Gaussian Mixture Models

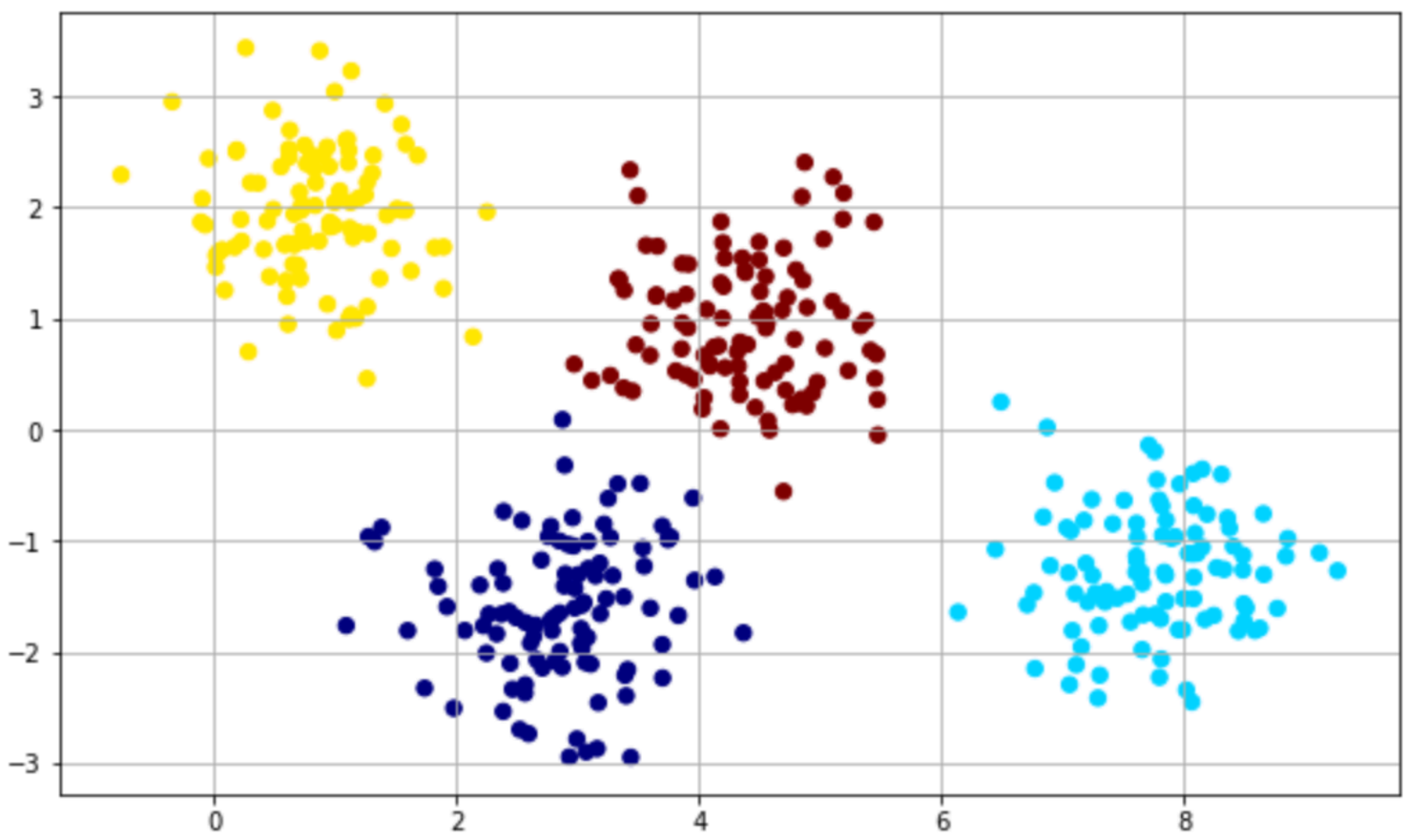
PHY3287 - Computational Astronomy
Dr Andrea DeMarco
andrea.demarco@um.edu.mt

k-Means

- k-Means clustering is simple and easy to understand
- Has some practical challenges in its application
- Non-probabilistic nature, hard clustering - not good for many real-world situations
- This lecture will introduce Gaussian Mixture Models (GMMs) - an extension of the k-Means idea, but also far more powerful
- GMMs are not classifiers, but can be used to build classification systems

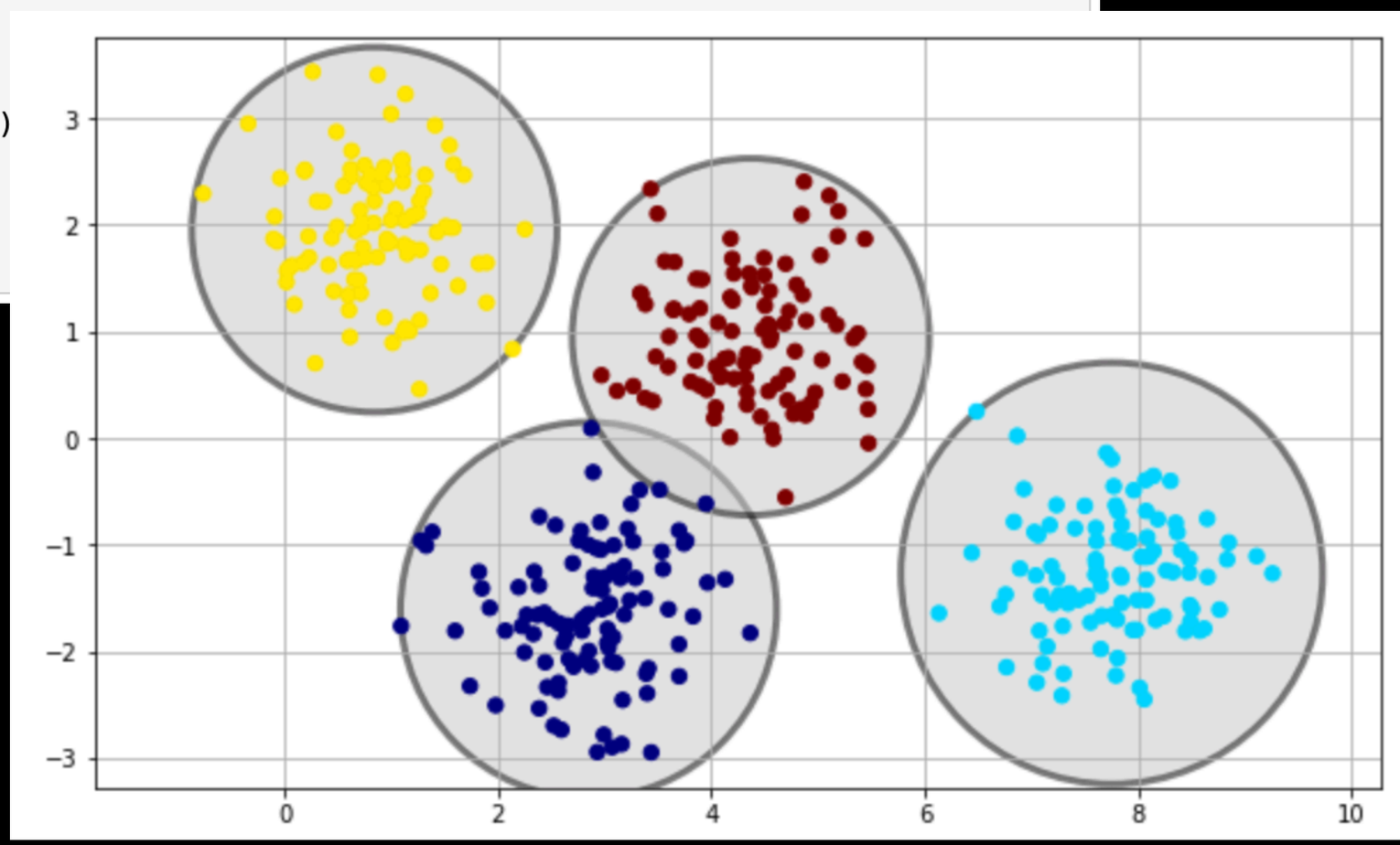
k-Means Weak Points

```
# Generate some data
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=400, centers=4,
                       cluster_std=0.60, random_state=0)
X = X[:, ::-1] # flip axes for better plotting

# Plot the data with K Means Labels
from sklearn.cluster import KMeans
kmeans = KMeans(4, random_state=0)
labels = kmeans.fit(X).predict(X)
fig, gmm1 = plt.subplots(figsize=(10, 6))
gmm1.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='jet');
gmm1.grid()
```



