GMM Clustering and Cluster Validation Lab

May 18, 2019

0.1 1. KMeans vs GMM on a Generated Dataset

In the first example we'll look at, we'll generate a Gaussian dataset and attempt to cluster it and see if the clustering matches the original labels of the generated dataset.

We can use sklearn's make_blobs function to create a dataset of Gaussian blobs:

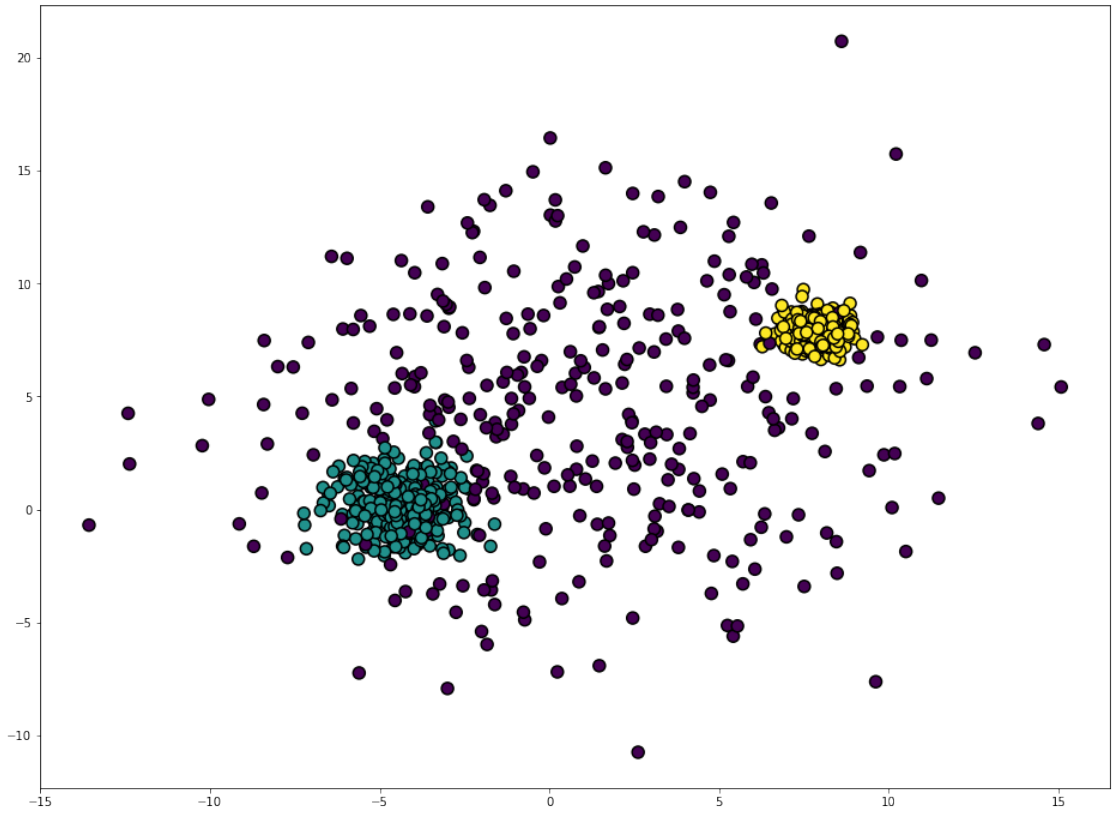
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import cluster, datasets, mixture

        %matplotlib inline

        n_samples = 1000

        varied = datasets.make_blobs(n_samples=n_samples,
                                     cluster_std=[5, 1, 0.5],
                                     random_state=3)
        X, y = varied[0], varied[1]

