《机器学习》 – 监督学习

华为技术有限公司

目录

1 实验介绍 3
1.1 实验目的 3
1.2 实验清单 3
1.3 开发平台介绍 4
2 APP 评分预测 5
2.1 实验介绍 5
2.1.1 简介 5
2.1.2 实验目的 5
2.2 实验环境要求 5
2.3 实验总体设计 5
2.4 实验步骤 6
2.4.1 数据读取 6
2.4.2 线性回归 7
2.4.3 **SVM 回归** 8
2.4.4 KNN 回归 9
2.4.5 思考题 10
2.4.6 模型评估与选择 11
2.5 实验小结 14
2.6 参考答案 14
3 银行存款预测 16
3.1 实验介绍 16
3.2 实验环境要求 16
3.3 实验总体设计 16
3.4 实验过程 16
3.4.1 上传数据集至 OBS 16
3.4.2 创建预测分析项目 17
3.4.3 数据标注并训练 18

3.4.4 模型部署 19
3.4.5 服务测试 19
3.5 实验小结 21
3.6 创新设计 21
4 糖尿病预测 23
4.1 实验说明 23
4.2 实验建模流程要求 23
4.2.1 环境要求 23
4.2.2 实验实现步骤要求 23
4.3 参考答案 25
5 信用违约预测 25
5.1 实验说明 25
5.2 实验建模流程要求 25
5.2.1 环境要求 25
5.2.2 实验实现步骤要求 25
5.3 参考答案 28

1 实验介绍

机器学习分为监督学习、无监督学习、半监督学习、强化学习。监督学习是指利用一组已知类别的样本调整分类器的参数，使其达到所要求性能的过程，也称为监督训练或有教师学习。分类和回归是监督学习中的两种典型任务。

本章实验涉及线性回归、SVM 回归、KNN 回归等回归算法，以及逻辑回归、决策树、随机森林等分类算法，通过不同算法的效果对比来加深对算法的理解。

本章实验难度包含：初级、中级、高级。

初级实验：银行存款预测实验

中级实验：APP 评分预测实验、糖尿病预测实验

高级实验：信用违约预测实验

1.1 实验目的

本章实验分为回归与分类两部分，通过本章实验的学习，您将能够：

了解华为云 ModelArts 自动学习服务

掌握回归与分类任务的区别与流程

掌握线性回归、SVM 回归、KNN 回归等回归算法的原理与使用

掌握逻辑回归、决策树、随机森林等分类算法的原理与使用

掌握模型评估的方法

1.2 实验清单

实验	简述	难度	软件环境	开发环境
APP 评分预测（回归+分类）	基于 APP 下载信息，分别使用线性回归、SVM 回归、KNN 回归对 APP 评分进行预测；进而将评分数据离散化，用决策树模型训练分类模型。	中级	Python3	本地 PC

银行存款预测（分类）	基于银行客户是否存款数据，使用 ModelArts 自动学习服务，实现是否存款预测。	初级	Python3	ModelArts
糖尿病预测（分类）	基于 Diabete 数据，使用逻辑回归进行糖尿病预测。	中级	Python3	本地 PC
信用违约预测（分类）	基于信贷数据，分别使用逻辑回归、随机森林算法实现信用违约预测。	高级	Python3	本地 PC

1.3 开发平台介绍

ModelArts 是面向开发者的一站式 AI 开发平台，为机器学习与深度学习提供海量数据预处理及半自动化标注、大规模分布式 Training、自动化模型生成，及端-边-云模型按需部署能力，帮助用户快速创建和部署模型，管理全周期 AI 工作流。

