Master M2 MVA 2015/2016 - Graphical models - HWK 1

Author: Lilian Besson (`lilian.besson at ens-cachan.fr`).

Ex.1 : Learning in discrete graphical models

Let Z and X be discrete random variables taking respectively M and K different values with $\mathbb{P}(Z = m) = \pi_m$ and $\mathbb{P}(X = k|Z = m) = \theta_{m,k}$.

If we have n i.i.d. observations for X: $(x_i)_{i=1\dots n}$ and n i.i.d. observations for Z: $(z_i)_{i=1\dots n}$, we can study the log-likelihood for a vector of parameters $(\boldsymbol{\pi}, \boldsymbol{\theta})$. Maximizing this log-likelihood will give two estimators $\hat{\boldsymbol{\pi}}^{(n)}_{\mathrm{ML}}$ and $\hat{\boldsymbol{\theta}}^{(n)}_{\mathrm{ML}}$.

MLE for Z: $\hat{\boldsymbol{\pi}}^{(n)}_{\mathrm{ML}}$

As Z is independent of X, we will first estimate π, and then θ:

$$\begin{aligned} l(\boldsymbol{\pi}) = \log\mathcal{L}(\boldsymbol{\pi}) &= \sum_{i=1}^{n} \log\left(\mathbb{P}_{\boldsymbol{\pi}}(Z = z_i)\right) \\ &= \sum_{i=1}^{n} \log\left(\prod_{k=1}^{K} \pi_k^{z_{i,k}}\right) = \sum_{i=1}^{n}\sum_{k=1}^{K} z_{i,k}\log\pi_k \\ &= \sum_{k=1}^{K}\left(\sum_{i=1}^{n} z_{i,k}\right)\log\pi_k = \sum_{k=1}^{K} n_k \log\pi_k \end{aligned}$$

With the usual convention for indicator variables encoding multinomial variables, and the number of time we observed the k^{th} class for Z is written $n_k \overset{\text{def}}{=} \sum_{i=1}^{n} z_{i,k}$ (for $k = 1 \dots K$).

We look for the *maximum* value of this quantity $l(\boldsymbol{\pi})$ (with the constraints that $\sum_{k=1}^{K} \pi_k = 1$, and the implicit constraints that $\pi_k \geqslant 0$[1]). With the convex analysis point of view, we need to *minimize* the (convex) function $f(\boldsymbol{\pi}) = -l(\boldsymbol{\pi}) = -\sum_{k=1}^{K} n_k \log\pi_k$ subject to $\mathbf{1}^T.\boldsymbol{\pi} = 1$. So the **Lagrangian** of this problem is:

$$L(\boldsymbol{\pi}, \lambda) = -\sum_{k=1}^{K} n_k \log\pi_k + \lambda\left(\sum_{k=1}^{K}\pi_k - 1\right)$$

This convex problem satisfies Slater's condition (e.g. the vector $\boldsymbol{\pi} = \frac{1}{K}\mathbf{1}$ is in the interior of the constrained domain), so **we have strong duality** here, and maximizing the dual will give the optimal solution of the primal. As the Lagrangian is convex w.r.t. $\boldsymbol{\pi}$, canceling its derivatives will be enough (for the values $\pi_k \neq 0$):

$$\frac{\partial L(\boldsymbol{\pi}, \lambda)}{\partial \pi_k} = -\frac{n_k}{\pi_k} + \lambda = 0 \implies \pi_k = \frac{n_k}{\lambda}$$

The constraint $\mathbf{1}^T.\boldsymbol{\pi} = 1$ gives $\sum_{k=1}^{K} n_k = \lambda$ so $\lambda = K$ the total number of observed values of Z.

Hence $\hat{\pi}^{(n)}_k = \frac{n_k}{K}$ is the max-log-likelihood estimator (MLE)[2] for $\boldsymbol{\pi}$, based on the p observations z_k (with $n_k \overset{\text{def}}{=} \sum_{i=1}^{n} z_{i,k}$) following a multinomial distribution.

[1] Indeed $0\log 0 = 0$ by convention, and if $\pi_k = 0$, $n_k = 0$, so π_k can be 0.

[2] Remark: we just proved a classical result.

MLE for X: $\hat{\boldsymbol{\theta}}_{\mathrm{ML}}^{(n)}$

Now let focus on X.

$$\begin{aligned}
l_2(\boldsymbol{\pi}, \boldsymbol{\theta}) = \log \mathcal{L}(\boldsymbol{\pi}, \boldsymbol{\theta}) &= \sum_{i=1}^{n} \log \left(\mathbb{P}_{\boldsymbol{\pi},\boldsymbol{\theta}}(Z = z_i, X = x_i) \right) \\
&= \sum_{i=1}^{n} \log \left(\mathbb{P}_{\boldsymbol{\pi}}(Z = z_i) \mathbb{P}_{\boldsymbol{\theta}}(X = x_i | Z = z_i) \right) \\
&= l_1(\boldsymbol{\pi}) + \sum_{i=1}^{n} \log \left(\prod_{m=1}^{M} \prod_{k=1}^{K} \theta_{m,k}^{x_{i,m} z_{i,k}} \right) \\
&= l_1(\boldsymbol{\pi}) + \sum_{i=1}^{n} \sum_{m=1}^{M} \sum_{k=1}^{K} (x_{i,m} z_{i,k} \log \theta_{m,k}) \\
&= l_1(\boldsymbol{\pi}) + \sum_{m=1}^{M} \sum_{k=1}^{K} \left(\sum_{i=1}^{n} x_{i,m} z_{i,k} \right) \log \theta_{m,k} \\
&= l_1(\boldsymbol{\pi}) + \sum_{m=1}^{M} \sum_{k=1}^{K} n_{m,k} \log \theta_{m,k}
\end{aligned}$$

Where l_1 is the previous log-likelihood (studied before for only Z), and we defined $n_{m,k} \overset{\text{def}}{=} \sum_{i=1}^{n} x_{i,m} z_{i,k}$ the number of times we observed the class (m, k) for the couple (X, Z).

