# Project FTML

Moustapha DIOP
Marc MONTEIL
Theo PERINET
Mathieu RIVIER
Martin POULARD

June 17, 2022

# 1 Bayes estimator and Bayes risk

## 1.1 Question 1

Nous proposons comme espace de départ $X : \{0, 1, 2, 3\}$ et $Y : \{0, 1\}$.

$$X \rightsquigarrow U([0, 1, 2]) + B(\frac{2}{3}) \qquad\qquad Y \rightsquigarrow \begin{cases} B(\frac{\pi}{4}) \ si \ X = 0 \\ B(\frac{\sqrt{2}}{2}) \ si \ X = 1 \\ B(\frac{e}{3}) \ si \ X = 2 \\ B(-\cos 42) \ si \ X = 3 \end{cases}$$

On va prendre la ″0-1" loss.

$$\begin{aligned} f^*(x) &= \arg\min_{z \in Y} \mathbb{E}\big[l(Y, z)|X = x\big] \\ &= \arg\min_{z \in Y} \mathbb{P}\big(Y \neq z|X = x\big) \\ &= \arg\max_{z \in Y} \mathbb{P}\big(Y = z|X = x\big) \end{aligned}$$

$$\begin{aligned} &f^*(0) = \arg\max_{z \in Y} \mathbb{P}\big(Y = z|X = 0\big) = 1 \ car \ \frac{\pi}{4} > \frac{1}{2} \\ &f^*(1) = 1 \ car \ \frac{\sqrt{2}}{2} > \frac{1}{2} \\ &f^*(2) = 1 \ car \ \frac{e}{3} > \frac{1}{2} \\ &f^*(3) = 0 \ car \ -\cos 42 < \frac{1}{2} \end{aligned} \tag{1}$$

On va maintenant calculer le risque de Bayes.

$$\begin{aligned} R^* &= \mathbb{E}_{(X,Y)}[l(Y, f^*(X))] \\ &= \mathbb{E}_X[\mathbb{E}_Y[l(Y, f^*(X)|X]] \end{aligned}$$

On va déterminer $\mathbb{E}_Y[l(Y, f^*(X)|X]$

On pose $g(x) = \mathbb{E}_Y[l(Y, f^*(X)|X = x]$

On va faire une disjonction de cas sur X.

<u>Premier cas X = 0 :</u>

$\mathbb{E}_Y[l(Y, f^*(X)|X = 0] = \mathbb{P}(Y \neq f^*(0)|X = 0) = \mathbb{P}(Y \neq 1|X = 0) = \mathbb{P}(Y = 0|X = 0) = 1 - \frac{\pi}{4}$

Deuxième cas X = 1 :

$\mathbb{E}_Y[l(Y, f^*(X)|X = 1] = 1 - \frac{\sqrt{2}}{2}$

Troisième cas X = 2 :

$\mathbb{E}_Y[l(Y, f^*(X)|X = 2] = 1 - \frac{e}{3}$

Quatrième cas X = 3 :

$\mathbb{E}_Y[l(Y, f^*(X)|X = 3] = -\cos(42)$

$$\begin{aligned} \textit{Ainsi } \mathbb{E}_{(X,Y)}[l(Y, f^*(X))] &= \mathbb{E}_X(g(X)) \\ &= \mathbb{P}(X = 0)g(X = 0) + \mathbb{P}(X = 1)g(X = 1) + \mathbb{P}(X = 2)g(X = 2) \\ &+ \mathbb{P}(X = 3)g(X = 3) \textit{ par linéarité de l'espérance.} \end{aligned}$$

On va maintenant calculer $\mathbb{P}(X = x)$

Premier cas x = 0 :

$\mathbb{P}(X = 0) = \mathbb{P}(U(\{0, 1, 2\}) + B(\frac{2}{3}) = 0) = \frac{1}{3} * (1 - \frac{2}{3}) = \frac{1}{9}$

Deuxième cas x = 1 :

$\mathbb{P}(X = 1) = \frac{1}{3}$

Troisième cas x = 2 :

