信用卡交易检测

参考：https://github.com/oneapi-src/credit-card-fraud-detection

一、项目介绍

本项目由个人实现，主要目的在于比较 原生sklearn库 与 intel优化sklearn库 中的部分算法（dbscan 和 random_forest）。

已上传到 github：https://github.com/HBY-STAR/credit-card-fraud-detection_intel

问题描述：

2021 年，与信用卡欺诈相关的损失超过 120 亿美元，同比增长近 11%。就重大财务损失、信任和信誉而言，这是银行、客户和商户面临的一个令人担忧的问题。 电子商务相关欺诈一直在以约 13% 的复合年增长率 (CAGR) 增加。由于欺诈性信用卡交易急剧增加，在交易时检测欺诈行为对于帮助消费者和银行非常重要。机器学习可以通过训练信用卡交易模型，然后使用这些模型更快、更准确地检测欺诈交易，在预测欺诈方面发挥至关重要的作用。

项目简历：（解决方案）

分别基于 xgboost 和 random_forest 算法，给出了两种解决方案，代码分别在 src_xgboost 和 src_randomforest 中。

环境如下：

```
# 在其他系统上运行需更改源代码中的文件路径格式
OS                              Windows

Python                          3.11.4
joblib                          1.2.0
numpy                           1.24.3
pandas                          2.1.1
pathlib                         1.0.1
scikit-learn                    1.2.2
xgboost                         2.0.0
```

数据预处理：由于数据集高度不平衡，故尝试通过聚类分析选出一些簇来进行训练，并比较使用这些簇与使用所有数据训练模型的性能。本项目通过dbscan算法进行聚类。使用的库为：sklearn.cluster.DBSCAN。

```
# flag: True则使用intel加速方案，False则使用原生sklearn
def DBSCAN_Clustering(data_raw, features_of_interest, epsilon, min_samp, flag):
    if flag:
        from sklearnex import patch_sklearn
        patch_sklearn()
    from sklearn.cluster import DBSCAN
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    data_for_clustering = data_raw[features_of_interest]
    data_for_clustering_scaled = scaler.fit_transform(data_for_clustering)
    lst_clustering_time = []
    #可训练多次取最小值来使得训练时间更稳定
    start_time = time.time()
    db = DBSCAN(eps=epsilon, min_samples=min_samp, n_jobs=-1).fit(data_for_clustering_scaled)
    lst_clustering_time.append(time.time()-start_time)
    clustering_time = min(lst_clustering_time)
    data_for_clustering['Clusters'] = db.labels_
    return data_for_clustering, clustering_time
```

模型训练：本项目分别使用 xgboost 和 random_forest 算法来进行模型训练，并尝试通过超参数调整优化模型。xgboost模块本身为intel加速方案。random_forest 通过 flag 来确定是否使用intel加速方案。使用的库为：sklearn.ensemble.RandomForestClassifier 和 xgboost。

```
# flag: True则使用intel加速方案，False则使用原生sklearn
def rf_model_train(df_for_training,class_for_training, param_dict,flag):
    if flag:
        from sklearnex import patch_sklearn
        patch_sklearn("random_forest_classifier")
    from sklearn.ensemble import RandomForestClassifier
    rf_clt = RandomForestClassifier(**param_dict, random_state=42,n_jobs=-1)
    lst_training_time = []
    #可训练多次取最小值来使得训练时间更稳定
    start_time = time.time()
    rf_clt.fit(df_for_training.drop(columns=['Clusters']), class_for_training)
    lst_training_time.append(time.time()-start_time)
    train_time = min(lst_training_time)
    return rf_clt, train_time
```

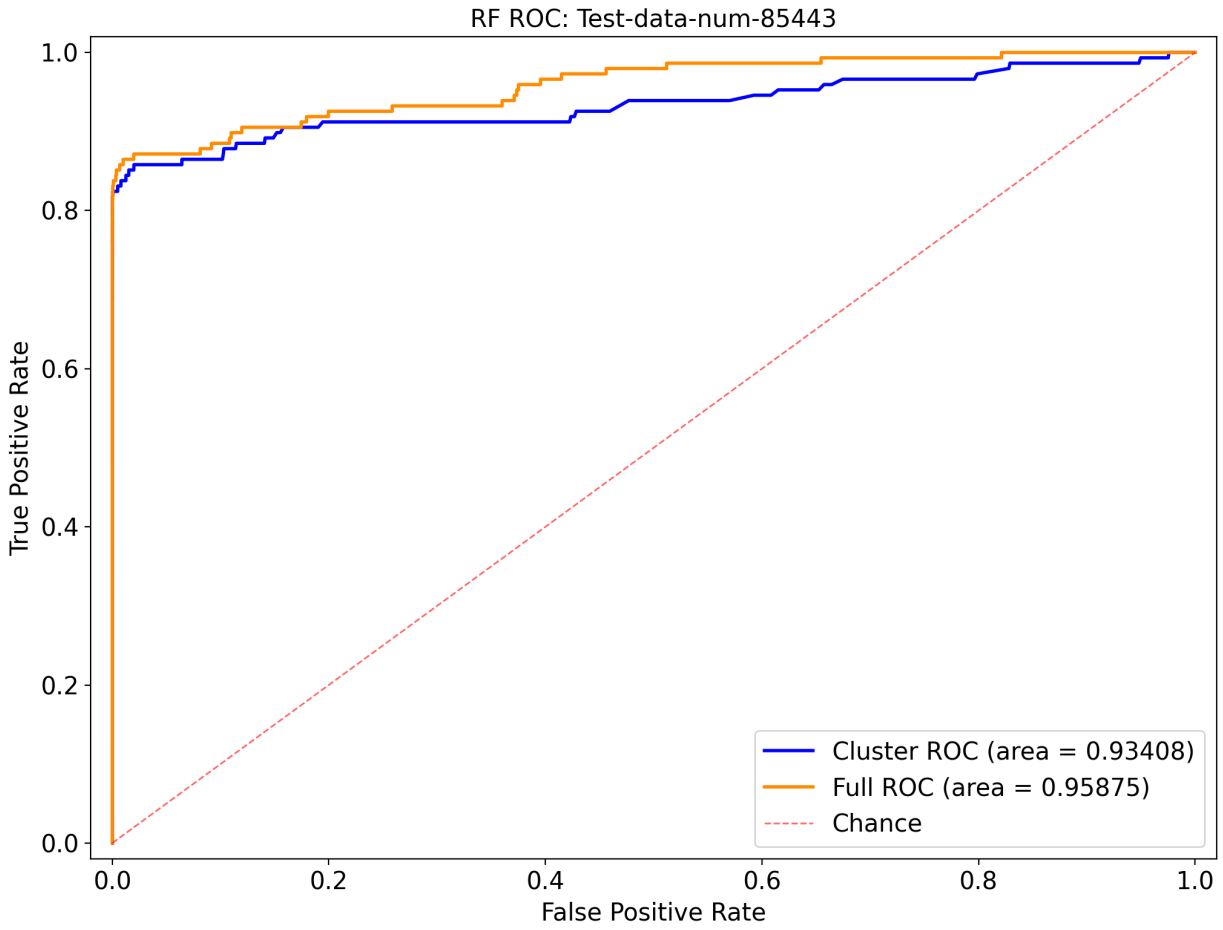
训练结果：保存在result文件夹中。分别使用 i5-11400H 和 i5-13600K 来进行训练。这两颗芯片分别为面向 Mobile 市场和面向 Desktop 市场的中端芯片。对于每种训练结果，分为log日志文件和roc曲线图片。

