Mixture models and density estimation with PYTHON

Objective:

To understand the Gaussian mixture model; how Expectation-Maximization is used to estimate the model parameters; and how information criteria and cross-validation is used to choose the number of clusters. How the density of data can be estimated by histograms, Gaussian mixture models, kernel density estimators and k-nearest neighbor density estimation, as well as understand density based and distance based outlier/anomaly detection methods.

Material: Lecture notes *”Introduction to Machine Learning and Data Mining”* as well as the files provided from DTU Learn.

11.1 The Gaussian Mixture Model and the EM algorithm

In the exercise last week we considered k-means clustering and hierarchical clustering. Today we will consider clustering based on the Gaussian mixture model (GMM). We recall that the multivariate Gaussian distribution is given by

$$\mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_{(k)},\boldsymbol{\Sigma}_{(k)}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma}_{(k)})}}\exp\big\{-\tfrac{1}{2}(\boldsymbol{x}_i-\boldsymbol{\mu}_{(k)})^\top\boldsymbol{\Sigma}_{(k)}^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_{(k)})\big\}.$$

In the Gaussian mixture model we use a mixture of K multivariate Gaussians to model the data. We give each Gaussian a mixture coefficient w_k between zero and one such that all coefficients sum to one, $\sum_{k=1}^{K} w_k = 1$. The probability of a data vector $\boldsymbol{x}_i$ is then modeled as a weighted sum of Gaussian distributions,

$$p(\boldsymbol{x}_i|\boldsymbol{w},\{(\boldsymbol{\mu}_{(1)},\boldsymbol{\Sigma}_{(1)}),\ldots,(\boldsymbol{\mu}_{(K)},\boldsymbol{\Sigma}_{(K)})\}) = \sum_{k=1}^{K} w_k\cdot\mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_{(k)},\boldsymbol{\Sigma}_{(k)})$$

The parameters of the model, which comprises the mixture coefficients and the K means and covariance matrices, are found by the Expectation-Maximization (EM) algorithm that progress in the following way:

1. Initialize the mixture coefficient $\boldsymbol{w}$, mean and covariance of each Gaussian $\boldsymbol{\mu}_{(k)}$ and $\boldsymbol{\Sigma}_{(k)}$ by random.

2. (E-step) Calculate the expectation $P(k|\boldsymbol{x}_i,\boldsymbol{w},\{(\boldsymbol{\mu}_{(1)},\boldsymbol{\Sigma}_{(1)}),\ldots,(\boldsymbol{\mu}_{(K)},\boldsymbol{\Sigma}_{(K)})\})$ for each data point.

3. (M-step) Optimize $\boldsymbol{w}$ and $\{(\boldsymbol{\mu}_{(1)},\boldsymbol{\Sigma}_{(1)}),\ldots,(\boldsymbol{\mu}_{(K)},\boldsymbol{\Sigma}_{(K)})\}$ by maximizing the expected likelihood.

4. Keep doing 2 and 3 until the clusters do not change or a maximum number of iterations have progressed.

The EM-algorithm is closely related to the k-means algorithm but rather than operating with hard assignment of each data point, each data point is assigned a given probability of belonging to each cluster based on Bayes rule,

$$P(k|\boldsymbol{x}_i,\boldsymbol{w},\{(\boldsymbol{\mu}_{(1)},\boldsymbol{\Sigma}_{(1)}),\ldots,(\boldsymbol{\mu}_{(K)},\boldsymbol{\Sigma}_{(K)})\}) = \frac{w_k\cdot N(\boldsymbol{x}_i|\boldsymbol{\mu}_{(k)},\boldsymbol{\Sigma}_{(k)})}{\sum_{k=1}^{K} w_k\cdot N(\boldsymbol{x}_i|\boldsymbol{\mu}_{(k)},\boldsymbol{\Sigma}_{(k)})},$$

while each cluster is not only described by its center (mean) but also by its covariance. In order to get hard assignments of the data points from the results obtained by the above EM-algorithm we assign the observations to the clusters with highest probability.

An important property of the estimated Gaussian mixture model is that the GMM density integrates to one, i.e.

$$\int \sum_k w_k \cdot \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{(k)}, \boldsymbol{\Sigma}_{(k)}) d\boldsymbol{x} = \sum_k w_k \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{(k)}, \boldsymbol{\Sigma}_{(k)}) d\boldsymbol{x} = 1$$

This follows from the fact that each Gaussian integrates to one $\int \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{(k)}, \boldsymbol{\Sigma}_{(k)}) d\boldsymbol{x} = 1$ and the constraint $\sum_{k=1}^{K} w_k = 1$. An important consequence of this is that when additional clusters are introduced they have to take density mass from existing clusters.

