Problem 1

The Gini index and entropy are the criteria to calculate the information gain. The decision tree algorithms use the information gain to split a node.

![title]

both measures measure the impurities of the information of a node, having multiple classes (in this case 2)

Entropy in statistics is analogous to entropy in thermodynamics where it signifies disorder. If there are multiple classes in a node, there is disorder in that node.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Problem 2

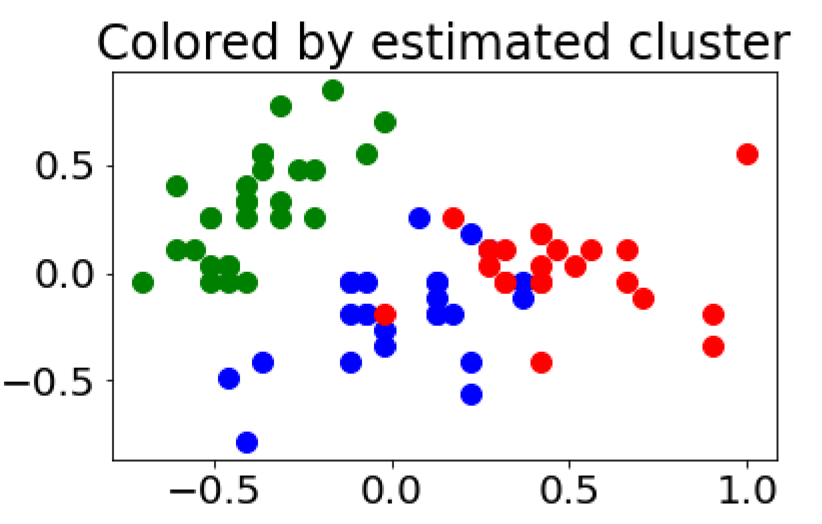
![title]

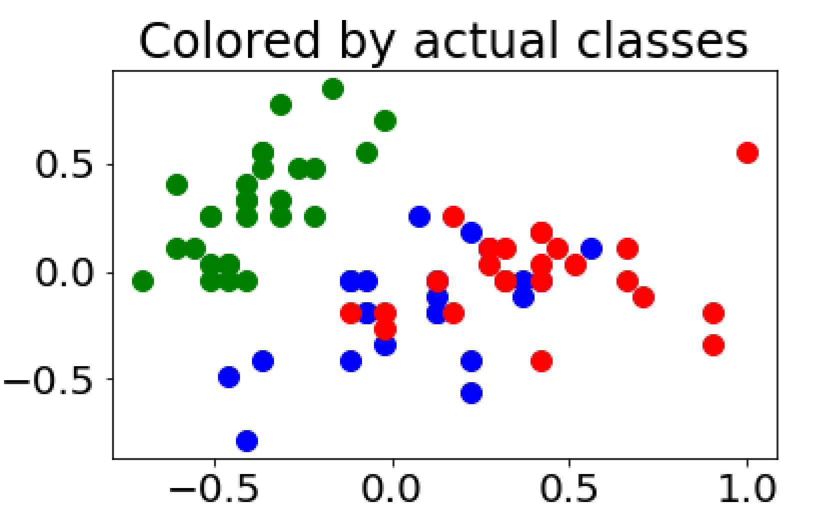
Problem 6

In [1]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import mean_squared_error
import pandas as pd
import kmeans
import kmeansIris
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
def kernelKmeans(kernel,k,X,maxIterations=100):
    K=[]

    for i in range(len(X)):
        aux=[]
        for j in range(len(X)):
            aux.append(kernel(np.dot(X[i],X[j])))
        K.append(list(aux))
    K=np.array(K)
    X_train, X_test, R_train, R_test = K[::2],K[1::2],X[::2],X[1::2]
    model = kmeans.kmeans(k, X_train)
    model.kmeanstrain(X_train,maxIterations)

