【华子】机器学习013-用朴素贝叶斯分类器估算个人收入阶层 -

(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)

每个人都有权利追求幸福的生活，我等屌丝也不例外，但是，怎么样才能知道自己到底是屌丝阶层还是富帅阶层了?

此处，我们介绍如何利用朴素贝叶斯分类器估算个人的收入阶层，所使用的数据集来源于美国人口普查收入数据集，虽然美国的数据集可能不太适合中国的特色社会主义情况，但此处所使用的分类方法和数据分析方法同样也适用于中国国情。

1. 准备数据集

本项目的数据集来源于美国人口普查收入数据集.首先我们来熟悉一下这个数据集，下面是关于这个数据集的基本信息：

属性	数据范围	说明
1-age	连续值，int	年龄
2-workclass	离散值，String	工作类别
3-fnlwgt	连续值，int	序号
4-education	离散，String	教育程度
5-education_num	连续值，Int	受教育时间
6-maritial_status	离散，String	婚姻状况
7-occupation	离散，String	职业
8-relationship	离散，String	关系
9-race	离散，String	种族
10-sex	离散，String	性别
11-capital_gain	连续值，int	资本收益
12-capital_loss	连续值，int	资本损失
13-hours_per_week	连续值，int	每周工作小时数
14-native_country	离散，String	原始国籍
Label-Income	>50K, <=50K. 两个离散值	收入（作为标记）

age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

稍微总结一下：这个数据集总共包含有32561个样本，每个样本含有14个属性，1个标记（收入水平，大于50K/y，还是小于等于50K/y）。其中属性中含有int类型和String类型，即在用Pandas读取数据集后，数据类型有些混杂，我们后面需要进一步处理。

1.1 读取数据集

废话不说，直接上代码，读取整个数据集到内存中。

```
# 准备数据集
dataset_path='E:\PyProjects\DataSet\CensusIncome/adult.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32561个样本，每一个样本含有14个features，一个label
# print(df.head())
raw_set=df.values
```

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

RangeIndex: 32561 entries, 0 to 32560 Data columns (total 15 columns): 0 32561 non-null int64 1 32561 non-null object 2 32561 non-null int64 3 32561 non-null object 4 32561 non-null int64 5 32561 non-null object 6 32561 non-null object 7 32561 non-null object 8 32561 non-null object 9 32561 non-null object 10 32561 non-null int64 11 32561 non-null int64 12 32561 non-null int64 13 32561 non-null object 14 32561 non-null object dtypes: int64(6), object(9) memory usage: 3.7+ MB None

----------**完**----------

为了进一步了解整个数据集的数据结构，可以用下面的代码来打印各列数值的基本情况。

```
def print_col_info(dataset):
    '''
    1, values
    2, value type num'''
    col_num=dataset.shape[1]  #列的数量15
    for i in range(col_num):
        print('\ncol-{} info: '.format(i))
        temp=np.sort(list(set(dataset[:,i])))
        print('values: {}'.format(temp))
        print('values num: {}'.format(temp.shape[0]))

print_col_info(raw_set)
```

----------输--------出----------

col-0 info: values: [17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 90] value num: 73

col-1 info: values: [' ?' ' Federal-gov' ' Local-gov' ' Never-worked' ' Private' ' Self-emp-inc' ' Self-emp-not-inc' ' State-gov' ' Without-pay'] value num: 9

col-2 info: values: [12285 13769 14878 ... 1366120 1455435 1484705] value num: 21648

col-3 info: values: [' 10th' ' 11th' ' 12th' ' 1st-4th' ' 5th-6th' ' 7th-8th' ' 9th' ' Assoc-acdm' ' Assoc-voc' ' Bachelors' ' Doctorate' ' HS-grad' ' Masters' ' Preschool' ' Prof-school' ' Some-college'] value num: 16

col-4 info: values: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16] value num: 16

col-5 info: values: [' Divorced' ' Married-AF-spouse' ' Married-civ-spouse' ' Married-spouse-absent' ' Never-married' ' Separated' ' Widowed'] value num: 7

