# Probabilistic Graphical Models - Homework 1

Maxence Gélard

November 19th 2021

## 1 Linear classification

### 1.1 Generative Model (LDA)

**Question a.**

Let $\{(x_i, y_i)\}_{i\in 1...n}$ be *i.i.d* observations samples from the given distribution:

$$y \sim \mathrm{B}(\pi), x|y = i \sim \mathcal{N}(\mu_i, \Sigma) \tag{1}$$

We would like to estimate the parameters $\pi$, $\mu = (\mu_0, \mu_1)$ and $\Sigma$ using **Maximum Likelihood estimation** (MLE). The likelihood writes:

$$\begin{aligned}
L_{(x_i,y_i)}(\pi, \mu, \Sigma) \equiv L_{X,Y}(\pi, \mu, \Sigma) &= \prod_{i=1}^{n} p(x_i, y_i|\pi, \mu, \Sigma) \\
&= \prod_{i=1}^{n} p(y_i|\pi, \mu, \Sigma)p(x_i|y_i, \pi, \mu, \Sigma) \\
&= \prod_{i=1}^{n} p(y_i|\pi)p(x_i|y_i, \mu, \Sigma) \\
&= \prod_{i=1}^{n} \pi^{y_i}(1-\pi)^{1-y_i} \frac{1}{\sqrt{2\pi det(\Sigma)}} exp(-\frac{1}{2}(x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i}))
\end{aligned}$$

Thus the log-likelihood can be written:

$$\begin{aligned}
l_{(x_i,y_i)}(\pi, \mu, \Sigma) &\equiv log(L_{X,Y}(\pi, \mu, \Sigma)) \\
&= \sum_{i=1}^{n} y_i log(\pi) + (1-y_i)log(1-\pi) - \frac{1}{2}log(2\pi) - \frac{1}{2}log(det(\Sigma)) - \frac{1}{2}(x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i})
\end{aligned}$$

In order to derive the MLE estimators for $\pi$, $\mu$ and $\Sigma$, we aims at maximizing $l_{X,Y}$. The existence of unique maximizers is guaranteed as $l_{X,Y}$ is concave, so we can derive estimators by considering the critical point of $l_{X,Y}$:

**Estimator for $\pi$**

$$\begin{aligned}
\frac{\partial l_{X,Y}}{\partial \pi} &= \sum_{i=1}^{n} \frac{y_i}{\pi} - \frac{1-y_i}{1-\pi} \\
&= \sum_{i=1}^{n} \frac{y_i - \pi}{\pi(1-\pi)} \\
&= \frac{1}{\pi(1-\pi)} \left( \left( \sum_{i=1}^{n} y_i \right) - n\pi \right)
\end{aligned}$$

So, $\frac{\partial l_{X,Y}}{\partial \pi} = 0$ brings:

$$\boxed{\hat{\pi} = \frac{1}{n}\sum_{i=1}^{n} y_i} \tag{2}$$

**Estimator for $\mu$**

Let $j \in \{0, 1\}$ and let's note $N_j = \sum_{i=1}^{n} \mathbb{1}_{\{y_i=j\}}$.

$$\begin{aligned}
\frac{\partial l_{X,Y}}{\partial \mu_j} &= \frac{\partial}{\partial \mu_j}\left(-\frac{1}{2}\sum_{i=1}^{n}(x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i})\right) \\
&= \frac{\partial}{\partial \mu_j}\left(-\frac{1}{2}\sum_{i=1}^{n} x_i^T\Sigma^{-1}x_i - 2\mu_{y_i}^T\Sigma^{-1}x_i + \mu_{y_i}^T\Sigma^{-1}\mu_{y_i}\right) \\
&= \sum_{i=1}^{n}\Sigma^{-1}x_i - \Sigma^{-1}\mu_{y_i}
\end{aligned}$$

So, $\frac{\partial l_{X,Y}}{\partial \mu_j} = 0$ brings:

$$\boxed{\hat{\mu_j} = \frac{1}{N_j}\sum_{\substack{i=1 \\ y_i=j}}^{n} x_i} \tag{3}$$

**Estimator for $\Sigma$** We have:

$$l_{X,Y} = C - \frac{n}{2}log(det(\Sigma)) - \frac{1}{2}\sum_{i=1}^{n}(x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i})$$

With $C$ a depending on $\pi$ so a constant in $\Sigma$ and $\mu$. We can use the "trace trick":

$$\begin{aligned}
-\frac{1}{2}\sum_{i=1}^{n}(x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i}) &= -\frac{1}{2}\sum_{i=1}^{n} Tr\left((x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i})\right) \\
&= -\frac{1}{2}Tr\left(\sum_{i=1}^{n}(x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i})\right) \\
&= -\frac{n}{2}Tr\left(\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{y_i})(x_i - \mu_{y_i})^T\right)\Sigma^{-1}\right)
\end{aligned}$$

Thus, given that $\frac{\partial log(det(\Sigma))}{\partial \Sigma^{-1}} = -\frac{\partial log(det(\Sigma^{-1}))}{\partial \Sigma^{-1}} = -(\Sigma^{-1})^{-T} = -\Sigma$, we have:

$$\frac{\partial l_{X,Y}}{\partial \Sigma^{-1}} = \frac{n}{2}\Sigma - \frac{n}{2}\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{y_i})(x_i - \mu_{y_i})^T$$

Finally, $\frac{\partial l_{X,Y}}{\partial \Sigma^{-1}} = 0$ brings:

$$\boxed{\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{y_i})(x_i - \mu_{y_i})^T} \tag{4}$$

