8 : EM, K-Means algorithm to cluster a set of data and comparison

K Means

load a sample dataset

```
from sklearn import datasets
iris_dataset = datasets.load_iris()


iris_dataset.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
import pandas

inputs = pandas.DataFrame(iris_dataset.data)

# Assigning column names to inputs
inputs.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width' ]

outputs = pandas.DataFrame(iris_dataset.target)

# Assigning column names to outputs
outputs.columns = ['targets' ]
```

fit the model on inputs

```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3)    # 3 centroids


model.fit(inputs)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

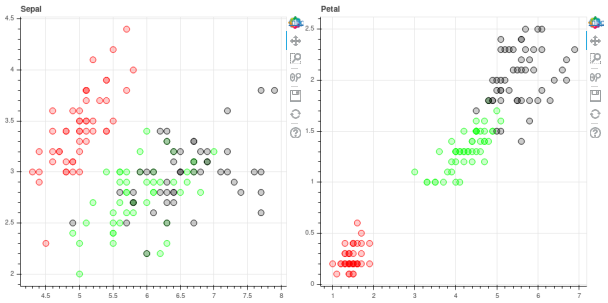
plotter helper function

```
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import row
output_notebook()


def custom_plotter( first_list, second_list, labels,  colormap_outputs):

    colormap =   ['red', 'lime', 'black']

    first_label, second_label = labels
    first_output, second_output = colormap_outputs



    colors = [ colormap[x] for x in  first_output]

    figure1 = figure(width=500, plot_height=500, title = first_label)
    figure1.circle( *first_list, color=colors, fill_alpha=0.2, size=10)
    plot = figure1

    empty = None
    if  second_list is not empty:

        colors = [ colormap[x] for x in  second_output]

        figure2 = figure(width=500, plot_height=500, title = second_label)
        figure2.circle( *second_list, color=colors, fill_alpha=0.2, size=10)

        plot = row(figure1, figure2)

    show(plot)
```

```
'''                         old
def custom_plotter( first_list, second_list, labels,  colormap_outputs):

    # Note: List unpacking is done at scatter method

    first_label, second_label = labels
    first_output, second_output = colormap_outputs

    import matplotlib.pyplot as plt

    plt.figure(figsize=(14,7))    # better if figure is resized


    # Create a colormap  of 3 colors for clusters ( centroids )
    # If you do not pass colormap, plot will be in uni color
    import numpy
    colormap = numpy.array(['red', 'lime', 'black'])



    # ploting first_list

    plt.subplot(1, 2, 1)     # creates a portion in a figure   # Note : 1,2 is common
    plt.scatter( *first_list, c = colormap[first_output] )
    plt.title( first_label)
                             # give a title to the figure

    empty = None
    if  second_list is not empty:    # for plotting single figure ( essential in later section )


        # similarly ploting second list
        plt.subplot(1, 2, 2)
        plt.scatter( *second_list,  c = colormap[second_output] )
        plt.title( second_label )
'''
```

Plotting based on the outputs from the dataset

```
figure1_inputs = [ inputs.sepal_length, inputs.sepal_width ]
figure2_inputs = [ inputs.petal_length, inputs.petal_width ]
labels = ['Sepal', 'Petal']
color_mapper = [outputs.targets, outputs.targets]



custom_plotter( figure1_inputs, figure2_inputs, labels, color_mapper)
```

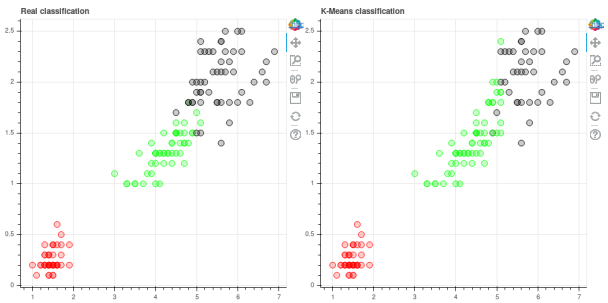


assigning predictions of a model to a equivalent variable