col-6 info: values: [' ?' ' Adm-clerical' ' Armed-Forces' ' Craft-repair' ' Exec-managerial' ' Farming-fishing' ' Handlers-cleaners' ' Machine-op-inspct' ' Other-service' ' Priv-house-serv' ' Prof-specialty' ' Protective-serv' ' Sales' ' Tech-support' ' Transport-moving'] value num: 15

col-7 info: values: [' Husband' ' Not-in-family' ' Other-relative' ' Own-child' ' Unmarried' ' Wife'] value num: 6

col-8 info: values: [' Amer-Indian-Eskimo' ' Asian-Pac-Islander' ' Black' ' Other' ' White'] value num: 5

col-9 info: values: [' Female' ' Male'] value num: 2

col-10 info: values: [0 114 401 594 914 991 1055 1086 1111 1151 1173 1409 1424 1455 1471 1506 1639 1797 1831 1848 2009 2036 2050 2062 2105 2174 2176 2202 2228 2290 2329 2346 2354 2387 2407 2414 2463 2538 2580 2597 2635 2653 2829 2885 2907 2936 2961 2964 2977 2993 3103 3137 3273 3325 3411 3418 3432 3456 3464 3471 3674 3781 3818 3887 3908 3942 4064 4101 4386 4416 4508 4650 4687 4787 4865 4931 4934 5013 5060 5178 5455 5556 5721 6097 6360 6418 6497 6514 6723 6767 6849 7298 7430 7443 7688 7896 7978 8614 9386 9562 10520 10566 10605 11678 13550 14084 14344 15020 15024 15831 18481 20051 22040 25124 25236 27828 34095 41310 99999] value num: 119

col-11 info: values: [0 155 213 323 419 625 653 810 880 974 1092 1138 1258 1340 1380 1408 1411 1485 1504 1539 1564 1573 1579 1590 1594 1602 1617 1628 1648 1651 1668 1669 1672 1719 1721 1726 1735 1740 1741 1755 1762 1816 1825 1844 1848 1876 1887 1902 1944 1974 1977 1980 2001 2002 2042 2051 2057 2080 2129 2149 2163 2174 2179 2201 2205 2206 2231 2238 2246 2258 2267 2282 2339 2352 2377 2392 2415 2444 2457 2467 2472 2489 2547 2559 2603 2754 2824 3004 3683 3770 3900 4356] value num: 92

col-12 info: values: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 70 72 73 74 75 76 77 78 80 81 82 84 85 86 87 88 89 90 91 92 94 95 96 97 98 99] value num: 94

col-13 info: values: [' ?' ' Cambodia' ' Canada' ' China' ' Columbia' ' Cuba' ' Dominican-Republic' ' Ecuador' ' El-Salvador' ' England' ' France' ' Germany' ' Greece' ' Guatemala' ' Haiti' ' Holand-Netherlands' ' Honduras' ' Hong' ' Hungary' ' India' ' Iran' ' Ireland' ' Italy' ' Jamaica' ' Japan' ' Laos' ' Mexico' ' Nicaragua' ' Outlying-US(Guam-USVI-etc)' ' Peru' ' Philippines' ' Poland' ' Portugal' ' Puerto-Rico' ' Scotland' ' South' ' Taiwan' ' Thailand' ' Trinadad&Tobago' ' United-States' ' Vietnam' ' Yugoslavia'] value num: 42

col-14 info: values: [' <=50K' ' >50K'] value num: 2

从上面打印的各列基本信息可以看出：

1，数据集有很多列包含有很多字符串类型，这些包含字符串类型的列被Pandas解读为object类型，而数值型的列被Pandas解读为int类型（此处只有整数，故全都是int）.

2，在字符串列，每一个字符串前面都有一个空格，这个需要后期去除掉。

3，有几个字符串列包含有" ?"这样的缺失值，需要后期做进一步处理。

1.2 数据处理一：对字符串进行归一化处理

可以使用Pandas的Series对应的map函数，结合Lambda表达式来轻松处理，如下代码

```
# 数据处理一：去除字符串数值前面的空格
str_cols=[1,3,5,6,7,8,9,13,14]
for col in str_cols:
    df.iloc[:,col]=df.iloc[:,col].map(lambda x: x.strip())

# print_col_info(df.values) # 检查发现所有的字符串列都已经掉了空格
```

