【华子】机器学习012-用随机森林构建汽车评估模型及模型的优化提升方法 -

(本文所使用的Python库和版本号: Python 3.5, Numpy 1.14, scikit-learn 0.19, matplotlib 2.2)

在前面的文章中（**|机器学习007-用随机森林构建共享单车需求预测模型|**）已经介绍了用随机森林方法构建共享单车需求预测模型，在代码实现层面上来讲，构建随机森林模型非常简单。

下面我们同样使用随机森林算法构建汽车评估模型，用于根据汽车的六个基本特性来评估汽车的质量。

1. 准备数据集

本项目所使用的数据集来源于加利福尼亚大学欧文分校（UCI）大学的公开数据集：https://archive.ics.uci.edu/ml/datasets/Car+Evaluation。这是专门用于解决多分类问题的一个小型数据集，该数据集的基本信息为：

Abstract: Derived from simple hierarchical decision model, this database may be useful for testing constructive induction and structure discovery methods.

Data Set Characteristics:	Multivariate	**Number of Instances:**	1728	**Area:**	N/A
Attribute Characteristics:	Categorical	**Number of Attributes:**	6	**Date Donated**	1997-06-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	807764

即整个数据集专门用于多分类模型，没有缺失值，一共有1728个样本，每个样本含有6个关于汽车的基本属性，每个样本对应于一个标记，表示汽车质量的好坏，如下所示：

数据集说明	取值范围	含义说明
属性 1-buying	vhigh, high, med, low	购买价格
属性 2-maint	vhigh, high, med, low	维护价格
属性 3-doors	2, 3, 4, 5more	车门数
属性 4-persons	2, 4, more	载人数
属性 5-lug_boot	small, med, big	行李箱大小
属性 6-safety	low, med, high	评估的安全性
标记	unacc, acc, good, vgood	汽车质量

在对数据集有了基本了解的基础上，可以用代码来具体分析，此处我用pandas来提取数据集中的原始数据，代码如下：

```
# 准备数据集
dataset_path='D:\PyProjects\DataSet\CarEvaluation/car.data'
df=pd.read_csv(dataset_path,header=None)
print(df.info()) # 加载没有问题
# 原数据集包含有1728个样本，每一个样本含有6个features，一个label
print(df.head())
raw_set=df.values
```

----------输--------出----------

RangeIndex: 1728 entries, 0 to 1727 Data columns (total 7 columns): 0 1728 non-null object 1 1728 non-null object 2 1728 non-null object 3 1728 non-null object 4 1728 non-null object 5 1728 non-null object 6 1728 non-null object dtypes: object(7) memory usage: 94.6+ KB None 0 1 2 3 4 5 6 0 vhigh vhigh 2 2 small low unacc 1 vhigh vhigh 2 2 small med unacc 2 vhigh vhigh 2 2 small high unacc 3 vhigh vhigh 2 2 med low unacc 4 vhigh vhigh 2 2 med med unacc

----------完----------

通过df.info()可以看出该数据集的7列都是object类型，故而难以直接应用到机器学习领域，需要做进一步的类型转换处理，如下代码：

```
# 数据集中的特征向量包括有多个String，故而type是object，需要转换为数值
from sklearn import preprocessing
```

```
label_encoder=[] # 放置每一列的encoder
encoded_set = np.empty(raw_set.shape)
for i,_ in enumerate(raw_set[0]):
    # fit_tranform
    encoder=preprocessing.LabelEncoder()
    encoded_set[:,i]=encoder.fit_transform(raw_set[:,i])
    print(encoder.classes_)
    label_encoder.append(encoder)
print('-------')
dataset_X = encoded_set[:, :-1].astype(int)
dataset_y = encoded_set[:, -1].astype(int)
# print(dataset_X.shape) # (1728, 6)
# print(dataset_y.shape) #(1728,)
print(dataset_X[:5]) # 可以看出每个特征向量都将string转变为int
print(dataset_y[:5]) # 检查没有问题

# 将数据集拆分为train set 和test set
from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y=train_test_split(dataset_X,dataset_y,
                                                  test_size=0.3,random_state=42)
# print(train_X.shape) # (1209, 6)
# print(train_y.shape) # (1209,)
# print(test_X.shape) # (519, 6)
```

