Politecnico di Torino

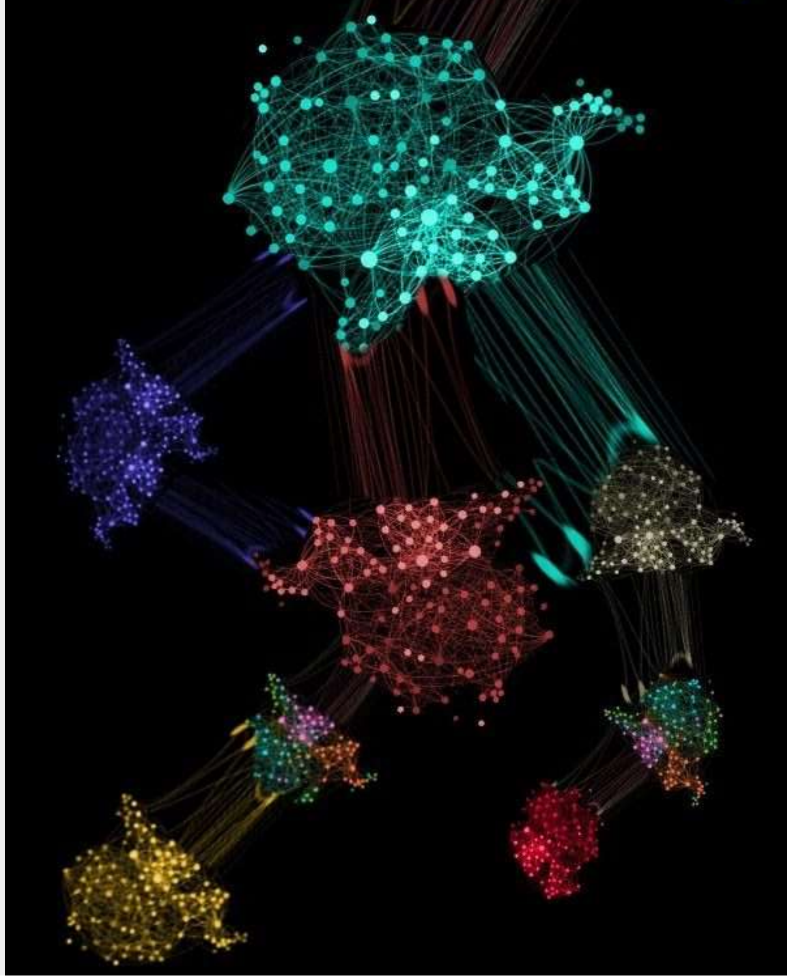
Lab 2: Unsupervised Learning

K-Means, Gaussian Mixture Models (GMM), Expectation Maximization (EM)

Machine Learning and Deep Learning 2023

Eros Fanì & Debora Caldarola

eros.fani@polito.it, debora.caldarola@polito.it



Unsupervised learning

What is unsupervised learning?

- Introduction to clustering
  - Introduction
  - Metrics
- K-Means
  - how does it work?
  - PROs, CONs
- GMMs
  - what is a GMM?
  - How does it differ from K-means?
  - PROs, CONs

Unsupervised Learning

- **Goal**: discover patterns or structure within a dataset without the use of explicit labels or guidance
    - Identify the hidden structure behind the data

- No feedback mechanism for the algorithm to determine how close or far it is from the optimal solution

Unsupervised Learning

- Input dataset $X = \{x1, x2, ..., xn\}$, $xi \in R^m$

- **Goal**: find **f: X→Y** mapping the input data to a set of output variables Y
  - f is not provided with explicit target variables or labels
  - f should discover underlying patterns, structure or relationships within the data
  - f should be able to generalize to new, unseen data
- Particularly useful for problems where the desired outcome is not known in advance

