■ 基于 SVM 与随机森林算法

支付宝花呗违约率分析

Haiwang Luo 罗海王

阿里 Python 数据分析第四周产出报告

目　　录

一、实验目的及要求

1.1 任务与具体要求

- 数据集链接：https://www.dropbox.com/s/nn9g44vb2frfx5t/alipay_huabei_GS1.csv?dl=0
    - 分析这里的数据，针对这个数据集构建一个分析违约率的分类器
- 创建多种分类器，综合比较效果
- 掌握 GridSearchCV，优化算法模型的参数
- 使用 Pipeline 管道机制进行流水线作业。因为在做分类之前，我们还需要一些准备过程，比如数据规范化，或者数据降维等。
- 问题的定性：二分类问题，根据用户的行为及特征属性，来推断用户是否有违约的可能。

1.2 任务简要分解

- 做分类的时候往往都是有步骤的，比如先对数据进行规范化处理，你也可以用 PCA 方法（一种常用的降维方法）对数据降维，最后使用分类器分类。
- Python 有一种 Pipeline 管道机制。管道机制就是让我们把每一步都按顺序列下来，从而创建 Pipeline 流水线作业。每一步都采用 (‘名称’, 步骤) 的方式来表示。
- 分类问题：分类是一种基于一个或多个自变量确定因变量所属类别的技术。

1.3 数据集特征值与目标值的简要说明

ID 用户 ID
LIMIT_BAL 花呗透支额度
Sex 男 1 女 2
education 研究生 1 本科 2 高中 3 其他 4
marriage 已婚 1 单身 2 其他 3
pay_n 内部划分的不同时期的客户还款情况 不用管时期怎么划分
BILL_AMT 内部划分的不同时期的账单金额
default.payment.next.month 违约 1 守约 0

二、背景知识梳理

2.1 SVM(Support Vector Machine,支持向量机)

2.1.1 SVM 的工作原理

- 分类方法：是在这组分布中找出一个超平面作为决策边界，使模型在数据点的分类误差尽量接近与小，尤其是在未知数据集上的分类误差（泛化误差）尽量小。
- 超平面：在几何中，超平面是空间的一个子空间，它是维度比所在空间小一维的空间。例：三维数据空间的超平面是二维平面；二维数据空间的超平面是一条直线。
- 边际（margin）：决策边界 B2 可以平移的最远的 0 误差边界 b11, b12，且 B2 到 b11, b12 的距离相等，则 b11, b12 之间的距离叫 B2 这条决策边界的边际，记为 d，且 d = 2 / |||w||
- 支持向量机就是通过找出边际最大的决策边界，来对数据进行分类的分类器。因此 SVM 分类器也叫做最大边际分类器。

2.1.2 SVM 参数更新的推导原理（数学推导详见第三周 SVM 参数推导.pdf）

- 第一步：构建寻找使边界间隔最大的超平面：设置二分类标签为 1 或 -1
- 第二步：先把 svm 的决策问题变成关于参数 w, b 的最优化问题。
- 第三步：把损失函数问题转为拉格朗日函数 L(w,b alpha)的形态，此时要引入拉格朗日参数 alpha。
    - 此时就把目标函数转变成 以 alpha 为参数时 L(w,b alpha)的最大值，再以 w,b 为参数时 L(w,b alpha)的最小值问题。
    - 但是在对 w,b 求偏导数的时候，偏导数结果还带有未知参数 alpha，因此无法直接求解 w,b
- 第四步：在满足 KKT 的四个条件下，强对偶关系，用拉格朗日对偶公式，
    - 把拉格朗日函数转为拉格朗日对偶函数：min(w,b) max(alpha >= 0)L(w,b alpha) = max(alpha >= 0)min(w,b)L(w,b alpha)
    - 而在求 L(w,b alpha)的最小值时：对 w,b 求偏导数为 0 替换带入，此时 L(w,b alpha)可以变成只含有唯一参数 alpha，而不含有 w,b 参数的式子
    - 最终转化为一个 关于 参数 alpha，求最大值的目标函数问题，从而可以解出 alpha，再带回原来求偏导数的式子，可以求出 w,b
- 第五步：根据求得的 w,b 参数，即可得到在测试集上的决策函数 f(x_test) = sign(wT * x_test + b) , 这里给的是一个二分类问题，所以 f(x_test) 函数的取值是 1 或 -1

