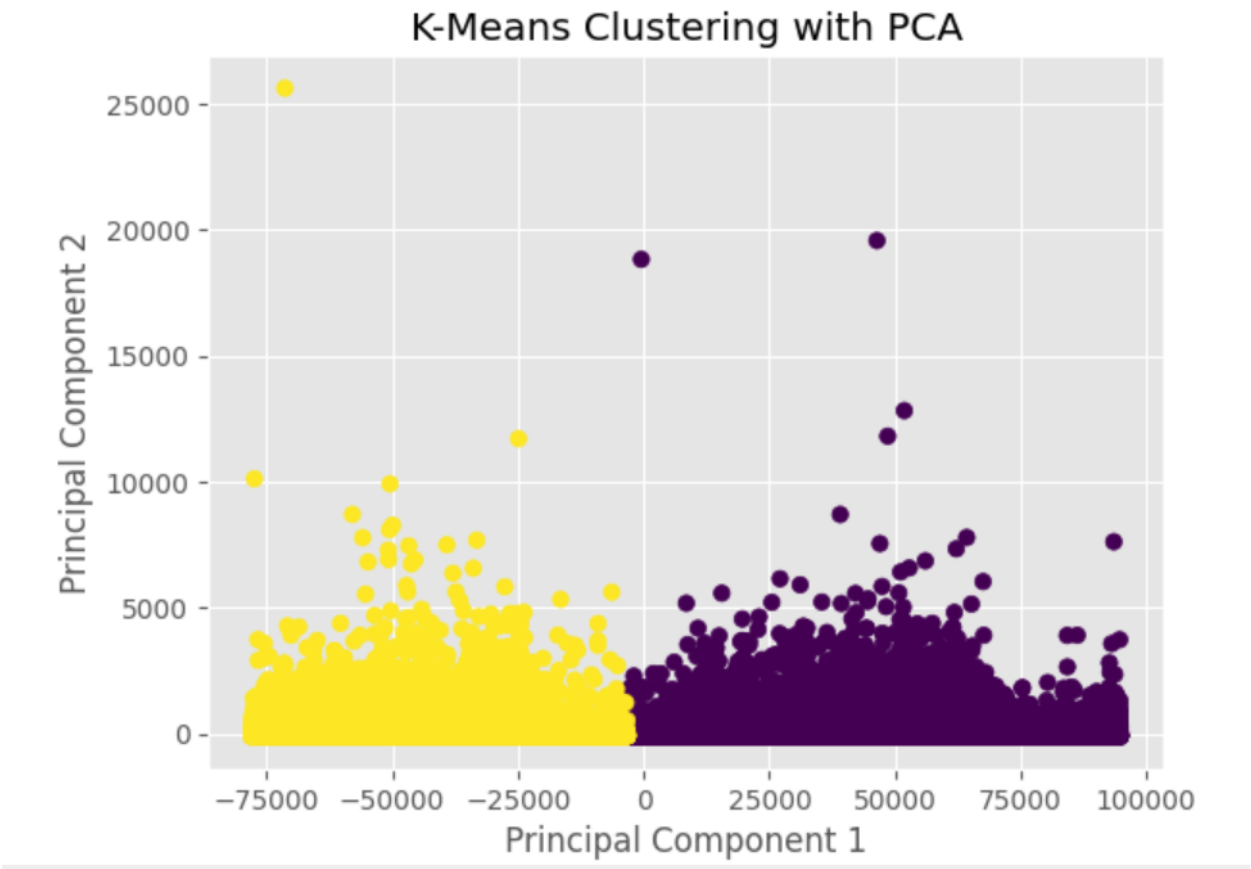
空值检查

空值检查用于识别数据中的缺失值，有助于评估数据完整性和质量，指导后续的数据清洗和处理策略。通过对于所有特征列进行检查，得到结果为本数据集中并没有缺失值，可以正常进行下一步。

离群点分析

离群点检测有助于深入了解数据的结构和特点，为后续的异常检测和数据理解提供重要参考，同时增强数据集的质量，使模型性能更好。本次使用的方法是聚类+PCA的组合。

首先，聚类可以将数据分为不同的簇，使得同一簇内的数据点相似度较高，而不同簇之间的相似度较低。这有助于识别数据中的潜在模式和群集。然后，通过在聚类后的簇上应用主成分分析，将数据映射到主成分空间，减少数据的维度同时保留主要的变化，并且容易可视化。