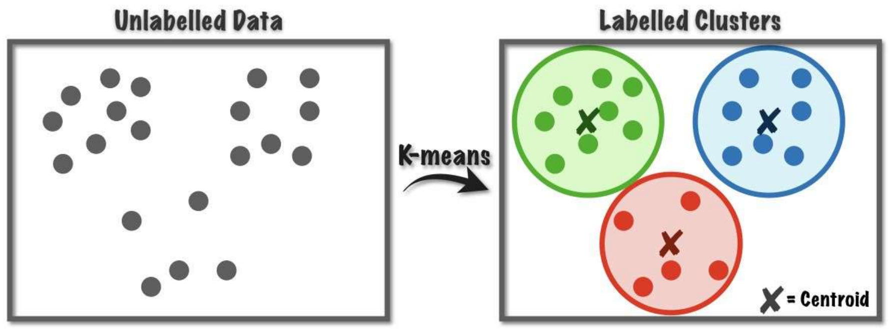
- EXAMPLES
  - clustering
  - dimensionality reduction
  - anomaly detection

Clustering

- Unsupervised ML technique involving grouping similar data points together based on their similarity or distance from each other
- **Goal**: identify natural groupings, or **clusters**, in a dataset without any prior knowledge or labels

- Good clusters if:
  - high intra-cluster similarity
  - low inter-cluster similarity

- EXAMPLES:
  - **k-means clustering**
  - hierarchical clustering
  - DBSCAN clustering
  - **density-based clustering (GMM)**



Clustering metrics

SUPERVISED CLUSTERING METRICS:

- Require true labels to be computed
- Examples: homogeneity, completeness, V-measure

UNSUPERVISED CLUSTERING METRICS:

- Do not require true labels to be computed
- Example: Silhouette coefficient

https://scikit-learn.org/stable/modules/clustering.html#homogeneity-completeness
https://scikit-learn.org/stable/modules/clustering.html#silhouette-coefficient

K-Means

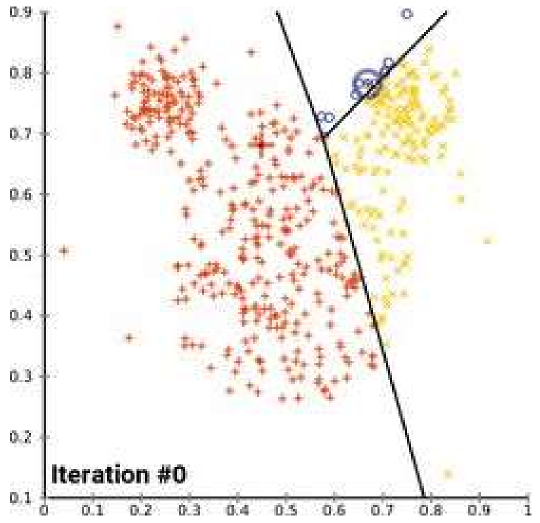
1) Sample K centroids randomly
2) Assign each sample to the nearest centroid
3) Compute new cluster centroids
4) Repeat until convergence

PROs:

- **Simple and efficient**
- Scalable
- **Interpretable results**
- Flexible
- **Guaranteed to converge**

CONs:

- **Sensible to initialization**
- **Difficult to choose K**
- Sensitive to outliers
- Limited to linear boundaries
- Biased towards equal-sized clusters
- Only handles numerical data
- **Assumes spherical clusters**



Exercise 1: K-Means with Sklearn

Explore the K-means algorithm on the Iris dataset

- Import the Iris dataset
- Evaluate K-means with 3 clusters (= 3 real labels: Virginica, Versicolour, Setosa) with several random seeds
    - How does the random initialization of the centroids affect the final clusters?
- Evaluate K-means with a fixed random seed, with K = 2, 3, … 20
    - What is the best K according to the supervised clustering metrics? And according to the unsupervised clustering metrics? Why?
- Reduce the data with PCA and apply k-means again. Visualize the results using PCA with 2 components (suggestion: use the script in the next slides)

