Introduction to Machine Learning

Homework 4

Author: Xinyu Liu

Performing k-Means By Hand

In [123]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

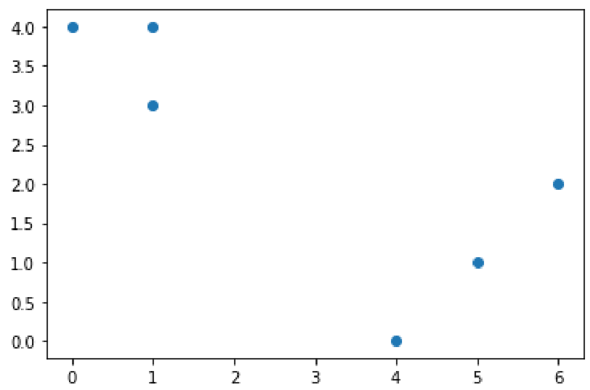
1. Plot the observations.

In [124]:

```
# x <- cbind(c(1, 1, 0, 5, 6, 4), c(4, 3, 4, 1, 2, 0))
sample = pd.DataFrame()
sample['1'] = [1, 1, 0, 5, 6, 4]
sample['2'] = [4, 3, 4, 1, 2, 0]
plt.scatter(sample['1'],sample['2'])
```

Out[124]:

```
<matplotlib.collections.PathCollection at 0x27e55b26f48>
```



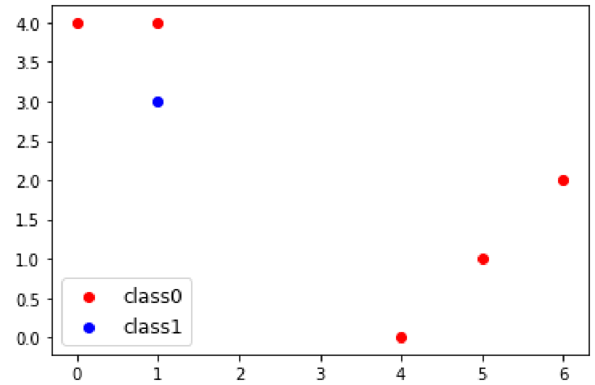
2. Randomly assign a cluster label to each observation. Report the cluster labels for each observation and plot the results with a different color for each cluster (remember to set your seed first).

In [125]:

```
n, p = 1, .5
np.random.seed(1)
random_cluster = np.random.binomial(n, p, size=6)
class0 = sample[random_cluster==0]
class1 = sample[random_cluster==1]
scatter0=plt.scatter(class0['1'],class0['2'],marker='o',c='r')
scatter1=plt.scatter(class1['1'],class1['2'],marker='o',c='b')
plt.legend((scatter0, scatter1),
           ('class0', 'class1'),
           scatterpoints=1,
           loc='lower left',
           ncol=1,
           fontsize=12)
```

Out[125]:

<matplotlib.legend.Legend at 0x27e559e51c8>



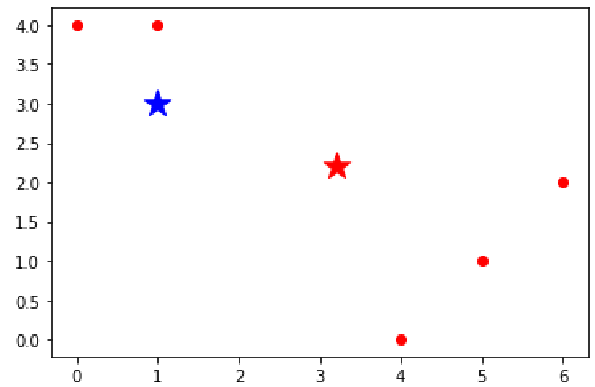
3. Compute the centroid for each cluster.

