Laboratory 10

In this laboratory we will focus on Gaussian Mixture Models.

Gaussian Mixture Models

Gaussian Mixture Models allow approximating the density of a R.V. $\boldsymbol{X}$ when the density of $\boldsymbol{X}$ is not known. The GMM density consists of a weighted sum of M Gaussians:

[handwritten: DIVIDI IL DATASET IN SUBSET CHE ABBIANO UNA DISTRIBUZIONE GAUSSIANA]

$$\boldsymbol{X} \sim GMM(\boldsymbol{M}, \boldsymbol{S}, \boldsymbol{w}) \implies f_{\boldsymbol{X}}(\boldsymbol{x}) = \sum_{g=1}^{M} w_g \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$$

[handwritten: PESO]

where

[handwritten: MEDIA, COVARIANZA, PESO]

$$\boldsymbol{M} = [\boldsymbol{\mu}_1 \ldots \boldsymbol{\mu}_M] , \quad \boldsymbol{S} = [\boldsymbol{\Sigma}_1 \ldots \boldsymbol{\Sigma}_M] , \boldsymbol{w} = [w_1 \ldots w_M]$$

are the components means, covariance matrices and weights, respectively.

We have seen that a GMM can be interpreted as the marginal distribution obtained by marginalizing the joint density

$$f_{\boldsymbol{X}_i, G_i}(\boldsymbol{x}_i, g) = w_g \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g) , \quad f_{\boldsymbol{X}_i}(\boldsymbol{x}_i) = \sum_{g=1}^{M} f_{\boldsymbol{X}_i, G_i}(\boldsymbol{x}_i, g) .$$

G_i is a discrete *hidden* random variable that represents the Gaussian component that was responsible for the generation of $\boldsymbol{x}_i$. The joint density can be expressed as a product of the component (cluster) conditional distribution for $\boldsymbol{X}_i$ and the prior distribution for G_i:

$$f_{\boldsymbol{X}_i|G_i}(\boldsymbol{x}_i|g) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g) , \quad P(G_i = g) = w_g , \quad f_{\boldsymbol{X}_i, G_i}(\boldsymbol{x}_i, g) = f_{\boldsymbol{X}_i|G_i}(\boldsymbol{x}_i|g) P(G_i = g)$$

Write a function `logpdf_GMM(X, gmm)` that computes the log-density of a GMM for a set of samples contained in matrix X. As we did in Laboratory 4, we will assume that X is a matrix of samples of shape (D, N), where D is the size of a sample and N is the number of samples in X. As in Laboratory 4, you can either compute directly the 1-D vector of densities or use a `for` loop to compute the density for each sample. To encode a GMM you can use different ways. A possible, simple approach consists in representing a GMM as a list of component parameters:

```
gmm = [(w1, mu1, C1), (w2, mu2, C2), ...]
```

Suggestions:

- You can make use of the function `logpdf_GAU_ND` that you wrote for Laboratory 4 to compute the terms $\log \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$ for all samples $\boldsymbol{x}_i, i = 1 \ldots N$ and for all components $g = 1 \ldots M$.

- You can arrange the terms in a matrix S with shape (M, N). Element S[g, i] of S contains the logarithm of the joint probability of sample $\boldsymbol{x}_i$ and component (cluster) g, and a row S[g, :] therefore contains all joint densities for all samples. These can be computed as in LAboratory 5: `S[g, :] = logpdf_GAU_ND(X, mu, C) + numpy.log(w)` where w, mu, C = gmm[g].

- Finally, you can compute the log-marginal $\log f_{\boldsymbol{X}_i}(\boldsymbol{x}_i)$ for all samples $\boldsymbol{x}_i$ by using the log-sum-exp function: `logdens = scipy.special.logsumexp(S, axis=0)`. The result will be an array of shape (N,), whose component i will contain the log-density for sample $\boldsymbol{x}_i$.

NOTE: The process is the *same* we used to compute the log-densities $\log f_{\boldsymbol{X}_i}(\boldsymbol{x}_i) = \log \sum_c f_{\boldsymbol{X}_i, C_i}(\boldsymbol{x}_i, c)$ for the MVG classifier in Laboratory 5. Indeed, we can interpret each Gaussian component as representing a sub-class (i.e. clusters) of our data.

To check that the log-density is correct, you can compute the log-density over a reference dataset using a reference GMM. The file `GMM_data_4D.npy` contains a set $\boldsymbol{X}$ of 4-dimensional samples (as a numpy matrix of shape (4×1000)). It can be loaded using `numpy.load`. File `Data/GMM_4D_3G_init.json` contains a reference GMM. It can be loaded with function `load_gmm` contained in file `Data/GMM_load.py`.

[handwritten: gmm]

The log-densities for all samples in $\boldsymbol{X}$ can be loaded using `numpy.load` from file `Data/GMM_4D_3G_init_ll.npy`. Check your log-density function provides the same results. You can also verify the log-density on 1-D data (stored as a 1×1000 matrix). You can find data, GMM and log-likelihoods in the `Data` folder. The names of the files are the same as for the 4-D dataset, replacing `4D` with `1D`.