How to visualize the results (1/3)

```python
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np

reduced_data = PCA(n_components=2).fit_transform(iris.data)
kmeans_model.fit(reduced_data)

# Step size of the mesh. Decrease to increase the quality of the VQ.
h = 0.02  # point in the mesh [x_min, x_max]x[y_min, y_max].

# Plot the decision boundary. For that, we will assign a color to each
x_min, x_max = reduced_data[:,0].min() - 1, reduced_data[:, 0].max() + 1
y_min, y_max = reduced_data[:,1].min() - 1, reduced_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans_model.predict(np.c_[xx.ravel(), yy.ravel()])
```

How to visualize the results (2/3)

```python
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(
    Z,
    interpolation="nearest",
    extent=(xx.min(), xx.max(), yy.min(), yy.max()),
    cmap='Set2',
    aspect="auto",
    origin="lower",
)

plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=iris.target, cmap='Pastel2')
# Plot the centroids as a white X
centroids = kmeans_model.cluster_centers_
```

How to visualize the results (3/3)

```
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    zorder=10,
)
plt.title(
    "K-means clustering on the digits dataset (PCA-reduced data)\n"
    "Centroids are marked with white cross"
)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

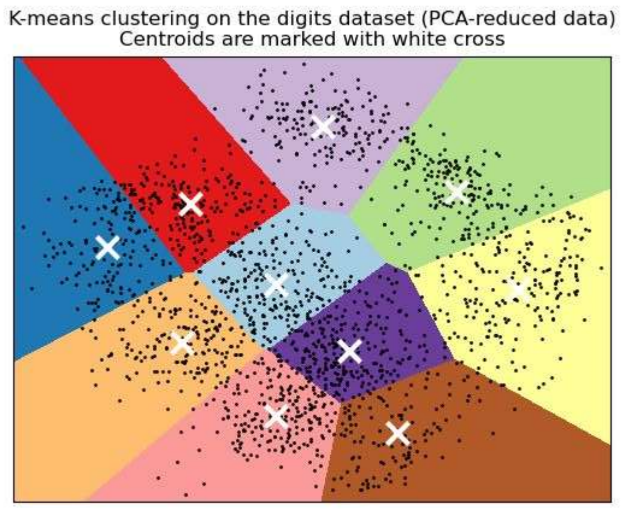


Image from sklearn official tutorials

Tips

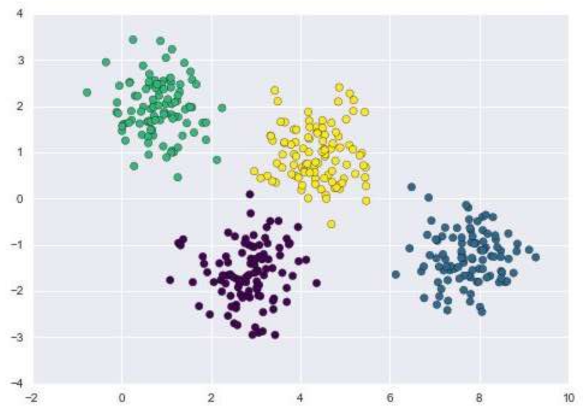
- Remember to standardize the data
- Refer to the following documentation:
    - https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html
    - https://scikit-learn.org/stable/modules/clustering.html#silhouette-coefficient
    - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
    - https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
    - https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Gaussian Mixture Models (GMMs)

