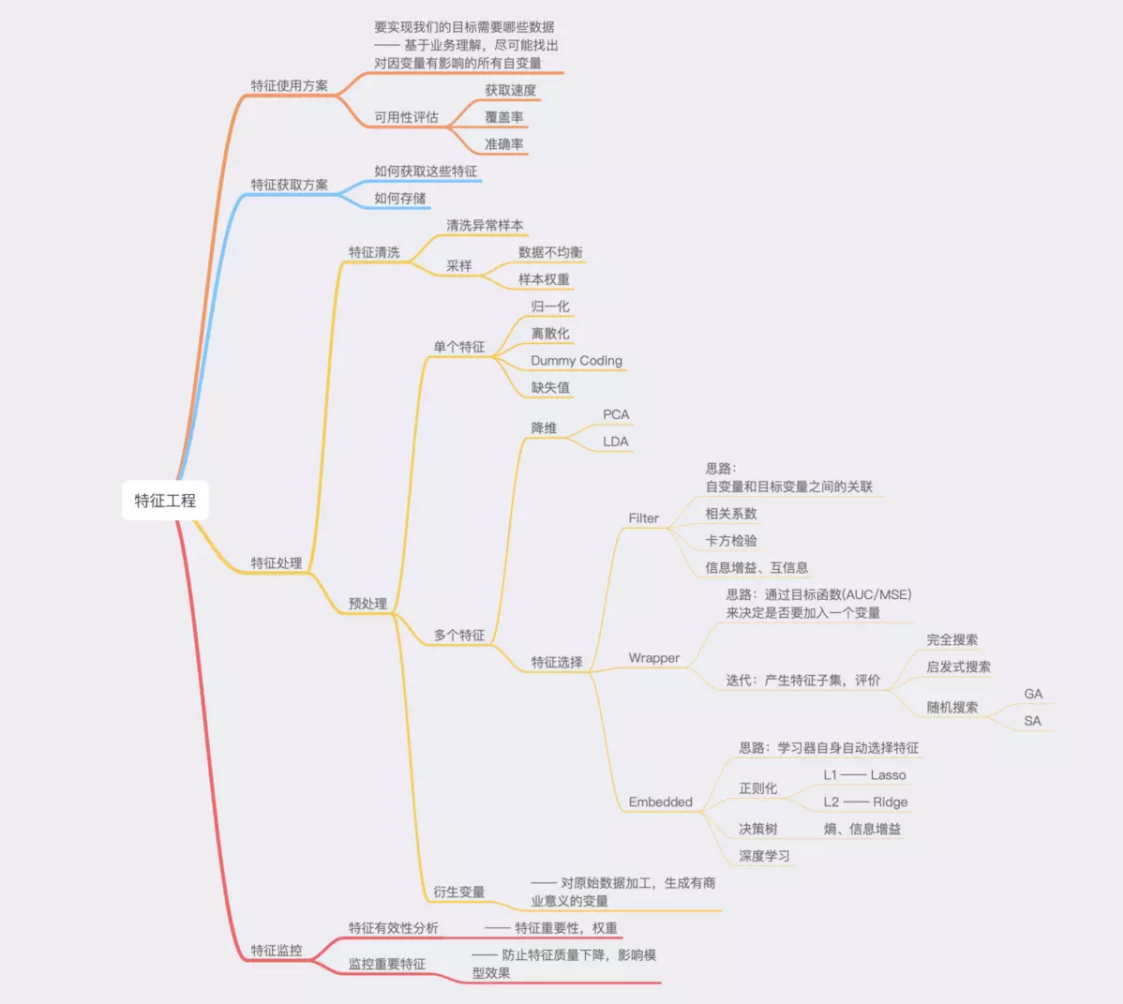
特征工程

2020 年 5 月 4 日

目录

特征工程是一个将原始数据变形成各种特征的过程，“造” 出来的特征通常能够更好的表达问题中的信息，从而让我们能够更好的构建预测模型，提高模型的准确性。

特征工程包括以下方面



特征处理是特征工程的核心部分，sklearn 提供了较为完整的特征处理方法，包括数据预处理、特征选择、降维等。

1 数据分类

数据的类型可以分为：1. **连续性的变量**：比如，身高，体重，化验值等等，2. **分类变量 (categorical data)**：其变量值是定性的，表现为互不相容的类别或属性。分类变量可分为无序变量和有序变量两类。- 无序分类变量也叫名义变量 (nominal) 是指所分类别或属性之间无程度和顺序的差别，例如, 性别 (男、女)，药物反应 (阴性、阳性), 血型 (O、A、B、AB)。- 有序分类变量 (ordinal) 是指各类别之间有程度的差别。如衣服的尺码 (S,M,L,XL,XXL)。

[1]:
```
import pandas as pd
df = pd.DataFrame([
            ['green', 'M', 10.1, 'class1'],
            ['red', 'L', 13.5, 'class2'],
            ['blue', 'XL', 15.3, 'class1']])

df.columns = ['color', 'size', 'price', 'classlabel']
df
```

[1]:

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class2
2	blue	XL	15.3	class1

1.1 编码类别标签

类别通常是无序分类变量，需要将其转换为数值表征，记住此时的数值只代表一个类别，并不表征数值关系。

[2]:
```
from sklearn.preprocessing import LabelEncoder

class_le = LabelEncoder()
y = class_le.fit_transform(df['classlabel'].values)
y
```

[2]:
```
array([0, 1, 0])
```

[3]:
```
# 逆向转换
class_le.inverse_transform(y)
```

[3]:
```
array(['class1', 'class2', 'class1'], dtype=object)
```

1.2 名义变量使用独热编码 (one-hot encoding)

[4]:
```
X = df[['color', 'size', 'price']].values


# color column
color_le = LabelEncoder()
X[:, 0] = color_le.fit_transform(X[:, 0])
X
```

[4]:
```
array([[1, 'M', 10.1],
       [2, 'L', 13.5],
       [0, 'XL', 15.3]], dtype=object)
```

虽然 color 转化为了 0, 1, 2, 但并不能直接使用来建模, 因为在实际使用中, 会认为 2 大于 1, 也就是 red 大于 green. 实际却不是这样的, 所以需要用到 one-hot encoding, 需要使用哑编码 (dummy variable), 每一个值被表示为一个向量.

[5]:
```
from sklearn.preprocessing import OneHotEncoder
# 不设定 sparse=True 的话, onehot 会返回一个 sparse matrix, 可以用 toarray() 将之变回 dense
enc = OneHotEncoder(handle_unknown='ignore',sparse=False)
enc.fit_transform(df['color'].values.reshape(-1,1))
print(enc.transform(df['color'].values.reshape(-1,1)))
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
```

[6]:
```
# pandas 中的 get_dummies 函数是生成哑编码更简单的方法
pd.get_dummies(df[['price', 'color', 'size']])
```

[6]:

	price	color_blue	color_green	color_red	size_L	size_M	size_XL
0	10.1	0	1	0	0	1	0

1	13.5	0	0	1	1	0	0
2	15.3	1	0	0	0	0	1

2 数据集

数据集：判断一个贷款者在后续两年内是否会违约的概率。

[7]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[8]:
```
df = pd.read_csv("data/cs-training.csv",usecols=range(1,12))

df.head() # 显示前面几行的数据
```

[8]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	\
0	1	0.766127	45	
1	0	0.957151	40	
2	0	0.658180	38	
3	0	0.233810	30	
4	0	0.907239	49	

	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	\
0	2	0.802982	9120.0	
1	0	0.121876	2600.0	
2	1	0.085113	3042.0	
3	0	0.036050	3300.0	
4	1	0.024926	63588.0	

	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	\
0	13	0	
1	4	0	
2	2	1	
3	5	0	
4	7	0	

```
   NumberRealEstateLoansOrLines  NumberOfTime60-89DaysPastDueNotWorse  \
0                             6                                     0
1                             0                                     0
2                             0                                     0
3                             0                                     0
4                             1                                     0

   NumberOfDependents
0                 2.0
1                 1.0
2                 0.0
3                 0.0
4                 0.0
```

类别 - SeriousDlqin2yrs - 用户在后续两年内出现 90 天以上的还款逾期

特征 - RevolvingUtilizationOfUnsecuredLines - 信用卡余额占比 - age
- 借款人年龄 - NumberOfTime30-59DaysPastDueNotWorse
- 过去有逾期 30-59 天还款的次数 - DebtRatio - 月度生活成本占月收入的比率 - MonthlyIncome
- 月收入 - NumberOfOpenCreditLinesAndLoans
- 包括车贷房贷在内的贷款笔数 - NumberOfTimes90DaysLate
- 过去有逾期 90 天以上还款的次数 - NumberRealEstateLoansOrLines
- 房贷的信贷次数 - NumberOfTime60-89DaysPastDueNotWorse
- 过去有逾期 60-89 天还款的次数 - NumberOfDependents
- 家庭中需要抚养者的人数

