Machine Learning Programming
Assignment 4: Gaussian Mixture Models

Professors: Baptiste Busch, Aude Billard
Assistants: Laila El Hamamsy, Matthieu Dujany
and Victor Faraut
Contacts:
baptiste.busch@epfl.ch, aude.billard@epfl.ch
laila.elhamamsy@epfl.ch, matthieu.dujany@epfl.ch, victor.faraut@epfl.ch

Winter Semester 2018

Introduction

In this practical, you will implement classification, regression, and sampling algorithms based on Gaussian Mixture Models (GMM) for several datasets.

Submission Instructions

Deadline: December 18, 2017 @ 6pm. Assignments must be turned in by the deadline. 1pt will be removed for each day late. A day late starts one hour after the deadline.
Procedure: From the course Moodle webpage, the student should download and extract the .zip file named `TP5-GMMApps-Assignment.zip` which contains the following files:

Part 1 - Classification	Part 2 - Regression	Part 3 - Sampling
plot_boundaries.m	my_gmr.m	TO_BE_PROVIDED.m
TO_BE_PROVIDED.m	my_regression_metrics.m	TO_BE_PROVIDED.m
TO_BE_PROVIDED.m	cross_validation_gmr.m	-
-	test_gmr.m	-
test_gmm_classify.m	test_gmr_metrics.m	test_sampling.m

As well as `TP5-GMMApps-Datasets.zip` containing the datasets required to test your functions.

Assignment Instructions

The assignment consists on implementing the blue colored MATLAB functions from scratch. The functions name as `TO_BE_PROVIDED.m` must be defined by you, i.e. you should decide the name, the input and output parameters. The functions can be tested with the `test_*.m`. These testing scripts depend on `ML_toolbox`, which must be downloaded from: `https://github.com/epfl-lasa/ML_toolbox`. Before proceeding make sure that all the sub-directories of the ML_toolbox have been added to your MATLAB search path. This can be done as follows in the MATLAB command window:

```
>> addpath(genpath('path_to_ML_toolbox'))
```

Once you have tested your functions, you can submit them as a .zip file with the name: `TP5-GMMApps-SCIPER.zip` on the submission link in the Moodle webpage. Your submission archive should contain ONLY the following:

- `TP5-GMMApps-Assignment-SCIPER`
  - `Part1`
  - `Part2`
  - `Part3`

DO NOT upload `ML_toolbox`, `check_utils_encr`, `utils` or the `Datasets` directory.

NOTE: This assignment will be less guided than the previous ones. There will not be a testing functions for every task you have to implement and the barebone function structure might not be provided. To help you in this assignment, you have access to all the previous functions developped during the whole semester. Those functions are provided in the `utils` folder. It is not necessary to use all of them.

1 Part 1: Classification with GMM+Bayes

In the previous assignment, we have implemented the GMM model to fit a dataset in a complete unsupervised manner. To recall, given a dataset $\mathbf{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$ where $\mathbf{x}^i \in \mathbb{R}^N$, the probability density function (pdf) of a K-component GMM is of the form,

$$p(\mathbf{x}|\Theta) = \sum_{k=1}^{K} \alpha_k p(\mathbf{x}|\mu^k, \mathbf{\Sigma}^k) \tag{1}$$

where $p(\mathbf{x}|\mu^k, \mathbf{\Sigma}^k)$ is the multivariate Gaussian pdf with mean μ^k and covariance $\mathbf{\Sigma}^k$

$$p(\mathbf{x}|\mu^k, \Sigma^k) = \frac{1}{(2\pi)^{N/2}|\Sigma^k|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu^k)^T(\Sigma^k)^{-1}(\mathbf{x}-\mu^k)\right\}. \tag{2}$$

$\Theta = \{\theta^1, \ldots, \theta^k\}$ is the complete set of parameters $\theta_k = \{\alpha_k, \mu^k, \Sigma^k\}$, where α_k are the priors (or mixing weights) of each Gaussian component, satisfying the constraint $\sum_{k=1}^{K} \alpha_k = 1$. These parameters are typically learned by maximizing the likelihood of (1) through the iterative Expectation-Maximization algorithm, which was implemented in the previous assignment.

In this part, we want to decide to which class each datapoint belongs to with models of each class learnt *a priori*, in other words classification. To perform classification, we need to create a **training** dataset (i.e. a labeled dataset to build our model) and a **testing** dataset to evaluate the performance of the algorithm. To perform classification with GMMs, the first step is to learn a GMM for each class from the **training** set. Given a model for each class, we can use the Gaussian Likelihood Discriminant Rule to classify the **testing** datapoints.

1.1 Learning a GMM for each class

You will have access to a library of functions in `utils/datasets` folder that generate datasets of 2 to 4 classes as shown in Fig. 1. Those functions return sets of (x, y) points where the class labels y are stored in the last column of the returned variable. Each functions can take optional parameters but the easiest usage is to use the default values by passing no arguments. An example of the functions calling is written in the file `datasetsdemo.m`.

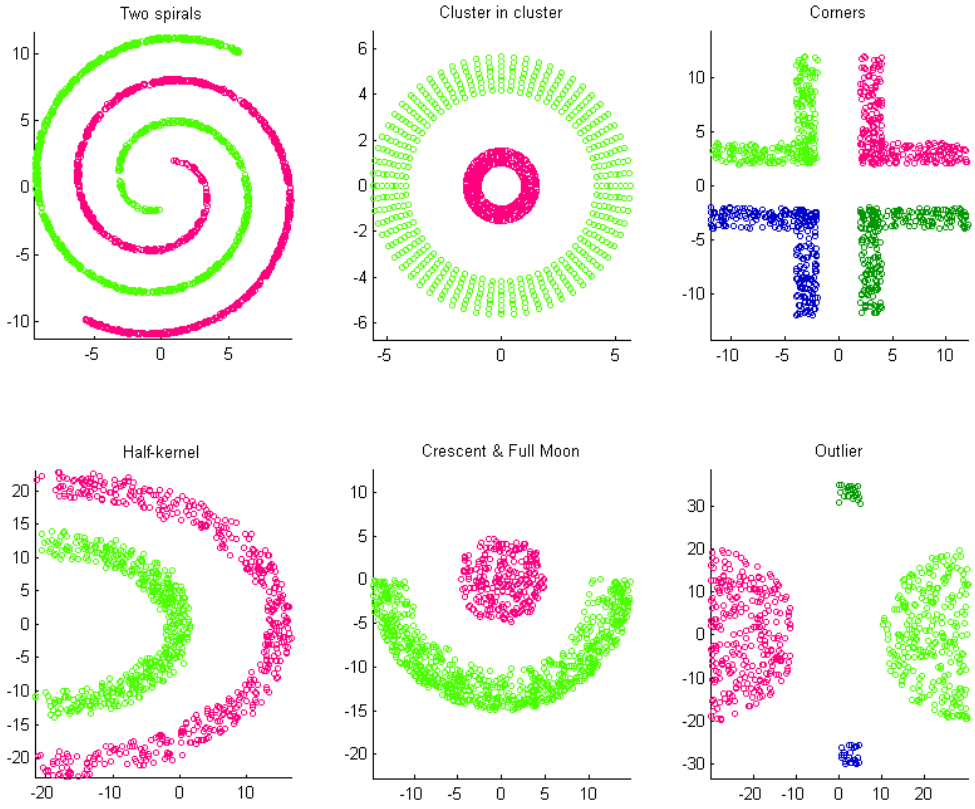

Figure 1: Functions to generate datasets for classification

TASK 1: Dataset loading and preprocessing (2pts)