$\mathbb{P}(X = 2) = \frac{1}{3}$

Quatrième cas x = 3 :

$\mathbb{P}(X = 3) = \frac{2}{9}$

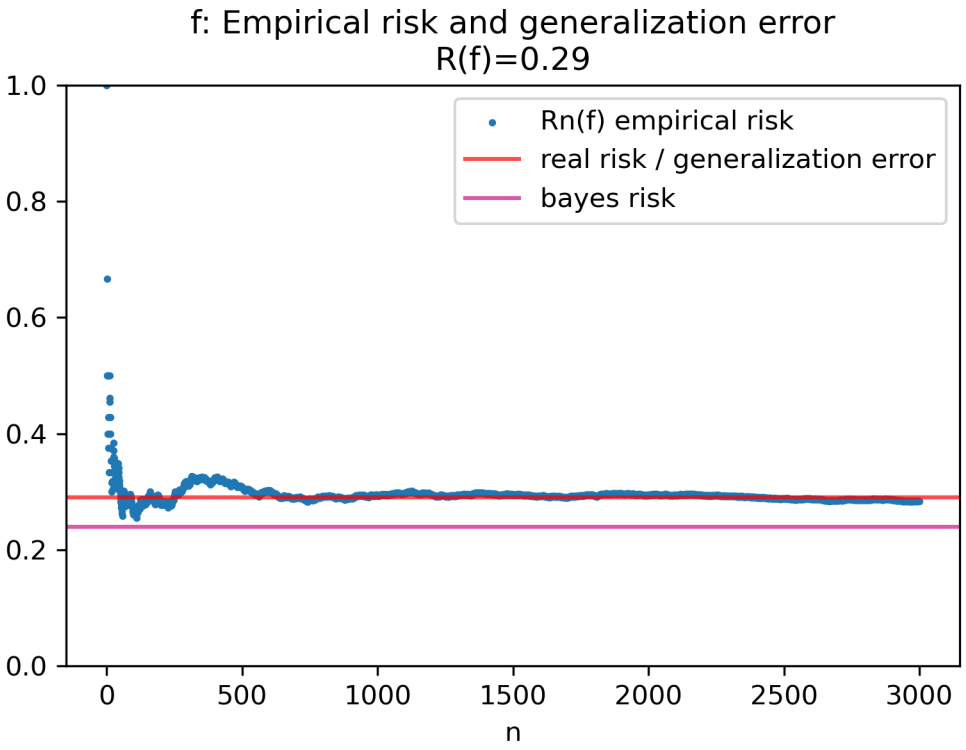
$$\begin{aligned}
\textit{Donc } \mathbb{E}_X(g(X)) &= \frac{1}{9} * (1 - \frac{\pi}{4}) + \frac{1}{3} * (1 - \frac{\sqrt{2}}{2}) + \frac{1}{3} * (1 - \frac{e}{3}) + \frac{2}{9} * (-\cos(42)) \\
&= \frac{7 - e - 2 * \cos(42)}{9} - \frac{\pi}{36} - \frac{\sqrt{2}}{6} \\
&\approx 0,24
\end{aligned}$$

## 1.2 Question 2

$$\textit{On pose } \tilde{f} = \begin{cases} X \to Y \\ x \mapsto 1 \end{cases}$$

$R(\tilde{f}) = 1 - \frac{\pi}{36} - \frac{\sqrt{2}}{6} + \frac{2 * \cos(42) - e}{9} \approx 0,29 > R^*$. Ce qui est cohérent, car le Bayésien est tout le temps inférieur au risque réel.

Avec la simulation proposée, on retrouve bien le risque réel qui tend vers 0,29 :



# 2 Bayes risk with absolute loss

## 2.1 Question 1

On doit trouver un setup tel que le risque de Bayes soit différent entre l'absolute loss et la square loss.

On prend $X \in \mathbb{R},\ X \rightsquigarrow N(0,1)$ et $Y$ est une VAR $\in \mathbb{R}$ et pour $X = 0,\ Y \rightsquigarrow U(-2,-1,0,10,100)$.