GMM estimation: the EM algorithm

The EM algorithm can be used to estimate the parameters of a GMM that maximize the likelihood for a training set $\boldsymbol{X}$. The EM algorithm consists of two steps:

[handwritten: È LA PROBABILITÁ APOSTERIORI DI OGNI COMPONENTE PER OGNI PUNTO DI DATI, DATO IL MODELLO CORRENTE]

- **E-step:** compute the posterior probability for each component of the GMM for each sample, using an estimate $(\boldsymbol{M}_t, \boldsymbol{\mathcal{S}}_t, \boldsymbol{w}_t)$ of the model parameters. These quantities are also called *responsibilities*

$$\gamma_{g,i} = P(G_i = g | \boldsymbol{X}_i = \boldsymbol{x}_i, \boldsymbol{M}_t, \boldsymbol{\mathcal{S}}_t, \boldsymbol{w}_t)$$

[handwritten: CI DICE QUANTO UN PARTICOLARE COMPONENTE È RESPONSABILE PER UN DATO PUNTI DI DATI]

[handwritten: È UGUALE A]

- **M-step:** Update the model parameters. For each component, we can use the statistics

[handwritten: P = np.exp (log Sj - log Sj MARG)]

$$Z_g = \sum_{i=1}^{N} \gamma_{g,i}, \quad \boldsymbol{F}_g = \sum_{i=1}^{N} \gamma_{g,i}\boldsymbol{x}_i, \quad \boldsymbol{S}_g = \sum_{i=1}^{N} \gamma_{g,i}\boldsymbol{x}_i\boldsymbol{x}_i^T$$

[handwritten: DEVI AGGIORNARE I PARAMETRI USANDO LE RESPONSAB. CALCOLATE]

to obtain the new parameters

[handwritten: Con il calcolo di queste statistiche calcoli i nuovi valori]

$$\boldsymbol{\mu}_{g\,t+1} = \frac{\boldsymbol{F}_g}{Z_g}, \quad \boldsymbol{\Sigma}_{g\,t+1} = \frac{\boldsymbol{S}_g}{Z_g} - \boldsymbol{\mu}_{g\,t+1}\boldsymbol{\mu}_{g\,t+1}^T, \quad w_{g\,t+1} = \frac{Z_g}{\sum_{g'=1}^{M} Z_{g'}} = \frac{Z_g}{N}, \quad g = 1 \dots M$$

Implement the GMM EM estimation procedure. The estimation procedure should iterate from an initial estimate of the GMM, and keep computing E and M -steps until a convergence criterion is met.

[handwritten: QUANDO I DATI NON CAMBIANO PIÙ]

Suggestions:

- The computation of the posterior distributions $\gamma_{g,i} = P(G_i = g | \boldsymbol{X}_i = \boldsymbol{x}_i, \boldsymbol{M}_t, \boldsymbol{\mathcal{S}}_t, \boldsymbol{w}_t)$ can be done in the same way as we computed class posterior probabilities for a MVG model. We can also re-use most of the code that was used to compute the log-densities in the previous section.

- Build matrix `S` containing component conditional densities, and add to each row the corresponding log-prior $\log P(G_i = g) = \log w_g$. The results is the matrix of joint densities $f_{\boldsymbol{X}_i, G_i}(\boldsymbol{x}_i, g)$ (the same matrix `S` that we built in the previous section).

- Compute the marginal densities using the log-sum-exp function (see previous section or the MVG laboratory)

- Compute the posterior distribution for each sample:

$$\gamma_{g,i} = P(G_i = g | \boldsymbol{X}_i = \boldsymbol{x}_i, \boldsymbol{M}_t, = \boldsymbol{\mathcal{S}}_t, \boldsymbol{w}_t) = \frac{f_{\boldsymbol{X}_i, G_i}(\boldsymbol{x}_i, g)}{f_{\boldsymbol{X}_i}(\boldsymbol{x}_i)} = e^{\log f_{\boldsymbol{X}_i, G_i}(\boldsymbol{x}_i, g) - \log f_{\boldsymbol{X}_i}(\boldsymbol{x}_i)}$$

You can compute the logarithm of all $\gamma_{g,i}$'s by removing, from each row of the joint log-densities matrix `S`, the row vector containing the N marginal log-densities computed in the previous step. The $M \times N$ matrix of posterior probabilities can then be obtained by computing the exponential of each element of the result. This is the same procedure we used to compute class posterior probabilities for the MVG classifier.

- The statistics can be computed from the matrix of components posterior probabilities.

- As stopping criterion, we can use the log-likelihood of the training set

$$\ell(\boldsymbol{M}_t, \boldsymbol{\mathcal{S}}_t, \boldsymbol{w}_t) = \sum_{i=1}^{n} \log GMM(\boldsymbol{x}_i | \boldsymbol{M}_t, \boldsymbol{\mathcal{S}}_t, \boldsymbol{w}_t) = \sum_{i=1}^{n} f_{\boldsymbol{X}_i}(\boldsymbol{x}_i) .$$

The log-likelihood for the training set using the current parameter estimate can be computed from the outputs of function `logpdf_GMM`. Alternatively, you can compute the log-likelihood from the marginal densities at the end of the E-step

– NOTE: In this case you would obtain the log-likelihood for the previous M-step, not for the one of the current EM iteration

– NOTE: In the following we will use the *average* log-likelihood, i.e. we will divide the log-likelihood by N

We stop the iterations when the average log-likelihood increases by a value lower than a threshold: $\ell(\boldsymbol{M}_t, \boldsymbol{\mathcal{S}}_t, \boldsymbol{w}_t) - \ell(\boldsymbol{M}_{t-1}, \boldsymbol{\mathcal{S}}_{t-1}, \boldsymbol{w}_{t-1}) \leq \Delta_l$. In the examples below $\Delta_l = 10^{-6}$ was used.