11.1.1 Inspect and run the script `ex11_1_1.py`. The script loads the synth1 data from the file `Data/synth1` with the `loadmat` function and fits a $K = 4$ component Gaussian mixture model to the data. Notice how the script makes a scatter plot of the data and the clustering using the `clusterplot` function in the toolbox.

Script details:

- *To fit a Gaussian Mixture Model in Python you can use the class* `GaussianMixture` *from the package* `sklearn.mixture`*. You will need two methods:* `fit` *and* `predict`*. Type* `help(GaussianMixture)` *to learn how to use the class.*
- *When creating GaussianMixture object you can specify number of clusters, covariance type (full/diagonal), number of repetitions with different initial seeds.*
- *You can extract the fitted cluster means (centroids) by calling the method* `means_` *of* `GaussianMixture` *class object.*
- *Type* `clusterplot(X,clusterid,centroids,cov_matrices)` *to plot the data and clustering.*
- *Type* `help(clusterplot)` *to learn more about how to use the clustering plot tool in the toolbox. Note that you can specify* `covars` *parameter to plot ellipsoids of the gaussians corresponding to GMM.*

Sometimes the estimated models for the same value of K are not necessarily the same due to different initial placement of the centroids. In such cases you might try setting the parameter `n_init` to 10 or more. What will this do, and why should it solve the problem?

Try to change the structure of the covariance matrix to be a diagonal matrix by setting the parameter `covariance_type` to `'diagonal'`. How does this affect the generated clusters? What do you think might be benefits and drawbacks of restricting the covariance matrices?

11.1.2 **Discussion:** In k-means based on Euclidean distance observations are assigned the centroids they are the closest to. Is this also the case when clustering by the Gaussian mixture model or is it possible that points are assigned a cluster that is further away than other clusters in terms of Euclidean distance?

Can the scaling of the variables seriously affect the results we get when clustering by the GMM or is the model able to take the scaling of the data into account?

11.1.3 **Discussion:** For supervised learning we used cross-validation to evaluate performance and estimate the number of parameters in our models. In last weeks exercise, we found that this approach could not be used in the k-means algorithm. What happens if we validate the number of clusters for Gaussian mixture model based on the EM algorithm using cross-validation, i.e., split the data into training and test data, train the model on the training data and evaluate how likely the test data points are based on the learned parameters, $\boldsymbol{w}$ and $\{\boldsymbol{\mu}_{(1)}, \boldsymbol{\Sigma}_{(1)}, \ldots, \boldsymbol{\mu}_{(K)}, \boldsymbol{\Sigma}_{(K)}\}$ using the logarithm of the likelihood of the test data,

$$\log L^{\text{test}} = \sum_{i=1}^{N_{\text{test}}} \log[p(\boldsymbol{x}_i^{(\text{test})}|\boldsymbol{w}, \{(\boldsymbol{\mu}_{(1)}, \boldsymbol{\Sigma}_{(1)}), \ldots, (\boldsymbol{\mu}_{(K)}, \boldsymbol{\Sigma}_{(K)})\})]?$$

Script details:

- *Remember, that in the GMM each cluster is represented by a mean vector and a covariance matrix. In one dimension, it is the well known bell-shaped probability distribution.*
- *If we have 100 data points and use 1 cluster, what would be the optimal solution for the GMM using the EM-algorithm? How well would this solution generalize to test data?*
- *If we have 100 data points and use 100 clusters, what would be the optimal solution? How well would this solution generalize?*
- *What if we use some intermediate number of clusters?*

Apart from cross-validation the optimal number of clusters are sometimes derived by penalizing model complexity based on the Bayesian Information Criteria (BIC) or Akaike's Information Criteria (AIC). The two information criteria are defined by

$$\text{BIC} = -2\log L + p\log(N), \qquad \text{AIC} = -2\log L + 2p$$

where p is the number of free parameters in the model, i.e., the total number of estimated variables in $\boldsymbol{w}$, $\{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K\}$ whereas N is the number of observations and $\log L$ is the log likelihood of observing the data, i.e.,

$$\log L = \sum_{i=1}^{N} \log[p(\boldsymbol{x}_i|\boldsymbol{w}, \{\boldsymbol{\mu}_{(1)}, \boldsymbol{\Sigma}_{(1)}, \ldots, \boldsymbol{\mu}_{(K)}, \boldsymbol{\Sigma}_{(K)}\})].$$

The two information criteria define a trade-off between modeling the data well, i.e., minimize $-2\log L$, and penalizing complexity of the model, i.e., $p\log(N)$ and $2p$ respectively, such that the model with lowest AIC and BIC value indicates the model with best trade-off. Note, that AIC and BIC do not require splitting the data in test and training sets, but are computed directly on the whole training data.

