PML Sheet 9

Julian Zimmerlin 6[illegible]
Leander Zimmermann 1[illegible]

Theory

E-step: $p(z \mid \mu, \Sigma, \pi, x)$:

$$p(z_{ij}=1 \mid \mu, \Sigma, \pi, x_i) \overset{\text{Bayes}}{=} \frac{p(x_i \mid z_{ij}=1, \mu, \Sigma) \cdot p(z_{ij}=1 \mid \mu, \Sigma, \pi)}{\sum_{j'} p(x_i \mid z_{ij'}, \mu, \Sigma) \cdot p(z_{ij'}=1 \mid \pi)} \quad (= p(z_{ij}=1 \mid \pi) = \pi_j)$$

$$= \frac{\pi_j \mathcal{N}(x_i; \mu_j, \Sigma_j)}{\sum_j \pi_j \mathcal{N}(x_i; \mu_j, \Sigma_j)} =: \gamma_{ij}$$

Complete-data likelihood:

$$p(x, z \mid \mu, \Sigma, \pi) = \prod_{i=1}^{n} p(x_i, z_i \mid \mu, \Sigma, \pi) \quad \| \; x_i \text{ independent given } \theta$$

$$= \prod_{i=1}^{n} p(x_i \mid z_i, \mu, \Sigma) \cdot p(z_i \mid \pi) \quad \| \; x_i \perp\!\!\!\perp \pi \mid z_i, \; z_i \perp\!\!\!\perp \mu, \Sigma \mid \pi$$

$$= \prod_{i=1}^{n} \prod_{j=1}^{k} \mathcal{N}(x_i; \mu_j, \Sigma_j)^{z_{ij}} \pi_j^{z_{ij}}$$

$$\mathbb{E}_{p(z \mid x, \mu, \Sigma, \pi)}\left[p(x, z \mid \mu, \Sigma, \pi)\right] = \sum_z p(x, z \mid \mu, \Sigma, \pi) \cdot p(z \mid x, \mu, \Sigma, \pi)$$

$$= \sum_{z \in \{0,1\}^{n \times k}} \prod_{i=1}^{n} \prod_{j=1}^{k} \mathcal{N}(x_i; \mu_j, \Sigma_j)^{z_{ij}} \pi_j^{z_{ij}} \cdot \left(\underbrace{\frac{\pi_j \mathcal{N}(x_i; \mu_j, \Sigma_j)}{\sum_{j'} \pi_{j'} \mathcal{N}(x_i; \mu_{j'}, \Sigma_{j'})}}_{\gamma_{ij}} \right)^{z_{ij}}$$

$$= \sum_{z \in \{0,1\}^{n \times k}} \prod_{i=1}^{n} \prod_{j=1}^{k} \left(\gamma_{ij} \cdot \pi_j \mathcal{N}(x_i; \mu_j, \Sigma_j) \right)^{z_{ij}}$$

Complete-data LOG likelihood:

$$\log p(x, z \mid \mu, \Sigma, \pi) = \sum_{i=1}^{n} \sum_{j=1}^{k} z_{ij} \cdot \left(\log \pi_j + \log \mathcal{N}(x_i; \mu_j, \Sigma_j) \right)$$

$$\Rightarrow \mathbb{E}_{p(z \mid x, \mu, \Sigma, \pi)}\left[\log p(x, z \mid \mu, \Sigma, \pi)\right] = \sum_{i=1}^{n} \sum_{j=1}^{k} \gamma_{ij} \cdot \left(\log \pi_j + \log \mathcal{N}(x_i; \mu_j, \Sigma_j) \right)$$

M-Step

Means:

$$\nabla_{\mu_j} \sum_{ij} r_{ij} \left(\log \pi_j + \log \mathcal{N}(x_i \mid \mu_j, \Sigma_j)\right)$$

$$= \sum_{ij} r_{ij} \cdot \frac{\nabla_{\mu_j} \left(\frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \cdot \exp\left(-\frac{1}{2}(x_i-\mu_j)^T \Sigma^{-1}(x_i-\mu_j)\right)\right)}{\mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

$$= \sum_{ij} r_{ij} \frac{\mathcal{N}(x_i \mid \mu_j, \Sigma_j)}{\mathcal{N}(x_i \mid \mu_j, \Sigma_j)} \cdot \left(-\frac{1}{2} 2\Sigma^{-1}(x_i-\mu_j) \cdot (-1) \cdot \mathbb{1}_{j=j}\right)$$

$$= \sum_{ij} r_{ij} \Sigma^{-1}(x_i-\mu_j) \overset{!}{=} 0 \quad | \cdot \Sigma$$

$$\Leftrightarrow \sum_i r_{ij} x_i = \sum_i r_{ij} \mu_j \Leftrightarrow \mu_j = \frac{1}{\sum_i r_{ij}} \sum_i r_{ij} x_i$$