2.1.3 SVM 算法的优缺点

- 优点：它工作的效果很明显，有很好的分类作用。
    - 它在高维空间中同样是有效的。它在尺寸数量大于样本数量的情况下，也是有效的。
    - 它在决策函数（称为支持向量）中使用训练点的子集，因此它的内存也是有效的
- 缺点：拥有大量的数据集时，它表现并不好，因为它所需要的训练时间更长。
    - 当数据集具有很多噪声，也就是目标类重叠时，它的表现性能也不是很好。
    - SVM 不直接提供概率估计，这些是使用昂贵的五重交叉验证来计算的。它是 Python scikit-learn 库的相关 SVC 方法。

2.2 DT (Decision Tree, 决策树分类算法)

2.2.1 决策树算法的工作原理

- 决策树以树状结构构建分类或回归模型。它通过将数据集不断拆分为更小的子集来使决策树不断生长。最终长成具有决策节点（包括根节点和内部节点）和叶节点的树。最初决策树算法它采用采用 Iterative Dichotomiser 3（ID3）算法来确定分裂节点的顺序。
- 决策树的定义可以为： 决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。
- 信息熵和信息增益用于被用来构建决策树。
    - 信息熵是衡量元素无序状态程度的一个指标，即衡量信息的不纯度。直观上说地理解，信息熵表示一个事件的确定性程度。
    - 信息熵度量样本的同一性，如果样本全部属于同一类，则信息熵为 0；如果样本等分成不同的类别，则信息熵为 1。
    - 信息增益测量独立属性间信息熵的变化。它试图估计每个属性本身包含的信息，构造决策树就是要找到具有最高信息增益的属性(即纯度最高的分支)。
    - 采用信息熵进行节点选择时，通过对该节点各个属性信息增益进行排序，选择具有最高信息增益的属性作为划分节点，过滤掉其他属性。
- 决策树模型存在的一个问题是容易过拟合。因为在其决策树构建过程中试图通过生成长一棵完整的树来拟合训练集，因此却降低了测试集的准确性。
    - 通过剪枝技术可以减少小决策树的过拟合问题。

2.3 集成学习和基于决策树算法的随机森林 (Random Forest) 算法

2.3.1 集成学习的概念

- 通过建立几个模型组合来解决单一预测问题，工作原理是生成多个分类器/模型，各自独立地学习和做出预测，
- 这些预测最后结合成组合预测，因此优于任何一个单分类的做出预测。

2.3.2 随机森林定义及工作原理

- 训练集：特征值 目标值
- 随机：两个随机：训练集随机：bootstrap 随机有放回的抽样 ：N 个样本中随机有放回的抽样 N 个
- 特征值随机：M 个特征中随机有放回的抽样 m 个特征 （$M >> m$），这样可以起到特征降维的效果
- 森林：包含多个决策树的分类器，随机森林最后的分类取决于多棵树的投票表决，从而消除了单棵树的偏差。

2.3.3 随机森林的好处

- 集成的基本思想是算法的组合提升了最终的结果。
- 随机森林在决策树生增长的同时为模型增加了额外的随机性。它在分割节点时，不是搜索全部样本最重要的特征，而是在随机特征子集中搜索最佳特征。
- 这种方式使得决策树具有多样性，从而能够得到更好的模型。

三、数据分析流程分析

3.1 导入数据与划分数据集

3.1.1 导入数据：使用 pandas.read_csv('https://.....', header='infer', index_col=None)
3.1.2 随机抽样：DataFrame.sample(n=None, frac=None, replace=False, weights=None, random_state=None, axis=None)[source]

n=3：提取 3 行数据列表
frac=0.8： 抽取其中 80%
random_state=None：取得数据不重复
random_state=1：可以取得重复数据
axis：选择抽取数据的行还是列
axis=0: 在行中随机抽取 n 行
axis=1: 在列中随机抽取 n 列

