Mushroom Dataset - 决策树

下面的示例数据需要我们通过一些蘑菇的若干属性判断这个品种是否有毒。 数据链接：https://www.kaggle.com/uciml/mushroom-classification ，

每个样本描述了蘑菇的 22 个属性，比如形状、气味等等 响应为这个蘑菇是否可食用。

其中 6513 个样本做训练，1611 个样本做测试。

导入工具包

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

#显示中文
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
```

Reading the file

调用 head 函数看看每个特征的基本情况

In [2]:

```
dpath = "./data/"
df = pd.read_csv(dpath + "mushrooms.csv")

df.head()
```

In [3]:

```
#数据基本信息
df.info()
```

特征编码

特征全是类别型变量，很多模型需要数值型的输入（Logistic 回归、xgboost...），通常对类别行特征独热编码（OneHotEncoder） LableEncoder 虽然也可以将类别型变量变成数值，但结果是有序的，而颜色等特征是没有序关系，与实际不符。 决策树最好也将类型特征用独热编码转换，但这样会使得分支特别细。 LightGBM 里的决策树支持类别型特征，只需 LableEncoder 编码即可 这里我们为了简单也用 LableEncoder（用 LabelEncoder 会使得特征分支少很多可能，有些情形下性能并不好，但这个例子很简单）

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
for col in df.columns:
    df[col] = labelencoder.fit_transform(df[col])
df.head()
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.countplot(x="gill-color", hue="class",data=df)
```

数据集是一个文件，我们自己分出一部分来做测试吧（不是校验集）

```
y = df['class']
X = df.drop('class', axis = 1)
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,ra
ndom_state=4)
columns = X_train.columns
```

体验一下分裂及 gini 计算

```
df1 = pd.DataFrame({'gill-color':X_train['gill-color'], 'class':y_trai
n})
df1.sort_values(by=['gill-color'],ascending=True)

gini=[]
for i in  range(11) :
    threshold = i + 0.5
    dl  = df[df['gill-color'] <= threshold]
    dr  = df[df['gill-color'] > threshold]

    dl_0 = (dl['class'] == 0).sum()
    dl_1 = (dl['class'] == 1).sum()
    dr_0 = (dr['class'] == 0).sum()
    dr_1 = (dr['class'] == 1).sum()

    total_l = dl_0 + dl_1
    total_r = dr_0 + dr_1
    total = total_l + total_r

    h_l = 2.0 * dl_0/total_l * dl_1/total_l
    h_r = 2.0 * dr_0/total_r * dr_1/total_r
    gini.append(h_l*total_l/total + h_r*total_r/total)

print(gini)
```

默认参数的决策树

```
from sklearn.tree import DecisionTreeClassifier

model_tree = DecisionTreeClassifier()
model_tree.fit(X_train, y_train)
y_prob = model_tree.predict_proba(X_test)[:,1] # This will give you positive class prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
model_tree.score(X_test, y_pred)
print ('The AUC of default Desicion Tree is', roc_auc_score(y_test,y_pred))
df = pd.DataFrame({"columns":list(columns), "importance":list(model_tree.feature_importances_.T)})
df.sort_values(by=['importance'],ascending=False)
plt.bar(range(len(model_tree.feature_importances_)), model_tree.feature_importances_)
plt.show()
```

决策树超参数调优

决策树的超参数有：

1. max_depth（树的深度）或 max_leaf_nodes（叶子结点的数目）、
2. min_samples_leaf（叶子结点的最小样本数）、min_samples_split（中间结点的最小样本树）、min_weight_fraction_leaf（叶子节点的样本权重占总权重的比例）
3. min_impurity_split（最小不纯净度）也可以调整
4. max_features（最大特征数目）、

这个数据集的任务不难，深度设为 3-10 之间

```
from sklearn.tree import DecisionTreeClassifier

model_DD = DecisionTreeClassifier()

max_depth = range(3,10,1)
min_samples_leaf = range(5,15,1)
tuned_parameters = dict(max_depth=max_depth, min_samples_leaf=min_samples_leaf)
```