Your first task will be to choose a dataset and load it properly, separating the x and y values for each datapoints and store them into X and Y arrays. Please bear in mind that the Y labels should range from 1 to the number of class. Once you have properly loaded a dataset, you will have to implement a function that learns a GMM for each of the class label. Remember that for a classification problem it is not wise to train a model on the whole dataset. You will have to make the appropriate transformations in the file `test_gmm_classify.m`.

TASK 2: Implement a function to train GMMs for each class labels (2pts)

Using the previously developed functions for learning the parameters of a Gaussian Mixture model via EM algorithm, learn a GMM for each of the classes on the **training** dataset. This function should return the learned parameters (α, μ, σ) for all the GMMs, stored in a structure array. You have to choose the paramters of the GMM to better fit your dataset. For the initialization parameters of the K-Means algorithm you can keep the default ones.

Test Implementation

You can test that the implementation of your function is correct by running the visualization code between the two `ADD CODE` blocks. This will produce an image of the contour of the GMM

for each of the class labels. An example is provided on Fig. 2, and 3 for the `half kernel` dataset.

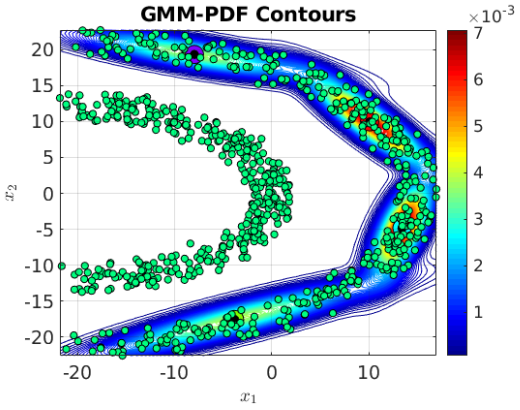

Figure 2: GMM fitting on the first class of the dataset

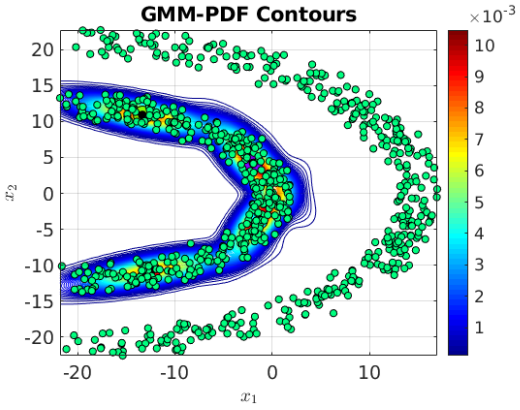

Figure 3: GMM fitting on the second class of the dataset

1.2 Gaussian Maximum Likelihood Discriminant Rule

A maximum likelihood classifier chooses the class label that is the most likely. For a binary classification problem, a new datapoint $\mathbf{x}'$ belongs to class 1 if the corresponding likelihood is superior to the likelihood of class 2,

$$p(y = 1|\mathbf{x}') > p(y = 2|\mathbf{x}'). \tag{3}$$

By Bayes, we have

$$p(y = i|\mathbf{x}') = \frac{p(\mathbf{x}'|y = i)p(y = i)}{p(\mathbf{x}')}, \text{with class } i = 1, 2. \tag{4}$$

Replacing this term in the Maximum Likelihood (ML) Discriminant Rule for $\mathbf{x}'$ in class 1 (equation 3), we obtain

$$p(\mathbf{x}'|y = 1)p(y = 1) > p(\mathbf{x}'|y = 2)p(y = 2). \tag{5}$$

In general, one can assume equal class distribution, i.e. $p(y = 1) = p(y = 2)$. By replacing in the previous equation, the discriminant rule becomes

$$p(\mathbf{x}'|y = 1) > p(\mathbf{x}'|y = 2). \tag{6}$$

For a GMM composed of K^i multivariate Gaussian functions, the conditional densities to belong to class i is similar to Equation 1,

$$p(\mathbf{x}'|y = i) = \sum_{k=1}^{K^i} \alpha_k p(\mathbf{x}'|\mu^k, \mathbf{\Sigma}^k), \tag{7}$$

with $p(\mathbf{x}'|\mu^k, \mathbf{\Sigma}^k)$ defined according to Equation 2.

In the case of a multi-class problem with $i = 1...I$ classes, the ML Discriminant Rule is the minimum of the log-likelihood (i.e., equivalent to maximizing the likelihood). **Assuming equal class distribution**, the ML Discriminant Rule is:

$$c_i(\mathbf{x}') = \underset{i}{\operatorname{argmin}} \left\{ -\log(p(\mathbf{x}'|y=i)) \right\}. \tag{8}$$

In the case that one **does not make the assumption of equal class distribution**, according to Equation 6, the ML Discriminant Rule is:

$$c_i(\mathbf{x}') = \underset{i}{\operatorname{argmin}} \left\{ -\log(p(\mathbf{x}'|y=i)p(y=i)) \right\}. \tag{9}$$

TASK 3: Implement a function to predict labels of given datapoints (3pts)

Implement ML Discriminant Rule to estimate labels `y_est` for each datapoints in `X_test` from Equation 8; i.e. **assuming equal class distribution.** For this assignment you don't have to consider the case of unbalanced dataset but bear in minds that it requires extra care in case you happens to encounter it in the future. Please note that you also need to call this function in `plot_boundaries.m`.

Implementation Hint: Useful functions `my_gaussPDF()`.

