lecture10-clustering

December 9, 2021

1 Lecture 10: Clustering

1.0.1 Applied Machine Learning

Volodymyr Kuleshov Cornell Tech

2 Part 1: Gaussian Mixture Models for Clustering

Clustering is a common unsupervised learning problem with numerous applications.

We will start by defining the problem and outlining some models for this problem.

3 Review: Unsupervised Learning

We have a dataset *without* labels. Our goal is to learn something interesting about the structure of the data: * Clusters hidden in the dataset. * Outliers: particularly unusual and/or interesting datapoints. * Useful signal hidden in noise, e.g. human speech over a noisy phone.

4 Clustering

Clustering is the problem of identifying distinct components in the data. * A cluster $C_k \subseteq \mathcal{X}$ can be thought of as a subset of the space $\mathcal{X}$. * Datapoints in a cluster are more similar to each other than to points in other clusters * Clusters are usually defined by their centers, and potentially by other shape parameters.

5 Review: K-Means

K-Means is the simplest example of a clustering algorithm. * The algorithm seeks to find K hidden clusters in the data. * Each cluster is characterized by its centroid (its mean). * The clusters reveal interesting structure in the data.

We seek centroids c_k such that the distance between the points and their closest centroid is minimized:

$$J(\theta) = \sum_{i=1}^{n} ||x^{(i)} - \text{centroid}(f_\theta(x^{(i)}))||,$$

where $\text{centroid}(k) = c_k$ denotes the centroid for cluster k.

This is best illustrated visually (from Wikipedia):

K-Means has a number of limitations: * Clustering can get stuck in local minima * Measuring clustering quality is hard and relies on heuristics * Cluster assignment is binary and doesn't estimate confidence

6 Gaussian Mixture Models

Gaussian mixtures define a model of the form:

$$P_\theta(x, z) = P_\theta(x|z)P_\theta(z)$$

* $z \in \mathcal{Z} = \{1, 2, \ldots, K\}$ is discrete and follows a categorical distribution $P_\theta(z = k) = \phi_k$. * $x \in \mathbb{R}$ is continuous; conditioned on $z = k$, it follows a Normal distribution $P_\theta(x|z = k) = \mathcal{N}(\mu_k, \Sigma_k)$.

The parameters θ are the μ_k, Σ_k, ϕ_k for all $k = 1, 2, \ldots, K$.

7 Gaussian Mixtures for Clustering

GMMs are useful for supervised *and* unsupervised learning (clustering):

$$P_\theta(x, z) = P_\theta(x|z)P_\theta(z)$$

* This model postulates that our observed data is comprised of K clusters with proportions specified by $\phi_1, \phi_2, \ldots, \phi_K$ * The points within each cluster follow a Normal distribution * Each point belongs to the cluster most likely to have generated it (according to $P_\theta(x|z = k)P_\theta(z = k)$)

Intuitively, a GMM represents well the two clusters in the geyser dataset:

Raw data	Single Gaussian	Mixture of Gaussians

We can also look at our Iris flower dataset.

[1]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn import datasets

# Load the Iris dataset
iris = datasets.load_iris(as_frame=True)

# print part of the dataset
iris_X, iris_y = iris.data, iris.target
pd.concat([iris_X, iris_y], axis=1).head()
```

[1]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
```

```
2                4.7               3.2               1.3               0.2
3                4.6               3.1               1.5               0.2
4                5.0               3.6               1.4               0.2
```

```
   target
0       0
1       0
2       0
3       0
4       0
```

Recall that we have previously seen how to fit GMMs on a labeled (x, y) dataset using using maximum likelihood.

Let's try this on our Iris dataset again (recall that this is called GDA).

[3]:
```
# we can implement these formulas over the Iris dataset
X = iris_X.to_numpy()[:,:2]
d = 2 # number of features in our toy dataset
K = 3 # number of clases
n = X.shape[0] # size of the dataset

# these are the shapes of the parameters
mus = np.zeros([K,d])
Sigmas = np.zeros([K,d,d])
phis = np.zeros([K])

# we now compute the parameters
for k in range(3):
    X_k = X[iris_y == k]
    mus[k] = np.mean(X_k, axis=0)
    Sigmas[k] = np.cov(X_k.T)
    phis[k] = X_k.shape[0] / float(n)
```

This code lets us generate predictions from the GMM model that we just learned by implementing Bayes' rule.