π_j:

$$L(\pi, \lambda) = \sum_{ij} r_{ij} \left(\log \pi_j + \log \mathcal{N}(x_i \mid \mu_j, \Sigma_j)\right) + \lambda \left(\sum_j \pi_j - 1\right)$$

$$\frac{d}{d\pi_j} L(\pi, \lambda) = \sum_{ij} r_{ij} \frac{1}{\pi_j} \cdot \mathbb{1}_{j=j} + \lambda \sum_j \mathbb{1}_{j=j}$$

$$= \sum_i r_{ij} \frac{1}{\pi_j} + \lambda \overset{!}{=} 0 \Rightarrow \frac{1}{\pi_j} = \frac{-\lambda}{\sum_i r_{ij}} \Leftrightarrow \pi_j = -\frac{\sum r_{ij}}{\lambda}$$

$$\frac{d}{d\lambda} L(\pi, \lambda) = \sum_j \pi_j - 1 \overset{!}{=} 0$$

$$\Rightarrow \sum_j \frac{\sum_i r_{ij}}{-\lambda} - 1 = 0 \Leftrightarrow \frac{1}{-\lambda} \underbrace{\sum_{ji} r_{ij}}_{=n} = 1 \Leftrightarrow \lambda = -n$$

$$\Rightarrow \pi_j = \frac{\sum_i r_{ij}}{n}$$

Covariances

$$\nabla_{\Sigma_j} \sum_{ij} r_{ij} \left(\log \pi_j + \log \mathcal{N}(x_i \mid \mu_j, \Sigma_j)\right)$$

$f(\Sigma_j) = |\Sigma|^{-1/2}$

$g(\Sigma_j) = \exp(\dots)$

$$= \sum_{ij} r_{ij} \frac{1}{\mathcal{N}(x_i \mid \mu_j, \Sigma_j)} \nabla_{\Sigma_j} \left(\frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \cdot \exp\left(-\frac{1}{2}(x_i-\mu_j)^T \Sigma^{-1}(x_i-\mu_j)\right)\right)$$

$$= \sum_i r_{ij} \frac{1}{\mathcal{N}(x_i \mid \mu_j, \Sigma_j)} \cdot \frac{1}{(2\pi)^{d/2}} \left(|\Sigma_j|^{-1/2} \cdot \exp\left(-\frac{1}{2}(x_i-\mu_j)^T \Sigma_j^{-1}(x_i-\mu_j)\right) \cdot -\frac{1}{2} \cdot \right.$$

$$\left. -\left(\Sigma_j^{-T}(x_i-\mu_j)(x_i-\mu_j)^T \Sigma_j^{-T}\right) + -\frac{1}{2} |\Sigma_j|^{-\frac{3}{2}} \cdot |\Sigma_j| \Sigma_j^{-T} \cdot \exp\left(-\frac{1}{2}(x_i-\mu_j)^T \Sigma_j^{-1}\right.\right.$$

$$= \sum_i r_{ij} \cdot \left(\frac{1}{2} \left|\Sigma_j^{-T}(x_i-\mu_j)(x_i-\mu_j)^T \Sigma_j^{-T}\right| + \frac{1}{2} \Sigma_j^{-T}\right) \overset{!}{=} 0$$

$$\sum_i r_{ij} \left(\frac{1}{2} \Sigma_j^{-1} (x_i - \mu_j)(x_i - \mu_j)^T \Sigma_j^{-1} - \frac{1}{2} \Sigma_j^{-1} \right) = 0 \quad | \cdot \frac{1}{2} \Sigma_j \cdot \Sigma_j^{-1} \text{ eingeben}$$

$$\Leftrightarrow \sum_i r_{ij} \left((x_i - \mu_j)(x_i - \mu_j)^T \Sigma_j^{-1} - I \right) = 0$$

$$\Leftrightarrow \sum_i r_{ij} (x_i - \mu_j)(x_i - \mu_j)^T \Sigma_j^{-1} = I \cdot \sum_i r_{ij} \quad | \cdot \Sigma_j \; | : \sum_i r_{ij}$$

$$\Leftrightarrow \Sigma_j = \frac{1}{\sum_i r_{ij}} \sum_i r_{ij} (x_i - \mu_j)(x_i - \mu_j)^T$$