具体内容请参考平台介绍 PPT。

2 APP 评分预测

2.1 实验介绍

2.1.1 简介

本次实验利用数据预处理与特征工程中处理好的数据集来训练一个回归和分类模型，对评分的预测。

首先线性回归、SVM、KNN 回归算法，训练三个回归模型。接下来将评分数据离散化，用决策树模型训练分类模型。这样可以更好学习算法的使用和模型的评估。

2.1.2 实验目的

掌握线性回归算法的应用实践

掌握 SVM 算法的应用实践

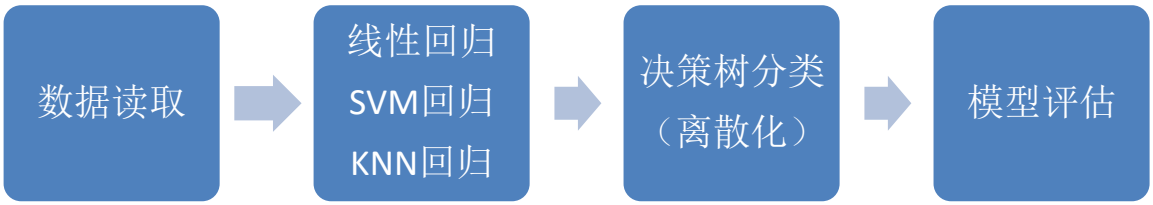
掌握决策树算法的应用实践

掌握 KNN 算法的应用实践

2.2 实验环境要求

本地 PC，Python3

2.3 实验总体设计



2.4 实验步骤

2.4.1 数据读取

代码：

```
#导入相关库
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
#读取数据集
data_after_pca = pd.read_csv('after_pca.csv',index_col=0)
data = pd.read_csv('AppDataV2.csv',index_col=0)
data_after_var = pd.read_csv("data_after_var",index_col=0)
data_after_filter = pd.read_csv("df_after_filter.csv",index_col=0)
#首先确定样本的数据的标签
X = data.drop(["Rating"],axis='columns')
Y = data["Rating"]
X_var = data_after_var.drop(["Rating"],axis='columns')
Y_var = data_after_var["Rating"]
X_pca = data_after_pca.drop(["Rating"],axis='columns')
Y_pca = data_after_pca["Rating"]
X_filter = data_after_filter.drop(["Rating"],axis='columns')
Y_filter = data_after_filter["Rating"]
X.info()
输出：
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10240 entries, 0 to 10239
Data columns (total 40 columns):
Reviews                                10240 non-null int64
Size                                   10240 non-null float64
Installs                               10240 non-null float64
Type                                   10240 non-null int64
Price                                  10240 non-null float64
Content Rating                         10240 non-null int64
Genres                                 10240 non-null int64
Category_ART_AND_DESIGN                10240 non-null int64
Category_AUTO_AND_VEHICLES             10240 non-null int64
```

```
Category_BEAUTY                          10240 non-null int64
Category_BOOKS_AND_REFERENCE      10240 non-null int64
...
```

代码：

```
#数据集划分
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 10)
```

2.4.2 线性回归

sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True)

参数说明：

fit_intercept：默认 True，是否计算模型的截距，为 False 时，则数据中心化处理。

normalize：默认 False，是否中心化，或者使用 sklearn.preprocessing.StandardScaler()。

copy_X：默认 True，否则 X 会被改写。

代码：

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import
mean_squared_error,mean_absolute_error,accuracy_score,accuracy_score,r2_score


#初始化线性回归模型
linreg = LinearRegression()
#训练模型
linreg.fit(X_train,y_train)
#训练集上的 MSE
linreg_pred_train = linreg.predict(X_train)
linreg_mse_train = mean_squared_error(linreg_pred_train,y_train)
#输出测试集上的测试结果
linreg_pred_test=linreg.predict(X_test)
linreg_mse_test = mean_squared_error(linreg_pred_test,y_test)
print("训练集 MSE：", linreg_mse_train)
print("测试集 MSE：", linreg_mse_test)
```

输出：

```
训练集 MSE：  0.2308388846541144
测试集 MSE：  0.22350603712434897
```

2.4.3 SVM 回归

sklearn.svm.SVR（kernel ='rbf'，degree = 3，gamma ='auto_deprecated'，coef0 = 0.0，tol = 0.001，C = 1.0，verbose = False，max_iter = -1 ）

参数说明：

kernel：指定要在算法中使用的内核类型。可以是'linear'，'poly'，'rbf'，'sigmoid'，'precomputed'或者 callable 之一，默认为 rbf。

degree：多项式 poly 函数的维度，默认是 3，选择其他核函数时会被忽略。

gamma： ‘rbf’，‘poly’ 和 ‘sigmoid’ 的核函数参数。默认是’ auto’ ，则会选择 1/n_features

coef0：核函数的常数项。对于 ‘poly’ 和 ‘sigmoid’ 有用，默认值= 0.0。

tol：停止训练的误差值大小，默认值= 1e-3。

C：惩罚参数，默认= 1.0。C 越大，相当于惩罚松弛变量，希望松弛变量接近 0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C 值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。

verbose：日志。

max_iter：最大迭代次数。-1 为无限制。

代码：

```
from sklearn.svm import SVR
from sklearn.metrics import accuracy_score
#初始化决策树模型
svr=SVR(kernel='rbf',C=1)
#训练
svr.fit(X_train,y_train_int)
#训练集上的 MSE
svr_pred_train = svr.predict(X_train)
svr_mse_train = mean_squared_error(svr_pred_train,y_train)
#输出测试集上的测试结果
svr_pred_test=svr.predict(X_test)
svr_mse_test = mean_squared_error(svr_pred_test,y_test)
print("训练集 MSE：", svr_mse_train)
print("测试集 MSE：", svr_mse_test)
```

输出：

```
训练集 MSE： 0.28852538417710893
测试集 MSE： 0.44229386180754227
```

2.4.4 KNN 回归

sklearn.neighbors.KNeighborsRegressor(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30)：

参数说明：

n_neighbors：knn 算法中指定以最近的几个最近邻样本具有投票权，默认参数为 5

algrithm：即内部采用什么算法实现。有以下几种选择参数：

'ball_tree':球树、

'kd_tree':kd 树、

'brute':暴力搜索、

'auto':自动根据数据的类型和结构选择合适的算法。默认情况下是 ‘auto’ 。

暴力搜索就不用说了大家都知道。具体前两种树型数据结构哪种好视情况而定。KD 树是对依次对 K 维坐标轴，以中值切分构造的树,每一个节点是一个超矩形，在维数小于 20 时效率最高 ball tree 是为了克服 KD 树高维失效而发明的，其构造过程是以质心 C 和半径 r 分割样本空间，每一个节点是一个超球体。一般低维数据用 kd_tree 速度快，用 ball_tree 相对较慢。超过 20 维之后的高维数据用 kd_tree 效果反而不佳，而 ball_tree 效果要好，具体构造过程及优劣势的理论大家有兴趣可以去具体学习。

leaf_size:这个值控制了使用 KD 树或者球树时， 停止建子树的叶子节点数量的阈值。这个值越小，则生成的 KD 树或者球树就越大，层数越深，建树时间越长，反之，则生成的 KD 树或者球树会小，层数较浅，建树时间较短。默认是 30.

请根据线性回归的实现和 KNN 的参数说明，训练一个 KNN 模型。代码填写：

```
#初始化 knn 模型
knn_model = KNeighborsRegressor(n_neighbors=50)
#训练
knn_model.fit(X_train,y_train)
#训练集上的 MSE
knn_pred_train = knn_model.predict(X_train)
knn_mse_train = mean_squared_error(knn_pred_train,y_train)
#输出测试集上的测试结果
knn_pred_test=knn_model.predict(X_test)
knn_mse_test = mean_squared_error(knn_pred_test,y_test)
print("训练集 MSE：", knn_mse_train)
print("测试集 MSE：", knn_mse_test)
```

输出：

