

Python 数据分析实践 (Data Analysis Action)

Chap 11 图像颜色数据分析和识别

内容：

- 分类算法
- 分类模型的性能比较
- 图像数据的预处理
- 算法：支持向量机（Support Vector Machine, SVM），k-最近邻（k-Nearest Neighbor, kNN），决策树（Decision Tree, DT）算法和不同算法的性能比较
- 应用领域：水质环境监测，流行色预测，等

实践：

- 数据探索、缺失处理、
- 数据变换、数据归一化
- 图像预处理
- 分类算法（Scikit中的SVM, kNN和DT）和算法比较

实例：

- 实例1：基于水色图像的水质评价（Python数据分析与挖掘实战 第九章）
 - 实例2：国际服装流行色预测
-

这节课是在前面数据分析的基础上，对图像数据进行预处理和分类建模。针对图像类型的数据，必须进行图像的预处理，将不同格式类型的图像数据转换为能够被分类算法处理的数据类型，然后通过有监督的分类算法进行建模和评估，并应用分类模型对新的未知图像进行分类预测。本节课通过颜色图像实例来进行图像的实践分析和分类，也适用于不同行业的多种图像类型数据。注意，本节课中未涉及更多更精确的图像处理操作，具体细节如果有兴趣，可以选读图像处理相关课程。

准备工作：导入库，配置环境等

In [1]:

```
from __future__ import division
import os, sys

# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw
import codecs
import csv
```

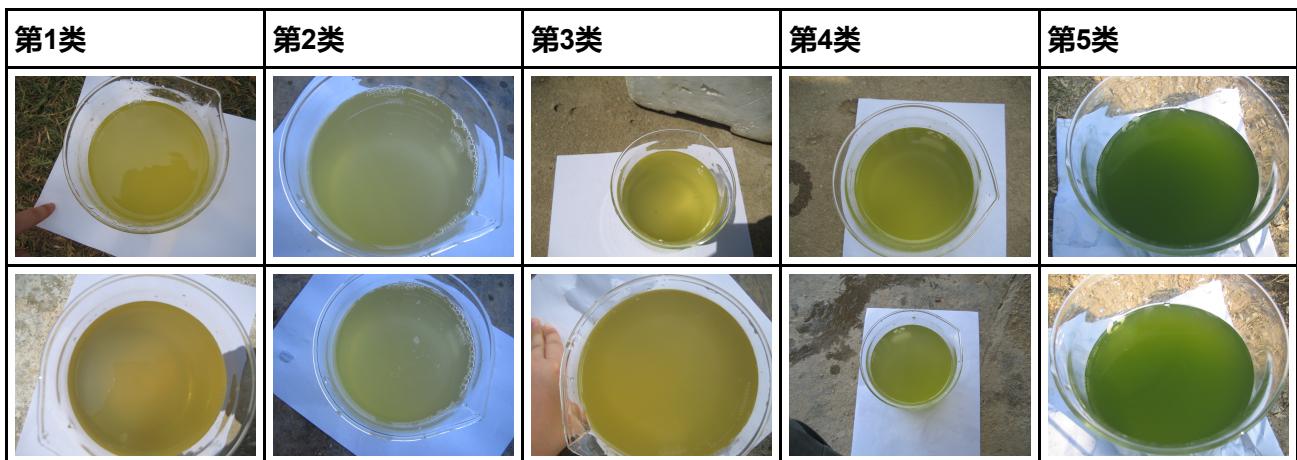
In [2]:

```
# 运行代码时会有很多warning输出，如提醒新版本之类的
import warnings
warnings.filterwarnings('ignore')
```

实例1：基于水色图像的水质评价系统

1. 问题背景

某地区的鱼池塘水样的数据，水质分为5类：



有经验的渔业人员、环保人员等可通过观察水色变化来调控水质或发行污染源等。由于这些多是通过经验和肉眼观察进行判断，存在主观性和观察偏倚，使得观察结果的可比性、可重复性降低，难以推广。