[5]:
```
# we can implement this in numpy
def gda_predictions(x, mus, Sigmas, phis):
    """This returns class assignments and p(y|x) under the GDA model.

    We compute \arg\max_y p(y|x) as \arg\max_y p(x|y)p(y)
    """
    # adjust shapes
    n, d = x.shape
    x = np.reshape(x, (1, n, d, 1))
    mus = np.reshape(mus, (K, 1, d, 1))
    Sigmas = np.reshape(Sigmas, (K, 1, d, d))
```

```
    # compute probabilities
    py = np.tile(phis.reshape((K,1)), (1,n)).reshape([K,n,1,1])
    pxy = (
        np.sqrt(np.abs((2*np.pi)**d*np.linalg.det(Sigmas))).reshape((K,1,1,1))
        * -.5*np.exp(
            np.matmul(np.matmul((x-mus).transpose([0,1,3,2]), np.linalg.
 ↪inv(Sigmas)), x-mus)
        )
    )
    pyx = pxy * py
    return pyx.argmax(axis=0).flatten(), pyx.reshape([K,n])

idx, pyx = gda_predictions(X, mus, Sigmas, phis)
```

We can visualize the GMM as follows:

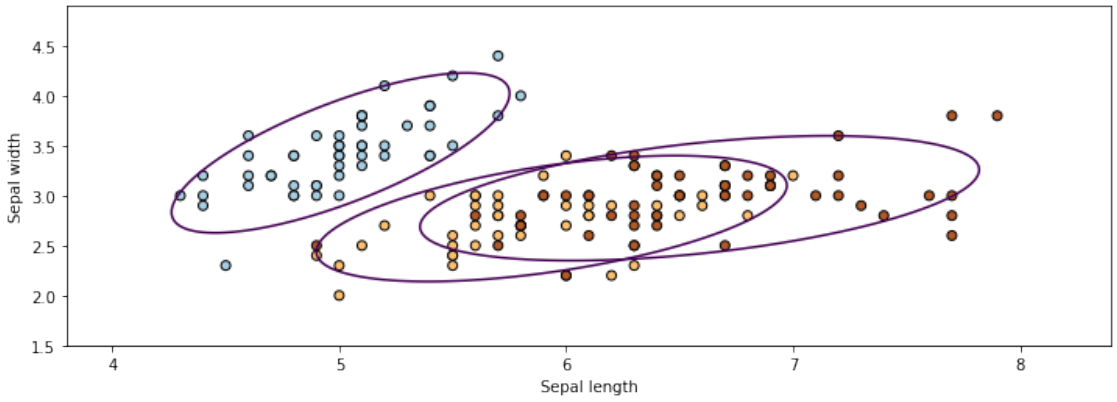
[10]:
```
# https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.
 ↪html
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm
plt.rcParams['figure.figsize'] = [12, 4]

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
Z, pyx = gda_predictions(np.c_[xx.ravel(), yy.ravel()], mus, Sigmas, phis)
logpy = np.log(-1./3*pyx)

# Put the result into a color plot
Z = Z.reshape(xx.shape)
contours = np.zeros([K, xx.shape[0], xx.shape[1]])
for k in range(K):
    contours[k] = logpy[k].reshape(xx.shape)
for k in range(K):
    plt.contour(xx, yy, contours[k], levels=np.logspace(0, 1, 1))

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=iris_y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.show()
```

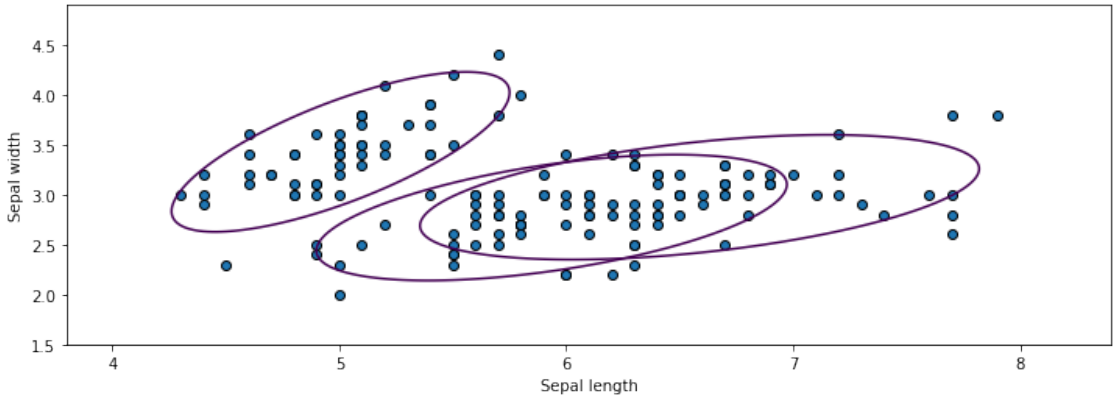


In this example, we used labels to train the GMM. But what if we don't have labels?

[11]:
```
# Put the result into a color plot
for k in range(K):
    plt.contour(xx, yy, contours[k], levels=np.logspace(0, 1, 1))

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.show()
```



These clusters intuitively make sense even on unlabeled data.

But how do we learn them if we don't have labels?

8 Maximum Marginal Likelihood Learning

Maximum marginal (log-)likelihood is a way of learning any proabilistic model on an unsupervised dataset $\mathcal{D}$ by maximizing:

$$\frac{1}{n}\sum_{i=1}^{n}\log P_\theta(x^{(i)}) = \frac{1}{n}\sum_{i=1}^{n}\log\left(\sum_{z\in\mathcal{Z}} P_\theta(x^{(i)}, z)\right).$$

- This asks P_θ to assign a high probability to the training data in $\mathcal{D}$.
- However, we need to use $P(x) = \sum_{z\in\mathcal{Z}} P(x, z)$ to compute this probability because z is not observed.