- Overcoming the drawbacks of K-means
- The idea behind GMMs
- GMMs for handling not well separable data
- GMMs for generating new data

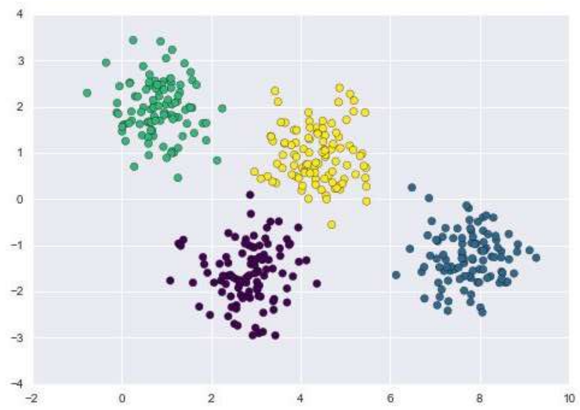
Main issues with *K*-means

- Poor performance in real-world scenarios due to the distance-from-cluster idea used to determine cluster membership



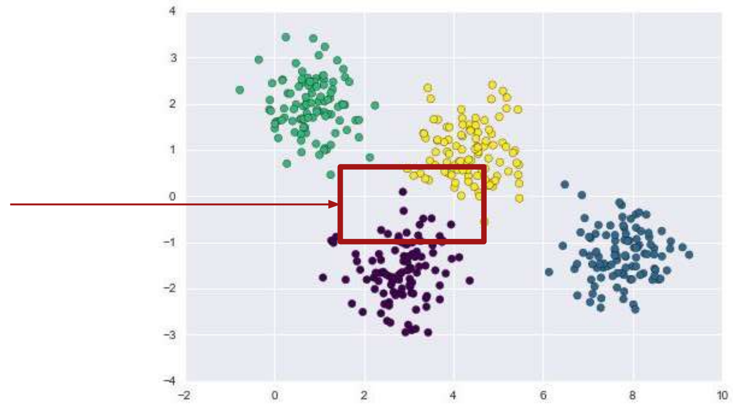
Main issues with *K*-means

- Poor performance in real-world scenarios due to the distance-from-cluster idea used to determine cluster membership
- Given well separable data, *K*-means finds suitable clusters



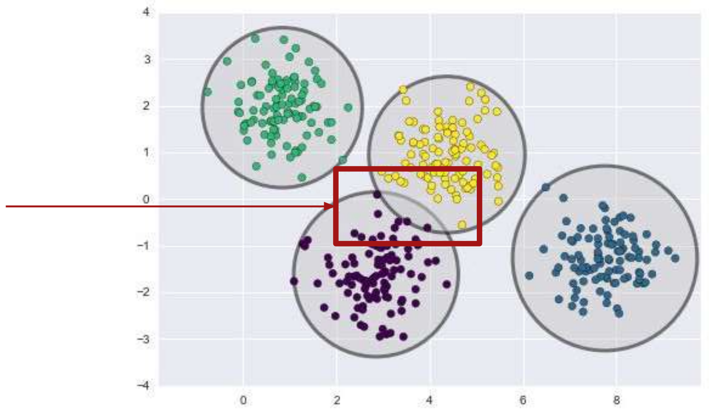
Main issues with *K*-means

- Poor performance in real-world scenarios due to the distance-from-cluster idea used to determine cluster membership
- Given well separable data, *K*-means finds suitable clusters
    - What happens when there is some overlap?



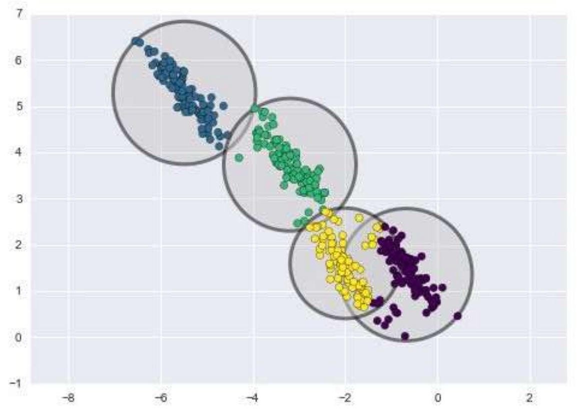
Main issues with *K*-means

- Poor performance in real-world scenarios due to the distance-from-cluster idea used to determine cluster membership
- Given well separable data, *K*-means finds suitable clusters
    - What happens when there is some overlap?
    - **No measure of uncertainty cluster assignment**



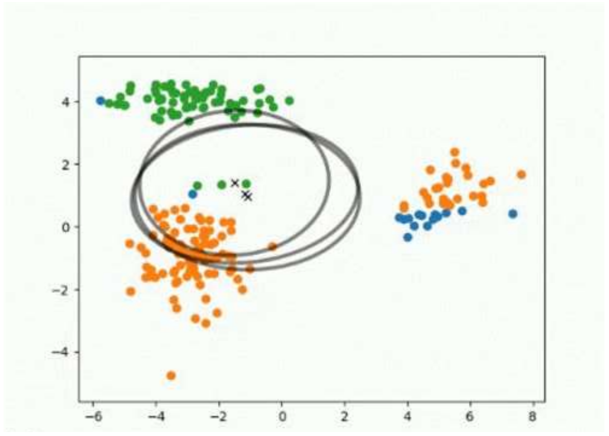
Main issues with *K*-means

- Poor performance in real-world scenarios due to the distance-from-cluster idea used to determine cluster membership
- Given well separable data, *K*-means finds suitable clusters
    - What happens when there is some overlap?
    - **No measure of uncertainty cluster assignment**
- The clusters must be **circular**
    - … and they may not be a good fit to the data
    - Not flexible enough



Gaussian Mixture Models (GMMs)

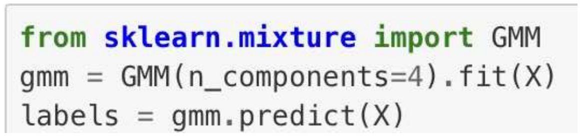
A Gaussian mixture model (GMM) attempts to find a **mixture** of multi-dimensional **Gaussian probability distributions** that best model any input dataset.