log日志中分别以 //Not patch 和 //patch 来标识使用 intel 加速方案与否的训练结果：

```
// Not patch 不使用intel加速方案
INFO:root:Train:
INFO:root:=======> Reading Data...
...

// patch 使用intel加速方案
INFO:root:Train:
INFO:root:=======> Reading Data...
...
```

图片为绘制的roc曲线，通过文件名标识是否使用 intel 加速方案。下面以 RF_ROC_patch.png 为例:



m_train.log, m_hyper.log 分别为运行 m_run_benchmarks_train.py 和 m_run_benchmarks_hyper.py 得到的日志文件，通过控制运行时参数 [-i] 来决定是否使用intel加速方案。

```
parser.add_argument('-i',
                    '--intel',
                    default=False,
                    action="store_true",
                    help="use intel accelerated technologies where available")
```

m_predict.log, ROC_not_patch.png, ROC_patch.png 为运行m_run_benchmarks_predict.py 得到的文件，通过控制运行时参数 [-i] 来决定是否使用intel加速方案。

二、结果分析

在使用 i5-11400H 和 i5-13600K的情况下使用 intel 加速方案对于 dbscan 和 random_forest 算法均有非常显著的提升，说明在笔记本和台式机上intel加速方案均可以很好地支持。下面以 i5-13600K 得到的结果进行具体分析：

（1）random_forest:

在执行 dbscan 时：

not patch: 158.078 s　　patch:5.717 s

not patch: 144.927 s　　patch:5.815 s

在训练 random_forest 时：

cluster data:（num:961）

not patch: 0.149 s　　patch:1.152 s

full data:（num:199364）

not patch: 25.101 s　　patch:0.960 s

在对 random_forest 进行超参数调整时：

cluster data:

not patch: 17.951 s　　patch:16.481 s

full data:

not patch: 2488.291 s　　patch:95.057 s

使用 random_forest 进行预测：

f1_score:(test_data_num: 854430)

cluster:

not patch: 0.918　　patch: 0.919

full:

not patch: 0.925　　patch: 0.921

roc_area:(test_data_num: 85443)

cluster:

not patch: 0.898　　patch: 0.933

full:

not patch: 0.968　　patch: 0.959

由以上数据可以很明显地看出，在数据量较大的情况下（full data），使用intel方案有非常大的速度提升，同时在模型的 f1_score 和 roc_area 上与原生sklearn相差无几。同时值得注意的是，在对于经过了 dbscan 聚类的数据上，对于roc曲线面积，使用 intel 加速方案有着较明显的性能提升（0.898 vs 0.933）。说明 intel 加速方案在较小但更具代表性的数据上有着更为出色的表现。当数据量较小时，虽然 intel 加速方案有时较慢，但这些数据的训练时间均只有不到一分钟的时间。在相比于将超参数调整时间从 2488s 优化到不到 100s 这样25倍的提升上，偶尔有不到1分钟的差距显然无关紧要。

（2）xgboost:

由于xgboost本身就为intel优化的库，因此仅作与随机森林算法进行对比用。训练结果数据也已给出，可自行浏览分析。在本项目中，xgboost比使用intel加速方案的random_forest算法还要快，且模型性能与随机森林算法相差不大。

三、总结

intel 的 oneAPI AI Kit 中的sklearn优化库无疑是进行机器学习的一大帮手，使用优化库中的算法可以极大提升训练模型的速度，而且几乎不需要对代码进行改动，只需要：

```
from sklearnex import patch_sklearn
patch_sklearn()
```

然后导入需要使用的包，就可以直接获得明显的训练速度提升。

因此下次在训练模型之前，不要忘了去浏览一下 oneAPI AI Kit，看看intel是否对你需要使用的算法进行了加速，如果有的话，记得 patch！