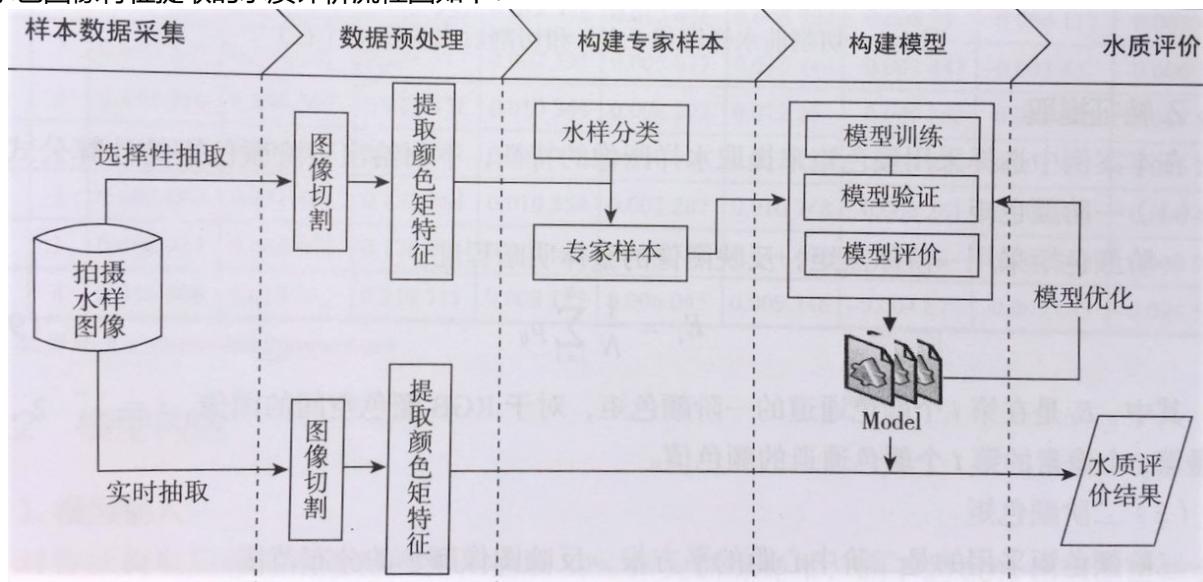
目前，数字图像处理技术基于计算机视觉，为计算机监控和识别在图像数据的应用提供更大的空间。

2. 分析方法与过程

通过拍摄水样，采集得到水样图像。然而图像数据的维度过大，不易分析，需要从中提取水样图像的特征，提取反映图像本质的一些关键指标，从而可以让分类算法对这些特征进行分类模型的构建，达到自动进行图像识别或分类的目的。

显然，图像特征提取是图像识别或分类的关键步骤，图特征提取的效果直接影响到图像识别和分类的性能。

基于水色图像特征提取的水质评价流程图如下：



这个评价系统包括以下步骤：

1. 从采集到的原始水样图像中进行选择性抽取与实时抽取，形成建模数据和增量数据。
2. 对步骤1中形成的两个数据集进行数据预处理，包括 图像切割 和 颜色矩(Color Moment) 特征提取。
3. 利用 步骤2 形成的已完成数据预处理的建模数据，由有经验的专家对水样图像进行分类标注，构建专
家样本，即有标注的训练数据
4. 利用 步骤3 的训练数据构建分类模型
5. 利用 步骤4 中构建好的分类模型进行水质评价。

3. 图像数据的特征表达

通常，图像特征主要包括：颜色特征、纹理特征、形状特征和空间关系特征等。与几何特征相比，本实例中，水色图像是均匀的，因此颜色特征更稳健，对于物体的大小和方向均不敏感，表现出较强的鲁棒性。因此，在本问题中，主要关注图像的 **颜色特征** 这一全局特征。

一般，颜色特征是基于像素点的特征，所有属于图像或图像区域的像素都有各自的贡献。目前，已经有大量的研究成果，颜色处理常用的方法有 直方图法 和 颜色矩 (color moment) 方法 等。