```
训练集 MSE： 0.2044843076171875
```

```
测试集 MSE:  0.20693282421875
```

接下来简单对三个模型的输出精度进行对比。

代码：

```
model_mse = pd.DataFrame(data=[[linreg_mse_train,knn_mse_train,svr_mse_train],
                               [linreg_mse_test,knn_mse_test,svr_mse_test]],
        columns=['Logistic_Regression','knn','SVR'],index=["training set","test set"])
model_mse
```

输出：

	Logistic_Regression	knn	SVR
training set	0.230839	0.204484	0.270277
test set	0.223506	0.206933	0.494534

代码：

```
plt.figure(figsize=(20, 10))
model_mse.plot(kind = 'bar')
```

输出：

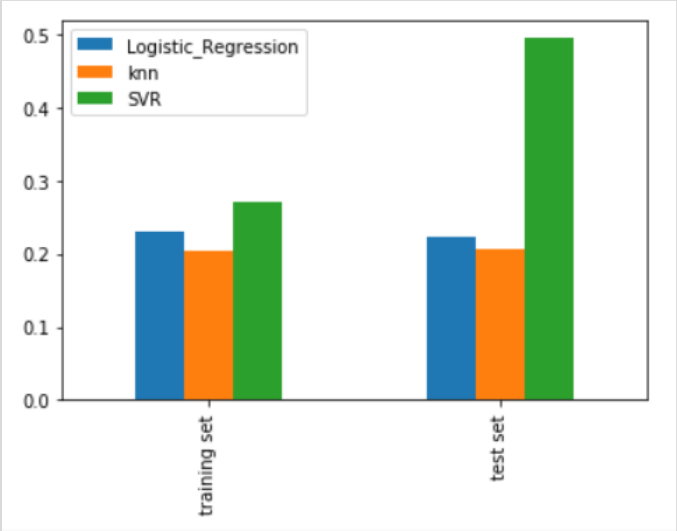


图2-1 LR、KNN、SVR 算法对比

2.4.5 思考题

将数据的标签变为整型，自行编码训练一个决策树分类模型，其中决策树模型使用方法如下：

DecisionTreeClassifier(criterion="mse",splitter="best",max_depth=None,min_samples_split=2,min_samples_leaf=1,min_weight_fraction_leaf=0.,
max_features=None,random_state=None,max_leaf_nodes=None)

参数说明：

criterion:切分质量的评价准则。默认为'mse'(mean squared error)。

splitter:指定了在每个节点切分的策略。有两种切分策略:

(1).splitter='best':表示选择最优的切分特征和切分点。

(2).splitter='random':表示随机切分。

max_depth:指定树的最大深度。如果为 None，则表示树的深度不限，直到每个叶子都是纯净的。

min_samples_split:默认为 2。它指定了分裂一个内部节点(非叶子节点)需要的最小样本数。如果为浮点数(0 到 1 之间)，最少样本分割数为 ceil(min_samples_split * n_samples)

min_samples_leaf:指定了每个叶子节点包含的最少样本数。如果为浮点数(0 到 1 之间)，每个叶子节点包含的最少样本数为 ceil(min_samples_leaf * n_samples)

min_weight_fraction_leaf:指定了叶子节点中样本的最小权重系数。默认情况下样本有相同的权重。

max_feature:

(1).如果是整数，则每次节点分裂只考虑 max_feature 个特征。

(2).如果是浮点数(0 到 1 之间)，则每次分裂节点的时候只考虑 int(max_features * n_features)个特征。

(3).如果是字符串'auto',max_features=n_features。

(4).如果是字符串'sqrt',max_features=sqrt(n_features)。

(5).如果是字符串'log2',max_features=log2(n_features)。

(6).如果是 None，max_feature=n_feature。

random_state:随机数生成器

max_leaf_nodes:

(1).如果为 None，则叶子节点数量不限。

(2).如果不为 None，则 max_depth 被忽略。

2.4.6 模型评估与选择

通过上面的对比，可以看出 4 个模型都欠拟合的状态。接下来将使用交叉验证、网格搜索和随机搜索的方式，选择模型的超参数。

实验目的:

（1）掌握交叉验证算法的应用实践

（2）掌握网络搜索的实现

（3）掌握随机搜索的实现

2.4.6.1 交叉验证

将用交叉验证来搜索 KNN 的模型中 n_neighbors 的最佳参数值。

代码：

```
from sklearn.model_selection import cross_val_score # K 折交叉验证模块

#建立测试参数集
k_range = range(15, 100)

k_scores = []

#藉由迭代的方式来计算不同参数对模型的影响，并返回交叉验证后的平均准确率
for k in k_range:
    knn = KNeighborsRegressor(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10)
    k_scores.append(scores.mean())