**Question b.**

Let's derive the form of the conditional distribution $p(y=1|x)$:

$$\begin{aligned} p(y=1|x) &= \frac{p(y=1,x)}{p(x)} \\ &= \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1)+p(x|y=0)p(y=0)} \ (Baye's\ rule) \\ &= \frac{1}{1+\frac{p(x|y=0)p(y=0)}{p(x|y=1)p(y=1)}} \end{aligned}$$

We have: $\frac{p(y=0)}{p(y=1)} = \frac{1-\pi}{\pi}$.
And:

$$\begin{aligned} p(x|y=0) &= \frac{1}{\sqrt{2\pi det(\Sigma)}} exp(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)) \\ p(x|y=1) &= \frac{1}{\sqrt{2\pi det(\Sigma)}} exp(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)) \\ \frac{p(x|y=0)}{p(x|y=1)} &= exp\left(-\frac{1}{2}\left(-2\mu_0^T\Sigma^{-1}x + 2\mu_1^T\Sigma^{-1}x + \mu_0\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1\right)\right) \\ &= exp\left((\Sigma^{-1}(\mu_1-\mu_0))^T + \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0)\right) \end{aligned}$$

Let's set: $w = \Sigma^{-1}(\mu_1-\mu_0)$ and $b = \frac{1}{2}(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1)$, and we get:

$$\boxed{p(y=1|x) = \frac{1}{1+\frac{1-\pi}{\pi}exp^{-w^Tx+b}}} \tag{5}$$

For the logistic regression, we had: $p(y=1|x) = \frac{1}{1+exp^{-w^Tx+b}}$, so we would get the same form with LDA if $\pi = 0.5$, which means that classes would be balanced.

**Question c.**

The implementation of the MLE estimators for LDA is given with the attached notebook (section 1.1), in a class called *LinearDiscriminantAnalysis*. Below are given the results of the LDA models on the 3 provided train sets (*trainA, trainB, trainC*). Figure 1 provided the decision boundary for the classification problem (*i.e* the line corresponding to $p(y=1|x) = 0.5$), while Figure 2 adds the level contours for $p(y=1|x)$.

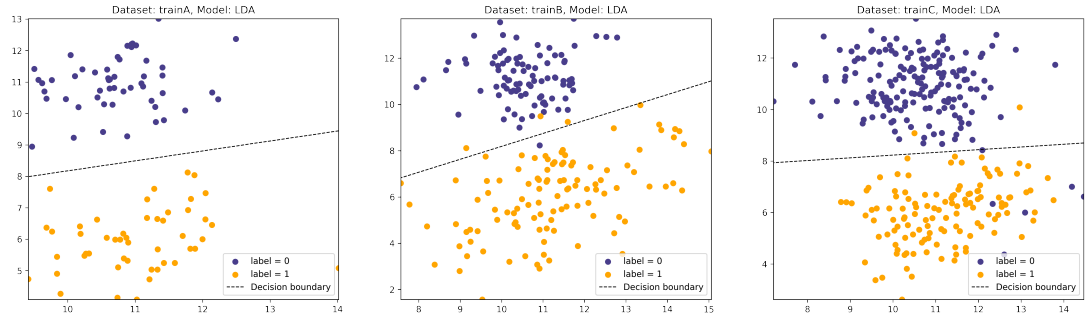


Figure 1: LDA model on the 3 train sets - with decision boundary

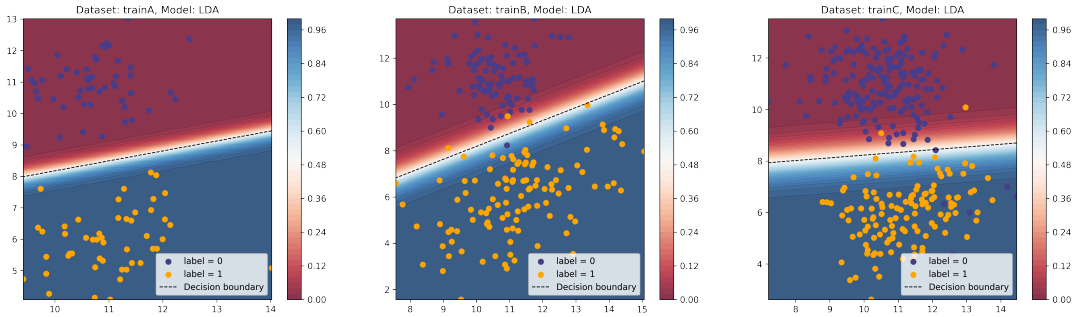


Figure 2: LDA model on the 3 train sets - with posterior probability levels

In addition, we provide the contour levels for the estimation of the two normal distributions, respectively $\mathcal{N}(\mu_0, \Sigma)$ and $\mathcal{N}(\mu_1, \Sigma)$.