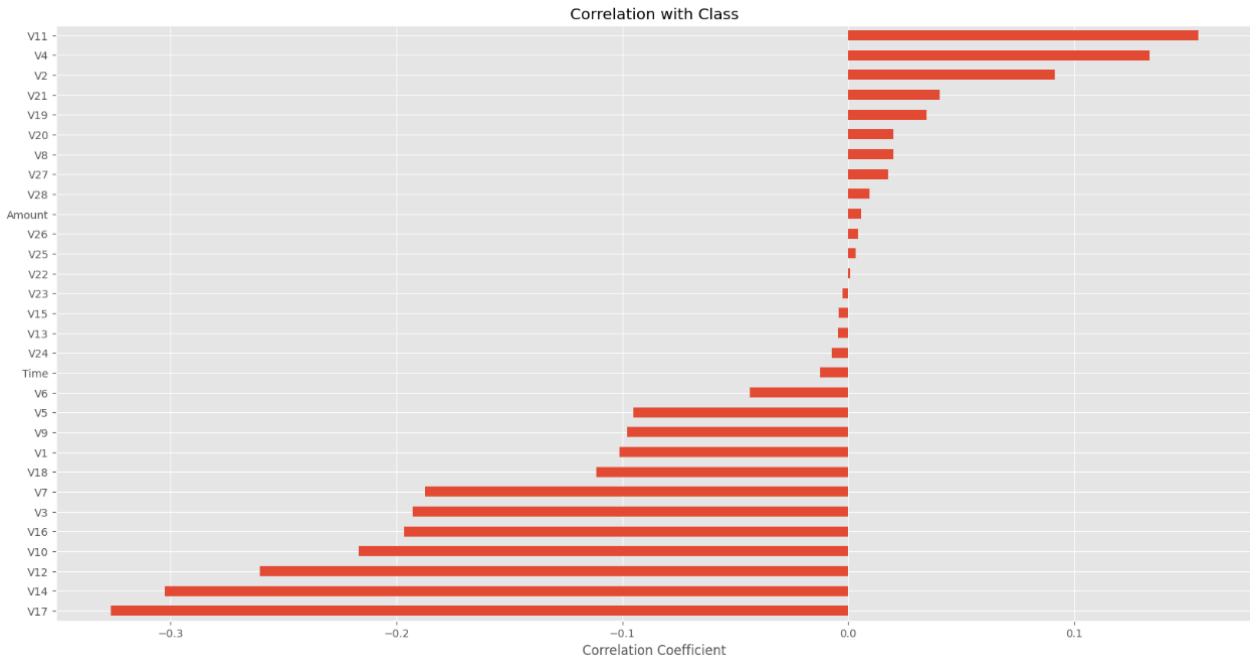


由上图可知，离群点确实存在，但数量不多，后续可以考虑使用一些操作来剔除他们，防止在模型训练阶段造成干扰

相关性分析

相关性分析用于衡量两个变量之间的线性关系强度，其原理基于相关系数的计算。通过相关性分析，我们可以了解变量之间的关联程度，帮助识别潜在的模式和趋势。

以下是计算所有特征与 label 的皮尔逊相关系数结果：



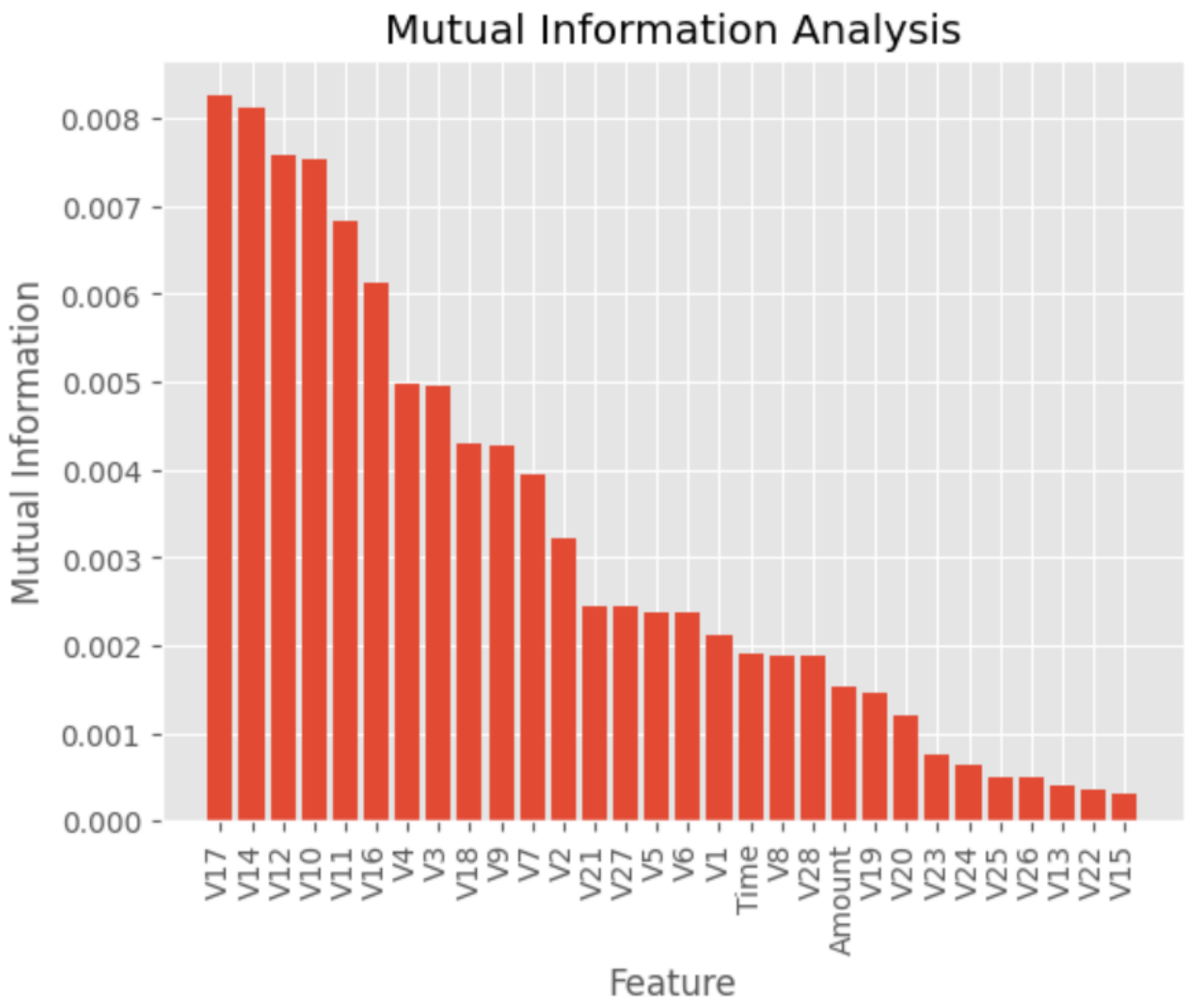
通过可视化图表，我们可以发现有一部分特征与 label 的相关性系数位于 0 附近，这表示二者线性关系很弱，可以考虑不使用他们。

这里有一点需要注意，这里的相关性分析只能检测线性关系，因此对于和 label 有非线性关系的特征，可能无法反映在相关性系数上，因此需要综合其他分析。

互信息分析

互信息是一种用于度量两个变量之间非线性关系的方法，它基于信息论的概念，衡量两个变量之间的信息共享程度，这里通过计算互信息的方式来研究特征与 label 的非线性联系。

以下是计算所有特征与 label 之间互信息的结果：

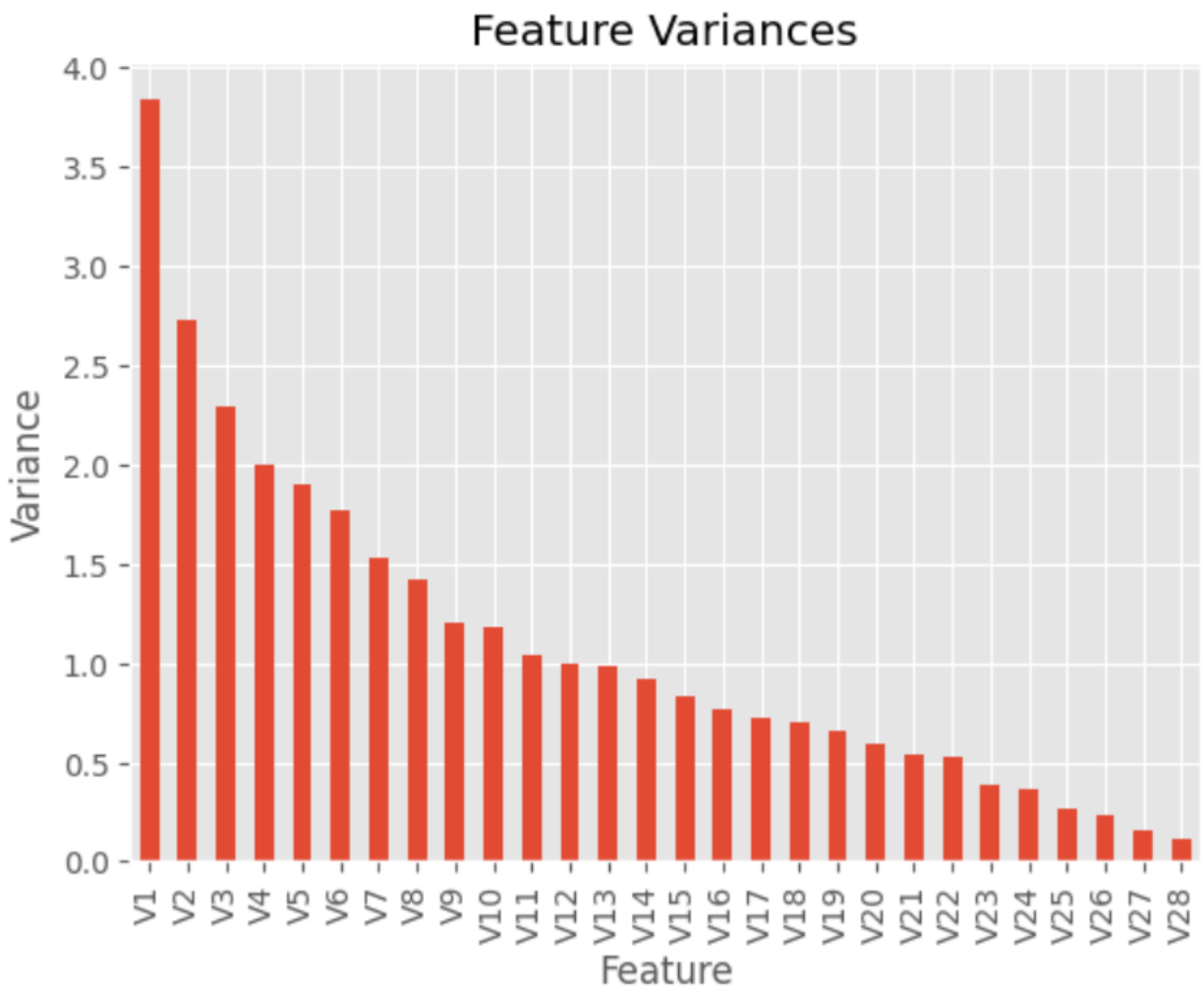


由图可以发现，同样地，有部分特征互信息的数值很低，这表示其与 label 的非线性关系很弱，也可以考虑放弃这些特征。

方差分析

方差分析也是常用的特征选择方法。在特征选择中，方差分析可用于评估不同特征的变化程度，帮助识别具有显著变化的特征，这些特征一般更有几率与 label 有较强关系。

以下为所有特征（除了”Time”与“Class”，这二者方差过大，无法进行绘图）的方差计算结果：



同理，通过方差分析，我们也可以排除一些可以不太合适的特征。

总结

在经过以上三种分析后，我们对于每一种分析都定义了一个阈值，分别为 corr = 0.05, mutual_info = 0.002, var = 0.5。如此，我们对于每种分析方法都可以筛选掉一部分的特征。为了综合三种分析方法的结果，我们取通过三个分析筛选的特征的交集，最终得到 15 个特征。