On va poser $l1$ = absolute loss et $l2$ = square loss. On va calculer explicitement les risques de Bayes en fonction des deux loss $l1$ et $l2$.

Premier cas $l1$ :

- si $z = -2$ on a $|-2+2| + |-1+2| + |0+2| + |10+2| + |100+2| = 117$
- si $z = -1$ on a 114
- si $z = 0$ on a 113
- si $z = 10$ on a 123
- si $z = 100$ on a 393

Donc $\arg\min_{z \in Y} \mathbb{E}(|Y - z||X = 0) = 0$. Ce qui est logique, car 0 est la médiane de l'ensemble cf Q2.

Deuxième cas $l2$ :

- si $z = -2$ on a $(-2+2)^2 + (-1+2)^2 + (0+2)^2 + (10+2)^2 + (100+2)^2 = 10553$
- si $z = -1$ on a 10324
- si $z = 0$ on a 10105
- si $z = 10$ on a 8465
- si $z = 100$ on a 38705

Donc $\arg\min_{z \in Y} \mathbb{E}((Y - z)^2 | X = 0) = 10$.

Donc $f_1^*(0) \neq f_2^*(0)$

Finalement on voit bien que deux loss différentes peuvent conduire à deux risques de Bayes différents pour un même setup.

## 2.2 Question 2

On veut minimiser sur $Y$ : $g(z) = \int_{y\in\mathbb{R}} |y-z|_{p(Y=y|X=x)}(y)\mathrm{d}y$

Donc $g(z) = \int_{z}^{+\infty} (y-z)_{p(Y=y|X=x)}(y)\mathrm{d}y + \int_{-\infty}^{z} (z-y)_{p(Y=y|X=x)}(y)\mathrm{d}y$

$$\begin{aligned} \textit{Donc } g'(z) &= -\int_{z}^{+\infty} p(Y=y|X=x)(y)\mathrm{d}y + \int_{-\infty}^{z} p(Y=y|X=x)(y)\mathrm{d}y \\ &= -\mathbb{P}(Y\geq z|X=x) + \mathbb{P}(Y\leq z|X=x) \end{aligned}$$

Or $g(z)$ est un minimum si $g'(z) = 0 \Rightarrow \mathbb{P}(Y\leq z|X=x) = \mathbb{P}(Y\geq z|X=x)$.
Donc $g(z)$ est minimal si $z$ est la médiane.
Donc $f^*(x) = z$.

# 3 Expected value of empirical risk

## 3.1 Question 1

On doit montrer que $\mathbb{E}(R_n(\hat{\theta})) = \mathbb{E}_\epsilon(\frac{1}{n}\|(I_n - X(X^TX)^{-1}X^T)\epsilon\|^2)$

Or $R_n(\hat{\theta}) = \frac{1}{n}\|Y - X\hat{\theta}\|^2$ *et* $\hat{\theta} = (X^TX)^{-1}X^TY$ *et* $Y = X\theta^* + \epsilon$

$$\begin{aligned}
\textit{Donc } R_n(\hat{\theta}) &= \frac{1}{n}\|X\theta^* + \epsilon - X(X^TX)^{-1}X^T(X\theta^* + \epsilon)\|^2 \\
&= \frac{1}{n}\|X\theta^* + \epsilon - X(X^TX)^{-1}X^TX\theta^* + X(X^TX)^{-1}X^T\epsilon\|^2 \\
&= \frac{1}{n}\|\epsilon(I_n - X(X^TX)^{-1}X^T)\|^2
\end{aligned}$$

Donc $\mathbb{E}(R_n(\hat{\theta})) = \mathbb{E}_\epsilon(\frac{1}{n}\|(I_n - X(X^TX)^{-1}X^T)\epsilon\|^2)$

## 3.2 Question 2

On doit montrer que $tr(A^TA) = \sum_{(i,j)\in[1,n]^2} a_{ji}^2$

Soit $A \in \mathbb{R}^{(n*n)}$, on sait que $(AA)_{i,j} = \sum_{k=[1,n]} a_{ik}a_{kj}$