Test Implementation

To test your implementation of this function you can run the last code block which will evaluate the accuracy of the predictions and plot the decision boundaries. An example of the results is shown in Fig. **??**fig:classification)

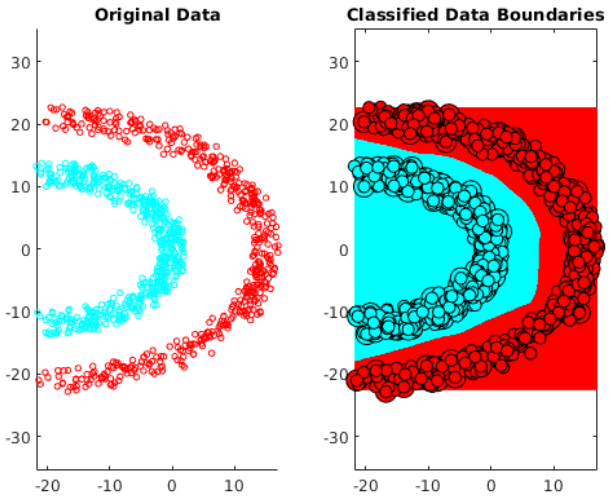

Figure 4: Decision boundaries after prediction on the test set

2 Part 2: Regression with GMM (GMR)

Regression problems are a category of *supervised learning* algorithms that seeks to approximate the true mapping function $\mathbf{y} = f(\mathbf{x})$ from inputs $\mathbf{x}$ to outputs $\mathbf{y}$ from a set of training data. As opposed to classification, in **regression**, the output variables are continuous $y \in \mathcal{R}$. Such regression problems are found in a wide spectrum of research, financial, commercial and areas.

Any regression problem is posed as follows, given a training data set of inputs $\mathbf{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$ of M samples, where $\mathbf{x}^i \in \mathbb{R}^N$ are N-dimensional inputs and a set of corresponding outputs $\mathbf{y} = \{\mathbf{y}^1, \ldots, \mathbf{y}^M\}$, where $\mathbf{y}^i \in \mathbb{R}^P$ are continuous P-dimensional outputs, we seek to approximate a continuous mapping function $\hat{f} : \mathbb{R}^N \rightarrow \mathbb{R}^P$, $\hat{\mathbf{y}} = \hat{f}(\mathbf{x})$, that best recovers the true *regressive function* $\mathbf{y} = f(\mathbf{x})$ (Figure 5, 6 and 7).

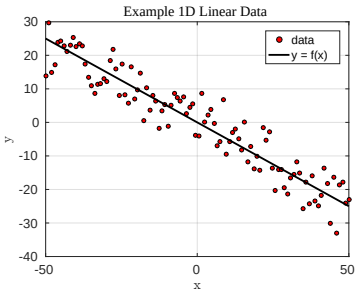

Figure 5: Regression example on 1D linear data. Black line, i.e. $y = f(x)$, is the true regressive function.

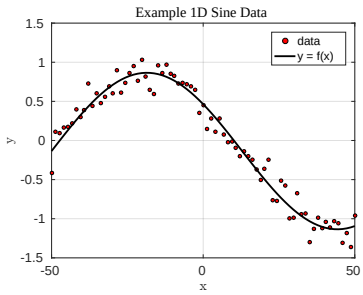

Figure 6: Regression example on 1D non-linear data. Black line, i.e. $y = f(x)$, is the true regressive function.

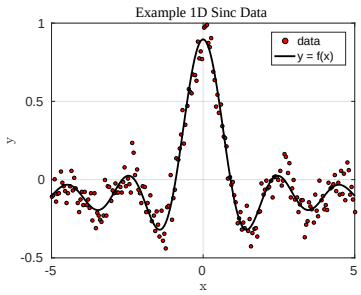

Figure 7: Regression example on 1D non-linear data. Black line, i.e. $y = f(x)$, is the true regressive function.

Many linear/non-linear methods exist to obtain this regressive function $f(\mathbf{x})$. In this assignment we will implement a version using our previously implemented GMMs, Gaussian Mixture Regression. For further reading on these regression methods, refer to the Regression Slides or the **Lecture Notes** from the Applied Machine Learning.

2.1 Gaussian Mixture Regression

Gaussian Mixture Regression is a parametric model-based method, capable of learning a regressive function of non-linear datasets. It stems from estimating a joint density of the inputs and outputs through a K-component Gaussian mixture model as follows:

$$p(\mathbf{x}, \mathbf{y}|\Theta) = \sum_{k=1}^{K} \alpha_k p(\mathbf{x}, \mathbf{y}|\mu^k, \mathbf{\Sigma}^k), \tag{10}$$

where α_k are the priors of each Gaussian component parametrized by $\{\mu^k.\mathbf{\Sigma}^k\}$ where

$$\sum_{k=1}^{K} \alpha_k = 1, \qquad \mu^k = \begin{bmatrix} \mu^k_{\mathbf{x}} \\ \mu^k_{\mathbf{y}} \end{bmatrix}, \qquad \mathbf{\Sigma}^k = \begin{bmatrix} \mathbf{\Sigma}^k_{\mathbf{xx}} & \mathbf{\Sigma}^k_{\mathbf{xy}} \\ \mathbf{\Sigma}^k_{\mathbf{yx}} & \mathbf{\Sigma}^k_{\mathbf{yy}} \end{bmatrix}. \tag{11}$$

Given this joint density, one can compute the conditional density $p(\mathbf{y}|\mathbf{x})$ which will provide our regressive function $y = f(x)$,

$$p(\mathbf{y}|\mathbf{x}) = \sum_{k=1}^{K} \beta_k p(\mathbf{y}|\mathbf{x}; \mu^k, \mathbf{\Sigma}^k) \tag{12}$$

where

$$\beta_k = \frac{\alpha_k p(\mathbf{x}|{\mu_{\mathbf{x}}}^k, {\mathbf{\Sigma}_{\mathbf{xx}}}^k)}{\sum_{k=1}^{K} \alpha_k p(\mathbf{x}|{\mu_{\mathbf{x}}}^k, {\mathbf{\Sigma}_{\mathbf{xx}}}^k)}. \tag{13}$$

The β_k parameters are known as the mixing weights (i.e. relative importance) of each K-th regressive model $p(\mathbf{y}|\mathbf{x};\mu^k,\mathbf{\Sigma}^k)$. The *regressive function* is then obtained by computing the expectation over the conditional density $\mathbb{E}\{p(\mathbf{y}|\mathbf{x})\}$ as follows:

$$y = f(\mathbf{x}) = \mathbb{E}\{p(\mathbf{y}|\mathbf{x})\} = \sum_{k=1}^{K} \beta_k(\mathbf{x})\tilde{\mu}^k(\mathbf{x}) \tag{14}$$

where

$$\tilde{\mu}^k(\mathbf{x}) = \mu_{\mathbf{y}}^k + \mathbf{\Sigma}_{\mathbf{yx}}^k(\mathbf{\Sigma}_{\mathbf{xx}}^k)^{-1}(\mathbf{x} - \mu_{\mathbf{x}}^k) \tag{15}$$