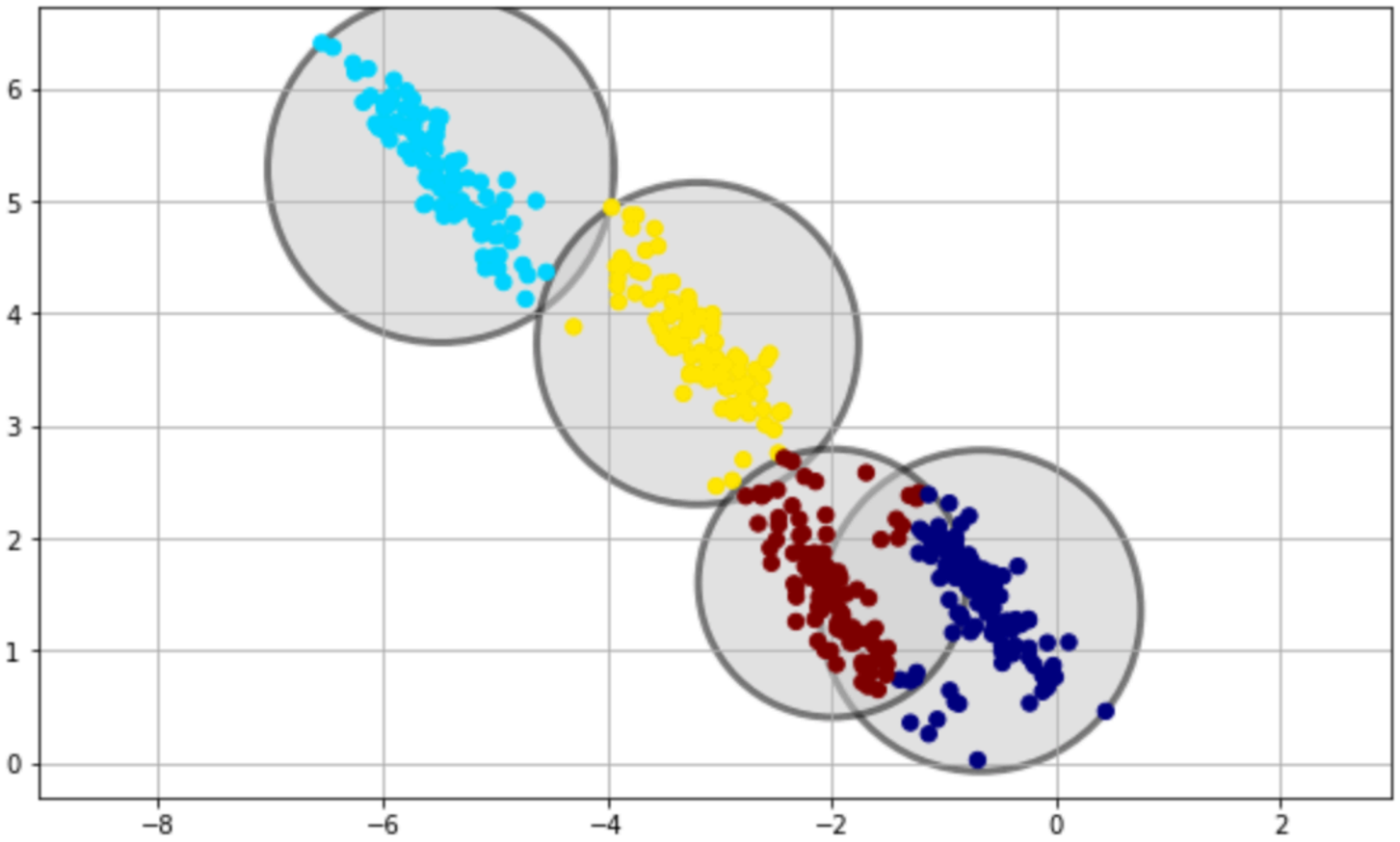
k-Means Weak Points

```
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist

def plot_kmeans(kmeans, X, n_clusters=4, rseed=0, ax=None):
    labels = kmeans.fit_predict(X)

    # plot the input data
    ax = ax or plt.gca()
    ax.axis('equal')
    ax.grid()
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='jet', zorder=2)

    # plot the representation of the KMeans model
    centers = kmeans.cluster_centers_
    radii = [cdist(X[labels == i], [center]).max()
             for i, center in enumerate(centers)]
    for c, r in zip(centers, radii):
        ax.add_patch(plt.Circle(c, r, fc='#CCCCCC',
                                ec='black', lw=3,
                                alpha=0.5, zorder=1))

fig, gmm2 = plt.subplots(figsize=(10, 6))
kmeans = KMeans(n_clusters=4, random_state=0)
plot_kmeans(kmeans, X, ax=gmm2)
```



k-Means Weak Points

```
rng = np.random.RandomState(13)
X_stretched = np.dot(X, rng.randn(2, 2))

fig, gmm3 = plt.subplots(figsize=(10, 6))
kmeans = KMeans(n_clusters=4, random_state=0)
plot_kmeans(kmeans, X_stretched, ax=gmm3)
```



k-Means Weak Points

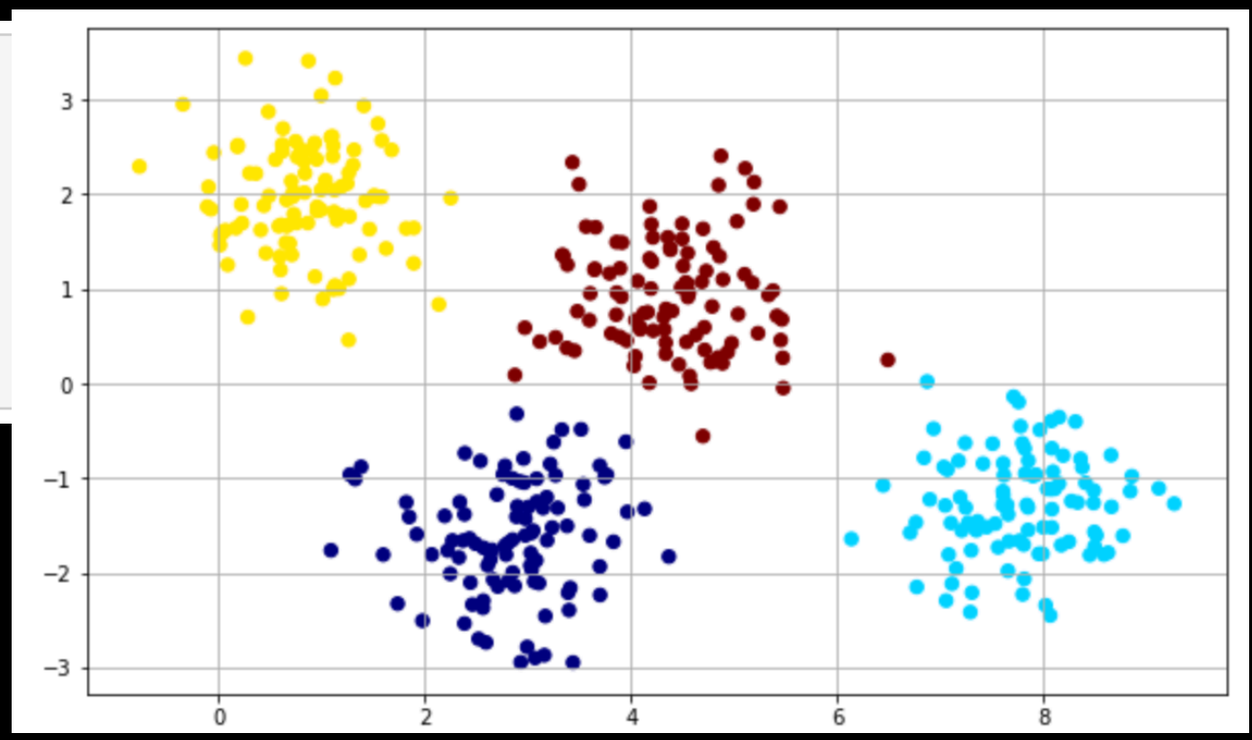
- Lack of flexibility in cluster shape
- Lack of probabilistic cluster assignment
- These 2 problems are especially bad for low-dimensional datasets
- Generalize k-Means? e.g. measure uncertainty in cluster assignment by comparing the distances of each point to all cluster centres (not just the closest)
- Or allow cluster boundaries to be ellipses and not circle…