        plt.figure( figsize=(16,12))
        plt.scatter(X[:,0], X[:,1], c=y, edgecolor='black', lw=1.5, s=100, cmap=plt.get_cmap('
        plt.show()
```

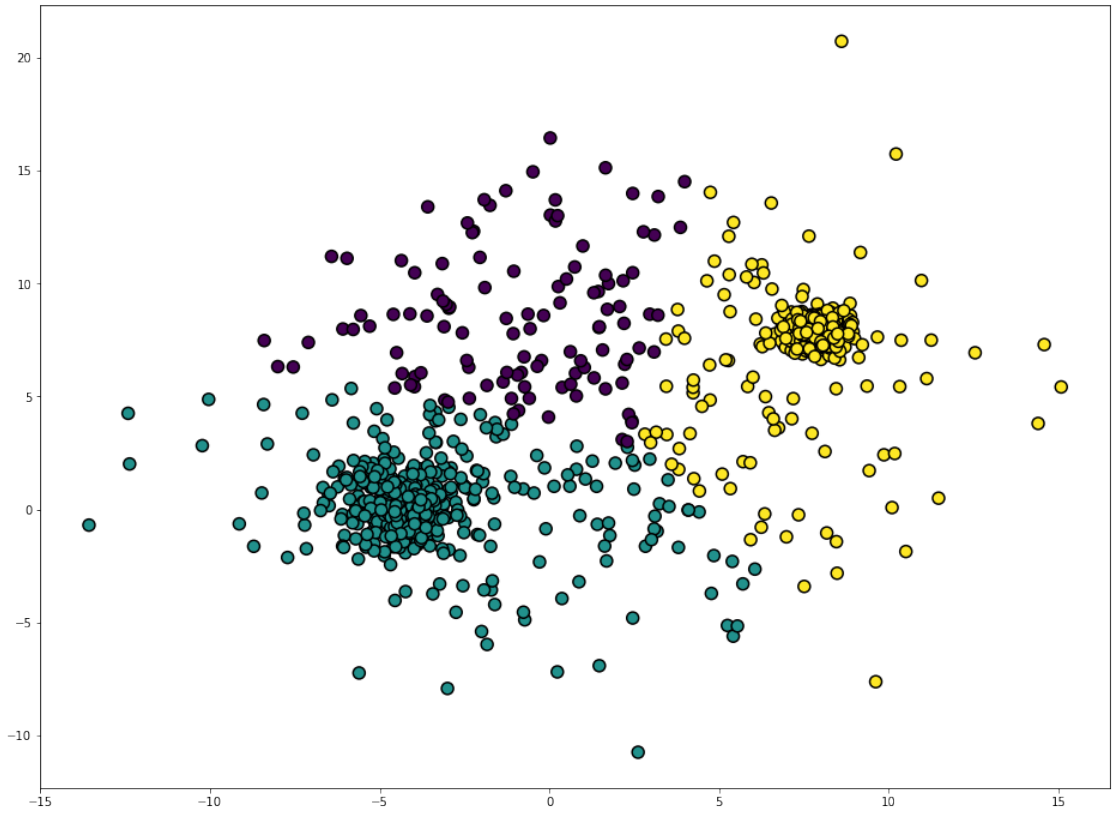


Now when we hand off this dataset to the clustering algorithms, we obviously will not pass in the labels. So let's start with KMeans and see how it does with the dataset. WIll it be to produce clusters that match the original labels?

```
In [2]: from sklearn.cluster import KMeans

        kmeans = KMeans(n_clusters=3)
        pred = kmeans.fit_predict(X)