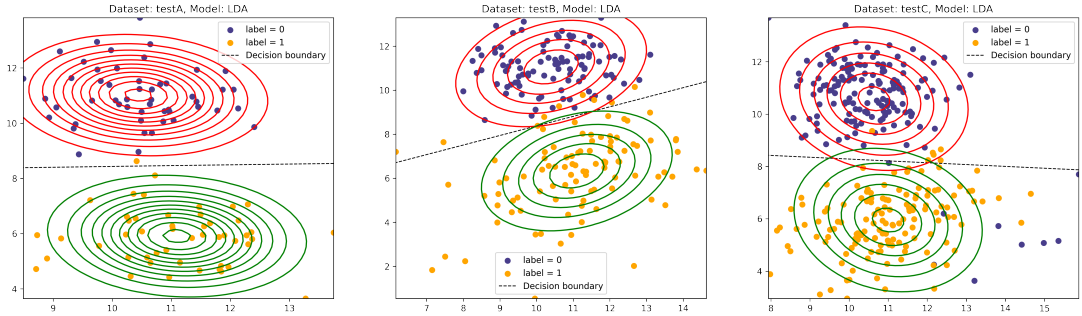


Figure 3: LDA model on the 3 train sets - Gaussian estimation

One can state that the assumption of having the same covariance matrix for the two class tends to hold for the dataset *trainA*. Indeed, for this dataset, we get the following parameters:

$$\Sigma = \begin{pmatrix} 0.59 & 0.14 \\ 0.14 & 0.82 \end{pmatrix}$$
$$\mu_0 = \begin{pmatrix} 10.7 \\ 10.9 \end{pmatrix}$$
$$\mu_1 = \begin{pmatrix} 11.0 \\ 6.0 \end{pmatrix}$$

If we estimate the covariance matrices separately for points whose labels are 0 and for points whose labels are 1, we get:

$$\Sigma_0 = \begin{pmatrix} 0.46 & 0.10 \\ 0.10 & 0.71 \end{pmatrix}$$
$$\Sigma_1 = \begin{pmatrix} 0.72 & 0.18 \\ 0.18 & 0.93 \end{pmatrix}$$

So we can see that the condition $\Sigma = \Sigma_0 = \Sigma_1$ approximately holds. However, looking at the results for the dataset *trainC*, one gets:

$$\Sigma = \begin{pmatrix} 1.28 & -0.06 \\ -0.06 & 1.67 \end{pmatrix}$$
$$\mu_0 = \begin{pmatrix} 10.6 \\ 10.8 \end{pmatrix}$$
$$\mu_1 = \begin{pmatrix} 11.2 \\ 6.0 \end{pmatrix}$$

And for the covariance matrices estimated separately among classes:

$$\Sigma_0 = \begin{pmatrix} 1.3 & -0.43 \\ -0.43 & 1.8 \end{pmatrix}$$
$$\Sigma_1 = \begin{pmatrix} 1.3 & 0.46 \\ 0.46 & 1.44 \end{pmatrix}$$

In this case, the assumption of having the same covariance matrix doesn't hold anymore with especially the covariance between the two features that are opposite of each other in class 0 and in class 1.

## 1.2 Logistic Regression

The logistic regression model is the following:

$$y_i|x_i, \beta \sim \mathrm{B}(g(x_i^T\beta)) \tag{6}$$

with:

$$\forall t \in \mathbb{R}, g(t) = \frac{1}{1+e^{-t}}$$

The logistic regression algorithm has been implemented using a fixed step gradient descent method. To do so, we have been minimizing the negative log-likelihood:

$$-L_{(x_i,y_i)}(\beta) = -\frac{1}{n}\sum_{i=1}^{n} y_i log(g(x_i^T\beta)) + (1-y_i)log(1-g(x_i^T\beta)) \tag{7}$$

In the provided *Python* implementation, we have been using the following gradient formula (given $X$ the features matrix and $y$ the vector of labels, and $L$ the loss, i.e the negative log-likelihood):

$$\nabla L(\beta) = \frac{1}{n}X^T(g(X\beta) - y) \tag{8}$$

Below is given the parameters of the logistic regression learnt on the three train sets, as well as the plot of decision boundaries and probability levels.

| Dataset | $w_0$ | $w_1$ | $w_2$ |
|---|---|---|---|
| Train A | 0.014 | 0.50 | -0.67 |
| Train B | 0.28 | 0.59 | -0.77 |
| Train C | -0.038 | 0.49 | -0.68 |

In order to plot the line $p(y=1|x) = 0.5$, we could plot the line $x^T\beta = 0$ (because $g(0) = \frac{1}{2}$), with $\beta = (w_0, w_1, w_2)$, *i.e* the line defined by the equation $y = -\frac{1}{w_2}(w_0 + w_1 x)$. A practical way to deal with boundary lines is *Python* is to use the *contour* function that would plot the line defining $p(y=1|x) = 0.5$ using a two dimensional grid. This will also be done for linear regression in the next section.