The right part of this log-likelihood is exactly similar to the first one, with a double sum on m and k instead of one sum. With $N' \overset{\text{def}}{=} MK$, we can easily see a bijection between $\{1, \dots, M\} \times \{1, \dots, K\}$ and $\{1, \dots, N'\}$, and then rewrite that big double sum as:

$$\sum_{m=1}^{M} \sum_{k=1}^{K} n_{m,k} \log \theta_{m,k} = \sum_{i'=1}^{N'} n_{i'} \log \theta_{i'}$$

This has the form of the first log-likelihood we studied earlier, with just N' classes instead of N.

So with a similar proof, we will find the ML estimator for $\boldsymbol{\theta}$ to be $\hat{\theta}_{m,k}^{(n)} = \frac{n_{m,k}}{MK}$ the observed frequency of the couple class (m, k).

Conclusion:

Therefore, the maximum likelihood estimators $\hat{\boldsymbol{\pi}}_{\mathrm{ML}}^{(n)}$ and $\hat{\boldsymbol{\theta}}_{\mathrm{ML}}^{(n)}$ for this observation are:

$$\hat{\pi}_k^{(n)} = \frac{n_k}{K}, \text{ and } \hat{\theta}_{m,k}^{(n)} = \frac{n_{m,k}}{MK}$$

Where we defined:

- $n_k \overset{\text{def}}{=} \sum_{i=1}^{n} z_{i,k}$ the number of times we observed the class k for the Z (for $k = 1 \dots K$).
- $n_{m,k} \overset{\text{def}}{=} \sum_{i=1}^{n} x_{i,m} z_{i,k}$ the number of times we observed the class (m, k) for the couple (X, Z) (for $k = 1 \dots K$).

Ex.2 : Linear classification

Note:

The Python program for this problem 2 is included in the `zip` archive, and requires usual scientific Python modules (`numpy`, `matplotlib`, `pandas`) and the machine-learning module `scikit-learn`.

I first implemented what was asked using the linear classification models provided by `scikit-learn` (in order to trust and compare my implementations, the plots and the computed scores); and then I implemented the 4 required classification models "manually"[3].

The first part *2.1* is a long proof, and the rest of this exercise gives details on 4 different classifier algorithms, for which we give different performance measures, the learned parameters, some decision boundary plots etc, and finally a performance comparison (*spoiler: QDA seems to win here*).

2.1 : Generative model (LDA)

(a) Deriving the MLE for this model (formally)

For this model, the parameters are: $\pi \in [0,1]$, $\mu_0 \in \mathbb{R}^2$, $\mu_1 \in \mathbb{R}^2$, and $\Sigma \in S^2_{++} \subset \mathbb{R}^4$.

Let $\boldsymbol{\theta} = (\pi, \mu_0, \mu_1, \Sigma) \in \mathbb{R}^9$, and n the number of i.i.d. observations $(x_i, y_i) \in \mathbb{R}^2 \times \{0,1\}$.

The **likelihood** is $\mathcal{L}_{(x_i,y_i)_{1\leqslant i\leqslant n}}(\boldsymbol{\theta}) = \prod_{i=1}^n \mathbb{P}_{\boldsymbol{\theta}}(X = x_i, Y = y_i)$, because the observations are i.i.d.

By taking the log, we find, as usual, $l(\boldsymbol{\theta}) \overset{\text{def}}{=} \log \mathcal{L}_{(x_i,y_i)}(\boldsymbol{\theta}) = \sum_{i=1}^n \log \mathbb{P}_{\boldsymbol{\theta}}(X = x_i, Y = y_i)$. And $\mathbb{P}_{\boldsymbol{\theta}}(X = x_i, Y = y_i) = \mathbb{P}_{\boldsymbol{\theta}}(X = x_i | Y = y_i)\mathbb{P}_{\boldsymbol{\theta}}(Y = y_i)$ by taking the conditional probability on $Y = y_i$.

By assumptions on the distributions of (X, Y), we know that

- For Y: $\mathbb{P}_{\boldsymbol{\theta}}(Y = y_i) = \pi^{y_i}(1-\pi)^{1-y_i}$ (Bernoulli of parameter π),
- For X when $y_i = 0$: $\mathbb{P}_{\boldsymbol{\theta}}(X = x_i | Y = 0) = \frac{1}{\sqrt{n\det(\Sigma)}} \exp\left(-\frac{1}{2}(x_i - \mu_0)^T \Sigma^{-1}(x_i - \mu_0)\right)$,
- For X when $y_i = 1$: $\mathbb{P}_{\boldsymbol{\theta}}(X = x_i | Y = 1) = \frac{1}{\sqrt{n\det(\Sigma)}} \exp\left(-\frac{1}{2}(x_i - \mu_1)^T \Sigma^{-1}(x_i - \mu_1)\right)$.