9 Interpretations of Marginal Likelihood

Recall that our objective is:

$$\max_\theta \frac{1}{n}\sum_{i=1}^{n}\log\left(\sum_{z\in\mathcal{Z}} P_\theta(x^{(i)}, z)\right)$$

- Our intuition of assigning high probability to data still holds
- z encodes the cluster id; we ask P_θ to be high for *at least one* z
- We are still minimizing KL divergence between P_θ and P_{data}

The Kullback-Leibler (KL) divergence $D(\cdot\|\cdot)$ between the model distribution and the data distribution is:

$$D(P_{\text{data}}\|P_\theta) = \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x})\log\frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}.$$

The KL divergence is always non-negative, and equals zero when P_{data} and P_θ are identical. This makes it a natural measure of similarity that's useful for comparing distributions.

10 Optimizing Marginal Likelihood

How do we optimize the marginal likelihood objective?

$$\max_\theta \frac{1}{n}\sum_{i=1}^{n}\log\left(\sum_{z\in\mathcal{Z}} P_\theta(x^{(i)}, z)\right)$$

- Note that we can't flip the sum and the log!
- Because of that, our closed-form solutions don't apply
- In fact, the objective now has many local minima!

Optimizing the likelihood of mixture models is hard.

A Gaussian has a single maximum, but a mixture has many and its objective is non-convex (hard to optimize).

11 Recovering Clusters from GMMs

Given a trained model $P_\theta(x, z) = P_\theta(x|z)P_\theta(z)$, we can look at the *posterior* probability

$$P_\theta(z = k \mid x) = \frac{P_\theta(z = k, x)}{P_\theta(x)} = \frac{P_\theta(x|z = k)P_\theta(z = k)}{\sum_{l=1}^{K} P_\theta(x|z = l)P_\theta(z = l)}$$

of a point x belonging to class k.

- The posterior defines a "soft" assignment of x to each class.
- This is in contrast to the hard assignments from K-Means.

12 Beyond Gaussian Mixtures

We will focus on Gaussian mixture models in this lecture, but there exist many other kinds of clustering: * Hierarchical clusters * Points belonging to multiple clusters (e.g. topics) * Clusters in graphs

See the `scikit-learn` guide for more!

Part 2: Expectation Maximization

We will now describe expecation maximization (EM), an algorithm that can be used to fit Gaussian mixture models.

13 Review: Gaussian Mixture Models

Gaussian mixtures are latent-variable probabilistic models that are useful for clustering. They define a model

$$P_\theta(x, z) = P_\theta(x|z)P_\theta(z)$$

* $z \in \{1, 2, \ldots, K\}$ is discrete and follows a categorical distribution $P_\theta(z = k) = \phi_k$. * $x \in \mathbb{R}$ is continuous; conditioned on $z = k$, it follows a Normal distribution $P_\theta(x|z = k) = \mathcal{N}(\mu_k, \Sigma_k)$.

The parameters θ are the μ_k, Σ_k, ϕ_k for all $k = 1, 2, \ldots, K$.

14 Review: Learning GMMs

Gaussian mixtures are latent variable models, and we can learn them using maximum marginal log-likelihood:

$$\max_\theta \sum_{x \in \mathcal{D}} \log P_\theta(x) = \max_\theta \sum_{x \in \mathcal{D}} \log \left(\sum_{z \in \mathcal{Z}} P_\theta(x, z) \right)$$

- Unlike in supervised learning, cluster assignments are latent.
- Hence, there is not a closed form solution for θ.
- We will see specialized algorithm for this task.

15 Expectation Maximization: Intuition

Expecation maximization (EM) is an algorithm for maximizing marginal log-likelihood

$$\max_{\theta} \sum_{x^{(i)} \in \mathcal{D}} \log \left(\sum_{z \in \mathcal{Z}} P_\theta(x^{(i)}, z) \right)$$

that can also be used to learn Gaussian mixtures.

We want to optimize the marginal log-likelihood

$$\max_{\theta} \sum_{x^{(i)} \in \mathcal{D}} \log \left(\sum_{z \in \mathcal{Z}} P_\theta(x^{(i)}, z) \right).$$

* If we know the true $z^{(i)}$ for each $x^{(i)}$, we maximize

$$\max_{\theta} \sum_{x^{(i)}, z^{(i)} \in \mathcal{D}} \log \left(P_\theta(x^{(i)}, z^{(i)}) \right).$$

and it's easy to find the best θ (use solution for supervised learning). * If we know θ, we can estimate the cluster assignments $z^{(i)}$ for each i by computing $P_\theta(z|x^{(i)})$.

Expectation maximization alternates between these two steps.