11.1.4 **Discussion:** Which of the two information criteria BIC and AIC will in general penalize model complexity the most?

11.1.5 Inspect and run the script `ex11_1_5.py`. We will use BIC, AIC, and 10-fold crossvalidation to assess the best number of clusters for the synth1 data set. Use the script to compute the three measures for $K = 1, \ldots, 10$, and use 10 replicates to avoid bad solutions due to poor initial conditions.

Script details:

- *As before, use* `sklearn.mixture.GaussianMixture` *class to fit the models.*
- *As usual, use the module* `sklearn.model_selection` *to set up the crossvalidation folds.*
- `GaussianMixture` *class has* `bic` *and* `auc` *methods to compute BIC and AUC scores automatically. Type* `help(GaussianMixture)` *to read more.*
- *To evaluate the model fit on the test set in terms of negative log likelihood, you can use the method* `score` *(after fitting the model). For instance:*
  `NLOGL = -gmm.score(X_test).sum().`

What are the benefits and drawbacks of AIC and BIC versus cross-validation?

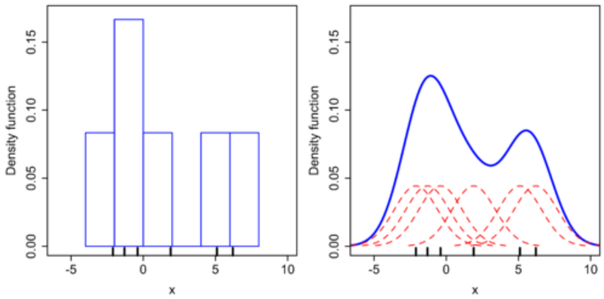

Figure 1: Estimation of the data density by histogram (left) and kernel density estimation (right). The Kernel specifies a shape around each data point (here a univariate normal distribution) and the density is estimated by summing over these Gaussians placed at each observation

11.1.6 **(Optional):** Use a Gaussian mixture model to cluster the Old Faithful data set, `Data/faithful`, using AIC, BIC, or crossvalidation to select the number of clusters.

11.1.7 **(Optional):** Use a Gaussian mixture model to cluster the Iris data set, `Data/iris.xls`, using AIC, BIC, or crossvalidation to select the number of clusters.

11.2 Density estimation

We will apart from the Gaussian mixture model (we just used) now consider the following two approaches to density estimation: kernel density estimation, and k-nearest neighbors density estimation.

Kernel density estimation is a non-parametric method of estimating the probability density function of a random variable. Inference about the population are made, based on a finite data sample. The estimated kernel density for a variable x is given by

$$f(x) = \frac{1}{N}\sum_{i=1}^{N} K(x - x_i) \tag{1}$$

where K is the kernel function that must integrate to one, N is the sample size, and $f(x)$ is the estimated density (see also figure 1.)

We will presently consider density estimation based on the kernel formed by the normal distribution [1] , i.e.,

$$K(x - x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(- \frac{(x - x_i)^2}{2\sigma^2}\Big).$$

We will initially consider an artificially generated one-dimensional data set formed by a mixture of three Gaussians defined as follows

$$p(x) = \frac{1}{3}\mathcal{N}(x|1,1) + \frac{1}{3}\mathcal{N}(x|3,0.5) + \frac{1}{3}\mathcal{N}(x|6,2)\ . \tag{2}$$

11.2.1 Inspect and run the script `ex11_2_1.py`. Notice how the script generates 1000 data objects according to the model in equation (2), and plots a histogram of the data with 50 bins in the range -10 to 10.

Script details:

[1] Notice, that for this choice of kernel function the kernel density estimation can be considered a Gaussian mixture model where the number of clusters is the same as the number of observations, i.e., there is a Gaussian around each observation, while the mixing coefficient is fixed to be the same for all classes, $w_n = 1/N$.