1.3 数据处理二：删除包含缺失值的样本

虽然上面的df.info()结果中显示数据集的每一列中都是non-null，即不包含缺失值，但实际上，该数据集中包含有很多个"?"数值，即为缺失值。对数据集中的缺失值的处理方式有很多种，比如可以使用插值法来填充该缺失值，或者使用前一个值或后一个值来填充缺失值。由于此处我们的数据集样本比较多，故而可以直接将包含有该缺失值的样本删除。

```
# 数据处理二：  删除缺失值样本
# 将?字符串替换为NaN缺失值标志
df.replace("?",np.nan,inplace=True) #inplace=True不创建新的对象,直接在原始对象上尽心修改
# 此处直接删除缺失值样本
df.dropna(inplace=True)
# print(df.shape) # (30162, 15)
```

即原始数据集含有32561个样本，删除含有缺失值的样本后，最终由30162个样本

1.4 数据处理三：对字符串进行编码

字符串对于人很友好，我们一眼就能看明白，但是计算机就犯难了，它不明白是啥玩意儿，所以我们有必要把字符串编码为数值类型的数据，编码方法可以参考我的另一篇文章**|机器学习002-标记编码方法|**.此处直接上代码：

```
# 数据处理三：对字符数据进行编码
from sklearn import preprocessing
label_encoder=[] # 放置每一列的encoder
encoded_set = np.empty(df.shape) #存放编码后的数据
for col in range(df.shape[1]):
    encoder=None
    if df.iloc[:,col].dtype==object: # 字符型数据
        encoder=preprocessing.LabelEncoder()
        encoded_set[:,col]=encoder.fit_transform(df.iloc[:,col])
    else:  # 数值型数据
        encoded_set[:,col]=df.iloc[:,col]
    label_encoder.append(encoder)

# print_col_info(encoded_set) # 全都是数字，没有问题
```

1.5 数据处理四：对数值进行范围缩放

从print_col_info()函数的结果可以看出，在某些特征列中，数值范围变化特别大，比如col10，其数值最小为0，最大为99999。故需要对类似的这些特征进行范围缩放。关于数据缩放方法，可以参考前面文章**|机器学习001-数据预处理技术（均值移除，范围缩放，归一化，二值化，独热编码）|**，此处只使用"范围缩放"的方法。

```
# 数据处理四：对某些列进行范围缩放
# print(encoded_set.dtype) # float64 没问题
cols=[2,10,11]
data_scalers=[] # 专门用来放置scaler
for col in cols:
    data_scaler=preprocessing.MinMaxScaler(feature_range=(-1,1))
    #np.ravel：将多维的array 变成一维(按行，视图模式)
    encoded_set[:,col]=np.ravel(data_scaler.fit_transform(encoded_set[:,col].reshape(-1,1)))
    data_scalers.append(data_scaler)

# print_col_info(encoded_set) # 已经发生了改变，没问题
```

打印的结果比较长，暂时未列出

1.6 数据处理五：拆分数据集为train set和test set

如下代码：

```
dataset_X,dataset_y=encoded_set[:,:-1],encoded_set[:,-1]
# 数据处理五：拆分数据集为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(dataset_X.shape) # (30162, 14)
# print(dataset_y.shape) # (30162,)
# print(train_X.shape) # (21113, 14)
# print(train_y.shape) # (21113,)
# print(test_X.shape) # (9049, 14)
```