1. (**E-Step**) Given an estimate θ_t of the weights, compute $P_\theta(z|x^{(i)})$. and use it to "hallucinate" expected cluster assignments $z^{(i)}$.
2. (**M-Step**) Find a new θ_{t+1} that maximizes the marginal log-likelihood by optimizing $P_\theta(x^{(i)}, z^{(i)})$ given the $z^{(i)}$ from step 1.

This process increases the marginal likelihood at each step and eventually converges.

16 Expectation Maximization: Definition

Formally, EM learns the parameters θ of a latent-variable model $P_\theta(x, z)$ over a dataset $\mathcal{D} = \{x^{(i)} \mid i = 1, 2, ..., n\}$ as follows.

For $t = 0, 1, 2, \ldots$, repeat until convergence: 1. (**E-Step**) For each $x^{(i)} \in \mathcal{D}$ compute $P_{\theta_t}(z|x^{(i)})$ 2. (**M-Step**) Compute new weights θ_{t+1} as

$$\theta_{t+1} = \arg\max_{\theta} \sum_{i=1}^{n} \mathbb{E}_{z^{(i)} \sim P_{\theta_t}(z|x^{(i)})} \log P_\theta(x^{(i)}, z^{(i)})$$

Since assignments $P_{\theta_t}(z|x^{(i)})$ are "soft", M-step involves an expectation.

17 Expectation Maximization: Definition

Formally, EM learns the parameters θ of a latent-variable model $P_\theta(x, z)$ over a dataset $\mathcal{D} = \{x^{(i)} \mid i = 1, 2, ..., n\}$ as follows.

For $t = 0, 1, 2, \ldots$, repeat until convergence: 1. (**E-Step**) For each $x^{(i)} \in \mathcal{D}$ compute $P_{\theta_t}(z|x^{(i)})$ 2. (**M-Step**) Compute new weights θ_{t+1} as

$$\begin{aligned}\theta_{t+1} &= \arg\max_{\theta} \sum_{i=1}^{n} \mathbb{E}_{z^{(i)} \sim P_{\theta_t}(z|x^{(i)})} \log P_\theta(x^{(i)}, z^{(i)}) \\ &= \arg\max_{\theta} \sum_{i=1}^{n} \sum_{k=1}^{K} P_{\theta_t}(z = k|x^{(i)}) \log P_\theta(x^{(i)}, z = k)\end{aligned}$$

18 Understanding the E-Step

Intuitively, we hallucinate $z^{(i)}$ in the E-Step.

In practice, the $P_{\theta_t}(z|x^{(i)})$ define "soft" assignments, and we compute a vector of class probabilities for each $x^{(i)}$.

19 Understanding the M-Step

Since class assignments from E-step are probabilistic, we maximize an expectation:

$$\begin{aligned}\theta_{t+1} &= \arg\max_{\theta} \sum_{i=1}^{n} \mathbb{E}_{z^{(i)} \sim P_{\theta_t}(z|x^{(i)})} \log P_\theta(x^{(i)}, z^{(i)}) \\ &= \arg\max_{\theta} \sum_{i=1}^{n} \sum_{k=1}^{K} P_{\theta_t}(z = k|x^{(i)}) \log P_\theta(x^{(i)}, z = k)\end{aligned}$$

For many interesting models, this is tractable.

20 Pros and Cons of EM

EM is a very important optimization algorithm in machine learning. * It is easy to implement and is guaranteed to converge. * It works in a lot of imporant ML models.

Its limitations include: * It can get stuck in local optima. * We may not be able to compute $P_{\theta_t}(z|x^{(i)})$ in every model.

Part 3: Expectation Maximization in Gaussian Mixture Models

Next, let's work through how Expectation Maximization works in Gaussian Mixture Models.

21 Review: Gaussian Mixture Models

Gaussian mixtures define a model of the form:

$$P_\theta(x, z) = P_\theta(x|z)P_\theta(z)$$

* $z \in \mathcal{Z} = \{1, 2, \ldots, K\}$ is discrete and follows a categorical distribution $P_\theta(z = k) = \phi_k$. * $x \in \mathbb{R}$ is continuous; conditioned on $z = k$, it follows a Normal distribution $P_\theta(x|z = k) = \mathcal{N}(\mu_k, \Sigma_k)$.

The parameters θ are the μ_k, Σ_k, ϕ_k for all $k = 1, 2, \ldots, K$.

22 Review: Expectation Maximization

Formally, EM learns the parameters θ of a latent-variable model $P_\theta(x, z)$ over a dataset $\mathcal{D} = \{x^{(i)} \mid i = 1, 2, ..., n\}$ as follows.

For $t = 0, 1, 2, \ldots$, repeat until convergence: 1. (**E-Step**) For each $x^{(i)} \in \mathcal{D}$ compute $P_{\theta_t}(z|x^{(i)})$ 2. (**M-Step**) Compute new weights θ_{t+1} as

$$\theta_{t+1} = \arg\max_\theta \sum_{i=1}^{n} \mathbb{E}_{z^{(i)} \sim P_{\theta_t}(z|x^{(i)})} \log P_\theta(x^{(i)}, z^{(i)})$$

Since assignments $P_{\theta_t}(z|x^{(i)})$ are "soft", M-step involves an expectation.