针对不平衡数据集的处理

处理不平衡数据集的必要性体现在确保机器学习模型在面对不同类别样本时具有更好的泛化性能上。在不平衡数据中，模型可能会更倾向于预测占多数的类别，而对于少数类别的

预测性能较差。这会导致模型的准确性、召回率等指标出现偏差，使得模型对于少数类别的识别能力不足。通过采用过采样、欠采样等方法，可以平衡类别分布，提高模型对于少数类别的学习效果，增强模型的鲁棒性和实际应用价值。

在数据层面，本次项目我们一共测试了五种处理方法，分别为：随机过采样、SMOTE、SMOTETomek、ADASYN、NearMiss。这其中包括了过采样、欠采样以及组合采样的方法。

注：除了在数据层面的处理外，我们还尝试了模型层面调整少数类别的权重来对抗样本的不均衡的方法，见模型测试与选择部分。

随机过采样

随机过采样是一种简单直观的方法，通过复制少数类别的样本来增加其数量，以实现类别平衡。采样后前后黑白样本比例与数据集维度如下：

```
Before ROS: Counter({0: 227451, 1: 394})
After ROS: Counter({0: 227451, 1: 227451})
x_train_ros shape:(454902, 15)
y_train_ros shape:(454902, 1)
```

SMOTE过采样

SMOTE通过在特征空间中插值生成合成的少数类别样本，从而缓解了随机过采样的问题。它选择一个少数类别样本和其最近邻居，然后在它们之间的线段上生成新的合成样本。采样后前后黑白样本比例与数据集维度如下：

```
Before SMOTE: Counter({0: 227451, 1: 394})
After SMOTE: Counter({0: 227451, 1: 227451})
x_train_smote shape:(454902, 15)
y_train_smote shape:(454902, 1)
```

组合采样（SMOTETomek）

结合了SMOTE和Tomek Links，先使用SMOTE进行过采样，然后使用Tomek Links进行欠采样。

```
Before CS: Counter({0: 227451, 1: 394})
After CS: Counter({0: 227451, 1: 227451})
x_train_cs shape: (454902, 15)
y_train_cs shape: (454902, 1)
```

ADASYN过采样

ADASYN是对SMOTE的改进，引入了自适应机制。ADASYN根据每个少数类别样本周围的多数类别样本的数量来调整生成合成样本的数量，使得在密度较低的区域生成更多的合成样本，从而提高模型对稀疏区域的适应性。

```
Before ADASYN: Counter({0: 227451, 1: 394})
After ADASYN: Counter({0: 227451, 1: 227444})
x_train_adasyn shape: (454895, 15)
y_train_adasyn shape: (454895, 1)
```

NearMiss欠采样

NearMiss是一种欠采样方法，它选择保留与多数类别样本最近的少数类别样本，以减少样本数量。

```
Before NearMiss: Counter({0: 227451, 1: 394})
After NearMiss: Counter({0: 394, 1: 394})
x_train_nearmiss shape: (788, 15)
y_train_nearmiss shape: (788, 1)
```

模型训练与调参

TODO：

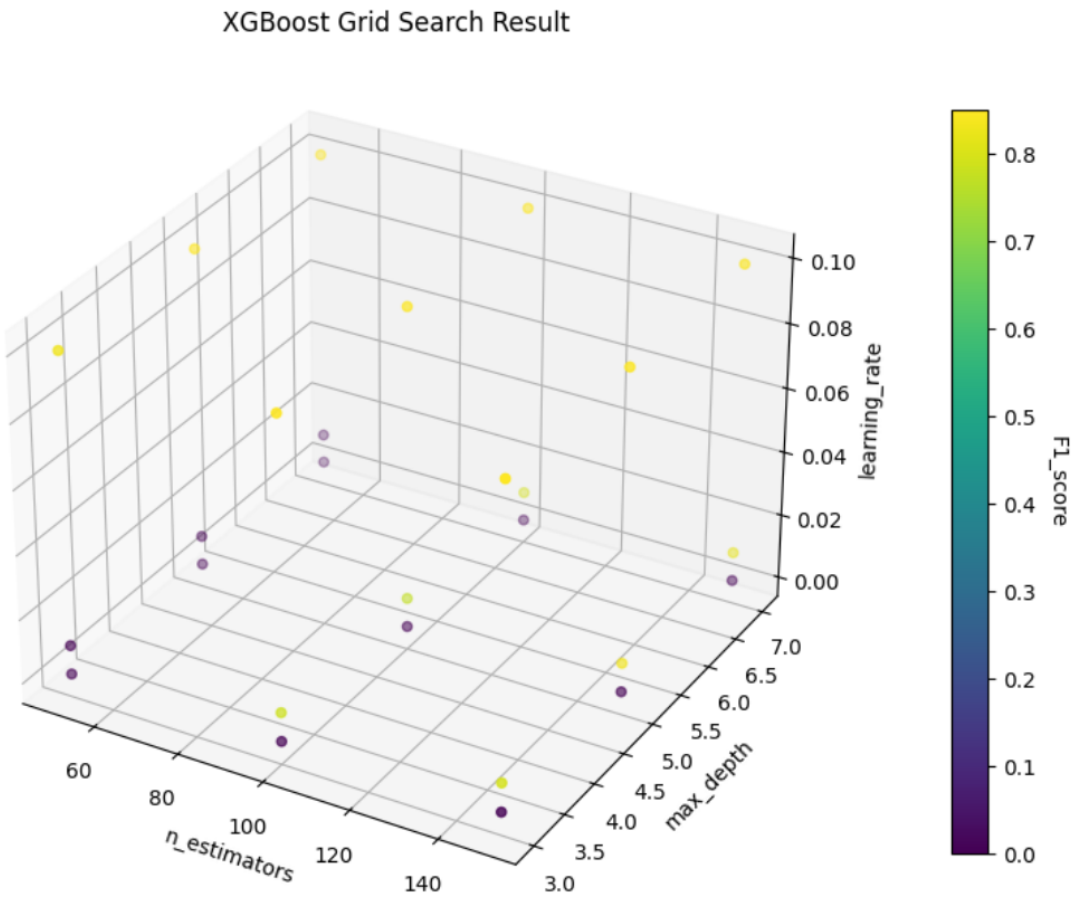
梯度提升 (XGBoost)

梯度提升是一种集成学习算法，通过串联多个弱分类器（通常是决策树）来构建一个强分类器。它以迭代的方式训练模型，不断减小残差来逐步改进预测效果，通常以决策树作为

基本的弱分类器，在每一轮迭代中，新的决策树都会尝试去纠正前面所有树的残差，从而逐步提升整体的预测性能。

梯度提升中值得关注的参数包括：learning_rate、max_depth和n_estimators。对这三个参数进行带有交叉验证的网格搜索结果如下

```
best_params = {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50}
f1_score_value = 0.8777777777777778
```