GMMs
Expectation Maximization

- A GMM attempts to find a mixture of Gaussian probability distributions that best model an input dataset

- In the simplest case, GMMs can be used to find clusters in the same manner as k-Means

```
from sklearn.mixture import GaussianMixture as GMM

gmm = GMM(n_components=4).fit(X)
labels = gmm.predict(X)

fig, gmm4 = plt.subplots(figsize=(10, 6))
gmm4.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='jet');
gmm4.grid()
```




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

```
[[0.    0.463 0.    0.537]
 [1.    0.    0.    0.   ]
 [1.    0.    0.    0.   ]
 [0.    0.    0.    1.   ]
 [1.    0.    0.    0.   ]]
```

GMMs
Expectation Maximization

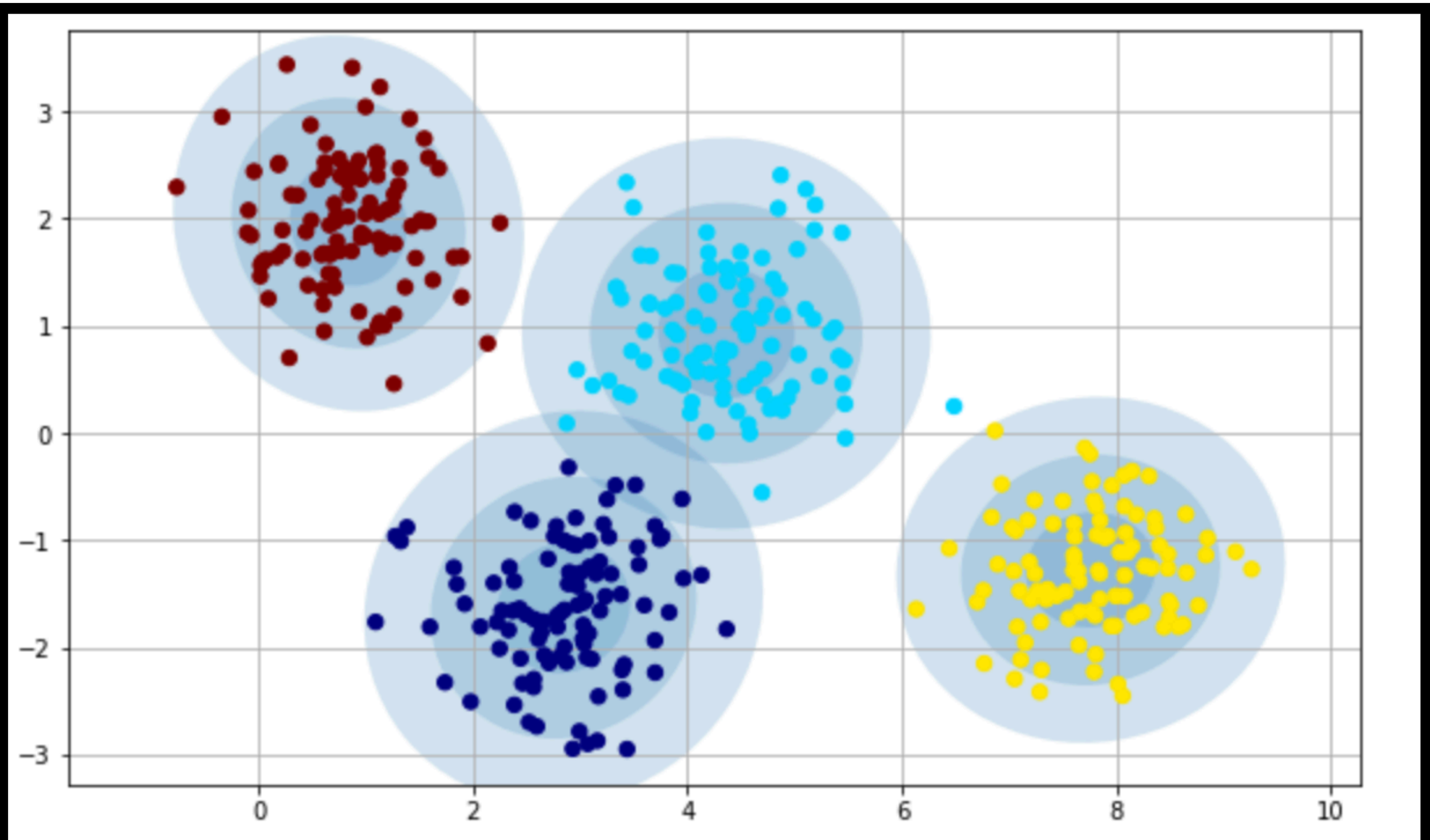
- Under the hood, a similar process to k-Means. GMM applies an Expectation-Maximization approach to do the following:
    - Choose starting guesses for location and shape
    - Repeat until converged:
        - E-Step: for each sample, find weights encoding the probability of membership in each cluster
        - M-Step: for each cluster, update its location, normalisation and shape based on all samples assigned to it, making use of weights

GMMs
Cluster Membership

```python
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='jet', 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.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
    ax.grid()
```

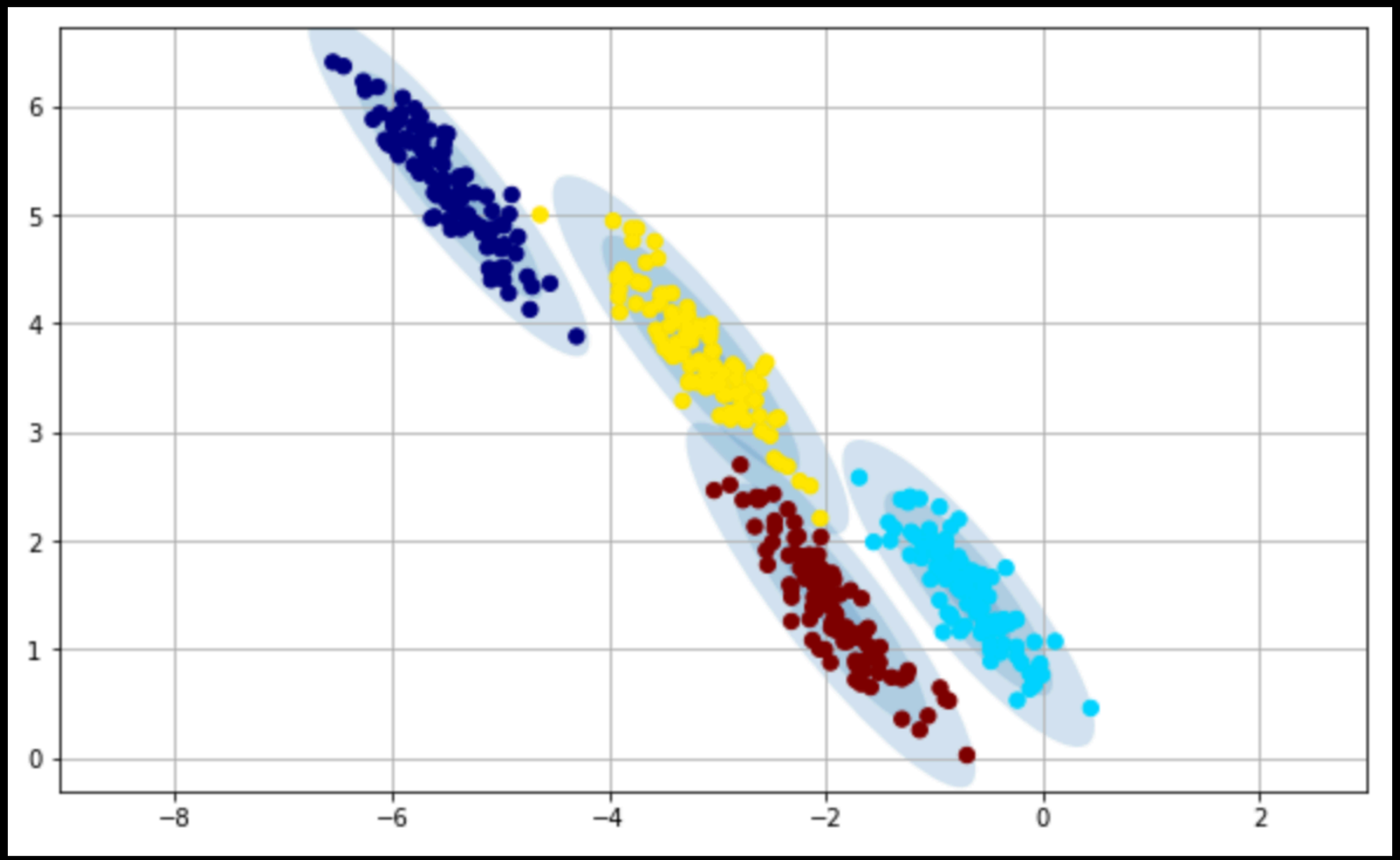
GMMs
Cluster Membership

```
gmm = GMM(n_components=4)

fig, gmm6 = plt.subplots(figsize=(10, 6))
plot_gmm(gmm, X, ax=gmm6)
plt.show()
```



GMMs
Cluster Membership

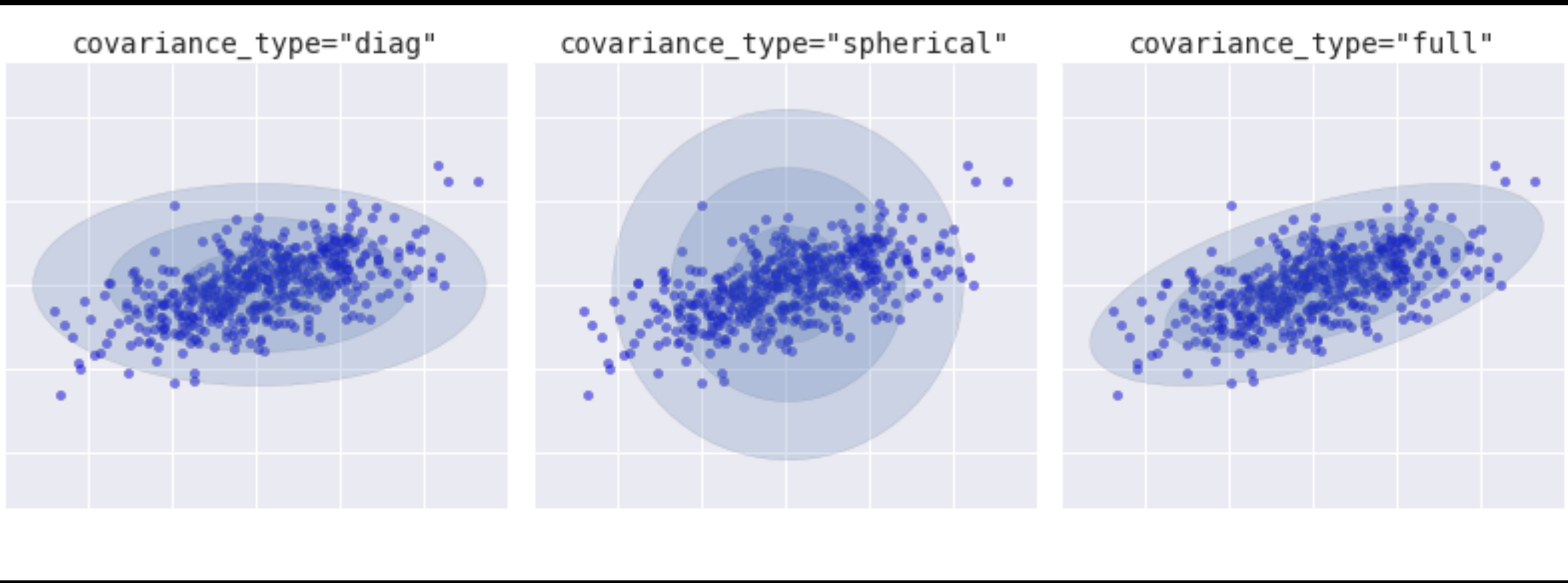
```
rng = np.random.RandomState(13)
X_stretched = np.dot(X, rng.randn(2, 2))

fig, gmm6 = plt.subplots(figsize=(10, 6))
gmm = GMM(n_components=4, covariance_type='full')
plot_gmm(gmm, X_stretched, ax=gmm6)
```



Covariances in GMMs

- Quite important to pick the right covariance_type.
- This hyper-parameter controls the degrees of freedom in the shape of each cluster
- The default is diagonal covariance - the size of the cluster along each dimension can be set independently, with the resulting ellipse constrained to align with the axes
- Slightly simper/faster is spherical covariance - constrains the shape such that all dimensions are equal (similar to k-Means characteristics)
- A more complicated/expensive model is full covariance - allows clusters to be modelled as an ellipse with arbitrary orientation.

Covariances in GMMs



GMM Training

- The GMM function is essentially a weighted sum of Gaussian functions, and is written as:

$$f(x \mid \mu, \Sigma) = \sum_{k=1}^{M} c_k \frac{1}{\sqrt{2\pi \mid \Sigma_k \mid}} \exp\left[(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)\right]$$

- The parameters of the GMM are therefore:
  - A set of weights, one scalar per component, all summing up to 1
  - A set of mean vectors, one per component
  - A set of covariance matrices, one per component

GMM Training

- In a standard Gaussian, all the data belonged to one Gaussian component
- How can we train a GMM if we do not know in advance which mixture is supposed to account for which part of the distribution?

Expectation-Maximization: Intuition

- Every data point will have a likelihood of being produced by each component in the mixture
- Start with an initial estimate of the GMM parameters
- Iteratively improve the parameters by evaluating how much this current setup fits the data, and tweak the parameters to improve this fit
- The initial estimate can be completely random
- Smarter: k-means for initial means, k=M, global data covariance for all mixtures, weights as ratio of points in each cluster
- An algorithm to do all this exists: Expectation-Maximization (sometimes called Baum-Welch Estimation)

GMM Expectation-Maximization

- Notation is getting complex, so let's simplify the probability of an observation x, for a GMM function as:

$$p(x) = \sum_{k=1}^{M} c_k N(x \mid \mu_k, \Sigma_k)$$

- For a given vector x, we can evaluate the posterior probabilities (sometimes called responsibilities) i.e. the probability of a particular component

$$y_k(x) = p(k \mid x) = \frac{p(k)p(x \mid k)}{p(x)} = \frac{c_k N(x \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{M} c_j N(x \mid \mu_j, \Sigma_j)}$$

GMM Expectation-Maximization

- E-Step: Evaluate the responsibilities of every component using the current parameter values

- M-Step: Re-estimate the parameters using the current responsibilities

$$\mu_j = \frac{\sum_{n=1}^{N} y_j(x_n) x_n}{\sum_{n=1}^{N} y_j(x_n)}$$ **Means Update**

$$c_j = \frac{1}{N} \sum_{n=1}^{N} y_j(x_n)$$ **Weights Update**

$$\Sigma_j = \frac{\frac{1}{N} \sum_{n=1}^{N} y_j(x_n)(x_n - \mu_j)(x_n - \mu_j)^T}{\frac{1}{N} \sum_{n=1}^{N} y_j(x_n)}$$ **Covariance Update**

GMM Expectation-Maximization

- Evaluate the likelihood of the entire model for your training data
- If there is no convergence, repeat E-M steps
- Or, repeat for a fixed number of times
- Note: in general, it is worth investigating if means-only updates are enough

Pitfall: Gaussian Singularity

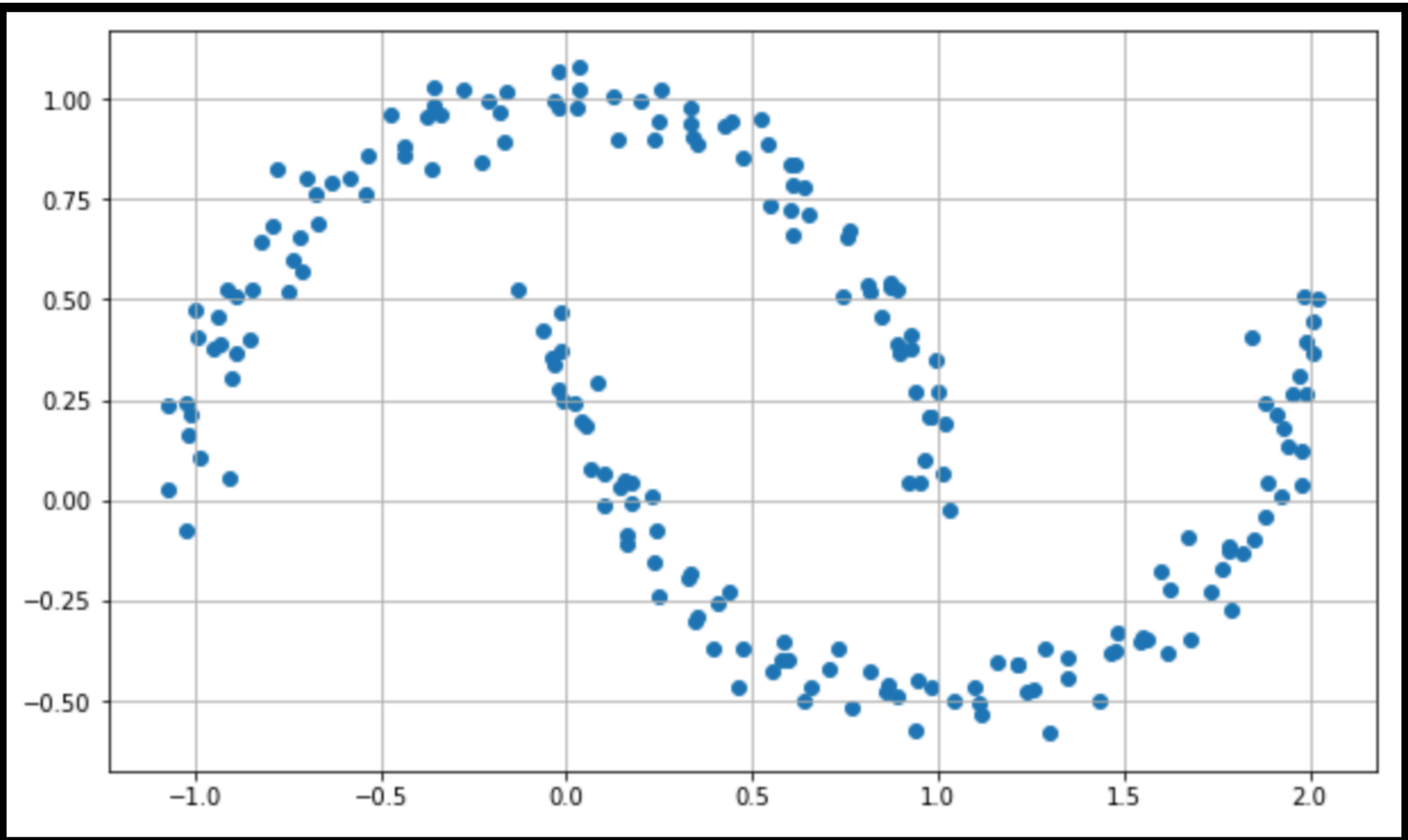
- The most common issue with GMM training is the “singularity”:
- A Gaussian component is fitted to a single point
- Variance is zero
- Covariance matrix becomes singular
- The component collapses to a spike, with infinite likelihood
- Solution: provide a sensible minimum variance (or variance floor) for all components

GMM as Density Estimation

- GMMs are often categorised as a clustering algorithm.
- However, fundamentally, a GMM is an algorithm for density estimation
- The result of a GMM fit to some data is technically not a clustering model, but a generative probabilistic model describing data distribution

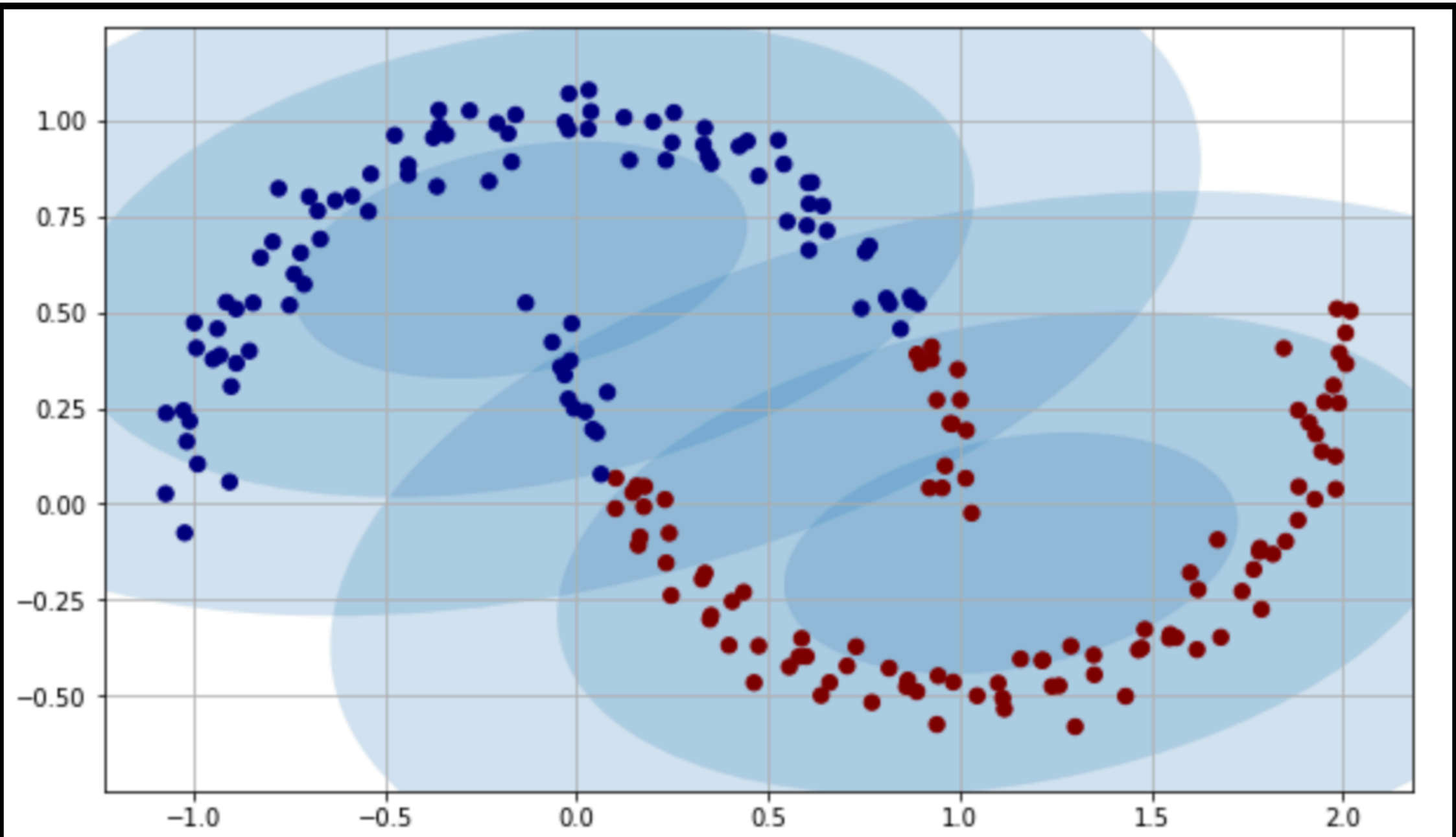
GMM as Density Estimation

```
from sklearn.datasets import make_moons
Xmoon, ymoon = make_moons(200, noise=.05, random_state=0)
fig, gmm7 = plt.subplots(figsize=(10, 6))
gmm7.scatter(Xmoon[:, 0], Xmoon[:, 1]);
gmm7.grid()
```



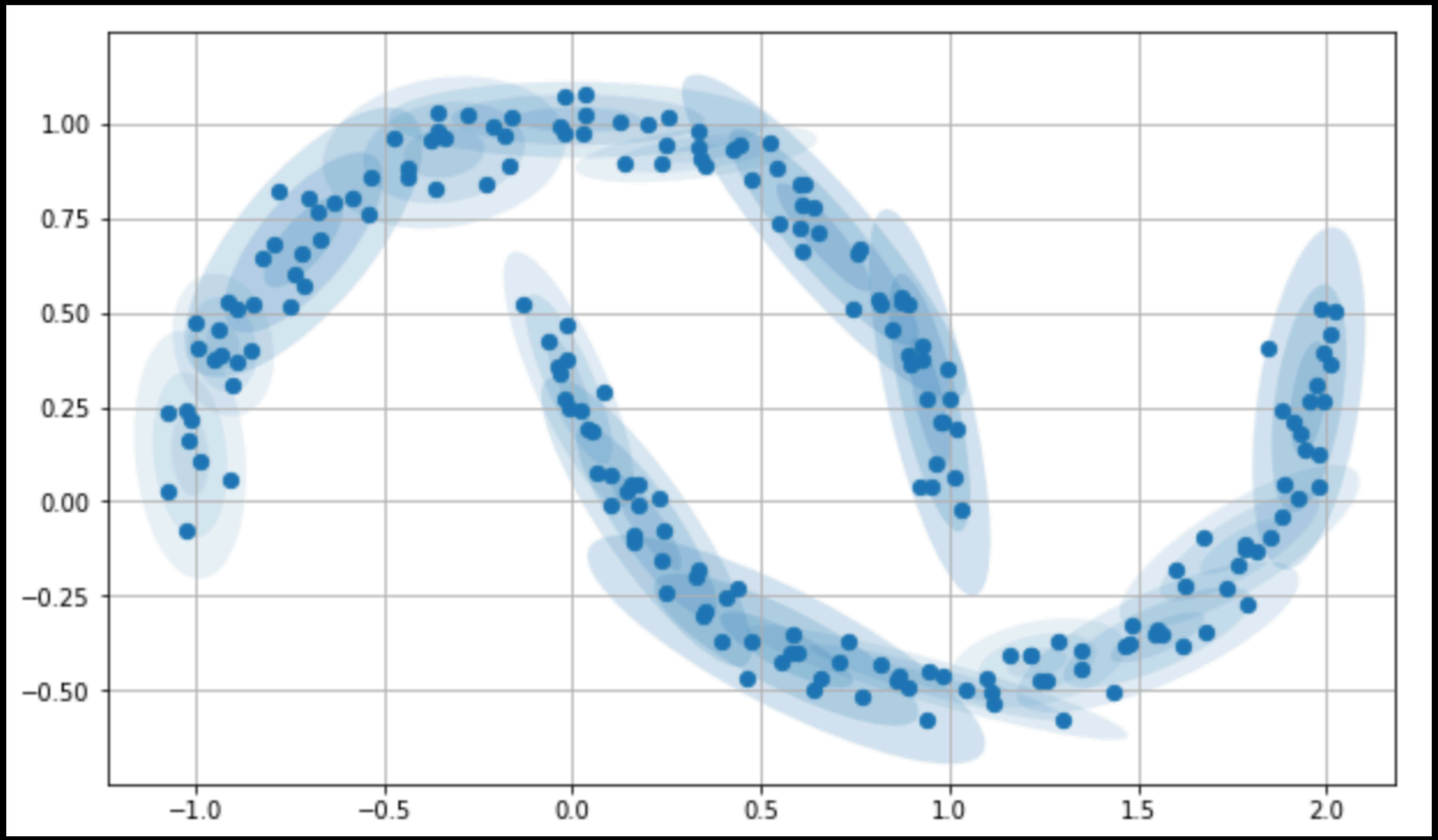
GMM as Density Estimation

```
gmm2_clusters = GMM(n_components=2, covariance_type='full')

fig, gmm8 = plt.subplots(figsize=(10, 6))
plot_gmm(gmm2_clusters, Xmoon, ax=gmm8)
```



GMM as Density Estimation

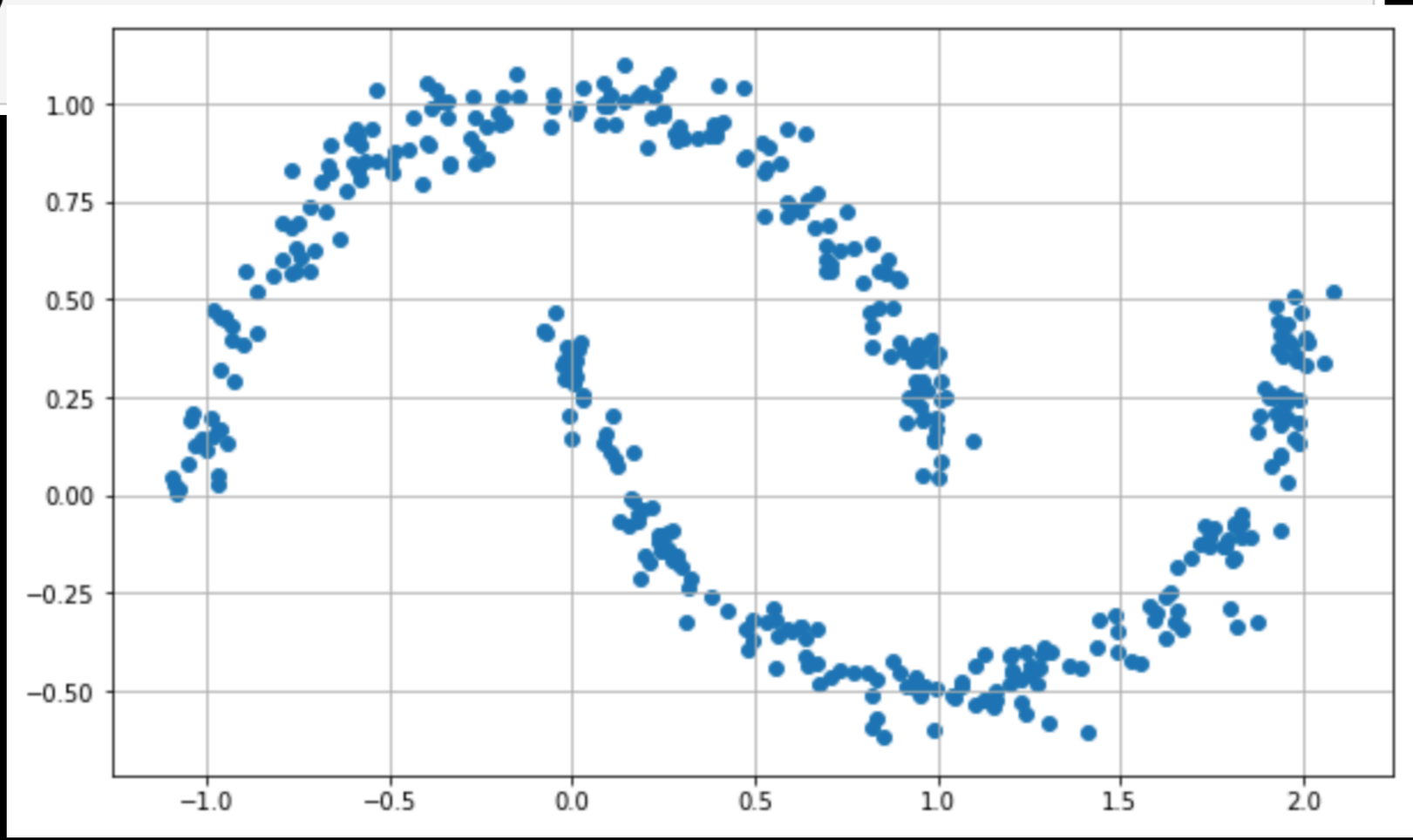
- But put in a to more components, and ignore cluster labels…



GMM as a Generative Model

- Think about a GMM as a model that is configured in such a way that we can sample from it to generate the data we have observed (and other similar data)

```
Xnew = gmm16_clusters.sample(400)[0]

fig, gmm10 = plt.subplots(figsize=(10, 6))
gmm10.scatter(Xnew[:, 0], Xnew[:, 1]);
gmm10.grid()
```

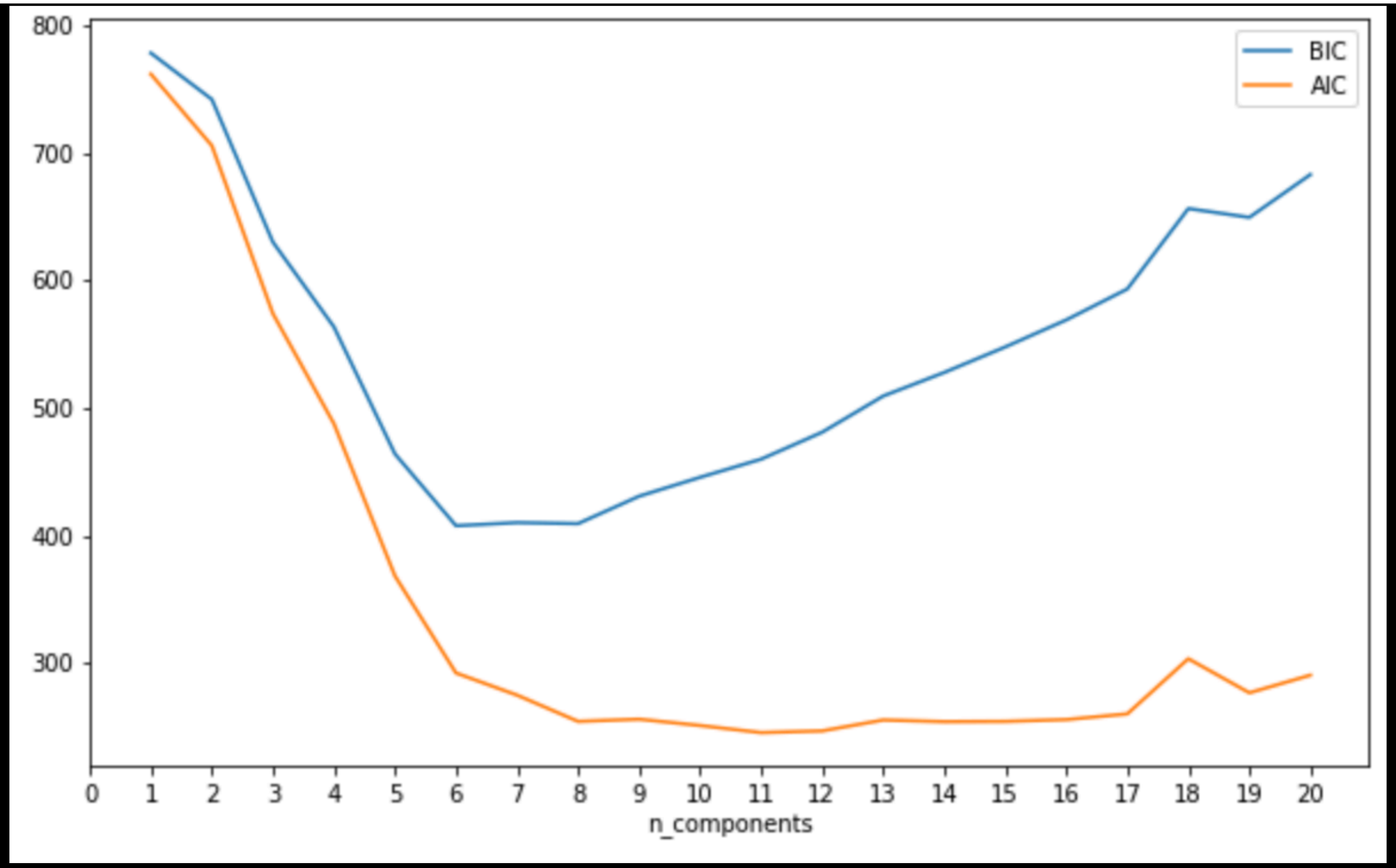


How Many Components?

- Similar problem to picking ‘k’ in k-Means
- We have one important advantage - being a probabilistic model, we know exactly what needs to be optimised - the likelihood of the data under the model
- We can use cross-validation to avoid over-fitting
- A useful pair of criteria are the Akaike Information Criterion (AIC), or the Bayesian Information Criterion (BIC)

How Many Components?

```
n_components = np.arange(1, 21)
models = [GMM(n, covariance_type='full', random_state=0).fit(Xmoon)
          for n in n_components]

fig, gmm11 = plt.subplots(figsize=(10, 6))
gmm11.plot(n_components, [m.bic(Xmoon) for m in models], label='BIC')
gmm11.plot(n_components, [m.aic(Xmoon) for m in models], label='AIC')
gmm11.legend(loc='best')
gmm11.set_xlabel('n_components')
gmm11.set_xticks(np.arange(0, 21, 1));
```



How Many Components?

- AIC tells us our initial fit of 16 components was probably too many, 8-12 components would have been a better choice

- BIC opts for simpler models than AIC, not necessarily a bad thing to aim for!

- The choice of number of components measures how well a GMM works as density estimator, not how well it can work as a clustering algorithm.

- So this optimisation can/should change, if you want a ‘classifier’

GMM Classification

- We have so far seen GMMs as ‘complex’ representations of a data distribution
- But GMMs can be used as a tool for classification
- If we have a labelled dataset of samples and classes, and each class has a ‘complex’ density form, then we could build a GMM per class
- The GMM would give us a probabilisitic reading of whether a future test sample was ‘generated’ from the GMM i.e. a likelihood of what we can normally expect
- The class model giving the best likelihood gives us a classification result

GMM Bayes Classifier

- Recall Gaussian Naive Bayes
- Apply the same principle, replacing a Gaussian with a GMM i.e. GMM Bayes

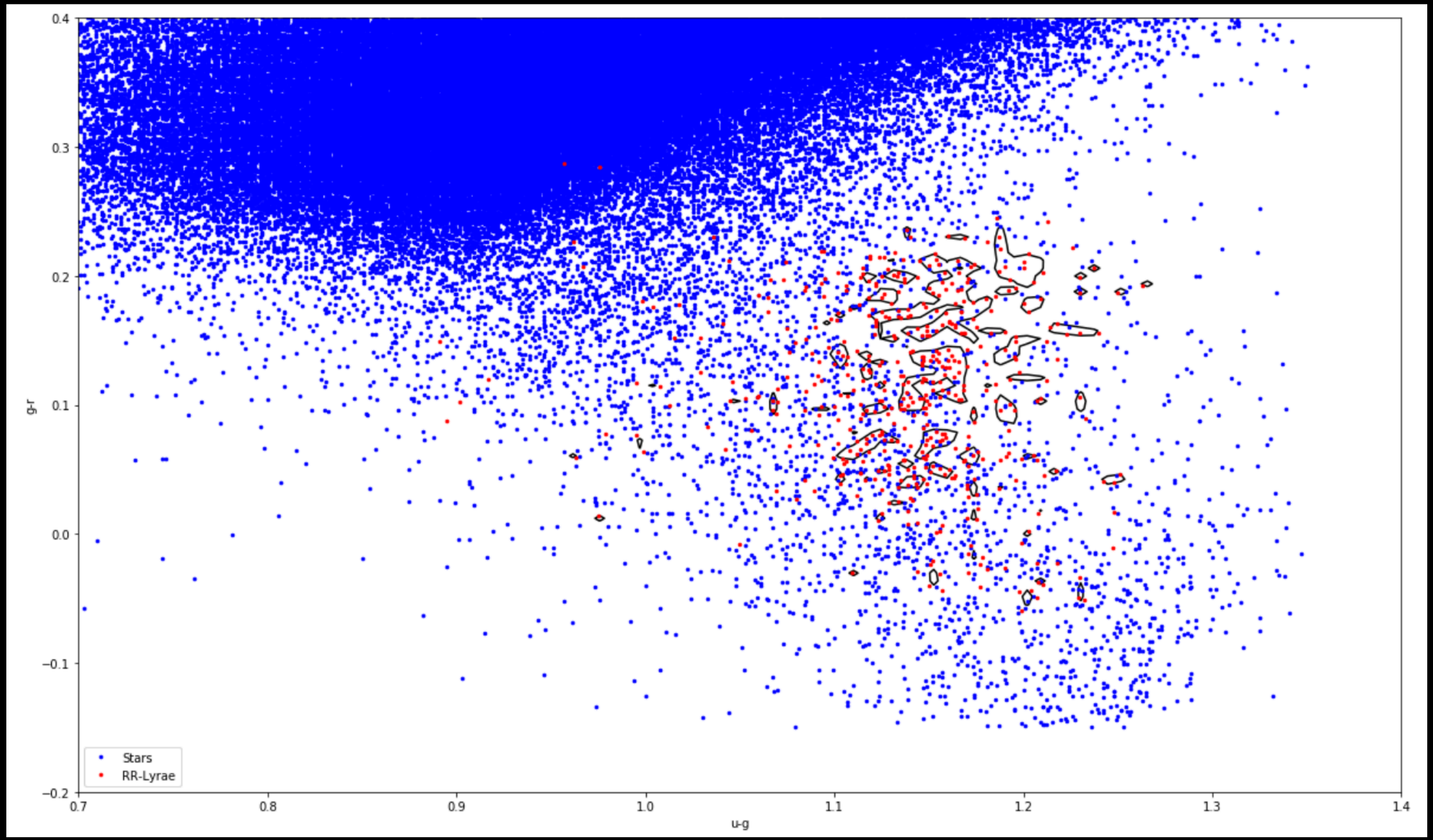
```
# Load up RRLyrae Dataset
data = np.load('./data/rrlyrae.npz')
samples = data['data']
labels = data['labels']

# Fit the NGMM aive Bayes classifier to all original dimensions
gmm_nb = GMMBayes(128) # 128 components per class
gmm_nb.fit(samples, labels)

# now predict
labels_pred = gmm_nb.predict(samples)

#get completeness score (equivalent to recall)
completeness_score = recall_score(labels,labels_pred)
#get contamination score (equivalent to 1-precision)
contamination_score = (1-precision_score(labels,labels_pred))

print('Completeness: %f'%completeness_score)
print('Contamination: %f'%contamination_score)
```

```
Completeness: 0.983437
Contamination: 0.074074
```

GMM Bayes Classifier

```
# Load up RRLyrae Dataset
data = np.load('./data/rrlyrae.npz')
samples = data['data']
labels = data['labels']
samples_2d = samples[:,0:2]

# stars are indicated by 0 labels, and rrlyrae by 1 labels
stars = (labels == 0)
rrlyrae = (labels == 1)

# Fit the GMM Naive Bayes classifier to all original dimensions
gmm_nb = GMMBayes(128) # 128 components per class
gmm_nb.fit(samples_2d, labels)

# predict the classification probabilities on a grid
xlim = (0.7, 1.4)
ylim = (-0.2, 0.4)
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100),
                     np.linspace(ylim[0], ylim[1], 100))

#convert to 1-D arrays
oned_xx = xx.ravel()
oned_yy = yy.ravel()

# get probabilities of each x,y coordinate from the NB classifier
# i.e. returns the probability of the samples for each class in the model.
Z = gmm_nb.predict_proba(np.column_stack((oned_xx,oned_yy)))
Z = Z[:, 1].reshape(xx.shape)

# Plot the samples again
fig, ax_gmm_nb = plt.subplots(figsize=(20, 12))
ax_gmm_nb = plt.axes()
ax_gmm_nb.plot(samples[stars, 0], samples[stars, 1], '.', ms=5, c='b', label='Stars')
ax_gmm_nb.plot(samples[rrlyrae, 0], samples[rrlyrae, 1], '.', ms=5, c='r', label='RR-Lyrae')

ax_gmm_nb.legend(loc=3)
ax_gmm_nb.set_xlabel('u-g')
ax_gmm_nb.set_ylabel('g-r')

# Now plot the contour between the two classes at p=0.5
# i.e. for every coordinate in xx,yy, if z is at 0.5
# (equal probability) of both classes, then draw the contour at that coordinate
ax_gmm_nb.contour(xx, yy, Z, [0.5], colors='k')

plt.show()
```

GMM Bayes Classifier



GMM Bayes Classifier

- Naive Bayes is now a lot more powerful, our assumptions about the data distribution are no longer simplistic
- How about running a comparison between:
    - Gaussian Naive Bayes
    - GMM Naive Bayes
    - LDA/QDA
    - Logistic Regression
    - SVC Linear
    - SVC Radial Basis Function
    - Tree Ensemble

Classifier Competition