#可视化数据
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validation_score')
plt.show()
```

输出：

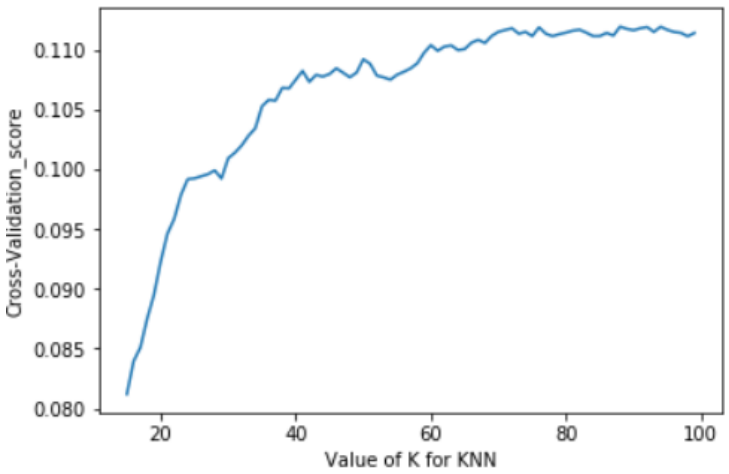


图2-2 不同 K 值对模型的影响

根据上图结果，可知 n_neighbors 的数值在 62 左右最佳，大于此值后模型的表现没有明显提升。

2.4.6.2 参数搜索

本小节分别用将用网格搜索和随机搜索的方式，对决策树分类器和 SVM 回归模型的超参数进行搜索。

代码：

```
###决策树分类器
from sklearn.model_selection import GridSearchCV

params =
[{'criterion':['gini'],'max_depth':[30,50,60,100],'min_samples_leaf':[2,3,5,10],'min_impurity_decrease':[0
.1,0.2,0.5]},
            {'criterion':['gini','entropy']},
            {'max_depth': [30,60,100], 'min_impurity_decrease':[0.1,0.2,0.5]}]



best_model = GridSearchCV(dtree, param_grid=params,cv = 5,scoring ="accuracy")
best_model.fit(X_train,y_train_int)
print('最优分类器:',best_model.best_params_,'最优分数:', best_model.best_score_)  # 得到最优的参数
和分值
```

输出：

```
最优分类器: {'criterion': 'gini', 'max_depth': 30, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2} 最
优分数: 0.781982421875
```

接下来用随机搜索搜索 SVM 回归模型的参数。

代码：

```
from sklearn.model_selection import RandomizedSearchCV


params_svr = {'kernel': ['rbf'], 'C': np.logspace(-3, 2, 6), 'gamma':np.arange(0,10,2)}
best_svr_model = RandomizedSearchCV(svr, param_distributions=params_svr,cv = 3,scoring
="neg_mean_squared_error")
best_svr_model.fit(X,Y)
print('最优分类器:',best_svr_model.best_params_,'最优分数:', best_svr_model.best_score_)  # 得到最优
的参数和分值
```

输出：

```
最优分类器: {'kernel': 'rbf', 'gamma': 4, 'C': 1.0} 最优分数: -0.24169698734960568
```

2.5 实验小结

本章通过代码实践，帮助学习者了解了机器学习算法实践应用的流程，并使用处理过的 APP 评分数据进行回归和分类建模，最后通过交叉验证、网络搜素和随机搜索等算法对模型进行超参数寻优。

2.6 参考答案

代码：

```
y_train_int = y_train.astype(int)
y_test_int = y_test.astype(int)
y_train_int.head()
```

输出：

```
2077    4
4387    3
8974    4
8189    4
6541    4
Name: Rating, dtype: int32
```

代码：

```
from sklearn.tree import DecisionTreeClassifier
初始化决策树模型
dtree=DecisionTreeClassifier(max_depth=8, min_samples_leaf=5, random_state=42)
#训练
dtree.fit(X_train,y_train_int)
#训练集上的 MSE
dtree_pred_train = dtree.predict(X_train)
dtree_mse_train = dtree.score(X_train,y_train_int)
#输出测试集上的测试结果
dtree_pred_test=dtree.predict(X_test)
dtree_mse_test =dtree.score(X_test,y_test_int)
print("训练集 MSE：", dtree_mse_train)
print("测试集 MSE：", dtree_mse_test)
```

输出：