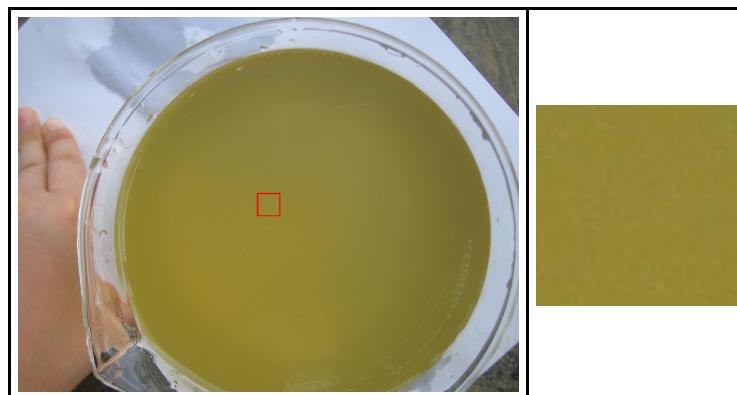
本问题中选用 颜色矩。一副图像的色彩分布可以认为是一种概率分布，图像可以由各个颜色的阶距来描述。颜色矩 包含各个颜色通道的一阶矩、二阶矩和三阶矩。对于一副RGB颜色空间的图像，具有R、G 和 B 三个颜色通道，每个颜色通道有一阶矩，二阶矩和三阶矩，因此，共有9个分量，即9个特征维度。

4. 图像预处理

4-1 图像切割

采集到的水样图像包含盛水容器，容器的颜色与水体颜色差异很大，同时水体位于图像中央，为了提取水色的特征，需要提取水样图像中央部分具有代表意义的图像。

具体实施方式是：提取水样图像中央 101×101 像素的图像。设原始图像的大小为 $M \times N$ ，则截取宽 从 第 $\text{fix}(M/2) - 50$ 个像素点到 第 $\text{fix}(M/2) + 50$ 个像素点，长从 第 $\text{fix}(N/2) - 50$ 到 第 $\text{fix}(N/2) + 50$ 个像素点的子图像。



In [3]:

```
# 定义切割图像类
class DataSeg():
    def __init__(self):

        self.data_path = 'data/images' # 定义原始图像数据路径
        # 获取原始图像数据中的所有图像名称
        self.image_names = os.listdir(self.data_path)

        self.location_data_path = 'data/seglocation_images' # 定义标注了切割图像位置的定位图像数
        据路径
        #如果路径不存在，则创建定位图像数据路径
        if not os.path.isdir(self.location_data_path):
            os.makedirs(self.location_data_path)

        self.seg_data_path = 'data/seg_images' # 定义切割图像数据路径
        # 如果路径不存在，则创建切割图像数据路径
        if not os.path.isdir(self.seg_data_path):
            os.makedirs(self.seg_data_path)

    # 定义图像切割方法
    def seg(self):
        # print ('start!...')

        for image_name in self.image_names:
            # 打开图像，获取图像宽度weight 和高度height
            img = Image.open(self.data_path + '/' + image_name)
            loc_img = Image.open(self.data_path + '/' + image_name)
            weight, height = img.size

            # 定义切割图像坐标left, right, upper, lower, 图像左上角坐标为 (0, 0)
            # 截取宽度 (left, wight)
            # 截取高度 (upper, lower)
            left = (weight / 2) - 50
            right = (weight / 2) + 51
            upper = (height / 2) - 50
            lower = (height / 2) + 51

            # 在原图像中标注切割图像位置
            draw = ImageDraw.Draw(loc_img)
            draw.line([(left, upper), (left, lower)], fill=(255, 0, 0), width = 5)
            draw.line([(left, upper), (right, upper)], fill=(255, 0, 0), width = 5)
            draw.line([(right, lower), (left, lower)], fill=(255, 0, 0), width = 5)
            draw.line([(right, lower), (right, upper)], fill=(255, 0, 0), width = 5)

            # 保存定位图像到文件
            loc_img.save(self.location_data_path + '/' + image_name)

            # 切割图像，切割坐标[left, upper, right, lower]
            box = [left, upper, right, lower]
            seg_img = img.crop(box)

            # 保存切割图像到文件
            seg_img.save(self.seg_data_path + '/' + image_name)

        # print ('end!')
```

In [4]:

```
import time
start = time.time()

#创建切割图像类实例
data = DataSeg()
#调用切割图像类的切割方法
data.seg()

end = time.time()
print('用时 %.4f 秒' % (end - start))
```