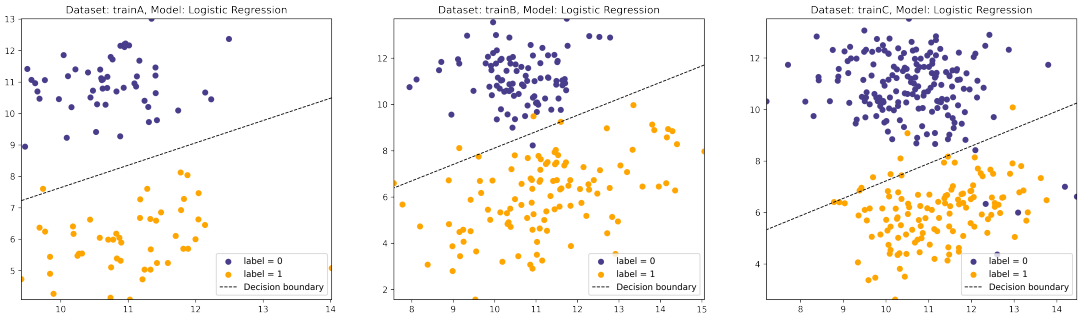


Figure 4: Logistic regression model on the 3 train sets - with decision boundary

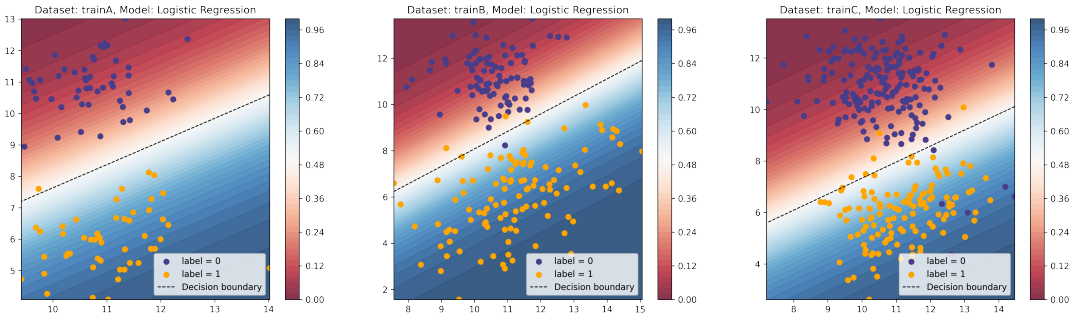


Figure 5: Logistic regression model on the 3 train sets - with posterior probability levels

In addition to this implementation, as discussed during the lecture, it may be interesting to implement the logistic regression using the Iterative Reweighted Least Square scheme in order to avoid over-fitting for data that are linearly separable. In the provided *Python* code, we give an implementation of the IRLS algorithm for the logistic regression.

At iteration $t$ of the IRLS algorithm, we have the following coefficient update:

$$\beta^{t+1} = \beta^t - H^{-1}\nabla L(\beta) \tag{9}$$

where H correspond to the following Hessian matrix: $H = -X^T P X$, with $P$ the diagonal matrix whose diagonal elements are $\sigma(X_i\beta)(1-\sigma(X_i\beta))$, $\sigma$ representing the sigmoid function.

Below is given the parameters of the logistic regression (IRLS algorithm) learnt on the three train sets, as well as the plot of decision boundaries and probability levels.

| Dataset | $w_0$ | $w_1$ | $w_2$ |
|---|---|---|---|
| Train A | 4.77 | 0.36 | -1.03 |
| Train B | 1.32 | 0.26 | -0.47 |
| Train C | 2.55 | 0.03 | -0.35 |

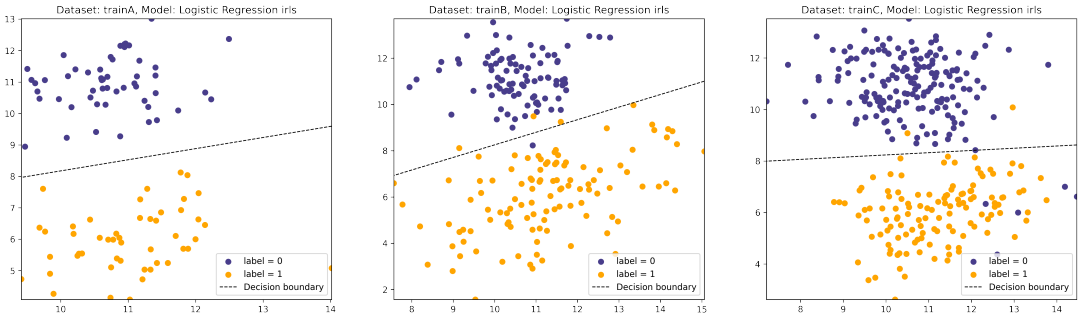


Figure 6: Logistic regression model (IRLS) on the 3 train sets - with decision boundary

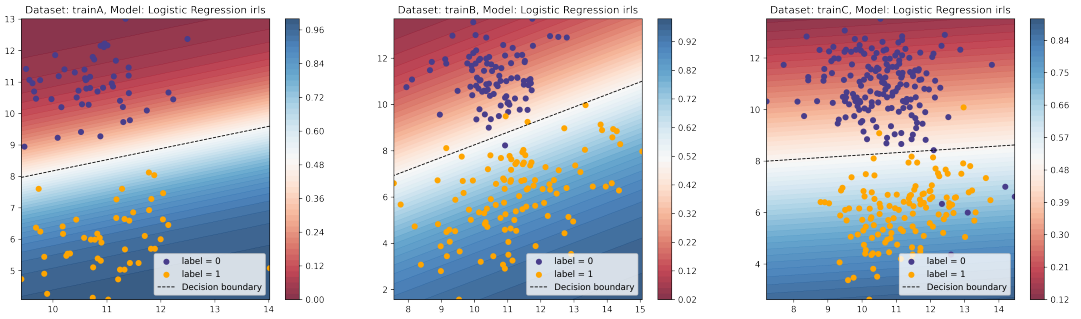


Figure 7: Logistic regression model (IRLS) on the 3 train sets - with posterior probability levels

## 1.3 Linear Regression

The linear regression model is defined as follows:

$$y_i = x_i^T \beta + \epsilon_i, \;\; \epsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{10}$$

In the *Python* implementation, we have been using the normal equation to learn the parameters:

$$\beta = (X^T X)^{-1} X^T y \tag{11}$$

Below is given the parameters of the linear regression learnt on the three train sets, as well as the plot of decision boundaries and probability levels.

| Dataset | $w_0$ | $w_1$ | $w_2$ |
|---|---|---|---|
| Train A | 1.38 | 0.056 | -0.18 |
| Train B | 0.88 | 0.083 | -0.15 |
| Train C | 1.64 | 0.017 | -0.16 |