Hence:

$$\begin{aligned} l(\boldsymbol{\theta}) = &-\frac{1}{2} n \log n + \frac{1}{2} n \log \det(\Sigma) \\ &+ \sum_{i=1}^n \left(y_i \log(\pi)\right) + \sum_{i=1}^n \left((1-y_i)\log(1-\pi)\right) \\ &- \frac{1}{2} \sum_{i=1}^n \left((x_i - \mu_{y_i})^T \Sigma^{-1}(x_i - \mu_{y_i})\right) \end{aligned}$$

For the part depending on π, $\sum_{i=1}^n \left(y_i \log(\pi)\right) + \sum_{i=1}^n \left((1-y_i)\log(1-\pi)\right) = n_1 \log\left(\frac{\pi}{1-\pi}\right) + n \log(1-\pi)$, where $n_1 \overset{\text{def}}{=} \sum_{i=1}^n y_i$ counts the number of points in the class 1 (ie. the *empirical frequency* of the class 1).

[3] The signature of the classes used for the classifiers is simple and similar to the one used in the whole `scikit-learn` package, but I implemented the 4 models myself (classes `myLDA`, `myLogisticRegression`, `myLinearRegression`, `myQDA`). A classifier gets trained with `clf.fit(X,y)`, predicts labels with `clf.predict(X)` and the score is computed with `clf.score(X,y)`.

For the part depending on μ_0 and μ_1, $(x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i}) = y_i(x_i - \mu_0)^T\Sigma^{-1}(x_i - \mu_0) + (1 - y_i)(x_i - \mu_1)^T\Sigma^{-1}(x_i - \mu_1)$ because, as usual, either y_i or $1 - y_i$ is 0 and 1 (mutually exclusive).

Now, we can see that this function l (of $\boldsymbol{\theta}$) is in fact **concave** in π, μ_0, μ_1 and Σ (**separately**):

- for π: $\pi \mapsto n_1 \log\left(\frac{\pi}{1-\pi}\right) + n \log(1 - \pi)$ is concave as sum of weighted concave functions,
- for μ_0 (and idem for μ_1): $\mu_j \mapsto -\frac{1}{2}\sum_{i=1}^n y_i \left((x_i - \mu_j)^T\Sigma^{-1}(x_i - \mu_j)\right)$ is the opposite of a weighted sum of quadratic functions (in μ_j), so it is concave (indeed $\Sigma^{-1} \in S_{++}^2$ is symmetric p.s.d.),
- for Σ: first, $\Sigma \mapsto \frac{1}{2}n \log\det(\Sigma)$ is well defined and concave (on S_{++}^2, as seen during the lecture), and
  $\Sigma \mapsto -\frac{1}{2}\sum_{i=1}^n \left((x_i - \mu_{y_i})^T\Sigma^{-1}(x_i - \mu_{y_i})\right)$ is affine in Σ^{-1}, so[4] is concave in Σ, and finally their sum is concave.

For l concave (and of class $\mathcal{C}^1$), we know that the maximum log-likelihood estimator ($\hat{\boldsymbol{\theta}}^{(n)}$) can be found by computing its derivatives (or gradients) for π, μ_0, μ_1 and Σ and canceling them (separately).

For π: This part is the easier to compute, let start here:

$\frac{\partial l}{\partial \pi}(\boldsymbol{\theta}) = n\frac{\mathrm{d}}{\mathrm{d}\pi}\log(1 - \pi) + n_1\frac{\mathrm{d}}{\mathrm{d}\pi}\log\left(\frac{\pi}{1-\pi}\right) = n/(\pi - 1) + n_1/(\pi(1 - \pi))$. So $\frac{\partial l}{\partial \pi}(\boldsymbol{\theta}) = 0 \Leftrightarrow \frac{1}{1-\pi}\left(\frac{n_1}{\pi} - n\right) \Leftrightarrow \pi = \frac{n_1}{n}$.

Therefore we have $\hat{\pi}^{(n)} = \frac{n_1}{n}$ the empirical estimator of the probability π.

Note: *this result is not surprising*: the MLE for the probability of y to be in class 1 (drawn from a Bernoulli) is the *empirical frequency of the class* 1 (we remind that $n_1 \overset{\text{def}}{=} \sum_{i=1}^n y_i$).

For μ_1: The part of $l(\boldsymbol{\theta})$ depending on μ_1 is only $-\frac{1}{2}\sum_{i=1}^n y_i \left((x_i - \mu_1)^T\Sigma^{-1}(x_i - \mu_1)\right)$.

Therefore, $\nabla_{\mu_1} l(\boldsymbol{\theta}) = -\frac{1}{2}\sum_{i=1}^n y_i \nabla_{\mu_1}\left((x_i - \mu_1)^T\Sigma^{-1}(x_i - \mu_1)\right)$ by linearity of the operator ∇_{μ_1}. But each term inside the sum is a quadratic form (in μ_1) given by the symmetric p.s.d. matrix Σ^{-1}.

We have this classical result:

Lemma 1. *If $f(x) = (x - x_0)^T A(x - x_0)$ (for A symmetric), then $\nabla_x f(x) = 2A(x - x_0)$.*

Proof. We write $f(x + h) - f(h) = (x + h - x_0)^T A(x + h - x_0) - (x - x_0)^T A(x - x_0) = h^T A(x - x_0) + (x - x_0)^T Ah + o(\|h\|)$. And because A is symmetric, the linear part of that expression is $(2A(x - x_0))^T h = (\nabla_x f(x))^T h = \langle \nabla_x f(x), h\rangle$, and this gives the gradient as claimed. □