    return model,X_train, X_test, R_train, R_test
```

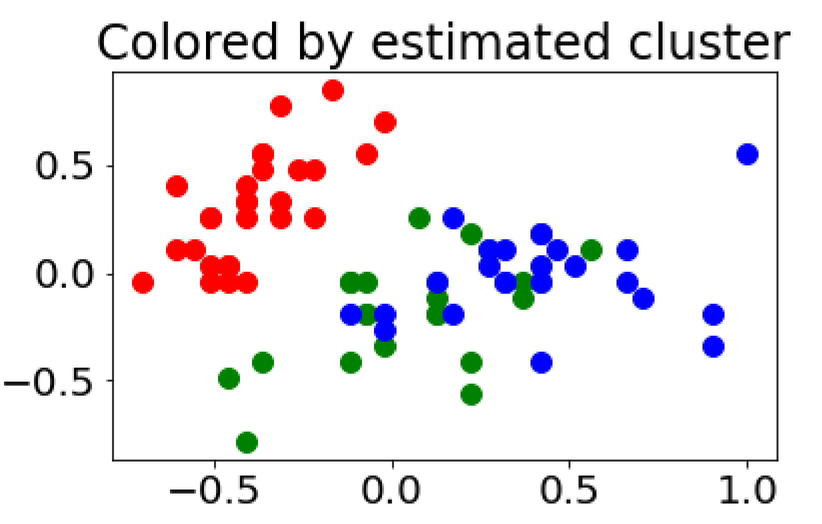




In [2]:
```
kmIris,X_train,X_test, R_train, R_test=kernelKmeans(lambda x:(x),3,kmeansIris.iris)
```

In [3]:
```
from sklearn.model_selection import train_test_split
cluster = np.ravel(kmIris.kmeansfwd(X_test))
#%% Scatter plot of sepal length and width colored by cluster estimate
plt.figure()
clu0 = np.where(cluster==0)
clu1 = np.where(cluster==1)
clu2 = np.where(cluster==2)
plt.plot(R_test[clu0,0],R_test[clu0,1],'go')
plt.plot(R_test[clu1,0],R_test[clu1,1],'bo')
plt.plot(R_test[clu2,0],R_test[clu2,1],'ro')
plt.title('Colored by estimated cluster')
plt.tight_layout()
from sklearn.metrics.cluster import rand_score
rand_score(kmeansIris.testt,cluster)
```

Out[3]: 1.0



In [4]:
```
np.mean([[0,0],[1,1]])
```

Out[4]: 0.5

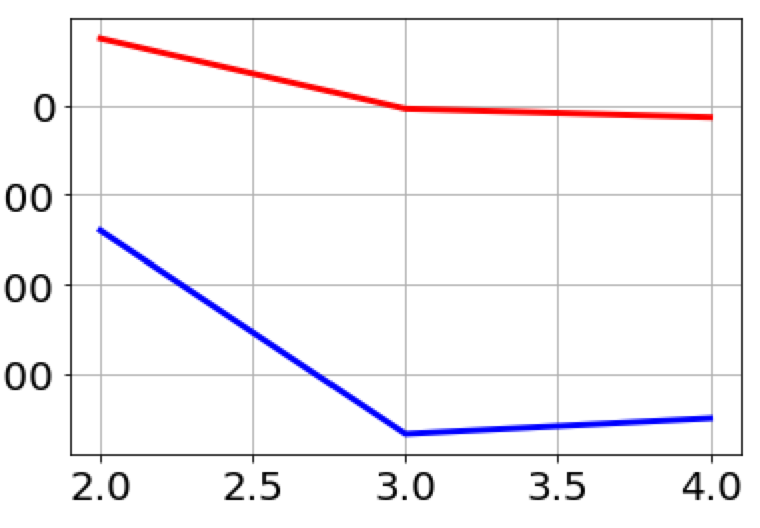
In [5]:
```
kmeansIris.testt
```

Out[5]:
```
array([2., 2., 1., 1., 1., 0., 2., 1., 0., 0., 2., 1., 1., 0., 2., 0., 1.,
       0., 0., 0., 2., 1., 2., 2., 1., 0., 1., 0., 1., 2., 0., 2., 1., 2.,
       1., 2., 2., 0., 1., 0., 0., 1., 1., 1., 0., 2., 0., 2., 1., 0., 0.,
       1., 1., 2., 2., 1., 2., 0., 2., 0., 0., 2., 0., 1., 0., 0., 1., 2.,
       0., 0., 2., 0., 2., 2., 0.])
```

In [6]:
```
from sklearn.mixture import GaussianMixture

t=range(2,5)
F=[]
S=[]
for i in t:
    s = GaussianMixture(n_components=i, random_state=0,covariance_type="spherical").fit(kmeansIris.iris)
    f = GaussianMixture(n_components=i, random_state=0,covariance_type='full').fit(kmeansIris.iris)
    S.append(s.bic(kmeansIris.iris))
    F.append(f.bic(kmeansIris.iris))
plt.grid()
plt.plot(t,S,'r')
plt.plot(t,F,'b')
```

Out[6]: [<matplotlib.lines.Line2D at 0x1c71f34f0d0>]



In [7]:
```
f.means_,f.covariances_
```

Out[7]:
```
(array([[ 0.53554155,  0.05622111,  0.70002904,  0.76942316,  2.        ],
        [-0.40713128,  0.27043091, -0.73047538, -0.73360656,  0.        ],
        [ 0.04505673, -0.21099554,  0.15959253,  0.09784836,  1.        ],
        [ 0.22219935, -0.15269502,  0.46674576,  0.52797738,  2.        ]]),
 array([[[ 8.07366465e-02,  7.07763459e-03,  3.75404143e-02,
          -1.90780684e-02,  0.00000000e+00],
         [ 7.07763459e-03,  6.00399696e-02,  1.31107964e-03,
           1.23160193e-02,  0.00000000e+00],
         [ 3.75404143e-02,  1.31107964e-03,  2.22227480e-02,
          -6.96439168e-03,  0.00000000e+00],
         [-1.90780684e-02,  1.23160193e-02, -6.96439168e-03,
           2.64730448e-02,  0.00000000e+00],
         [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
           0.00000000e+00,  1.00000000e-06]],

        [[ 2.87876474e-02,  3.55066094e-02,  2.44804121e-03,
           3.86189122e-03,  0.00000000e+00],
         [ 3.55066094e-02,  7.85320722e-02,  2.70751292e-03,
           6.39874065e-03,  0.00000000e+00],
         [ 2.44804121e-03,  2.70751292e-03,  2.99087378e-03,
           1.36597526e-03,  0.00000000e+00],
         [ 3.86189122e-03,  6.39874065e-03,  1.36597526e-03,
           6.65243779e-03,  0.00000000e+00],
         [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
           0.00000000e+00,  1.00000000e-06]],

        [[ 6.17294975e-02,  3.01559817e-02,  2.77432288e-02,
           2.04243829e-02,  0.00000000e+00],
         [ 3.01559817e-02,  5.32654189e-02,  1.91569311e-02,
           2.30532787e-02,  0.00000000e+00],
         [ 2.77432288e-02,  1.91569311e-02,  2.19305243e-02,
           1.75247972e-02,  0.00000000e+00],
         [ 2.04243829e-02,  2.30532787e-02,  1.75247972e-02,
           2.26314778e-02,  0.00000000e+00],
         [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
           0.00000000e+00,  1.00000000e-06]],

        [[ 6.02903297e-02,  2.50294116e-02,  2.02000505e-02,
           1.40828791e-02,  0.00000000e+00],
         [ 2.50294116e-02,  3.37272467e-02,  7.07119172e-03,
           1.56869393e-02,  0.00000000e+00],
         [ 2.02000505e-02,  7.07119172e-03,  1.24292669e-02,
           1.61511490e-03,  0.00000000e+00],
         [ 1.40828791e-02,  1.56869393e-02,  1.61511490e-03,
           3.14836055e-02,  0.00000000e+00],
         [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
           0.00000000e+00,  1.00000000e-06]]]))
```

In [ ]:

choice i=3 that minimizes BIC for full and sphere

In [8]:
```
dataset = pd.read_csv('twomoons.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(linkage='single',affinity="cosine").fit(X)
mean_squared_error(clustering.labels_,y)
```

Out[8]: 0.19

In [9]:
```
clustering.labels_
```

Out[9]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

In [10]:
```
from sklearn.cluster import SpectralClustering
clustering = SpectralClustering(n_clusters=2,assign_labels='discretize',random_state=0,gamma=0.002).fit(X)
mean_squared_error(clustering.labels_,y)
```

Out[10]: 0.77

We can see AgglomerativeClustering with single link and cosine affinity get less error