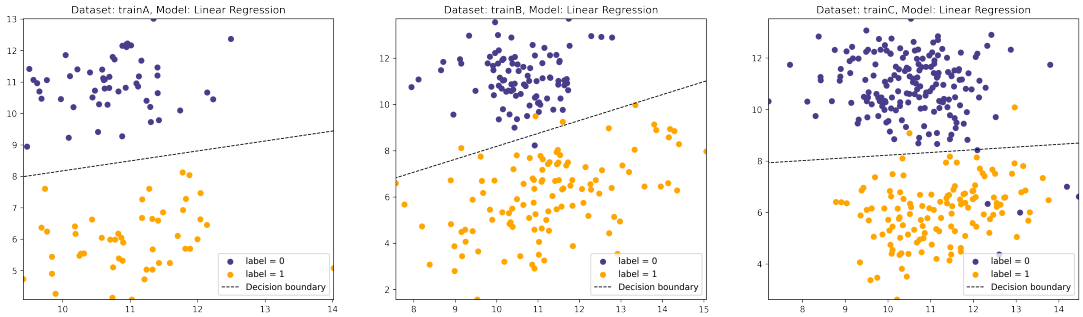


Figure 8: Linear regression model on the 3 train sets - with decision boundary

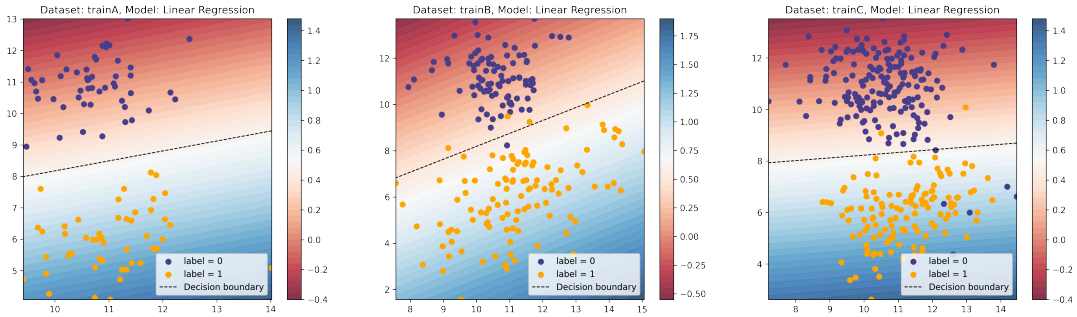


Figure 9: Linear regression model on the 3 train sets - with posterior probability levels

## 1.4 Application

**Question a.**

Below is given the misclassification error (fraction of the misclassified data) on the training data on the test data for the different models

| Dataset A | LDA | Logistic Regression | Logistic Regression (IRLS) | Linear Regression |
|---|---|---|---|---|
| Train | 0.0% | 0.01% | 0.0% | 0.0% |
| Test | 0.01% | 0.01% | 0.01% | 0.01% |

| Dataset B | LDA | Logistic Regression | Logistic Regression (IRLS) | Linear Regression |
|---|---|---|---|---|
| Train | 0.02% | 0.02% | 0.02% | 0.02% |
| Test | 0.045% | 0.045% | 0.045% | 0.045% |

| Dataset C | LDA | Logistic Regression | Logistic Regression (IRLS) | Linear Regression |
|---|---|---|---|---|
| Train | 0.03% | 0.04% | 0.023% | 0.03% |
| Test | 0.04% | 0.046% | 0.043% | 0.04% |

**Question b.**

For all datasets, the training error is less than the test error, which is expected as we learnt parameters based on minimizing a loss function (or equivalently maximizing the log-likelihood) on the training set. So the increase in miss-classification error on the test set only reflects the fact that a model can't generalize perfectly to any new data. However, one can states that the test error is really close to the train set, which confirms the fact that the test sets have been drawn from the same distribution as the train sets.

In order to provide further interpretation, the comparison of the decision boundaries for the four models (on the three datasets) is given below:

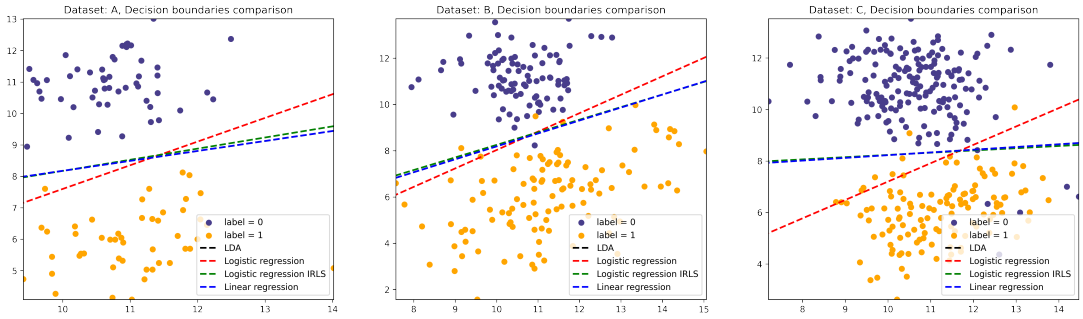


Figure 10: Decision boundaries comparison

First thing we may notice is that the decision boundaries for Linear regression, LDA are the same. This similarity in results is confirmed by the miss-classification errors which are always the same for these two models. This result may come from the fact that behind the two models, there is a normal distribution assumption which is made, even if not expressed in the same way (in the linear regression model, the normal assumption is made on the labels $y \sim \mathcal{N}(\beta^T x, \sigma^2)$, so if we added a prior on $x$, this model could be turned into a generative model as LDA, but with another distribution on $y$, i.e we would no longer have $y \sim \mathrm{B}(\pi)$). Note that this only hold because we did a linear regression with the assumption that our "real" labels were finally binary labels (0 or 1).