23 Deriving the E-Step

In the E-step, we compute the posterior for each data point x as follows

$$P_\theta(z = k \mid x) = \frac{P_\theta(z = k, x)}{P_\theta(x)} = \frac{P_\theta(x|z = k)P_\theta(z = k)}{\sum_{l=1}^{K} P_\theta(x|z = l)P_\theta(z = l)}$$

$P_\theta(z \mid x)$ defines a vector of probabilities that x originates from component k given the current set of parameters θ

24 Deriving the M-Step

At the M-step, we optimize the expected log-likelihood of our model.

$$\max_\theta \sum_{x \in D} \mathbb{E}_{z \sim P_{\theta_t}(z|x)} \log P_\theta(x, z) =$$
$$\max_\theta \left(\sum_{k=1}^{K} \sum_{x \in D} P_{\theta_t}(z_k|x) \log P_\theta(x|z_k) + \sum_{k=1}^{K} \sum_{x \in D} P_{\theta_t}(z_k|x) \log P_\theta(z_k) \right)$$

As in supervised learning, we can optimize the two terms above separately.

We will start with $P_\theta(x \mid z = k) = \mathcal{N}(x; \mu_k, \Sigma_k)$. We have to find μ_k, Σ_k that optimize

$$\max_\theta \sum_{x^{(i)} \in D} P(z = k|x^{(i)}) \log P_\theta(x^{(i)}|z = k)$$

Note that this corresponds to fitting a Gaussian to a dataset whose elements $x^{(i)}$ each have a weight $P(z = k|x^{(i)})$.

Similarly to how we did this the supervised regime, we compute the derivative, set it to zero, and

obtain closed form solutions:

$$\mu_k = \frac{\sum_{i=1}^{n} P(z = k|x^{(i)})x^{(i)}}{n_k}$$
$$\Sigma_k = \frac{\sum_{i=1}^{n} P(z = k|x^{(i)})(x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^\top}{n_k}$$
$$n_k = \sum_{i=1}^{n} P(z = k|x^{(i)})$$

Intuitively, the optimal mean and covariance are the emprical mean and convaraince of the dataset $\mathcal{D}$ when each element $x^{(i)}$ has a weight $P(z = k|x^{(i)})$.

Similarly, we can show that the class priors are

$$\phi_k = \frac{n_k}{n}$$
$$n_k = \sum_{i=1}^{n} P(z = k|x^{(i)})$$

25 EM in Gaussian Mixture Models

EM learns the parameters θ of a Gaussian mixture model $P_\theta(x, z)$ over a dataset $\mathcal{D} = \{x^{(i)} \mid i = 1, 2, ..., n\}$ as follows.

For $t = 0, 1, 2, \ldots$, repeat until convergence: 1. (**E-Step**) For each $x^{(i)} \in \mathcal{D}$ compute $P_{\theta_t}(z|x^{(i)})$ 2. (**M-Step**) Compute parameters μ_k, Σ_k, ϕ_k using the above formulas

Part 4: Generalization in Probabilistic Models

Let's now revisit the concepts of overfitting and underfitting in GMMs.

26 Review: Data Distribution

We will assume that the dataset is sampled from a probability distribution $\mathbb{P}$, which we will call the *data distribution*. We will denote this as

$$x \sim \mathbb{P}.$$

The dataset $\mathcal{D} = \{x^{(i)} \mid i = 1, 2, ..., n\}$ consists of *independent and identicaly distributed* (IID) samples from $\mathbb{P}$.

27 Review: Gaussian Mixture Models

Gaussian mixtures are latent-variable probabilistic models that are useful for clustering. They define a model

$$P_\theta(x, z) = P_\theta(x|z)P_\theta(z)$$

* $z \in \{1, 2, \ldots, K\}$ is discrete and follows a categorical distribution $P_\theta(z = k) = \phi_k$. * $x \in \mathbb{R}$ is continuous; conditioned on $z = k$, it follows a Normal distribution $P_\theta(x|z = k) = \mathcal{N}(\mu_k, \Sigma_k)$.

The parameters θ are the μ_k, Σ_k, ϕ_k for all $k = 1, 2, \ldots, K$.

28 Review: Generalization

In machine learning, **generalization** is the property of predictive models to achieve good performance on new, heldout data that is distinct from the training set.

How does generalization apply to probabilistic unsupervised models like GMMs?