```
from sklearn.mixture import GMM
gmm = GMM(n_components=4).fit(X)
labels = gmm.predict(X)
```

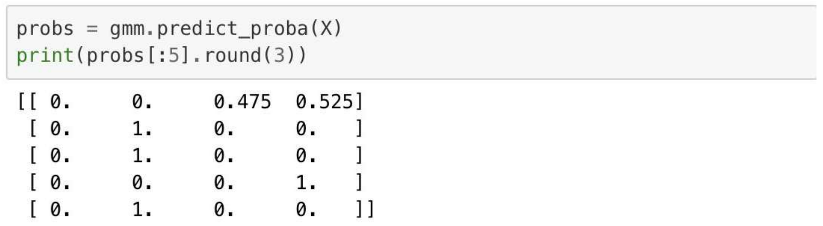
Gaussian Mixture Models (GMMs)

A Gaussian mixture model (GMM) attempts to find a **mixture** of multi-dimensional **Gaussian probability distributions** that best model any input dataset.



```
from sklearn.mixture import GMM
gmm = GMM(n_components=4).fit(X)
labels = gmm.predict(X)
```

Each point belongs to the cluster with a given probability.



```
probs = gmm.predict_proba(X)
print(probs[:5].round(3))
```

```
[[ 0.     0.     0.475  0.525]
 [ 0.     1.     0.     0.   ]
 [ 0.     1.     0.     0.   ]
 [ 0.     0.     0.     1.   ]
 [ 0.     1.     0.     0.   ]]
```

The position and the shape of each cluster are defined by mean and covariance.

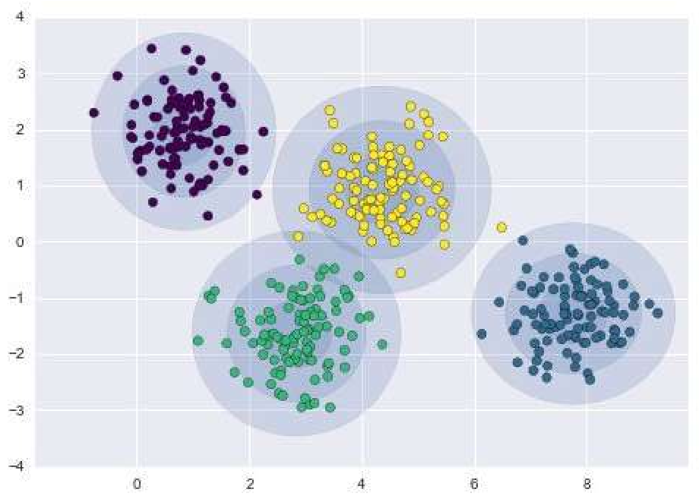
GMMs in short

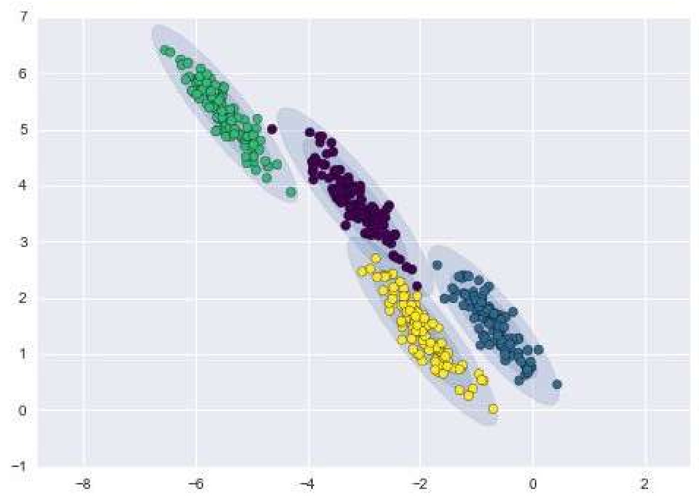
Based on an **Expectation-Maximization** approach.

1. Choose starting guesses for random assignments and shapes
2. Repeat until convergence:
    - **a.** **E-step**: for each point, find the weights encoding the probability of membership in each cluster
    - **b.** **M-step**: for each cluster, update its location, normalization, and shape based on *all* data points, making use of the weights

- The resulting clusters are associated with a **smooth** Gaussian model, rather than a hard-edged sphere.
- The optimal solution may be missed, so **multiple random initializations** are used.
- No distance measures are used. The points are assigned to the clusters based on probability distributions.

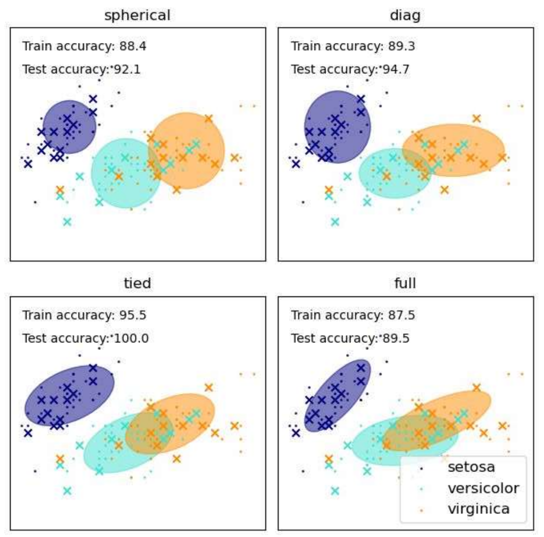
Example of resulting clusters with GMMs