3 预处理

[9]:
```
df.shape # 数据的维度
```

[9]:
```
(150000, 11)
```

3.1 数据初探

[10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 11 columns):
 #   Column                                Non-Null Count   Dtype  
---  ------                                --------------   -----  
 0   SeriousDlqin2yrs                      150000 non-null  int64  
 1   RevolvingUtilizationOfUnsecuredLines  150000 non-null  float64
 2   age                                   150000 non-null  int64  
 3   NumberOfTime30-59DaysPastDueNotWorse  150000 non-null  int64  
 4   DebtRatio                             150000 non-null  float64
 5   MonthlyIncome                         120269 non-null  float64
 6   NumberOfOpenCreditLinesAndLoans       150000 non-null  int64  
 7   NumberOfTimes90DaysLate               150000 non-null  int64  
 8   NumberRealEstateLoansOrLines          150000 non-null  int64  
 9   NumberOfTime60-89DaysPastDueNotWorse  150000 non-null  int64  
 10  NumberOfDependents                    146076 non-null  float64
dtypes: float64(4), int64(7)
memory usage: 12.6 MB
```

[11]:
```
df.describe() # 描述性统计
```

[11]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	\
count	150000.000000	150000.000000	150000.000000	
mean	0.066840	6.048438	52.295207	
std	0.249746	249.755371	14.771866	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.029867	41.000000	
50%	0.000000	0.154181	52.000000	
75%	0.000000	0.559046	63.000000	
max	1.000000	50708.000000	109.000000	

	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	\
count	150000.000000	150000.000000	1.202690e+05	
mean	0.421033	353.005076	6.670221e+03	

std	4.192781	2037.818523	1.438467e+04
min	0.000000	0.000000	0.000000e+00
25%	0.000000	0.175074	3.400000e+03
50%	0.000000	0.366508	5.400000e+03
75%	0.000000	0.868254	8.249000e+03
max	98.000000	329664.000000	3.008750e+06

	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	\
count	150000.000000	150000.000000	
mean	8.452760	0.265973	
std	5.145951	4.169304	
min	0.000000	0.000000	
25%	5.000000	0.000000	
50%	8.000000	0.000000	
75%	11.000000	0.000000	
max	58.000000	98.000000	

	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	\
count	150000.000000	150000.000000	
mean	1.018240	0.240387	
std	1.129771	4.155179	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	1.000000	0.000000	
75%	2.000000	0.000000	
max	54.000000	98.000000	

	NumberOfDependents
count	146076.000000
mean	0.757222
std	1.115086
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	20.000000

[12]:
```
# 统计每个类别的数据量
df.SeriousDlqin2yrs.value_counts()
```

[12]:
```
0    139974
1     10026
Name: SeriousDlqin2yrs, dtype: int64
```

[13]:
```
# 计算每个类别的均值
df.SeriousDlqin2yrs.mean()
```

[13]:
```
0.06684
```

[14]:
```
# 计算每个类别的标准差
df.SeriousDlqin2yrs.std()
```

[14]:
```
0.24974553092871982
```

[15]:
```
# 查看特征的取值
df.NumberOfDependents.unique() # 受抚养者，查看有多少取值
```

[15]:
```
array([ 2.,  1.,  0., nan,  3.,  4.,  5.,  6.,  8.,  7., 20., 10.,  9.,
       13.])
```

[16]:
```
# 观察特征取值的频数
df.NumberOfDependents.value_counts()
```

[16]:
```
0.0     86902
1.0     26316
2.0     19522
3.0      9483
4.0      2862
5.0       746
6.0       158
7.0        51
8.0        24
9.0         5
10.0        5
13.0        1
20.0        1
```

```
Name: NumberOfDependents, dtype: int64
```

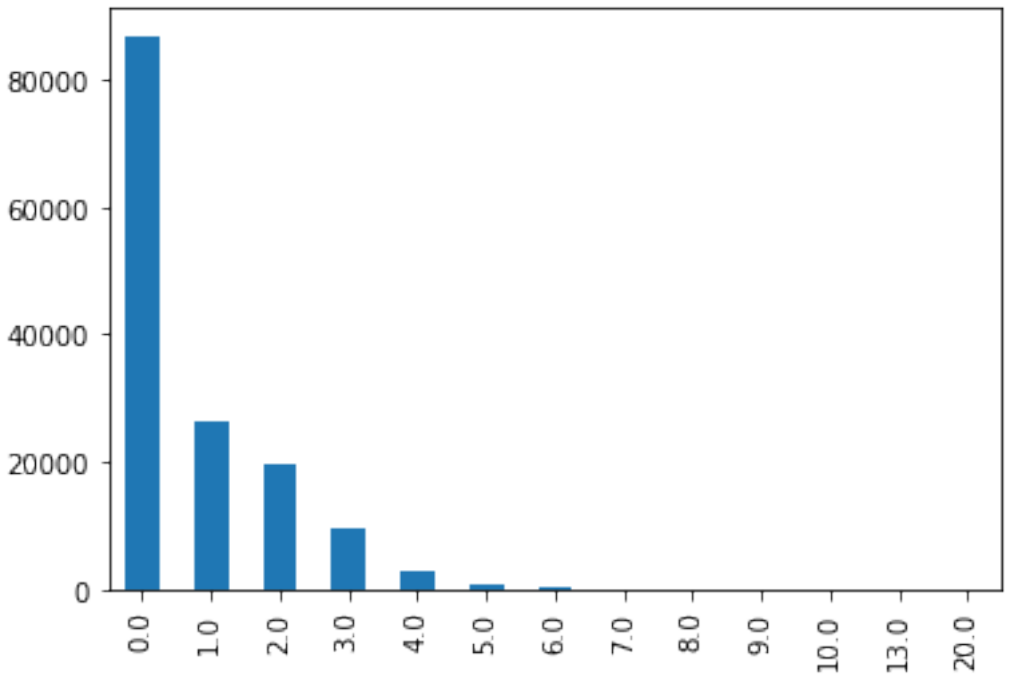
[17]:
```
# 分组求平均值
df.groupby("SeriousDlqin2yrs").mean()
```

[17]:

SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	NumberOfDependents
0	6.168855	52.751375	0.280109	357.151168	6747.837774	8.493620	0.135225	1.020368	0.126666	0.743417
1	4.367282	45.926591	2.388490	295.121066	5630.826493	7.882306	2.091362	0.988530	1.828047	0.948208

[18]:
```
# 按特征取值画出直方图
pd.value_counts(df.NumberOfDependents).plot(kind='bar')
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a74112eac8>
```



[19]:
```
# 计算交叉频数表
pd.crosstab(df.NumberOfTimes90DaysLate, df.SeriousDlqin2yrs)
```

[19]:

SeriousDlqin2yrs	0	1
NumberOfTimes90DaysLate		
0	135108	6554
1	3478	1765
2	779	776
3	282	385
4	96	195
5	48	83
6	32	48
7	7	31
8	6	15
9	5	14
10	3	5
11	2	3
12	1	1
13	2	2

```
14                            1      1
15                            2      0
17                            0      1
96                            1      4
98                          121    143
```

4 处理缺失值

4.1 统计缺失值

[20]:
```
# 计算每个特征缺失值个数
df.isnull().sum()
```

[20]:
```
SeriousDlqin2yrs                        0
RevolvingUtilizationOfUnsecuredLines    0
age                                     0
NumberOfTime30-59DaysPastDueNotWorse    0
DebtRatio                               0
MonthlyIncome                       29731
NumberOfOpenCreditLinesAndLoans         0
NumberOfTimes90DaysLate                 0
NumberRealEstateLoansOrLines            0
NumberOfTime60-89DaysPastDueNotWorse    0
NumberOfDependents                   3924
dtype: int64
```

4.2 删除有缺失值的行

[21]:
```
# axis=0 删除有缺失值的行，axis=1 删除有缺失值的列
# how='all' 只删除全是缺失值的行或列
df.dropna(axis=0)
```

[21]:
```
        SeriousDlqin2yrs  RevolvingUtilizationOfUnsecuredLines  age  \
0                      1                              0.766127   45
1                      0                              0.957151   40
2                      0                              0.658180   38
3                      0                              0.233810   30
```

```
4                       0                                0.907239   49
…                     …                                   …   …
149994                  0                                0.385742   50
149995                  0                                0.040674   74
149996                  0                                0.299745   44
149998                  0                                0.000000   30
149999                  0                                0.850283   64

        NumberOfTime30-59DaysPastDueNotWorse  DebtRatio  MonthlyIncome  \
0                                          2   0.802982         9120.0
1                                          0   0.121876         2600.0
2                                          1   0.085113         3042.0
3                                          0   0.036050         3300.0
4                                          1   0.024926        63588.0
…                                        …         …              …
149994                                     0   0.404293         3400.0
149995                                     0   0.225131         2100.0
149996                                     0   0.716562         5584.0
149998                                     0   0.000000         5716.0
149999                                     0   0.249908         8158.0

        NumberOfOpenCreditLinesAndLoans  NumberOfTimes90DaysLate  \
0                                    13                        0
1                                     4                        0
2                                     2                        1
3                                     5                        0
4                                     7                        0
…                                   …                        …
149994                                7                        0
149995                                4                        0
149996                                4                        0
149998                                4                        0
149999                                8                        0

        NumberRealEstateLoansOrLines  NumberOfTime60-89DaysPastDueNotWorse  \
0                                  6                                     0
```

```
1                                  0                                  0
2                                  0                                  0
3                                  0                                  0
4                                  1                                  0
…                                  …                                  …
149994                             0                                  0
149995                             1                                  0
149996                             1                                  0
149998                             0                                  0
149999                             2                                  0

        NumberOfDependents
0                      2.0
1                      1.0
2                      0.0
3                      0.0
4                      0.0
…                      …
149994                 0.0
149995                 0.0
149996                 2.0
149998                 0.0
149999                 0.0

[120269 rows x 11 columns]
```

4.3 填补缺失值

4.3.1 简单的填补

常数 比较直接的填补方式，就是填补常数。

[22]:
```
# 使用常量填补缺失值
df.loc[df.NumberOfDependents.isnull(),'NumberOfDependents'] = 0
```

平均值 使用最多的是用平均值填充。

[23]:
```
df = pd.read_csv("data/cs-training.csv",usecols=range(1,12))
```

[24]:
```
from sklearn.impute import SimpleImputer

imr = SimpleImputer(missing_values=np.nan, strategy='mean')
imr = imr.fit(df.MonthlyIncome.values.reshape(-1,1))
imputed_data = imr.transform(df.MonthlyIncome.values.reshape(-1,1))
imputed_data
```

[24]:
```
array([[9120.        ],
       [2600.        ],
       [3042.        ],
       …,
       [6670.22123739],
       [5716.        ],
       [8158.        ]])
```

[25]:
```
df.MonthlyIncome = imputed_data
```

[26]:
```
df.describe()
```

[26]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	\
count	150000.000000	150000.000000	150000.000000	
mean	0.066840	6.048438	52.295207	
std	0.249746	249.755371	14.771866	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.029867	41.000000	
50%	0.000000	0.154181	52.000000	
75%	0.000000	0.559046	63.000000	
max	1.000000	50708.000000	109.000000	

	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	\
count	150000.000000	150000.000000	1.500000e+05	
mean	0.421033	353.005076	6.670221e+03	
std	4.192781	2037.818523	1.288045e+04	
min	0.000000	0.000000	0.000000e+00	
25%	0.000000	0.175074	3.903000e+03	
50%	0.000000	0.366508	6.600000e+03	