----------**输**--------**出**----------

['high' 'low' 'med' 'vhigh'] ['high' 'low' 'med' 'vhigh'] ['2' '3' '4' '5more'] ['2' '4' 'more'] ['big' 'med' 'small'] ['high' 'low' 'med'] ['acc' 'good' 'unacc' 'vgood'] [[3 3 0 0 2 1] [3 3 0 0 2 2] [3 3 0 0 2 0] [3 3 0 0 1 1] [3 3 0 0 1 2]] [2 2 2 2 2]

----------**完**----------

可以看出转换之后的数据集都是int型，故而可以输入到模型中进行训练和预测。同时，为了训练和测试的方便，将整个数据集划分为训练集（占比70%，即1209个样本）和测试集（占比30%，即519个样本）。

##########**小**************结**##########

1，由于本次数据集的属性和标记都是string类型，故而需要先转变为数值型。转变是通过LabelEncoder()函数完成的。

2，这里使用的转变器（即LabelEncoder()实例）需要保存，便于以后对新样本属性进行转换，或者对预测出来的标记再反向转变成string，此处将其保存到label_encoder这个list中。

##########

2. 构建随机森林分类模型和模型评估

2.1 随机森林分类模型的构建

随机森林分类模型的构建非常简单，可以参考**机器学习007-用随机森林构建共享单车需求预测模型**]。如下代码先构建一个随机森林分类器，然后用训练集来训练该分类器，最后用测试集来检查模型的好坏，打印出模型评价指标。关于模型评价指标的具体含义和计算方法，可以参考**机器学习011-分类模型的评估：准确率，精确率，召回率，F1值**。

```
# 建立随机森林分类器
from sklearn.ensemble import RandomForestClassifier
rf_classifier=RandomForestClassifier(n_estimators=200,max_depth=8,random_state=37)
rf_classifier.fit(train_X,train_y) # 用训练集进行训练

# 用测试集评估模型的准确率，精确率，召回率，F1值：
def print_model_evaluations(classifier,test_X, test_y,cv=5):
    from sklearn.cross_validation import cross_val_score
    accuracy=cross_val_score(classifier,test_X,test_y,
                             scoring='accuracy',cv=cv)
    print('准确率：{:.2f}%'.format(accuracy.mean()*100))
    precision=cross_val_score(classifier,test_X,test_y,
                             scoring='precision_weighted',cv=cv)
    print('精确度：{:.2f}%'.format(precision.mean()*100))
    recall=cross_val_score(classifier,test_X,test_y,
                             scoring='recall_weighted',cv=cv)
    print('召回率：{:.2f}%'.format(recall.mean()*100))
    f1=cross_val_score(classifier,test_X,test_y,
                             scoring='f1_weighted',cv=cv)
    print('F1  值：{:.2f}%'.format(f1.mean()*100))

print_model_evaluations(rf_classifier,test_X,test_y)
```

----------**输**--------**出**----------

准确率：89.19% 精确度：88.49% 召回率：89.19% F1 值：88.32%

----------**完**----------

2.2 随机森林分类模型的全面评估

更进一步的，为了更全面的评估该模型，可以将模型在测试集上的混淆矩阵和分类报告打印出来，关于混淆矩阵和分类报告，可以参考**机器学习011-分类模型的评估：准确率，精确率，召回率，F1值。**如下所示：

```
# 打印模型的混淆矩阵和各个类别的评价指标
# 使用sklearn 模块计算混淆矩阵
from sklearn.metrics import confusion_matrix
test_y_pred=rf_classifier.predict(test_X)
confusion_mat = confusion_matrix(test_y, test_y_pred)
print(confusion_mat) #看看混淆矩阵长啥样
print('*'*50)
from sklearn.metrics import classification_report
print(classification_report(test_y, test_y_pred))
```

----------**输**--------**出**----------

[[108 2 7 1] [9 8 0 2] [3 0 355 0] [3 0 0 21]]

precision recall f1-score support

0 0.88 0.92 0.90 118 1 0.80 0.42 0.55 19 2 0.98 0.99 0.99 358 3 0.88 0.88 0.88 24

avg / total 0.95 0.95 0.94 519

----------**完**----------

从上面的分类报告中可以看出，这个模型在类别2上表现最好，精确率和召回率均在98%以上，但在类别1，虽然精确率有80%，但召回率却低至42%，所得到的F1值也只有55%，表明这个模型还有进一步优化的空间（出现这种结果也有可能是test set中类别2的样本数最多，而类别1的样本数最少导致的）。