In [126]:

```
mean0=class0.mean()
mean1=class1.mean()
plt.scatter(class0['1'],class0['2'],marker='o',c='r')
plt.scatter(class1['1'],class1['2'],marker='o',c='b')
plt.scatter(mean0['1'],mean0['2'],marker='*',c='r',s=300)
plt.scatter(mean1['1'],mean1['2'],marker='*',c='b',s=300)
```

Out[126]:

<matplotlib.collections.PathCollection at 0x27e55ce7588>



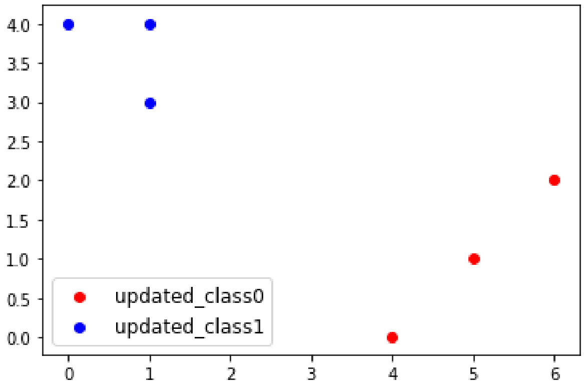
4. Assign each observation to the centroid to which it is closest, in terms of Euclidean distance. Report the cluster labels for each observation.

In [127]:

```
condition = (sample['1']-mean0['1'])**2+(sample['2']-mean0['2'])**2<=(sample['1']-mean1['1'])**2+(sampl
e['2']-mean1['2'])**2
updated_class0 = sample[condition]
updated_class1 = sample[~condition]
scatter0=plt.scatter(updated_class0['1'],updated_class0['2'],marker='o',c='r')
scatter1=plt.scatter(updated_class1['1'],updated_class1['2'],marker='o',c='b')
plt.legend((scatter0, scatter1),
           ('updated_class0', 'updated_class1'),
           scatterpoints=1,
           loc='lower left',
           ncol=1,
           fontsize=12)
```

Out[127]:

<matplotlib.legend.Legend at 0x27e55d3d548>



5. Repeat (3) and (4) until the answers/clusters stop changing.

In [128]:

```
updated_mean0=updated_class0.mean()
updated_mean1=updated_class1.mean()
updated_condition = (sample['1']-updated_mean0['1'])**2+(sample['2']-updated_mean0['2'])**2<=(sample['1
']-updated_mean1['1'])**2+(sample['2']-updated_mean1['2'])**2
updated1_class0 = sample[updated_condition]
updated1_class1 = sample[~updated_condition]
while updated1_class0.equals(updated_class0)!=True:
    updated_class0 = updated1_class0
    updated_class1 = updated1_class1
    updated_mean0=updated_class0.mean()
    updated_mean1=updated_class1.mean()
    updated_condition = (sample['1']-updated_mean0['1'])**2+(sample['2']-updated_mean0['2'])**2<=(sampl
e['1']-updated_mean1['1'])**2+(sample['2']-updated_mean1['2'])**2
    updated1_class0 = sample[updated_condition]
    updated1_class1 = sample[~updated_condition]
print('The answers/clusters stop changing')
```

The answers/clusters stop changing

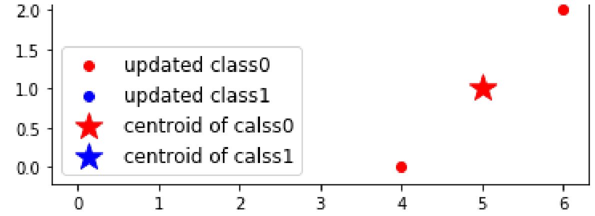
6. Reproduce the original plot from (1), but this time color the observations according to the clusters labels you obtained by iterating the cluster centroid calculation and assignments.

In [129]:

```
scatter0=plt.scatter(updated_class0['1'],updated_class0['2'],marker='o',c='r')
scatter1=plt.scatter(updated_class1['1'],updated_class1['2'],marker='o',c='b')
centroid_of_scatter0=plt.scatter(updated_mean0['1'],updated_mean0['2'],marker='*',c='r',s=300)
centroid_of_scatter1=plt.scatter(updated_mean1['1'],updated_mean1['2'],marker='*',c='b',s=300)

plt.legend((scatter0, scatter1, centroid_of_scatter0, centroid_of_scatter1),
           ('updated class0', 'updated class1','centroid of calss0', 'centroid of calss1'),
           scatterpoints=1,
           loc='lower left',
           ncol=1,
           fontsize=12)
```

Out[129]:

<matplotlib.legend.Legend at 0x27e55e2a948>



Clustering State Legislative Professionalism

1. Load the state legislative professionalism data. See the codebook (or above) for further reference.

In [130]:

```
import pyreadr
rawdata = pyreadr.read_r('D:\\UChicago\\Winter Course\\Machine Learing\\HW4\\problem-set-4\\Data and Co
debook\\legprof-components.v1.0.RData') # also works for Rds
rawdata=rawdata['x']
```

In [131]:

```
rawdata.columns
```

Out[131]:

```
Index(['fips', 'stateabv', 'state', 'sessid', 't_slength', 'slength',
       'salary_real', 'expend', 'year', 'mds1', 'mds2'],
      dtype='object')
```

In [132]:

```
rawdata.head()
```

Out[132]:

	fips	stateabv	state	sessid	t_slength	slength	salary_real	expend	year	mds1	mds2
0	1	AL	Alabama	1973/4	46.000000	36.000000	1.768022	125.097298	1974.0	-1.706181	0.384820
1	1	AL	Alabama	1975/6	110.000000	74.000000	2.933038	203.846588	1976.0	-1.212882	-0.081507
2	1	AL	Alabama	1977/8	83.000000	60.000000	2.082810	184.011520	1978.0	-1.414966	0.127900
3	1	AL	Alabama	1979/80	65.000000	60.000000	1.694951	175.986252	1980.0	-1.543154	0.272688
4	1	AL	Alabama	1981/2	218.680008	149.100006	3.472914	204.123642	1982.0	-0.501364	-1.003878

2. Munge the data:

a. select only the continuous features that should capture a state legislature’s level of “professionalism” (session length (total and regular), salary, and expenditures);

In [133]:

```
rawdata_a=rawdata[['t_slength', 'slength',
        'salary_real', 'expend', 'sessid']]
rawdata_a.describe()
```

Out[133]:

	t_slength	slength	salary_real	expend
count	889.000000	889.000000	945.000000	945.000000
mean	147.598909	136.385219	55.815556	599.507685

	t_slength	slength	salary_real	expend
std	86.9[illegible]	81.1[illegible]	47.1[illegible]	724.1[illegible]
min	36.000000	36.000000	0.000000	40.135218
25%	91.000000	85.199997	20.110354	219.925156
50%	128.510002	120.000000	41.958910	395.104340
75%	171.000000	158.000000	80.081367	650.286011
max	549.540009	521.850006	254.940305	5523.100830

b. restrict the data to only include the 2009/10 legislative session for consistency;

In [134]:

```
rawdata_b=rawdata_a[rawdata_a.sessid.isin(['2009/10'])]
rawdata_b.describe()
```

Out[134]:

	t_slength	slength	salary_real	expend
count	49.000000	49.000000	50.000000	50.000000
mean	147.797958	138.545306	55.905603	746.223404
std	84.076746	73.987710	49.315340	863.386317
min	40.000000	40.000000	0.000000	70.429386
25%	97.419998	93.000000	20.013152	281.785282
50%	127.770000	123.000000	41.953275	538.180916
75%	159.000000	151.229996	83.034459	733.870728
max	458.149994	427.149994	213.405133	5523.100830

c. omit all missing values;

In [135]:

```
rawdata_c=rawdata_b.dropna()
rawdata_c.describe()
```

Out[135]:

	t_slength	slength	salary_real	expend
count	49.000000	49.000000	49.000000	49.000000
mean	147.797958	138.545306	54.991325	744.473014
std	84.076746	73.987710	49.396392	872.243924
min	40.000000	40.000000	0.000000	70.429386
25%	97.419998	93.000000	19.694006	277.078888
50%	127.770000	123.000000	40.328055	535.142319
75%	159.000000	151.229996	77.429867	724.911560
max	458.149994	427.149994	213.405133	5523.100830

In [136]:

```
rawdata_c.head()
```

Out[136]:

	t_slength	slength	salary_real	expend	sessid
18	116.550003	104.550003	1.050421	535.142319	2009/10

	t_slength	slength	salary_real	expend	sessid
37	128.510002	127.800003	74.805863	1493.835083	2009/10
56	286.129990	197.379997	48.393666	631.132935	2009/10
75	80.230000	80.230000	30.669025	516.637619	2009/10
94	390.000000	270.000000	213.405133	5523.100830	2009/10

d. standardize the input features;

In [137]:

```
from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()
column_names_to_normalize = ['t_slength', 'slength', 'salary_real', 'expend']
x = rawdata_c[column_names_to_normalize].values
x_scaled = min_max_scaler.fit_transform(x)
df_temp = pd.DataFrame(x_scaled, columns=column_names_to_normalize, index = rawdata_c.index)
rawdata_d=pd.DataFrame()
rawdata_d[column_names_to_normalize] = df_temp
```

In [138]:

```
rawdata_d.describe()
```

Out[138]:

	t_slength	slength	salary_real	expend
count	49.000000	49.000000	49.000000	49.000000
mean	0.257797	0.254540	0.257685	0.123617
std	0.201068	0.191109	0.231468	0.159966
min	0.000000	0.000000	0.000000	0.000000
25%	0.137319	0.136898	0.092285	0.037899
50%	0.209901	0.214387	0.188974	0.085227
75%	0.284587	0.287305	0.362830	0.120030
max	1.000000	1.000000	1.000000	1.000000

e. and anything else you think necessary to get this subset of data into workable form (hint: consider storing the state names as a separate object to be used in plotting later)

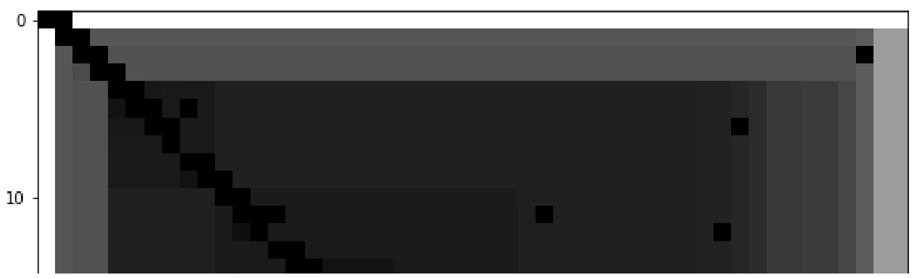
3. Diagnose clusterability in any way you'd prefer (e.g., sparse sampling, ODI, etc.); display the results and discuss the likelihood that natural, non-random structure exist in these data.

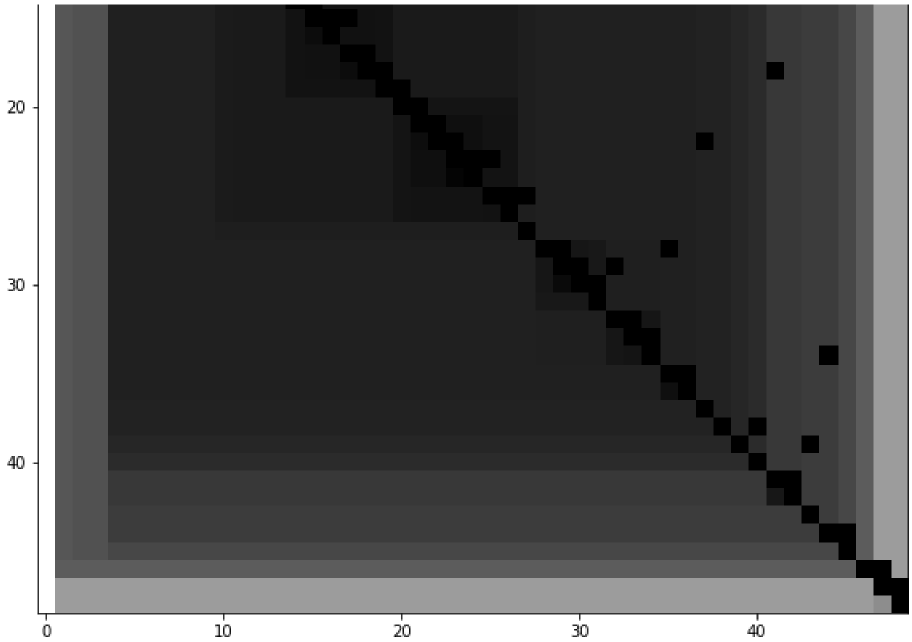
In [139]:

```
from pyclustertend import ivat
```

In [140]:

```
ivat(rawdata_d[column_names_to_normalize])
```



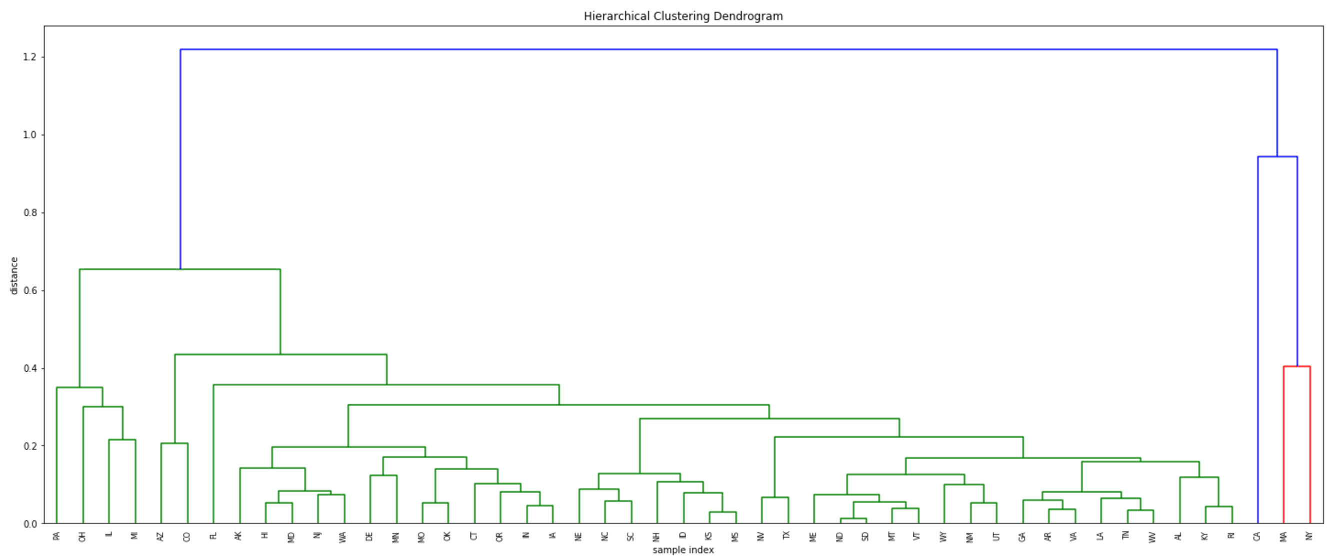


Discussion: from the visualization we can tell that there is a natural tendency of clustering.

4. Fit an agglomerative hierarchical clustering algorithm using any linkage method you prefer, to these data and present the results. Give a quick, high level summary of the output and general patterns.

In [141]:

```
from scipy.cluster.hierarchy import dendrogram, linkage
# https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/
data4=rawdata_d.merge(rawdata['stateabv'], how = 'left', left_index=True, right_index=True).set_index('
stateabv')
# generate the linkage matrix
Z = linkage(data4, 'average')
# calculate full dendrogram
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8., # font size for the x axis labels
    labels=data4.index.values
)
plt.show()
```



In [142]:

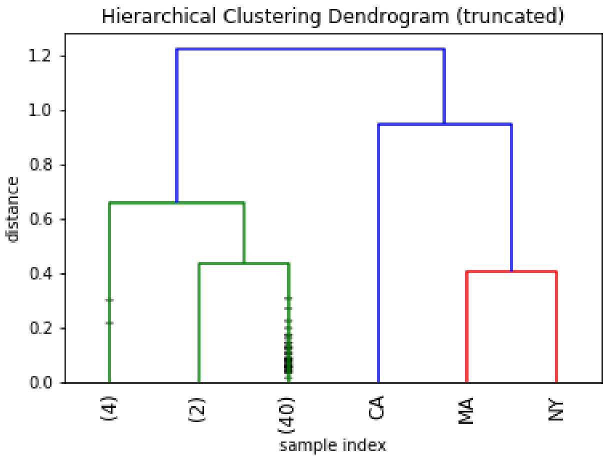
```
## This (very very briefly) compares (correlates) the actual pairwise distances of
## all your samples to those implied by the hierarchical clustering. The closer the value is to 1,
## the better the clustering preserves the original distances
from scipy.cluster.hierarchy import cophenet
from scipy.spatial.distance import pdist

c, coph_dists = cophenet(Z, pdist(data4))
c
```

Out[142]:

```
0.9292124863441844
```

In [143]:

```
## Dendrogram Truncation
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=6,  # show only the last p merged clusters
    show_leaf_counts=True,  # otherwise numbers in brackets are counts
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
    labels=data4.index.values
)
plt.show()
```



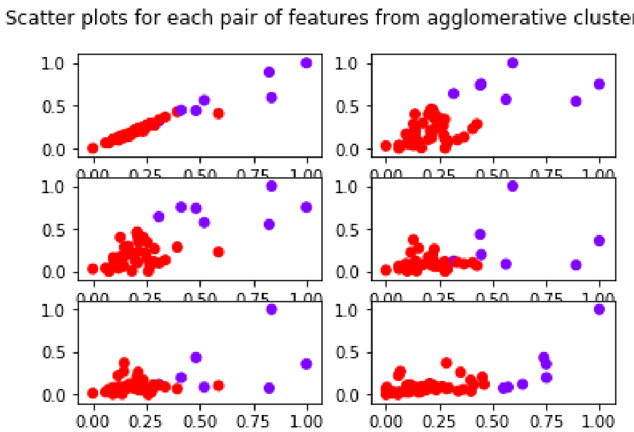
In [144]:

```
from sklearn.cluster import AgglomerativeClustering
hierarchical = AgglomerativeClustering().fit(data4)
hlabels = hierarchical.labels_
hierarchical_result = pd.DataFrame(index = data4.index)
hierarchical_result['hierarchical_classification']=hlabels
print('One of the class contains '+ str(hierarchical_result[hierarchical_result.hierarchical_classifica
tion==0].index.values))
fig, axs = plt.subplots(3, 2)
axs[0, 0].scatter(data4['t_slength'], data4['slength'], c=hlabels, cmap='rainbow')
axs[1, 0].scatter(data4['t_slength'], data4['salary_real'], c=hlabels, cmap='rainbow')
axs[2, 0].scatter(data4['t_slength'], data4['expend'], c=hlabels, cmap='rainbow')
axs[0, 1].scatter(data4['slength'], data4['salary_real'], c=hlabels, cmap='rainbow')
axs[1, 1].scatter(data4['slength'], data4['expend'], c=hlabels, cmap='rainbow')
axs[2, 1].scatter(data4['salary_real'], data4['expend'], c=hlabels, cmap='rainbow')
fig.suptitle('Scatter plots for each pair of features from agglomerative clustering')
```

```
One of the class contains ['CA' 'IL' 'MA' 'MI' 'NY' 'OH' 'PA']
```

Out[144]:

```
Text(0.5, 0.98, 'Scatter plots for each pair of features from agglomerative clustering')
```



Discussion: Note that given different linkage methods, the clustering process can be extremely different. In the method of 'average' linkage, CA, MA and NY appear to be different from other state at the root level, signaling their difference in the professionalism status. In the method of 'ward'linkage, 'CA' 'IL' 'MA' 'MI' 'NY' 'OH' 'PA' are of the same class.

5. Fit a k-means algorithm to these data and present the results. Give a quick, high level summary of the output and general patterns. Initialize the algorithm at k = 2, and then check this assumption in the validation questions below.

In [145]:

```
data4.head()
```

Out[145]:

	t_slength	slength	salary_real	expend
stateabv				
AL	0.183068	0.166731	0.004922	0.085227
AK	0.211670	0.226785	0.350535	0.261047
AZ	0.588617	0.406509	0.226769	0.102831
AR	0.096209	0.103913	0.143713	0.081833
CA	0.837020	0.594085	1.000000	1.000000

In [146]:

```
from sklearn.cluster import KMeans
kmeans2 = KMeans(n_clusters=2)
y_kmeans2 = kmeans2.fit_predict(data4)

kmean_result = pd.DataFrame(index = data4.index)
kmean_result['kmean_classification']=y_kmeans2
print('One of the class contains '+ str(kmean_result[kmean_result.kmean_classification==1].index.values
))
print('the centers for each class are: ' + str(kmeans2.cluster_centers_))
```

```
One of the class contains ['CA' 'MA' 'MI' 'NY' 'OH' 'PA']
the centers for each class are: [[0.19887878 0.19850188 0.19209605 0.09085658]
 [0.68004704 0.65614964 0.72774008 0.35840113]]
```

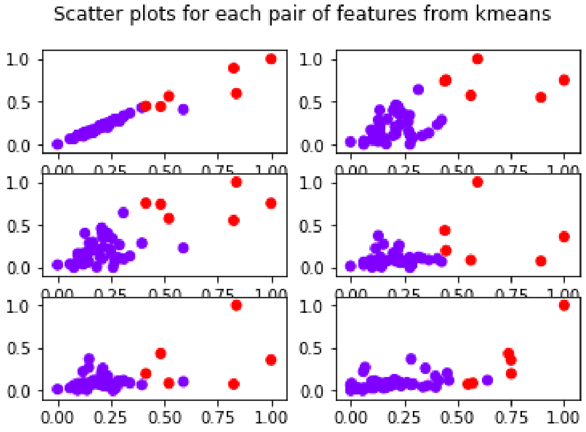
In [147]:

```
fig, axs = plt.subplots(3, 2)
axs[0, 0].scatter(data4['t_slength'], data4['slength'], c=y_kmeans2, cmap='rainbow')
```

```
axs[1, 0].scatter(data4['t_slength'], data4['salary_real'], c=y_kmeans2, cmap='rainbow')
axs[2, 0].scatter(data4['t_slength'], data4['expend'], c=y_kmeans2, cmap='rainbow')
axs[0, 1].scatter(data4['slength'], data4['salary_real'], c=y_kmeans2, cmap='rainbow')
axs[1, 1].scatter(data4['slength'], data4['expend'], c=y_kmeans2, cmap='rainbow')
axs[2, 1].scatter(data4['salary_real'], data4['expend'], c=y_kmeans2, cmap='rainbow')
fig.suptitle('Scatter plots for each pair of features from kmeans')
```

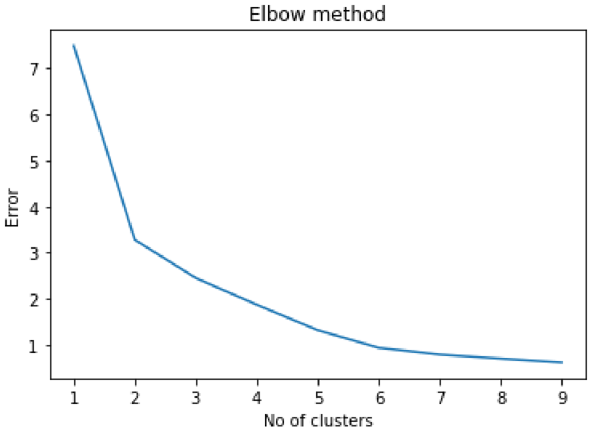
Out[147]:

```
Text(0.5, 0.98, 'Scatter plots for each pair of features from kmeans')
```



In [148]:

```
Error =[]
for i in range(1, 10):
    kmeans = KMeans(n_clusters = i).fit(data4)
    Error.append(kmeans.inertia_)
import matplotlib.pyplot as plt
plt.plot(range(1, 10), Error)
plt.title('Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```



Disucssion: From the elbow method we can conclude that k=2 is relatively a better choice. From the classification centers we can tell that professional jurisdiction tends to better perform other states on all of the four features.

6. Fit a Gaussian mixture model via the EM algorithm to these data and present the results. Give a quick, high level summary of the output and general patterns. Initialize the algorithm at k = 2, and then check this assumption in the validation questions below.

In [149]:

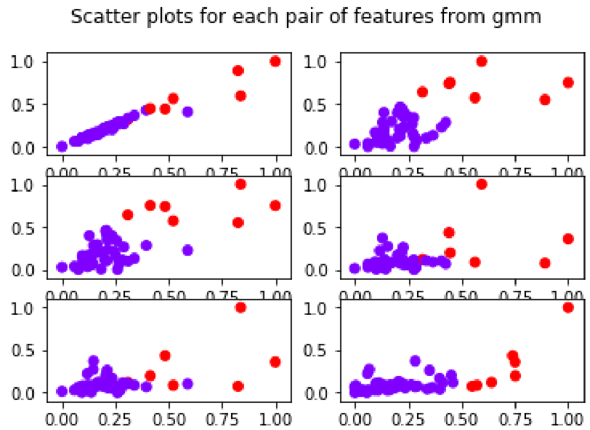
```
# https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=2).fit(data4)
```

```
labels = gmm.predict(data4)
gmm_result = pd.DataFrame(index = data4.index)
gmm_result['gmm_classification']=labels
print('One of the class contains '+ str(gmm_result[gmm_result.gmm_classification==1].index.values))
print('the centers for each class are: ' + str(gmm.means_))
# gmm.predict_proba(data4)
fig, axs = plt.subplots(3, 2)
axs[0, 0].scatter(data4['t_slength'], data4['slength'], c=labels, cmap='rainbow')
axs[1, 0].scatter(data4['t_slength'], data4['salary_real'], c=labels, cmap='rainbow')
axs[2, 0].scatter(data4['t_slength'], data4['expend'], c=labels, cmap='rainbow')
axs[0, 1].scatter(data4['slength'], data4['salary_real'], c=labels, cmap='rainbow')
axs[1, 1].scatter(data4['slength'], data4['expend'], c=labels, cmap='rainbow')
axs[2, 1].scatter(data4['salary_real'], data4['expend'], c=labels, cmap='rainbow')
fig.suptitle('Scatter plots for each pair of features from gmm')

# Check all properties
# vars(gmm)
```

```
One of the class contains ['CA' 'IL' 'MA' 'MI' 'NY' 'OH' 'PA']
the centers for each class are: [[0.19655289 0.19597313 0.18255725 0.09023843]
 [0.63179213 0.61218657 0.71646013 0.32744723]]
```

Out[149]:

```
Text(0.5, 0.98, 'Scatter plots for each pair of features from gmm')
```



7. Compare output of all in visually useful, simple ways (e.g., present the dendrogram, plot by state cluster assignment across two features like salary and expenditures, etc.). There should be several plots of comparison and output.

Discussion: as is shown above, the outputs of kmeans and hierarchical are identical but for gmm method recognizes 'IL' apart from the six in the other two methods.

8. Select a single validation strategy (e.g., compactness via min(WSS), average silhouette width, etc.), and calculate for all three algorithms. Display and compare your results for all three algorithms you fit (hierarchical, k-means, GMM). Hint: Here again, we didn't cover this in R in class, but think about using the clValid package, though there are many other packages and ways to validate cluster patterns across iterations.

In [195]:

```
# Here I am comparing WSS across three methods:

#######################
# 1. agglomerative    #
#######################
mean0 = data4[hierarchical_result.hierarchical_classification==0].mean()
mean1 = data4[hierarchical_result.hierarchical_classification==1].mean()
agglomerative_wss = ((data4[hierarchical_result.hierarchical_classification==0]-mean0)**2).sum().sum()+
\
            ((data4[hierarchical_result.hierarchical_classification==1]-mean1)**2).sum().sum()
print('WSS of agglomerative clustering (k=2) is {}'.format(agglomerative_wss))
#############
# 2. Kmeans #
#############
```

```
#############
kmeans2 = KMeans(n_clusters = 2).fit(data4)
kmeans2.inertia_
print('WSS of kmean (k=2) is {}'.format(kmeans2.inertia_))
#############
# 3. gmm    #
#############
gmm_wss = ((data4[gmm_result.gmm_classification==0]-gmm.means_[0])**2).sum().sum()+\
            ((data4[gmm_result.gmm_classification==1]-gmm.means_[1])**2).sum().sum()
print('WSS of gmm (k=2) is {}'.format(gmm_wss))

print('The method with smallest WSS is kmeans of {}'.format(min(agglomerative_wss, kmeans2.inertia_, gm
m_wss)))
```

```
WSS of agglomerative clustering (k=2) is 3.321908124470716
WSS of kmean (k=2) is 3.2842452937295263
WSS of gmm (k=2) is 3.32234463635682
The method with smallest WSS is kmeans of 3.2842452937295263
```

9. Discuss the validation output, e.g.,

- What can you take away from the fit?
- Which approach is optimal? And optimal at what value of k?
- What are reasons you could imagine selecting a technically “sub-optimal” clustering method, regardless of the validation statistics?

First, different approaches can give different results, in this case we know we wanna deal with the case where k=2 but it may take extra effort to also tune the k in general unsupervised clustering problems. Second, accorrding to the WSS it seems that the method with smallest WSS is kmeans at k=2, which is further confirmed by the elbow plot above. Third, there can for instance be realistic judgement on what the reasonable clustering method should be. In class we discuss the implementation of gmm that improves the performance at margin samples. Also as mentioned before it is more important to find the elbow k value than trying to get as lowest WSS as we can because of concern of overfitting.