```
75%                                0.000000       0.868254   7.400000e+03
max                               98.000000  329664.000000   3.008750e+06

       NumberOfOpenCreditLinesAndLoans  NumberOfTimes90DaysLate  \
count                    150000.000000            150000.000000
mean                          8.452760                 0.265973
std                           5.145951                 4.169304
min                           0.000000                 0.000000
25%                           5.000000                 0.000000
50%                           8.000000                 0.000000
75%                          11.000000                 0.000000
max                          58.000000                98.000000

       NumberRealEstateLoansOrLines  NumberOfTime60-89DaysPastDueNotWorse  \
count                 150000.000000                         150000.000000
mean                       1.018240                              0.240387
std                        1.129771                              4.155179
min                        0.000000                              0.000000
25%                        0.000000                              0.000000
50%                        1.000000                              0.000000
75%                        2.000000                              0.000000
max                       54.000000                             98.000000

       NumberOfDependents
count       146076.000000
mean             0.757222
std              1.115086
min              0.000000
25%              0.000000
50%              0.000000
75%              1.000000
max             20.000000
```

[27]:
```
df.isnull().sum()
```

[27]:
```
SeriousDlqin2yrs                        0
RevolvingUtilizationOfUnsecuredLines    0
age                                     0
NumberOfTime30-59DaysPastDueNotWorse    0
DebtRatio                               0
MonthlyIncome                           0
NumberOfOpenCreditLinesAndLoans         0
NumberOfTimes90DaysLate                 0
NumberRealEstateLoansOrLines            0
NumberOfTime60-89DaysPastDueNotWorse    0
NumberOfDependents                   3924
dtype: int64
```

4.3.2 使用插值填补缺失值

[28]:
```
df = pd.read_csv("data/cs-training.csv",usecols=range(1,12))
```

[29]:
```
from sklearn.neighbors import KNeighborsRegressor
income_imputer = KNeighborsRegressor(n_neighbors=1)

# 数据分为两部分，有缺失的和无缺失的，用无缺失的数据建立模型来判断缺失数据的可能取值
train_w_monthly_income = df[df.MonthlyIncome.isnull()==False]
train_w_null_monthly_income = df[df.MonthlyIncome.isnull()==True]
print(train_w_monthly_income.shape)
print(train_w_null_monthly_income.shape)
```

```
(120269, 11)
(29731, 11)
```

[30]:
```
# 用房产贷款次数以及未结束贷款次数来训练月收入
cols = ['NumberRealEstateLoansOrLines', 'NumberOfOpenCreditLinesAndLoans']
income_imputer.fit(train_w_monthly_income[cols], train_w_monthly_income.
 ↪MonthlyIncome)
```

[30]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                    weights='uniform')
```

[31]:
```
# 再用模型预测缺失值中的月收入
new_values = income_imputer.predict(train_w_null_monthly_income[cols])
```

[32]:
```
df.loc[df.MonthlyIncome.isnull(),'MonthlyIncome'] = new_values
```

[33]:
```
# 使用常量填补缺失值
df.loc[df.NumberOfDependents.isnull(),'NumberOfDependents'] = 0
```

[34]:
```
df.info()
```

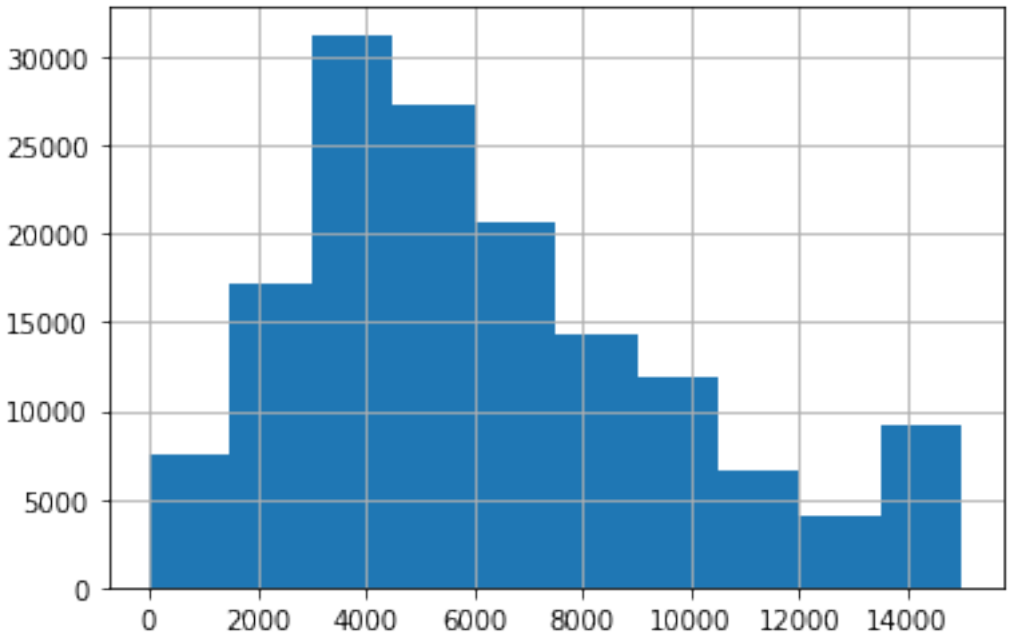
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 11 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   SeriousDlqin2yrs                      150000 non-null  int64
 1   RevolvingUtilizationOfUnsecuredLines  150000 non-null  float64
 2   age                                   150000 non-null  int64
 3   NumberOfTime30-59DaysPastDueNotWorse  150000 non-null  int64
 4   DebtRatio                             150000 non-null  float64
 5   MonthlyIncome                         150000 non-null  float64
 6   NumberOfOpenCreditLinesAndLoans       150000 non-null  int64
 7   NumberOfTimes90DaysLate               150000 non-null  int64
 8   NumberRealEstateLoansOrLines          150000 non-null  int64
 9   NumberOfTime60-89DaysPastDueNotWorse  150000 non-null  int64
 10  NumberOfDependents                    150000 non-null  float64
dtypes: float64(4), int64(7)
memory usage: 12.6 MB
```

4.4 变量离散化

[35]:
```
def cap_values(x, cap):
    if x > cap:
        return cap
    else:
        return x
# 设定上限
df.MonthlyIncome = df.MonthlyIncome.apply(lambda x: cap_values(x, 15000))
```

[36]:
```
df.MonthlyIncome.hist()
```

[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a7418f7448>
```



4.4.1 指定分组个数

[37]:
```
# 变量离散化，分为 15 个 bin
df['income_bins'] = pd.cut(df.MonthlyIncome, bins=15, labels=False)
pd.value_counts(df.income_bins)
```

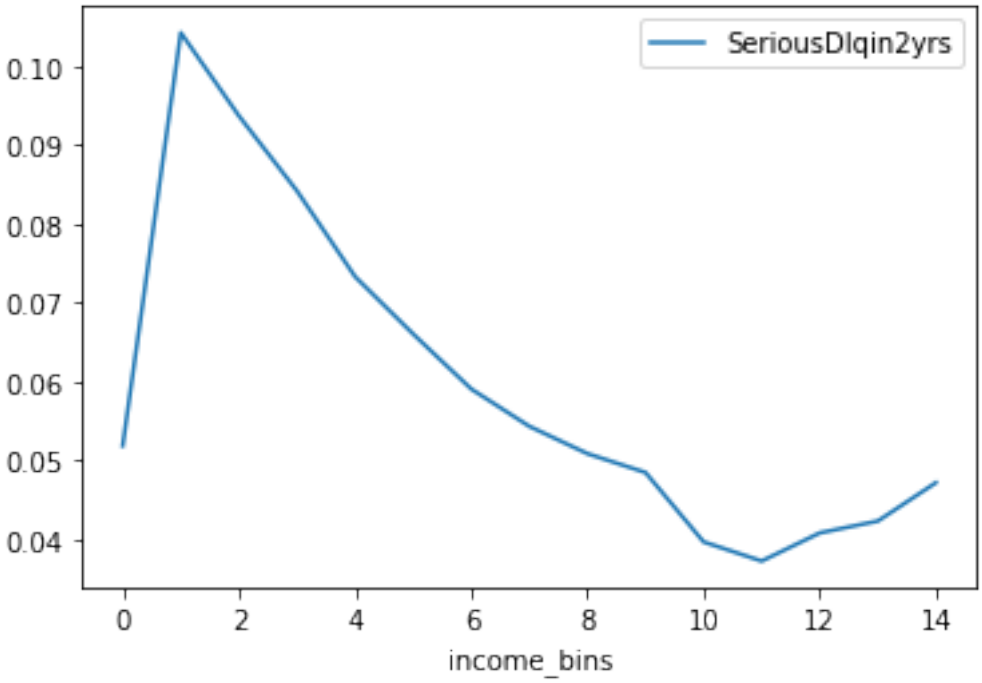
[37]:
```
3     23168
4     19944
```

```
5     15583
6     14475
2     14038
7     10766
8      8609
14     7775
9      7672
1      7504
10     6298
0      4994
11     4454
12     2547
13     2173
Name: income_bins, dtype: int64
```

[38]:
```
df[["income_bins", "SeriousDlqin2yrs"]].groupby("income_bins").mean()
# 每个月收入分类中统计拖欠的平均频数
```

[38]:
```
             SeriousDlqin2yrs
income_bins
0                    0.051862
1                    0.104211
2                    0.093674
3                    0.084168
4                    0.073305
5                    0.066033
6                    0.059067
7                    0.054338
8                    0.050877
9                    0.048488
10                   0.039695
11                   0.037270
12                   0.040832
13                   0.042338
14                   0.047203
```