```
训练集 MSE：  0.793701171875
测试集 MSE：  0.7890625
```

3 银行存款预测

3.1 实验介绍

分类与回归是监督学习中两种典型任务。银行中有一种常见业务，即根据客户特征（年龄、工作类型、婚姻状况、文化程度、是否有房贷和是否有个人贷款）预测客户是否会办理存款业务，也是一种分类任务。本实验主要展示如何使用自动学习预测分析构建一个银行存款预测应用的具体步骤，以实现熟悉 ModelArts 平台及其自动学习服务、分类任务的目的。

3.2 实验环境要求

ModelArts 平台自动学习服务

3.3 实验总体设计

3.4 实验过程

3.4.1 上传数据集至 OBS

上传银行数据集文件至 OBS（OBS 操作指导参考：https://support.huaweicloud.com/qs-obs/obs_qs_0001.html），本案例上传至 OBS 路径“/ai-course-001/automl/bank/train.csv”，数据集文件位于./data/目录下。

表3-1 银行客户数据字段说明

字段名	含义	类型	描述
attr_1	年龄	Int	表征客户的年龄
attr_2	职业	String	表征客户所从事的职业
attr_3	婚姻情况	String	表征客户是否结婚或已离异
attr_4	教育情况	String	表征客户受教育的程度
attr_5	房产情况	String	表征客户名下是否有房产
attr_6	贷款情况	String	表征客户名下是否有贷款
attr_7	存款情况	String	表征客户名下是否有存款

3.4.2 创建预测分析项目

步骤 1　进入“ModelArts”管理控制台界面。单击左侧导航栏的“自动学习”，进入“自动学习”界面。

图3-1 自动学习界面

步骤 2　点击“预测分析”创建项目按钮，创建自动学习>预测分析项目，自定义项目名称（本例输入 exeML-bank-predict），训练数据选择 OBS 路径“/ai-course-001/automl/bank/train.csv”（之前数据集文件上传的 OBS 路径），点击“创建项目”完成预测分析项目创建。

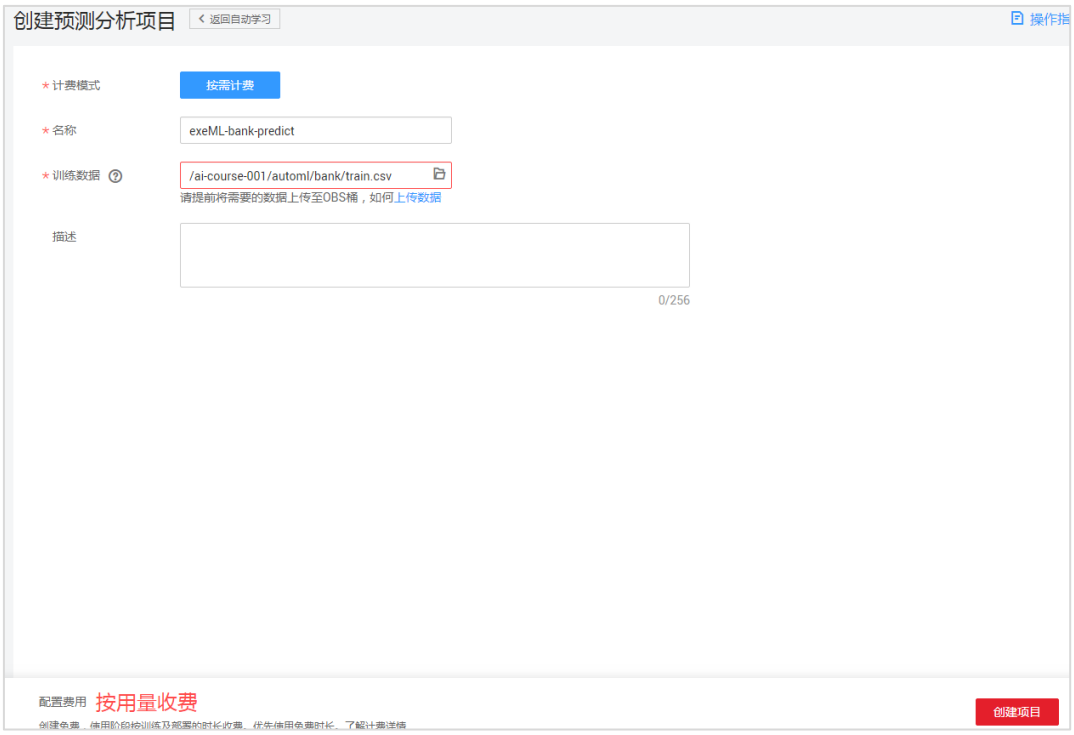


图3-2 创建项目

3.4.3 数据标注并训练

页面会自动跳转到数据标注界面，可以看到样本数据。标签列选择“attr_7”（是否会办理存款），标签列数据类型选择“离散值”，点击“训练”按钮，开始训练。

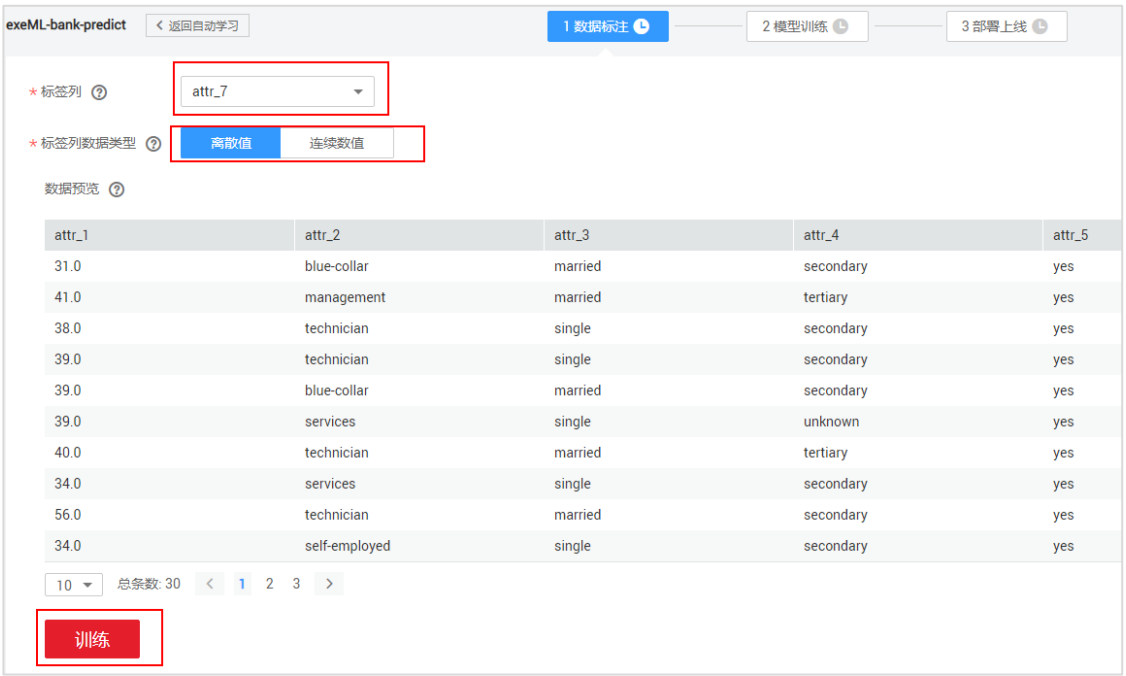


图3-3 数据标注

3.4.4 模型部署

步骤 1　在“模型训练”页面等待训练完成（预计 5 分钟），训练完成后，可以查看模型的精度：

图3-4 查看模型精度

步骤 2　点击“部署”按钮，将模型部署为一个在线服务：

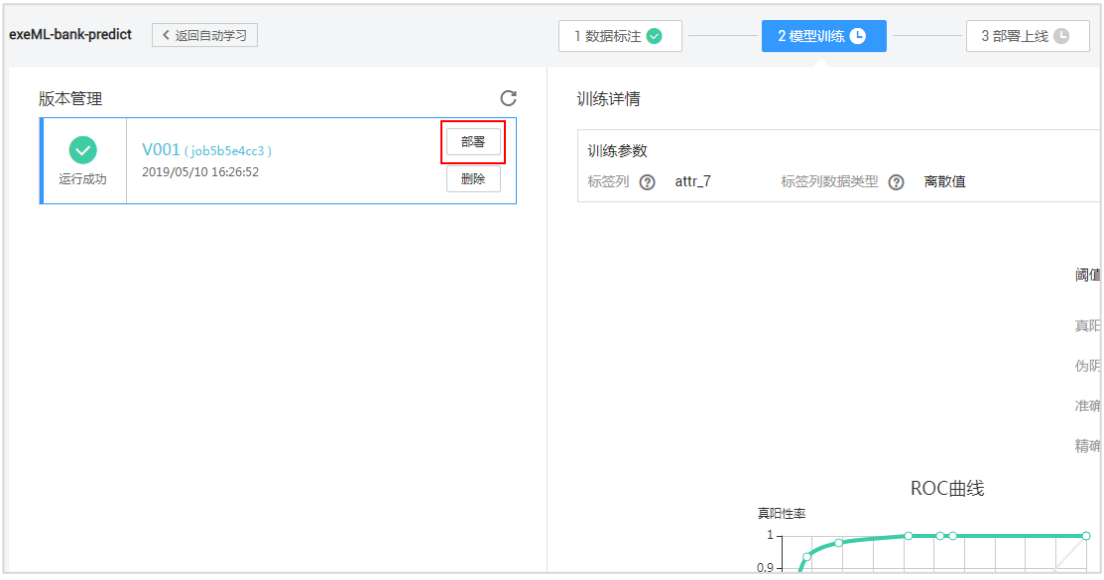


图3-5 部署模型

3.4.5 服务测试

步骤 1　在“部署上线”页面，等待服务部署成功。添加一条银行客户信息，然后点击“预测”按钮。如下图所示：

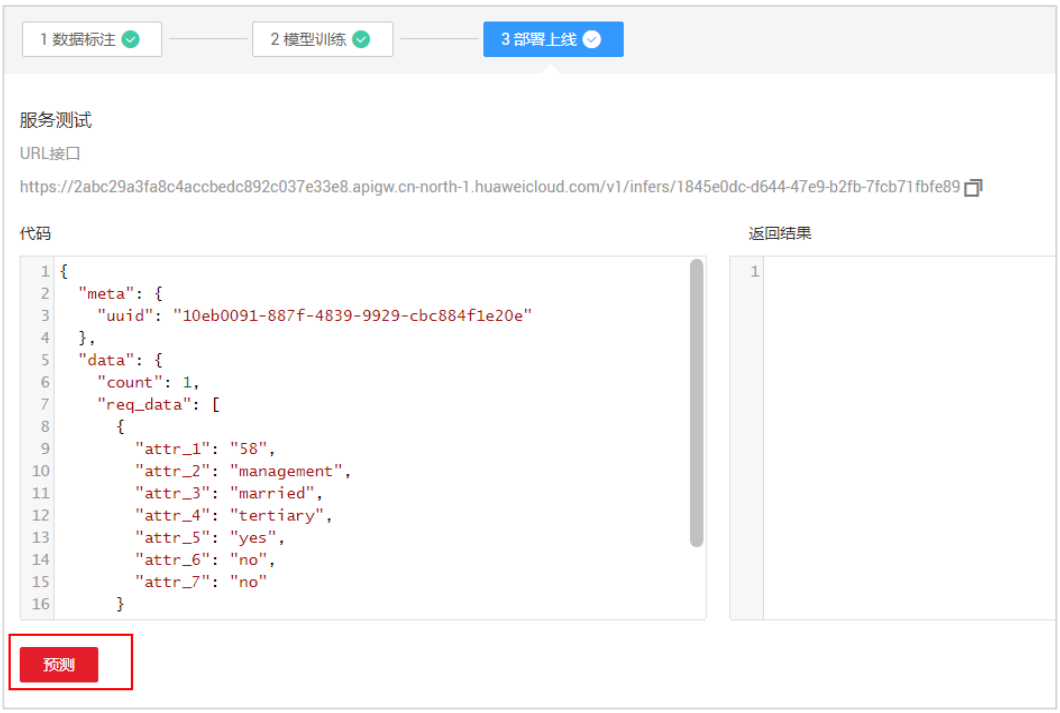
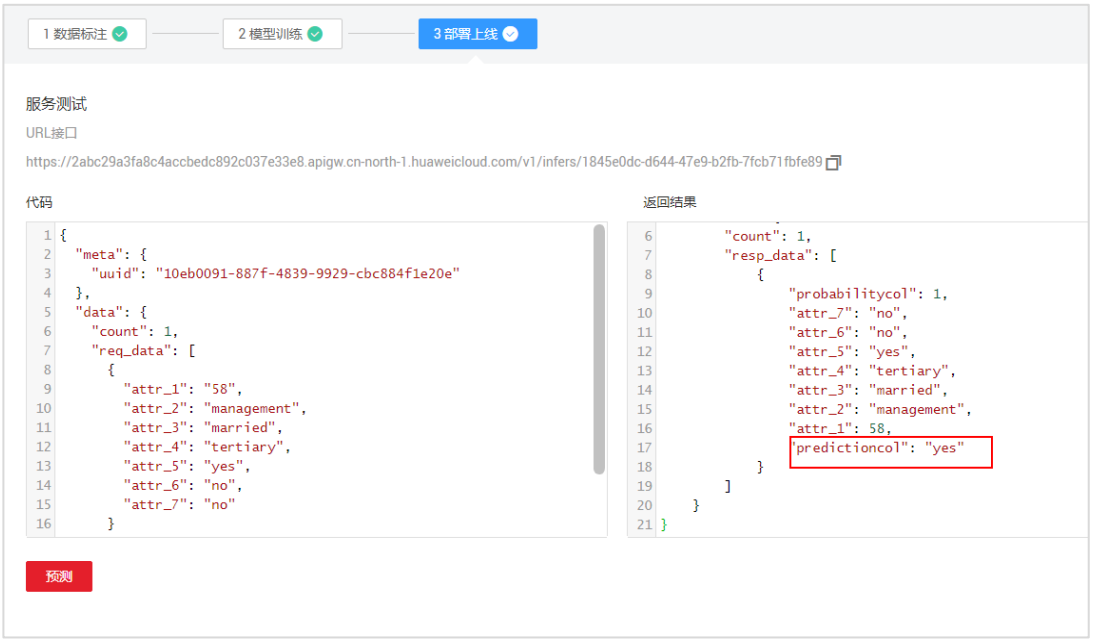


图3-6 输入预测代码

银行客户信息：