2.3 用该分类模型预测新样本数据

一个模型经过训练和优化之后，一旦达到了我们的分类要求，就可以用来预测新样本数据，如下我们自己构建了一个新的汽车样本，这个样本汽车的购买价格和维护价格都非常高，有2个车门，载人数2人，后备箱比较小，安全性比较低（很有可能是那种2座的豪华车吧。。。）。看看这个分类模型对这种车的质量评估怎么样。

```
# 看起来该随机森林分类器的分类效果还是很不错的，
# 那么可以用这个比较理想的模型来预测新数据，
new_sample=['vhigh','vhigh','2','2','small','low']
# 在把这个样本输入模型之前，需要将样本中的string转变为int
# 采用和上面train set相同的encoder来编码
encoded_sample=np.empty(np.array(new_sample).shape)
for i,item in enumerate(new_sample):
    encoded_sample[i]=int(label_encoder[i].transform([item]))
    # 这儿的item一定要加【】,否则报错。而且要转变为int类型
print(encoded_sample.reshape(1,-1)) # 和上面打印的print(encoder.classes_)对应一致

# 用成熟分类模型对该新样本进行分类，得到分类结果：
output=rf_classifier.predict(encoded_sample.reshape(1,-1))
print('output: {}, class: {}'.format(output,
       label_encoder[-1].inverse_transform(output)[0]))
```

----------**输**--------**出**----------

[[3. 3. 0. 0. 2. 1.]] output: [2], class: unacc

----------**完**----------

在将新样本数据输入模型之前，需要对样本数据进行转换（即对特征向量进行编码，将人可以阅读的字符串转变为机器可以阅读的数值），注意此时的转换要用到和前面训练集相同的转换方法，即使用前面放置到label_encoder这个list中的encoder来转换，可以将转换之后的数值打印出来进行验证。分类模型根据该样本的六个属性，判断出该汽车的质量为2，此时我们需要将2再反向转换为字符串（即反编码，或解码，即将机器可以阅读的数值转变为人可以阅读的字符串），经过解码后，发现该汽车的质量为“unacc”，即unacceptable。

可以想象一下，一辆价格老贵老贵，维护起来也老贵老贵，后备箱又小，只能坐两个人，而且安全性还非常低的汽车，你能接收吗？？？屌丝没钱不能接受，土豪虽然可以用这种车来泡妞，但是安全性太低，土豪也接收不了吧。。。

########################小********结################################**

1，随机森林分类模型的构建非常简单，直接调用sklearn模块中的RandomForestClassifier 类即可。

2，对分类模型的评估可以直接打印其整体的准确率，精确率，召回率，F1值，也可以打印该模型在各个不同类别上的评价指标，打印其混淆矩阵和分类报告。

###

3. 模型的优化提升方法

上面的分类模型貌似在测试集上的表现还不错，但是还有提升空间，主要有以下两个方面的优化提升。

3.1 模型超参数的优化—验证曲线

前面在定义随机森林分类器时，我们随机地定义该分类器的参数为：n_estimators=200,max_depth=8，但是这些随机定义的参数真的是最优参数组合吗？怎么获取这些参数的最优值了？这就是验证曲线的作用了。下面首先优化n_estimators参数，看看取不同值时，该模型的准确率是否有明确的改善。

验证曲线是，横轴为某个超参数的一系列值，由此来看不同参数设置下模型的准确率。

如下代码，**使用sklearn中的validation_curve可以验证不同参数取值时模型的准确率。**

```
# 提升模型的分类效果：优化模型的某个参数，
# 第一步：优化n_estimators参数，得出不同值对应的得分效果
from sklearn.model_selection import validation_curve
optimize_classifier1=RandomForestClassifier(max_depth=4,random_state=37)
parameter_grid=np.linspace(20,400,20).astype(int)  #起点20，终点400，生成20个数据
#得到训练分数和验证分数
train_scores,valid_scores=validation_curve(optimize_classifier1,train_X,train_y,
                                           'n_estimators',parameter_grid,cv=5)
# cv交叉验证的折数
# cv=4，会输 出4列结果，cv=5，会输出5列结果

# 打印优化结果
print('n_estimators optimization results-------->>>')
print('train scores: \n ',train_scores)
print('-'*80)
print('valid scores: \n ',valid_scores)
```