3.1.3 划分数据集：第一列 id 列给去掉，然后最后一列 default.payment.next.month 列 作为目标值，其他列作为特征值

```
data=pd.read_excel("../data/yanben.xls");
print(data.head())
#iloc 只能用数字索引，不能用索引名
#print(data.iloc[:,0:4])
##loc 只能通过 index 和 columns 来取，不能用数字
#print(data.loc[0:1,["序号","颜色","形状","重量"]])
#print(data['类别'])
x_data=data.iloc[:,0:4];
x_target=data['类别'];
```

3.2 特征工程、数据预处理 - 数据规范化与数据降维

3.2.1 数据规范化：

- 归一化: sklearn.preprocessing.MinMaxScaler 通过对原始数据进行变换把数据映射到(默认为[0,1])之间.
  - 什么时候进行归一化？归一化的作用？:当三个特征同等重要的时候，进行归一化, 归一化使得某一个特征对最终结果不会造成更大的影响。
  - 最大值与最小值非常容易受异常点影响，所以这种方法鲁棒性较差，只适合传统精确小数据场景。
- 标准化: 通过对原始数据进行变换把数据变换到均值为 0,方差为 1 范围内。在已有样本足够多的情况下比较稳定，适合现代嘈杂大数据场景。

3.2.2 数据降维 - 主成分分析 PCA：把高维数据转化为低维数据 -> 压缩数据，降低复杂度，损失少量信息

- 2.1 PCA 降维做的事情：找到一个合适的直线，通过一个矩阵运算得出主成分分析的结果
- 2.2 API： sklearn.decomposition.PCA(n_components=None)
  - n_components: 小数代表保留多少百分比的信息；整数代表减少到多少特征属性
  - PCA.fit_transform(X): numpy array 格式的数据 [n_samples, n_features]

3.3 Pipeline 和 GridSearchCV 结合使用 -> 优化算法模型的参数、操作流程自动化

3.4.1 pipeline 简单使用代码示例：

```
from sklearn.pipeline import Pipeline
estimators = [
   Pipeline([('sc', StandardScaler()),('pca',PCA(n_components=2)),
('knn',KNeighborsClassifier(n_neighbors=5))]),
   Pipeline([('ss', StandardScaler()),('svc', SVC())]),
   Pipeline([('ss', StandardScaler()),('svc', SVC())])
]
```

3.4.2 GridSearchCV 的应用示例：

GridSearchCV 是用交叉验证选出最优参数：

其中第一个参数是 pipeline，

第二个参数 param_grid 是关于参数多个选择的字典。

第三个参数 cv 是交叉验证的折数（例如 5 折交叉验证(5-fold cross validation)，

将数据集分成 5 份，轮流将其中 4 份做训练 1 份做验证，5 次的结果的均值作为对算法精度的估计）

```
# 先设置参数字典： 字典中的 key 是属性的名称，value 是可选的参数列表，需要注意下划线是两个
parameters_knn = {'n_neighbors': [i for i in range(1, 11)],
          'weights': ['uniform', 'distance'],
          'leaf_size': [20, 30, 40, 50]
          }
# 获取模型并设置参数
   # GridSearchCV: 进行交叉验证，选择出最优的参数值出来
   # 第一个输入参数：进行参数选择的模型，
   # param_grid： 用于进行模型选择的参数字段，要求是字典类型；cv: 进行几折交叉验证
   estimator_knn = GridSearchCV(estimators[0], param_grid=parameters_knn, cv=5, n_jobs=1) # 选择第一个模型, 并传入对应的参数
   # 模型训练-网格搜索
   estimator_knn.fit(x_train, y_train)
```