```
{
"meta": {
"uuid": "10eb0091-887f-4839-9929-cbc884f1e20e"
},
"data": {
"count": 1,
"req_data": [
{
"attr_1": "58",
"attr_2": "management",
"attr_3": "married",
"attr_4": "tertiary",
"attr_5": "yes",
"attr_6": "no",
"attr_7": "no"
}
]
}
}
```

步骤 2　可以看到预测结果，预测这位银行用户会办理银行存款：



至此，银行存款预测应用实验完成。

3.5 实验小结

本实验基于 ModelArts 开发平台自动学习服务，使用银行客户是否存款数据，实现客户是否存款预测。

3.6 创新设计

使用 ModelArts 自动学习服务，实现动物声音分类。参考如下链接：
https://gitee.com/ModelArts/ModelArts-Lab/blob/master/ExeML/ExeML_Sound_Classification/README.md?_from=gitee_search

4 糖尿病预测

4.1 实验说明

该数据集(pima-indians-diabetes.data)是来自美国疾病控制预防中心的数据，背景是记录美国的糖尿病症状信息，现在美国 1/7 的成年人患有糖尿病。但是到 2050 年，这个比例将会快速增长至高达 1/3。可以利用从 UCI 机器学习数据库里一个关于印第安人糖尿病数据集，通过数据挖掘相关算法来预测糖尿病，该问题本质上是一个二元分类问题。

4.2 实验建模流程要求

基于 Diabete 数据，使用逻辑回归进行糖尿病预测。

4.2.1 环境要求

Python 3.7

4.2.2 实验实现步骤要求

4.2.2.1 相关模块导入

步骤 1 要求导入相关数据读取、处理、分析、可视化，算法模块等

4.2.2.2 数据导入与初步探索

步骤 1 要求载入本地数据集(pima-indians-diabetes.data)，以 dataframe 形式存放后，命名为 df

步骤 2 查看数据尺寸、打印信息，判断特征的类型（名称性、数值型），目标变量分布以及查看是否均衡

步骤 3 对 df 特征进行相关性可视化

步骤 4 对 df 每个特征的分布进行可视化查看

步骤 5 对输入特征进行降维，选择 PCA，并按提示补充如下代码中划线部分内容。