are the local regressive functions, which represent a linear function whose slope is determined by $\mathbf{\Sigma}_{\mathbf{x}}^k$ (the variance of $\mathbf{x}$) and $\mathbf{\Sigma}_{\mathbf{xy}}^k$ (the covariance of $\mathbf{y}$ and $\mathbf{x}$). Our regressive function $y = f(\mathbf{x})$ is thus a linear combination of K regressive models. The advantage of GMR over other non-linear regressive methods is that we can also compute the uncertainty of the prediction (i.e. the regressed output $\hat{y}$) by computing the variance of the conditional function $Var\{p(\mathbf{y}|\mathbf{x})\}$ as follows:

$$Var\{p(\mathbf{y}|\mathbf{x})\} = \sum_{i=1}^{K} \beta_k(\mathbf{x}) \left(\tilde{\mu}^k(\mathbf{x})^2 + (\tilde{\mathbf{\Sigma}}^k)\right) - \left(\sum_{i=1}^{K} \beta_k(\mathbf{x})\tilde{\mu}^k(\mathbf{x})\right)^2 \tag{16}$$

where $\tilde{\mathbf{\Sigma}}^k$ represents the variance of the conditional density for each regressive function and is computed as follows:

$$\tilde{\mathbf{\Sigma}}^k = \mathbf{\Sigma}_{\mathbf{yy}}^k - \mathbf{\Sigma}_{\mathbf{yx}}^k(\mathbf{\Sigma}_{\mathbf{xx}}^k)^{-1}\mathbf{\Sigma}_{\mathbf{xy}}^k \tag{17}$$

NOTE: that $Var\{p(\mathbf{y}|\mathbf{x})\}$ represents the uncertainty of the prediction NOT of the model. To represent the uncertainty of the model one should compute the likelihood of the joint density; i.e. $\mathcal{L}(\Theta|\mathbf{X},\mathbf{y}) = \prod_{i=1}^{M}\sum_{k=1}^{K} \alpha_k p(\mathbf{x},\mathbf{y}|\mu^k,\mathbf{\Sigma}^k)$. Another advantage of GMR over most regression methods is that it can be used with multi-dimensional inputs $\mathbf{x} \in \mathbb{R}^N$ as well as multi-dimensional outputs $\mathbf{y} \in \mathbb{R}^P$, with the same formulation presented here. For a thorough description of the derivation of this method we recommend reading Hsi Guang Sung's PhD Thesis [1] and Cohn et al's seminal work on learning with statistical models [2].

Step 1. Estimate a Joint Density $p(\mathbf{x},\mathbf{y}|\Theta)$ of your Dataset {X,y} with a GMM

The first step in implementing GMR is to learn the joint density of the inputs $\mathbf{X} \in \mathbb{R}^{M\times N}$ and outputs $\mathbf{y} \in \mathbb{R}^{M\times P}$, where N and P correspond to the dimensionality of the input and output spaces, respectively. By loading the three different 1-D datasets provided in sub-blocks 1a)/1b)/1c) of the script `test_gmr.m` and then running the **third** code block and selecting the values of K and `cov_type` you will fit a GMM to your regression dataset and should see Figures 8,11 for the linear dataset loaded with sub-block 1a); Figures 9,12 for the non-linear dataset loaded with sub-block 1b) and Figures 10,13 for the non-linear sinc dataset loaded with sub-block 1c).

Step 2. Compute the Expectation and Variance of the Conditional $p(\mathbf{y}|\mathbf{x})$

Once the GMM parameters are estimated for the joint dataset {**X**,**y**} we can estimate the regressive function by implementing Equations 13, 14 and 16.

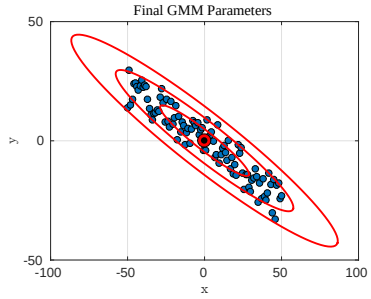

Figure 8: Selected GMM parameters ($K = 1$) for data from Fig. 5.

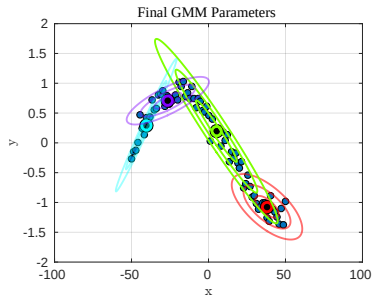

Figure 9: Final GMM parameters ($K = 4$) for data from Fig. 6.

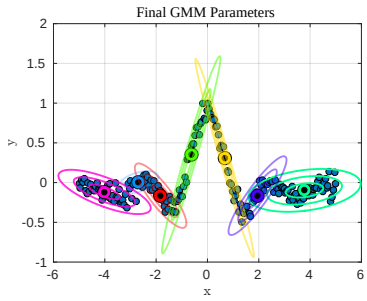

Figure 10: Final GMM parameters ($K = 7$) for data from Fig. 7.

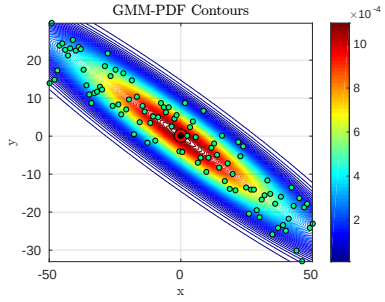

Figure 11: PDF of the fitted GMM for data from Fig. 5.

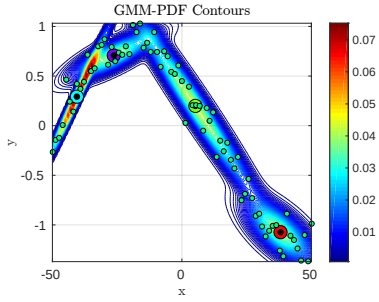

Figure 12: PDF of the fitted GMM for data from Fig. 6.

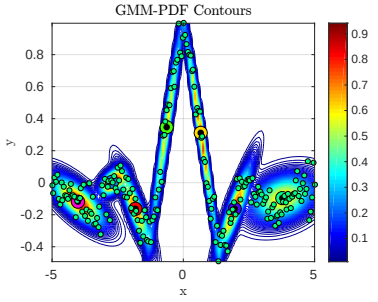

Figure 13: PDF of the fitted GMM for data from Fig. 7.