----------------------------------输---------出-----------------------------

n_estimators optimization results-------->>> train scores: [[0.78549223 0.80144778 0.80785124 0.79338843 0.80165289] [0.8 0.80972079 0.81095041 0.81921488 0.83057851] [0.8134715 0.81075491 0.81095041 0.81404959 0.81714876]

.....

valid scores: [[0.77459016 0.79338843 0.80082988 0.76763485 0.80497925] [0.79918033 0.79338843 0.80497925 0.80082988 0.8340249] [0.81967213 0.80578512 0.80082988 0.78008299 0.82572614] [0.79918033 0.80991736 0.7966805 0.78838174 0.82572614]

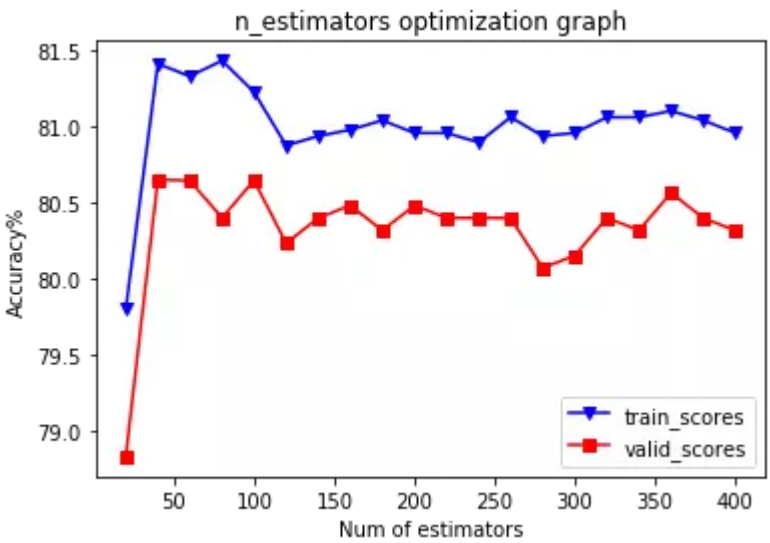
.....

----------**完**----------

得到的trains_scores和valid_scores矩阵很大，此处只显示一部分.虽然此处得到了验证曲线的结果，但是难以直接观察结果的好坏，故而我自己定义一个绘图函数，将验证曲线的结果绘制成图，代码如下：

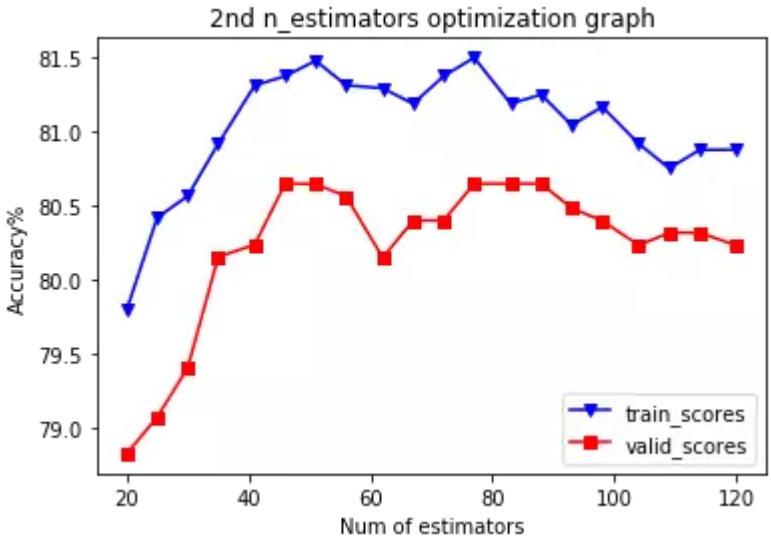
```
# 定义一个绘图函数，绘制train scores 和valid scores
def plot_valid_curve(grid_arr,train_scores,valid_scores,
                     title=None,x_label=None,y_label=None):
    '''plot train_scores and valid_scores into a line graph'''
    assert train_scores.shape==valid_scores.shape, \
        'expect train_scores and valid_scores have same shape'
    assert grid_arr.shape[0]==train_scores.shape[0], \
        'expect grid_arr has the same first dim with train_scores'
    plt.figure()
    plt.plot(grid_arr, 100*np.average(train_scores, axis=1),
             color='blue',marker='v',label='train_scores')
    plt.plot(grid_arr, 100*np.average(valid_scores, axis=1),
             color='red',marker='s',label='valid_scores')
    plt.title(title) if title is not None else None
    plt.xlabel(x_label) if x_label is not None else None
    plt.ylabel(y_label) if y_label is not None else None
    plt.legend()
    plt.show()

plot_valid_curve(parameter_grid,train_scores,valid_scores,
                 title='n_estimators optimization graph',
                 x_label='Num of estimators',y_label='Accuracy%')
```



