Lecture 15 Unsupervised Learning: Clustering

In addition to **dimension reduction**, another typical task of unsupervised machine learning is **clustering**: assigning the data samples into several groups (called clusters) based on their similarity -- similar samples should be in the same cluster, and dissimilar samples should be in different clusters.

Caution: Don't confuse clustering (unsupervised) with classification (supervised)!

K-Means Clustering (https://en.wikipedia.org/wiki/K-means_clustering)

Mathematical Description: Given a set of observations $(x^{(1)}, x^{(2)}, \dots, x^{(n)})$, where each observation is a p-dimensional real vector, k-means clustering aims to partition the n samples into $K (\leq n)$ sets $S = S_1, S_2, \dots, S_K$ so as to minimize the within-cluster sum of squares (i.e. variance). Formally, the objective is to find the best parition of groups such that minimize the "loss function" of S

$$\min_{S} \sum_{i=1}^{K} \sum_{x \in S_i} \|x - \mu_i\|^2$$

where μ_i is the mean of points in S_i.

How to solve it: The exact solution to the K-means problem is NP hard (https://en.wikipedia.org/wiki/P_versus_NP_problem). In practical, the common approach is to apply Lloyd's algorithm (https://en.wikipedia.org/wiki/Lloyd%27s_algorithm) to find the heuristic solutions (local minimum).

In the iterative algorithm, each iteration contains two steps:

- (**assignment step**) Given the cluster center, update the cluster of data according to its nearest cluster center.
- (**update step**) Given the cluster assignment, update the cluster centers by calculating the means within each cluster.

The algorithm may converge if no further adjustments can be made.

Because the algorithm may stuck in local minimums, the practical strategy is to randomly initialize the algorithm, and run multiple parallel programs to find the best partition with smallest "loss function ".

Caution: Don't confuse K-means clustering with kNN classification!

Below we will apply the functions in scikit-learn (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html). Note that to visualize the results of k-means on high-dimensional data, it is often combined with dimension reductions. Another pratical strategy is to use dimension reduction to pre-process the data, and apply clustering on the reduced datasets.

It is also worth noting that in practice, determining true number of clusters (K) is a very hard problem (https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set).

In [1]:

```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y = True)

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0)
y_km = kmeans.fit_predict(X)
```