In [3]: plt.figure( figsize=(16,12))
        plt.scatter(X[:,0], X[:,1], c=pred, edgecolor='black', lw=1.5, s=100, cmap=plt.get_cma
        plt.show()
```



How good of a job did KMeans do? Was it able to find clusters that match or are similar to the original labels?

Let us now try clustering with GaussianMixture:

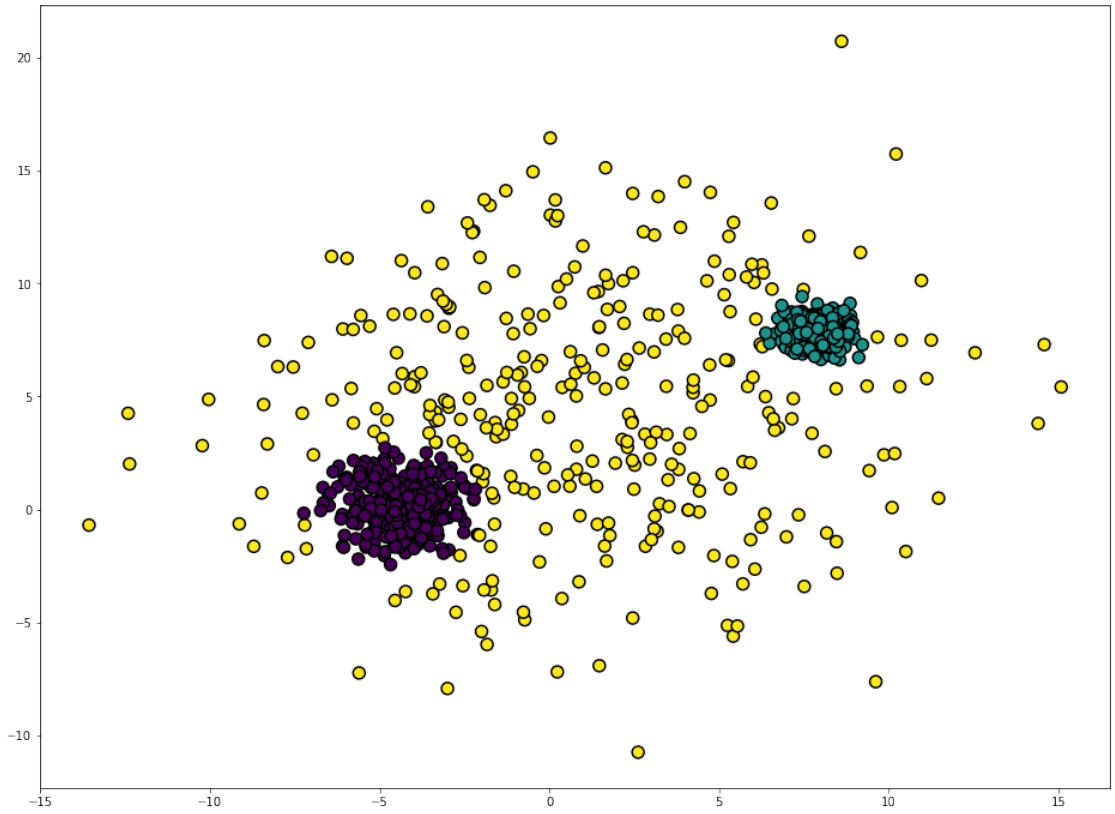
```
In [4]: # TODO: Import GaussianMixture
        from sklearn.mixture import GaussianMixture

        # TODO: Create an instance of Gaussian Mixture with 3 components
        gmm = GaussianMixture(n_components=3).fit(X)

        # TODO: fit the dataset
        gmm = gmm.fit(X)

        # TODO: predict the clustering labels for the dataset
        pred_gmm = gmm.predict(X)

In [5]: # Plot the clusters
        plt.figure( figsize=(16,12))
        plt.scatter(X[:,0], X[:,1], c=pred_gmm, edgecolor='black', lw=1.5, s=100, cmap=plt.get_
        plt.show()
```



By visually comparing the result of KMeans and GMM clustering, which one was better able to match the original?

1 2. KMeans vs GMM on The Iris Dataset

For our second example, we'll take a dataset that has more than two features. The Iris dataset is great for this purpose since it is reasonable to assume it's distributed according to Gaussian distributions.

The Iris dataset is a labeled dataset with four features:

```
In [6]: import seaborn as sns

        iris = sns.load_dataset("iris")