用时 44.21 秒

4-2 特征提取

本问题中选用 颜色矩 (color moment) 来提取水样图像的特征。对于一副RGB颜色空间的图像，具有R、G 和 B 三个颜色通道，每个颜色通道有一阶矩，二阶矩和三阶矩，因此，共有9个分量，即9个特征维度。

(1) 一阶颜色矩

采用一阶原点距，反映图像的整体明暗程度。 $E_i = \frac{1}{N} \sum_{j=1}^N p_{ij}$, 其中 E_i 是在第*i*个颜色通道的一阶颜色矩，对于 RGB 颜色空间的图像， $i = 1, 2, 3$, p_{ij} 是第*j*个像素的第*i*个颜色通道的颜色值。

(2) 二阶颜色矩

二阶颜色矩采用二阶中心距的平方根，反映图像颜色的分布范围。 $\sigma_i = \sqrt{\frac{1}{N} \sum_{j=1}^N (p_{ij} - E_i)^2}$, 其中 σ_i 是在第*i*个颜色通道的二阶颜色矩。

(3) 三阶颜色矩

三阶颜色距采用的是三阶中心距的立方根，反映图像颜色分布的对称性。 $s_i = \sqrt[3]{\frac{1}{N} \sum_{j=1}^N (p_{ij} - E_i)^3}$, 其中 s_i 是在第*i*个颜色通道的三阶颜色矩

提取切割后的图像颜色矩，作为图像的颜色特征。 每个图像文件名第一个数字是类别，第二个数字是序号。得到每个图片的9个特征值。 对所有的图片都进行同样的操作， 存储在 `data/moment.csv` 文件，数据样本如下：

	类别	序号	R通道一阶矩	G通道一阶矩	B通道一阶矩	R通道二阶矩	G通道二阶矩	B通道二阶矩	R通道三阶矩	G通道三阶矩	B通道三阶矩
0	1	1	0.582823	0.543774	0.252829	0.014192	0.016144	0.041075	-0.012643	-0.016090	-0.041536
1	1	10	0.641660	0.570657	0.213728	0.015439	0.011178	0.013708	0.009727	-0.003724	-0.003779
2	1	11	0.603684	0.576719	0.282254	0.008659	0.007075	0.012204	-0.004695	-0.002571	-0.009451
3	1	12	0.589706	0.593743	0.252242	0.007908	0.005941	0.010568	0.003303	-0.003417	-0.005273
4	1	13	0.591096	0.592093	0.253595	0.007448	0.006495	0.012152	0.000496	-0.002236	-0.005096

In [5]:

```
# 定义图像颜色矩特征类
class DataMoment():
    def __init__(self):
        # 获取全部切割图像名称
        self.seg_data_path = 'data/seg_images'
        self.segimage_names = os.listdir(self.seg_data_path)
        self.segimage_names.sort()

    # 定义图像颜色矩特征提取方法
    def moment(self):
        color_features = [] # 定义存储图像特征的特征列表
        # 列表的第一项是特征序列中每个维度的含义
        str = ['类别', '序号', 'R通道一阶矩', 'G通道一阶矩', 'B通道一阶矩', 'R通道二阶矩',
               'G通道二阶矩', 'B通道二阶矩', 'R通道三阶矩', 'G通道三阶矩', 'B通道三阶矩']

        color_features.append(str)

        for image_name in self.segimage_names:
            color_feature = []
            image_name_list = image_name.split('.')[0].split('_')
            color_feature.extend(image_name_list)
            img = Image.open(self.seg_data_path + '/' + image_name)
            #RGB图像的颜色分离
            r, g, b = img.split()
            r = np.array(r)/255.0
            g = np.array(g)/255.0
            b = np.array(b)/255.0
            #
            # 抽取颜色矩特征
            # 一阶颜色矩, 均值
            r_mean = np.mean(r) # np.sum(h)/float(N)
            g_mean = np.mean(g) # np.sum(s)/float(N)
            b_mean = np.mean(b) # np.sum(v)/float(N)
            color_feature.extend([round(r_mean, 9), round(g_mean, 9), round(b_mean, 9)])

            # 二阶颜色矩, 均方差
            r_std = np.std(r) # np.sqrt(np.mean(abs(h - h.mean())**2))
            g_std = np.std(g) # np.sqrt(np.mean(abs(s - s.mean())**2))
            b_std = np.std(b) # np.sqrt(np.mean(abs(v - v.mean())**2))
            color_feature.extend([round(r_std, 9), round(g_std, 9), round(b_std, 9)])

            # 三阶颜色矩, 三阶中心距的立方根
            r_skewness = np.mean((r - r_mean) ** 3)
            g_skewness = np.mean((g - g_mean) ** 3)
            b_skewness = np.mean((b - b_mean) ** 3)
            # np.cbrt():以元素方式返回数组的立方根
            r_thirdMoment = np.cbrt(r_skewness)
            g_thirdMoment = np.cbrt(g_skewness)
            b_thirdMoment = np.cbrt(b_skewness)
            # 添加颜色矩特征
            color_feature.extend([round(r_thirdMoment, 9), round(g_thirdMoment, 9), round(b_thirdMoment, 9)])
            color_features.append(color_feature)

        # 返回颜色矩特征
        return color_features
```