TASK 4: Implement my_gmr.m function (4pts)

The inputs for the `my_gmr.m` function are the GMM parameters $\Theta = \{\alpha, \mu, \mathbf{\Sigma}\}$ estimated in the previous step, as well as $\mathbf{X}$ the input data to regress on and the variables `in` and `out` which correspond to the indices of the input (N) and output (P) dimensions of the D-dimensional $D = (N + P)$ joint dataset used to train the GMM parameters. The outputs are `y_est` and `var_est` which correspond to Eq.14 and Eq. 16, respectively.

```
function [y_est, var_est] = my_gmr(Priors, Mu, Sigma, X, in, out)
% Inputs -----------------------------------------------------------------
%   o Priors:  1 x K array representing the prior probabilities of the K GMM
%              components.
%   o Mu:      D x K array representing the centers of the K GMM components.
%   o Sigma:   D x D x K array representing the covariance matrices of the
%              K GMM components.
%   o x:       N x M array representing M datapoints of N dimensions.
%   o in:      1 x N array representing the dimensions of the GMM parameters
%                to consider as inputs.
%   o out:     1 x P array representing the dimensions of the GMM parameters
%                to consider as outputs.
% Outputs ----------------------------------------------------------------
%   o y_est:     P x N array representing the retrieved N datapoints of
%                P dimensions, i.e. expected means.
%   o var_est:   P x P x M array representing the M expected covariance
%                matrices retrieved.
```

Test Implementation

To evaluate this function you should run one of sub-block 1 of in the testing script `test_gmr.m`. This will load a 1D Regression Datasets as in Figure 5, 6, and 7. There are three possible dataset, available by changing the value in `dataset_type` to either `1d-sine`, `1d-linear`, or `1d-sinc` If you obtain plots as the ones in Figure 14, 15 and 7 (using `cov_type='full'`) your implementation is correct for 1D datasets.

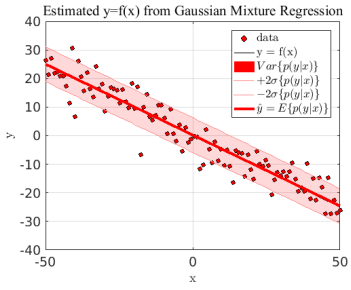

Figure 14: GMR (K=1) Result of data from Fig. 5.

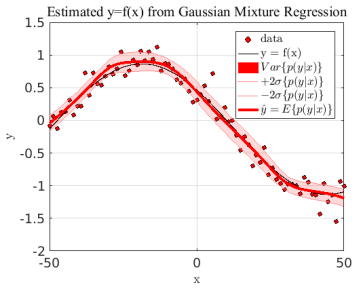

Figure 15: GMR (K=4) Result of of data from Fig. 6.

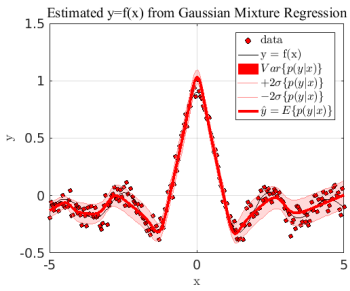

Figure 16: GMR (K=7) Result of data from Fig. 7.

You can also try using different Covariance matrix types, for example by setting `cov_type = 'diag'`you should get plots similar to Figures 17-20.

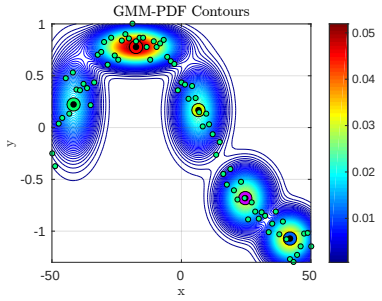

Figure 17: GMR (K=5) Result of 1D non-linear Dataset.

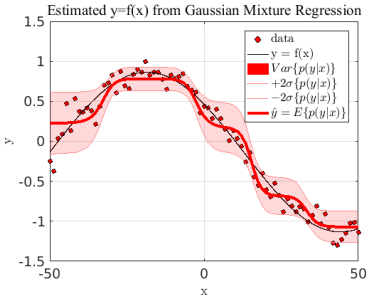

Figure 18: GMR (K=5) Result of 1D non-linear Dataset.

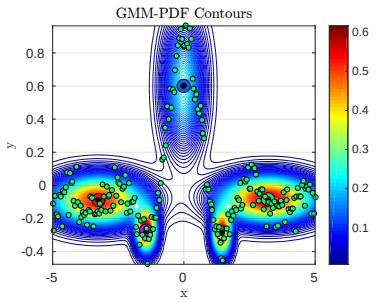

Figure 19: GMR (K=5) Result of 1D non-linear Dataset.

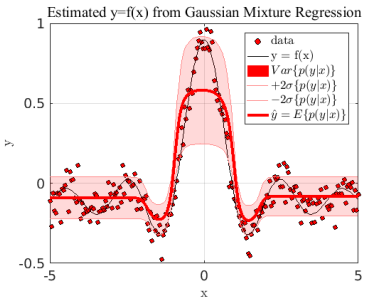

Figure 20: GMR (K=5) Result of 1D non-linear Sinc Dataset.

Additionally, to validate `my_gmr.m` function on **2D Datasets** you can run the sub-blocks 2 to load different 2D Datasets in the MATLAB testing script `test_gmr.m`. The list of available 2D dataset is `2d-cossine` and `2d-gmm`. Change the value in `dataset_type` to load a different dataset. You should obtain visualization similar to Fig. 21, and 22 depending on the dataset you have selected.

From all of the previous results, it is evident that GMR is a powerful algorithm, capable of modeling regressive functions with different complexities and topologies. For all datasets except

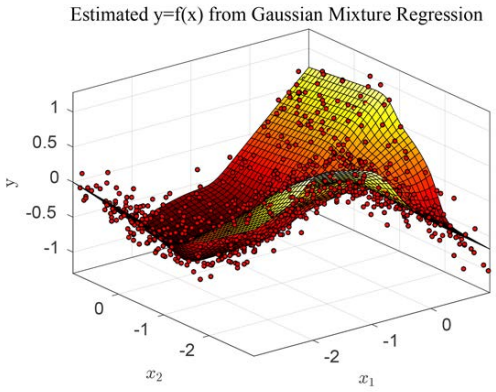

Figure 21: GMR (K=9) Result for dataset `2d-cossine`.

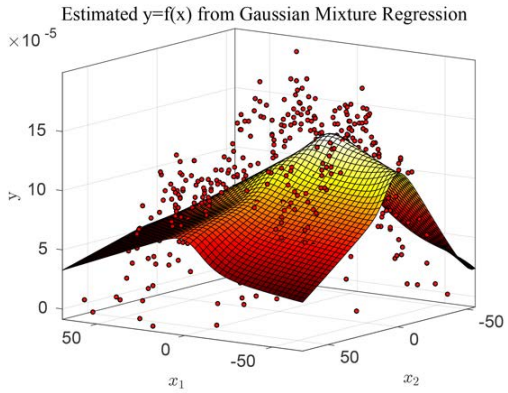

Figure 22: GMR (K=4) Result for dataset `2d-gmm`.