```
### 对输入特征进行降维处理
```

```
from sklearn.decomposition import PCA
from sklearn import preprocessing                    #调用标准化模块
                                                     #降维训练前需要对数据标准化
pca = PCA(              )                            #  保留 99%信息的主成分个主成分
X_pca =pca.fit(X_std).transform(X_std)
```

步骤 6 结合相关性分析和降维后的结论，选择数据进行拆分为训练集和测试集，拆分比例设置为 0.1，指定以 Target 的比例做分层抽样。部分提示如下：

```
from collections import Counter
from sklearn.model_selection import train_test_split
```

步骤 7 选择逻辑回归算法对拆分后的数据进行建模训练和预测，其中要求将原始模型做 5 折交叉验证，评估指标选择 f1。

部分提示如下：

```
#引入逻辑回归和交叉验证的库
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

#引入评价指标的库
from sklearn.metrics import f1_score
```

步骤 8 对逻辑回归的几个重要参数进行网格搜索，网格设置参考如下：

```
c_range=[0.001,0.01,0.1,1.0]
solvers = ['liblinear','lbfgs','newton-cg','sag']
max_iters=[80,100,150,200,300]
tuned_parameters= dict(solver=solvers, C=c_range,max_iter=max_iters)
```

步骤 9 根据搜索参数，最后确认模型，进行预测。