3.4 分类器的性能评价原理

3.5.1 混淆矩阵：混淆矩阵是一张表，这张表通过对比已知分类结果的测试数据的预测值和真实值表来描述衡量分类器的性能。

在二分类的情况下，混淆矩阵是展示预测值和真实值四种不同结果组合的表。

	猜中	猜错
正例	TP	FP
反例	TN	FN

FN：被错误的分为 负类 的数据，（即 真实为 正，预测为 负）。

TP：被正确的分为 正类 的数据，（即 真实为 正，预测也为 正）。

1.1 假正例&假负例：假正例和假负例用来衡量模型预测的分类效果。假正例是指模型错误地将负例预测为正例。假负例是指模型错误地将正例预测为负例。

主对角线的值越大（主对角线为真正例和真负例），模型就越好；副对角线给出模型的最差预测结果。

3.5.2 准确率是模型预测正确的部分。Accuracy = (TP + TN) / (TP + TN + FP + FN)

当数据集不平衡，也就是正样本和负样本的数量存在显著差异时，单独依靠准确率不能评价模型的性能。精度和召回率是衡量不平衡数据集的更好的指标。

精度是指在所有预测为正例的分类中，预测正确的程度为正例的效果。精度越高越好。精度 = TP / (TP + FP)

召回率是指在所有预测为正例（被正确预测为真的和没被正确预测但为真的）的分类样本中，召回率是指预测正确的程度。它，也被称为敏感度或真正率（TPR）。

召回率 = TP / (TP + FN)

3.5.3 F-1 值：通常实用的做法是将精度和召回率合成一个指标 F-1 值更好用，特别是当你需要一种简单的方法来衡量两个分类器性能时。

F-1 值是精度和召回率的调和平均值。F1 = 2 * 精度 * 召回率 /（精度 + 召回率）

普通的通常均值将所有的值平等对待，而调和平均值给予较低的值更高的权重，从而能够更多地惩罚极端值。

所以，如果精度和召回率都很高，则分类器将得到很高的 F-1 值。

3.5.4 接受者操作曲线（ROC）和曲线下的面积（AUC）

ROC 曲线是衡量分类器性能的一个很重要指标，它代表模型准确预测的程度。

ROC 曲线通过绘制真正率 TP 和假正率 FP 的关系来衡量分类器的敏感度。

如果分类器性能优越，则真正率将增加，曲线下的面积会接近于 1.如果分类器类似于随机猜测，真正率将随假正率线性增加。AUC 值越大，模型效果越好。

3.5.5 累积精度曲线 CAP 代表一个模型沿 y 轴为真正率的累积百分比与沿 x 轴的该分类样本累积百分比。

CAP 不同于接受者操作曲线（ROC，绘制的是真正率与假正率的关系）。与 ROC 曲线相比，CAP 曲线很少使用。

四、实验伪代码及功能描述

4.1 数据采集与预处理

4.1.1 读取数据

```
import pandas as pd

local_path =
pd.read_csv("https://www.dropbox.com/s/nn9g44vb2frfx5t/alipay_huabei_GS1.
csv?dl=0")
```

- 在线读取文件内容可能加载过慢，可以选择下载到本地使用

```
local_path =
pd.read_csv("/Users/haiwangluo/Desktop/alipay_huabei_GS1.csv")
```

```
# 查看一下 文件内容的前几行信息
print(local_path.head())

# 文件过大，进行随机抽样，frac 确定抽取的比例， random_state 以确保可重复性的例子。
local_path = pd.DataFrame.sample(local_path, frac=0.3, random_state=1)

# 另一种对文件处理对方式：选择前 1000 行数据
# local_path = local_path[0: 1000]
```

4.1.2 筛选特征值与目标值

```
# 属性特征值的列范围为，第二列到倒数第二列
x_data = local_path.iloc[:, 1: -1]
# print(x_data.head())

# 目标值为最后一列
x_target = local_path['default.payment.next.month']
# print(x_target.head())
```

4.1.3 数据集划分，要注意划分的之后的变量的接收顺序

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_data, x_target,
test_size=0.2)
```

4.2 使用 Pipeline 管道机制，流水线化构建算法模型

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier


# StandardScaler() -> 标准化: 通过对原始数据进行变换把数据变换到均值为0,方差为1 范围内.
# PCA() -> 主成分分析PCA：把高维数据转化为低维数据 -> 压缩数据，降低复杂度，损失少量信息

estimator_models = [
    Pipeline(steps=[('ss', StandardScaler()),
                    ('pca', PCA()),
                    ('svc', SVC())
                    ]),
    Pipeline(steps=[('ss', StandardScaler()),
                    ('pca', PCA()),
                    ('rfc', RandomForestClassifier())
                    ])
]
```