`2d-gmm` (i.e. 2D highly nonlinear dataset), GMR approximated a regressive function very close to the true one. The example dataset from the latter case is composed of a highly nonlinear output with much noise, for such datasets GMR would give a good estimate if we used as much parameters as possible, for example if we set $K = 50$ this would yield the regressive function in Figure 23.

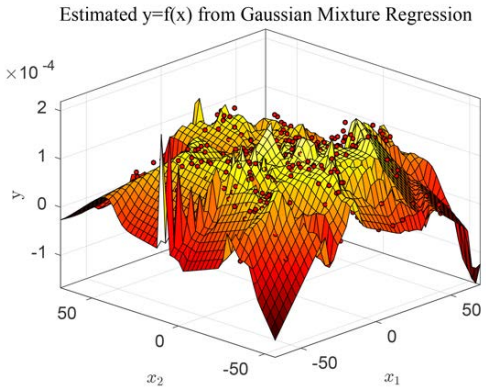

Figure 23: GMR (K=50) Result for data from Fig. 22.

This model provides a fairly better estimate than when $K = 4$, however, the complexity of the model is extremely high (i.e. 340 parameters for a dataset of 500 data-points). For this reason, we must apply a model selection procedure using cross-validation as we do for classification problems. This will be implemented in the second part of the assignment.

NOTE: For such type of datasets (Figure 23), GMR might not be the most suitable algorithm. Other type of *local* non-parametric methods would be more suitable, such as LWR (Locally Weighted Regression) [3], LWPR (Locally Weighted Projected Regression) [4] or GPR (Gaussian Process Regression) [5]. If you are interested in learning about these methods, they are covered in the Advanced Machine Learning Course offered at EPFL during the Spring semesters.

2.2 Regression Metrics

The metric you will use during this practical to compare the performance of each regressor will be the $Normalized\ Mean\ Square\ Error$ $(NMSE)$ or $the\ Coefficient\ of\ Determination$ (R^2).

If you have a vector of prediction $\hat{\mathbf{y}}$ and a vector $\mathbf{y}$ of the observed values corresponding to this prediction, you can compute the *Mean Square Error* (MSE) of the predictor :

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \tag{18}$$

The *Normalized Mean Square Error* ($NMSE$) is simply the MSE normalized by the variance of the observed values :

$$NMSE = \frac{MSE}{VAR(\mathbf{y})} = \frac{\frac{1}{M}\sum_{i=1}^{M}(\hat{\mathbf{y}}_i - \mathbf{y}_i)^2}{\frac{1}{M-1}\sum_{i=1}^{M}(\mathbf{y}_i - \mu)^2}, \tag{19}$$

where μ is the mean of the observed values : $\mu = \frac{1}{M}\sum_{i=1}^{M}\mathbf{y}_i$.
The *Coefficient of Determination* (R^2) is defined as :

$$R^2 = \left(\frac{\sum_{i=1}^{M}(\mathbf{y}_i - \overline{\mathbf{y}})(\hat{\mathbf{y}}_i - \overline{\hat{\mathbf{y}}})}{\sqrt{\sum_{i=1}^{M}(\mathbf{y}_i - \overline{\mathbf{y}})^2}\sqrt{\sum_{i=1}^{M}(\hat{\mathbf{y}}_i - \overline{\hat{\mathbf{y}}})^2}}\right)^2 = \frac{\left(\sum_{i=1}^{M}(\mathbf{y}_i - \overline{\mathbf{y}})(\hat{\mathbf{y}}_i - \overline{\hat{\mathbf{y}}})\right)^2}{\sum_{i=1}^{M}(\mathbf{y}_i - \overline{\mathbf{y}})^2\sum_{i=1}^{M}(\hat{\mathbf{y}}_i - \overline{\hat{\mathbf{y}}})^2} \tag{20}$$

where $\overline{\mathbf{y}}$ and $\overline{\hat{\mathbf{y}}}$ denotes respectively the average of the observed and predicted values of y.

TASK 5: Implement my_regression_metrics.m function (3pts)

In this task, you have to implement the `my_regression_metrics.m` function according to Equations 18 19 20.

```
function [MSE, NMSE, Rsquared] = my_regression_metrics( yest, y )
%MY_REGRESSION_METRICS Computes the metrics (MSE, NMSE, R squared) for
%   regression evaluation
%
%   input ----------------------------------------------------------------
%
%       o yest  : (P x M), representing the estimated outputs of P-dimension
%       of the regressor corresponding to the M points of the dataset
%       o y     : (P x M), representing the M continuous labels of the M
%       points. Each label has P dimensions.
%
%   output ---------------------------------------------------------------
%
%       o MSE       : (1 x 1), Mean Squared Error
%       o NMSE      : (1 x 1), Normalized Mean Squared Error
%       o R squared : (1 x 1), Coefficent of determination
```

Test Implementation

To evaluate this function you should run one of first sub-blocks (1a/1b/1c) of the `first` code block in the testing script `test_gmr_metrics.m`. This will load a Regression Dataset as in Figure 6, and 7, and a high dimensional dataset, The Wine Dataset[1]. By running the **second**

[1]The wine dataset can be found online: https://archive.ics.uci.edu/ml/datasets/Wine. In this we use the dataset for the white wine.

and **third** code block, you will fit a GMM and compute the output of GMR, respectively. Once you have implemented the `my_regression_metrics.m` function, you can run the **fourth** code block to evaluate your metrics with the encrypted `test_regression_metrics.p` function. If you get the following messages on your MATLAB command window, you can move on to the next task.

```
--- Checking my_regression_metrics.m ---
[Test1] Checking MSE against MLTOOLBOX function : Correct.
[Test2] Checking NMSE against MLTOOLBOX function : Correct.
[Test3] Checking R squared against MLTOOLBOX function : Correct.
```

TASK 6: Implement cross_validation_gmr.m function (2pts)

In this task, you have to implement the `cross_validation_gmr.m` function which computes the five introduced metrics for a range of K (i.e. AIC, BIC, MSE, NMSE and R^2). You will have to implement a F-fold cross-validation, where the data is first split into test and train parts as given by the valid_ratio, and the mean and standard deviation of each metric is calculated for the test data over F-folds. Be careful to use the function `split_regression_data` instead of the normal `split_data` function. `split_data` makes some assumption on the input data which are not valid in this case.