```

```

4.3 参考答案

参考：逻辑回归-糖尿病预测.ipynb

5 信用违约预测

5.1 实验说明

风险管控已经成为了今年金融市场的重要主题之一，银行作为贷方，随时都面临着借贷者违约的风险。传统的专家规则在金融科技时代逐渐过时，机器学习和金融业务的交叉也延伸到信贷领域。违约预测就是其中一重要应用。本实验基于信贷业务场景中一个月内的抽样数据，数据集有 34 个维度，Target 表示客户在接下来一个月是否有违约。模型生成后可使用当前月的数据预测接下来一个月客户是否会违约。

本地离线数据地址和名称：dataset-credit-default.csv

5.2 实验建模流程要求

违约预测只有违约和没有预约两种结果，属于二分类问题。针对二分类问题，可使用的算法有逻辑回归、朴素贝叶斯、支持向量机、树模型等。考虑到实验的完整性和实用性，本实验选用业界常用的逻辑回归和随机森林来做对比。考虑到样本极度不均衡，模型评价选用综合指标 f1_score。

5.2.1 环境要求

Python 3.7

5.2.2 实验实现步骤要求

步骤 1 要求导入相关数据读取、处理、分析、可视化，算法模块等

步骤 2 数据读取，数据框类型，命名为 df

步骤 3 查看 Target 的分布 ,是否违约（1 是，0 否）

步骤 4 可视化观察特征相关性

步骤 5 存储相关性过高的特征对,对于相关性过高的的特征，删除其中一个（根据工程经验，以 0.8 为界）

```
# 选择出符合内容的单元格对应的行、列标签
cols_pair_to_drop = []
for index_ in corr_matrix.index:
    for col_ in corr_matrix.columns:
        if corr_matrix.loc[index_,col_] >= 0.8 and index_!=col_ and (col_,index_) not in
cols_pair_to_drop:
            cols_pair_to_drop.append((index_,col_))
#丢弃特征对中的一个
cols_to_drop = np.unique([col[1] for col in cols_pair_to_drop])  #对于一维数组或者列表，unique 函数去除其中重复的元素，并按元素由大到小返回一个新的无元素重复的元组或者列表
df.drop(cols_to_drop,axis=1,inplace=True)
df.head()
```

步骤 6 打印出缺失率最高的前 15 个特征以及对应的缺失率

步骤 7 可视化：针对 Couple_Year_Income 和 Couple_L12_Month_Pay_Amount 制作箱型图来判定下如何填充。

步骤 8 选择 IOR 方法筛选 Couple_Year_Income 异常值

```
item = 'Couple_Year_Income'
iqr = df[item].quantile(0.75) - df[item].quantile(0.25)
q_abnormal_L = df[item] < df[item].quantile(0.25) - 1.5 * iqr
q_abnormal_U = df[item] > df[item].quantile(0.75) + 1.5 * iqr
#取异常点的索引
print(item + '中有' + str(q_abnormal_L.sum() + q_abnormal_U.sum()) + '个异常值')
item_outlier_index = df[q_abnormal_L|q_abnormal_U].index
```

步骤 9 根据筛选出的异常值索引，删除 Couple_Year_Income 的异常值并用中位数填补缺失值。

步骤 10 选择 IOR 方法筛选 Couple_L12_Month_Pay_Amount 异常值

步骤 11 根据筛选出的异常值索引，删除 Couple_L12_Month_Pay_Amount 的异常值并用中位数填补缺失值

步骤 12 查看 df 中仍有少量缺失值的特征是哪些，提取出列名，封装到列表结构中，列表命名为 null_col

步骤 13 使用众数填充 null_col 中每列缺失值，直接改变 df。

步骤 14 从 df 中删除无分类意义的特征列 Cust_No。

步骤 15 使用 factorize 函数，对 df 剩余全部名称型特征进行标签编码，部分提示如下：

```
### 查看数据集剩余的名称性特征
con_col=[]
for col in df.columns:
    if df.dtypes[col] == np.object:
        con_col.append(col)
con_col
```

步骤 16 将数据集作 9:1 的切分成训练集和测试集，并查看两个集中不同两类别的数量。

步骤 17 引入 StandardScaler 标准化工具库，对训练集和测试集做标准化

步骤 18 使用 Logistic Regression 对标准化后的数据进行建模

步骤 19 对训练集 X_train_std，y_train 进行过采样，指定过采样比例，此处为 2:1。

```
from imblearn import over_sampling
print('Original dataset shape {}'.format(Counter(y_train)))
#ratio 指定过采样比例，此处为 2:1
smote_model = over_sampling.SMOTE(random_state=7, ratio=0.5)
X_train_res,y_train_res = smote_model.fit_sample(X_train_std,y_train)
print('Resampled dataset shape {}'.format(Counter(y_train_res)))
```

步骤 20 使用 Logistic Regression 对过采样且标准化后的数据 X_train_res,y_train_res 进行建模，且使用交叉验证（5 折，默认 scoring）进行查看。

步骤 21 对上一步中的 Logistic Regression 模型，调节常用的 C 和 solver 两个参数，使用网格搜索+交叉验证法

步骤 22 查看逻辑回归模型在测试集上的效果，选择多个评估指标 accuracy_score、precision_score 、recall_score 、f1_score

步骤 23 使用 RandomForest 对标准化后的数据 ***（注意此处不是过采样后的数据）*** 进行建模，设置样本权重参数 class_weight='balanced'，然后对测试集进行预测，并选择多个评估指标 accuracy_score、precision_score 、recall_score 、f1_score 查看预测得分。

步骤 24 参数调优，选择坐标下降方式对 class_weight 进行搜索，部分提示如下:

```
param_test0 = {'class_weight':[{0:1,1:3},{0:1,1:5},{0:1,1:10},{0:1,1:20},'balanced']}
```

```
```

步骤 25 参数调优，选择坐标下降方式对 n_estimators 进行搜索，部分提示如下:

```
param_test1 = {'n_estimators':range(10,101,10)}
```

步骤 26 根据两次参数调整后的随机森林模型重新对测试集进行预测，选择多个评估指标 accuracy_score、precision_score 、recall_score 、f1_score，对预测结果进行评估，并打印出在各个指标上得分

5.3 参考答案

参考：信用违约预测.ipynb