实验 4-1 利用决策树算法构建价格预测模型

建议课时：40 分钟

一、 实验目的

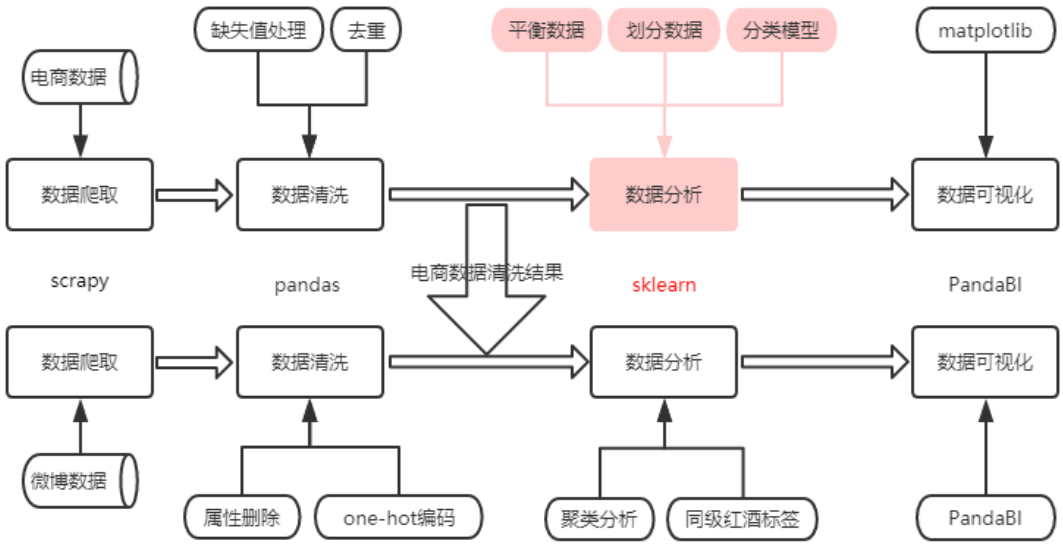
- 了解数据分析的处理流程
- 了解 sklearn，xgboost 的使用

二、 实验环境

Python3 开发环境，第三方包有 pandas，sklearn，xgboost

三、 实验步骤

本节处理的内容有：



模型构建处理流程如下：

- 读取数据
- 平衡数据
- 处理离散型特征
- 划分测试集训练集
- 模型构建
- 模型效果
- 特征选择再转向模型构建，择优保留最佳模型

1. 读取数据

```
import pandas as pd
df = pd.read_csv("wine_processed.csv")
```

```
df.head(2)
```

	keyword	price	产区	原产地	口感	国产/进口	特性	甜度	颜色	冰酒/贵腐/甜酒	...	长相思 (Sauvignon Blanc)	内比奥罗 (Nebbiolo)	佳美 (Gamay)	品丽珠 (Cabernet Franc)
0	通化/tonghua	0-50	其它	法国	饱满	进口	普通餐酒	半干型	宝石红	0	...	0	0	0	0
1	长城/greatwall	0-50	其它	中国	柔和	国产	普通餐酒	干型	宝石红	0	...	0	0	0	0

2 rows × 34 columns

2. 平衡数据

观察每个类别的数据，采用欠采样/过采样的方法对数据进行平衡处理

```
df.groupby("price").size()
```

```
price
0-50                          1842
100-150                       1425
1000-2000                      226
150-250                       1699
2000-9223372036854775807       414
250-500                       1590
50-100                        3029
500-1000                       726
dtype: int64
```

由此可见，[1000-2000],[2000-MAX],[500-1000]的价格区间需要增长一定的量，通过复制的方式增加该价格区间的数据量；[50-100]的数据量较多，欠采样该价格区间的数据。处理代码如下：

```
# 平衡数据
tmp1 = df[df.price=='1000-2000']
tmp2 = df[df.price=='2000-9223372036854775807']
tmp3 = df[df.price=='500-1000']
for i in range(6):
    df = df.append(tmp1, ignore_index=True)
for j in range(3):
    df = df.append(tmp2,ignore_index=True)
df = df.append(tmp3,ignore_index=True)

## 删除50-100的一半数据
tmp4 = df[df.price=='50-100']
df = df[df.price!='50-100']
df = df.append(tmp4[::2],ignore_index=True)

print(df.groupby("price").size())
```

```
price
0-50                        1842
100-150                     1425
1000-2000                   1582
150-250                     1699
2000-9223372036854775807    1656
250-500                     1590
50-100                      1515
500-1000                    1452
dtype: int64
```