NOTE: The high dimensional data is projected from the original 11 dimensions to 2 dimensions using PCA before cross-validation. If you want to test on the original dataset, you should modify code sub-block.

```
function [ MSE_F_fold, NMSE_F_fold, R2_F_fold, AIC_F_fold, BIC_F_fold, ...
std_MSE_F_fold, std_NMSE_F_fold, std_R2_F_fold, std_AIC_F_fold, std_BIC_F_fold]
= cross_validation_gmr( X, y, cov_type, plot_iter, F_fold, tt_ratio,k_range )
%CROSS_VALIDATION_REGRESSION Implementation of F-fold cross-validation
% for gaussian mixture regression.
%
%   input ----------------------------------------------------------------
%
%       o X         : (N x M), a data set with M samples each being of
%                     dimension N each column corresponds to a datapoint
%       o y         : (P x M) array representing the y vector assigned to
%                        each datapoints
%       o F_fold    : (int), the number of folds in cross-validation.
%       o valid_ratio  : (double), Training/Testing Ratio.
%       o k_range   : (1 x K), Range of k-values to evaluate
%
%   output ---------------------------------------------------------------
%
%       o MSE_F_fold      : (1 x K), Mean Squared Error computed for each
%                           value of k averaged over the number of folds.
%       o NMSE_F_fold     : (1 x K), Normalized Mean Squared Error
%                           computed for each value of k averaged over the
%                           number of folds.
%       o R2_F_fold       : (1 x K), Coefficient of Determination computed
%                           for each value of k averaged over the number of
%                           folds.
%       o AIC_F_fold      : (1 x K), Mean AIC Scores computed for each
%                           value of k averaged over the number of folds.
```

```
29  %         o BIC_F_fold      : (1 x K), Mean BIC Scores computed for each
30  %                           value of k averaged over the number of folds.
31  %         o std_MSE_F_fold  : (1 x K), Standard Deviation of MSE computed
32  %                           for each value of k.
33  %         o std_NMSE_F_fold : (1 x K), Standard Deviation of NMSE
34  %                           computed for each value of k.
35  %         o std_R2_F_fold   : (1 x K), Standard Deviation of R^2 computed
36  %                           for each value of k averaged over the number of
37  %                           folds.
38  %         o std_AIC_F_fold  : (1 x K), Standard Deviation of AIC Scores
39  %                           computed for each value of k averaged over the
40  %                           number of folds.
41  %         o std_BIC_F_fold  : (1 x K), Standard Deviation of BIC Scores
42  %                           computed for each value of k averaged over the
43  %                           number of folds.
44  %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Test Implementation

Assuming that your `cross_validation_gmr.m` function is correctly implemented, by running the **fifth** block of code in the script `test_gmr_metrics.m`, one can visualize the cross-validation plots for each metric. For the BIC/AIC metrics, you should obtain plots similar to to Figure 24, 25 and 26.

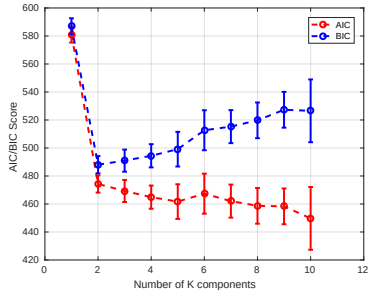

Figure 24: AIC/BIC Scores on training set of 10-fold cross-validation for the **1d-sine** dataset with a range of K from 1 to 10.

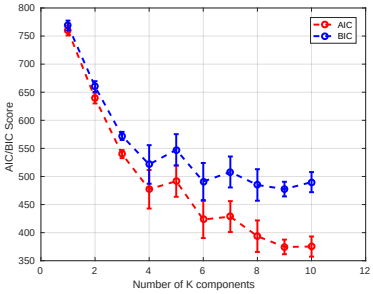

Figure 25: AIC/BIC Scores on training set of 10-fold cross-validation for the **1d-sinc** dataset with a range of K from 1 to 10.

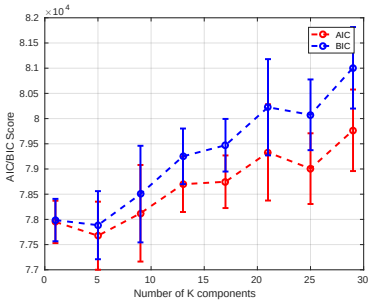

Figure 26: AIC/BIC Scores on training set of 10-fold cross-validation for the projected **Wine** dataset (to 2 principal components) with a range of K from 1 to 31 (in 4 steps).

According to the AIC/BIC metrics, one could infer that for the **1D-sine** dataset K=2 is the optimal model, for the **1D-sinc** dataset one could choose either K=4 or K=6 and for the **Wine** Dataset according to the AIC/BIC, it is not very clear which K is the optimal, but from the plot one would should K=5. Now, for the regression metrics, you should obtain plots similar to Figure 27, 28 and 29, for each of the datasets. Note that the axis for the MSE metric is on the left hand side of the plot, and the axis for $NMSE$ and R^2 metrics is on the right hand side of the plots. Clearly for the **1D-sine** dataset $K = 2$ is the required number of clusters to effectively model this data for regression, this corresponds to the optimal K from the AIC/BIC analysis. Regarding the **1D-sinc** dataset, it can be seen that $K = 4$ is the optimal value for regression, disambiguating the results from AIC/BIC. Finally, for the **Wine** dataset, the regression metrics do not really clarify which K would be the optimal, since both NMSE and R^2 metric seem to yield the same values for all K's, which corresponds to the AIC/BIC plot. However, we do have a peak in the MSE at K = 13, which yields the lowest MSE.

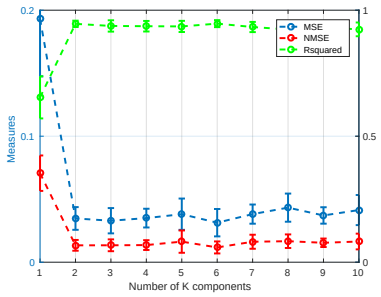

Figure 27: Result of 10-fold cross-validation for **1d-sine** dataset with a range of K from 1 to 10.

