工艺优化-机器学习-实验总结

郑卓民

已知：

给定了一组数据，25个变量中其中有一个目标变量 y

重点参数为$x4, x5, x6, x13, x16, x17, x18, x20$（共8个参数）

期望$y >= 22$的比例超过90$\%$以上

简要分析与处理：

1. 可提前将数据中$y >= 22$的部分记一变量$y1$为$y1 = 1$，其余为$y1 = 0$，将目标分为了两类，方便进行逻辑回归。
2. 重点在于寻找一个合适的模型，利用已有数据，使得训练出来的决策边界能满足我们需要的准确度（即：利用该决策边界预测出来的结果中，$y >= 22$的空间里（即$y1 = 1$），有90%以上的结果是 真的 达到 $y >= 22$（$y1 = 1$））。

实验过程：

简要记录一些失败的尝试（模型）：

多元线性回归：

```
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
print('Score: {:.2f}'.format(classifier.score(X_test, y_test)))
```

```
输出结果为： Score: 0.75
```

高斯朴素贝叶斯：

```
classifier_gaussiannb=GaussianNB()
classifier_gaussiannb.fit(X_train,y_train)
print('Score: {:.2f}'.format(classifier_gaussiannb.score(X_test, y_test)))
```

```
输出结果为： Score: 0.65
```

随机森林：

```
rf_regressor=RandomForestRegressor(max_depth=11,n_estimators=120)
rf_regressor.fit(X_train,y_train)
print('Score: {:.2f}'.format(rf_regressor.score(X_test, y_test)))
```

```
输出结果为： Score: 0.44
```

上述模型训练效果都不太好，考虑使用多项式回归

多项式回归（效果较好）：

此部分利用sklearn库来实现多项式回归。

此处步骤主要为：利用polynomial features构造系数，进行多项式回归。

关于polynomial features的相关参数如下：

1. degree : The degree of the polynomial features. Default = 2.
2. interaction_only : boolean, default = False, If true, only interaction features are produced: features that are products of at most degreedistinct input features.
3. include_bias : boolean, If True (default), then include a bias column, the feature in which all polynomial powers are zero (i.e. a column of ones - acts as an intercept term in a linear model).

主要考虑degree和interaction_only两个参数。

首次使用$degree = 2, interaction_only = False$的时候，已经比上述模型取得更好的效果：

```
poly_reg = PolynomialFeatures(degree=2)
x_poly = poly_reg.fit_transform(X_train)
logistic = LogisticRegression(solver='liblinear')
logistic.fit(x_poly, y_train)
print('Score : ', logistic.score(x_poly, y_train))
```

```
输出结果为： Score: 0.8061538461538461
```

使用$degree = 3, interaction_only = False$:

```
poly_reg = PolynomialFeatures(degree=3)
x_poly = poly_reg.fit_transform(X_train)
logistic = LogisticRegression(solver='liblinear')
logistic.fit(x_poly, y_train)
print('Score : ', logistic.score(x_poly, y_train))
```

```
输出结果为： Score: 0.8125874125874126
```

使用$degree = 3, interaction_only = True$:

```
poly_reg = PolynomialFeatures(degree=3, interaction_only=True)
x_poly = poly_reg.fit_transform(X_train)
logistic = LogisticRegression(solver='liblinear')
logistic.fit(x_poly, y_train)
print('Score : ', logistic.score(x_poly, y_train))
```

```
输出结果为： Score: 0.9205168363351606
```

完整代码：

训练模型前还对数据进行了一些预处理

```python
# -*- coding: utf-8 -*-
import pandas as pd
from sklearn import preprocessing
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, classification_report
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression

# 读取数据
data = pd.read_csv("data2.csv")
X = data.iloc[:, 2:]
y = data.iloc[:, 0]

# 数据处理
#imp = ['x4','x5','x6','x13','x16','x17','x18','x20']
imp = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9',
    'x10', 'x11', 'x12','x13', 'x14', 'x15', 'x16','x17',
    'x18', 'x19', 'x20', 'x21', 'x22', 'x23', 'x24']
X_train_conti_std = X[imp]

X_train = pd.DataFrame(data=X_train_conti_std, columns=imp)

# 填充缺失项
for column in list(X_train.columns[X_train.isnull().sum() > 0]):
    mean_val = X_train[column].mean()
    X_train[column].fillna(mean_val, inplace=True)

# 标准化
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)

# 多项式回归
poly_reg = PolynomialFeatures(degree=3, interaction_only=True)

# 特征处理
x_train_poly = poly_reg.fit_transform(X_train)

# 定义逻辑回归模型
logistic = LogisticRegression(solver='liblinear')

# 训练模型
logistic.fit(x_train_poly, y)
y_pred = logistic.predict(x_train_poly)

print("--------------------------------------------")
print("多项式系数：")
print(logistic.coef_)
print("--------------------------------------------")
print('score: ', logistic.score(x_train_poly, y))
print('accuracy_score: ', accuracy_score(y, y_pred))
print("--------------------------------------------")
print("混淆矩阵")
cm = confusion_matrix(y, y_pred)
print(cm)
print("--------------------------------------------")
print('precision_score: ', precision_score(y, y_pred))
print("--------------------------------------------")
print('classification_report: ')
print(classification_report(y, y_pred))
print("--------------------------------------------")
```

```
-------------------------------------------
多项式系数：
[[ 0.80888463 -0.36928486  0.02390702 ...  0.08185608 -0.32006064
  -0.21254639]]
-------------------------------------------
score:  0.9205168363351606
accuracy_score:  0.9205168363351606
-------------------------------------------
混淆矩阵
[[2054  280]
 [ 126 2648]]
-------------------------------------------
precision_score:  0.9043715846994536
-------------------------------------------
classification_report:
              precision    recall  f1-score   support

           0       0.94      0.88      0.91      2334
           1       0.90      0.95      0.93      2774

    accuracy                           0.92      5108
   macro avg       0.92      0.92      0.92      5108
weighted avg       0.92      0.92      0.92      5108

-------------------------------------------
```