Ex09_Julian_Zimmerlin_6009977___Leander_Zimmermann_4165446

June 30, 2022

[1]:
```
import numpy as np
import scipy
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd
from seaborn import pairplot
from typing import Tuple
from sklearn.cluster import KMeans
```

1 Exercise 3: Gaussian Mixture Model EM-Algorithm

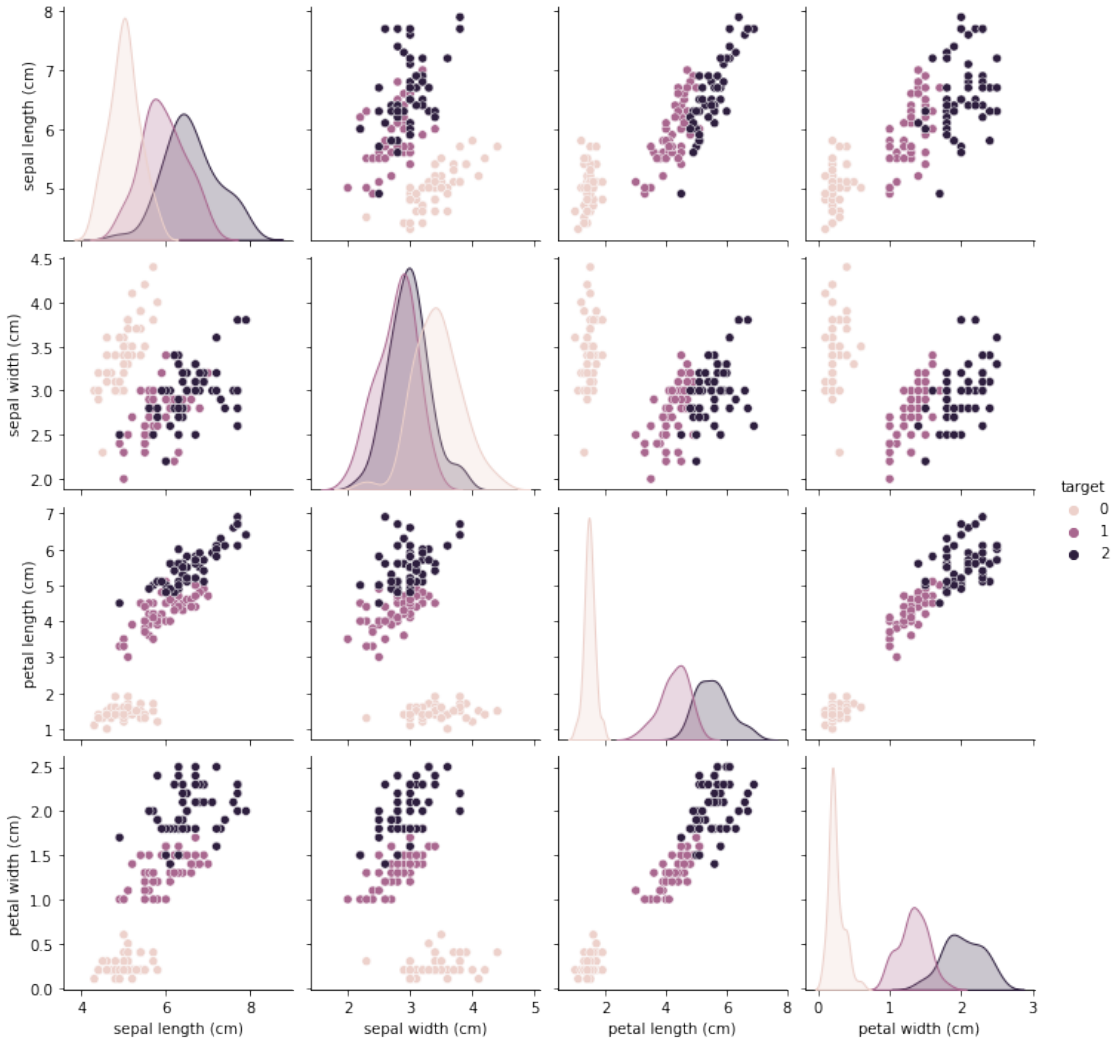
In this notebook we will implement the EM-Algorithm for GMMs and apply it to "real" dataset: Iris. This is perhaps the best known dataset in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The are four features, the sepal length in cm the sepal width in cm, the petal length in cm and the petal width in cm.