So this lemma can be applied here on each quadratic form of the sum, they all share the same $A = \Sigma^{-1}$ and $x_0 = \mu_1$, so:

$$\begin{aligned}
\nabla_{\mu_1} l(\boldsymbol{\theta}) &= -\frac{1}{2}\sum_{i=1}^n y_i \left(2\Sigma^{-1}(x_i - \mu_1)\right) \\
&= -\Sigma^{-1}\left(\sum_{i=1}^n y_i(x_i - \mu_1)\right) \\
&= \Sigma^{-1}\left(n_1\mu_1 - \sum_{i=1}^n y_i x_i\right)
\end{aligned}$$

[4] In fact, **this is not true** here in general, but at the end we can check that the stationary points of the log-likelihood l are maximizing points, for example by computing the Hessian w.r.t. $\boldsymbol{\theta}$ (not easy to compute).

n_1 was defined above as $\sum_{i=1}^n y_i$, and now conclude that $\nabla_{\mu_1} l(\boldsymbol{\theta}) = 0 \Leftrightarrow \Sigma^{-1}\left(n_1\mu_1 - \sum_{i=1}^n y_i x_i\right) = 0$, but Σ^{-1} is invertible so this is equivalent to $(n_1\mu_1 - \sum_{i=1}^n y_i x_i) = 0$, which gives $\nabla_{\mu_1} l(\boldsymbol{\theta}) = 0 \Leftrightarrow \mu_1 = \hat{\mu_1}^{(n)} = \frac{1}{n_1}\sum_{i=1}^n y_i x_i$.

Similarly, the same computations for μ_0 gives exactly the same derivation (except all the y_i are replaced with $1 - y_i$), and therefore $\nabla_{\mu_0} l(\boldsymbol{\theta}) = 0 \Leftrightarrow \mu_0 = \hat{\mu_0}^{(n)} = \frac{1}{n_0}\sum_{i=1}^n (1-y_i)x_i$ where $n_0 \overset{\text{def}}{=} n - n_1 \overset{\text{def}}{=} \sum_{i=1}^n (y_i)$ is the empirical frequency of the class 0.

Remark: *as expected*, the MLE for the centers μ_0 and μ_1 *are respectively the empirical mean of the points* (x_i) *in class* 0 *(for which* $y_i = 0$*) and of the points* (x_i) *in class* 1 *(*$y_i = 1$*).*

For Σ: The part of $l(\boldsymbol{\theta})$ depending on Σ has two terms, one being $\frac{n}{2}\log\det(\Sigma)$ and the second one is the big sum $\sum_{i=1}^n y_i(x_i-\mu_0)^T\Sigma^{-1}(x_i-\mu_0) + (1-y_i)(x_i-\mu_1)^T\Sigma^{-1}(x_i-\mu_1)$.

For the first term, we have this classical lemma:

Lemma 2. *If* $f(\Sigma) = \log\det(\Sigma^{-1})$ *on* S_{++}^d*, then* $\nabla_\Sigma(f) = \Sigma$.

Proof. It was done in the lecture 1, see the lecture notes (from 2013), page 14. □

The second term can be written as $\sum_{i=1}^n y_i(x_i-\mu_0)^T\Sigma^{-1}(x_i-\mu_0) + (1-y_i)(x_i-\mu_1)^T\Sigma^{-1}(x_i-\mu_1) = \text{Trace}(\Sigma^{-1}\tilde{\Sigma})$ where $\tilde{\Sigma} \overset{\text{def}}{=} \sum_{i=1}^n \left(y_i(x_i-\mu_1)(x_i-\mu_1)^T + (1-y_i)(x_i-\mu_0)(x_i-\mu_0)^T\right)$ is the class-wise empirical variance matrix (if we assumed μ_0, μ_1 to be known).

But the unity of the stationary point let us choose the ML estimators for μ_j, so this gives $\nabla_\Sigma l(\boldsymbol{\theta}) = \frac{n}{2}\Sigma - \frac{n}{2}\tilde{\Sigma} = 0 \Leftrightarrow \Sigma = \hat{\Sigma}^{(n)} = \tilde{\Sigma}$.

Finally, we find $\hat{\Sigma}^{(n)} = \left(\sum_{i=1}^n y_i(x_i - \hat{\mu_1}^{(n)})(x_i - \hat{\mu_1}^{(n)})^T\right) + \left(\sum_{i=1}^n (1-y_i)(x_i - \hat{\mu_0}^{(n)})(x_i - \hat{\mu_0}^{(n)})^T\right)$.

Conclusion: max likelihood estimators for LDA

We have been able to derive formulas for the MLE $\hat{\boldsymbol{\theta}}^{(n)}$, which gives an easy to implement algorithm for the **Linear Discriminant Analysis model**:

- For the frequency of class 1: $\hat{\pi}^{(n)} = \frac{n_1}{n} = \frac{1}{n}\sum_{i=1}^n y_i$ is the empirical frequency of the class 1.
- For the center of the ellipsoids: $\hat{\mu_1}^{(n)} = \frac{1}{n_1}\sum_{i=1}^n y_i x_i$, and $\hat{\mu_0}^{(n)} = \frac{1}{n_0}\sum_{i=1}^n (1-y_i)x_i$ are empirical means of the points in class 1 (resp. 0).
- $\hat{\Sigma}^{(n)} = \left(\sum_{i=1}^n y_i(x_i - \hat{\mu_1}^{(n)})(x_i - \hat{\mu_1}^{(n)})^T\right) + \left(\sum_{i=1}^n (1-y_i)(x_i - \hat{\mu_0}^{(n)})(x_i - \hat{\mu_0}^{(n)})^T\right)$ is the class-wise empirical variance matrix (empirical estimate of the common shape of the ellipsoid for both classes).

These max-likelihood estimators all have very meaningful interpretations (as discussed above).