得到最优参数组合`'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50`

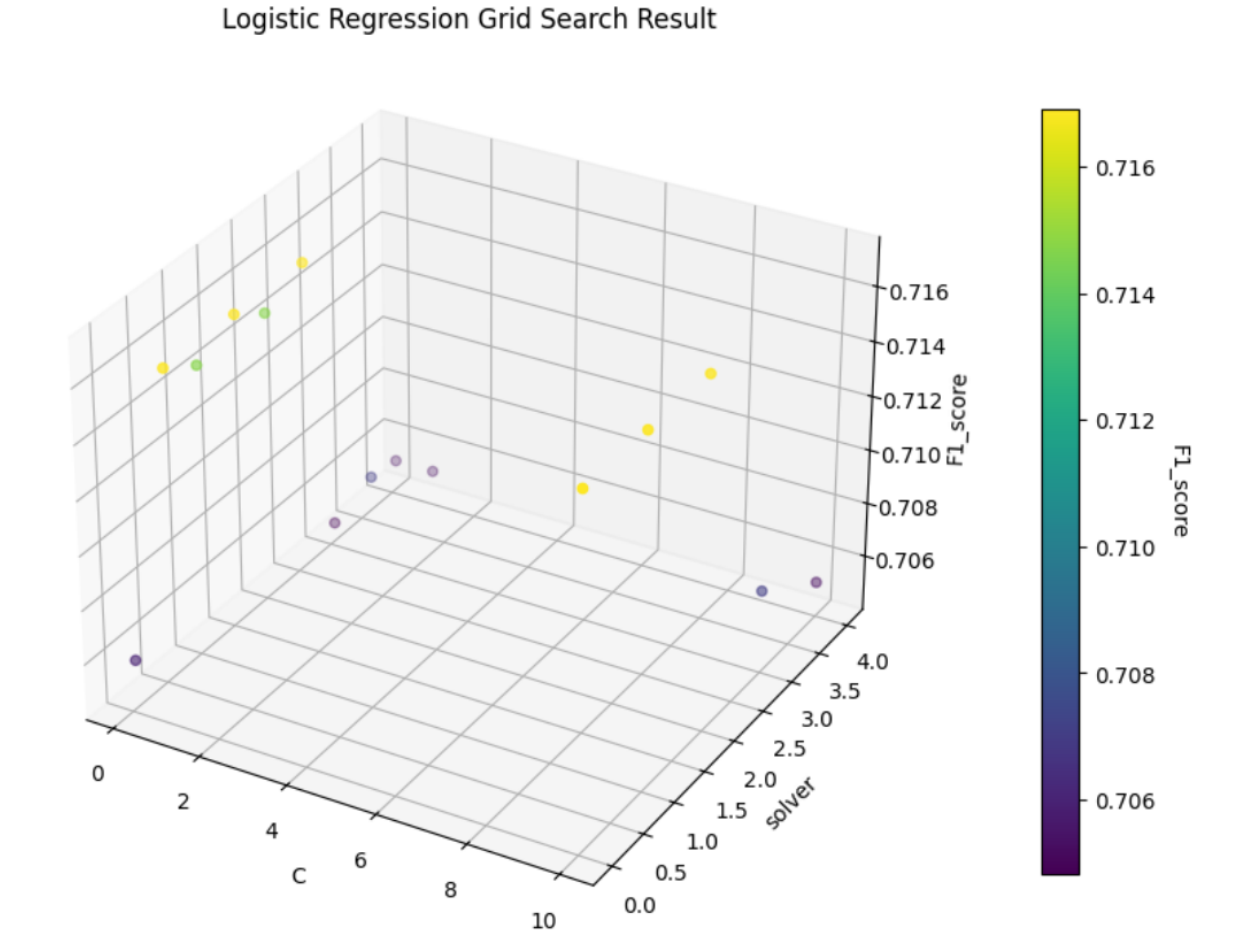
在未经处理的数据集上 f1分数=0.89

逻辑回归（Logistic Regression）

逻辑回归是一种用于解决分类问题的统计学习方法，主要用于二分类问题。尽管名为“回归”，但实质上是一种分类模型，通过将线性回归的结果经过一个逻辑函数（Sigmoid函数）转换为概率值，从而进行分类预测。其输出可以解释为属于某一类别的概率。

逻辑回归中值得关注的参数包括：C和solver。对这两个参数进行带有交叉验证的网格搜索结果如下

```
best_params = {'C': 1, 'solver': 'liblinear'}
f1_score_value = 0.6951219512195121
```



得到最优参数组合`'C': 1, 'solver': 'liblinear'`

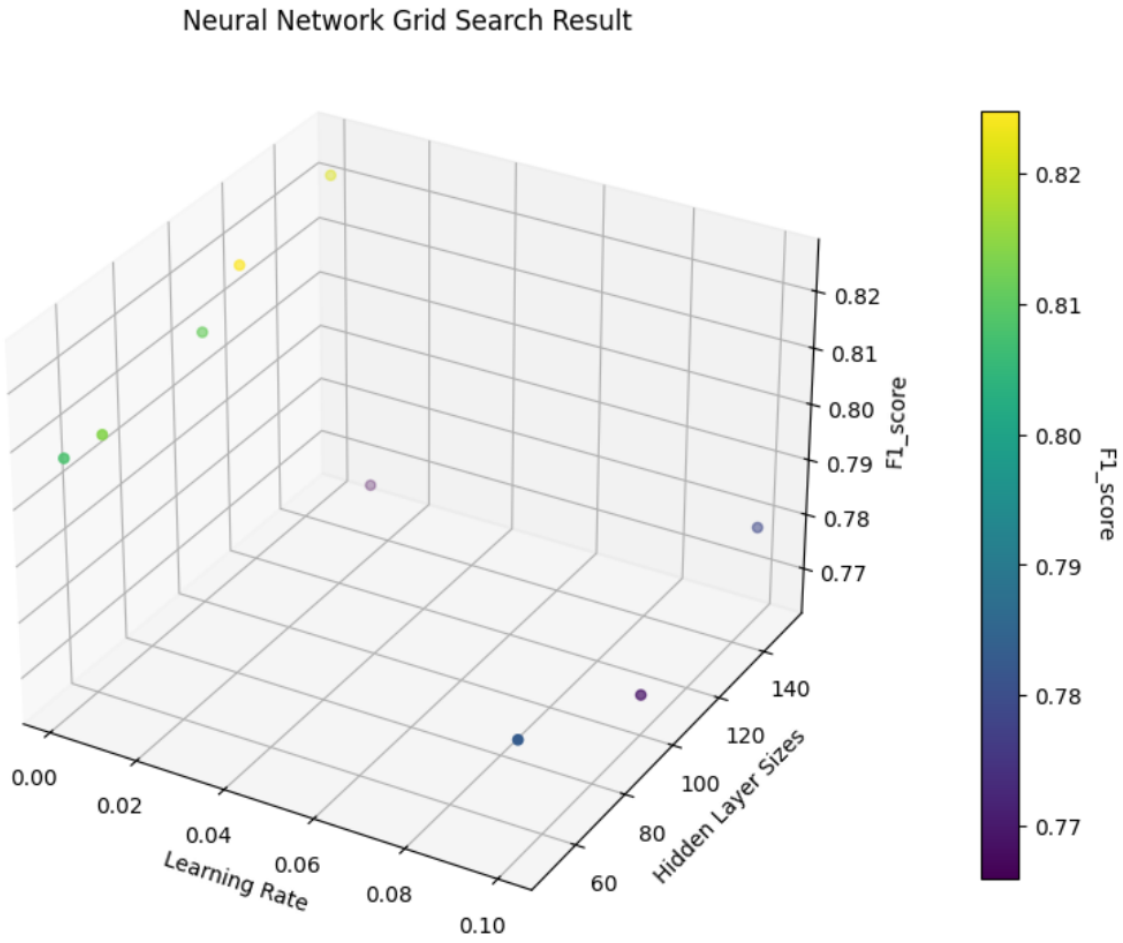
在未经处理的数据集上 f1分数=0.69

人工神经网络 ANN

人工神经网络是一种模仿生物神经网络结构和功能的计算系统。它由大量的人工神经元组成，通过学习和训练能够识别模式、进行分类和预测。本次实验使用的是仅有三层的简单mlp，包括输入层、隐藏层和输出层。