Donc $(A^TA)_{i,j} = \sum_{k=[1,n]} a_{ki}a_{kj}$

Donc $tr(A^TA) = \sum_{i=1}^{n}\sum_{k=1}^{n} a_{ki}a_{ki} = \sum_{i=1}^{n}\sum_{k=1}^{n} a_{ki}^2$

## 3.3 Question 3

On doit monter que $\mathbb{E}_\epsilon[\frac{1}{n}\|A\epsilon\|^2] = \frac{\sigma^2}{n}tr(A^TA)$

On sait que $\|A\epsilon\|^2 = \sum_{i=1}^{n}(A_i\epsilon)^2$ et $[A\epsilon] = [\epsilon^TA^T]$

$$\begin{aligned}
\mathbb{E}_\epsilon[\frac{1}{n}\|A\epsilon\|^2] &= \frac{1}{n}\mathbb{E}_\epsilon[\sum_{i=1}^{n}(\epsilon^T A_i^T)^2] \\
&= \frac{1}{n}\mathbb{E}_\epsilon[\sum_{i=1}^{n}\epsilon^T A_i^T A_i \epsilon] \ \ car\ (\epsilon^T A_i)^2 = \epsilon^T A_i A_i^T \epsilon \\
&= \frac{1}{n}\mathbb{E}_\epsilon[\sum_{i=1}^{n}\epsilon^T(\sum_{j=1}^{n}A_{ij}A_{ij})\epsilon] \ \ par\ définition\ du\ produit\ matriciel \\
&= \frac{1}{n}\mathbb{E}_\epsilon[\sum_{i=1}^{n}\epsilon^T\epsilon(\sum_{j=1}^{n}A_{ij}A_{ij})] \ \ car\ (\sum_{j=1}^{n}A_{ij}A_{ij})\ est\ un\ réel \\
&= \frac{1}{n}\mathbb{E}_\epsilon[\epsilon^T\epsilon \sum_{(i,j)\in[1,n]^2} A_{ji}^2] \\
&= \frac{1}{n}\mathbb{E}_\epsilon[\epsilon^2 \sum_{(i,j)\in[1,n]^2} A_{ji}^2] \ \ car\ \epsilon^2 = \epsilon^T\epsilon \\
&= \frac{1}{n}tr(A^TA)\mathbb{E}_\epsilon[\epsilon^2] \ \ d'après\ l'étape\ 2 \\
&= \frac{\sigma^2}{n}tr(A^TA) \ \ car\ \mathbb{E}_\epsilon[\epsilon^2] = \sigma^2
\end{aligned}$$

Finalement $\mathbb{E}_\epsilon[\frac{1}{n}\|A\epsilon\|^2] = \frac{\sigma^2}{n}tr(A^TA)$

### 3.4 Question 4

On pose $A = I_n - X(X^TX)^{-1}X^T$

On veut calculer $A^TA$

$$\begin{aligned}
A^TA &= (I_n - X(X^TX)^{-1}X^T)^T(I_n - X(X^TX)^{-1}X^T) \\
&= (I_n - X(X^TX)^{-1}X^T)(I_n - X(X^TX)^{-1}X^T)\ car\ (AB)^T = (B^TA^T)\ et\ car\ (A+B)^T = A^T + B^T \\
&= I_n - 2X(X^TX)^{-1}X + X(X^TX)^{-1}X^TX(X^TX)^{-1}X^T \\
&= I_n - X(X^TX)^{-1}X = A
\end{aligned}$$

Finalement $A^TA = A$

### 3.5 Question 5

$$\begin{aligned}\mathbb{E}[R_n(\hat{\theta})] &= \mathbb{E}_\epsilon[\frac{1}{n}\|(I_n - X(X^TX)^{-1}X^T)\epsilon\|^2] \ d'après\ la\ l'étape\ 1 \\ &= \mathbb{E}_\epsilon[\frac{1}{n}\|A\epsilon\|^2] \ en\ posant\ A = I_n - X(X^TX)^{-1}X^T \\ &= \frac{\sigma^2}{n} tr(A^TA) \ d'après\ l'étape\ 3 \\ &= \frac{\sigma^2}{n} tr(A) \ d'après\ l'étape\ 4\end{aligned}$$