However, even if the miss-classification error are close, one could tend to prefer LDA by looking at the probability levels. Indeed, in the case of LDA, the variation of probability levels is sharper than for linear regression, meaning that it succeeded in making a better and more sure separation of the data into two classes.

Concerning LDA, looking at the miss-classification errors, we get back the fact that the results are better on dataset A than on dataset B and C, that we explained earlier by the fact that the assumption of constant covariance matrix among labels didn't hold for the two latter datasets.

Beyond these two models, we have the Logistic Regression with IRLS that gives a decision boundary really close to the LDA's and Linear Regression's ones, with also very close results in terms of miss-classification errors. We can notice a better train miss-classification error on dataset C (only 0.023%), but with the same test error as the two other models, which may reflects once again the fact that it is easy for logistic regression to over-fit the data.

In general, the logistic regression (classic gradient descent) leads to worse results than LDA / Linear regression (or the IRLS version of Logistic Regression). This can be explained by the choice of the method, which was a simple fixed step gradient descent. As we saw in lectures, with this algorithm, nothing prevent the coefficient vector $\beta$ to have higher norm than for other models, which leads to overfitting / worse generalization capacity. We tried to limit this behaviour using a maximum number of iteration, but this can be seen for example on the parameter $w_1$ where the parameter is one order of magnitude higher in Logistic Regression compared to Linear Regression for example. Moreover, the difference in results may also come from the fact that in Logistic Regression, we do not make any assumption on the data distribution. Also, in the first section, we noticed that the decision boundary expression of LDA gets back to the Logistic Regression one if $\pi = 0.5$, with $\pi$ corresponding to the proportion of data points in class 1. For the different datasets, one gets for LDA: $\pi_A = 0.48, \pi_B = 0.55, \pi_C = 0.42$. So the classes are not perfectly balanced, which can provide an additional interpretation on the difference of results between LDA and Logistic Regression.

# 2 Gaussian mixture models and EM

## 2.1 Math

We consider the following Gaussian mixture model (GMM) for multivariate data:

$$X_0 \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k)$$

with $X_0 \in \mathbb{R}^p$, $\sum_{k=1}^{K} \pi_k = 1$, $\pi_k \in [0, 1]$.

Let's introduce the latent variables $Z$, such that for $i \in \{1, ..., n\}$, $Z_i \sim \mathcal{M}(1, \pi)$ and $X_i | Z_{i,k} \sim \mathcal{N}(\mu_k, \Sigma_k)$. (We

thus have $\sum_{k=1}^{K} Z_{i,k} = 1$). Let's also introduce $\tau_{i,k}$ being the probability for observation $i$ to belong to cluster $k$ (given $x_i$ and the value of the model parameters $\theta = (\mu_k, \Sigma_k)_{k\in\{1,...,K\}}$):

$$\tau_{i,k} = \frac{\pi_k \mathcal{N}(x_i; \mu_k, \Sigma_k)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(x_i; \mu_l, \Sigma_l)}$$

The Expectation Maximization algorithm, used to estimate the model's parameters, writes as follow:

- **Initialization**: Select $K$ and initialize the clusters (randomly or using K-Means for example).
- **M-Step**: Given the $\tau_{i,k}$, we can compute new estimate of models parameters:

$$\hat{\pi_k} = \frac{1}{n} \sum_{i=1}^{n} \tau_{i,k}$$
$$\hat{\mu_k} = \frac{1}{n_k} \sum_{i=1}^{n} \tau_{i,k} x_i$$
$$\hat{\Sigma_k} = \frac{1}{n_k} \sum_{i=1}^{n} \tau_{i,k} (x_i - \hat{\mu_k})(x_i - \hat{\mu_k})^T$$

  with $n_k$ being the noramlized factor corresponding to $\sum_{i=1}^{n} \tau_{i,k}$.

- **E-Step**: Compute the $\tau_{i,k}$ using the new estimates of the model's parameters.
- **Loop**: Get back to the M-step if there was no significant change in the log-likelihood.

**Proves of updates** We first need two results:

- We saw during the lecture that the $Z_i$ are independent given the observations, based on Baye's rule: $p((z_i)|(x_i), \pi, \theta) = \frac{p((x_i,z_i)|\pi,\theta)}{p(x_i|\pi,\theta)}$, with the numerator and the denominator being expressed as a product over all the samples.
- Let $i \in \{1, ..., n\}$.

$$\begin{aligned}
p(z_i|x_i, \pi, \theta) &= \frac{p(x_i, z_i|\pi, \theta)}{p(x_i|\pi, \theta)} \\
&= \frac{p(x_i|z_i, \pi, \theta) p(z_i|\pi)}{p(x_i|\pi, \theta)} \\
&= \frac{\prod_{k=1}^{K} (\pi_k \mathcal{N}(x_i; \mu_k, \Sigma_k))^{z_{i,k}}}{\sum_{l=1}^{K} \pi_l \mathcal{N}(x_i; \mu_l, \Sigma_l)}, \ as\ Z_i \sim \mathcal{M}(1, \pi) \\
&= \prod_{k=1}^{K} \left( \frac{\pi_k \mathcal{N}(x_i; \mu_k, \Sigma_k)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(x_i; \mu_l, \Sigma_l)} \right)^{z_{i,k}}, \ because\ only\ one\ z_{i,k} = 1 \\
&= \prod_{k=1}^{K} \tau_{i,k}^{z_{i,k}}.
\end{aligned}$$