The next two cells will load the dataset into the notebook.

[2]:
```
# Data as dataframe
data = load_iris(as_frame=True)

df_X = data["data"]
df_y = data["target"]

# Train data in numpy arrays
X = df_X.to_numpy()
y = df_y.to_numpy()

# Dataset in a df for plotting
df = pd.concat([df_X, df_y], axis=1)
```

[3]:
```
pairplot(df, hue="target")
```

[3]:
```
<seaborn.axisgrid.PairGrid at 0x7f78105c75e0>
```



Based on the above pairplot one can already guess that a Gaussian Mixture model can nicely model this data, so let's do that!

1.1 Exercise 3.1

Here we will implement the E-step and M-step as functions. You already derived all the formulas for the EM algorithm by yourself in Exercise 2. Here are the solutions you should have arrived at:

1.1.1 E-step

This requires to compute

$$r_{ij} = \frac{\pi_j^{old} \mathcal{N}(x_i; \mu_j^{old}, \Sigma_j^{old})}{\sum_{j=1}^{k} \pi_j^{old} \mathcal{N}(x_i; \mu_j^{old}, \Sigma_j^{old})}$$

for each datapoint. This equals the posterior probabilities of cluster assignments given the current parameter candidates.

NOTE: For numerical stability, you may want to perform most computations in log-space and perform normalization using the "logsumexp" trick (i.e. using np.logsumexp).

1.1.2 M-step

The maximization is possible in closed form, and given by:

$$R_j = \sum_{i=1}^{n} r_{ij} \qquad \pi_j = \frac{R_j}{n} \qquad \mu_j = \frac{1}{R_j}\sum_{i=1}^{n} r_{ij} x_i \qquad \Sigma_j = \frac{1}{R_j}\sum_{i=1}^{n} r_{ij}(x_i - \mu_j)(x_i - \mu_j)^T$$

NOTE: There are some numerical considerations * In theory $r_{ij} \neq 0$ for all j, i. Yet in practise, we use floating point numbers which have limited precision. As a result e.g. R_j can be zero if all datapoints are far away from a cluster, which would lead to a division by zero. * The covariance matrix should be symmetric postive definite. It may be required to ensure this i.e. by adding a small value to the diagonals.

[4]:
```
# Some variables you may want to use
n = X.shape[0] # Number of samples
d = X.shape[1] # Dimensionality of each sample

# Here we know the number of clusters, there are only 3 types of iris plants in␣
 ↪the data.
k = 3
```

[118]:
```
def E_step(X, pi, mu, cov) -> np.array:
    """
    Perform the E step

    args:
        X (n, d): data, n samples of dimensionality d
        pi (k): mixture weights of Gaussians
        mu (k,d): means of Gaussians
        cov (k,d,d): covariances of Gaussians
    returns:
        r (n,k): posterior probability for cluster assignments
                    for each datatpoint
    """
    n = X.shape[0]  # Number of samples
    d = X.shape[1]  # Dimensionality of each sample
    k = pi.shape[0] # number of clusters

    r = np.zeros((n,k))

    for i, x_i in enumerate(X):
        norm = np.sum([pi[j] * scipy.stats.multivariate_normal.pdf(x_i,␣
 ↪mean=mu[j], cov=cov[j]) for j in range(k)])
        for j in range(k):
```

```
            g = scipy.stats.multivariate_normal.pdf(x_i, mean=mu[j], cov=cov[j])
            r[i,j] = (pi[j]*g) / norm
    return r
```

[140]:
```
# Test for correctness:
test1 = E_step(X, np.array([1/3, 1/3, 1/3]), np.ones((3,4)), [np.eye(4) for k␣
↪in range(3)])
test2 = E_step(X, np.array([1/3, 1/3, 1/3]), np.array([[-1.,0.,3., 0.], [0.,2.
↪,0., 1.], [5.,5.,5.,5.]]), [np.eye(4) for k in range(3)])
assert test1.shape[0] == n, "The first dimension should be of shape N"
assert test1.shape[1] == k, "The second dimension should be of shape K"
assert np.isclose(test1.mean(0), np.array([1/3, 1/3, 1/3]), atol=1e-2,␣
↪rtol=1e-2).all(), "If all clusters are the same, the posterior probabilities␣
↪should be uniform."
assert np.isclose(test2.mean(0), np.array([2.93392254e-05, 2.85799805e-01, 7.
↪14170855e-01]), atol=1e-2, rtol=1e-2).all(), "There is something wrong :("
```