[39]:
```
# 画出图来，可以明显发现在 1-2 个 bin 中的 拖欠 最多
cols = ["income_bins", "SeriousDlqin2yrs"]
df[cols].groupby("income_bins").mean().plot();
```



4.4.2 自定义分组

[40]:
```
# 自定义 bin
mybins = [0] + list(range(20,80,5)) + [120]
df['age_bin'] = pd.cut(df.age, bins=mybins)
pd.value_counts(df['age_bin'])
```

[40]:
```
(45, 50]     18829
(50, 55]     17861
(55, 60]     16945
(60, 65]     16461
(40, 45]     16208
(35, 40]     13611
(65, 70]     10963
(30, 35]     10728
```

```
(75, 120]    10129
(25, 30]      7730
(70, 75]      7507
(20, 25]      3027
(0, 20]          0
Name: age_bin, dtype: int64
```

4.5 数据标准化

[41]:
```
# 标准化数据
from sklearn.preprocessing import StandardScaler

# 将月收入标准化
df['monthly_income_scaled'] = StandardScaler().fit_transform(np.array(df.
↪MonthlyIncome).reshape(-1, 1))
```

[42]:
```
df.columns
```

[42]:
```
Index(['SeriousDlqin2yrs', 'RevolvingUtilizationOfUnsecuredLines', 'age',
       'NumberOfTime30-59DaysPastDueNotWorse', 'DebtRatio', 'MonthlyIncome',
       'NumberOfOpenCreditLinesAndLoans', 'NumberOfTimes90DaysLate',
       'NumberRealEstateLoansOrLines', 'NumberOfTime60-89DaysPastDueNotWorse',
       'NumberOfDependents', 'income_bins', 'age_bin',
       'monthly_income_scaled'],
      dtype='object')
```

[43]:
```
# 特征
features = ['SeriousDlqin2yrs',
            'RevolvingUtilizationOfUnsecuredLines',
            'NumberOfTime30-59DaysPastDueNotWorse',
            'DebtRatio',
            'NumberOfOpenCreditLinesAndLoans',
            'NumberOfTimes90DaysLate',
            'NumberRealEstateLoansOrLines',
            'NumberOfTime60-89DaysPastDueNotWorse',
            'NumberOfDependents',
            'income_bins',
```

```
            'age_bin',
        ]
```

[44]:
```
# 进行 one-hot 编码
X = pd.get_dummies(df[features], columns = ['income_bins', 'age_bin'])
```

5 分割数据集

[45]:
```
from sklearn.model_selection import train_test_split
X,y = X.iloc[:,1:].values, X.iloc[:,0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=0)
```

5.1 层次分割

[46]:
```
import numpy as np
# 帮助函数，计算各标签比例
def label_frequency(labels):
    counts = np.unique(labels, return_counts=True)[1]
    n = len(labels)
    return counts / float(n)
```

[47]:
```
# 原始数据中各标签的比例
label_frequency(y)
```

[47]:
```
array([0.93316, 0.06684])
```

[48]:
```
# train_test_split 后的比例
label_frequency(y_train), label_frequency(y_test)
```

[48]:
```
(array([0.93341905, 0.06658095]), array([0.93255556, 0.06744444]))
```

[49]:
```
# stratified 之后的标签比例， 更接近原始比例
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
↪test_size=0.3, random_state=0)
label_frequency(y_train), label_frequency(y_test)
```

```
[49]: (array([0.9331619, 0.0668381]), array([0.93315556, 0.06684444]))
```

6 特征选择（Feature Selection）

通常，我们会收集很多和预测目标相关的特征，但是我们很难确定哪些特征会真正对目标的预测产生影响。为了提高可解释性，或者说模型的泛化能力，我们通常会从原始特征选择一组子集。好的特征选择能够提升模型的性能，更能帮助我们理解数据的特点、底层结构，对进一步改善模型、算法都有着重要作用。

特征选择主要有两个功能：

- 减少特征数量、降维，使模型泛化能力更强，减少过拟合
- 增强对特征和特征值之间的理解

特征选择的方法分为三类: + Filter 方法：采用某种度量准则对于特征进行筛选。计算特征的一些属性，例如，方差、距离、可分性度量、相关性度量、信息论度量，然后进行排序，选择特征权重最高的字段作为特征。

- Wrapper 方法：通过在选择过程中，添加或者删除特定特征，验证模型效果，从而找到最优的特征组合。Wrapper 方法类似一套搜索算法，将特征作为输入，模型性能作为输出，搜索最优的组合。常见的方法有稳定性选择 (Stability Selection) 和递归特征消除 (Recursive Feature Elimination) 两种方式。
- Embedded 方法：嵌入式特征选择是将特征选择过程与学习器训练过程融为一体，两者在同一个优化过程中完成。例如利用回归模型的系数选择特征，基于树模型的特征选择（随机森林的平均不纯度减少（mean decrease impority）和平均精确率减少（Mean Decrease Accuracy））。

6.1 特征权重排序算法（filter）

6.1.1 单变量特征选择

单变量特征选择能够对每一个特征进行测试，衡量该特征和因变量之间的关系，根据得分去掉不好的特征。

sklearn 提供了 2 个实现单变量特征选择的函数：* SelectKBest(score_func=chi2,k=10) 此函数表示基于卡方检验选择前 10 个最好的特征 - socre_func: 用于回归的 f_regression、mutual_info_regression，用于分类的 chi2、f_classif、mutual_info_classif。* SelectPercentile 选择结果最好的前 x% 的特征 * 对每个特征进行检验: false positive rate SelectFpr, false discovery

rate SelectFdr, or family wise error SelectFwe. GenericUnivariateSelect 可以利用定义的策略进行单变量特征选择。可以利用超参数搜索的方式进行单变量选择。

[50]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
# 读取 wine 数据
df_wine = pd.read_csv('data/wine.data', header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
'Alcalinity of ash', 'Magnesium', 'Total phenols',
'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']

print('Class labels', np.unique(df_wine['Class label']))
df_wine.head()
```

```
Class labels [1 2 3]
```

[50]:

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	\
0	1	14.23	1.71	2.43	15.6	127	
1	1	13.20	1.78	2.14	11.2	100	
2	1	13.16	2.36	2.67	18.6	101	
3	1	14.37	1.95	2.50	16.8	113	
4	1	13.24	2.59	2.87	21.0	118	

	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	\
0	2.80	3.06	0.28	2.29	
1	2.65	2.76	0.26	1.28	
2	2.80	3.24	0.30	2.81	
3	3.85	3.49	0.24	2.18	
4	2.80	2.69	0.39	1.82	

	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	5.64	1.04	3.92	1065
1	4.38	1.05	3.40	1050
2	5.68	1.03	3.17	1185

```
3              7.80  0.86                          3.45      1480
4              4.32  1.04                          2.93       735
```

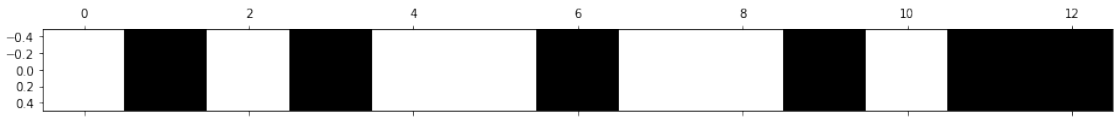
[51]:
```
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)
```

[52]:
```
# 以 chi2 结合 SelectKbest 为例
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest

select = SelectKBest(chi2, k=6) # 选出前 6
X_uni_selected = select.fit_transform(X_train, y_train)

print(X_train.shape)
print(X_uni_selected.shape)
```

```
(124, 13)
(124, 6)
```

[53]:
```
import matplotlib.pyplot as plt
%matplotlib inline
# 查看选出了哪几个 feature, 黑色是选出来的
mask = select.get_support()
print(mask)
# visualize the mask. black is True, white is False
plt.matshow(mask.reshape(1, -1), cmap='gray_r');
```

```
[False  True False  True False False  True False False  True False  True
  True]
```



6.2 wrapper 方法

6.2.1 递归特征消除（RFE）

递归特征消除的主要思想是，反复构建模型，然后选出其中贡献最差的特征，把选出的特征剔除，然后在剩余特征上继续重复这个过程，直到所有特征都已遍历。这个基模型需要一个 coef_ 属性，这个属性是用来描述特征的权重也就是其重要性，这样在每一轮训练过程中消除掉一些权重较小的特征直至训练结束。

[54]:
```
from sklearn.feature_selection import RFE
from sklearn.svm import SVC

svc = SVC(kernel="linear", C=1)
rfe = RFE(estimator=svc,
          n_features_to_select=6,  # 要选出几个 feature
          step=1)  # 每次剔除出几个 feature
rfe.fit(X_train, y_train)

X_rfe_selected = rfe.transform(X_train)
```

[55]:
```
# 查看选出了哪几个 feature
mask = rfe.get_support()
print(mask)
plt.matshow(mask.reshape(1, -1), cmap='gray_r');
```

```
[ True False  True False False  True  True False False  True False  True
 False]
```

6.3 嵌入式方法（embedded）

SelectFromModel 是一个元转换器（meta-transformer），可以与任意拟合后具有 coef_ 或者 feature_importances_ 属性的模型结合使用。如果 coef_ 或者 feature_importances_ 属性小于某个阈值，那么该特征可以认为是不重要的可以删除的。除了指定数值的阈值，函数具有内建的阈

值计算机制，可以通过字符串参数指定。比如 “mean”，“median” 等。

6.3.1 基于惩罚项

使用带惩罚项的基模型，除了筛选出特征外，同时也进行了降维。例如使用结合 L1 惩罚项的逻辑回归模型。

[56]:
```
from sklearn.linear_model import LogisticRegression
# sklearn 里想用 L1 正则，把 penalty 参数设为 'l1' 即可
lr = LogisticRegression(penalty='l1', C=0.1, solver="liblinear")
lr.fit(X_train, y_train)
print('Training accuracy:', lr.score(X_train, y_train))
print('Test accuracy:', lr.score(X_test, y_test))
```