(b) Shape of the points in class 1

The conditional distribution $\mathbb{P}(Y = 1|X)$ is shaped as the weighted sum of two Gaussian distributions (in $\mathbb{R}^2$, ie. two ellipsoids), both of variance Σ, one centered in μ_1 and having weight π, and the second one centered in μ_0 and having weight $(1-\pi)$.

Because the two ellipsoids have the same shape, in fact it is like if the two clouds of points were separated by a line, parallel to the direction of the big axis of the ellipse, and centered between the two means μ_0, μ_1.

Furthermore, this leads to a linear decision boundary, exactly as for the Logistic and Linear regressions. Concretely, we clearly observe this behavior on the decision boundary plot included below[5]:

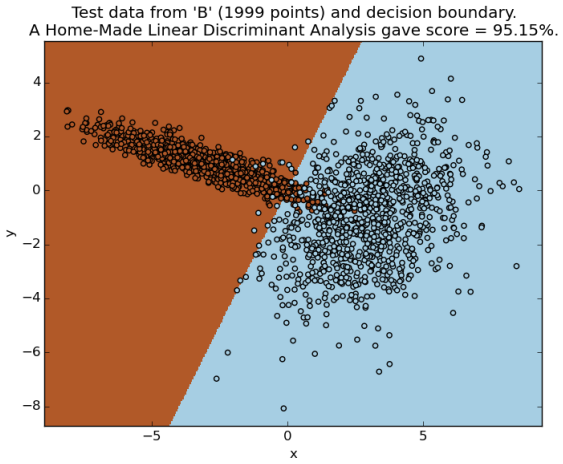

Figure 1: Decision boundary for the LDA classifier (test **B**).

(c) Implementation

See the included program for details, in the class `myLDA`, especially the `fit` and `predict` methods.

The algorithm is simple, we just compute some empirical mean vectors (μ_0, μ_1) or covariance matrices (Σ), by using `numpy.mean()` and `numpy.cov()`.

- From the training dataset **A** (with 149 points), we learned these values of the parameters $\pi, \mu_0, \mu_1, \Sigma$: π = `0.328`, μ_0 = `[ 2.89 -0.89 ]` and μ_1 = `[ -2.67 0.85 ]`, Σ = `[[ 5.05 -2.36 ], [ -2.36 1.27 ]]`.
- From the training dataset **B** (with 299 points), we learned these values of the parameters $\pi, \mu_0, \mu_1, \Sigma$: π = `0.50`, μ_0 = `[ 3.34 -0.84 ]` and μ_1 = `[ -3.22 1.09 ]`, Σ = `[[ 6.72 -0.28 ], [ -0.28 3.48 ]]`.
- From the training dataset **C** (with 399 points), we learned these values of the parameters $\pi, \mu_0, \mu_1, \Sigma$: π = `0.62`, μ_0 = `[ 2.79 -0.84 ]` and μ_1 = `[ -2.95 -0.97 ]`, Σ = `[[ 5.77 -0.53 ], [ -0.53 9.49 ]]`.

2.2 : Logistic regression

(a) Implementation

As suggested, we used an iterative re-weighted least squares algorithm (**IRLS**) to implement the logistic regression.

[5] All the figures are included in the `zip` archive of the project, in the `fig/` directory, please have a look for more illustrations.

See the included program for details, in the class `myLogisticRegression`, especially the `fit` and `predict` methods. IRLS is a Newton method, rewritten in this case where we have known forms for the gradient and the Hessian. In fact, if $\eta_i \overset{\text{def}}{=} \sigma(w^T x_i + b)$ (the sigmoid function), then looking for the extreme point is like trying to minimize the norm of the gradient (because the problem is strictly convex): $\sum_{i=1}^{n} x_i(y_i - \eta_i)$ which is a weighted (by x_i) account of the number of errors. We used this as a stopping criteria for the iterative loop in IRLS: if the gradient is small enough, we must be very close to the convergence, so we can stop.

In practice, we start with $w^{(t=0)} = w^{(0)} = \mathbf{0}$ a zero vector (as suggested in an old Francis Bach lecture note for this course). We found that w^t increases in norm, greatly, but still produce a good classifier at the end. The algorithm works well (see the scores below) but I might have made a scaling mistake somewhere (the matrices and vectors in IRLS should not be that small and the final $\tilde{w}^*$ should have the same order of magnitude as the ones given by `LinearRegression`).

At the end, the classifier acts like a linear decision rule ($f(X) = w^T X + b$), but b varies in this case (for the linear regression, we found $b \simeq 0.50$ each time).

- From the training dataset **A** (with 149 points), we learned these values of the parameters w, b: $w =$ `[ -2.12 -1.55 ]`, and $b =$ `-0.25` (for `scikit-learn`'s implementation). $w =$ `[ -5.46 -9.36 ]`, and $b =$ `-1.0` (for my implementation, after dividing by `abs(b)` to rescale).
- From the training dataset **B** (with 299 points), we learned these values of the parameters w, b: $w =$ `[ -1.50 0.87 ]`, and $b =$ `0.99` (for `scikit-learn`'s implementation). $w =$ `[ -1.26 0.76 ]`, and $b =$ `1.0` (for my implementation, after dividing by `abs(b)` to rescale).
- From the training dataset **C** (with 399 points), we learned these values of the parameters w, b: $w =$ `[ -2.00 0.55 ]`, and $b =$ `0.72` (for `scikit-learn`'s implementation). $w =$ `[ -2.30 0.74 ]`, and $b =$ `1.0` (for my implementation, after dividing by `abs(b)` to rescale).