[245]:
```
def M_step(r, X) -> Tuple[np.array, np.array, np.array]:
    """
    Maximize expectation of complete data log likelihood

    args:
        r (n,k): posterior probability for cluster assignments
                    for each datatpoint
    returns:
        X (n, d): data, n samples of dimensionality d
        pi (k): mixture weights of Gaussians
        mu (k,d): means of Gaussians
        cov (k,d,d): covariances of Gaussians
    """
    n = X.shape[0]  # Number of samples
    d = X.shape[1]  # Dimensionality of each sample
    k = r.shape[1]  # number of clusters

    R = r.sum(axis=0)
    assert R.shape == (k,), f"{R.shape}"

    pi = R / n
    assert pi.shape == (k,)

    mu = np.zeros((k,d))
    covs = np.zeros((k,d,d))
    for j in range(k):
        mu[j] = np.sum([r[i,j] * x_i for i, x_i in enumerate(X)], axis=0) * (1/
↪R[j])
        covs[j] = np.sum([r[i,j] * np.outer((x_i-mu[j]),(x_i-mu[j])) for i, x_i␣
↪in enumerate(X)], axis=0) * (1/R[j])
```

```
    return pi, mu, covs
```

[246]:
```
# Test for correctness
pi_test1, mu_test1, cov_test1 = M_step(test1, X)
assert pi_test1.shape[0] == k, "There should be one mixture coefficient for␣
 ↪each cluster K."
assert mu_test1.shape[0] == k and mu_test1.shape[1] == d, "The shape of the␣
 ↪cluster means should be (K, d)."
assert cov_test1.shape[0] == k and cov_test1.shape[1] == d and cov_test1.
 ↪shape[2] == d, "The shape of the cluster covariance matrices should be␣
 ↪(K,d,d)"
assert np.isclose(pi_test1.sum(), 1., atol=1e-3, rtol=1e-3), "The mixture␣
 ↪coefficients must sum to one."
assert all([np.all(np.linalg.eigvalsh(cov_test1[j]) > 0.) for j in range(k)]),␣
 ↪"The estimated covariance matrix is not positive definite"
assert np.isclose(pi_test1, np.array([1/3, 1/3, 1/3]), atol=1e-2, rtol=1e-2).
 ↪all() and np.isclose(mu_test1.mean(0), np.array([5.84333333, 3.05733333, 3.
 ↪758     , 1.19933333]), atol=1e-2, rtol=1e-2).all() and np.isclose(cov_test1.
 ↪mean(), 0.6058022499999997, atol=1e-2, rtol=1e-2) , "Something went wrong :("
```

1.2 Exercise 3.2

While not required for the algorithm, it is a good idea to compute the total data negative loglikelihood. Recall that for a Gaussian Mixture model this is given by

$$-\ln p(x) = -\sum_{i=1}^{n} \ln \sum_{j=1}^{k} \pi_j \mathcal{N}(x_i; \mu_j, \Sigma_j)$$

NOTE: As we saw in the EXAMple question each iteration must decrease the negative loglikelihood.

[247]:
```
def negativeloglikelihood(X, pi, mu, cov) -> float:
    """
    Computes the marginal negative log likelihood.

    args:
        X (n, d): data, n samples of dimensionality d
        pi (k): mixture weights of Gaussianss
        mu (k,d): means of Gaussians
        cov (k,d,d): covariances of Gaussians
    """
    n = X.shape[0]  # Number of samples
    d = X.shape[1]  # Dimensionality of each sample
    k = pi.shape[0] # number of clusters
```

```
    logll = 0
    for x_i in X:
        logll += np.log(np.sum([pi[j] * scipy.stats.multivariate_normal.
↪pdf(x_i, mean=mu[j], cov=cov[j]) for j in range(k)]))

    return -logll
```

[248]:
```
# Test for correctness
assert np.isclose(negativeloglikelihood(X, pi_test1, mu_test1, cov_test1), 379.
↪914630122269, atol=1e-2, rtol=1e-2), "Something is wrong :("
```

1.3 Exercise 3.3

Implement an initialization scheme. As you learned in the lecture, EM will only converge to a **local** extrema. Thus to find a good solution you should implement a good initialization or run the algorithm multiple times on different random initializations.

Here we provide some suggestions, but you are free to try out your own initializations! * You can use a simple clustering algorithm e.g. KMeans to find good initializations. Be free to use any implementation you want, e.g. from sklearn (already imported). * You can derive one from the data. * You can use a random initialization. Note if it is "bad" then you may run into numerical issues.