4.3 SVM 算法模型

4.3.1 SVM 模型设置参数

```
# n_components: 小数代表保留多少百分比的信息；整数代表减少到多少特征属性
# C: 默认值为1:错误分类样本的惩罚因子，C 越大，惩罚力度越大，倾向于选择复杂的模型，减少错分样本，C 越小，惩罚力度越小，倾向于选择简答的模型
# gamma: 核函数的系数('Poly', 'RBF' and 'Sigmoid'), 默认是gamma = 1 / n_features ,n_features 为样本原始特征数量
# kernel: 默认选择径向基核函数'rbf ,也可选择'linear','poly','sigmod'核函数
parameters_svm = [{'pca__n_components': [0.40, 0.64, 0.85, 0.95],
                   'svc__C': [0.01, 0.1, 1, 10],
                   'svc__gamma': [0.001, 0.1],
                   'svc__kernel': ['rbf']
                   },
                  {'pca__n_components': [0.40, 0.64, 0.85, 0.95],
                   'svc__C': [0.01, 0.1, 1, 10],
                   'svc__kernel': ['linear']
                   }]
```

4.3.2 SVM 模型评估与模型预测

```
from sklearn.model_selection import GridSearchCV
import datetime
from sklearn.metrics import confusion_matrix, classification_report

# 调用 datetime 函数，-> 获取算法运行时间
ts_svm = datetime.datetime.now()
# 使用 GridSearchCV 调参数， 选择到对应的算法模型在 estimator_models 列表中的位置，调取参数列表
# CV -> 进行几折验证
estimator_svm = GridSearchCV(estimator_models[0],
param_grid=parameters_svm, cv=3)
# 对训练集的特征值与目标值进行计算
estimator_svm.fit(x_train, y_train)
# 用训练好的模型对测试集的特征值进行预测，得出预测的目标结果
y_predict_svm = estimator_svm.predict(x_test)
tp_svm = datetime.datetime.now() - ts_svm

print('SVM 模型的算法的训练时间为:%s\n 预测准确率为: %.5f\n 最好的参数组合为: %s' %
(tp_svm, estimator_svm.best_score_, estimator_svm.best_params_))
# 查看算法的表现结果，以 精确率、召回率、F1_score 来进行综合判断
print('SVM 模型的算法产出结果：\n\t%s' % classification_report(y_test,
y_predict_svm, target_names=['违约 0', '守约 1']))
# 查看 混淆矩阵，做辅助判断
print(confusion_matrix(y_test, y_predict_svm))
```

4.3.3 SVM 模型的实际算法结果表现

```
SVM模型的算法的训练时间为:0:09:57.462252
预测准确率为: 0.80861
最好的参数组合为: {'pca__n_components': 0.95, 'svc__C': 1, 'svc__gamma': 0.1, 'svc__kernel': 'rbf'}
SVM模型的算法产出结果:
                  precision    recall  f1-score   support

        违约 0       0.84      0.94      0.89      1421
        守约 1       0.61      0.32      0.42       379

    accuracy                           0.81      1800
   macro avg       0.72      0.63      0.66      1800
weighted avg       0.79      0.81      0.79      1800

[[1341   80]
 [ 256  123]]
```

4.4 随机森林算法模型

4.4.1 随机森林模型设置参数

```
# n_estimators：这是森林中树木的数量，即基评估器的数量。n_estimators 越大，模型的效果往往越好。
# criterion： 即CART 树做划分时对特征的评价标准。分类模型和回归模型的损失函数是不一样的。分类RF 对应的CART 分类树默认是基尼系数gini,另一个可选择的标准是信息增益。
# max_depth：决策树最大深度。值越大，决策树越复杂，越容易过拟合。
# max_features：决策树划分时考虑的最大特征数。默认是"None",意味着划分时考虑所有的特征数；如果是"log2"意味着划分时最多考虑log2(n_features)个特征。

parameters_rfc = {'pca__n_components': [0.40, 0.64, 0.85, 0.95],
                  'rfc__n_estimators': [10, 100, 500],
                  'rfc__criterion': ['gini', 'entropy'],
                  'rfc__max_depth': [None, 5, 8, 15, 25, 30],
                  'rfc__max_features': ['sqrt', 'log2', None]
                  }
```