In [6]:

```
#实例化抽取图像的颜色矩特征类，生成的特征文件命名为data路径下的moment_2.csv，该文件的编码方式为utf-8
data = DataMoment()
#print('start!...')

# 抽取颜色矩特征
feature = data.moment()
outfile = 'data/moment_2.csv'
# 将颜色矩特征写入文件
with open(outfile, 'w', newline='', encoding='utf-8') as out:
    csv_writer = csv.writer(out, dialect='excel')
    for f in feature:
        csv_writer.writerow(f)
#print('end!')
```

5. 分析过程和方法

5-1 数据分析

In [7]:

```
-*- coding: utf-8 -*-
import pandas as pd

# datafile= 'data/moment.csv' #数据文件
datafile= 'data/moment_2.csv' #数据文件
#读取原始数据，指定utf-8编码，包含中文
data = pd.read_csv(datafile, encoding = 'utf-8')
data.head()
```

Out[7]:

类别	序号	R通道一阶矩	G通道一阶矩	B通道一阶矩	R通道二阶矩	G通道二阶矩	B通道二阶矩	R通道三阶矩		
0	1	1	0.582765	0.543520	0.251279	0.013076	0.016572	0.041060	-0.012178	-0
1	1	10	0.642353	0.570849	0.214091	0.013977	0.011087	0.011769	0.008270	0.0
2	1	11	0.603865	0.576691	0.283321	0.006960	0.006016	0.010644	-0.005514	-0
3	1	12	0.591891	0.592854	0.251016	0.005501	0.004953	0.008057	-0.003779	-0
4	1	13	0.592875	0.591586	0.252296	0.005697	0.005498	0.010185	-0.003793	-0

In [8]:

```
data[u'R通道一阶矩'].describe()
```

Out[8]:

```
count    203.000000
mean      0.540036
std       0.065828
min       0.213625
25%      0.502348
50%      0.544478
75%      0.583425
max       0.700655
Name: R通道一阶矩, dtype: float64
```

In [9]:

```
data[u'G通道二阶矩'].describe()
```

Out[9]:

```
count    203.000000
mean      0.006784
std       0.003188
min       0.003880
25%      0.005130
50%      0.005792
75%      0.007112
max       0.032012
Name: G通道二阶矩, dtype: float64
```

5-2 数据变换

观察数据，发现数据中特征值的取值范围很小，彼此之间的区分度太小，因此，我们需要对所有特征都统一进行扩大，从而提高区分度和准确度。方法可以有以下两种：

1. 统一乘以一个常数 K (不能太大，也不能太小，例如这里选择10或者30)
2. min-max归一化 (见上节课slide)

哪种方法好？后面可以通过实验来验证。

5-2 数据分割

为了检验模型是否有效，我们需要把数据分割为训练数据train和测试数据test，这样，在train数据上构建的模型，可以在test数据上进行评估和检验。

5-3 构建模型

这个实例中的数据有标注的类别，采用有监督的分类算法。回忆以前使用过决策树算法（Decision Tree）和朴素贝叶斯算法（Naive Bayes），这里我们选择支持向量机（Support Vector Machine）算法。