```
predictions = model.labels_
```

Plotting based on the predictions from the model

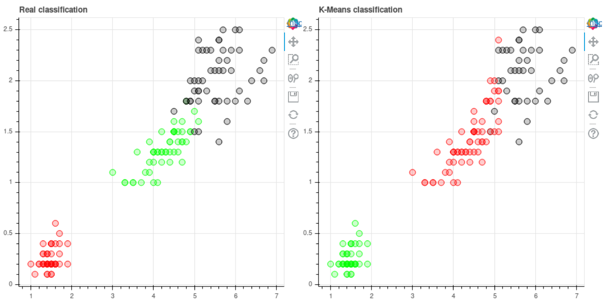
```
figure1_inputs = [ inputs.petal_length, inputs.petal_width ]
figure2_inputs = [ inputs.petal_length, inputs.petal_width ]  #same inputs
labels  = ['Real classification', 'K-Means classification']
color_mapper = [outputs.targets, predictions]                 # predictions



custom_plotter( figure1_inputs, figure2_inputs, labels, color_mapper)
```



flip predictions (Interchanging 1 and 0) and re plotting (This can be ignored)

```python
import numpy
flipped_predictions = numpy.choose(predictions, [1, 0, 2]).astype(numpy.int64)

print('predictions \n', predictions, end='\n'*3)
print('flipped predictions \n', flipped_predictions, end='\n'*3)



figure1_inputs = [ inputs.petal_length, inputs.petal_width ]
figure2_inputs = [ inputs.petal_length, inputs.petal_width ]
labels  = ['Real classification', 'K-Means classification']
color_mapper = [outputs.targets, flipped_predictions]         # flipped predictions


# color will  be inverted
custom_plotter( figure1_inputs, figure2_inputs, labels, color_mapper)
```

```
predictions
 [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 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]


flipped predictions
 [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 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```



Accuracy measure of K-means model

```python
from sklearn import  metrics

percent = metrics.accuracy_score(outputs, predictions) * 100

print('Accuracy of K Means : {0:.2f}%'.format(percent))
```

Accuracy of K Means : 89.33%

GMM Gausian Mixture Model

prepare a dataframe to fit

```python
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaler.fit(inputs)

# pandas is already imported without aliasing, hence not pd

scaled_data = scaler.transform(inputs)
dataframe = pandas.DataFrame(scaled_data, columns = inputs.columns)


# Inspecting dataframe
dataframe.sample(5)
```

	sepal_length	sepal_width	petal_length	petal_width
29	-1.385353	0.328414	-1.226552	-1.315444
70	0.068662	0.328414	0.592246	0.790671
142	-0.052506	-0.822570	0.762758	0.922303
76	1.159173	-0.592373	0.592246	0.264142
16	-0.537178	1.939791	-1.397064	-1.052180

fit dataframe to gmm (gausian_mixture_model)

```python
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture( n_components=3 )       # Note : parameter is not cluster
gmm.fit(dataframe)
```

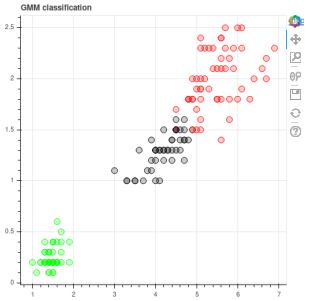
```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
        means_init=None, n_components=3, n_init=1, precisions_init=None,
        random_state=None, reg_covar=1e-06, tol=0.001, verbose=0,
        verbose_interval=10, warm_start=False, weights_init=None)
```

obtaining predections from gausian_mixture_model

```python
gmm_predictions = gmm.predict(dataframe)
```

plotting based on predictions from gausian_mixture_model

```
figure1_inputs = [ inputs.petal_length, inputs.petal_width ]
figure2_inputs = None
labels  = ['GMM classification', None]
color_mapper = [gmm_predictions, None]         # gmm_predictions



custom_plotter( figure1_inputs, figure2_inputs, labels, color_mapper)
```



Accuracy measure of gausian mixture model

```
percent = metrics.accuracy_score(outputs, gmm_predictions) * 100

# No idea why it is 30
print('Accuracy of Gausian Mixture model: {0:.2f}%'.format(percent))
```

```
Accuracy of Gausian Mixture model: 0.00%
```

```

```