        iris.head()
```

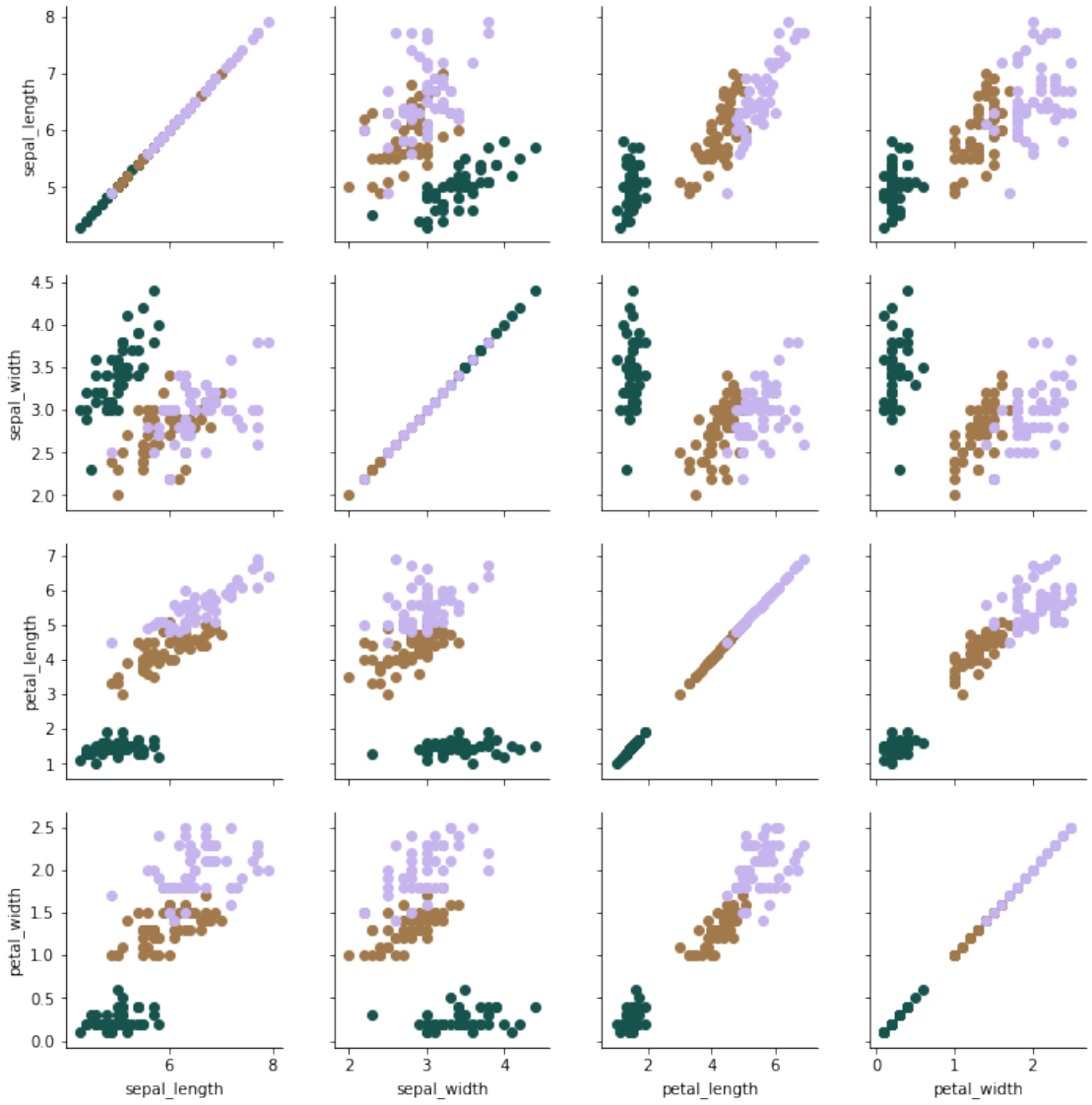
Out[6]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

How do you visualize a datset with four dimensions?

There are a few ways (e.g. PairGrid, t-SNE, or project into a lower number number dimensions using PCA). Let's attempt to visualize using PairGrid because it does not distort the dataset -- it merely plots every pair of features against each other in a subplot:

```
In [7]: g = sns.PairGrid(iris, hue="species", palette=sns.color_palette("cubehelix", 3), vars=
        g.map(plt.scatter)
        plt.show()
```

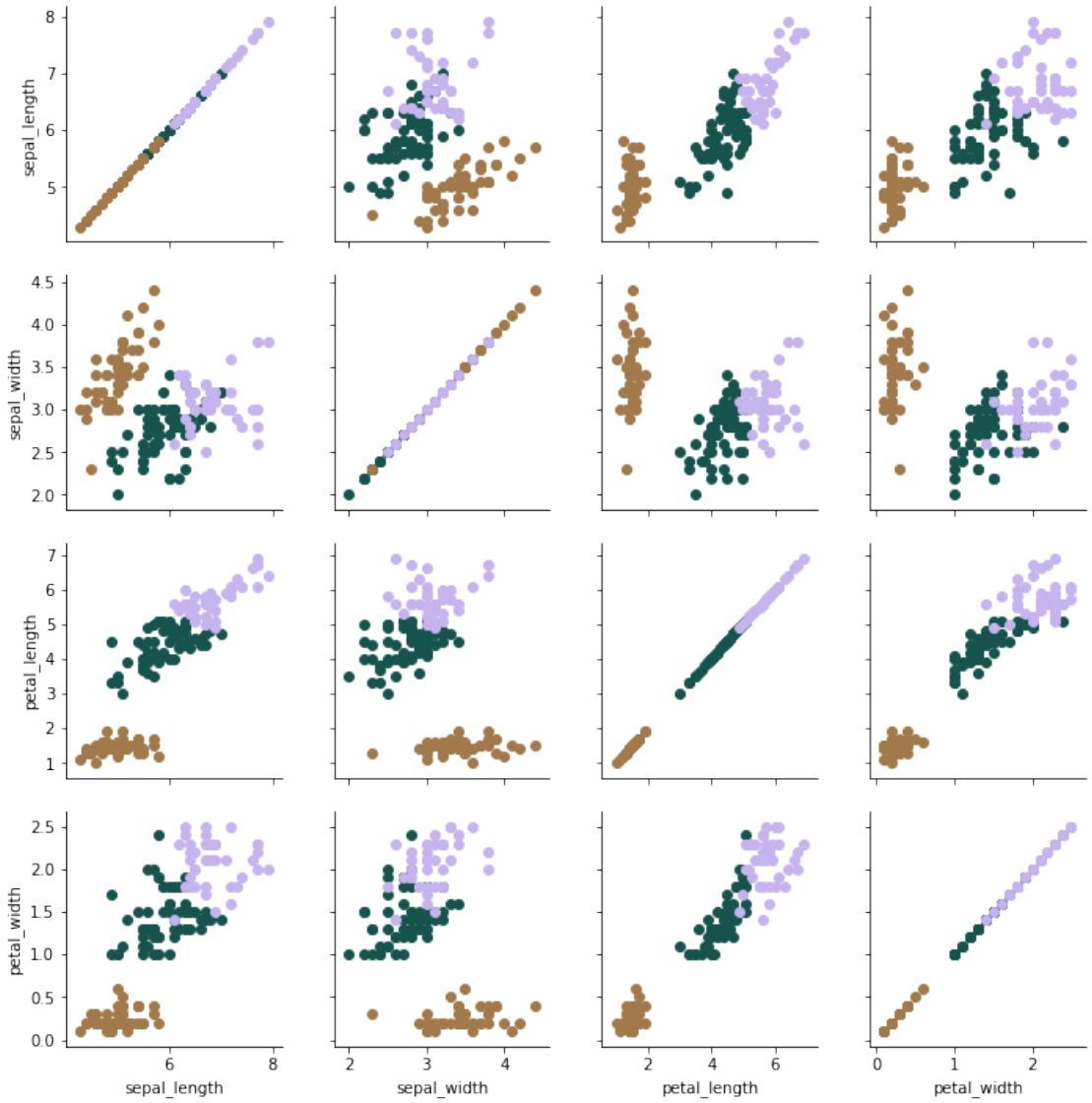


If we cluster the Iris datset using KMeans, how close would the resulting clusters match the original labels?

```
In [8]: kmeans_iris = KMeans(n_clusters=3)
        pred_kmeans_iris = kmeans_iris.fit_predict(iris[['sepal_length','sepal_width','petal_le
```

```
In [9]: iris['kmeans_pred'] = pred_kmeans_iris

        g = sns.PairGrid(iris, hue="kmeans_pred", palette=sns.color_palette("cubehelix", 3), va
        g.map(plt.scatter)
        plt.show()
```



How do these clusters match the original labels?

You can clearly see that visual inspection is no longer useful if we're working with multiple dimensions like this. So how can we evaluate the clustering result versus the original labels?

You guessed it. We can use an external cluster validation index such as the adjusted Rand score which generates a score between -1 and 1 (where an exact match will be scored as 1).

```
In [10]: # TODO: Import adjusted rand score
         from sklearn.metrics import adjusted_rand_score
```

```
        # TODO: calculate adjusted rand score passing in the original labels and the kmeans p
        iris_kmeans_score = adjusted_rand_score(iris['species'], iris['kmeans_pred'])

        # Print the score
        iris_kmeans_score
```

```
Out[10]: 0.7302382722834697
```

What if we cluster using Gaussian Mixture models? Would it earn a better ARI score?

```
In [11]: gmm_iris = GaussianMixture(n_components=3).fit(iris[['sepal_length','sepal_width','pe
         pred_gmm_iris = gmm_iris.predict(iris[['sepal_length','sepal_width','petal_length','p
```

```
In [12]: iris['gmm_pred'] = pred_gmm_iris

         # TODO: calculate adjusted rand score passing in the original
         # labels and the GMM predicted labels iris['species']
         iris_gmm_score = adjusted_rand_score(iris['species'], iris['gmm_pred'])

         # Print the score
         iris_gmm_score
```

```
Out[12]: 0.9038742317748124
```

Thanks to ARI socres, we have a clear indicator which clustering result better matches the original dataset.