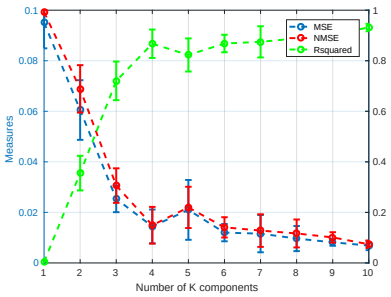

Figure 28: Result of 10-fold cross-validation for **1d-sinc** dataset with a range of K from 1 to 10.

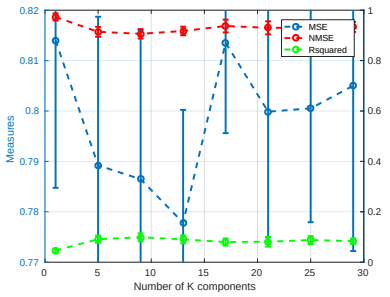

Figure 29: Result of 10-fold cross-validation for projected **Wine** dataset (to 2 principal components) with a range of K from 1 to 31 (in 4 steps).

If we test GMR on the original 11-D data from the Wine Dataset, this should yield a plot like the one in Figure 30. It can be seen that this dataset needs about $K = 10$ clusters. This takes high computational time since the data is high dimensional and with a high data points, and is shown here for reference, and need not be replicated.

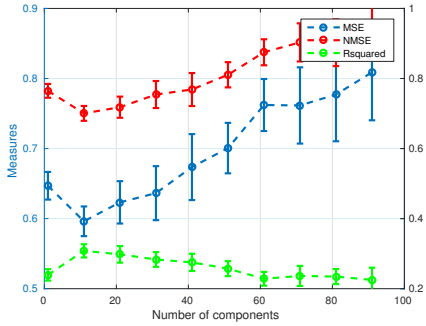

Figure 30: Result of 10-fold cross-validation for the high dimensional dataset (unprojected) with a range of K from 1 to 100 (in 10 steps).

3 Part 3: Sampling from a learned GMM

The beauty of GMMs is that it is able to learn the underlying structure of a given set of points, which is often refered as **density estimation**. That is to say, the result of a GMM fit to some data is technically not a clustering model, but a generative probabilistic model describing the distribution of the data. Therefore, one can draw samples from such a probabilistic model, i.e. generate new data points that are not part of the dataset but share the same underlying structure.

Drawing samples from a GMM is using the probabilistic model as a random generator. There are two types of elements to draw from. First, one must select which Gaussian k to draw a sample from based on the **prior probabilities** α of the GMM. Once a Gaussian is selected a datapoint is drawn from the **multivariate distribution** given by the paramater μ and σ of the selected Gaussian.

TASK 7: Implement a function to draw samples from a fitted GMM (2pts)

Your task is to implement new samples from a fitted GMM. First, load a dataset from the data function is the `utils` folder, and train a GMM model, as you did in Part 1 of the assignment.

Select the GMM parameters to properly fit your data. Then define a function that given a GMM model, draw `nbPoints` samples from the probabilistic model. Drawing a sample is performed in two steps, first drawing an integer value k, from the `prior distribution` of the GMM, representing which Gaussian to draw the sample from. Then, draw a multidimensional sample from the the $k-th$ Gaussian distribution.

`Implementation Hint:` `Useful functions randsrc(), mvrnd().`

Test implementation

To test your implementation run the visualization after the first `ADD CODE` block. This plot shows side-by-side the original data and the sampled ones. You should observe a similar structure as in Fig. 32. If not, first try to increase the number of Gaussians K in your GMM. Not enough Gaussians will result in your model not properly fitting the structure of your dataset as in Fig. 31.

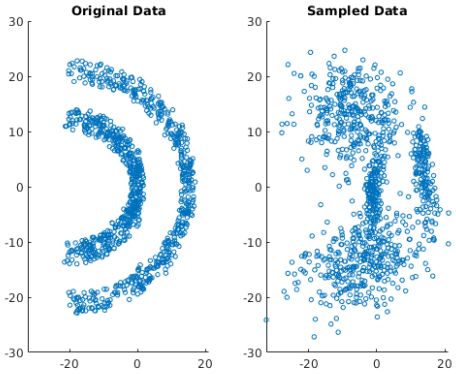

Figure 31: Sampling from a GMM with unsufficient number of Gaussians

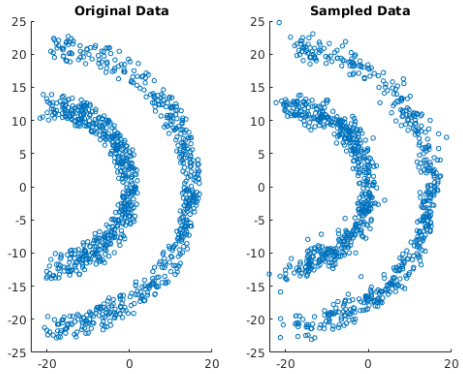

Figure 32: Sampling from a proper GMM fitting

TASK 8: Sample from a GMM fitted on high dimensional data (2pts)

Now that you are able to sample from low dimensional data, you have to perform the same thing but for a high dimensional dataset. For this task you will use the MNIST digit dataset given in the folder. Sample code to load the dataset and plot the figures are already provided as shown in Fig. 33. Bear in mind that fitting a GMM on a high-dimensionnal dataset is both time consuming and not-effective. You will have to implement a solution to overcome this issue.

Test implementation

If your sampling solution for the high dimensional dataset is correctly implemented you should observe a result similar to Fig. 34 when running the visualization of the drawn samples `XHat`. Remember that you probably need to provide a large k value in the param structure.

TASK 9: Sample from a GMM fitted to a specific class of data (2pts)

As you have noticed, the GMM is able to mostly generate random numbers in a completely unsupervised manner. For this task you will have to implement a solution to generate only numbers of a specific class. For example only some number 4 as in Fig. 35.

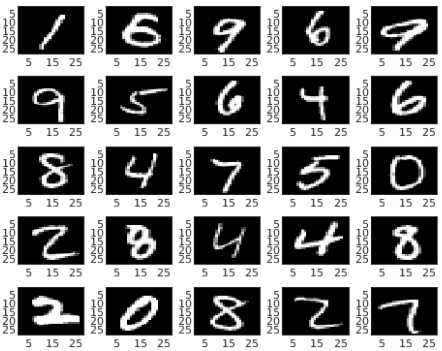

Figure 33: Visualization of samples from the digit dataset

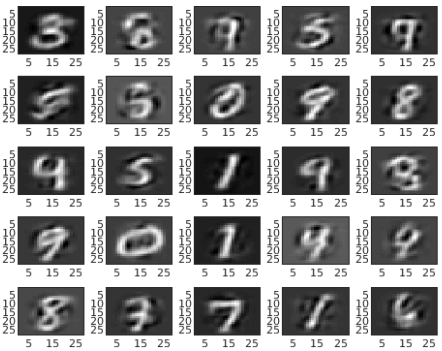

Figure 34: Sampling from a GMM fitted on the digit dataset

References

[1] Hsi G. Sung. *Gaussian mixture regression and classification*. PhD thesis, Rice University, Houston, Texas, 2004.

[2] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[3] William S. Cleveland and Susan J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[4] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *in Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000*, pages 1079–1086.

[5] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

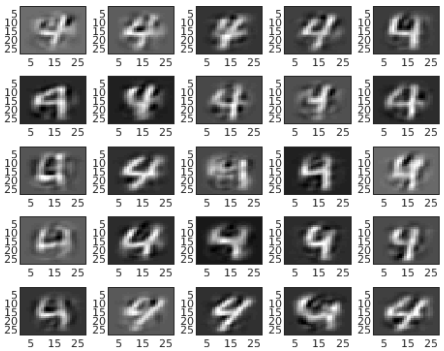

Figure 35: Sampling from a GMM fitted on a single class of the digit dataset