In [2]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

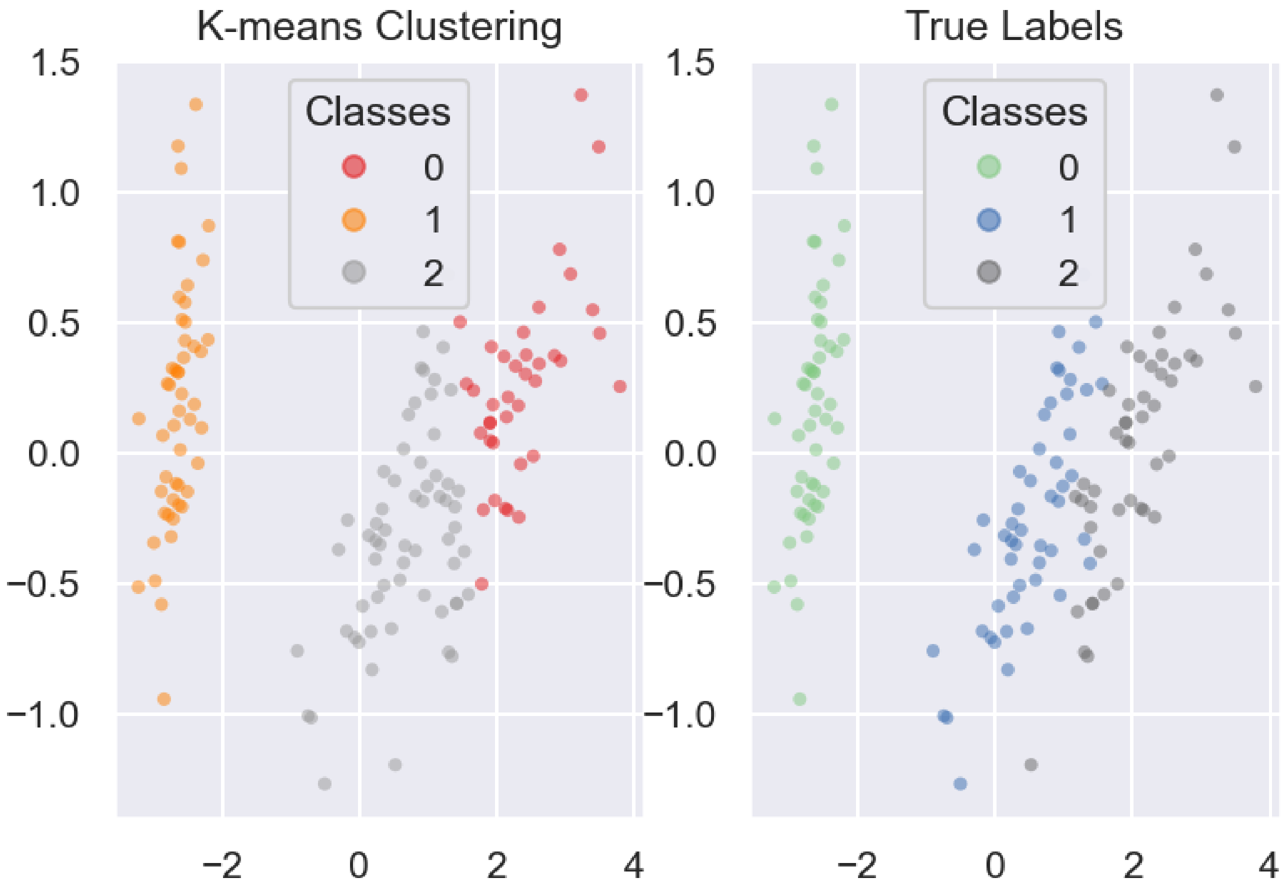
In [3]:

```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
fig, (ax1, ax2) = plt.subplots(1, 2,dpi=150)

fig1 = ax1.scatter(X_pca[:, 0], X_pca[:, 1],c=y_km, s=15, edgecolor='none', alpha=0.5
,cmap=plt.cm.get_cmap('Set1', 3))
fig2 = ax2.scatter(X_pca[:, 0], X_pca[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,cm
ap=plt.cm.get_cmap('Accent', 3))
ax1.set_title('K-means Clustering')
legend1 = ax1.legend(*fig1.legend_elements(), loc="best", title="Classes")
ax1.add_artist(legend1)
ax2.set_title('True Labels')
legend2 = ax2.legend(*fig2.legend_elements(), loc="best", title="Classes")
ax2.add_artist(legend2)
```

Out[3]:

```
<matplotlib.legend.Legend at 0x7fc55ee10750>
```



To quantitatively measure the performace, it is not a good idea to naively compute "accuracy" as in the classification case, because permutation of the label values does not affect clustering results, while severely affects the "accuracy". There are many good measures (https://scikit-learn.org/stable/modules/clustering.html) considering such effects in the clustering.

In [4]:

```
from sklearn import metrics
metrics.adjusted_rand_score(y_km, y)
```

Out[4]:

```
0.7302382722834697
```

In [5]:

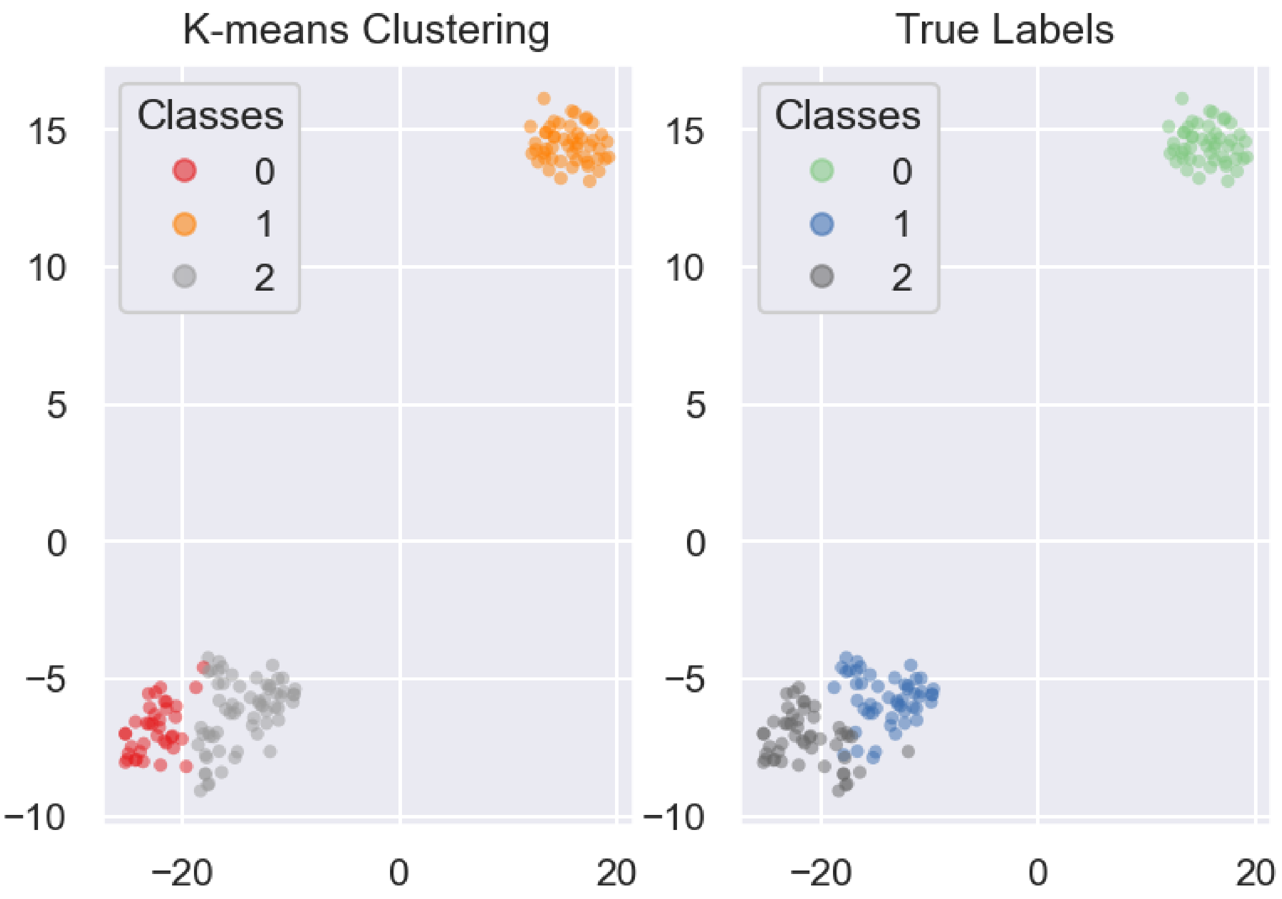
```
from sklearn.manifold import TSNE
tsne = TSNE(random_state=0, n_jobs = -1)
X_tsne = tsne.fit_transform(X)


fig, (ax1, ax2) = plt.subplots(1, 2,dpi=150)

fig1 = ax1.scatter(X_tsne[:, 0], X_tsne[:, 1],c=y_km, s=15, edgecolor='none', alpha=
0.5,cmap=plt.cm.get_cmap('Set1', 3))
fig2 = ax2.scatter(X_tsne[:, 0], X_tsne[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,
cmap=plt.cm.get_cmap('Accent', 3))
ax1.set_title('K-means Clustering')
legend1 = ax1.legend(*fig1.legend_elements(), loc="best", title="Classes")
ax1.add_artist(legend1)
ax2.set_title('True Labels')
legend2 = ax2.legend(*fig2.legend_elements(), loc="best", title="Classes")
ax2.add_artist(legend2)
```

Out[5]:

```
<matplotlib.legend.Legend at 0x7fc55729a3d0>
```



K-means Clustering

True Labels

How about clustering on TSNE results?

In [7]:

```
y_km_tsne = kmeans.fit_predict(X_tsne)
metrics.adjusted_rand_score(y_km_tsne, y)
```

Out[7]:

```
0.7726314170414115
```

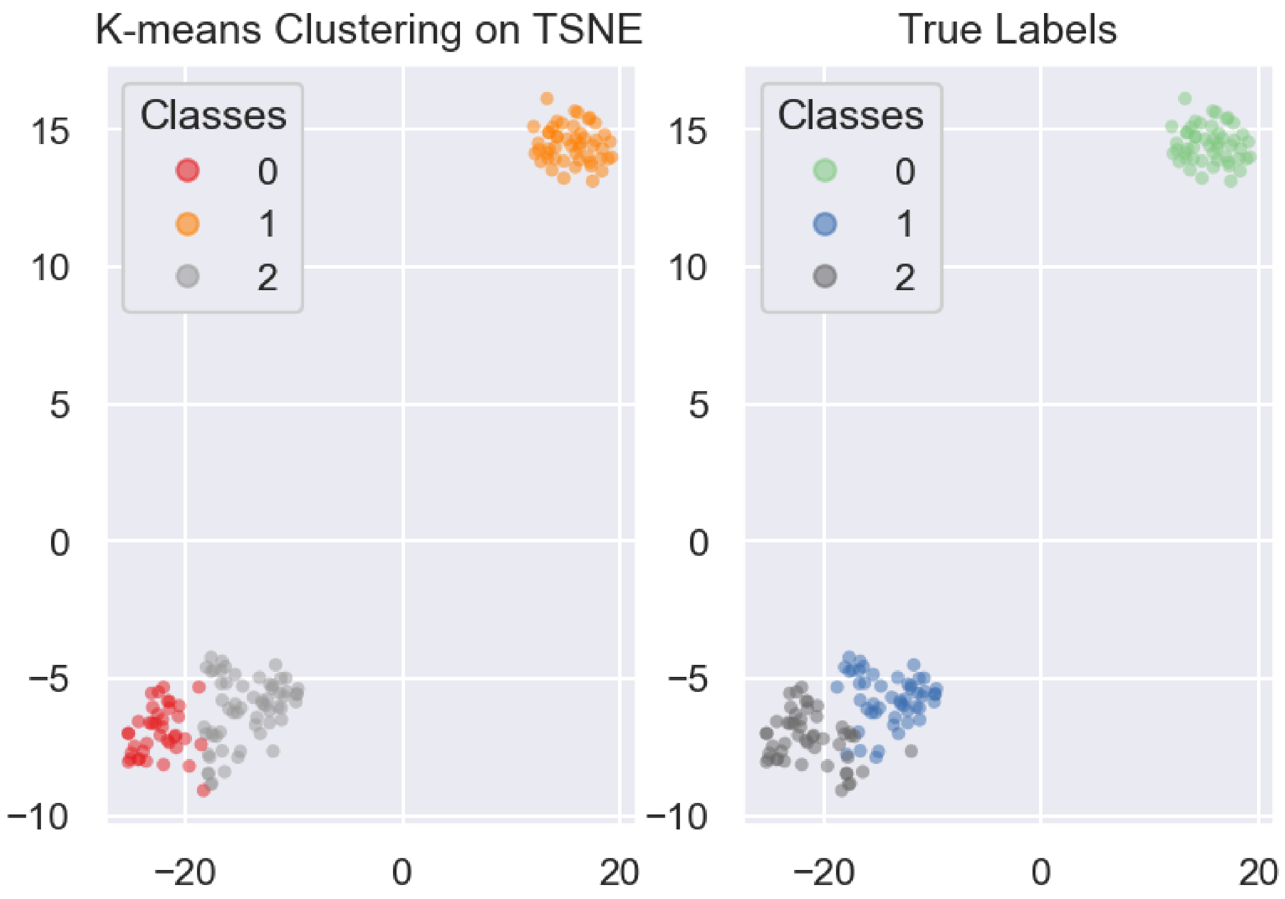
In [8]:

```python
fig, (ax1, ax2) = plt.subplots(1, 2,dpi=150)

fig1 = ax1.scatter(X_tsne[:, 0], X_tsne[:, 1],c=y_km_tsne, s=15, edgecolor='none', alpha=0.5,cmap=plt.cm.get_cmap('Set1', 3))
fig2 = ax2.scatter(X_tsne[:, 0], X_tsne[:, 1],c=y, s=15, edgecolor='none', alpha=0.5, cmap=plt.cm.get_cmap('Accent', 3))
ax1.set_title('K-means Clustering on TSNE')
legend1 = ax1.legend(*fig1.legend_elements(), loc="best", title="Classes")
ax1.add_artist(legend1)
ax2.set_title('True Labels')
legend2 = ax2.legend(*fig2.legend_elements(), loc="best", title="Classes")
ax2.add_artist(legend2)
```

Out[8]:

```
<matplotlib.legend.Legend at 0x7fc55f792790>
```



How about try other clustering methods?

In [9]:

```python
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components = 3,random_state=0)
y_gm = gm.fit_predict(X)
```

In [10]:

```python
metrics.adjusted_rand_score(y_gm, y)
```

Out[10]:

```
0.9038742317748124
```

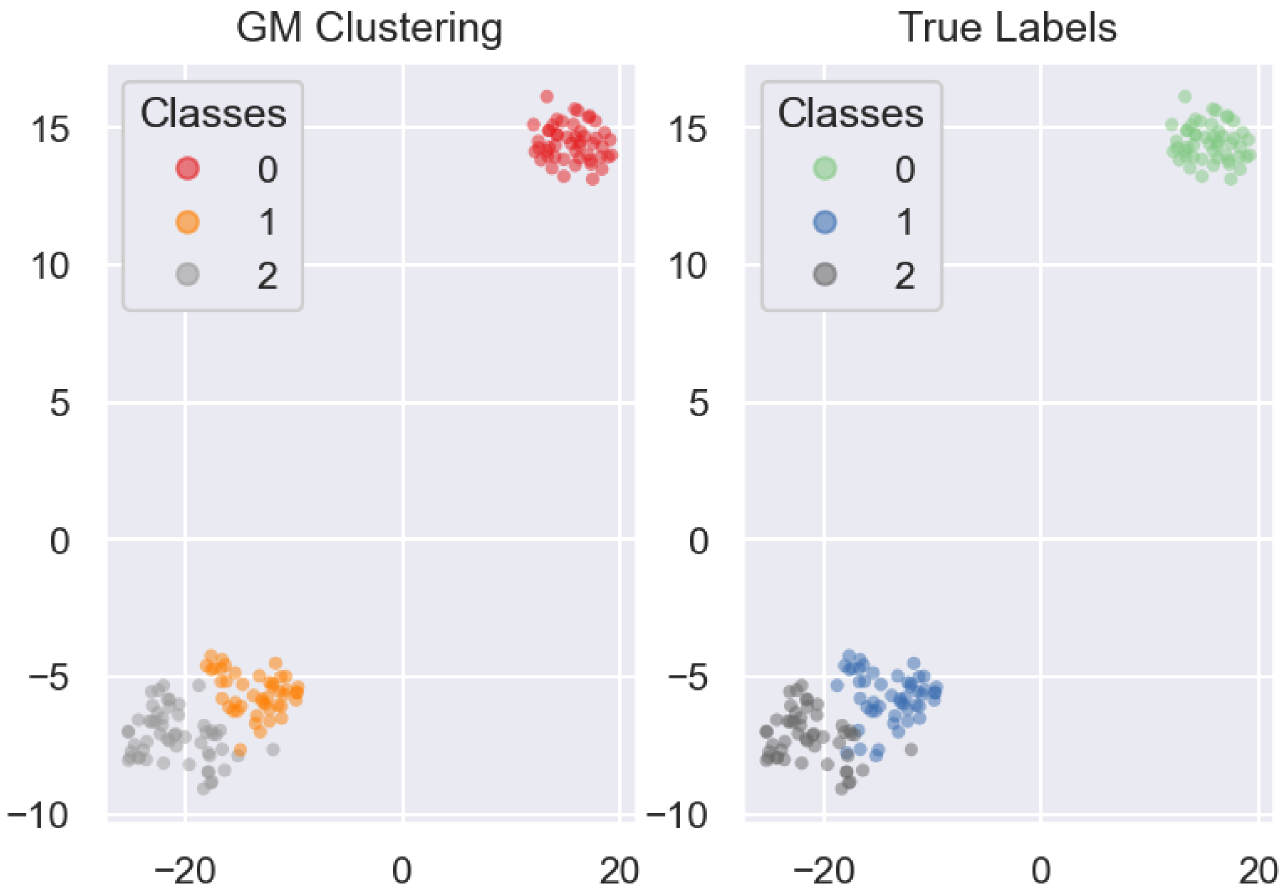
In [11]:

```
fig, (ax1, ax2) = plt.subplots(1, 2,dpi=150)

fig1 = ax1.scatter(X_tsne[:, 0], X_tsne[:, 1],c=y_gm, s=15, edgecolor='none', alpha=
0.5,cmap=plt.cm.get_cmap('Set1', 3))
fig2 = ax2.scatter(X_tsne[:, 0], X_tsne[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,
cmap=plt.cm.get_cmap('Accent', 3))
ax1.set_title('GM Clustering')
legend1 = ax1.legend(*fig1.legend_elements(), loc="best", title="Classes")
ax1.add_artist(legend1)
ax2.set_title('True Labels')
legend2 = ax2.legend(*fig2.legend_elements(), loc="best", title="Classes")
ax2.add_artist(legend2)
```

Out[11]:

```
<matplotlib.legend.Legend at 0x7fc55fe26ed0>
```



In [12]:

```
y_gm_tsne = gm.fit_predict(X_tsne)
metrics.adjusted_rand_score(y_gm_tsne, y)
```

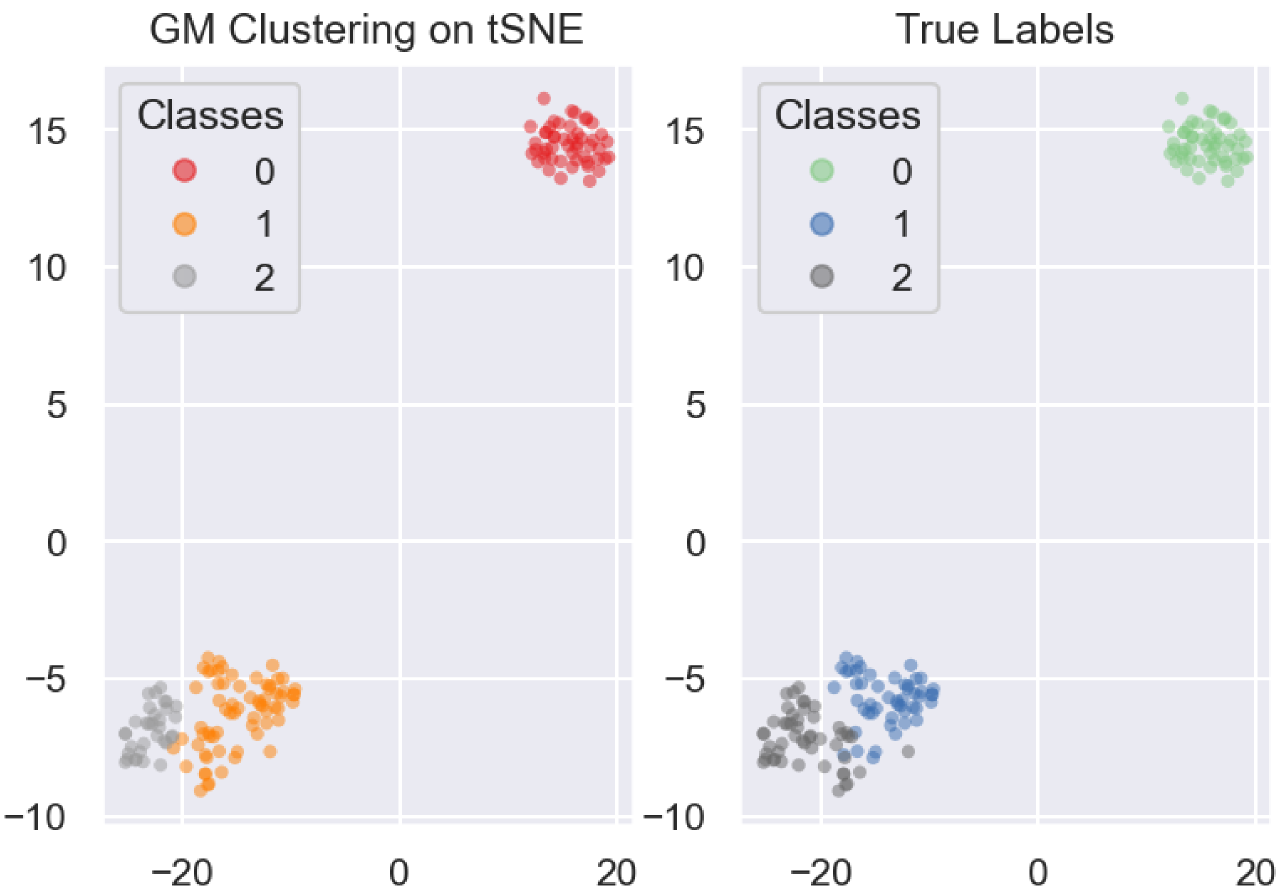
Out[12]:

```
0.7195837484778037
```

```
In [13]: fig, (ax1, ax2) = plt.subplots(1, 2,dpi=150)

         fig1 = ax1.scatter(X_tsne[:, 0], X_tsne[:, 1],c=y_gm_tsne, s=15, edgecolor='none', al
         pha=0.5,cmap=plt.cm.get_cmap('Set1', 3))
         fig2 = ax2.scatter(X_tsne[:, 0], X_tsne[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,
         cmap=plt.cm.get_cmap('Accent', 3))
         ax1.set_title('GM Clustering on tSNE')
         legend1 = ax1.legend(*fig1.legend_elements(), loc="best", title="Classes")
         ax1.add_artist(legend1)
         ax2.set_title('True Labels')
         legend2 = ax2.legend(*fig2.legend_elements(), loc="best", title="Classes")
         ax2.add_artist(legend2)
```

```
Out[13]: <matplotlib.legend.Legend at 0x7fc5608b2150>
```



Explore yourself and you may upload your results to [Kaggle (https://www.kaggle.com/uciml/iris)](https://www.kaggle.com/uciml/iris)!