In [48]:

```
from sklearn.model_selection import GridSearchCV
gird = GridSearchCV(model_DD, tuned_parameters, scoring='roc_auc',cv=10)
gird.fit(X_train, y_train)
print("Best: %f using %s" % (gird.best_score_, gird.best_params_))
y_prob = gird.best_estimator_.predict_proba(X_test)[:,1] # This will give you positive class prediction probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
gird.score(X_test, y_pred)
```

Out[50]:

```
print ('The AUC of GridSearchCV Desicion Tree is', roc_auc_score(y_test, y_pred))
```

```
import graphviz
import pydotplus
from sklearn import tree
#dotfile = StringIO.StringIO()
dot_data = tree.export_graphviz(gird.best_estimator_, out_file='best_tree.dot',feature_names=columns,  class_names='class',)

#$dot -Tpng best_tree.dot -o best_tree.png

test_means = gird.cv_results_[ 'mean_test_score' ]
#print(test_means.shape)
# plot results
test_scores = np.array(test_means).reshape(len(min_samples_leaf), len( max_depth))

for i, value in enumerate(min_samples_leaf):
    plt.plot(max_depth, test_scores[i], label= 'min_samples_leaf:'   + str(value))


plt.legend()
plt.xlabel( 'max_depth' )

plt.ylabel( 'AUC' )
plt.show()
```

```
#DD.grid_scores_
test_means = gird.cv_results_[ 'mean_test_score' ]

# plot results
test_scores = np.array(test_means).reshape(len(max_depth), len(min_sampl
es_leaf ))

for i, value in enumerate(max_depth):
    plt.plot(min_samples_leaf, test_scores[i], label= 'max_depth:'   + s
tr(value))


plt.legend(loc='lower right')
plt.xlabel( 'min_samples_leaf' )

plt.ylabel( 'AUC' )
plt.show()

plt.bar(columns, gird.best_estimator_.feature_importances_)
plt.xticks(rotation=-80)    # 设置 x 轴标签旋转角度
plt.show()

plt.barh(columns, gird.best_estimator_.feature_importances_)
#plt.xticks(rotation=-80)    # 设置 x 轴标签旋转角度
plt.xlabel(u'特征重要性',fontsize = 14)
plt.show()

df2 = pd.DataFrame({"columns":list(columns), "importance":list(gird.best
_estimator_.feature_importances_.T)})
df2.sort_values(by=['importance'],ascending=False)
```

进一步根据特征重要性选择特征值

```
from numpy import sort
from sklearn.feature_selection import SelectFromModel

# Fit model using each importance as a threshold
thresholds = sort(gird.best_estimator_.feature_importances_)
for thresh in thresholds:
  # select features using threshold
  selection = SelectFromModel(gird.best_estimator_, threshold=thresh, pr
efit=True)
  select_X_train = selection.transform(X_train)
```

```
  # train model
  selection_model = DecisionTreeClassifier()
  selection_model.fit(select_X_train, y_train)

# eval model
  select_X_test = selection.transform(X_test)
  y_pred = selection_model.predict(select_X_test)
  predictions = [round(value) for value in y_pred]
  accuracy = accuracy_score(y_test, predictions)
  print("Thresh=%.3f, n=%d, Accuracy: %.2f%%" % (thresh, select_X_train.
shape[1],
      accuracy*100.0))
```

最佳模型

```
from numpy import sort
from sklearn.feature_selection import SelectFromModel

# Fit model using each importance as a threshold
thresholds = sort(gird.best_estimator_.feature_importances_)
idx = len(thresholds) - 7
thresh = thresholds[idx]

# select features using threshold
selection = SelectFromModel(gird.best_estimator_, threshold=thresh, pref
it=True)
select_X_train = selection.transform(X_train)

# train model
selection_model = DecisionTreeClassifier()
selection_model.fit(select_X_train, y_train)

# eval model
select_X_test = selection.transform(X_test)
y_pred = selection_model.predict(select_X_test)
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_test, predictions)
print("Thresh=%.3f, n=%d, Accuracy: %.2f%%" % (thresh, select_X_train.sh
ape[1], accuracy*100.0))


selection_model.feature_importances_
```