人工神经网络中值得关注的参数包括：hidden_layer_sizes和learning_rate。对这两个参数进行带有交叉验证的网格搜索结果如下

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
best_params = {'hidden_layer_sizes': (100,), 'learning_rate_init': 0.01}
f1_score_value = 0.7381974248927038
```



得到最优参数组合`'hidden_layer_sizes': (100,), 'learning_rate_init': 0.01`

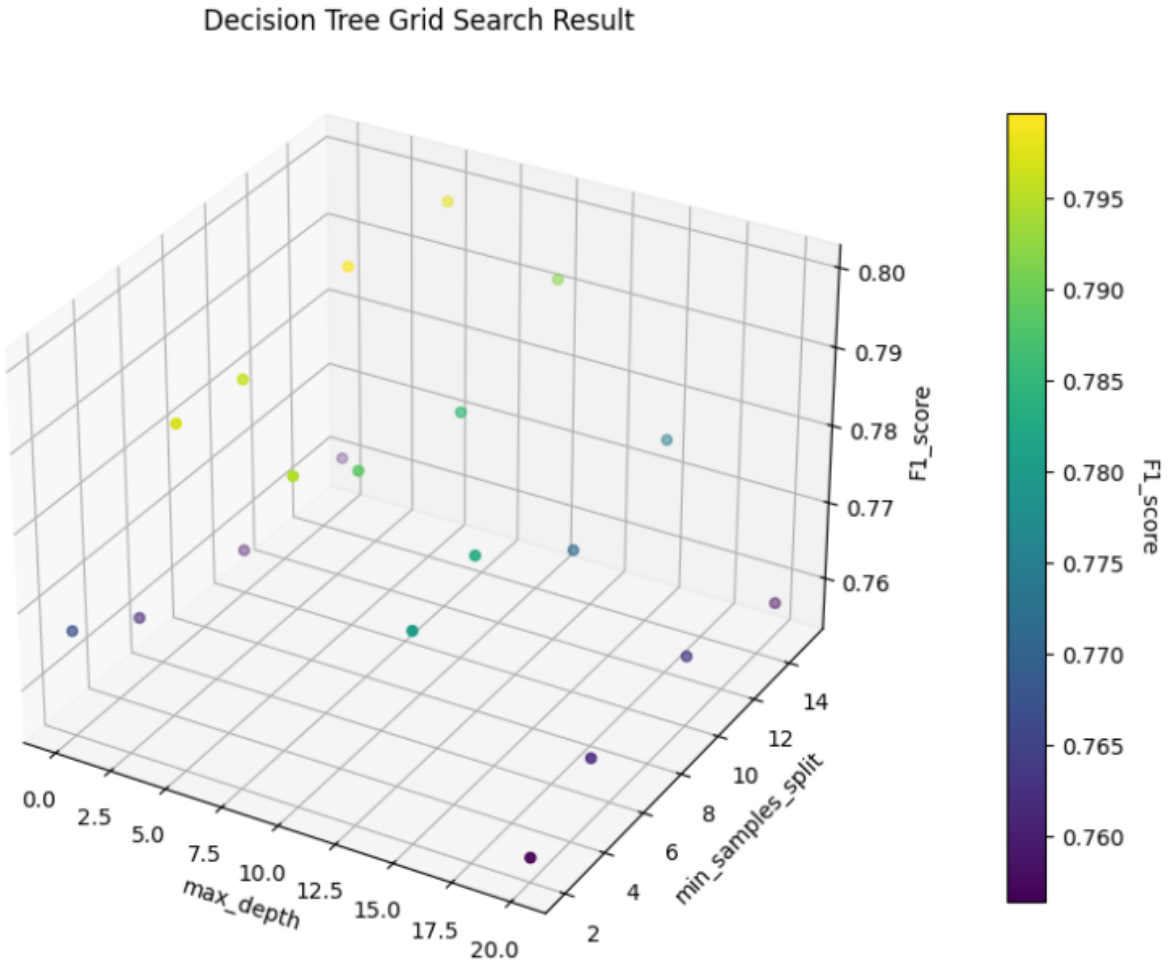
在未经处理的数据集上 f1分数=0.74

决策树 Decision Tree

决策树是一种机器学习算法，通过树状结构进行分类或回归分析。每个节点代表一个特征，分支表示特征取值，叶节点表示输出。通过逐步选择最优特征进行分割，决策树能够有效地学习数据模式，易于理解和解释。

决策树中值得关注的参数包括：max_depth和min_samples_split。对这两个参数进行带有交叉验证的网格搜索结果如下

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
best_params = {'max_depth': 5, 'min_samples_split': 10}
f1_score_value = 0.8415300546448088
```



得到最优参数组合`'max_depth': 5, 'min_samples_split': 10`

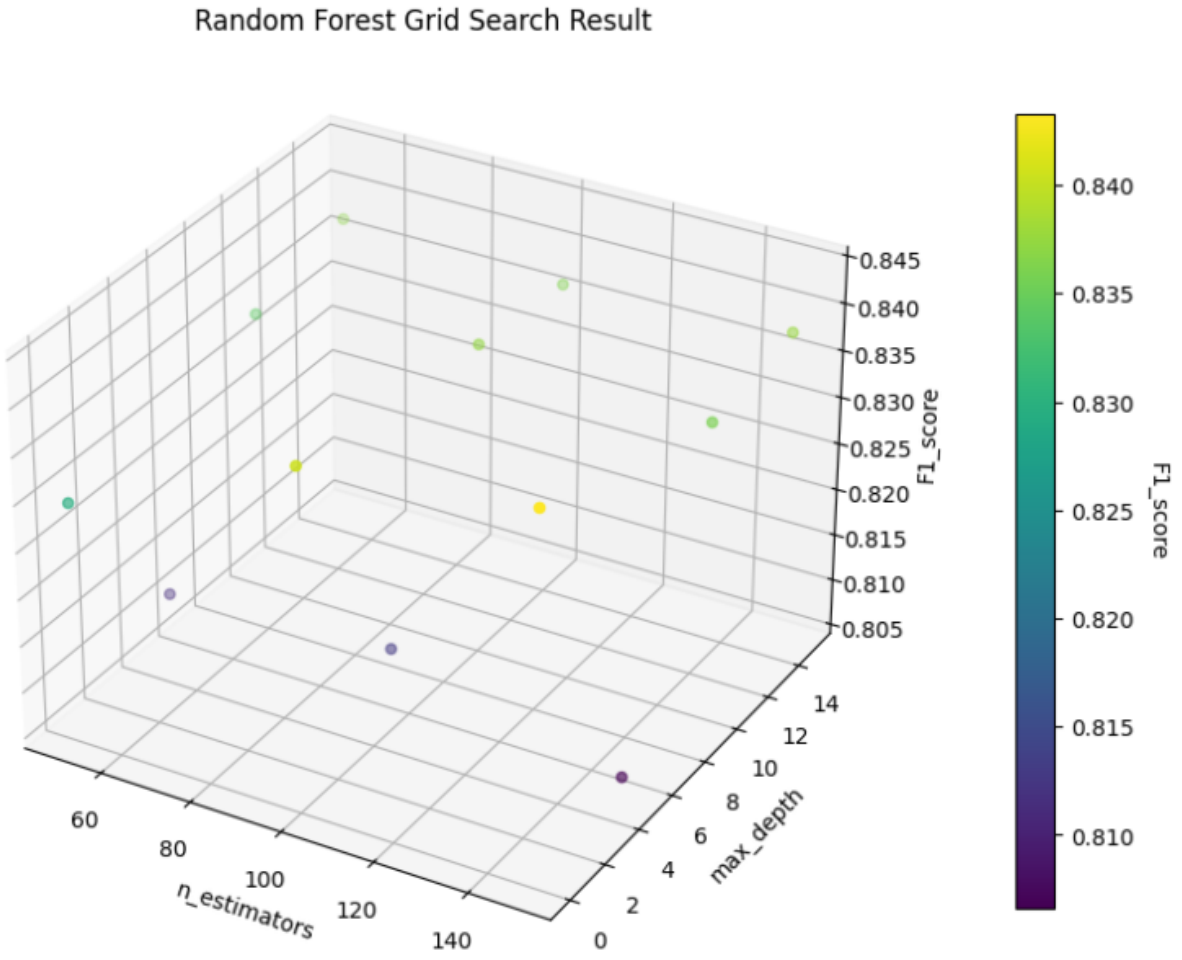
在未经处理的数据集上 f1分数=0.84

随机森林 Random Forest

随机森林是一种集成学习算法，通过构建多个决策树并随机选择特征进行训练，提高模型性能和泛化能力。通过投票或平均输出，随机森林降低过拟合风险，适用于多领域的分类和回归问题。