[249]:
```
def init_params():
    """
   Some initialization scheme

   returns:
        pi (k): mixture weights of Gaussians
        mu (k,d): means of Gaussians
        cov (k,d,d): covariances of Gaussians
    """
    n = X.shape[0]
    d = X.shape[1]
    kmeans = KMeans(n_clusters=k, random_state=42).fit(X)
    labels, counts = np.unique(kmeans.labels_, return_counts=True)

    pi = counts/n

    mu=kmeans.cluster_centers_

    cov = np.zeros((k,d,d))
    for j, label in enumerate(labels):
        cov[j] = np.cov(X[kmeans.labels_==label].T)

    return pi, mu, cov
```

[250]:
```
# Some tests for correctness
pi_test2, mu_test2, cov_test2 = init_params()
assert np.isclose(pi_test2.sum(), 1., atol=1e-3, rtol=1e-3), "The mixture␣
↪coefficients must sum to one."
assert all([np.all(np.linalg.eigvalsh(cov_test2[j]) > 0.) for j in range(k)]),␣
↪"The estimated covariance matrix is not positive definite"
```

1.4 Exercise 3.5

Implement the EM algorithm using the functions you just implemented above.

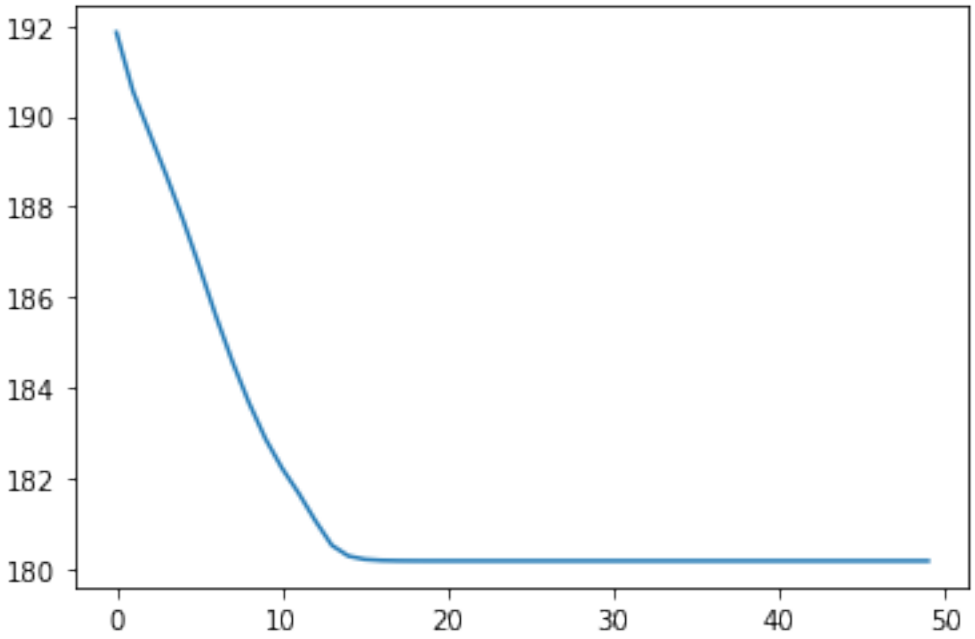
Run the algorithm on the data, the following cells should plot the "loss" history. Recall, if everything works this should be monotonically decreasing.

[252]:
```
def fit(pi, mu, cov, iters=50):
    """

    A function that runs the EM algorithm.

    args:
        pi (k): Initial mixture coefficients.
        mu (k,d): Initial cluster means.
        cov (k,d,d): Initial cluster covariance matrices.
        iters (int, optional): Number of iterations. Defaults to 50.

    returns:
        pi (k): Final mixture coefficients.
        mu (k,d): Final cluster means.
        cov (k,d,d): Final cluster covariance matrices.
        loss_hist: A list containing the negative loglikelihoods
                   at each iteration
    """
    loss_hist = np.zeros((iters,))
    for i in range(iters):
        r = E_step(X, pi, mu, cov)
        pi, mu, cov = M_step(r, X)
        loss_hist[i] = negativeloglikelihood(X, pi, mu, cov)


    return pi, mu, cov, loss_hist
```

[253]:
```
pi, mu, cov, loss = fit(*init_params())
```

[254]:
```
plt.plot(loss)
```

[254]:
```
[<matplotlib.lines.Line2D at 0x7f77b2f9a9d0>]
```



1.5 Exercise 3.7

Note that for now, we did not use the labels y. This is because our probabilistic model is *unsupervised*. We assume there is an underlying "clustered" structure but pretend we don't know it. This raises the question if we did recover the ground truth labels.