- *You can generate data from the Gaussian mixture model in the following way: First, you choose one of the components by random according to the mixture coefficients. Then, you generate the data according to the Gaussian distribution for the chosen component.*
- *Type* `help(np.random.multinomial)` *to learn how to generate a random variable from a discrete distribution.*
- *You can use* `help(np.random.normal)` *function to draw samples from normal distribution (particular gaussian from the mixture).*
- *The function* `numpy.hist` *can be used to plot a histogram. Use* `help(numpy.hist)` *to get help.*
- *To define the bins at which the histogram should be evaluated, you can define a vector of values and pass it to* `hist`*. To get 50 bins evenly spaced between* -10 *and* 10*, you can use the command* `x=np.linspace(-10,10,50)`.

Discussion:

- ◇ Can you identify each component of the mixture in the plot and its associated parameters (mean, variance, mixture coefficient?)
- ◇ Try changing the parameters of the mixture to see the effect on the resulting density.
- ◇ Try varying the number of bins in the histogram.

11.2.2 Inspect and run the script `ex11_2_2.py`. Explain how the script estimates the density using a kernel density estimator with a Gaussian kernel and a kernel width of 1, and plot the density on the range -10 to 10.

Script details:

- *Use* `gaussian_kde` *from the module* `scipy.stats.kde` *to fit a kernel density estimator.*
- *To define at which x-values the KDE should be evaluated, you can define a vector of values and pass it to the* `evaluate` *method of the fitted KDE estimator. For example, to get 100 points evenly spaced between* -10 *and* 10*, you can use the command* `x=np.linspace(-10,10,100)`

Discussion:

- ◇ Compare the result to the histogram.
- ◇ Try varying the kernel width in the KDE.
- ◇ How could you select an optimal kernel width?

Consider the following measure of density of the i^{th} observation $\boldsymbol{x}_i$ given based on its k-nearest neighbors (KNN)

$$\text{density}_{\boldsymbol{X}_{\setminus i}}(\boldsymbol{x}_i, K) = \frac{1}{\frac{1}{K}\sum_{\boldsymbol{x}' \in N_{\boldsymbol{X}_{\setminus i}}(\boldsymbol{x}_i, K)} d(\boldsymbol{x}_i, \boldsymbol{x}')}, \tag{3}$$

where $N_{\boldsymbol{X}}(\boldsymbol{x}, K)$ is the K observations in $\boldsymbol{X}$ which are nearest to $\boldsymbol{x}$, and $\boldsymbol{X}_{\setminus i}$ is simply $\boldsymbol{X}$ with observation i removed: $\boldsymbol{X}_{\setminus i}^T = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots \boldsymbol{x}_{i-2} & \boldsymbol{x}_{i-1} & \boldsymbol{x}_{i+1} & \boldsymbol{x}_{i+2} & \cdots & \boldsymbol{x}_N \end{bmatrix}$. This estimates the density as the inverse of the average distance to the k nearest neighbors.

If the dataset contains regions of varying densities it can be useful to define a notion of density that is relative to the neighborhood of the object. The following average relative density is one such approach

$$\text{ard}_{\boldsymbol{X}}(\boldsymbol{x}_i, K) = \frac{\text{density}_{\boldsymbol{X}_{\setminus i}}(\boldsymbol{x}_i, K)}{\frac{1}{K}\sum_{\boldsymbol{x}_j \in N_{\boldsymbol{X}_{\setminus i}}(\boldsymbol{x}_i, K)} \text{density}_{\boldsymbol{X}_{\setminus j}}(\boldsymbol{x}_j, K)}. \tag{4}$$

11.2.3 Inspect and run `ex11_2_3.py` and see how the script can be used to estimate the density using equation (3) as well as the average relative density using equation (4) with the 200-nearest neighbors based on the Euclidean distance measure, and plot the density on the range -10 to 10.

Script details:

- *To define at which x-values the nearest neighbor density estimator should be evaluated, you can define a vector of values, e.g., to get 100 points evenly spaced between -10 and 10, you can use the command* `x=np.linspace(-10,10,100)`*.*
- *To find the nearest neighbors and the distances needed to compute equation (3), you can fit the* `NearestNeighbors` *model from the module* `sklearn.neighbors` *and then use the method* `kneighbors()` *of to extract nearest neighbors distances.*

Discussion:

- ◇ Try different values for the number of neighbors.
- ◇ Compare your results to the three previous exercises.

11.3 Outlier detection

The following two definitions of an outlier are common:

> **Hawkin's Definition of an Outlier.** An outlier is an observation that differs so much from other observations as to arouse suspicion that it was generated by a different mechanism.
>
> **Probabilistic Definition of an Outlier.** An outlier is an object that has a low probability with respect to a probability distribution model of the data.