随机森林中值得关注的参数包括：n_estimators和max_depth。对这两个参数进行带有交叉验证的网格搜索结果如下

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
best_params = {'max_depth': None, 'n_estimators': 150}
f1_score_value = 0.8505747126436782
```



得到最优参数组合`'max_depth': None, 'n_estimators': 150`

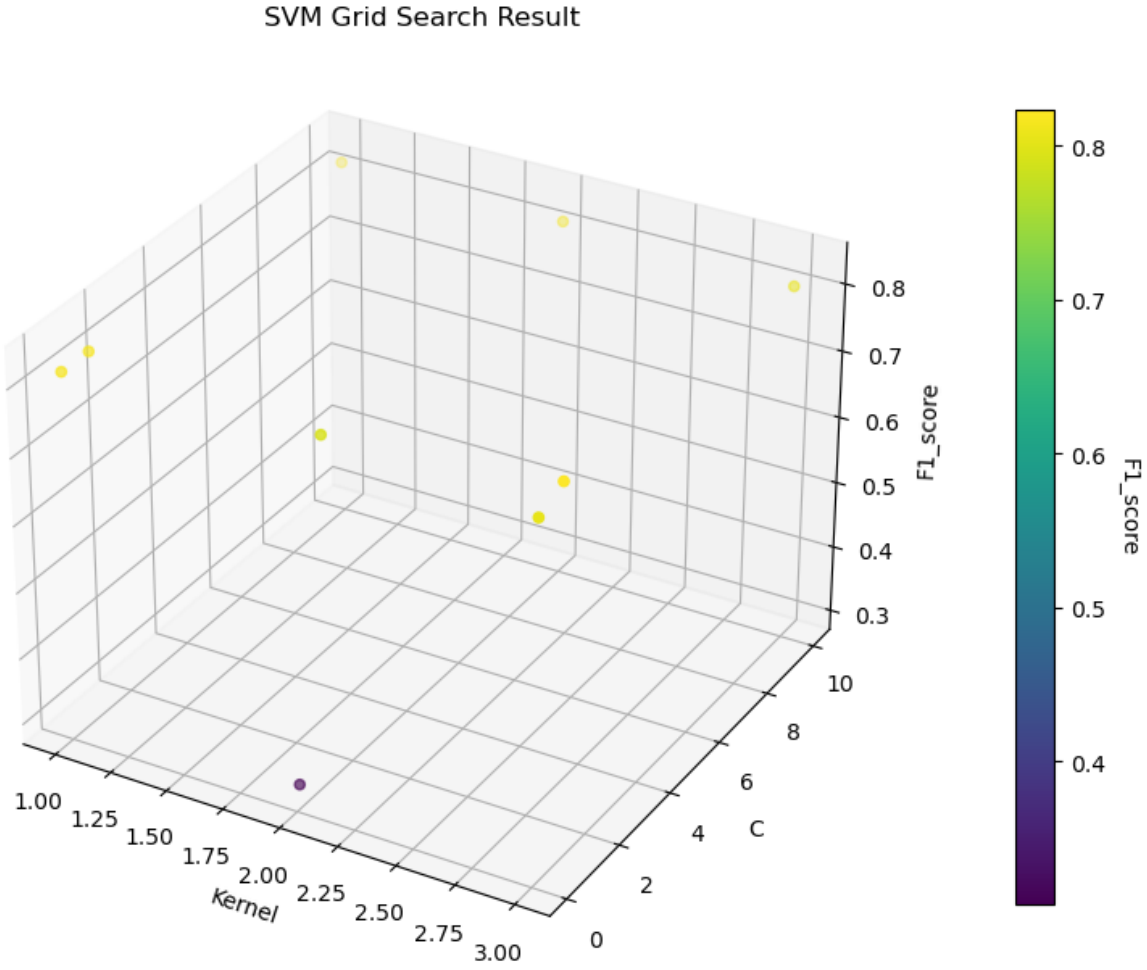
在未经处理的数据集上 f1分数=0.85

支持向量机 Support Vector Machines

支持向量机（SVM）是一种监督学习算法，用于分类和回归。它通过找到数据间的最优超平面来分隔不同类别，最大化间隔。SVM适用于高维空间，有效处理复杂数据结构，如非线性和线性可分问题。

SVM中值得关注的参数包括：kernel和C。对这两个参数进行带有交叉验证的网格搜索结果如下

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
best_params = {'C': 1, 'kernel': 'poly'}
f1_score_value = 0.8156424581005587
```



得到最优参数组合`'C': 1, 'kernel': 'poly'`

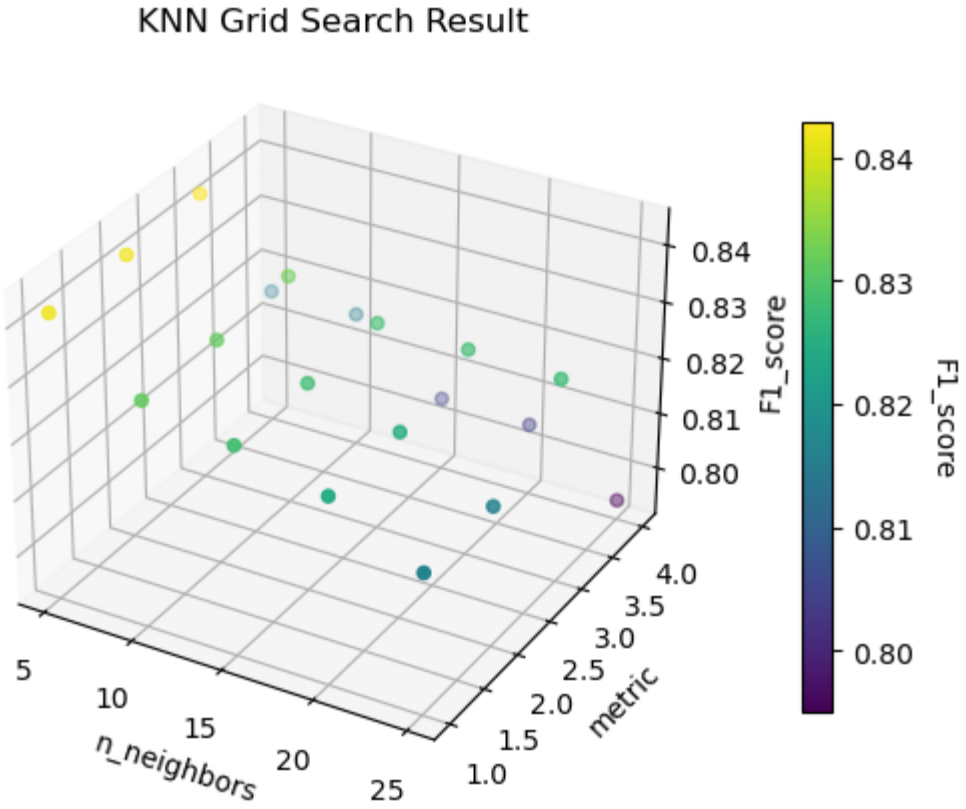
在未经处理的数据集上 f1分数=0.82

K最近邻 K-Nearest Neighbors

K最近邻（KNN）是一种基于实例的机器学习算法，通过测量数据点与邻近点的距离来进行分类或回归。它利用K个最近的邻居的投票或平均值来预测新数据点的标签或值。

KNN中值得关注的参数包括：n_neighbors和metric。对这两个参数进行带有交叉验证的网格搜索结果如下

```
best_params {'metric': 'manhattan', 'n_neighbors': 5}
f1_score_value 0.8666666666666666
```



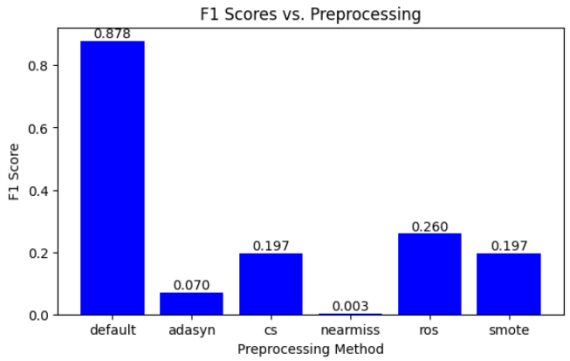
得到最优参数组合`'metric': 'manhattan', 'n_neighbors': 5`

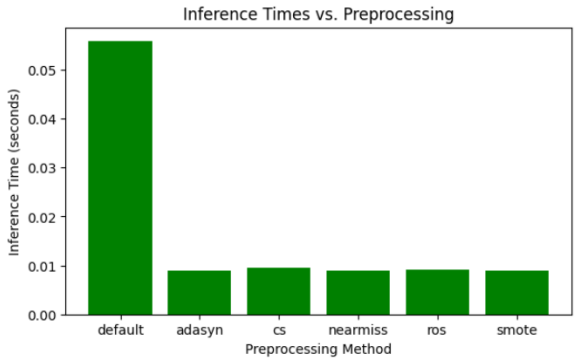
在未经处理的数据集上 f1分数=0.87

模型在不同预处理的数据集下的表现

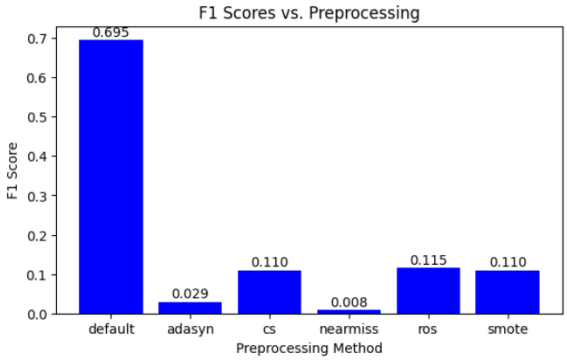
利用前文介绍的预处理方法，得到了**'default', 'adasyn', 'cs', 'nearmiss', 'ros', 'smote', 'fullfeature'**，7个不同预处理过的训练集，再将上述模型在这些训练集上训练，并在统一的测试集上测试，从而可以观察模型在不同预处理的数据集下的表现。