这几种算法都可以使用。参考本节课和前面课程中使用决策树和朴素贝叶斯算法，来比较不同算法的性能。

数据变换的第一种方法：

- 统一乘以一个常数 K (K=30, 也可以实验其他数字, 例如10, 20等, 观察结果)

In [10]:

```
#-*- coding: utf-8 -*-
import pandas as pd

# 加载数据
# datafile= 'data/moment.csv' #数据文件
datafile= 'data/moment_2.csv' #数据文件
#读取原始数据, 指定gbk编码, 包含中文
#data = pd.read_csv(datafile, encoding = 'gbk')
#读取原始数据, 指定utf-8编码, 包含中文
data = pd.read_csv(datafile, encoding = 'utf-8')

data = data.as_matrix()

# 打乱数据
from random import shuffle # 引入随机函数
shuffle(data) #随机打乱数据

#分割训练数据和测试数据
data_train = data[:int(0.8*len(data)), :] #选取前80%作为train
data_test = data[int(0.8*len(data)):, :] #选取后20%作为test

#构造特征和类别标签
# 数据变换 方法 (1) 统一放大30倍, 可以试试放大其他倍数的实验结果如何?
x_train = data_train[:, 2:] * 30
y_train = data_train[:, 0].astype(int) #取出class label
x_test = data_test[:, 2:] * 30
y_test = data_test[:, 0].astype(int) #取出class label

#导入模型并训练模型
from sklearn import svm # 导入支持向量机算法
model = svm.SVC() # 构造模型, 默认是线性核 kernel='linear'
model.fit(x_train, y_train) # 使用训练数据和训练数据对应的类标来训练模型

# 返回模型在测试数据的准确率
model.score(x_test, y_test) #返回准确率
#不放大的acc=0.1951, 30倍的acc=0.8080 (0.8293, 0.8536等? 为什么?), 20倍的acc=0.8049, 10倍的acc=0.7317

# 以下是5折交叉验证的代码
#from sklearn.model_selection import cross_val_score
#scores = cross_val_score(model, x_train, y_train, cv=5)
#print scores
#print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

Out[10]:

0.87804878048780488

数据变换的第二种方法：

- 使用min-max归一化处理

In [11]:

```
#-*- coding: utf-8 -*-
import pandas as pd

#加载数据

# datafile= 'data/moment.csv' #数据文件
datafile= 'data/moment 2.csv' #数据文件
#读取原始数据, 指定gbk编码, 包含中文
#data = pd.read_csv(datafile, encoding = 'gbk')
#读取原始数据, 指定utf-8编码, 包含中文
data = pd.read_csv(datafile, encoding = 'utf-8')

# 只取9个颜色矩数据, 过滤掉前两列的序号和类标
data2 = data[data.columns[2:]] #只取数据

# 数据变换采用第二个方法: min-max归一化
data2 = (data2 - data2.min(axis=0)) / (data2.max(axis=0) - data2.min(axis=0))

# 数据变换处理后, 合并类标
data1 = data[data.columns[0]] #只取类标
data2['Class'] = data1 # 合并
data2 = data2.as_matrix()

from random import shuffle # 引入随机函数
shuffle(data2) #随机打乱数据

#分割训练数据和测试数据
data_train = data2[:int(0.8*len(data2)), :] #选取前80%作为train
data_test = data2[int(0.8*len(data2)):, :] #选取后20%作为test

#构造特征和类别标签
x_train = data_train[:, :9]
y_train = data_train[:, 9].astype(int) #取出class label
x_test = data_test[:, :9]
y_test = data_test[:, 9].astype(int) #取出class label

#导入模型并训练模型
from sklearn import svm # 导入支持向量机模型
model = svm.SVC(C=5.0, kernel='linear') # 构造模型和指定参数
model.fit(x_train, y_train) # 在训练数据上训练模型