Or $A = I_n - X(X^TX)^{-1}X^T$

$$\begin{aligned}Donc\ tr(A) &= n - tr(X(X^TX)^{-1}X^T) \\ &= n - tr(X^TX(X^TX)^{-1}) \ \ car\ tr(BA) = tr(AB) \\ &= n - tr(I_d) \ \ car\ X \in \mathbb{R}^{n*d}\ donc\ X^TX \in \mathbb{R}^{d*d} \\ &= n - d\end{aligned}$$

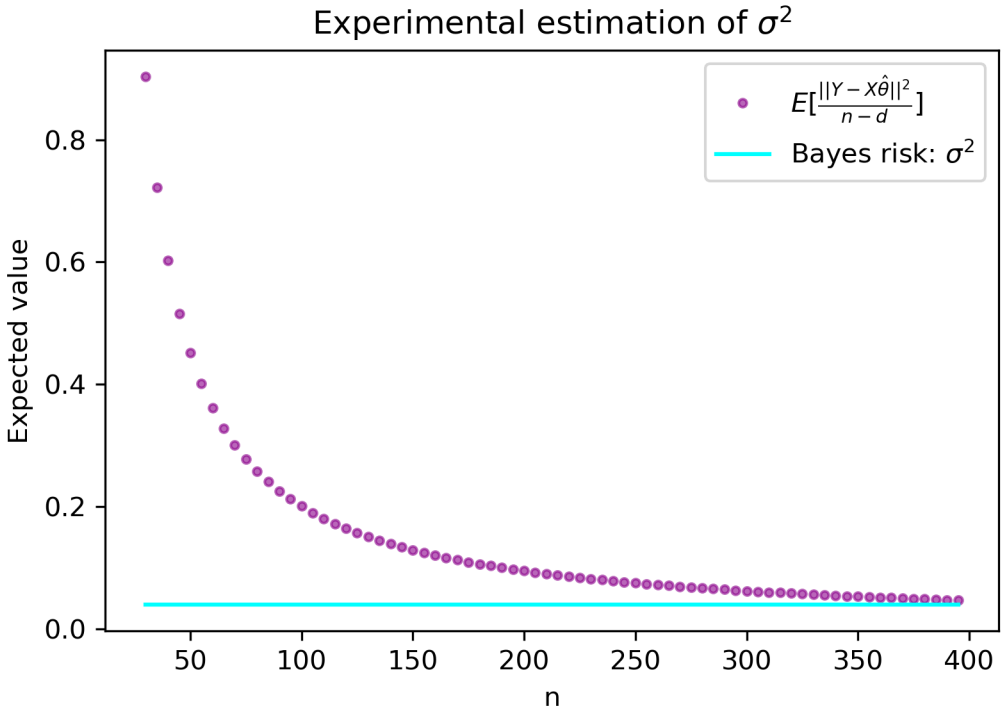
Finalement $E[R_n(\hat{\theta})] = \frac{(n-d)}{n}\sigma^2$

### 3.6 Question 6

Si on prend $\frac{\|Y - X\hat{\theta}\|^2}{n-d}$ à la place de $\frac{1}{n}\|Y - X\hat{\theta}\|^2$ on aura $\mathbb{E}[R_X(\hat{\theta})] = \sigma^2$

### 3.7 Question 7

Avec le set-up proposé, nous voyons que $E[\frac{||Y-X\hat{\theta}||^2}{n-d}]$ tend vers $\sigma^2$:


Experimental estimation of $\sigma^2$
$E[\frac{||Y-X\hat{\theta}||^2}{n-d}]$
Bayes risk: $\sigma^2$
Expected value
0.0
0.2
0.4
0.6
0.8
50
100
150
200
250
300
350
400
n

# 4 Regression

Dans cette partie, nous allons résoudre un problème de régression en comparant différents modèles de machine learning. Pour cela nous allons avoir besoin de deux bibliothèques :

- Numpy : pour charger les données et bénéficier des calculs matriciels vectorisés.
- Sklearn : qui contient tous les modèles de régressions dont nous avons besoin.