Let's now assume that we have computed all the $\tau_{i,k}$ and let's justify the M-step update formulas. In the GMM model, the $Z_i$'s are unknown, so we saw in class that we aim at maximizing the expectation of the complete log-likelihood (under the distribution of latent variables) with respect to the GMM parameter $\theta$:

$$E \equiv E_{(Z_i)}[L_{(x_i,Z_i)}(\pi,\theta)] = \sum_{i=1}^{N}\sum_{k=1}^{K}\tau_{i,k}log(\pi_k\mathcal{N}(x_i;\mu_k,\Sigma_k))$$
$$= \sum_{i=1}^{N}\sum_{k=1}^{K}\tau_{i,k}(log(\pi_k) + log(\frac{1}{(2\pi)^p det(\Sigma_k)^{1/2}}) - \frac{1}{2}(x_i-\mu_k)^T\Sigma_k^{-1}(x_i-\mu_k))$$

In order to estimate $\pi$, we need to maximize the concave function $\sum_{i=1}^{N}\sum_{k=1}^{K}\tau_{i,k}(log(\pi_k))$ (or minimize the opposite of it), under the constraint $\sum_{k=1}^{K}\pi_k = 1$. That corresponds to find a saddle point of the following Lagragian:

$$\mathcal{L}(\pi,\nu) = -\sum_{i=1}^{N}\sum_{k=1}^{K}\tau_{i,k}(log(\pi_k)) + \nu(\sum_{k=1}^{K}\pi_k - 1)$$

$\frac{\partial\mathcal{L}}{\partial\pi_k} = 0$ leads to $\pi_k = \nu\sum_{k=1}^{K}\tau_{i,k}$. And using the constraint on $\pi$, we get $\nu = \frac{1}{n}$, which gives update formula given above $\hat{\pi_k}$. The parameters for the gaussian distributions ($\mu_k$ and $\Sigma_k$) correspond to the same MLE estimator we did for the LDA, for which we gave the complete derivation, except that we have a factor $\tau_{i,k}$ in the sum for the estimators. Indeed:

$\frac{\partial E}{\partial\mu_k} = 0$ leads to, $\sum_{i=1}^{N}\tau_{i,k}\Sigma_k^{-1}(x_i-\hat{\mu_k})$, thus, $\hat{\mu_k} = \frac{1}{n_k}\sum_{i=1}^{n}\tau_{i,k}x_i$ (with previous definition of $n_k$).

Similarly, using the derivative of the logarithm of the determinant as for LDA, $\frac{\partial E}{\partial\Sigma_k^{-1}} = 0$ leads to:

$$\sum_{i=1}^{N}\frac{1}{2}\tau_{i,k}(-\Sigma_k + (x_i-\mu_k)(x_i-\mu_k)^T)$$

Thus, $\hat{\Sigma_k} = \frac{1}{n_k}\sum_{i=1}^{n}\tau_{i,k}(x_i-\hat{\mu_k})(x_i-\hat{\mu_k})^T$.

Let's now justify the E-step. We now fix the model's parameters $\theta$ and we saw in class that the E-step correspond to maximize $E$ with respect to $r((z_i))$ with any probability law $r$. We also proved that this is equivalent to minimize the following Kullback-Leibler divergence:

$$KL = -\sum_{(z_i)} r(z_i)log(\frac{p(z_i|x_i,\pi,\theta)}{r(z_i)})$$

The best way to minimize this divergence is to set it to zero, which can be made by choosing $r(z_i) = p(z_i|x_i,\pi,\theta)$, which is equivalent to compute the $\tau_{i,k}$ as we saw above: $p(z_i|x_i,\pi,\theta) = \prod_{k=1}^{K}\tau_{i,k}^{z_{i,k}}$.

## 2.2 Implementation

The *Python* implementation of the EM algorithm is provided in the notebook.

Before going to the application using the *Decathlon* dataset, it is worth making a sanity check and verify the behaviour of our EM algorithm on the previous datasets, looking for $K = 2$ clusters. Below is given the results of the EM algorithm on the three datasets:

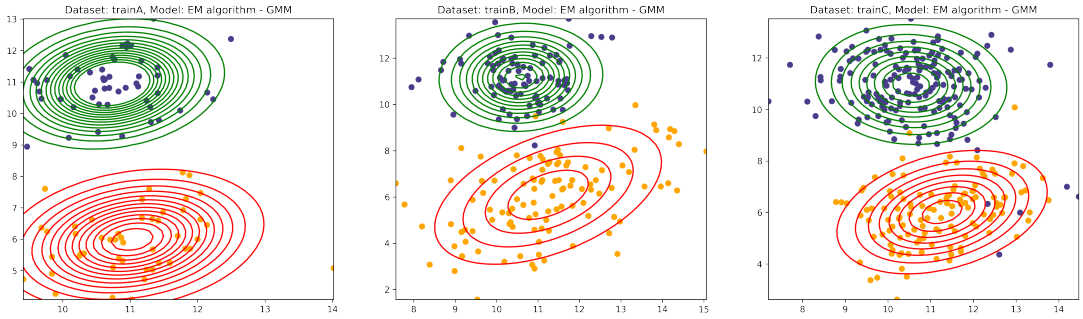


Figure 11: EM algorithm (GMM) on the three train datasets - K = 2

Looking at the results, we can say that the algorithm succeeded in identifying correctly the 2 classes we had in each dataset. Moreover, one can now go back to the observation we made in the LDA section: while gaussian distributions are quite similar in dataset A, we finally find that in datasets B and C, the two classes have different covariance matrices.