- **To verify your implementation, check your log-likelihood is increasing. If the log-likelihood becomes smaller at some iteration, then your implementation is very likely to be incorrect**. We also suggest that, when your implementation seems stable, you let the EM run for more iterations than those required to reach the stopping criterion to check that the log-likelihood does not start decreasing. After this check, of course, these additional iterations can be removed.

Apply the EM algorithm to the data in `GMM_data_4D.npy` . You can use `Data/GMM_4D_3G_init.json` as initial GMM. You can find a solution in file `GMM_4D_3G_EM.json` . The *average* log-likelihood for the trained GMM is -7.26325603 . The corresponding files are also available for the 1-D dataset. Below you can see the plot of the estimated density for the 1-D dataset.
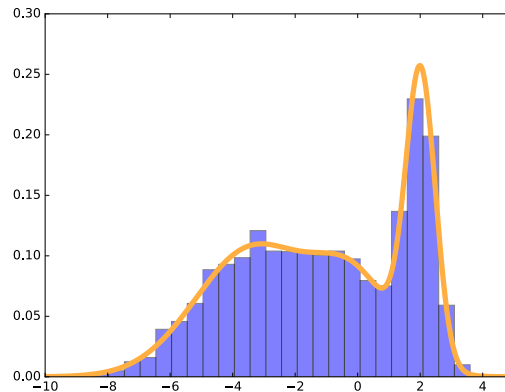
LA DIFFERENZA PRINCIPALE TRA EM E LBG E' CHE EM INIZIA CON UN MODELLO INIZIALE FORNITO DALL'UTENTE E CERCA DI OTTIMIZZARLO, MENTRE LBG E' IL MODO IN CUI INIZIANO. ESSO INIZIA CON UN SINGOLO COMPONENTE E RADDOPPIA IL NUMERO DI COMPONENTI AD OGNI ITERAZIONE CERCANDO DI OTTIMIZZARE IL MODELLO

L'OBIETTIVO E' TROVARE UN INSIEME DI VETTORI DI CODICE CHE MINIMIZZANO LA DISTORSIONE MEDIA QUADRATICA TRA I VETTORI DI DATI E IL VETTORE DI CODICE PIU' VICINO. E' MOLTO SIMILE AL K-MEANS, MA CON UNA DIFFERENZA CHE INIZIA CON UN SOLO VETTORE DI CODICE ED AD OGNI SI INIZIA CON UNA SINGOLA TUPLA CHE RADDOPPIA AD OGNI ITERAZIONE. FINO A RAGGIUNGERE NCOMPONENTS CHE IL NUMERO DI COMPONENTI DESIDERATE.

LBG algorithm

The EM algorithm requires an initial guess for the GMM parameters. The LBG algorithm allows us to incrementally construct a GMM with $2G$ components from a GMM with G components. Starting with a single-component GMM (i.e. a Gaussian density), we can build a 2-component GMM and then use the EM algorithm to estimate a ML solution for the 2-components model. We can then split the 2 components to obtain a 4-components GMM, and re-apply the EM algorithm to estimate its parameters, and so on.

We can use the Maximum Likelihood solution for a Gaussian density as starting point:

```
GMM_1 = [(1.0, mu, C)]
```
W

where `mu` and `C` are the empirical mean and covariance matrix of the dataset.

A possible way to split the GMM consists in replacing component $(w_g, \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$ with two components

$$\left(\frac{w_g}{2}, \boldsymbol{\mu}_g - \boldsymbol{d}_g, \boldsymbol{\Sigma}_g\right) \ , \quad \left(\frac{w_g}{2}, \boldsymbol{\mu}_g + \boldsymbol{d}_g, \boldsymbol{\Sigma}_g\right),$$

The displacement vector $\boldsymbol{d}_g$ can be computed, for example, by taking the leading eigenvector of $\boldsymbol{\Sigma}_g$ scaled by the square root of the corresponding eigenvalue, multiplied by some factor α:

```
U, s, Vh = numpy.linalg.svd(Sigma_g)
d = U[:, 0:1] * s[0]**0.5 * alpha
```

L'ordine in cui i componenti vengono aggiunti può influenzare il risultato finale del modello GMM (Gaussian Mixture Model) perché l'algoritmo Linde-Buzo-Gray (LBG) utilizzato qui divide un componente esistente in due nuovi componenti ad ogni iterazione. L'ordine in cui questi componenti vengono divisi e aggiunti può influenzare la convergenza dell'algoritmo. Nel tuo codice, i componenti vengono divisi in due nuovi componenti basati su $\mu + d$ e $\mu - d$. Questi due nuovi componenti vengono poi aggiunti alla lista newGmm. Se cambi l'ordine in cui questi componenti vengono aggiunti alla lista, potrebbe cambiare l'ordine in cui l'algoritmo LBG sceglie di dividere i componenti nelle iterazioni successive. Questo può portare a una soluzione finale diversa, anche se i componenti stessi sono gli stessi. In altre parole, l'ordine in cui i componenti vengono aggiunti può influenzare il percorso che l'algoritmo LBG prende per raggiungere la soluzione finale, il che può portare a soluzioni diverse a seconda dell'ordine iniziale dei componenti.