Nous avons à notre disposition un petit jeu de données composé de 1000 lignes et 20 colonnes.

Afin de pouvoir tester l'efficacité de nos modèles, nous avons décidé de séparer notre jeu de données en deux. 80% des données sont utilisées pour entraîner le modèle et les 20% restant pour le tester. De plus, cette séparation se fait de manière aléatoire avec un "random state" de 20. Tous nos modèles sont entraînés et testés avec les mêmes données afin de pouvoir les comparer avec le plus d'objectivité possible. Nous avons aussi utilisé les classes *RandomizedSearchCV* et *GridSearchCV* de *Sklearn* pour avoir un entraînement plus efficace grâce à la Cross Validation, qui permet aussi de réduire un peu l'overfitting, et de pouvoir trouver les meilleurs hyperparamètres rapidement. En effet la classe *RandomizedSearchCV* permet de rechercher les meilleurs hyperparamètres en les testant aléatoirement. Quant à la classe *GridSearchCV*, elle permet de chercher les meilleurs hyperparamètres dans une liste d'hyperparamètres définis. Nous avons aussi normalisé les données avec la fonction *StandardScaler* de *Sklearn* pour faciliter l'apprentissage.

Pour ce projet nous avons ainsi choisi 4 modèles de régression différents :

- Linear Regression
- Ridge
- Support Vector Regression (SVR)
- Elastic Net

Tout le code permettant d'avoir les résultats ci-dessous se situe dans le fichier **regression.py**.

## 4.1 Linear Regression

C'est le premier modèle que nous avons testé. C'est un algorithme simple et qui ne demande pas d'"hyperparameter tuning". Pour l'utiliser, nous avons importé la classe *LinearRegression* de la bibliothèque *Sklearn*.
Les résultats du R2 score sont les suivants :

- Train : 0,9007859535154005
- Test : 0,8865476326734985

On peut voir que le R2 score pour le jeu d'entraînement et de test sont proches de 0,9. La régression linéaire, le modèle de machine learning le plus simple, semble donc bien fonctionner avec ce jeu de données.

## 4.2 Ridge

Le *Ridge* est le deuxième modèle de machine learning que nous avons choisi. Cet algorithme nécessite un peu d'"hyperparameter tuning", donc nous avons donné les paramètres suivants à la classe *GridSearchCV* pour trouver la meilleure combinaison :

- une liste de "alpha"[1] allant de 0,01 à 5 avec un pas de 0,1.
- une liste de "solver"[2] : "svd", "cholesky", "lsqr", "sparse_cg", "sag", "saga"

Finalement l'algorithme a trouvé la combinaison de paramètres suivante :

- "alpha" = 1,09
- "solver" = "svd"

Les résultats du R2 score sont les suivants :

- Train : 0,9001967592907093
- Test : 0,8878966911309424

Ici aussi les R2 score pour le jeu d'entraînement et de test sont proches de 0,9. Le Ridge semble, aussi, bien fonctionner avec ce jeu de données.

## 4.3 Support Vector Regression (SVR)

Le troisième modèle que nous avons sélectionné est un Support Vector Regression. Nous avons utilisé la classe *SVR* de *Sklearn* pour entraîner cet algorithme. Tout comme l'algorithme précédent le *SVR* nécessite de l"hyperparameter tuning". Puisque le nombre de combinaison d'hyperparamètres est assez important, nous avons utilisé la classe *RandomizedSearchCV* pour trouver la meilleure combinaison d'hyperparamètres en testant des valeurs aléatoires dans :

- une liste de "C"[3] allant de 0,1 à 2 avec un pas de 0,1
- une liste de "kernel" : "linear", "rbf", "sigmoid"
- une liste d'"epsilon" allant de 0,1 à 1 avec un pas de 0,1

Finalement l'algorithme a trouvé la combinaison de paramètres suivante :

- "C" = 1,71
- "kernel" = "linear"
- "epsilon" = 0,2

Avec ces paramètres nous obtenons les R2 score suivant :

- Train : 0,8983293351966871
- Test : 0,8849284836837825

Tout comme pour le Rigde et la Linear Regression les R2 score pour le jeu d'entraînement et de test sont proches de 0,9. Le SVR semble, aussi, bien fonctionner avec ce jeu de données.