29 An Unsupervised Learning Dataset

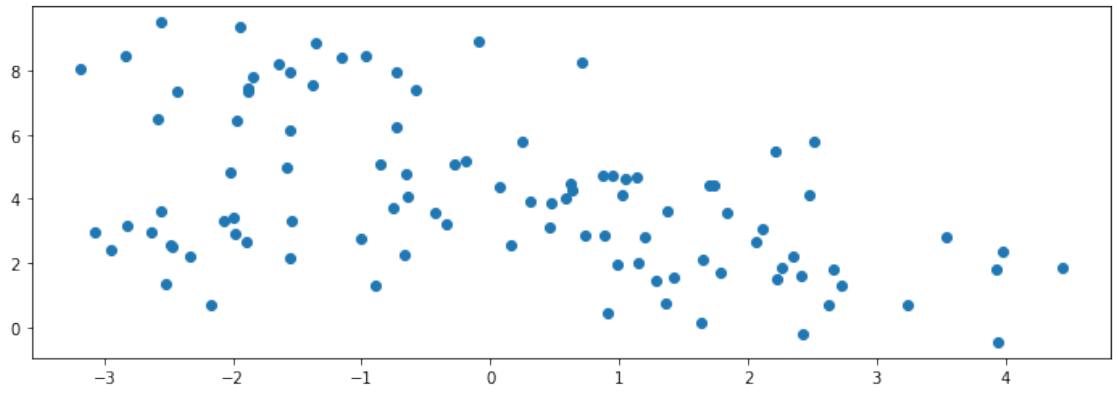
Consider the following dataset, consisting of a mixture of Gaussians.

[49]:
```
import numpy as np
from sklearn import datasets
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize'] = [12, 4]

# generate 150 random points
np.random.seed(0)
X_all, y_all = datasets.make_blobs(150, centers=4)

# use the first 100 points as the main dataset
X, y = X_all[:100], y_all[:100]
plt.scatter(X[:,0], X[:,1])
```

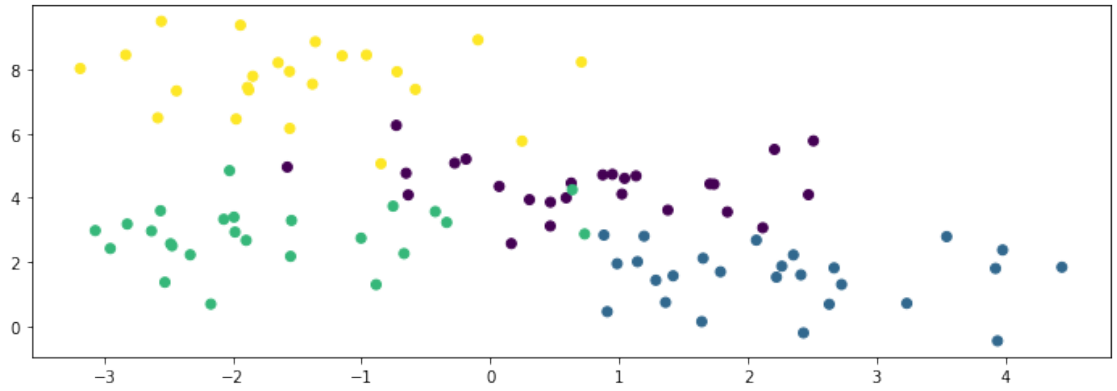
[49]:
```
<matplotlib.collections.PathCollection at 0x12b583780>
```



We know the true labels of these clusers, and we can visualize them.

[50]:
```
plt.scatter(X[:,0], X[:,1], c=y)
```

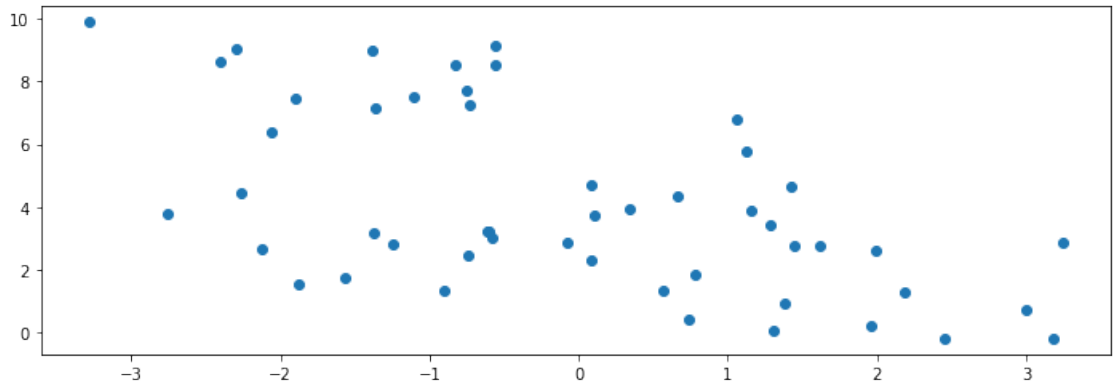
[50]:
```
<matplotlib.collections.PathCollection at 0x115b872b0>
```



We will also keep 50 points as a holdout set.

[51]:
```
# use the last 50 points as a holdout set
X_holdout, y_holdout = X_all[100:], y_all[100:]
plt.scatter(X_holdout[:,0], X_holdout[:,1])
```

[51]:
```
<matplotlib.collections.PathCollection at 0x12addcf98>
```



30 Underfitting in Unsupervised Learning

Underfitting happens when we are not able to fully learn the signal hidden in the data.