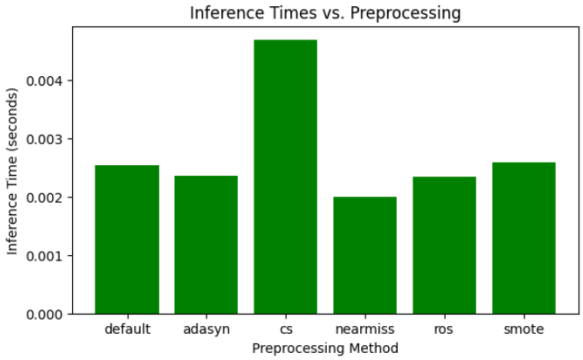
- 梯度提升 XG Boost



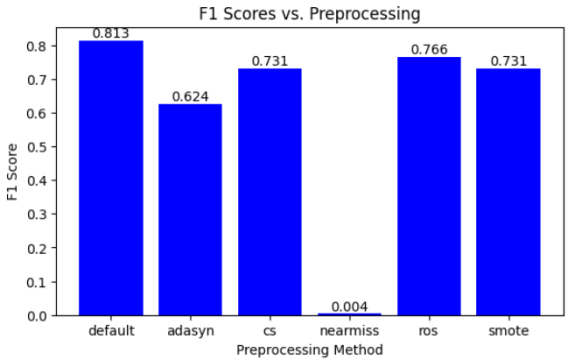


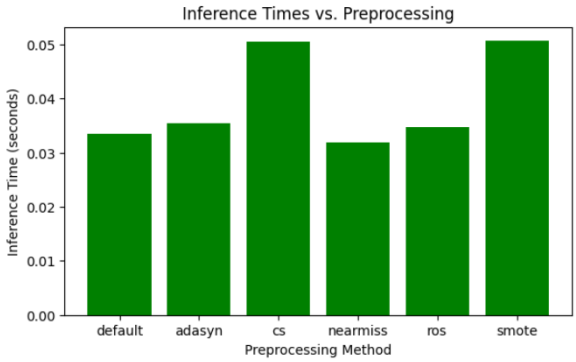
- 逻辑回归 Logistic Regression



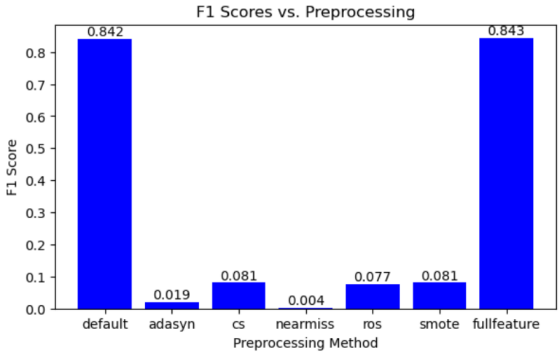


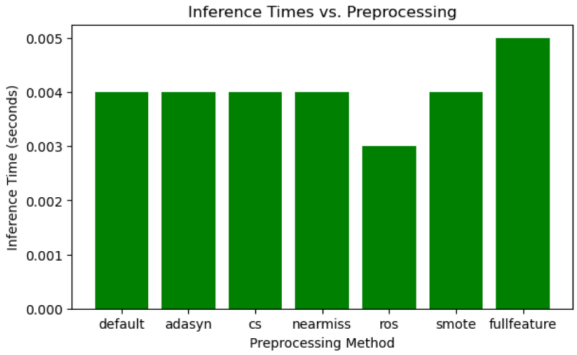
- 人工神经网络 Artificial Neural Networks



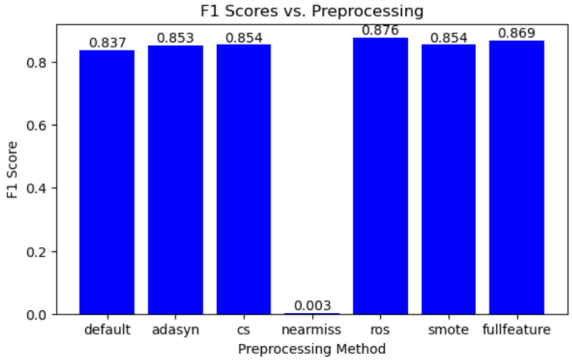


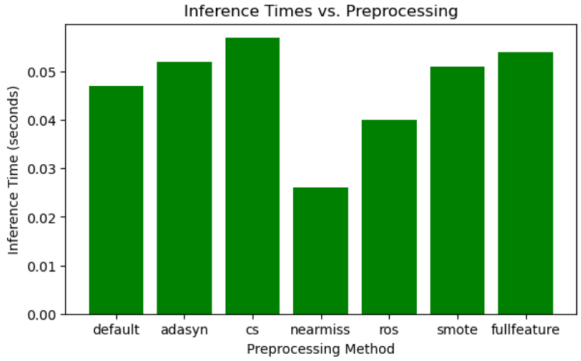
- 决策树 Decision Tree



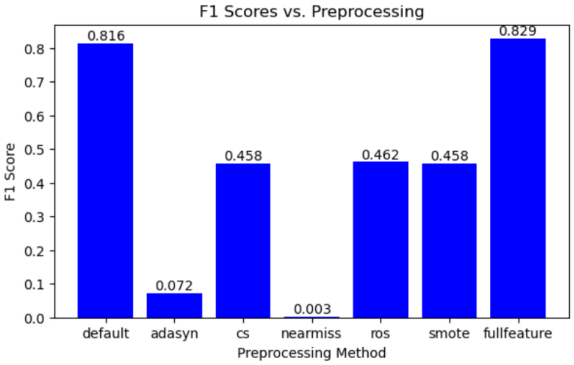


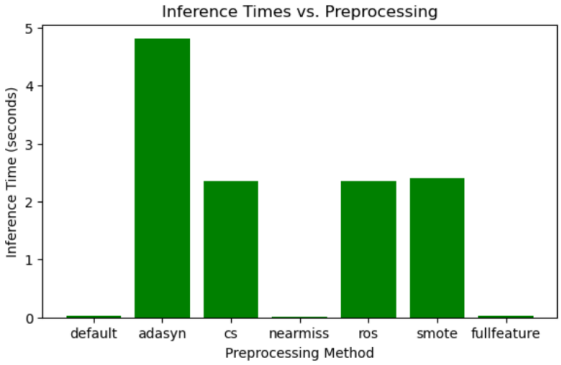
- 随机森林 Random Forest



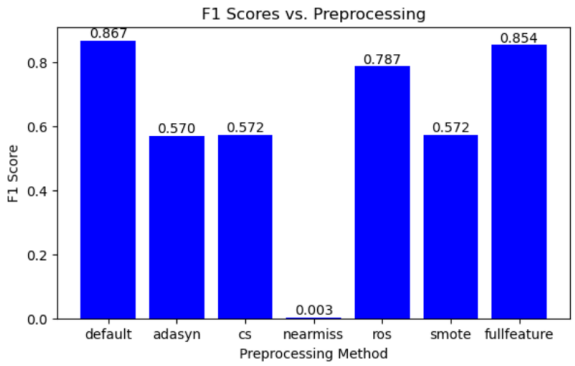


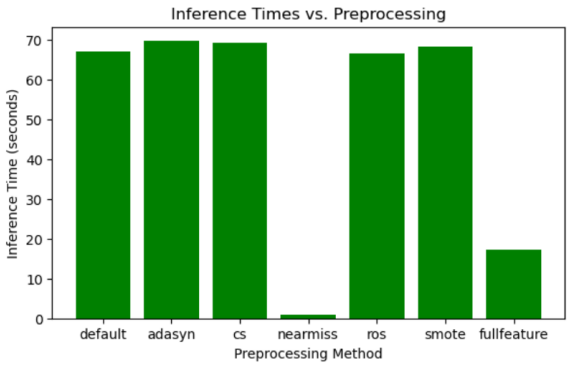
- 支持向量机 SVM





- K最近邻 KNN





总结

模型横向比较：

Model	Best Performing Dataset	Best F1 Score
Gradient Boosting (XG Boost)	Default	0.897
Logistic Regression	Default	0.698
Artificial Neural Networks	Default	0.813
Decision Tree	ROS	0.843
Random Forest	ROS	0.829
Support Vector Machine (SVM)	Full Feature	0.843
K-Nearest Neighbors (KNN)	Default	0.867