Thus, in order to detect outliers we need a mechanism that can establish whether an observation differs so much from other observations as to be deemed an outlier. An important tool to evaluate this is to create a model of the data density and then evaluate how likely the observations are given the density.

11.3.1 Inspect and run the script `ex11_3_1.py`. Prior to running the script you need to execute the command `X[-1,0]=-10` to add an outliner at -10 to the data. The script uses a kernel density estimator (as in exercise 11.2.2) with a Gaussian kernel to estimate the density at the x-values in the data set. Verify that the outlier you have introduced has the lowest density. Notice how the bar plot corresponds to the density of the 20 lowest-density points.

Script details:

- *You originally had $N = 1000$ data objects. What is the index of the introduced outlier?*
- *To get the KDE for each data object in the data set, you have to fit KDE model first (class* `gaussian_kde()` *from the module* `scipy.stats.kde`*), and then use* `evaluate()` *method.*
- *To find the indices and values of the 20 lowest-density points, you can use the methods* `argsort()` *and* `sort()` *respectively, on the output from* `evaluate()` *method.*
- *To make a bar plot, use the function* `bar`*.*

Discussion:

- ◇ Can the outlier be detected from this plot?
- ◇ Try different values of the kernel width.
- ◇ What happens when the kernel width is too large / too small?

11.3.2 The function `gausKernelDensity()` in the toolbox implements density estimation by the gaussian kernel density estimator using a very efficient implementation of leave-one-out cross-validation. I.e. the density of each observation is estimated from all other observations not including the observation itself in the estimate. Inspect and run the script `ex11_3_2.py` where it is used. Use the script to estimate the optimal kernel width by evaluating the estimated densities for a range of different kernel widths. Plot the leave-one-out density of the 20 lowest-density points in a bar plot. Script details:

- *To get the leave-one-out KDE for each data object in the data set for a kernel width of 5, you can use the command* `f,log_f=gausKernelDensity(X,5)`*.*
- *The log-density for all observations is given by* `logP=log_f.sum()`*. The optimal kernel width is the width with highest* `logP`*.*
- *To find the 20 lowest-density points, you can again use the function* `sort` *to sort the output from* `ksdensity`*.*
- *To make a bar plot, use the function* `bar`*.*

11.3.3 **(Optional):** Repeat exercise 11.3.1 using the KNN-density estimator as well as the KNN-average relative density.

11.4 Hand written digits

We will in this part of the exercise investigate if some of the hand written digits of each class can be considered outliers. To do this, inspect and run the script `ex11_4_1.py`.

11.4.1 Load the hand written digits data set from the file `Data/digits.mat` with `loadmat()` function. Restrict your analysis to images of the digit "2" by the command `X=X[y==2,:]`.

1) Use the function `gausKernelDensity()` from toolbox that implements density estimation by the gaussian kernel density estimator using the very efficient implementation of leave-one-out density estimation (as in exercise 11.3.2). Estimate the optimal kernel width evaluating a range of different widths. Make a bar plot of the leave-one-out densities of the 20 lowest-density data objects in the data set based on the optimal kernel width.

2) Use the KNN density estimation method (as in exercise 11.2.4) based on the Euclidean distance measure and using $K = 5$ neighbors. Make a bar plot of the densities of the 20 lowest-density data objects in the data set.

3) Use the KNN average relative density estimation method (as in exercise 11.2.4) also based on the Euclidean distance measure and using $K = 5$ neighbors. Make a bar plot of the densities of the 20 lowest-density data objects in the data set.

Plot the images of the 20 data objects with the lowest density / highest outlier score.

Script details:

- *The images are 16 by 16 pixels stored as 256 dimensional vectors. To plot an image, you must reshape the vector to a 16 by 16 matrix and plot it using the* `imshow()` *function. For example, to plot the* `i`*'th image in* `X`*, use the command:* `imshow(np.reshape(X[i-1,:], (16,16)).T, cmap=cm.binary)`*.*

Discussion:

- ◇ Does the three bar plots reveal any possible outliers in the data set?
- ◇ Can you identify any data objects that really are outliers?
- ◇ Do the three methods agree?
- ◇ Try plotting 20 random images of hand written 2's (for example the first 20 in the data set) for comparison. In what sense are the low-density 2's different from the randomly chosen ones?
- ◇ Try other digits than "2". Does the method in general identify digits that are outliers?
- ◇ Try the method on the whole data set with all digits (warning: this might take a long time to compute.)

References