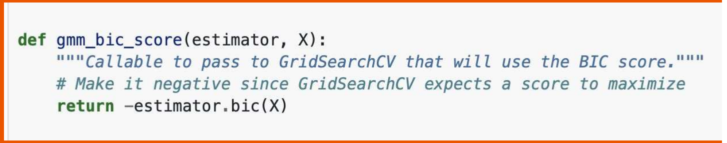
Relevant choices

1. Initialization - - to define the initial center of the model components
    a. k-means (can be computationally heavy)
    b. k-means++: pick first center at random, then subsequent centers are the most distant ones
    c. random
2. Number of components
3. Covariance type (constraint on the estimated classes)
    a. spherical
    b. diagonal
    c. tied
    d. full covariance



Exercise 2: GMMs

1. Build your first GMM to replicate the results obtained with K-means on the IRIS dataset
   - Suggestion: `n_components` is comparable to *K*
2. Try out the different **initializations**: how do the results differ?
   - Take a look here to see how to plot the obtained clusters.
3. Pick the best number of components by applying grid search. Use the **negative BIC (Bayes Information Criterion) score** as scoring method. Select the number of components having the lowest BIC.
   - In sklearn: `GridSearchCV`
   - Suggestion: define the function computing the BIC score and pass it to the GridSearch

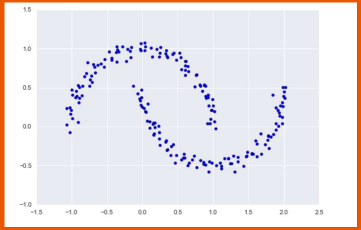


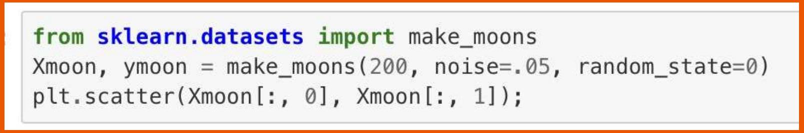
```
def gmm_bic_score(estimator, X):
    """Callable to pass to GridSearchCV that will use the BIC score."""
    # Make it negative since GridSearchCV expects a score to maximize
    return -estimator.bic(X)
```

1. How does the choice of covariance type affect the results? How do the built clusters compare with the actual dataset?
   - Take a look at this tutorial

Exercise 3: Handling not well separable data

1. Setup the Moons dataset from sklearn as:
   You should see something like





