一、 决策树的建立与分类

建立模型与分类过程如下：

1. 将原始数据集转化成 csv 文件并存储为 data.csv

	A	B	C	D	E	F
1	id	figure	hair_colo	age	class	
2	1	short	gold	old	1	
3	2	tall	red	old	1	
4	3	tall	gold	old	1	
5	4	short	gold	adult	1	
6	5	tall	black	child	2	
7	6	short	black	old	2	
8	7	tall	black	old	2	
9	8	tall	black	adult	2	
10	9	short	gold	child	2	
11						
12						
13						

2. 读取 csv 数据并序列化

In [1]:
```
from sklearn.feature_extraction import DictVectorizer
import csv
from sklearn import tree
from sklearn import preprocessing
from sklearn.externals.six import StringIO
from IPython.display import Image
```

In [2]:
```
data = open('E:\Working\DataMining\homework\hw4\data.csv', 'r')
reader = csv.reader(data)
headers = next(reader)

print(headers)

featureList = []
labelList = []

for row in reader:
    labelList.append(row[len(row) - 1])
    rowDict = {}
    for i in range(1, len(row) - 1):
        rowDict[headers[i]] = row[i]
    featureList.append(rowDict)

print(featureList)
print(labelList)
```

```
['id', 'figure', 'hair_color', 'age', 'class']
[{'figure': 'short', 'hair_color': 'gold', 'age': 'old'}, {'figure': 'tall', 'hair_color': 'red', 'age': 'old'}, {'figure': 'tall', 'hair_co
lor': 'gold', 'age': 'old'}, {'figure': 'short', 'hair_color': 'gold', 'age': 'adult'}, {'figure': 'tall', 'hair_color': 'black', 'age': 'ch
ild'}, {'figure': 'short', 'hair_color': 'black', 'age': 'old'}, {'figure': 'tall', 'hair_color': 'black', 'age': 'old'}, {'figure': 'tall',
'hair_color': 'black', 'age': 'adult'}, {'figure': 'short', 'hair_color': 'gold', 'age': 'child'}]
['1', '1', '1', '1', '2', '2', '2', '2', '2']
```

3. 将特征和类别编码为特征向量

In [3]:
```
v = DictVectorizer()
dummyX = v.fit_transform(featureList).toarray()

print("dummyX: " + str(dummyX))
print(v.get_feature_names())
print("labelList: " + str(labelList))

# vectorize class labels
lb = preprocessing.LabelBinarizer()
dummyY = lb.fit_transform(labelList)
print("dummyY: ", str(dummyY))
```

```
dummyX: [[ 0.  0.  1.  1.  0.  0.  1.  0.]
 [ 0.  0.  1.  0.  1.  0.  0.  1.]
 [ 0.  0.  1.  0.  1.  0.  1.  0.]
 [ 1.  0.  0.  1.  0.  0.  1.  0.]
 [ 0.  1.  0.  0.  1.  1.  0.  0.]
 [ 0.  0.  1.  1.  0.  1.  0.  0.]
 [ 0.  0.  1.  0.  1.  1.  0.  0.]
 [ 1.  0.  0.  0.  1.  1.  0.  0.]
 [ 0.  1.  0.  1.  0.  0.  1.  0.]]
['age=adult', 'age=child', 'age=old', 'figure=short', 'figure=tall', 'hair_color=black', 'hair_color=gold', 'hair_color=red']
labelList: ['1', '1', '1', '1', '2', '2', '2', '2', '2']
dummyY:  [[0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]]
```

4. 建立决策树，采用 CART 算法。

第一步选取特征为“发色是否为黑色”，其 gini 不纯度为 0.494，在所有的特征中最小，若发色为黑色，则分类为第二组，此时 gini 不纯度为 0，该分支可停止；若发色非黑色，则继续询问“年龄是否为儿童”，其 jini 不纯度为 0.32，若是儿童则分类为第一组，否则分类为第二组。至此，所有的叶结点的 gini 不纯度均为 0，算法停止。

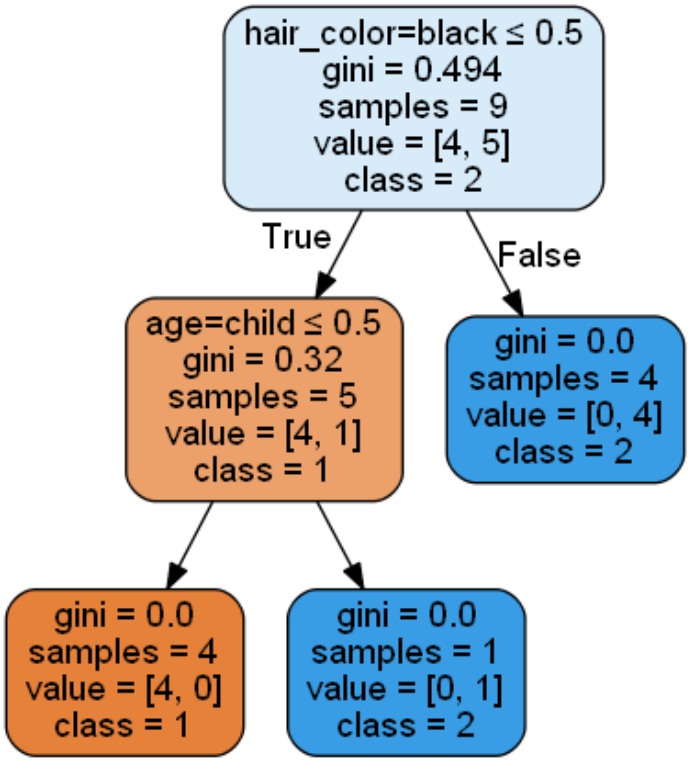
代码和决策树可视化如下：

In [4]:
```
clf = tree.DecisionTreeClassifier(criterion='gini')
clf = clf.fit(dummyX, dummyY)
print("clf: ", str(clf))
```

```
clf:  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [6]:
```
import pydotplus
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=v.get_feature_names(),
                                class_names=['1', '2'],
                                filled=True, rounded=True,
                                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Out[6]:



5. 分类（矮，黑色，成年）

首先询问“发色是否为黑色”，结果为发色是黑色，因此分类为第二组。分类结束。

二、 朴素贝叶斯分类

建立模型与分类过程如下：

代码输出中的所有概率均为取对数结果，采用伯努利离散朴素贝叶斯分类器，所有特征均为二值分布，且相互独立。

先验概率为[4/9, 5/9]，各个特征的后验概率的对数值可见代码中特征分布的输出。

对给定样本的后验概率分别为：

1/2 * 0 * 1/4 和 2/5 * 4/5 * 1/5，因此根据朴素贝叶斯分类器，选取最大化后验概率作为分类结果，样本被分到第二组。

```
from sklearn.naive_bayes import BernoulliNB

clf = BernoulliNB(alpha=2.0,fit_prior=True)
clf.fit(dummyX, dummyY)
```

```
D:\anaconda\lib\site-packages\sklearn\utils\validation.py:547: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[6]:
```
BernoulliNB(alpha=2.0, binarize=0.0, class_prior=None, fit_prior=True)
```

In [7]:
```
clf.class_log_prior_
```

Out[7]:
```
array([-0.81093022, -0.58778666])
```

In [8]:
```
clf.feature_log_prob_
```

Out[8]:
```
array([[-0.98082925, -1.38629436, -0.47000363, -0.69314718, -0.69314718,
        -1.38629436, -0.47000363, -0.98082925],
       [-1.09861229, -0.81093022, -0.81093022, -0.81093022, -0.58778666,
        -0.40546511, -1.09861229, -1.5040774 ]])
```

In [9]:
```
clf.class_count_
```

Out[9]:
```
array([ 4.,  5.])
```

In [10]:
```
clf.feature_count_
```

Out[10]:
```
array([[ 1.,  0.,  3.,  2.,  2.,  0.,  3.,  1.],
       [ 1.,  2.,  2.,  2.,  3.,  4.,  1.,  0.]])
```

In [11]:
```
clf.predict([[ 1., 0., 0., 1., 0., 1., 0., 0.]])
```

Out[11]:
```
array([1])
```

三、k-means 聚类

初始化并逐个增加数据点到聚类模型中：

In [29]:
```
from sklearn.cluster import KMeans
clf = KMeans(n_clusters=2)
X = np.array([[0, 4], [6, 5]])
clf.fit(X)
```

Out[29]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [32]:
```
points = np.array([[1, 3], [1, 2], [2, 1], [2, 2], [2, 3], [3, 2], [5, 3], [4, 3], [4, 5], [5, 4], [5, 5], [6, 4], [6, 5]])
for i in range(points.shape[0]):
    X = np.append(X , np.expand_dims(points[i],axis=0), axis=0)
    clf.fit_predict(X)
    print(clf.cluster_centers_)
    print(clf.labels_)
    print()
```

逐渐聚类的类中心和聚类结果如下，聚类的过程为对每个新输入的点根据其距离最近的类中心进行分类，然后归入该类，之后对所有的点迭代重复该操作，直到类中心的变化不超过阈值或到达迭代次数上限：

```
[[ 6.    5. ]
 [ 0.5  3.5]]
[1 0 1]

[[ 0.66666667  3.        ]
 [ 6.          5.        ]]
[0 1 0 0]

[[ 1.   2.5]
 [ 6.   5. ]]
[0 1 0 0 0]

[[ 1.2  2.4]
 [ 6.   5. ]]
[0 1 0 0 0 0]

[[ 1.33333333  2.5       ]
```

```
 [ 6.          5.        ]]
[0 1 0 0 0 0 0]

[[ 1.57142857  2.42857143]
 [ 6.          5.        ]]
[0 1 0 0 0 0 0 0]

[[ 1.57142857  2.42857143]
 [ 5.5         4.        ]]
[0 1 0 0 0 0 0 0 1]

[[ 1.57142857  2.42857143]
 [ 5.          3.66666667]]
[0 1 0 0 0 0 0 0 1 1]

[[ 1.57142857  2.42857143]
 [ 4.75        4.        ]]
[0 1 0 0 0 0 0 0 1 1 1]

[[ 1.57142857  2.42857143]
 [ 4.8         4.        ]]
[0 1 0 0 0 0 0 0 1 1 1 1]

[[ 1.57142857  2.42857143]
 [ 4.83333333  4.16666667]]
[0 1 0 0 0 0 0 0 1 1 1 1 1]

[[ 5.          4.14285714]
 [ 1.57142857  2.42857143]]
[1 0 1 1 1 1 1 1 0 0 0 0 0 0]

[[ 1.57142857  2.42857143]
 [ 5.125       4.25      ]]
[0 1 0 0 0 0 0 0 1 1 1 1 1 1 1]
```

对最后的聚类结果进行可视化如下：

In [35]:
```
import matplotlib.pyplot as plt
g1 = np.array([X[i] for i in range(X.shape[0]) if clf.labels_[i] == 0])
g2 = np.array([X[i] for i in range(X.shape[0]) if clf.labels_[i] == 1])
```

In [40]:
```
plt.scatter(g1[:, 0], g1[:, 1], marker='x',color='r')
plt.scatter(g2[:, 0], g2[:, 1], marker='o',color='b')
plt.scatter([1.57142857, 5.125], [2.42857143, 4.25], marker='^',color='g')
plt.show()
```