```
Training accuracy: 0.9516129032258065
Test accuracy: 0.9259259259259259
```

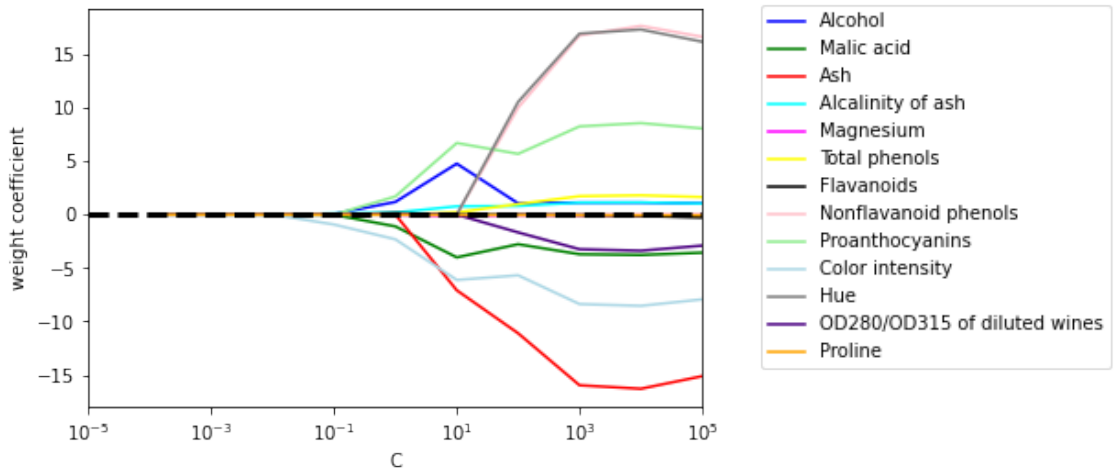
[57]:
```
# 用了 One-vs-Rest (OvR) 方法，所以会出现三行系数
lr.coef_
```

[57]:
```
array([[ 0.        ,  0.        ,  0.        , -0.39570184, -0.04408795,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.01442927],
       [ 0.06124419, -0.01832722,  0.        ,  0.11947868,  0.04320337,
         0.        ,  0.        ,  0.        ,  0.        , -0.9234132 ,
         0.        ,  0.        , -0.00570324],
       [ 0.        ,  0.        ,  0.        ,  0.        , -0.00172235,
         0.        , -1.1460492 ,  0.        ,  0.        ,  0.68198072,
         0.        ,  0.        , -0.00324442]])
```

可以看出系数矩阵是稀疏的 (只有少数非零系数)

[58]:
```
# 不同特征在不同正则强度下的系数反映
import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure()
ax = plt.subplot(111)
```

```
colors = ['blue', 'green', 'red', 'cyan',
          'magenta', 'yellow', 'black',
           'pink', 'lightgreen', 'lightblue',
           'gray', 'indigo', 'orange']

weights, params = [], []
for c in range(-4, 6):
    lr = LogisticRegression(penalty='l1', C=10**c, solver =␣
 ↪'liblinear',multi_class='auto',random_state=0)
    lr.fit(X_train, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)

for column, color in zip(range(weights.shape[1]), colors):
    plt.plot(params, weights[:, column],
             label=df_wine.columns[column+1],
             color=color)
plt.axhline(0, color='black', linestyle='--', linewidth=3)
plt.xlim([10**(-5), 10**5])
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.xscale('log')
plt.legend(loc='upper left')
ax.legend(loc='upper center',
          bbox_to_anchor=(1.38, 1.03),
          ncol=1, fancybox=True);
# plt.savefig('./figures/l1_path.png', dpi=300)
```



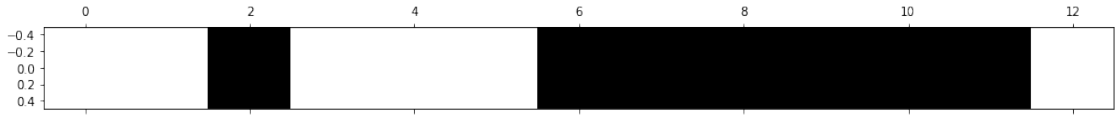
随着 L1 正则项增大，无关特征被排除出模型 (系数变为 0)，因此 L1 正则可以作为特征选择的一种方法。

结合 sklearn 的 SelectFromModel 进行选择

[59]:
```
from sklearn.feature_selection import SelectFromModel

model_l1 = SelectFromModel(lr, threshold='median', prefit=True)
X_l1_selected = model_l1.transform(X)
```

[60]:
```
# 查看选出了哪几个 feature, 黑色是选出来的
mask = model_l1.get_support()
print(mask)
plt.matshow(mask.reshape(1, -1), cmap='gray_r');
```

```
[False False  True False False False  True  True  True  True  True  True
 False]
```



6.3.2 基于树的特征选择

随机森林算法可以测量各个特征的重要性，因此可以作为特征选择的一种手段。随机森林提供了两种特征选择的方法：平均不纯度减少 mean decrease impurity 和平均精确率减少 mean decrease accuracy。

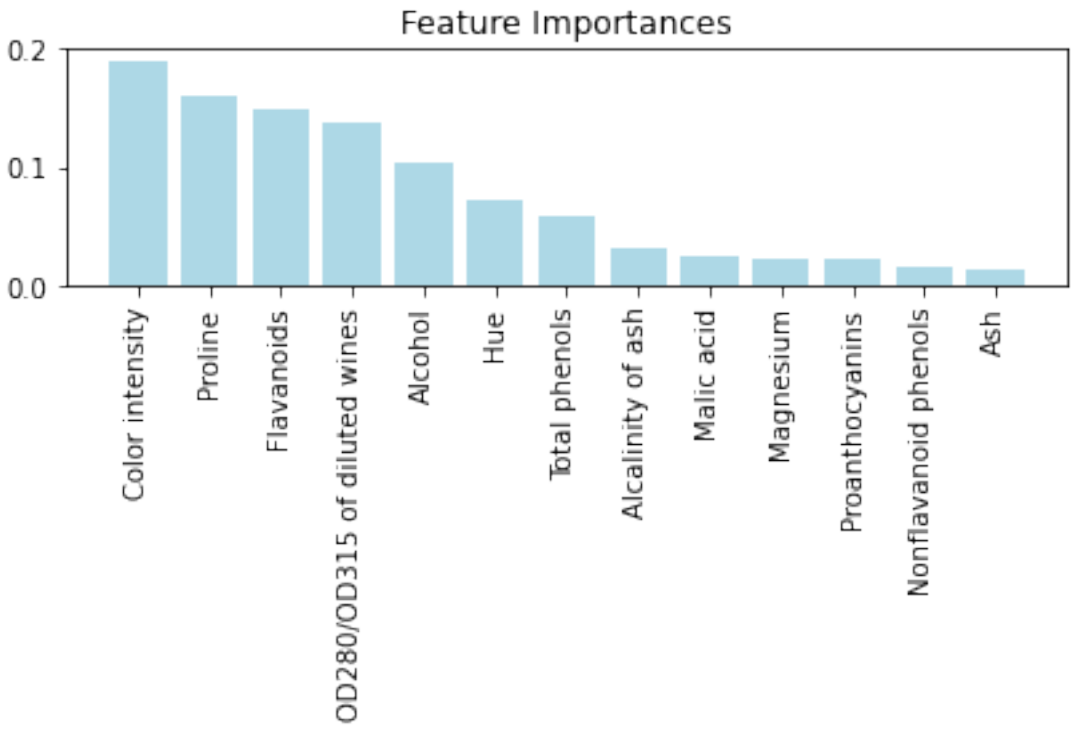
[61]:
```
from sklearn.ensemble import RandomForestClassifier

feat_labels = df_wine.columns[1:]

# 使用 decision tree 或 random forests 不需要 standardization 或 normalization
forest = RandomForestClassifier(n_estimators=1000,
                                random_state=0,
                                n_jobs=-1)
forest.fit(X_train, y_train)

# random forest 比较特殊，有 feature_importances 这个 attribute
importances = forest.feature_importances_

indices = np.argsort(importances)[::-1]
```

[62]:
```
for i, idx in enumerate(indices):
    print("%2d) %-*s %f" % (i + 1, 30,
                            feat_labels[idx],
                            importances[idx]))
```

```
 1) Color intensity                0.190583
 2) Proline                        0.159246
 3) Flavanoids                     0.149066
 4) OD280/OD315 of diluted wines   0.137214
 5) Alcohol                        0.103509
 6) Hue                            0.071931
 7) Total phenols                  0.059530
 8) Alcalinity of ash              0.032480
 9) Malic acid                     0.023645
10) Magnesium                      0.022201
11) Proanthocyanins                0.021967
12) Nonflavanoid phenols           0.015877
```

```
13) Ash                           0.012752
```

[63]:
```
plt.title('Feature Importances')
plt.bar(range(X_train.shape[1]),
        importances[indices],
        color='lightblue',
        align='center')

plt.xticks(range(X_train.shape[1]),
           feat_labels[indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.tight_layout()
#plt.savefig('./random_forest.png', dpi=300)
```



[64]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

select_rf = SelectFromModel(forest, threshold=0.1, prefit=True)
```

```
X_train_rf = select_rf.transform(X_train)

print(X_train.shape[1])  # 原始特征维度
print(X_train_rf.shape[1])  # 特征选择后特征维度
```

```
13
5
```

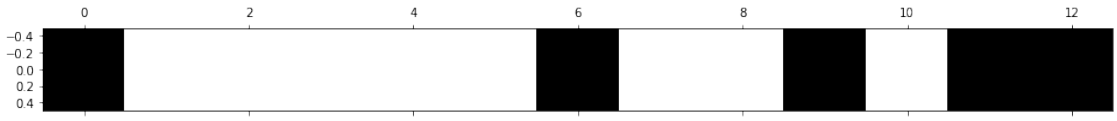
[65]:
```
# 查看选出的特征
mask = select_rf.get_support()
for f in feat_labels[mask]:
    print(f)
```

```
Alcohol
Flavanoids
Color intensity
OD280/OD315 of diluted wines
Proline
```