In practice, we are displacing the new components along the direction of maximum variance, using a step that is proportional to the standard deviation (i.e. the scale of the features along that direction) of the component we are splitting.

Implement the LBG algorithm. At each LBG iteration, we produce a 2G-components GMM from a G-components GMM. The 2G components GMM can be used as initial GMM for the EM algorithm. Retrain the 2G-GMM using the EM algorithm

NOTE: After the split, the initial iteration of the EM algorithm for the $2G$-components GMM may have a lower log-likelihood than you had at the end of the EM iterations for the G-components GMM.

NOTE: Depending on the order in which you insert the two components during the LBG updates your GMM may be equivalent, but with the components in a different order. If you want to obtain the same GMMs as in the files you should add the components in the same order as they appera in the above equation. File `GMM_4D_4G_EM_LBG.json` contains a 4-component GMM obtained using a factor $\alpha = 0.1$. The final average log-likelihood is `-7.25337844` .

You can also find the gmm for the 1-D dataset. Below you can see the plot of the estimated density.
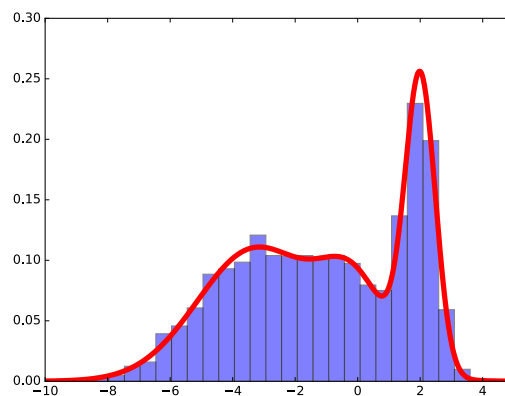
Constraining the eigenvalues of the covariance matrices

The GMM log-likelihood is not bounded above for $M \geq 2$. Indeed, we can have arbitrarily high log-likelihood by centering one component at one of the N samples, and letting the corresponding covariance matrix shrink towards zero. To avoid these kind of degenerate solutions, we can constrain the minimum values of the eigenvalues of the covariance matrices. $\qquad$ *VINCOLARE I VALORI MINIMI DEGLI AUTOVALORI DELLE MATRICE DI COVARIANZA*

A possible solution consists in constraining the eigenvalues of the covariance matrices to be larger or equal to a lower bound $\psi > 0$. It's possible to show that the solution can be obtained by modifying the output of the M-step in the previous section. In particular, after we have computed $\mathbf{\Sigma}_{g_{t+1}}$, we can compute its SVD: *È LA SCOMPOSIZIONE DI UNA MATRICE* $\qquad$ *U e V SONO ORTOGONALI*
A VIENE SCOMPOSTO IN $\qquad$ *Σ MATRICE DIAGONALE, DOVE cui ELEMENTI SONO I VALORI SINGOLARI*
$A = U * \Sigma * V^*$
$$U_g, S_g, U_g^T = \mathbf{\Sigma}_{g_{t+1}} \qquad \text{DI A.}$$
where $\mathbf{S}_g$ is a diagonal matrix of singular values, and replace the estimated covariance with

$$\mathbf{\Sigma}_{g_{t+1}} = \mathbf{U}_g \mathbf{S}'_g \mathbf{U}_g^T , \quad \mathbf{S}'_{g_{ii}} = \max(\psi, \mathbf{S}_{g_{ii}})$$

where $\mathbf{S}'_{g_{ii}}$ is the i-th element of the diagonal of matrix $\mathbf{S}'_g$. Assuming that `covNew` is the variable that contains $\mathbf{\Sigma}_{g_{t+1}}$, the procedure can be implemented as

VA MESSO NEL M-STEP DELL' EM

```
U, s, _ = numpy.linalg.svd(covNew)
s[s<psi] = psi
covNew = numpy.dot(U, mcol(s)*U.T)
```

ESEGUE LA DECOMPOSIZIONE A VALORI SINGOLARI CHE RAPPRESENTA LA MATRICE DI COV DI UN COMPONENTE DEL MODELLO DI MISCELA GAUSSIANA.
$s[s<psi] = psi$ È UN PASSAGGIO DI REGOLARIZZAZIONE ASSICURA CHE NESSUN VALORE SIA TROPPO PICCOLO POICHÉ VALORI PICCOLI POSSONO CREARE PROBLEMI DI STABILITÀ.
4 *SIGMA a DA LA MATRICE DI COV CON VALORI VALIDI NON MINORE DI psi.*

NOTE: The log-likelihood should again increase at each iteration.

NOTE: When applying the eigenvalue threshold, you should make sure that the constraint is met also by the inital GMM estimate. In particular, when using the LBG algorithm, you should apply the transformation also to the initial, single-component (MVG) solution, otherwise the first EM iteration of the 2-components GMM may result in a decrease of the log-likelihood.