(b) Decision boundary (one example)

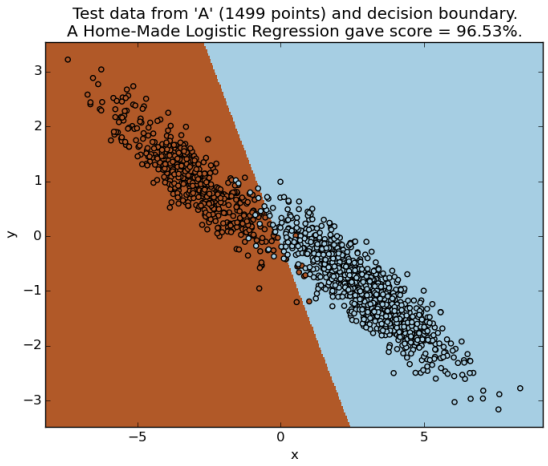

Figure 2: Decision boundary for the Logistic classifier (test **A**).

2.3 : Linear regression

(a) Implementation

As suggested, we solved the normal equations (with a Moore-Penrose pseudo-inverse) to implement the linear regression.

See the included program for details, in the class `myLinearRegression`, especially the `fit` and `predict` methods. Going from a linear regression to a linear classifier is done by using the formula for the posterior probability: $\mathbb{P}(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{1}{2}(y_i - w^T x_i - b)^2/\sigma^2)\right)$. So an unknown point x is affected to the class 0 by the classifier if $g(x) = 0 \Leftrightarrow \mathbb{P}(0|x) \geqslant 1/2 \Leftrightarrow \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{1}{2}(w^T x_i + b)^2/\sigma^2)\right) \geqslant 1/2$ (and that is the criteria implemented).

At the end, the classifier acts like a linear decision rule ($f(X) = w^T X + b$), and we observed that b did not vary much in this case: $b \simeq 0.50$ each time.

- From the training dataset **A** (with 149 points), we learned these values of the parameters w, b: $w =$ `[ -0.26 -0.37 ]`, and $b =$ `0.491`, and $\sigma^2 =$ `0.040`.
- From the training dataset **B** (with 299 points), we learned these values of the parameters w, b: $w =$ `[ -0.10 0.05 ]`, and $b =$ `0.499`, and $\sigma^2 =$ `0.054`.
- From the training dataset **C** (with 399 points), we learned these values of the parameters w, b: $w =$ `[ -0.13 -0.02 ]`, and $b =$ `0.507`, and $\sigma^2 =$ `0.062`.

(b) Decision boundary (one example)

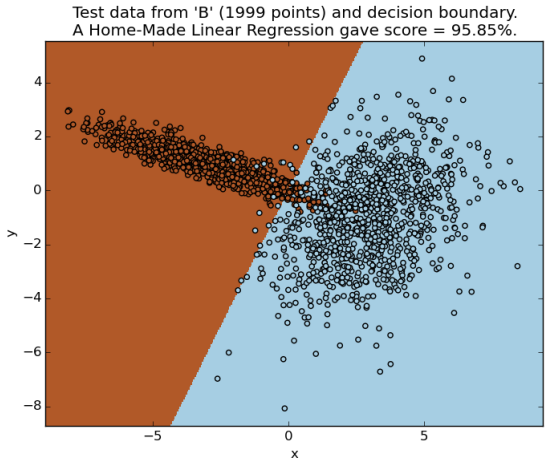

Figure 3: Decision boundary for the Linear classifier (test **B**).

2.4 : Scores on test and train data

(a) Scores for each model

Here are included the "raw" scores obtained for each model (with the `scikit-learn` implementations, but mine gave almost the same scores). We computed two different scores (for more details, see the files `output.txt` and `pretty_scores.txt` included in the archive):

- **Test score** is the prediction error for a classifier trained on the train dataset (149, 299 or 399 points), and prediction testing was done on the test dataset (1499, 1999 or 2999 points) *as usual*. It evaluates the prediction accuracy on new data.
- **Train score**, on the other hand, evaluates the loss of information in the training step: it is the prediction error for a classifier trained on the train dataset (**A**, **B** or **C**), but also tested on the same training dataset.

For the **LDA** model:

- **Test scores**: $97.93\%, 95.85\%, 95.80\%$ (*usual score*),
- **Train scores**: $98.66\%, 96.99\%, 94.49\%$.

For the **Logistic Regression** model:

- **Test scores**: $96.53\%, 95.70\%, 97.73\%$ (*usual score*),
- **Train scores**: $100\%, 97.99\%, 95.99\%$ (note: my `LogisticRegression` was the only one to get a train score of 100% at one point).

For the **Linear Regression** model:

- **Test scores**: $98.00\%, 95.85\%, 95.80\%$ (*usual score*),
- **Train scores**: $98.66\%, 96.99\%, 94.74\%$.

For the **QDA** model:

- **Test scores**: $98.00\%, 98.00\%, 96.20\%$ (*usual score*),
- **Train scores**: $99.33\%, 98.66\%, 94.74\%$.

The QDA model seems to be the better one here (as explained below).

(b) Comment on the results

The distribution from which **C** was generated appear to be more complicated than the ones for **B** and **A**, and as expected we observe that all the classifiers perform better on **A** or **B** than on **C**.