To test this, predict and plot the most likely cluster assignments for the data.

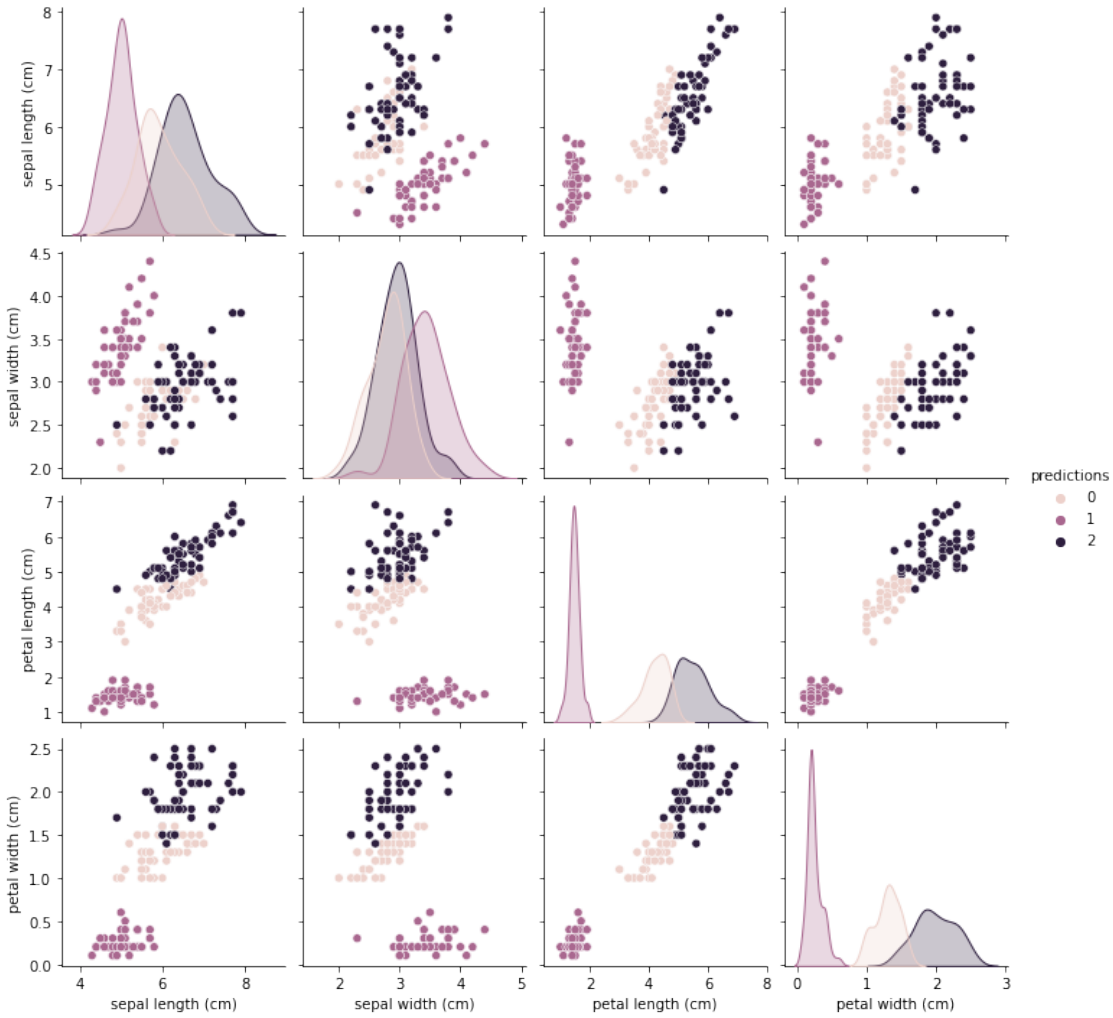
NOTE You already implemented the functions you require to do this.

Compere it to the plot on top using the ground truth cluster assignments. Does it recover the truth? If not, give some reasons why the approach may failed.