Diagonal and tied-covariance GMMs

The diagonal variant of the GMM we just implemented consists of a model whose components all have diagonal covariance matrices. The ML solution can be trivially obtained by modifying the M-step by keeping only the diagonal elements of $\boldsymbol{\Sigma}_{g\,t+1}$. The approach is the same as for the diagonal covariance MVG model: after we have computed $\boldsymbol{\Sigma}_{g\,t+1}$, we replace it with

$$\boldsymbol{\Sigma}_{g\,t+1} \longleftarrow \mathrm{Diag}(\boldsymbol{\Sigma}_{g\,t+1})$$

i.e., `Sigma_g = Sigma_g * numpy.eye(Sigma_g.shape[0])` (a more efficient implementation would store and make use of only the diagonal terms of the covariance matrix, however simply zero-ing the out-of-diagonal elements is already enough to evaluate the goodness of the approach).

We can also compute ML solutions for tied models, where each component has a covariance matrix $\boldsymbol{\Sigma}_g = \boldsymbol{\Sigma}$. The tied solution can also be obtained by modifying the M-step of the non-tied approach. In particular, we can compute matrices $\boldsymbol{\Sigma}_{g\,t+1}$ as in the standard M-step. Then, we can replace each matrix $\boldsymbol{\Sigma}_{g\,t+1}$ with

$$\boldsymbol{\Sigma}_{g\,t+1} \longleftarrow \frac{1}{N} \sum_{g=1}^{M} Z_g \boldsymbol{\Sigma}_{g\,t+1} = \sum_{g=1}^{M} w_{g\,t+1} \boldsymbol{\Sigma}_{g\,t+1}$$

NOTE: the minimum eigenvalue constraints we have discussed in the previous section should be applied AFTER we have computed the diagonal or the tied updates.

PARAMETRI
$\alpha = 0.1$ $\psi = 0.01$
$\Delta \ell = 10^{-6}$ NUMBER COMPONENT

GMM for classification

GMMs can be used for classification in a similar way as we did for the MVG classifier. In particular, we can train a GMM $GMM(\boldsymbol{M}_c, \boldsymbol{\mathcal{S}}_c, \boldsymbol{w}_c)$ for each class of the IRIS dataset, and then use the GMM log-density function to compute class-conditional distributions $f_{\boldsymbol{X}_t|C_t}(\boldsymbol{x}_t|c) = GMM(\boldsymbol{x}_t|\boldsymbol{M}_c, \boldsymbol{\mathcal{S}}_c, \boldsymbol{w}_c)$ for all classes. Results are in the table below (we consider only the accuracy / error rate metric in this laboratory; the LBG parameter α was set to 0.1; the minimum value for the eigenvalues of the covariance matrix ψ was set to 0.01; the EM stopping criterion was set to $\Delta_l = 10^{-6}$; the same number of components was chosen for all classes).

NOTE: The tied covariance GMM model assumes that the covariance of a single GMM $\boldsymbol{\Sigma}_{c,g} = \boldsymbol{\Sigma}_c$ are the same, but each GMM of each class has a different covariance matrix $\boldsymbol{\Sigma}_c$. **This is different from the tied MVG model.** In particular, for the 1-component model, the Tied GMM corresponds to the non-tied MVG model. We could implement also a globally tied GMM, which would correspond to the Tied MVG for 1-component GMMs, where the covariance matrices of each component of all the 3 GMMs would be the same. However, in this case we would not be able to train the 3 GMMs of each class independently, as the likelihood function would not factorize over the different parameters (as in the tied MVG case). We would then need to compute joint E and M steps to obtain the ML solution. You are free to try this on your own.

GENERALMENTE IL NUMERO DI COMPONENTI GMM È UGUALE AL NUMERO DI CLASSI DEL TUO DATASET.

| | Components | | | | |
GMM Type	1	2	4	8	16
Full Covariance (standard)	4.0%	4.0%	4.0%	4.0%	4.0%
Diagonal Covariance	4.0%	4.0%	6.0%	2.0%	4.0%
Tied Covariance	4.0%	4.0%	4.0%	4.0%	6.0%

PUOI USARE DUE MATRICI DI COV

- **DIAGONAL** : LA MATRICE DI COV DI OGNI COMPONENTE É UNA MATRICE DIAGONALE.
CIOÉ SI ASSUME CHE TUTTE LE CARATTERISTICHE SIANO INDIPENDENTI L'UNO
DALL'ALTRO. PUÓ PORTARE A MODELLI MENO ACCURATI SE LE CARAT
SONO CORRELATE

- **TIED**: TUTTI I COMPONENTI DEL GMM CONDIVIDONO LA STESSA MATRICE DI COV. QUESTO
SIGNIFICA CHE SI ASSUME CHE LA CORRELAZIONE TRA LE DIVERSE DIM DEI DATI SIA
LA STESSA PER TUTTI I COMPONENTI. PUOI AVERE MENO ACCURATEZZA SE LE CORRELAZ
CAMBIANO TRA I DIVERSI COMPONENTI.

n Components: É IL PARAMETRO CHE SPECIFICA IL NUMERO DI COMPONENTI CHE IL MODELLO GMM
DOVREBBE AVERE. OGNI COMPONENTE É UNA DISTRIBUZIONE GAUSSIANA, L'INTERO
MODELLO É UNA MISCELA. DETERMINA QUINDI QUANTE DIVERSE DISTRIBUZIONI SONO PRESENTI

For binary tasks we can also compute log-likelihood ratios

$$llr(\boldsymbol{x}_t) = \frac{GMM(\boldsymbol{x}_t|\boldsymbol{M}_1, \boldsymbol{S}_1, \boldsymbol{w}_1)}{GMM(\boldsymbol{x}_t|\boldsymbol{M}_0, \boldsymbol{S}_0, \boldsymbol{w}_0)}$$

and then proceed as we did for MVG and similar models.