[66]:
```
# 可视化特征选择结果，黑色的是选中的，白色的是滤过的
mask = select_rf.get_support()
print(mask)
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
```

```
[ True False False False False False  True False False  True False  True
  True]
```

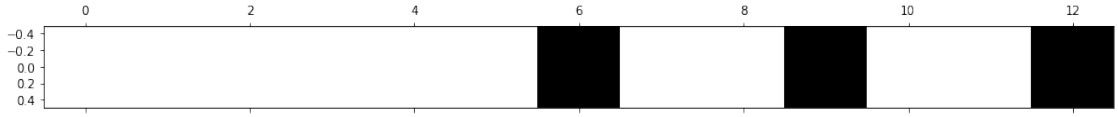
[66]: <matplotlib.image.AxesImage at 0x1a7455c0548>



也能将随机森林和 Sequential selection （递归特征清除）结合起来。

[67]:
```
from sklearn.feature_selection import RFE
select = RFE(RandomForestClassifier(n_estimators=100, random_state=0),
             n_features_to_select=3)

select.fit(X_train, y_train)
# visualize the selected features:
mask = select.get_support()
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
```

[67]: <matplotlib.image.AxesImage at 0x1a7456d8b88>



7 特征提取（Feature Extraction）

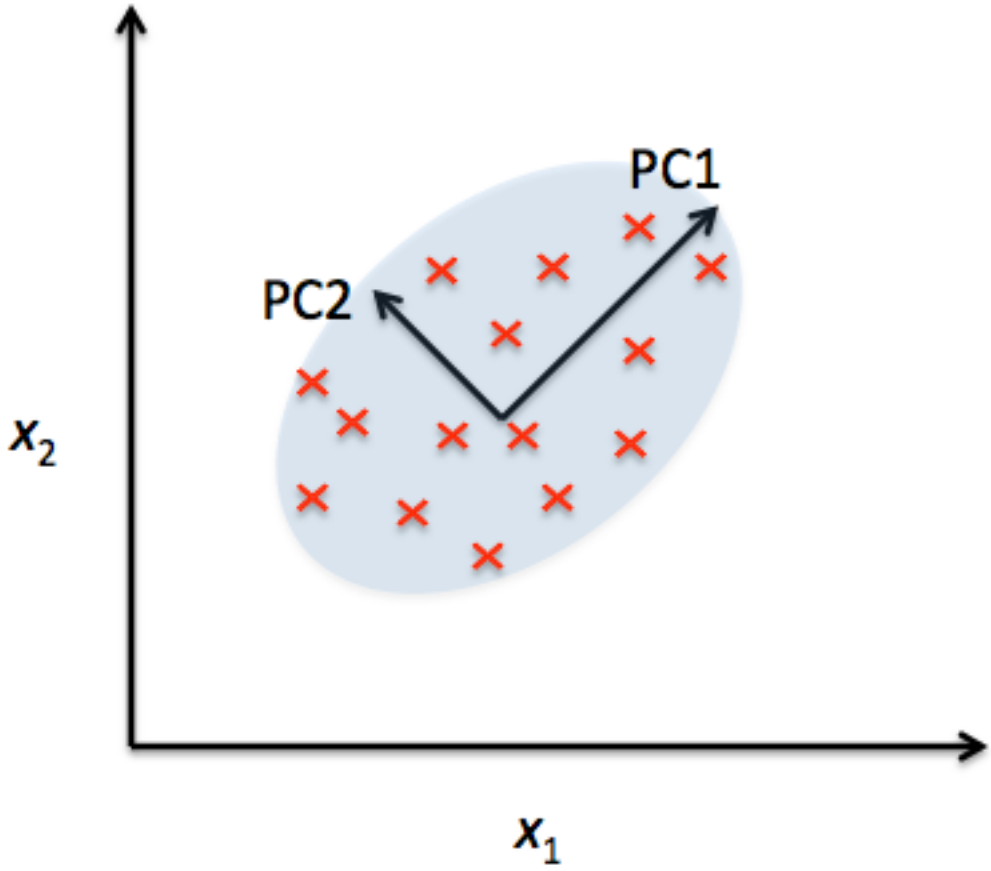
降维是一种对高维度特征数据预处理方法。降维是将高维度的数据保留下最重要的一些特征，去除噪声和不重要的特征，从而实现提升数据处理速度的目的。

7.1 主成分分析（PCA）

PCA(Principal Component Analysis)，即主成分分析方法，是一种使用最广泛的数据降维算法。PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。PCA 的工作就是从原始的空间中顺序地找一组相互正交的坐标轴，新的坐标轴的选择与数据本身是密切相关的。其中，第一个新坐标轴选择是原始数据中方差最大的方向，第二个新坐标轴选取是与第一个坐标轴正交的平面中使得方差最大的，第三个轴是与第 1,2 个轴正交的平面中方差最大的。依次类推，可以得到 n 个这样的坐标轴。通过这种方式获得的新的坐标轴，我们发现，大部分方差都包含在前面 k 个坐标轴中，后面的坐标轴所含的方差几乎为 0。于是，我们可以忽略余下的坐标轴，只保留前面 k 个含有绝大部分方差的坐标轴。事实上，这相当于只保留包含绝大部分方差的维度特征，而忽略包含方差几乎为 0 的特征维度，实现对数据特征的降维处理。

PCA 算法步骤：1. 标准化 d-维数据集; 2. 构建协方差阵（covariance matrix）; 3. 将协方差阵分解为它的特征值和特征向量; 4. 针对最大的 k 个特征值，选择对应的特征向量，k 则为新的特征空

间的维度 ($k \leq d$); 5. 利用最大的 k 个特征向量构建一个投影矩阵（projection matrix）; 6. 利用投影矩阵，将 d 维空间的数据集 X，变成 k 维空间的数据.



[68]:
```
import pandas as pd

df_wine = pd.read_csv('data/wine.data', header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
'Alcalinity of ash', 'Magnesium', 'Total phenols',
'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']

df_wine.head()
```

[68]:

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium
0	1	14.23	1.71	2.43	15.6	127
1	1	13.20	1.78	2.14	11.2	100
2	1	13.16	2.36	2.67	18.6	101
3	1	14.37	1.95	2.50	16.8	113
4	1	13.24	2.59	2.87	21.0	118

	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins
0	2.80	3.06	0.28	2.29
1	2.65	2.76	0.26	1.28
2	2.80	3.24	0.30	2.81
3	3.85	3.49	0.24	2.18
4	2.80	2.69	0.39	1.82

	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	5.64	1.04	3.92	1065
1	4.38	1.05	3.40	1050
2	5.68	1.03	3.17	1185
3	7.80	0.86	3.45	1480
4	4.32	1.04	2.93	735

[69]:
```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)
```

[70]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

7.2 协方差矩阵

协方差设随机变量 X 与 Y 的方差都存在, 则称 X 的离差 $X-E(X)$ 与 Y 的离差 $Y-E(Y)$ 乘积的数学期望为 X 与 Y 的协方差, 记为

$$\mathrm{cov}(X,Y)=E[(X-E(X))(Y-E(Y))]$$

特别地,$\mathrm{cov}(X,X)=D(X)$.

协方差矩阵设 $(X_1,X_2,\cdots,X_n)$ 是 n 维随机向量, 且每个分量 $X_i(i=1,2,\cdots,n)$ 的方差都存在, 则称矩阵

$$V=\begin{bmatrix}\mathrm{cov}(X_1,X_1) & \mathrm{cov}(X_1,X_2) & \cdots & \mathrm{cov}(X_1,X_n)\\ \mathrm{cov}(X_2,X_1) & \mathrm{cov}(X_2,X_2) & \cdots & \mathrm{cov}(X_2,X_n)\\ \vdots & \vdots & \ddots & \vdots\\ \mathrm{cov}(X_n,X_1) & \mathrm{cov}(X_n,X_2) & \cdots & \mathrm{cov}(X_n,X_n)\end{bmatrix}$$

n 维随机向量 $(X_1,X_2,\cdots,X_n)$ 的协方差矩阵.

样本的协方差矩阵

$$\sigma_{jk}=\frac{1}{n}\sum_{i=1}^{n}(x_j^{(i)}-\mu_j)(x_k^{(i)}-\mu_k)$$

7.3 计算特征值和特征向量

[71]:
```
import numpy as np
# 计算协方差矩阵
cov_mat = np.cov(X_train_std.T)

# 获取 eigenvalues 和 eigenvectors
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)

print('Eigenvalues \n %s' % eigen_vals)
```

```
Eigenvalues
 [4.8923083  2.46635032 1.42809973 1.01233462 0.84906459 0.60181514
 0.52251546 0.08414846 0.33051429 0.29595018 0.16831254 0.21432212
 0.2399553 ]
```