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

1. 本项目的数据集中有些特征列是纯数值，而有的列却是字符串型，故而需要单独进行数据处理

2. 对于字符串型数据，需要使用编码器进行编码，得到数值型数据，然后才能输入到分类器中进行分类。

3. 对于数值型数据，需要考虑数值的变化范围是否非常大，比如此处变换太大，那么我们就需要首先进行范围缩放。

4. 数据处理方式还有非常多：比如怎么处理缺失值的问题，是否需要将各个类别样本数转变成一样，是否需要对数据进行归一化处理等。

###

2. 构建朴素贝叶斯分类器

2.1 朴素贝叶斯分类器的构建和训练

在我前面的文章**[机器学习010-用朴素贝叶斯分类器解决多分类问题]**中，已经详细介绍了朴素贝叶斯的构建方法，此处还结合了模型的判断公式，并给出判断结果，如下代码：

```
# 1 建立朴素贝叶斯分类器模型
from sklearn.naive_bayes import GaussianNB
gaussianNB=GaussianNB()
gaussianNB.fit(train_X,train_y)

# 2 用交叉验证来检验模型的准确性，只是在test set上验证准确性
from sklearn.cross_validation import cross_val_score
num_validations=5
accuracy=cross_val_score(gaussianNB,test_X,test_y, scoring='accuracy',cv=num_validations)
print('准确率：{:.2f}%'.format(accuracy.mean()*100))
precision=cross_val_score(gaussianNB,test_X,test_y,scoring='precision_weighted',cv=num_validatio
ns)
print('精确度：{:.2f}%'.format(precision.mean()*100))

recall=cross_val_score(gaussianNB,test_X,test_y,scoring='recall_weighted',cv=num_validations)
```

```
print('召回率：{:.2f}%'.format(recall.mean()*100))
f1=cross_val_score(gaussianNB,test_X,test_y,scoring='f1_weighted',cv=num_validations)
print('F1  值：{:.2f}%'.format(f1.mean()*100))

# 3 打印性能报告
from sklearn.metrics import confusion_matrix
y_pred=gaussianNB.predict(test_X)
confusion_mat = confusion_matrix(test_y, y_pred)
print(confusion_mat) #看看混淆矩阵长啥样

from sklearn.metrics import classification_report
# 直接使用sklearn打印精度，召回率和F1值
target_names = ['<=50K', '>50K']
print(classification_report(test_y, y_pred, target_names=target_names))
```

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

准确率：79.84% 精确度：78.45% 召回率：79.84% F1 值：77.21% [[6420 347] [1518 764]]

precision recall f1-score support

<=50K 0.81 0.95 0.87 6767

>50K 0.69 0.33 0.45 2282

avg / total 0.78 0.79 0.77 9049

----------**完**----------

2.2 用训练好的模型来预测新样本

用训练好的模型来预测新样本在（**[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]**）已经介绍过，此处只需要注意：对新样本数据的处理也要采用训练数据集一样的处理方式，即使用训练集所用字符型编码器来对新样本数据进行编码，且使用相同的数据范围缩放器来进行缩放。如下为实现代码：

```
### 2.2 用训练好的模型来预测新样本
new_sample=[39, 'State-gov', 77516, 'Bachelors', 13,
            'Never-married', 'Adm-clerical', 'Not-in-family',
            'White', 'Male', 2174, 0, 40, 'United-States']
# 先对新样本中的字符型数据编码
encoded_sample=np.empty(np.array(new_sample).shape)
for i,item in enumerate(new_sample):
    if label_encoder[i] is not None:
        encoded_sample[i]=float(label_encoder[i].transform([item]))
    else:
        encoded_sample[i]=item

# print(encoded_sample) # 貌似没有错

# 再对数值较大列进行范围缩放
cols=[2,10,11]

for i,col in enumerate(cols):
```

```
    encoded_sample[col]=np.ravel(data_scalers[i].transform(encoded_sample[col].reshape(-1,1)))
print(encoded_sample) # 检查没有错

# 将新数据放入模型中进行预测
output=gaussianNB.predict(encoded_sample.reshape(1,-1))
print('output: {}, class: {}'.format(output,
       label_encoder[-1].inverse_transform([int(output)])))
```

----------输--------出----------

[39. 5. -0.91332458 9. 13. 4.

0. 1. 4. 1. -0.95651957 -1.
1.38.]
output: [0.], class: <=50K

----------完----------

即新样本经过朴素贝叶斯分类器得到的分类结果为"<=50K"的收入水平。

##########小********结##########**

1. 本项目构建的朴素贝叶斯分类器在测试集上的准确率，精确率，召回率等指标只有80%左右，还有优化空间。

2. 从性能报告中可以看出，测试集中<=50K的样本有6767个，而>50K的样本只有2282，两者的样本数不一样，这个可能是导致>50K这一类别各种指标比较低的原因，所以一个优化方向是准备各个类别中样本数完全一样的数据集，用于训练或测试。

3. 另外一个优化方向是，优化朴素贝叶斯分类器的各种超参数，取得这些超参数的最佳值，但同时要注意模型的过拟合现象。

##########

参考资料:

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