In `Data/ext_data_binary.npy` and `Data/ext_labels_binary.npy` you can find an additional, artificial dataset with two classes, 1 and 0. Split the dataset in model training / validation sets with the usual split function, train a GMM for each class, and then compute the LLRs on the validation set. From the LLRs, compute minimum and actual DCF for an application $\pi_T = 0.5$. You can find below the results for different models (also in this case we use the same number of components for both classes). We used the same values for α and ψ we used for the multiclass Iris task.

GMM Type	Components				
	1	2	4	8	16
	minDCF / actDCF				
Full Covariance	0.4984 / 0.5398	0.4302 / 0.4416	0.5195 / 0.5706	0.5804 / 0.6177	0.6364 / 0.6640
Diagonal Covariance	0.5203 / 0.5625	0.4643 / 0.4643	0.4213 / 0.4513	0.4781 / 0.4781	0.4870 / 0.5446
Tied Covariance	0.4984 / 0.5398	0.4984 / 0.5398	0.4416 / 0.4643	0.4278 / 0.4846	0.4383 / 0.5252

Project

In this section we apply the GMM models to classification of the project data.

PER NON AVERE TEMPI TROPPO LUNGHI

*FULL
DIAGONAL*

For each of the two classes, we need to decide the number of Gaussian components (hyperparameter of the model). Train full covariance models with different number of components for each class (suggestion: to avoid excessive training time you can restrict yourself to models with up to 32 components). Evaluate the performance on the validation set to perform model selection (again, you can use the minimum DCF of the different models for the target application). Repeat the analysis for diagonal models. What do you observe? Are there combinations which work better? Are the results in line with your expectation, given the characteristics that you observed in the dataset? Are there results that are surprising? (Optional) Can you find an explanation for these surprising results?

1

We have analyzed all the classifiers of the course. For each of the main methods (GMM, logistic regression, SVM — we ignore MVG since its results should be significantly worse than those of the other models, but feel free to test it as well) select the best performing candidate. Compare the models in terms of minimum and actual DCF. Which is the most promising method for the given application?

2

Now consider possible alternative applications. Perform a qualitative analysis of the performance of the three approaches for different applications (keep the models that you selected in the previous step). You can employ a Bayes error plot and visualize, for each model, actual and minimum DCF over a wide range of operating points (e.g. log-odds ranging from -4 to $+4$). What do you observe? In terms of minimum DCF, are the results consistent, preserving the relative ranking of the systems? What about actual DCF? Are there models that are well calibrated for most of the operating point range? Are there models that show significant miscalibration? Are there models that are harmful for some applications? We will see how to deal with these issue in the last laboratory.

3

USARE DIVERSI VALORI DI LOG-ODDS VUOL DIRE CALCOLARE GLI ERRORI IN BASE AL VARIARE DELLA SOGLIA E VEDERE COME CAMBIANO LE PREDIZIONI. QUINDI LI DISEGNI IN BASE A COME VARIA LA PRIOR CHE DA PIÙ PESO AD UNA CLASSE

Risultati punto 1

nComponents 1
 covType is Full
minDCF 0.2629128264208909
actDCF 0.6641225038402456
 covType is Diag
minDCF 0.25696044546851
actDCF 0.6842517921146953

nComponents 2
 covType is Full
minDCF 0.21698988735279057
actDCF 0.6152233742959549
 covType is Diag
minDCF 0.24887992831541217
actDCF 0.6593061955965182

nComponents 4
 covType is Full
minDCF 0.21614183307731696
actDCF 0.4256912442396313
 covType is Diag
minDCF 0.14810547875064006
actDCF 0.41789874551971323

nComponents 8
 covType is Full
minDCF 0.17860343061955963
actDCF 0.29782706093189965
 covType is Diag
minDCF 0.14626536098310292
actDCF 0.4078341013824884

nComponents 16
 covType is Full
minDCF 0.1631304403481823
actDCF 0.2786898361495136
 covType is Diag
minDCF 0.16215437788018433
actDCF 0.3735279057859703

nComponents 32
 covType is Full
minDCF 0.2337269585253456
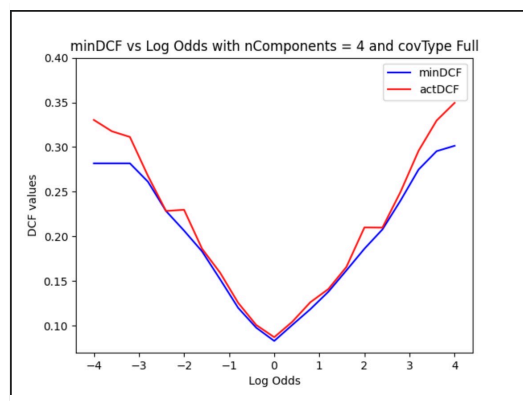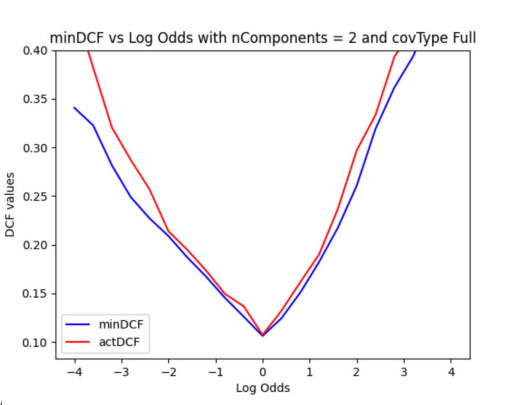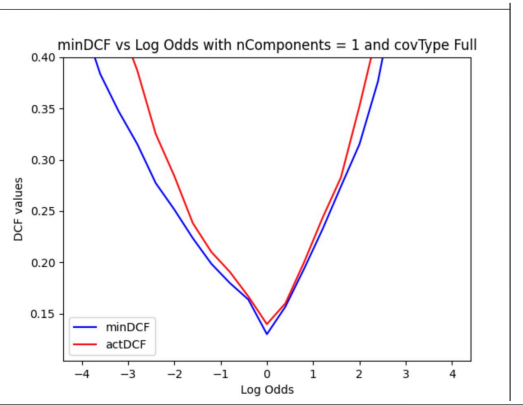actDCF 0.41761072708653346
 covType is Diag
minDCF 0.17661930363543263
actDCF 0.2595526113671275