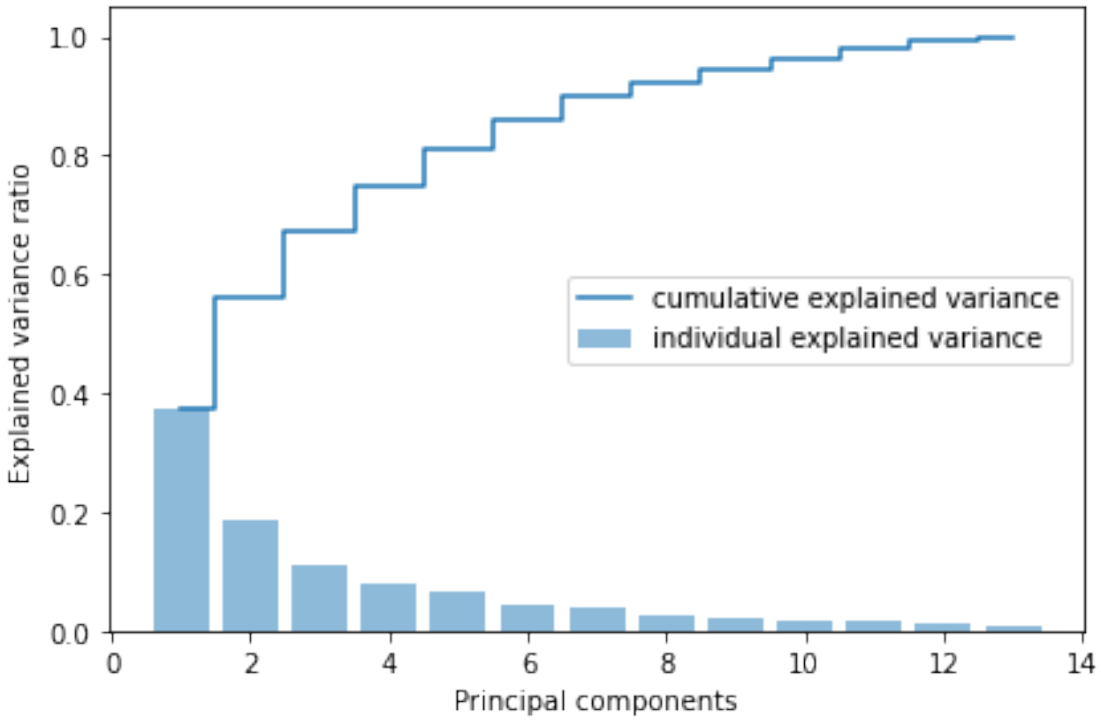
7.4 方差解释率

一个特征值的**方差解释率**为该特征值与特征值总和之比：

$$\frac{\lambda_j}{\sum_{i=1}^{d} \lambda_i}$$

[72]:
```
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)  # 解释 variance 的累积和
```

[73]:
```
# 按照方差解释率进行绘图
import matplotlib.pyplot as plt
%matplotlib inline

plt.bar(range(1, 14), var_exp, alpha=0.5, align='center',
        label='individual explained variance')
plt.step(range(1, 14), cum_var_exp, where='mid',
         label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('./figures/pca1.png', dpi=300)
```



第一个分量能解释将近 40% 的 variance, 前两个分量能解释近 60%。

7.5 特征变形

[74]:
```
# 构造（eigenvalue, eigenvector）元组
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in␣
 ↪range(len(eigen_vals))]

# 从高到低进行排序
eigen_pairs.sort(reverse=True)
```

为了方便，我们这里只取两个特征向量，后面的画图会方便观察。实际中，主成分的数量由计算效率和分类器的性能权衡决定。

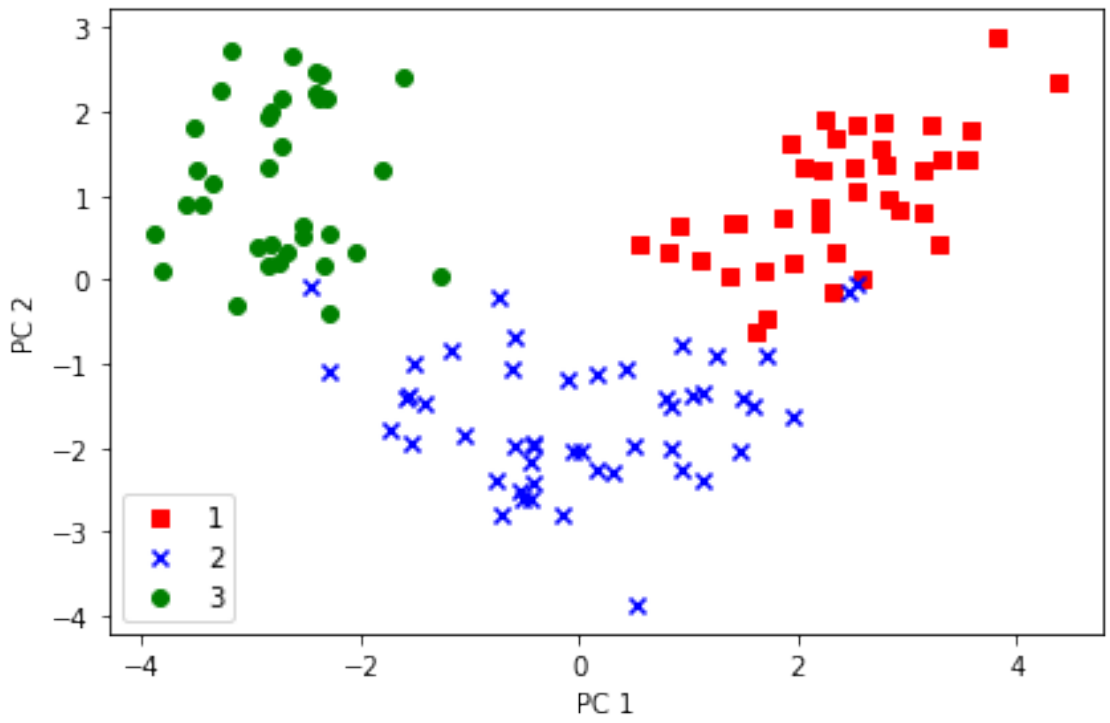
[75]:
```
# 由最高的两个特征向量构成 13*2 的投影矩阵
w = np.column_stack([eigen_pairs[0][1], eigen_pairs[1][1]])
print(w)
```

```
[[ 0.14669811  0.50417079]
 [-0.24224554  0.24216889]
 [-0.02993442  0.28698484]
 [-0.25519002 -0.06468718]
 [ 0.12079772  0.22995385]
 [ 0.38934455  0.09363991]
 [ 0.42326486  0.01088622]
 [-0.30634956  0.01870216]
 [ 0.30572219  0.03040352]
 [-0.09869191  0.54527081]
 [ 0.30032535 -0.27924322]
 [ 0.36821154 -0.174365  ]
 [ 0.29259713  0.36315461]]
```

利用投影矩阵 W, 我们可以得到转换后的数据 x'

$$x' = xW$$

[76]:
```
# 124×13 维的全体数据，投影成为两个主成分的数据
X_train_pca = X_train_std.dot(w)
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']

for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train==l, 0],
                X_train_pca[y_train==l, 1],
                c=c, label=l, marker=m)

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('./figures/pca2.png', dpi=300)
```

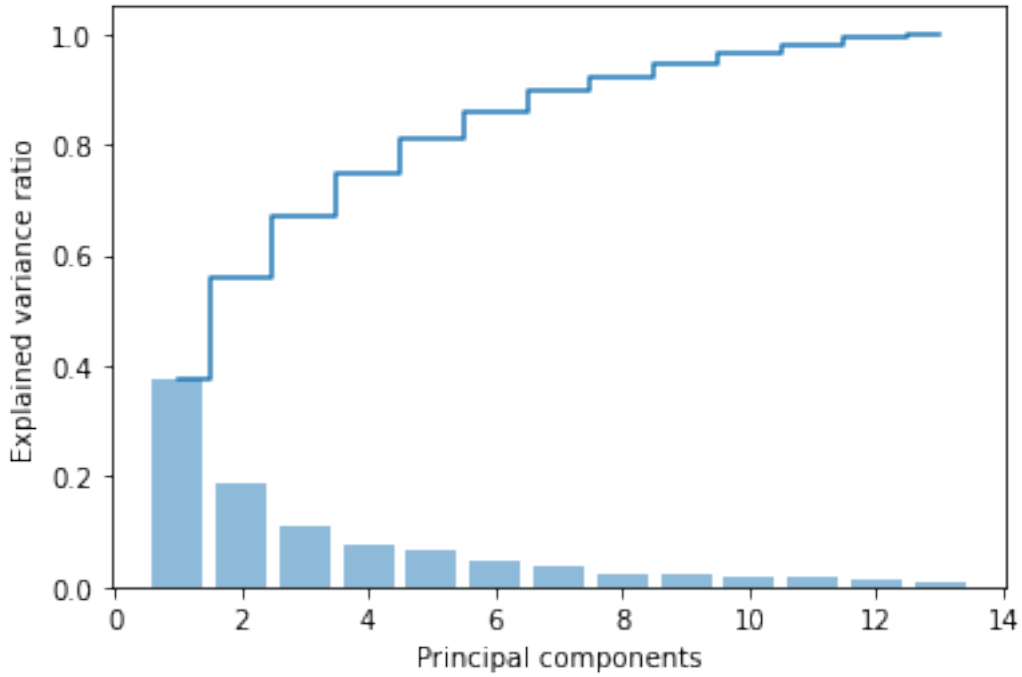


7.6 在 scikit-learn 中使用主成分分析

[77]:
```
from sklearn.decomposition import PCA

pca = PCA()
X_train_pca = pca.fit_transform(X_train_std)
pca.explained_variance_ratio_
```