# 在测试数据上得到模型的准确率
model.score(x_test, y_test) #返回准确率
# min-max 归一化: rbf的acc=0.4146, linear的acc=0.5366, C=5的acc=0.8293
```

Out[11]:

0.73170731707317072

观察结果

在这个数据上, 简单放大倍数比min-max归一化方法的结果要好。在其他数据上, 并不一定是同样的结果。

6. 保存模型

如果采用类似上述的实验，对比各种不同方法和参数对结果的影响，例如：

1. 不同数据变换的方法的不同结果
2. 不同放大倍数参数K的不同结果
3. 不同算法参数的影响（C值，核函数等）
4. 不同算法（决策树、支持向量机等）的不同结果

等等

我们需要将模型和结果保存固定下来，保存在 data/result/ 目录下

In [12]:

```
import pickle
pickle.dump(model, open('data/result/svm.model', 'wb'))

#最后一句保存模型，以后可以通过下面语句重新加载模型：
#model = pickle.load(open('../tmp/svm.model', 'rb'))

#导入输出相关的库，生成混淆矩阵
from sklearn import metrics
cm_train = metrics.confusion_matrix(y_train, model.predict(x_train)) #训练样本的混淆矩阵
cm_test = metrics.confusion_matrix(y_test, model.predict(x_test)) #测试样本的混淆矩阵

#保存结果
outputfile1 = 'data/result/cm_train.xls'
outputfile2 = 'data/result/cm_test.xls'
pd.DataFrame(cm_train).to_excel(outputfile1)
pd.DataFrame(cm_test).to_excel(outputfile2)
#pd.DataFrame(cm_train, index = range(1, 6), columns = range(1, 6)).to_excel(outputfile1)
#pd.DataFrame(cm_test, index = range(1, 6), columns = range(1, 6)).to_excel(outputfile2)
```

7. 对比不同算法的模型预测结果

朴素贝叶斯不能直接应用在本实例中，原因是：本实例中的数据特征是连续数字，不是离散类型。

In [13]:

```
#-*- coding: utf-8 -*-
import pandas as pd

# 加载数据
# datafile= 'data/moment.csv' #数据文件
datafile= 'data/moment_2.csv' #数据文件
#读取原始数据，指定gbk编码，包含中文
#data = pd.read_csv(datafile, encoding = 'gbk')
#读取原始数据，指定utf-8编码，包含中文
data = pd.read_csv(datafile, encoding = 'utf-8')

data = data.as_matrix()

# 打乱数据
from random import shuffle # 引入随机函数
shuffle(data) #随机打乱数据

#分割训练数据和测试数据
data_train = data[:int(0.8*len(data)), :] #选取前80%作为train
data_test = data[int(0.8*len(data)):, :] #选取后20%作为test

#构造特征和类别标签
# 数据变换 方法 (1) 统一放大30倍，可以试试放大其他倍数的实验结果如何？
x_train = data_train[:, 2:] * 30
y_train = data_train[:, 0].astype(int) #取出class label
x_test = data_test[:, 2:] * 30
y_test = data_test[:, 0].astype(int) #取出class label

#导入不同模型并训练多个模型
from sklearn.svm import SVC # 导入支持向量机算法
SVM_linear = SVC(kernel='linear') # 构造支持向量机模型，默认是线性核 kernel='linear'
SVM_linear.fit(x_train, y_train) # 使用训练数据和训练数据对应的类标来训练模型

from sklearn.svm import SVC # 导入支持向量机算法
SVM_rbf = SVC(kernel='rbf') # 构造支持向量机模型，默认是线性核 kernel='rbf'
SVM_rbf.fit(x_train, y_train) # 使用训练数据和训练数据对应的类标来训练模型

from sklearn.tree import DecisionTreeClassifier #导入决策树模型
tree = DecisionTreeClassifier(criterion='entropy') # 构建决策树模型
tree.fit(x_train, y_train) # 训练决策树模型

from sklearn.neighbors import KNeighborsClassifier # 导入k最近邻算法
k = 3 # 设定最近邻居个数K
kNN = KNeighborsClassifier(n_neighbors=k) # 构造k=3最近邻模型
kNN.fit(x_train, y_train) # 使用训练数据和训练数据对应的类标来训练模型