上图中可以看出，在estimators取值为50附近时，能够得到最高的准确率，故而我们可以进一步优化estimators在50附近的取值。如下代码：

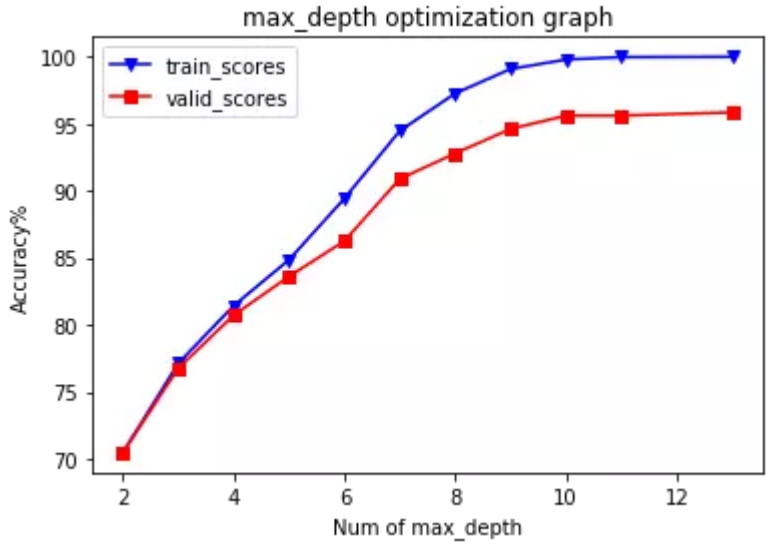
```
# 第二步：对n_estimators做进一步细致优化
# 图中可以看出，n_estimators在100以内所得到的准确率最高，故而需要进一步做更精细的优化
parameter_grid2=np.linspace(20,120,20).astype(int)
train_scores,valid_scores=validation_curve(optimize_classifier1,train_X,train_y,
                                           'n_estimators',parameter_grid2,cv=5)
plot_valid_curve(parameter_grid2,train_scores,valid_scores,
                 title='2nd n_estimators optimization graph',
                 x_label='Num of estimators',y_label='Accuracy%')
# 从图中可以看出准确率最高的点是第6,7，12附近，对应的estimators是46,51,77,
# 故而后面暂定为50
```



从上图中可以看出，准确率的最高点对应的estimators大约为46,51,77，故而我们确定最优的estimators参数的取值为50.

对于max_depth，可以采用同样的验证曲线来优化，得到最优值，如下代码和图：

```
# 第三步：对max_depth进行优化：
optimize_classifier2=RandomForestClassifier(n_estimators=50,random_state=37)
parameter_grid3=np.linspace(2,13,11).astype(int)
print(parameter_grid3) # [ 2  3  4  5  6  7  8  9 10 11 13]
train_scores3,valid_scores3=validation_curve(optimize_classifier2,train_X,train_y,
                                           'max_depth',parameter_grid3,cv=5)
plot_valid_curve(parameter_grid3,train_scores3,valid_scores3,
                 title='max_depth optimization graph',
                 x_label='Num of max_depth',y_label='Accuracy%')
# 从图中可以看出，取max_depth=10，11,13时准确率一样，故而取max_depth=10
```



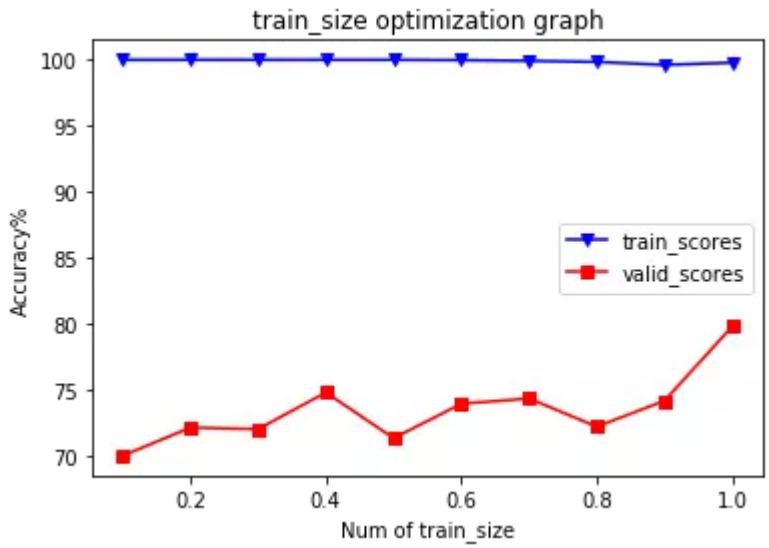
从上图中可以看出，准确率的最高点对应的max_depth大约为10,11,13，这几个点处的结果几乎一样，故而我们确定最优的max_depth参数的取值为10.