4.4.2 随机森林模型评估与模型预测

```
ts_rfc = datetime.datetime.now()
estimator_rfc = GridSearchCV(estimator_models[1],
param_grid=parameters_rfc, cv=3)
estimator_rfc.fit(x_train, y_train)

y_predict_rfc = estimator_rfc.predict(x_test)
tp_rfc = datetime.datetime.now() - ts_rfc

print('\n 随机森林模型的算法的训练时间为:%s\n 预测准确率为: %.5f\n 最好的参数组合
为: %s' % (tp_rfc, estimator_rfc.best_score_, estimator_rfc.best_params_))

print('随机森林模型的算法产出结果：\n\t%s' % classification_report(y_test,
y_predict_rfc, target_names=['违约 0', '守约 1']))

print(confusion_matrix(y_test, y_predict_rfc))
```

4.4.3 随机森林模型的实际算法结果表现

```
随机森林模型的算法的训练时间为:1:27:05.949805
预测准确率为: 0.81000
最好的参数组合为: {'pca__n_components': 0.95, 'rfc__criterion': 'entropy', 'rfc__max_depth': 15, 'rfc__max_features': 'sqrt', 'rfc__n_estimators': 100}
随机森林模型的算法产出结果:
                precision    recall  f1-score   support

       违约 0       0.83      0.94      0.88      1421
       守约 1       0.57      0.28      0.38       379

    accuracy                           0.80      1800
   macro avg       0.70      0.61      0.63      1800
weighted avg       0.78      0.80      0.78      1800

[[1341   80]
 [ 272  107]]
```

五、实验结果及结论

- 随机森林模型比支持向量机更加实用，预测准确率、精确率、召回率、f1_score 都很高，但算法训练相对比较耗时间，特别是在输入大量待调参数的情况下。
- SVM 模型预测精度，精确率、召回率、f1_score 都接近于或者略低于随机森林模型，但算法运行时间很快，作为单一分类器，能达到很接近于集成学习算法下的随机模型，也说明是一个很不错的算法模型。
- 结论：花呗违约率分析，作为一个二分类问题，应该更需要注意算法的预测准确程度，能否分类正确是最重要的，所以在本次实验中随机森林算法模型更好。

六、参考资源

1．sklearn 中 pipeline 的实现,及 GridSearchCV 寻找最优参数
https://blog.csdn.net/qq_34211618/article/details/103685975

2．pandas 系列 read_csv 与 to_csv 方法各参数详解
https://blog.csdn.net/u010801439/article/details/80033341

3．基于 SVM、Pipeline、GridSearchCV 的鸢尾花分类
https://blog.csdn.net/xiaosa_kun/article/details/84868406

4．数据样本，特征值，目标值，按比例划分
https://blog.csdn.net/u011066470/article/details/104447001

5．pandas 数据处理基础——筛选指定行或者指定列的数据
https://www.cnblogs.com/gangandimami/p/8983323.html

6．sklearn 中的 Pipeline：
https://www.cnblogs.com/wuliytTaotao/p/9329695.html

7．一文读懂机器学习分类算法（附图文详解）
https://blog.csdn.net/Datawhale/article/details/100788726

8．python 字典获取最大值的键的值：
https://www.cnblogs.com/demo-deng/p/9634111.html

9．sklearn 计算准确率、精确率、召回率、F1 score:
https://blog.csdn.net/hfutdog/article/details/88085878

10. SKLearn 中预测准确率函数介绍
https://www.cnblogs.com/hd-chenwei/p/a679d6eee216db0b261e0ebf855545ec.html

11. 机器学习各种算法结果可视化比对
https://blog.csdn.net/keepreder/article/details/47101585

12．sklearn.svm.SVC 的参数学习笔记
https://blog.csdn.net/qq_37007384/article/details/88410998