· DAI DATI FORNITI SEMBRA CHE CON UN NUMERO MAGGIORE DI COMPONENTI TENDA A FORNIRE UN MINDCF PIÚ BASSO, CHE É DESIDERABILE IN QUANTO INDICA UN ERRORE DI CLASSIFICAZIONE INFERIORE. PERÓ NON CONTINUA AD AUMENTARE QUESTO SUGGERISCE CHE POTREBBE ESSERCI UN PUNTO OTTIMALE DOPO IL QUALE L'AGGIUNTA DI ALTRE COMPONENTI NON MIGLIORA LE PRESTAZIONI.

· FULL SEMBRA AVERE MIGLIORI PRESTAZIONI FORSE PERCHÉ IL MODELLO É IN GRADO DI CATTURARE MEGLIO LE RELAZIONI TRA LE DIVERSE CARATTERISTICHE QUANDO É PERMESSO AVERE COV COMPLETE
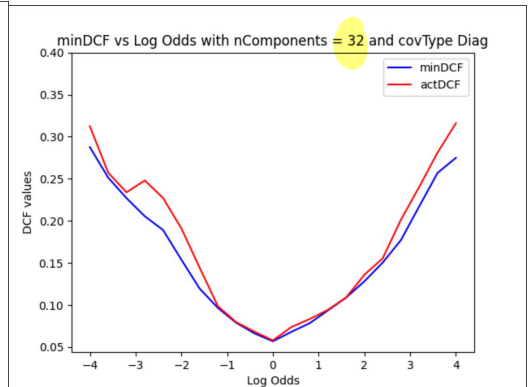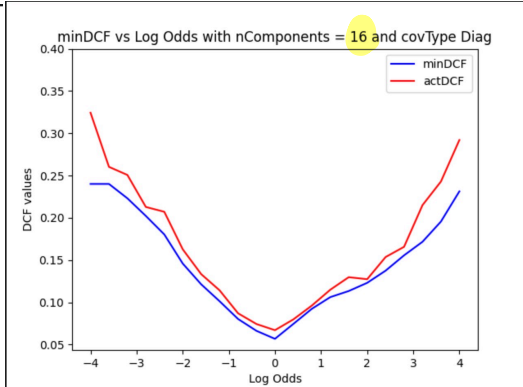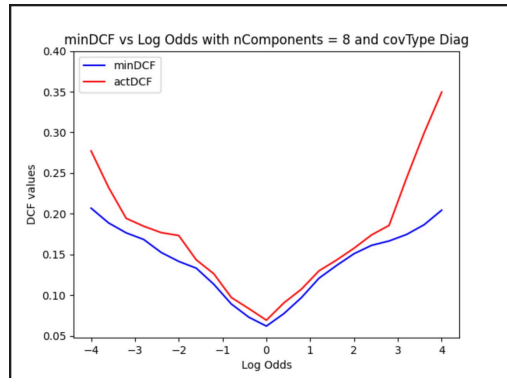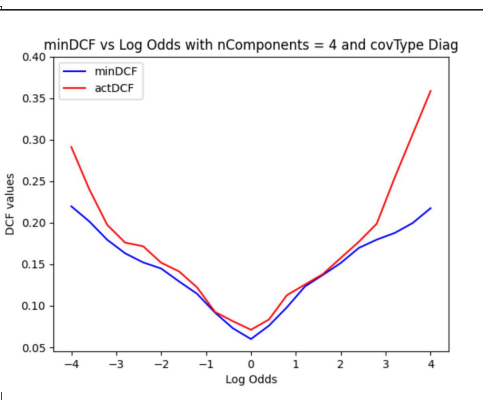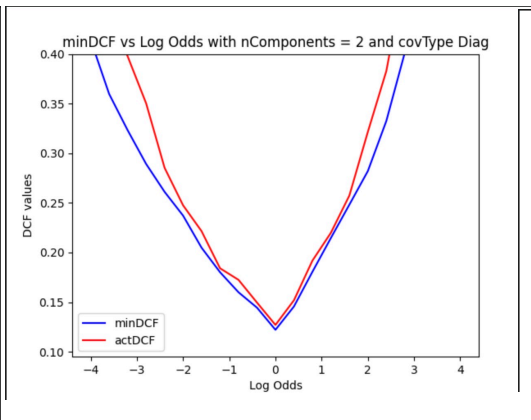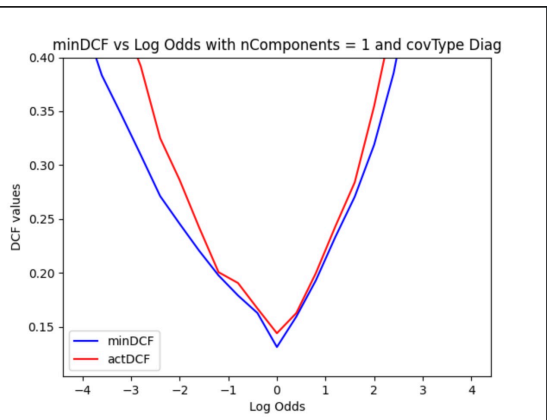
RISULTATO SORPRENDENTE POTREBBE ESSERE CHE IL MINDCF NON DIMINUISCE SEMPRE CON L'AUMENTO DEL NUMERO DI COMPONENTI. QUESTO POTREBBE ESSERE DOVUTO AL FATTO CHE UN MODELLO CON PIÚ COMPONENTI É PIÚ COMPLESSO E POTREBBE SOFFRIRE DI OVERFITTING