## 2.3 Application

We ran an Gaussian Mixture Models using EM algorithm with $K = 3$ clusters on the decathlon dataset. This dataset is composed of 41 data points, with 10 features for each, corresponding to the results of the athletes to the different parts of a decathlon.

We first run the EM algorithm using all the features, which leads to build 3 groups:

- Group 1: YURKOV, McMULLEN, Clay, Zsivoczk, Nool, Barras, Smith, Qi, Drew, Karlivans
- Group 2: BERNARD, ZSIVOCZKY, MARTINEAU, BOURGUIGNON, Sebrle, Karpov, Hernu, Bernard, Schoenbeck', Ojaniemi, Smirnov, Parkhomenko, Terek, Turi, Lorenzo, Korkizoglou, Uldal, Casarsa
- Group 3: SEBRLE, CLAY, KARPOV, WARNERS, HERNU, BARRAS, NOOL, Macey, Warners, Schwarzl, Pogorelov, Averyanov, Gomez

Even if it looks like that some athletes of the same nationality may have been grouped together, it is not totally true across all the dataset. So it is worth looking at the characteristics of each group, looking at the mean of each features for the different groups:

| Groups | Group 1 | Group 2 | Group 3 |
|---|---|---|---|
| 100m | 11.0 | 11.0 | 11.0 |
| Long.jump | 7.2 | 7.26 | 7.3 |
| Shot.put | 14.4 | 14.7 | 14.3 |
| High.jump | 2.0 | 1.98 | 1.96 |
| 400m | 49.8 | 49.6 | 49.5 |
| 110m.hurdle | 14.4 | 14.5 | 14.8 |
| Discus | 44.7 | 44.8 | 43.7 |
| Pole.vault | 4.70 | 4.71 | 4.86 |
| Javeline | 59.8 | 58.0 | 58.0 |
| 1500M | 276.1 | 278.0 | 281.5 |

Analyzing these results, we can see that for a large number of features, these groups are homogeneous. Nevertheless, we can identify some characteristics in each group, with for example athletes from group 3 that tends to perform better in 1500m than others, and athletes in group 1 being better at Javeline. We also have to mention that running the EM algorithm on this 10 features doesn't always lead to the exact same results, showing that it may be sensible to initialization.

To finish this analysis, we tried to only look at two features, here "1500m" and "Javelin" which have the highest standard deviations (respectively 120 and 20). We then run the GMM model with $K = 3$ clusters and we get the following normal distributions:

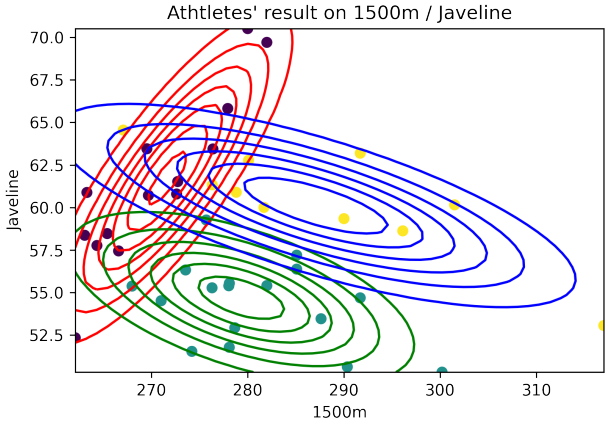


Figure 12: EM algorithm (GMM) on decathlon dataset (2 features) - K = 3

We get the following features' mean across the new groups:

| Groups | Group 1 | Group 2 | Group 3 |
|---|---|---|---|
| Javeline | 61.5 | 54.5 | 60.4 |
| 1500M | 270.4 | 280.8 | 288.0 |

Therefore, we have identified three clusters using the features "Javeline" and "1500m". Given the fact that the features values are close to each other, it is not surprising that we get very close cluster, but we can still say that EM algorithm succeeded in identifying 3 distinct clusters using these 2 features.

Finally, to conclude this report, it is worth comparing these results of the EM algorithm using two features with the results of a KMeans algorithm (with 3 clusters):

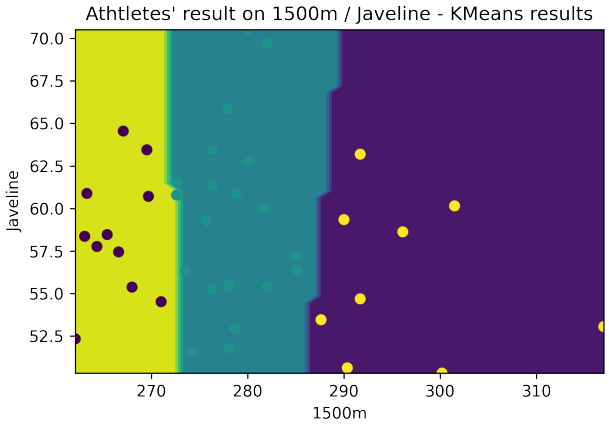


Figure 13: KMeans algorithm on decathlon dataset (2 features) - K = 3 (with clusters' boundaries)

Looking at the plot above, we understand why the Gaussian Mixture Models, inferred using the EM algorithm is interesting, as it allows to derive probabilities distributions. On the other hand, KMeans provides exclusive (i.e non-overlapping clusters), which in the end may lead to have less information on the clustering results.