3.2 训练集大小对模型的影响—学习曲线

前面我们通过验证曲线优化了模型中各种参数，得到了参数的最佳取值，但有的时候，训练集的大小也会对模型的效果有影响，此时我们可以用学习曲线来判断最佳的训练集大小。代码如下：

```
# 前面都是优化随机森林分类器的内置参数，但是没有考虑训练集的大小对模型效果的影响
# 前面都是用train_X来优化模型，train_X含有1209个样本，
# 下面考察一下训练集样本大小对模型效果的影响--即学习曲线
from sklearn.model_selection import learning_curve

optimize_classifier3=RandomForestClassifier(n_estimators=50,
                                            max_depth=10,
                                            random_state=37)
parameter_grid4=np.array([0.1,0.2,0.3,0.4,0.5,0.6,0.7,.8,.9,1.]) # dataset最多有1728个样本
train_sizes,train_scores4,valid_scores4=learning_curve(optimize_classifier3,
                                                       dataset_X,dataset_y,
                                                  train_sizes=parameter_grid4,cv=5)
# print(train_sizes) # [ 138  276  414  552  691  829  967 1105 1243 1382]
# 最大也只能到dataset_X样本数的80%，即 1728*0.8=1382
plot_valid_curve(parameter_grid4,train_scores4,valid_scores4,
                 title='train_size optimization graph',
                 x_label='Num of train_size',y_label='Accuracy%')
# 可以看出，在train_size=1382(即x轴为1.0)时得到的准确率最大，约为80%左右。
```



可以从图中看出，训练集大小貌似越大越好，因为训练集越大，模型训练的越充分，得到的valid_scores与train_scores的差距越小，这里的差距实际上就是“过拟合”现象。而此处通过提高训练集的大小，可以减小过拟合现象。

还有一点，learning_curve里面貌似把最大取值固定为整个数据集的80%

3.3 用最优参数重新建立模型，判断模型的质量

前面我们花了好长时间来优化模型，得到了最佳超参数和最佳训练集大小，那么，如果用这些参数来训练模型，得到模型的质量会怎么样了？非常好还是非常差？如下直接上代码。

```
# 用所有最优参数来重新构建模型，并判断此模型的好坏
train_X, test_X, train_y, test_y=train_test_split(dataset_X,dataset_y,
                                                  test_size=0.2,random_state=42)
# 最佳训练集大小为80%

rf_classifier=RandomForestClassifier(n_estimators=50,max_depth=10,random_state=37)
rf_classifier.fit(train_X,train_y) # 用训练集进行训练
print_model_evaluations(rf_classifier,test_X,test_y)
test_y_pred=rf_classifier.predict(test_X)
confusion_mat = confusion_matrix(test_y, test_y_pred)
print('confusion_mat: ------>>>>>')
print(confusion_mat) #看看混淆矩阵长啥样
print('*'*50)
print('classification report: -------->>>>>>')
print(classification_report(test_y, test_y_pred))
```

----------输--------出----------

准确率：89.32% 精确度：88.49% 召回率：89.32% F1 值：88.45% confusion_mat: ------>>>>>> [[71 7 5 0] [1 9 0 1] [0 0 235 0] [1 0 0 16]]

classification report: --------->>>>>> precision recall f1-score support

0 0.97 0.86 0.91 83 1 0.56 0.82 0.67 11 2 0.98 1.00 0.99 235 3 0.94 0.94 0.94 17

avg / total 0.96 0.96 0.96 346

----------**完**----------

貌似比第一次定义的模型在性能上提高了一点点。。。。

参考资料:

1, Python机器学习经典实例，Prateek Joshi著，陶俊杰，陈小莉译