[1] valeurs de régularisation
[2] algorithme minimisation
[3] valeur de régularisation

## 4.4 Elastic Net

Enfin, le dernier modèle que nous avons choisi est un *Elastic Net*. Cet algorithme comme les deux précédents nécessite un peu d'"hyperparameter tuning" afin de le rendre plus performant. Nous avons donc donné les paramètres suivants à la classe *GridSearchCV* pour trouver la meilleure combinaison :

- une liste de "alpha"[4] allant de 0,01 à 2 avec un pas de 0,02
- une liste de "l1_ratio" allant de 0,01 à 1 avec un pas de 0,01

Finalement l'algorithme a trouvé la combinaison de paramètres suivante :

- "alpha" = 0,33
- "l1_ratio" = 1,0

Les résultats du R2 score sont les suivants :

- Train : 0,8994901503005939
- Test : 0,8890421382247459

Tout comme pour le Rigde, la Linear Regression et le SVR les R2 score pour le jeu d'entraînement et de test sont proches de 0,9. Cet algorithme l'Elastic Net semble, aussi, bien fonctionner avec ce jeu de données.

## 4.5 Résultats

Nous allons maintenant comparer les résultats des différents modèles.

| | **Entraînement** | **Test** |
|---|---|---|
| **Linear Regression** | 0,9007859535154005 | 0,8865476326734985 |
| **Ridge** | 0,9001967592907093 | 0,8878966911309424 |
| **SVR** | 0,8983293351966871 | 0,8849284836837825 |
| **Elastic Net** | 0,8994901503005939 | 0,8890421382247459 |

Table 1: Tableau des R2 score des différents modèles

Nous pouvons voir que les modèles ont des R2 scores très proches. Cependant, sur le jeu de données de tests, l'Elastic Net est un peu meilleur que ses concurrents.

[4] valeur de régularisation

# 5 Classification

Dans cette partie, nous allons résoudre un problème de classification en comparant différents modèles de machine learning. Comme dans la partie précédente, nous allons avoir besoin de deux bibliothèques :

- Numpy : pour charger les données et bénéficier des calculs matriciels vectorisés.
- Sklearn : qui contient tous les modèles de classifications dont nous avons besoin.

Nous avons à notre disposition un petit jeu de données composé de 1000 lignes et 20 colonnes.

Comme dans la partie 4, nous avons séparé notre jeu de données en deux avec 80% de données pour l'entraînement et 20% pour les tests. De plus nous avons normalisé les données avec la classe *StandardScaler* de *Sklearn* pour faciliter l'apprentissage. Les classes *RandomizedSearchCV* et *GridSearchCV* de *Sklearn* sont utilisées pour trouver les meilleurs hyperparamètres rapidement.

Ainsi pour ce projet nous avons ainsi choisi 4 modèles de classification différents :

- Logistic Regression
- Support Vector Classifier (SVC)
- Perceptron
- Ridge Classifier

Tout le code permettant d'avoir les résultats ci-dessous se situe dans le fichier **classification.py**.

## 5.1 Logistic Regression

C'est le premier modèle que nous avons testé. La *Logistic Regression* nécessite un peu d'"hyperparameter tuning". Nous avons donc donné les paramètres suivants à la classe *RandomizedSearchCV* pour trouver la meilleure combinaison d'hyperparamètres en testant des valeurs aléatoires dans :

- une liste de "penalty" : "l1", "l2", "elasticnet"
- une liste de "C" allant de 0,01 à 3 avec un pas de 0,01
- une liste de "solver" : "liblinear", "sag", "saga"
- une liste de "l1_ratio" allant de 0 à 1 avec un pas de 0,01

Finalement l'algorithme a trouvé la combinaison de paramètres suivante :

- "penalty" = "elasticnet"
- "solver" = "saga"
- "l1_ratio" = 0,72
- "C" = 0,29

Les résultats de l'accuracy sont les suivants :

- Train : 0,9053
- Test : 0,884

On peut voir que l'accuracy pour le jeu d'entraînement et de test sont proches de 90%. La Logistic Regression semble donc bien fonctionner avec ce jeu de données.