minDCF vs Log Odds with nComponents = 1 and covType Full

minDCF vs Log Odds with nComponents = 2 and covType Full

minDCF vs Log Odds with nComponents = 4 and covType Full

minDCF vs Log Odds with nComponents = 8 and covType Full

minDCF vs Log Odds with nComponents = 16 and covType Full

minDCF vs Log Odds with nComponents = 32 and covType Full
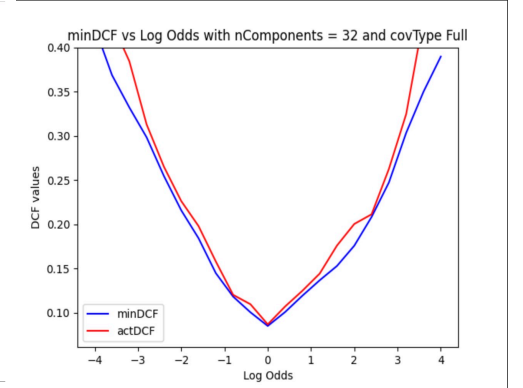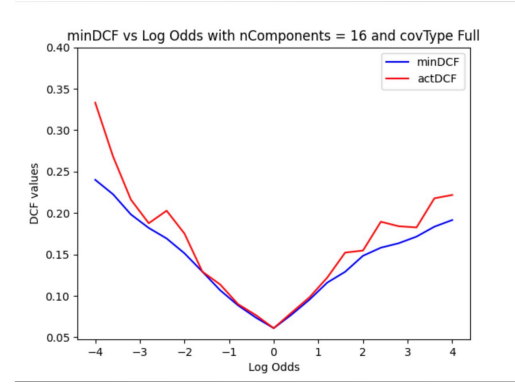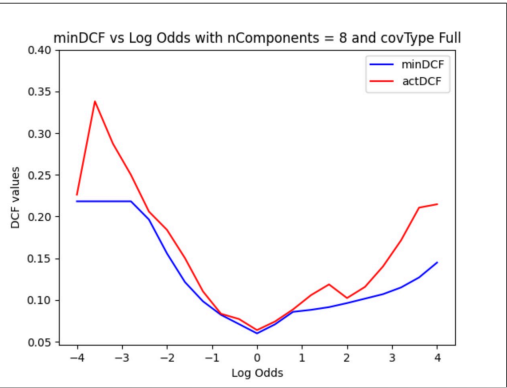
IL VALORE MINIMO DI ACTDCF =

```
minimum value of minDCF {'cn': 16, 'covType': 'Diag'  'min': 0.05685163850486431}
minimum value of actDCF {'cn': 32, 'covType': 'Diag'  'min': 0.05694764464925755}
```

BEST GMM

minDCF vs Log Odds with nComponents = 1 and covType Diag

minDCF vs Log Odds with nComponents = 2 and covType Diag

minDCF vs Log Odds with nComponents = 4 and covType Diag

minDCF vs Log Odds with nComponents = 8 and covType Diag

minDCF vs Log Odds with nComponents = 16 and covType Diag

minDCF vs Log Odds with nComponents = 32 and covType Diag

DEVI SCEGLIERE I MIGLIORI RESULTATI TRA:

{
- GMM
- LdR
- SVM

DOVRESTI PRENDERE LA MIGLIOR CONFIGURAZIONE DI TUTTI E PRENDERE QUELLE
E DISEGNARE IL BAYES ERROR

BEST LdR

```
Best minDCF {'lambda': 0.1, 'weight': False, 'min': 0.24660778289810548, 'expanded': True, 'PCA': -1, 'Z-norm': False}
Best actDCF  {'lambda': 0.0001, 'weight': False, 'min': 0.2616327444956477, 'expanded': False, 'PCA': -1, 'Z-norm': False}
```

BEST SVM

```
Best minDCF {'C': 1.0, 'kernel': True, 'type': 'RBF', 'min': 0.18453981054787505, 'gamma': 0.1}
Best actDCF  {'C': 0, 'kernel': False, 'type': 'min': 0.35813492063492064, 'gamma': 20}
```