[271]:
```
probs = E_step(X, pi, mu, cov)
predictions = probs.argmax(axis=1)
# The predictions should be an array of length n containing either 0,1 or 2.
df["predictions"] = predictions
```

[272]:
```
pairplot(df.drop("target", axis=1), hue="predictions")
```

[272]:
```
<seaborn.axisgrid.PairGrid at 0x7f77d2385b50>
```

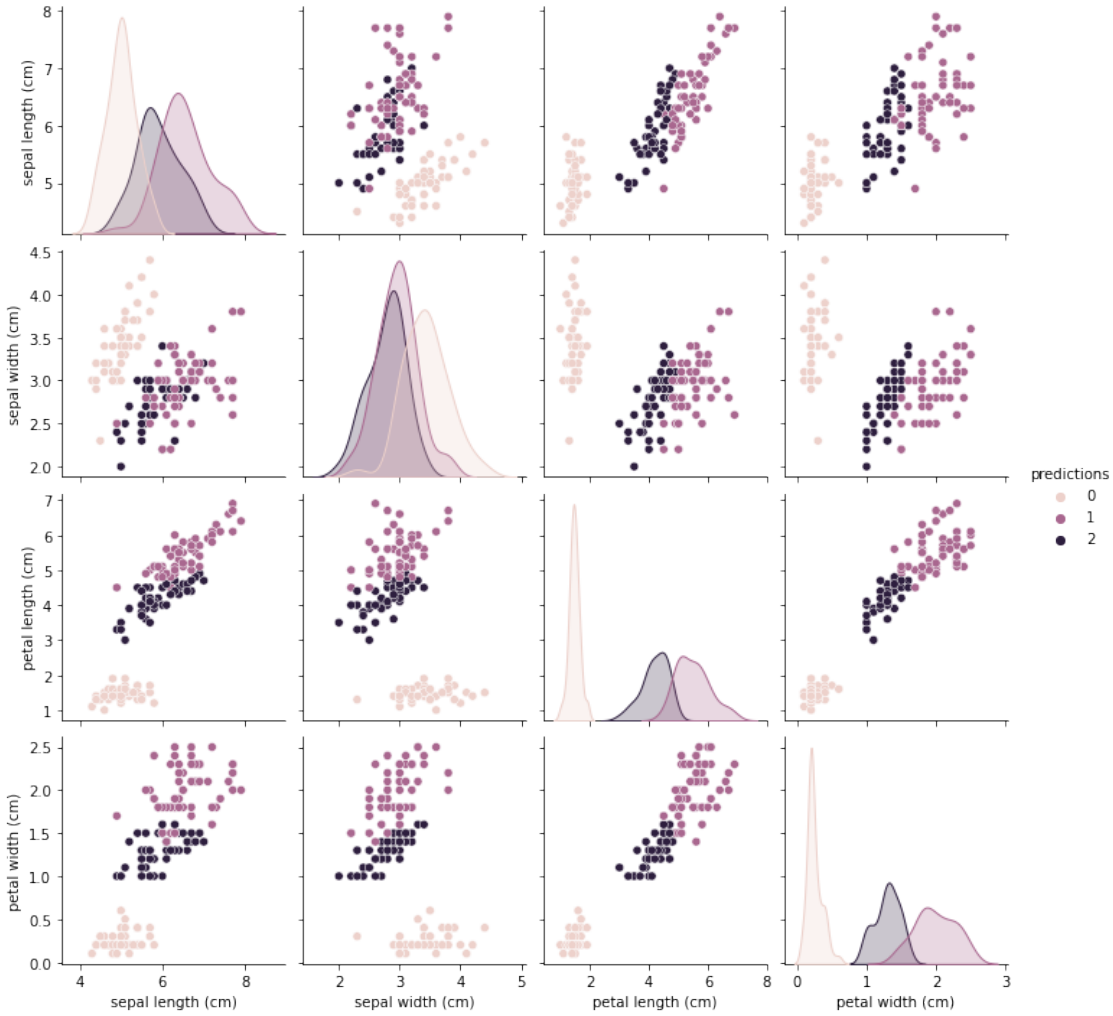


1.5.1 You can also compare your own implementation to Sklearn's

You could compare your own implementation to Sklearn's. This library uses K means as initialisation, so in case you used a different scheme, your results might look different!

[275]:
```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3, max_iter=50).fit(X)
gmm_scores = gmm.score_samples(X)

predictions = gmm.predict(X)

# The predictions should be an array of length N containing either 0,1 or 2.
df["predictions"] = predictions
pairplot(df.drop("target", axis=1), hue="predictions")
```

[275]: <seaborn.axisgrid.PairGrid at 0x7f77d1b31b20>



[ ]: