

# oneAPI项目实践报告

小组成员：金志吴，徐彦哲，张又文

## ▼ 问题陈述

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。

## ▼ 项目简介

本项目是一次基于官方机器学习库和英特尔 oneAPI AI分析工具包中数据分析和机器学习相关的优化库的代码实践，通过对问题的分析和机器学习模型的应用，借助提供的数据集进行训练和测试，以期完成信用卡交易欺诈检测这一项目任务

## ▼ 实践思路

在本次代码实践的过程中，首先我们注意到这个数据集的数据是极其不平衡的，具体来说阳性的数据占比只有大约0.1%，因此我们在代码实践前首先对提供的数据集进行预处理，包括数据清洗、特征选择和特征工程等步骤，尤其是运用了数据下处理的方法，进行数据平衡，以确保欺诈和非欺诈交易的样本数接近。

完成数据处理之后，开始着手对案例的分析处理：对原始数据进行特征选择，选择与欺诈检测相关的特征并提取这些基本的统计特征，如交易金额和交易时间。

在选择完统计特征之后，选择合适的机器学习算法（模型选择在“模型训练”模块详述，此处不再详述），将筛选过的数据集划分为训练集和测试集后使用训练集对所选取的不同模型各自进行训练并用测试集来评估各模型的性能（依照项目要求，主要以推理时间和F1分数为评价依据，同时我们小组还将准确度作为一个评价依据纳入考虑）

在完成初步的模型应用和测试评估后，结合评估结果对模型进行参数调整来优化模型并再次应用，以此实现对基于项目建构的模型的优化。

## ▼ 模型训练

在本次代码实践的过程中，我们选择构建六种不同类型的分类模型，即决策树、K-最近邻、逻辑回归、支持向量机、随机森林和XGBoost。如此选择的原因是这些模型的构建比较方便且在开设的课程上都有所介绍讲解，通过对不同模型、算法对同一案例的实现的比较可以实现对不同模型间的比较分析。

#### ▼ 决策树

在决策树模型中，使用“DecisionTreeClassifier”算法来建构模型。在算法构建过程中，通过设置最大深度来限制树最大分裂次数，再设置“criterion = 'entropy'”来决定何时停止分裂，最后将拟合模型后得到的预测值存储到“tree\_yhat”变量中

#### ▼ K-最近邻

K-最近邻模型使用KNeighborsClassifier算法建构模型，设置n\_neighbors的数值为5，而后将拟合模型后得到的预测值存储到knn\_yhat变量中

#### ▼ 逻辑回归

逻辑回归模型使用LogisticRegression算法并全部使用默认值，拟合模型后得到的预测值存入lr\_yhat变量中

#### ▼ 支持向量机

支持向量机模型使用SVC算法构建模型，和逻辑回归模型相同，此处也全部使用默认值，且默认内核就是“rbf”内核，之后把拟合模型后得到的预测值存入svm\_yhat变量中

#### ▼ 随机森林

随机森林模型使用RandomForestClassifier算法构建随机森林模型，之后的步骤类似决策树模型。随机森林和决策树模型之间的区别在于，决策树使用整个数据集来构建单个模型，随机森林则使用随机选择的特征来构建多个模型

#### ▼ XGBoost

XGBoost模型使用xgboost包提供的XGBClassifier算法构建模型，设置最大深度，最后将拟合模型后得到的预测值存入xgb\_yhat变量中

#### ▼ oneAPI的使用

在本次代码实践的过程中，我们小组对oneAPI的使用主要集中在并行计算和加速器利用这一方面。通过oneAPI利用不同硬件加速器的并行计算能力、加速数据处理和分析，在代码实践过程中对涉及的大量数据处理和模型计算进行任务并行化来提高计算效率。

一个具体的实践代码展示如下：

```
import os
import numpy as np
import warnings

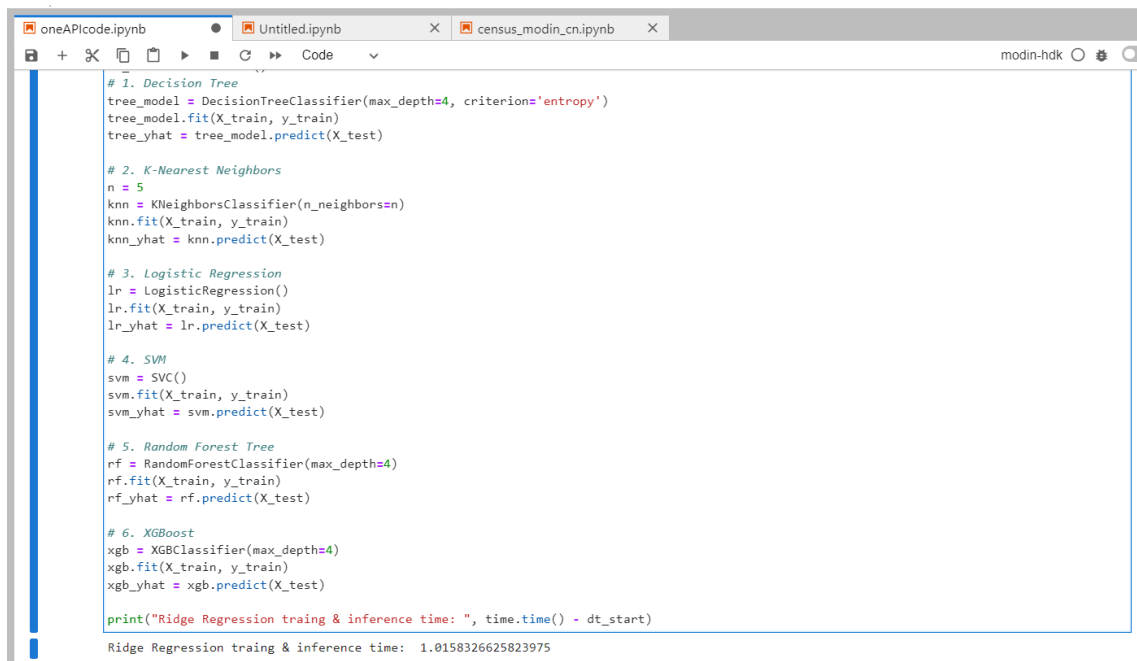
warnings.filterwarnings("ignore")
```

```
import pandas as pd
# import modin.pandas as pd
# import modin.config as cfg
# cfg.StorageFormat.put('hdk')
```

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

▼ 模型评估（此处均为oneAPI加速后的结果，加速前的结果将在对比分析模块展示）

▼ 推理时间



```
oneAPIcode.ipynb  Untitled.ipynb  X  census_modin_cn.ipynb  X  modin-hdk  [icons]

# 1. Decision Tree
tree_model = DecisionTreeClassifier(max_depth=4, criteria='entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)

# 2. K-Nearest Neighbors
n = 5
knn = KNeighborsClassifier(n_neighbors=n)
knn.fit(X_train, y_train)
knn_yhat = knn.predict(X_test)

# 3. Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)

# 4. SVM
svm = SVC()
svm.fit(X_train, y_train)
svm_yhat = svm.predict(X_test)

# 5. Random Forest Tree
rf = RandomForestClassifier(max_depth=4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)

# 6. XGBoost
xgb = XGBClassifier(max_depth=4)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)

print("Ridge Regression traing & inference time: ", time.time() - dt_start)
Ridge Regression traing & inference time: 1.0158326625823975
```

▼ 二分类准确度（F1分数）

```
F1 SCORE
F1 score of the Decision Tree model is 0.9010989010989011
F1 score of the KNN model is 0.9333333333333333
F1 score of the Logistic Regression model is 0.923076923076923
F1 score of the SVM model is 0.9090909090909091
F1 score of the Random Forest Tree model is 0.9111111111111111
F1 score of the XGBoost model is 0.923076923076923
F1 score time: 0.07437872886657715
```

---

## ▼ 精准度

```
ACCURACY SCORE
Accuracy score of the Decision Tree model is 0.9090909090909091
Accuracy score of the KNN model is 0.9393939393939394
Accuracy score of the Logistic Regression model is 0.9292929292929293
Accuracy score of the SVM model is 0.9191919191919192
Accuracy score of the Random Forest Tree model is 0.9191919191919192
Accuracy score of the XGBoost model is 0.9292929292929293
Accuracy score time: 0.3246936798095703
```

## ▼ 对比分析

### ▼ 和官方机器学习库的性能对比

在本次代码实践中，用官方的机器学习库得到的推理时间是：  
21.074803829193115，F1分数为：0.024142980575561523，精准度为：  
0.012557029724121094

在使用英特尔 oneAPI AI分析工具包中数据分析和机器学习相关的优化库后得到的推理时间为：1.0158326625823975，F1分数为：0.07437872886657715，精准度为：0.3246936798095703

具体展示如下：

```

# 1. Decision Tree
tree_model = DecisionTreeClassifier(max_depth=4, criterion='entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)

# 2. K-Nearest Neighbors
n = 5
knn = KNeighborsClassifier(n_neighbors=n)
knn.fit(X_train, y_train)
knn_yhat = knn.predict(X_test)

# 3. Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)

# 4. SVM
svm = SVC()
svm.fit(X_train, y_train)
svm_yhat = svm.predict(X_test)

# 5. Random Forest Tree
rf = RandomForestClassifier(max_depth=4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)

# 6. XGBoost
xgb = XGBClassifier(max_depth=4)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)

print("Ridge Regression traing & inference time: ", time.time() - dt_start)

```

Ridge Regression traing & inference time: 21.074803829193115

加速前的推理时间如上

```

# 1. Decision Tree
tree_model = DecisionTreeClassifier(max_depth=4, criterion='entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)

# 2. K-Nearest Neighbors
n = 5
knn = KNeighborsClassifier(n_neighbors=n)
knn.fit(X_train, y_train)
knn_yhat = knn.predict(X_test)

# 3. Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)

# 4. SVM
svm = SVC()
svm.fit(X_train, y_train)
svm_yhat = svm.predict(X_test)

# 5. Random Forest Tree
rf = RandomForestClassifier(max_depth=4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)

# 6. XGBoost
xgb = XGBClassifier(max_depth=4)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)

print("Ridge Regression traing & inference time: ", time.time() - dt_start)

```

Ridge Regression traing & inference time: 1.0158326625823975

加速后的推理时间如上

```
[40]: # 2. F1 score
dt_start = time.time()
print('F1 SCORE')
print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, tree_yhat)))
print('F1 score of the KNN model is {}'.format(f1_score(y_test, knn_yhat)))
print('F1 score of the Logistic Regression model is {}'.format(f1_score(y_test, lr_yhat)))
print('F1 score of the SVM model is {}'.format(f1_score(y_test, svm_yhat)))
print('F1 score of the Random Forest Tree model is {}'.format(f1_score(y_test, rf_yhat)))
print('F1 score of the XGBoost model is {}'.format(f1_score(y_test, xgb_yhat)))
print("F1 score time: ", time.time() - dt_start)

F1 SCORE
F1 score of the Decision Tree model is 0.9347826086956522
F1 score of the KNN model is 0.9574468085106385
F1 score of the Logistic Regression model is 0.9462365591397849
F1 score of the SVM model is 0.9333333333333332
F1 score of the Random Forest Tree model is 0.9555555555555556
F1 score of the XGBoost model is 0.9574468085106385
F1 score time: 0.024142980575561523
```

加速前运行得到的F1分数如上

```
F1 SCORE
F1 score of the Decision Tree model is 0.9010989010989011
F1 score of the KNN model is 0.9333333333333333
F1 score of the Logistic Regression model is 0.923076923076923
F1 score of the SVM model is 0.9090909090909091
F1 score of the Random Forest Tree model is 0.9111111111111111
F1 score of the XGBoost model is 0.923076923076923
F1 score time: 0.07437872886657715
```

加速后运行得到的F1分数如上

```
ACCURACY SCORE
Accuracy score of the Decision Tree model is 0.8787878787878788
Accuracy score of the KNN model is 0.9292929292929293
Accuracy score of the Logistic Regression model is 0.9292929292929293
Accuracy score of the SVM model is 0.9393939393939394
Accuracy score of the Random Forest Tree model is 0.9292929292929293
Accuracy score of the XGBoost model is 0.9494949494949495
Accuracy score time: 0.012557029724121094
```

加速前运行得到的精准度如上

```
ACCURACY SCORE
Accuracy score of the Decision Tree model is 0.9090909090909091
Accuracy score of the KNN model is 0.9393939393939394
Accuracy score of the Logistic Regression model is 0.9292929292929293
Accuracy score of the SVM model is 0.9191919191919192
Accuracy score of the Random Forest Tree model is 0.9191919191919192
Accuracy score of the XGBoost model is 0.9292929292929293
Accuracy score time: 0.3246936798095703
```

加速后运行得到的精准度如上

#### ▼ 多种算法结果的比较

在本次的代码实践过程中，我们用了决策树模型、K-最近邻、逻辑回归、支持向量机、随机森林和XGBoost等不同算法模型，并对其推理时间进行了比较，可以得到在本次代码实践的过程中，K-最近邻模型在推理时间上表现最好，而决策树模型在推理时间上表现最糟。

具体运行结果展示如下图：

```
Decision Tree traing & inference time: 0.7540731430053711
K-Nearest Neighbors traing & inference time: 0.02136826515197754
Logistic Regression traing & inference time: 0.031196117401123047
SVM traing & inference time: 0.029997587203979492
Random Forest Tree traing & inference time: 0.2108006477355957
XGBoost traing & inference time: 0.11529040336608887
```

#### ▼ 遇到的问题和收获感悟

##### ▼ 遇到的问题

在代码实践的过程中，第一次和第二次运行代码时不会报错，当第三次点击Run All Cells的时候遇到报错，但是用Restart kernel and Run All Cells则不会遇到这一问题。如果不用，就会报RuntimeError Traceback (most recent call last)报错。在网上查找资料后，得知可能是因为代码尝试从 DataFrame 中选择正常样本（Class 0）的随机子集，错误可能与 normal\_indices 是 Pandas 的 Index 对象有关，直接在其上应用 np.random.choice 可能导致问题。但是修改完之后，下面会报一系列RuntimeError的报错。再度查找了网上的资料，得知可能是由于Modin本身的问题，直接使用原生的 Pandas 库进行相关的操作可能是解决方法之一。

##### ▼ 收获与感悟

在本次代码实践的过程中，我们小组尝试在官方的机器学习库的基础上学习并使用了英特尔 oneAPI AI分析工具包中数据分析和机器学习相关的优化库。在使用实践的过程中，我们通过模型建构过程中的感悟和分析返还的运行结果评估指数，真切感受到英特尔 oneAPI AI分析工具包带来的性能上的优化，也认识到技术的优化能够对解决如本项目这样出现在实际生活中的问题带来的巨大帮助与便捷。与此同时，我们也能够对平时学习到的相关模型进行实践学习，得到了一个结合知识与实践的极好的机会。

通过本次的代码实践，我们团队感受到oneAPI提供的统一的编程模型带来的跨多个处理器架构进行项目构建的灵活性和可扩展性。这种灵活性和可扩展性在此次规模不大的项目中并不体现较大优势，但面对大规模数据集的处理和复杂算法的时候会起到很大的作用；此外，我们团队还切身感受到oneAPI工具和库提供的丰富功能，在此次项目处理的过程中还感受到了通过oneAPI来利用并行计算和加速器的计算能力，加快数据处理和分析的速度，从而实现更快更准确的欺诈检测。