- For datasets **A**, they all perform very well: it is linearly separable so getting a performance of 98% (on both training and testing datasets) is not surprising.
- For **B**, **QDA** is clearly better than the 3 others, and this is also understandable: the two ellipsoids in B overlap a little bit (around $(0, 0)$), and a conic frontier on class 0 (brown) will perform better, as illustrated in the decision boundary figures (see the included files for more details).
- For **C**, **Logistic regression** seems slightly better than the other models. In dataset **C**, there is an additional small ball of points (of class 1), and because they are dense, they are considered more important in Logistic regression than in Linear regression, so the linear frontier take them more into account for Logistic than for Linear regression:

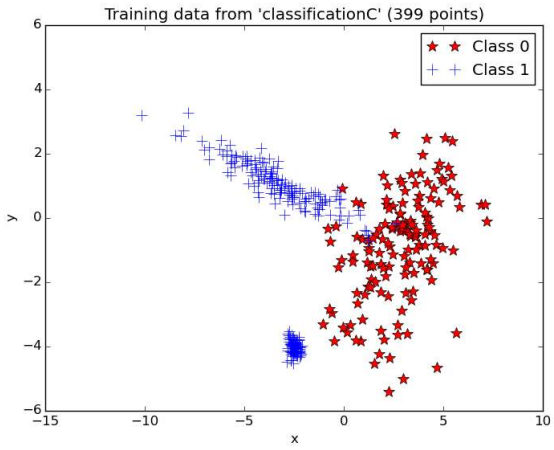

Figure 4: Training dataset (test **C**).

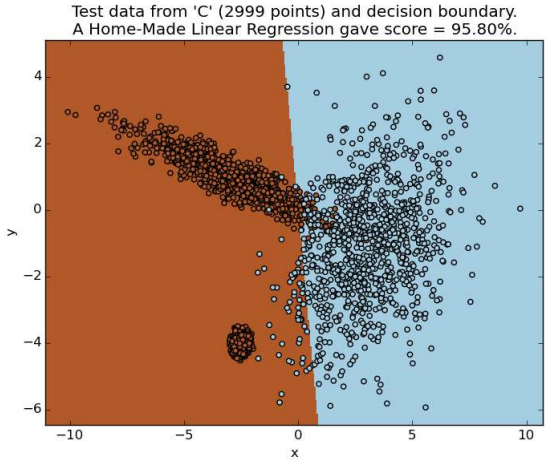

Figure 5: A linear frontier with score 95.80% from `LinearRegression` (test **C**).

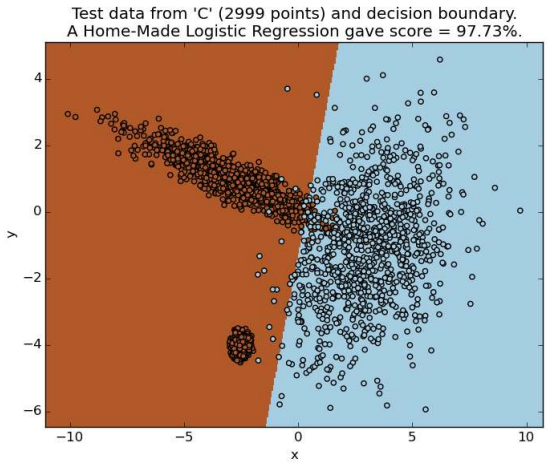

Figure 6: A better linear frontier with score 97.73% from `LogisticRegression` (test **C**).

In practice, the performance of these methods seemed to be already quite good[6]. When we computed the test score for a classifier trained on the (bigger) `test` dataset and tested on the (smaller) `train` dataset, we found almost the same performance). Therefore these classifiers are as efficient with 150 training sample as they are with 10 times more: we can interpret this by saying they do not need an unreasonable number of training points.

2.5 : QDA model

Max likelihood estimators for the QDA model

We can do again the same computations as for the LDA model (cf. *2.1 (a)*), and the estimators for π, μ_0 and μ_1 are found to be exactly the same.

What changes in this case is the variance matrix (ie. the shape of the ellipsoids): we no longer have one shared Σ, but two different Σ_0 and Σ_1, for which canceling the gradient of $l(\boldsymbol{\theta})$ gives the following MLE:

- $\hat{\Sigma_1}^{(n)} = \frac{1}{n_1} \sum_{i=1}^{n} y_i (x_i - \hat{\mu_1}^{(n)})(x_i - \hat{\mu_1}^{(n)})^T$,
- $\hat{\Sigma_0}^{(n)} = \frac{1}{n_0} \sum_{i=1}^{n} (1 - y_i)(x_i - \hat{\mu_0}^{(n)})(x_i - \hat{\mu_0}^{(n)})^T$.

(a) Implementation

See the included program for details, in the class `myQDA`, especially the `fit` and `predict` methods.

The algorithm is simple, very similar to the one for LDA, we just compute some empirical mean vectors (μ_0, μ_1) or covariance matrices (Σ_0, Σ_1), by using `numpy.mean()` and `numpy.cov()`[7] on the sub-arrays `X[y == 0]` for points (x_i) such that $y_i = 0$ (ie class 0), and `X[y == 1]` for points (x_i) such that $y_i = 1$ (ie class 1).

From the training dataset **A** (with 149 points), we got a test score of 98%, and we learned these values of the parameters $\pi, \mu_0, \mu_1, \Sigma_0, \Sigma_1$:

- $\pi =$ `0.328`,
- $\mu_0 =$ `[ 2.89 -0.89 ]` and $\mu_1 =$ `[ -2.67 0.85 ]`,
- $\Sigma_0 =$ `[[ 2.31 -1.04 ], [ -1.04 0.57 ]]`,
- $\Sigma_1 =$ `[[ 2.74 -1.31 ], [ -1.32 0.70 ]]`.