In the context of GMMs, this means not capturing all the clusters in the data.

Let's fit a GMM on our toy dataset.

[52]:
```
# fit a GMM
from sklearn import mixture
model = mixture.GaussianMixture(n_components=2)
model.fit(X)
```

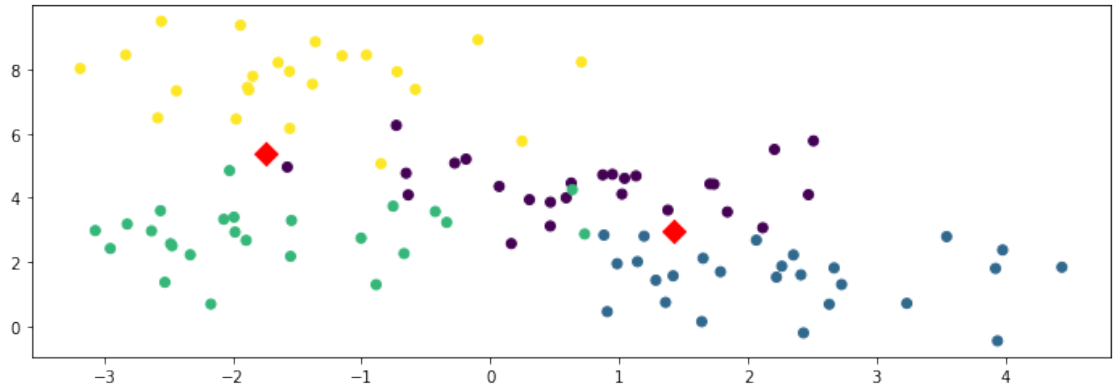
[52]:
```
GaussianMixture(n_components=2)
```

The model finds two distinct components in the data, but they fail to capture the true structure.

We can also measure the value of our objective (the log-likelihood) on the training and holdout sets.

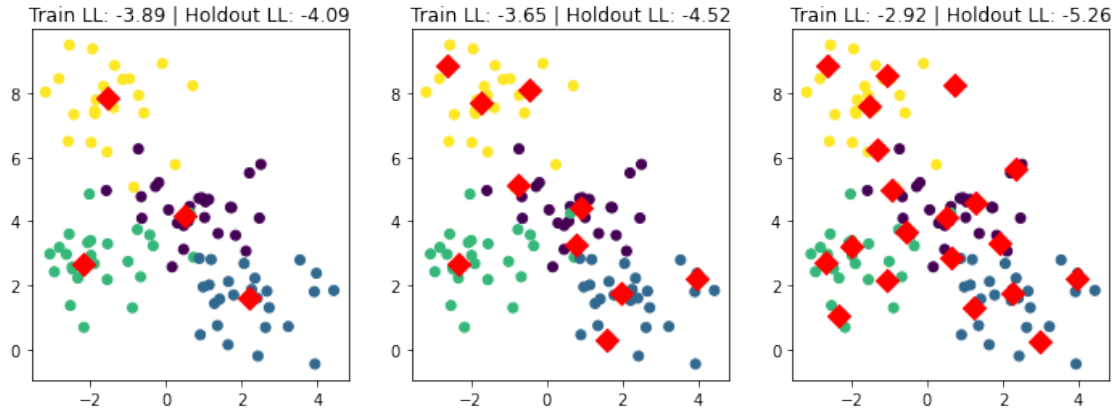
[53]:
```
plt.scatter(X[:,0], X[:,1], c=y)
plt.scatter(model.means_[:,0], model.means_[:,1], marker='D', c='r', s=100)
print('Training Set Log-Likelihood (higher is better): %.2f' % model.score(X))
print('Holdout Set Log-Likelihood (higher is better): %.2f' % model.
 ↪score(X_holdout))
```

```
Training Set Log-Likelihood (higher is better): -4.09
Holdout Set Log-Likelihood (higher is better): -4.22
```



Consider now what happens if we further increase the number of clusters.

[54]:
```
Ks = [4, 10, 20]
f, axes = plt.subplots(1,3)
for k, ax in zip(Ks, axes):
    model = mixture.GaussianMixture(n_components=k)
    model.fit(X)
    ax.scatter(X[:,0], X[:,1], c=y)
    ax.scatter(model.means_[:,0], model.means_[:,1], marker='D', c='r', s=100)
    ax.set_title('Train LL: %.2f | Holdout LL: %.2f' % (model.score(X), model.
 ↪score(X_holdout)))
```



31 Overfitting in Unsupervised Learning

Overfitting happens when we fit the noise, but not the signal.