从结果看，各个价格区间的数据量持平。

3. 处理离散型特征

算法不允许“绿色”这种非数值型特征，需要做转换处理

```
# 划分X,Y
X = df[df.columns.difference(['price'])]
Y = df['price']
print(X.shape)
print(Y.shape)
```

```
(12761, 33)
(12761,)
```

```
# 对非字符型特征进行数值编码
# X = pd.get_dummies(X)        #onehot编码
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
features = [col for col in X.columns.values if X[col].dtype == "object"]        #筛选出object类型的特征
encoders = dict()
for f in features:
    cc = le.fit(X[f])
    encoders[f] = cc.classes_
    X[f] = cc.transform(X[f])
#     X[f] = le.fit_transform(X[f])

# 保存字符转换的规则
import pickle
with open("encoders.dict", "wb") as f:
    pickle.dump(encoders, f)
```

4. 划分测试集训练集

```
from sklearn.model_selection import train_test_split

# 划分数据训练集，测试集
# 80%训练集，20%的测试集；为了复现实验，设置一个随机数,划分结果是确定的
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42 )
```

5. 模型构建

```
# 模型相关（载入模型--训练模型--模型预测）
from xgboost import XGBClassifier
model = XGBClassifier(learning_rate=0.25, n_estimators=100, objective= "multi:softmax", num_class=8, max_depth=12,
model.fit(x_train,y_train)            # 训练模型（训练集）
y_pred = model.predict(x_test)        # 模型预测（测试集），y_pred为预测结果
```

6. 模型效果

```
# 性能评估
from sklearn.metrics import accuracy_score   # 准确率
accuracy = accuracy_score(y_test,y_pred)
print("accuarcy: %.2f%%" % (accuracy*100.0))

from sklearn.metrics import  classification_report
#输出详细的分类性能
print(classification_report(y_pred,y_test,target_names=['0-50','50-100','100-150'
```

```
accuarcy: 68.31%
                  precision    recall  f1-score   support

            0-50       0.77      0.67      0.72       439
          50-100       0.51      0.53      0.52       266
         100-150       0.98      0.90      0.94       369
         150-250       0.55      0.59      0.57       324
         250-500       0.96      0.94      0.95       313
        500-1000       0.51      0.56      0.53       285
       1000-2000       0.37      0.41      0.39       249
2000-sys.maxsize       0.76      0.73      0.75       308

       micro avg       0.68      0.68      0.68      2553
       macro avg       0.67      0.67      0.67      2553
    weighted avg       0.70      0.68      0.69      2553
```

7. 特征选择与再训练

首先查看模型各个维度的重要性：

```
feature_importance=pd.DataFrame(list(model.get_booster().get_fscore().items()),
columns=['feature','importance']).sort_values('importance', ascending=False)
print('',feature_importance)
```

```
                    feature  importance
2                   keyword       25477
1                   alcohol       16205
10                     year       12589
3                        特性        9406
5                        口感        9345
8                        颜色        7279
7                        产区        7117
13                     原产地        6927
4   赤霞珠（Cabernet Sauvignon）        4773
9                        其它        4543
0                     国产/进口        3752
14               梅洛（Merlot）        3414
18    西拉/设拉子（Syrah/Shiraz）        2616
12                       甜度        2272
21     品丽珠（Cabernet Franc）        1233
17                     白葡萄酒        1051
15                     桃红葡萄酒         627
19          黑皮诺（Pinot Noir）         583
```

掌握特征的重要性排列之后，我们可以选择部分特征再做训练，对比之前的效果：

```
# 掌握特征重要性之后重新再训练
choose = list(feature_importance[feature_importance.importance>400]["feature"])
# 划分X,Y
# X = df[df.columns.difference(['price'])]
X = df[choose]
Y = df['price']
print(X.shape)
print(Y.shape)
```

结果如下：

```
(12761, 22)
(12761,)
/usr/local/python3/lib/python3.6/site-packages/ipykernel_launcher.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy

accuarcy: 68.59%
```

可见模型的准确率并不是很高，提高准确率的方法有三个方向：

- 区间数设置：我们定义的红酒价格区间是自己设立的，可以根据真实数据的效果修改区间数，再做尝试构建预测模型。
- 准确的数据：数据的来源是京东，且在处理价格的时候是采用 jieba 分词及语法的形式对瓶数进行提取，可能会存在识别错的情况，即引入了脏数

据，可以把处理条件更严格一些，或者爬取更为准确的数据，或者加入人工删选的步骤保证数据的准确度。

- 算法选择：本案例中采用的方法是 xgboost，采用的是其默认参数，可以修改参数优化模型，也可以尝试其他算法，同样的，还可以做多算法的融合结果。

本案例不做拓展，学生可尝试从这三个方向提高其准确率。