```
from sklearn.datasets import make_moons
Xmoon, ymoon = make_moons(200, noise=.05, random_state=0)
plt.scatter(Xmoon[:, 0], Xmoon[:, 1]);
```

1. Using the function `plot_gmm` from the next slide, see what happens when moving from `n_components=2` to `n_components >= 16`
   - Is the GMM able to model the overall distribution of the input data?

Useful plotting functions

```
from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()

    # Convert covariance to principal axes
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    # Draw the Ellipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))
```

```
def plot_gmm(gmm, X, label=True,
ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1],
c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1],
s=40, zorder=2)
    ax.axis('equal')

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_,
gmm.covars_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w
* w_factor)
```

Exercise 4: GMMs for generating new data

A trained GMM describes the distribution of the input data.

1. Load the MNIST dataset (`load_digits`) and plot the first 30 samples (will serve as reference later)
2. GMMs might have troubles converging in high dimensional spaces. Apply PCA and preserve 99% of the variance.
    a. How many principal components do you need to keep 99% of the variance?
3. Define and fit your GMM model:
    a. Use BIC to select the best number of components to fit the reduced data. Try a few values in the range [10, 250].
    b. Fit the GMM model with the best number of components to the reduced data. Check its convergence with `gmm.converged_`
4. It's time to generate new data following the learned distribution!
    a. Call the `sample(n_new_samples)` method on your GMM. The output should have dimensions `(n_new_samples, n_PCs)`
    b. Apply the inverse transform the PCA to the obtained data to return to the original space
    c. Plot the obtained digits: how similar are they to the original ones?

Some useful references

https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html

https://scikit-learn.org/stable/modules/mixture.html

https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_pdf.html#sphx-glr-auto-examples-mixture-plot-gmm-pdf-py

https://towardsdatascience.com/implement-expectation-maximization-em-algorithm-in-python-from-scratch-f1278d1b9137