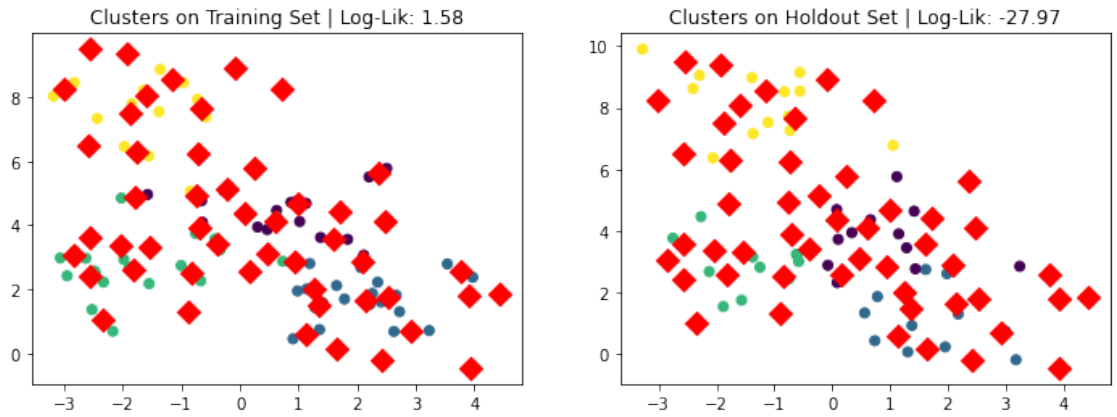
In our example, this means fitting small, local noise clusters rather than the true global clusters.

[55]:
```
model = mixture.GaussianMixture(n_components=50)
model.fit(X)

plt.subplot(121)
plt.title('Clusters on Training Set | Log-Lik: %.2f' % model.score(X))
plt.scatter(X[:,0], X[:,1], c=y)
plt.scatter(model.means_[:,0], model.means_[:,1], marker='D', c='r', s=100)

plt.subplot(122)
plt.title('Clusters on Holdout Set | Log-Lik: %.2f' % model.score(X_holdout))
plt.scatter(X_holdout[:,0], X_holdout[:,1], c=y_holdout)
plt.scatter(model.means_[:,0], model.means_[:,1], marker='D', c='r', s=100)
```

[55]:
```
<matplotlib.collections.PathCollection at 0x12aeb95c0>
```



32 Measuring Generalization Using Log-Likelihood

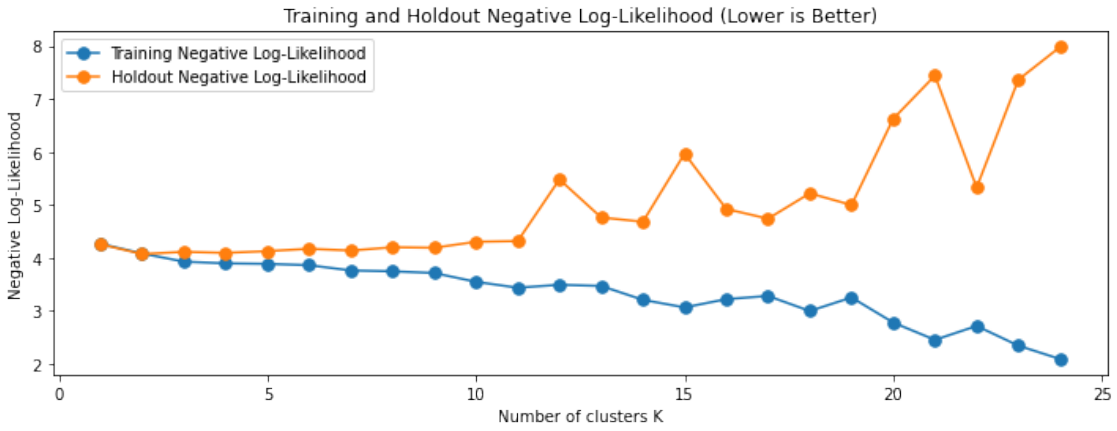
Probabilistic unsupervised models optimize an objective that can be used to detect overfitting and underfitting by comparing performance between training and holdout sets.

Below, we visualize the performance (measured via negative log-likelihood) on training and holdout sets as K increases.

[56]:
```
Ks, training_objs, holdout_objs = range(1,25), [], []
for k in Ks:
    model = mixture.GaussianMixture(n_components=k)
    model.fit(X)
    training_objs.append(-model.score(X))
    holdout_objs.append(-model.score(X_holdout))

plt.plot(Ks, training_objs, '.-', markersize=15)
plt.plot(Ks, holdout_objs, '.-', markersize=15)
plt.xlabel("Number of clusters K")
plt.ylabel("Negative Log-Likelihood")
plt.title("Training and Holdout Negative Log-Likelihood (Lower is Better)")
plt.legend(['Training Negative Log-Likelihood', 'Holdout Negative␣
 ↪Log-Likelihood'])
```

[56]:
```
<matplotlib.legend.Legend at 0x12c463320>
```



Warning: This process doesn't work as well as in supervised learning

For example, detecting overfitting with larger datasets will be paradoxically harder (try it!)

33 Summary

- Generalization is important for supervised and unsupervised learning.
- A probabilistic model can detect overfitting by comparing the likelihood of training data vs. that of holdout data.
- We can reduce overfitting by making the model less expressive.