数据集纵向比较：

Dataset	Best Model	Best F1 Score
Default	XG Boost	0.878
Adasyn	Random Forest	0.853
CS	Random Forest	0.854
NearMiss	Logistic Regression	0.008
ROS	Random Forest	0.876
SMOTE	Random Forest	0.854
Full Feature	XG Boost	0.897

最佳模型：

后续我们还使用 XGBoost、ANN、Logistic Regression 等算法在 full feature 的数据集下进行了测试，测试结果得到在**对于 full feature 数据集进行随机过采样后的数据集使用 XGBoostClassifier** 进行训练可以得到最好的结果，**f1 分数为 0.897** ，推理时间在 0.1 s 左右如下：

```
Before ROS: Counter({0: 227451, 1: 394})
After ROS: Counter({0: 227451, 1: 227451})
F1 Score: 0.897297297297297
```

OneAPI AI Kit 使用与加速分析

Intel® Distribution of Modin

本次项目使用了 **Intel® Distribution of Modin** 中提供的加速后的 pandas 库，它包含于 suite of Intel® AI and machine learning development tools and resourses。它的使用大大的加速了对于数据集的读取过程，因为它可以实现多核地调用资源。

以下是使用普通版本 pandas 以及 modin.pandas 的性能对比：（数据集即为实验给出的数据集）

```
normal_pd loading time: 1.4287314414978027
modin_pd loading time: 0.16702771186828613
speed up: 855.3858671215377%
```

可见使用了 modin.pandas 后，速度提升了8倍多，非常地可观。

Intel® Daal4py

- 在进行数据的归一化时，使用了 daal4py 提供的 normalization_zscore 接口进行处理，经过比较（比较对象为 sklearn 库提供的原始 StandardScaler，默认使用 z-score 归一化）性能提升很大，如下图所示：

```
Time taken by scikit-learn:  0.255415678024292
Time taken by Intel DAAL:  0.07747030258178711
speed up: 329.6949534366979%
```

- 在进行离群点检测时，使用 k-means 算法处理时调用 daal4py.kmeans_init, daal4py.kmeans_compute 运行时间也得到了很大提升

Intel® Extension For Scikit-Learn*

在机器学习模型训练、调参、测试过程中都使用了**Intel® Extension For Scikit-Learn***，大大提高了模型的训练和推理速度

模型训练：

下方图片以 **XGBoost 模型**训练过程，展示了 **Intel® Extension For Scikit-Learn*** 的加速效果，经过计算，**加速比 speed up 为 127.36%，训练时间缩短了 20.9% 十分有效**

```
Model: xgboost
Preprocess: default
Optimize: off
Training Time: 1.3453867435455322 seconds

Intel(R) Extension for Scikit-learn* enabled
(https://github.com/intel/scikit-learn-intelex)
Model: xgboost
Preprocess: default
Optimize: on
Training Time: 1.063979148864746 seconds
```

下方图片以 **决策树** 模型训练过程，展示了 **Intel® Extension For Scikit-Learn*** 的加速效果，经过计算，**加速比 speed up 为 143.34%，训练时间缩短了 30.2% 十分有效**

```
Model: decision_tree
Preprocess: default
Optimize: off
Training Time: 7.924038887023926 seconds

Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
Model: decision_tree
Preprocess: default
Optimize: on
Training Time: 5.528069496154785 seconds
```

模型推理：

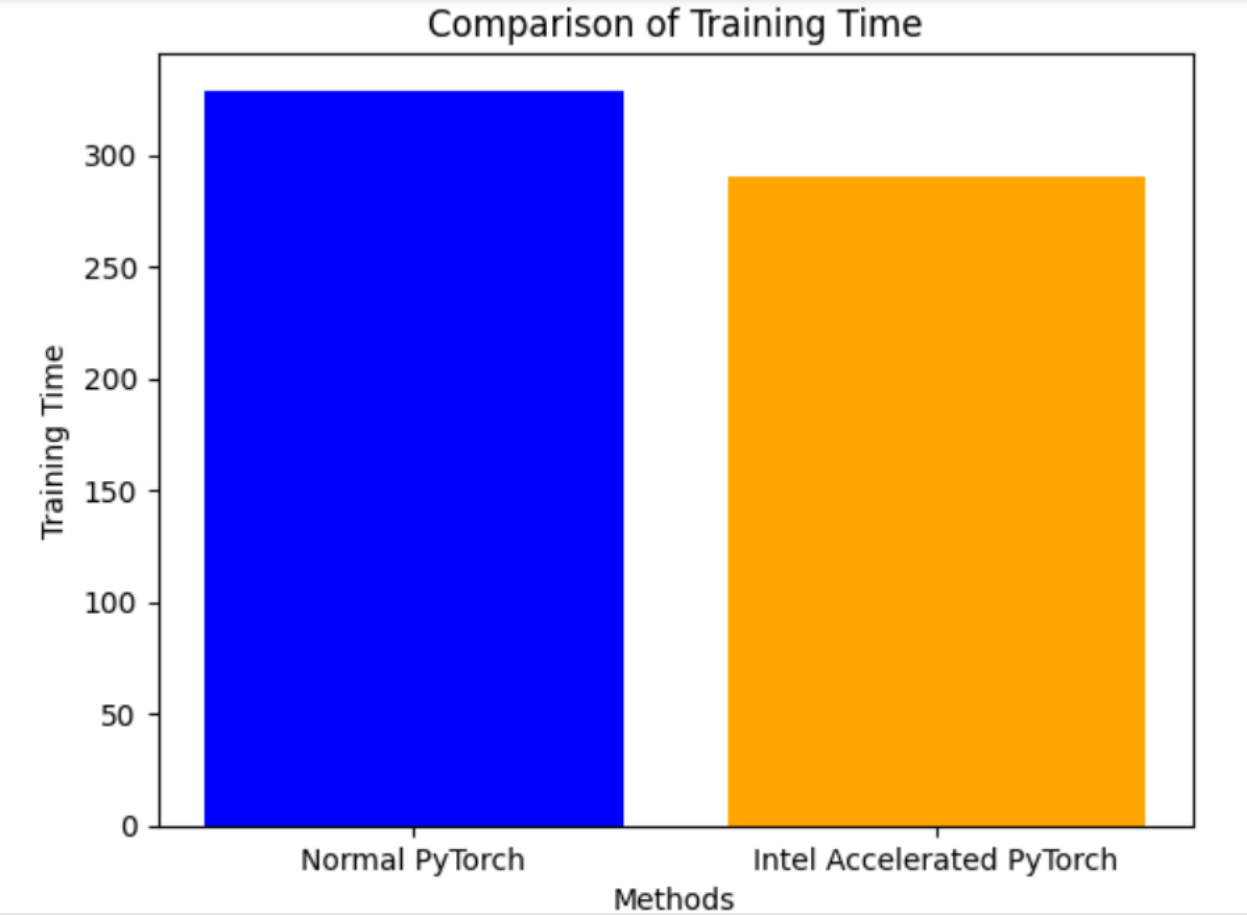
下方图片以 **决策树** 模型训推理过程，展示了 **Intel® Extension For Scikit-Learn*** 的加速效果，经过计算，**加速比 speed up 为 1880.67%，推理时间缩短了 94.68% 十分有效**

```
model_dt
F1 Score: 0.8415300546448088
Inference Time: 0.09370851516723633 seconds

model_dt_opt
F1 Score: 0.8415300546448088
Inference Time: 0.004982709884643555 seconds
```

Intel® Extension for PyTorch*

我们使用了基本的三层 MLP 进行预测的尝试，并对于是否使用 **Intel® Extension for PyTorch*** 提供的模型加速进行了实验分析，**结果得到训练加速比 speed up 为 113.15%，训练时间缩短了 11.62%** ，效果比较明显，如下图所示



总结

在实践中我们发现，Intel OneAPI 的使用可以在一整个机器学习项目的工作周期中给用户提供很大的帮助.

从加速读取数据的 **modin.pandas** → 加速数据预处理 **daal4py 库**的预处理方法 → 算法与模型层面 **sklearnex / ipex** 加速，效果都肉眼可见，并且使用很方便，以 sklearnex 为例，无需改变原来代码的结构，只需要两行代码即可完美适配优化后的算法，带来了非常大的便利。