## 5.2 Support Vector Classifier (SVC)

Le *SVC* est le deuxième modèle de classification que nous avons choisi. Tout comme l'algorithme précédent le *SVC* nécessite de l"hyperparameter tuning". Nous avons ainsi donné les paramètres suivants à la classe *GridSearchCV* pour trouver la meilleure combinaison :

- une liste de "C" allant de 0,01 à 3 avec un pas de 0,01
- une liste de "kernel" : "linear", "rbf", "sigmoid", "poly"

L'algorithme a finalement trouvé la combinaison de paramètres suivante :

- "C" = 0,04
- "kernel" = "linear"

Avec ces paramètres nous obtenons les accuracy suivantes :

- Train : 0,9053
- Test : 0,884

Nous obtenons les mêmes valeurs d'accuracy que le modèle de Logistic Regression. Le SVC semble, aussi, bien fonctionner avec ce jeu de données.

## 5.3 Perceptron

Le troisième modèle de que nous avons sélectionné est un *Preceptron*. Cet algorithme nécessite aussi de l"hyperparameter tuning" et puisqu'il existe énormément de combinaisons de paramètres différents, nous avons usé de la classe *RandomizedSearchCV* pour trouver la meilleure combinaison de paramètres en testant des valeurs aléatoires dans :

- une liste d'"alpha" allant de 0,001 à 0,1 avec un pas de 0,001.
- une liste de "l1_ratio" allant de 0,01 à 1 avec un pas de 0,01
- une liste de "eta0" : allant de 0,01 à 2,5 avec un pas de 0,01

Finalement l'algorithme a trouvé la combinaison de paramètres suivante :

- "penalty" = "elasticnet"
- "l1_ratio" = 0,22
- "eta0" = 0,34
- "alpha" = 0,002

Les résultats de l'accuracy sont les suivants :

- Train : 0,872
- Test : 0,86

On peut voir que l'accuracy pour le jeu d'entraînement et de test sont de l'ordre de 87%. Le Perceptron semble donc bien fonctionner avec ce jeu de données.

## 5.4 Ridge Classifier

Enfin le dernier modèle que nous avons choisi est un *Ridge Classifier*. Cet algorithme comme les trois précédents nécessite un peu d'"hyperparameter tuning" afin de le rendre plus performant. Nous avons donc donné les paramètres suivants à la classe *GridSearchCV* pour trouver la meilleure combinaison :

- une liste d'"alpha" allant de 0,01 à 4 avec un pas de 0,01.
- une liste de "solver" : "svd", "cholesky", "lsqr", "sparse_cg", "sag", "saga"

Finalement l'algorithme a trouvé la combinaison de paramètres suivant :

- "alpha" = 2,78
- "solver" = "lsqr"

Les résultats de l'accuracy sont les suivants :

- Train : 0,904
- Test : 0.884

On peut voir que l'accuracy pour le jeu d'entraînement et de test sont de l'ordre de 90%. Le Ridge Classifier semble donc aussi bien fonctionner avec ce jeu de données.

## 5.5 Résultats

Nous allons maintenant comparer les résultats des différents modèles.

| | **Entraînement** | **Test** |
|---|---|---|
| **Logistic Regression** | 0.9053 | 0,884 |
| **SVC** | 0.9053 | 0,884 |
| **Perceptron** | 0,872 | 0,86 |
| **Ridge Classifier** | 0,904 | 0,884 |

Table 2: Tableau des accuracy des différents modèles

Nous pouvons voir que les modèles ont des accuracy très proches. Cependant la Logistic Regression et le SVC donnent les meilleurs résultats sur ce jeu de données que les autres modèles.