# 返回模型在测试数据的准确率
print("SVM (线性核)模型的准确率: %.4f" % (SVM_linear.score(x_test, y_test))) #返回准确率
print("SVM (rbf) 模型的准确率: %.4f" % (SVM_rbf.score(x_test, y_test))) #返回准确率
print("决策树模型的准确率: %.4f" % (tree.score(x_test, y_test))) #返回准确率
print("kNN (k=3) 模型的准确率: %.4f" % (kNN.score(x_test, y_test))) #返回准确率
```

SVM (线性核)模型的准确率: 0.9756
SVM (rbf) 模型的准确率: 0.9756
决策树模型的准确率: 0.8780
kNN (k=3) 模型的准确率: 0.9024

观察并练习

对比并记录不同的算法和算法的不同参数对结果的影响，寻找出合适的模型算法和参数。

一、对于Sklearn库实现的SVM算法 (<http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC>)

[<http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC> (<http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC>)]

```
class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None) [source]
```

例如，可以对比不同的参数：

1. 不同的核函数 kernel='rbf'默认，或者'linear', 'poly', 'sigmoid'等
2. 不同的C代价参数

二、对于Sklearn库实现的决策树算法 (<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier>)

[<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier> (<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier>)]

```
class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_split=1e-07, class_weight=None, presort=False) [source]
```

可以选择的参数包括：

1. 不同的属性选择度量criterion='gini'或'entropy'

三、对于Sklearn库实现的k最近邻算法 (<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors>)

[<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors> (<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors>)]

```
class sklearn.neighbors.NearestNeighbors(n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=1, **kwargs) [source]
```

可以比较不同的k参数 跟n_neighbors (默认为5)

实例2：服装流行色预测

背景：

最早的流行色预测机构是在1915年成立的The Color Association of the United States(美国色彩协会)。国际流行色的预测是由总部设在法国巴黎的“国际流行色协会”完成。国际流行色协会各成员国专家每年召开两次会议，讨论未来十八个月的春夏或秋冬流行色定案。协会从各成员国提案中讨论、表决、选定一致公认的三组色彩为这一季的流行色，分别为男装、女装和休闲装。国际流行色协会发布的流行色定案是凭专家的直觉判断来选择的，西欧国家的一些专家是直觉预测的主要代表，特别是法国和德国专家，一直是国际流行色的先驱，他们对西欧的市场和艺术有着丰富的感受，以个人的才华、经验与创造力就能设计出代表国际潮流的色彩构图，他们的直觉和灵感非常容易得到其他代表的认同，也能得到世界的认同。中国的流行色是由中国流行色协会制定，他们通过观察国内外流行色的发展状况，取得大量的市场资料，然后对资料作分析和筛选而制定，在色彩定制中还加入了社会、文化、经济、地区等因素。除了纺织品和时装等时尚行业，各国各行业均成立了流行色的预测机构。

流行色预测的目的是为了帮助流行生产和销售企业（纺织品、时装、时尚等）准确抓住流行的动向，开展了时装、纺织品为主的色彩流行与预测的研究，指导企业的生产与销售，提前生产出符合潮流的产品，以得到更多的利润。

流行色预测怎么做？

1. **挖掘目标**：针对女装的流行色进行预测
2. **采集数据**：从淘宝或天猫等电商网站采集数据
3. **数据预处理**：从数据中过滤出女装数据，对应的图片、时间戳等，并从图片中选取服装图像的像素区域，获得图像的颜色距特征
4. **数据属性选择**：需要选取电商数据中的关键属性，如，
 - A. 服装数据（图像特征，时间戳，宝贝详情，销售量，累计评论，大家印象），商铺基本信息（信誉、资质、工商执照、动态评分等）等
5. **构建流行色预测模型**
 - A. 随机选取数据的80%做训练样本，剩余的20%作为测试样本
 - B. 选择分类算法（使用决策树，k-最近邻，支持向量机等分类算法）
 - C. 运用模型预测流行色（未流失、准流失或已流失）

结论

- 对颜色图像选用**颜色矩**特征。对图像数据的预处理包括图像分割，特征设计和抽取等。
- 比较不同数据标准化技术的性能，比较不同分类算法的性能。