From the training dataset **B** (with 299 points), we got a test score of 98%, and we learned these values of the parameters $\pi, \mu_0, \mu_1, \Sigma_0, \Sigma_1$:

- $\pi =$ `0.50`,
- $\mu_0 =$ `[ 3.34 -0.84 ]` and $\mu_1 =$ `[ -3.22 1.09 ]`,
- $\Sigma_0 =$ `[[ 2.54 1.06 ], [ 1.06 2.96 ]]`,

[6] On this very dataset, especially chosen to be (almost) linearly separable...

[7] They are NumPy built-in functions, but easy to implement manually if needed.

- Σ_1 = `[[ 4.18 -1.34 ], [ -1.34 0.52 ] ]`.

From the training dataset **C** (with 399 points), we got a test score of 96.20%, and we learned these values of the parameters $\pi, \mu_0, \mu_1, \Sigma_0, \Sigma_1$:

- π = `0.62`,
- μ_0 = `[ 2.79 -0.84 ]` and μ_1 = `[ -2.95 -0.97 ]`,
- Σ_0 = `[[ 2.90 1.25 ], [ 1.25 2.92 ]]`,
- Σ_1 = `[[ 2.87 -1.78 ], [ -1.78 6.57 ]]`.

Concretely, we check that the parameters are the same as the one learned for LDA, except the covariance matrices. We can check that all the Σ_j are symmetric.

For example, for **B**, we see that the ellipses are sort of orthogonal, as seen in the decision boundary plot ($\Sigma_0(B)$ gives the direction `[ 2.54 2.96 ]` for the blue points, and $\Sigma_1(B)$ gives the direction `[ 4.18 0.52 ]` for the brown points).

(b) Decision boundary (for each dataset)

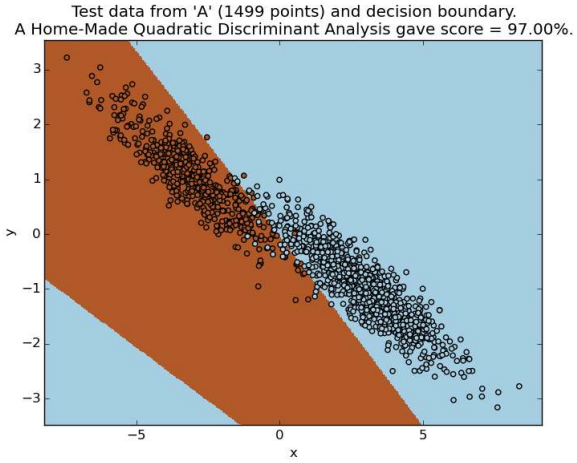

Figure 7: Decision boundary for the QDA classifier (test **A**).

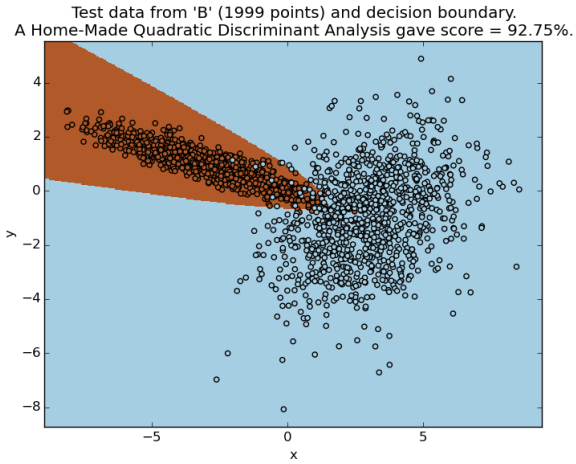

Figure 8: Decision boundary for the QDA classifier (test **B**), orthogonal ellipses.

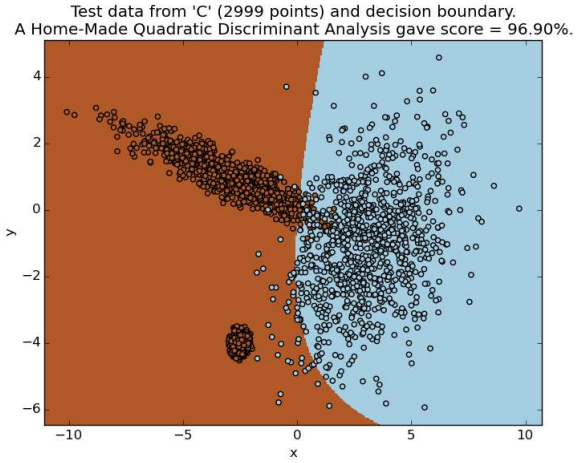

Figure 9: Decision boundary for the QDA classifier (test **C**).

(c) Scores for the QDA classifier

- **Test scores**: $98.00\%, 98.00\%, 96.20\%$ (*usual score*),
- **Train scores**: $99.33\%, 98.66\%, 94.74\%$.

(d) Comment on the results

This last classifier seems to be very efficient on the 3 datasets, and especially on **A** and **B**.

It always performs better than LDA on the 3 datasets (cf the scores above), and this behavior is normal: learning two different shapes for the ellipsoids (Σ_0, Σ_1 for QDA) is more general than a common shape (Σ for LDA), therefore more powerful to build a classifier.

Conclusion

Remark: I know this work is far from being perfect, but I already spent "too much" time on it, so I must stop now.

- The proofs for the max likelihood estimators should be done more carefully (especially the part when we find the stationary point and conclude that its a max, the functions are not concave or strictly concave so a consideration on the sign of the Hessian should be added).
- The code is too long, I should have optimized conciseness rather than generality. The most important parts are the `fit` and `predict` methods for the 4 classes I wrote for the 4 classifiers, and these parts are easy to check.