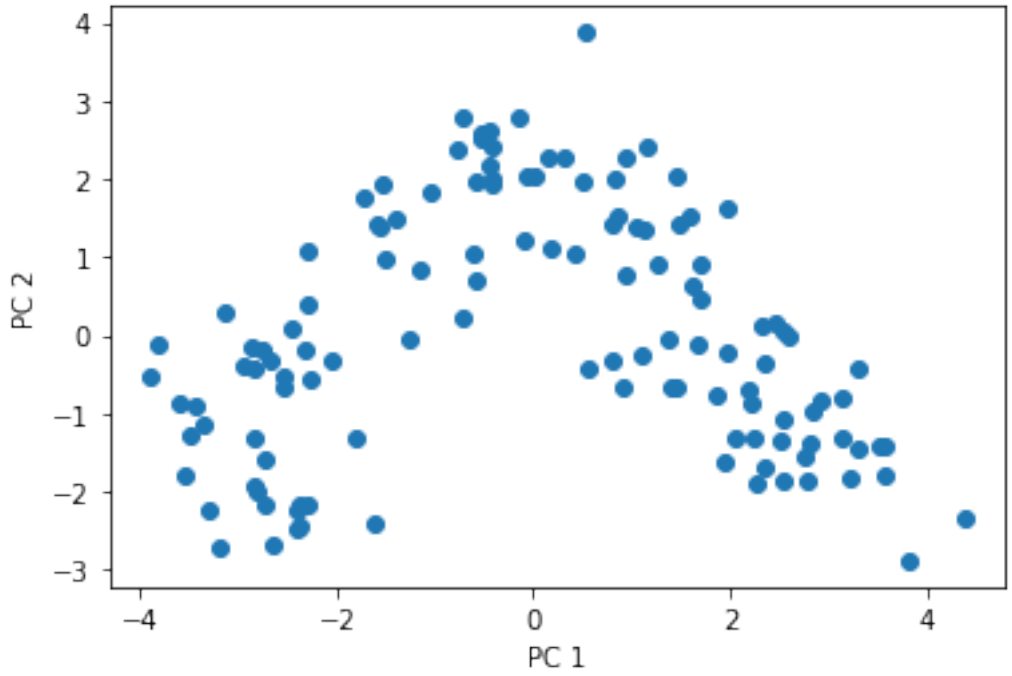
[77]:
```
array([0.37329648, 0.18818926, 0.10896791, 0.07724389, 0.06478595,
       0.04592014, 0.03986936, 0.02521914, 0.02258181, 0.01830924,
       0.01635336, 0.01284271, 0.00642076])
```

[78]:
```
plt.bar(range(1, 14), pca.explained_variance_ratio_, alpha=0.5, align='center')
plt.step(range(1, 14), np.cumsum(pca.explained_variance_ratio_), where='mid')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components');
```

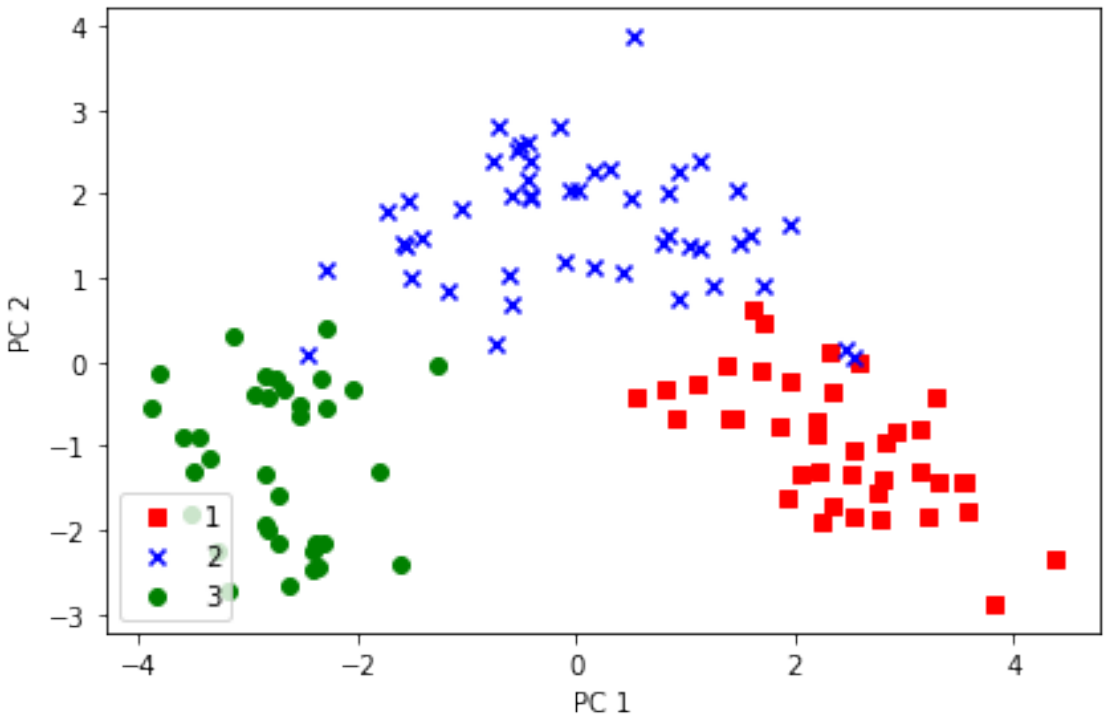


[79]:
```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
```

[80]:
```
plt.scatter(X_train_pca[:,0], X_train_pca[:,1])
plt.xlabel('PC 1')
plt.ylabel('PC 2');
```



[81]:
```
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']

for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train==l, 0],
                X_train_pca[y_train==l, 1],
                c=c, label=l, marker=m)

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('./figures/pca2.png', dpi=300)
```



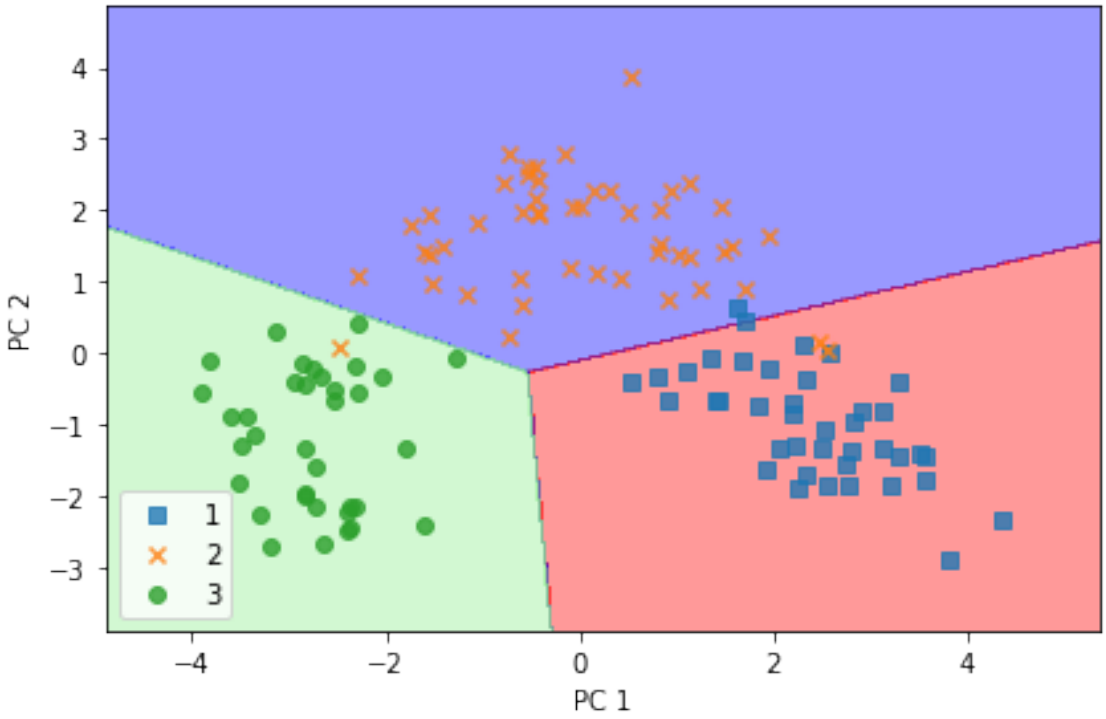
[82]:
```
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
```

```
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8,
                    marker=markers[idx], label=cl)
```

使用前 2 个主成分训练逻辑回归分类器

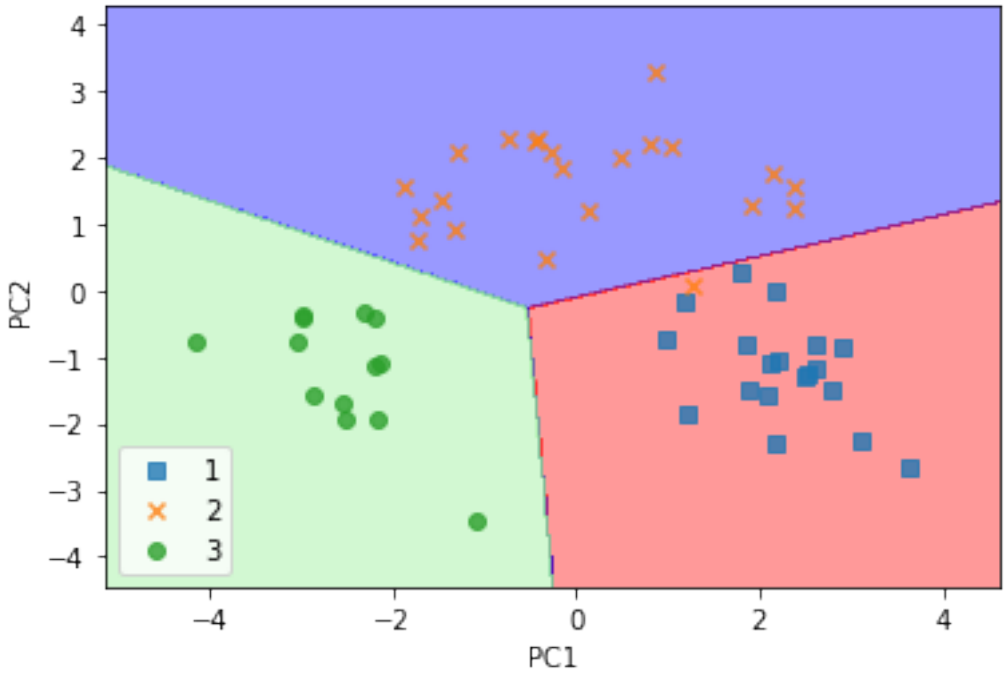
[83]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(multi_class='auto')
lr = lr.fit(X_train_pca, y_train)
```

[84]:
```
plot_decision_regions(X_train_pca, y_train, classifier=lr)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout();
# plt.savefig('./figures/pca3.png', dpi=300)
```



[85]:
```
# 在测试集上测试
plot_decision_regions(X_test_pca, y_test, classifier=lr)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend(loc='lower left')
```

[85]: <matplotlib.legend.Legend at 0x1a745c354c8>


4
3
2
1
0
−1
−2
−3
−4
PC2
−4
−2
0
2
4
PC1
1
2
3