Projet académique en Modèle Linéaire

Othmane BENCHEKROUN, Thomas CAYRAT

L3 MIASHS IDS - Université Lumière Lyon 2, Mai 2022

1 Introduction

Ce projet académique s'articule autour de l'étude de formes plus extensives du modèle linéaire généralisé afin de pallier aux problèmes de l'apprentissage d'un modèle où congestionne un grand nombre de variables explicatives. On aura alors recourt aux régressions linéaires pénalisées (Lasso ou Ridge) et à d'autres modélisations prédictives à partir de la non-linéarité de certaines régressions (les régressions à base de noyau, le plus inhérent étant la régression à noyau gaussien).

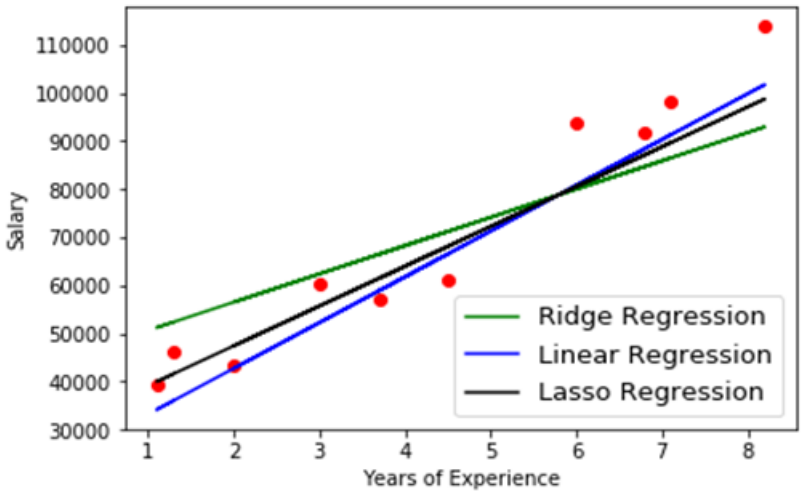

Figure 1 : Comparaison graphique entre les résidus de la droite de régression linéaire classique, de la droite de Ridge ainsi que de la droite de Lasso

Contents

2 Contexte des problèmes du modèle linéaire généralisé

Pour un nombre large de variables explicatives, une multitude de problèmes peuvent intervenir dans notre modèle :

2.1 Le sur-apprentissage/l'overfitting

Ce phénomène se déclenche souvent lorsque notre modèle est en parfaite adéquation avec les données étiquettées qu'on lui confère. Cependant ces données étant trop spécifiques, notre modèle ne peut pas autant correspondre à d'autres données, ce qui peut donner lieu à un coefficient de détermination R^2 bien trop faible par rapport aux données précédentes.

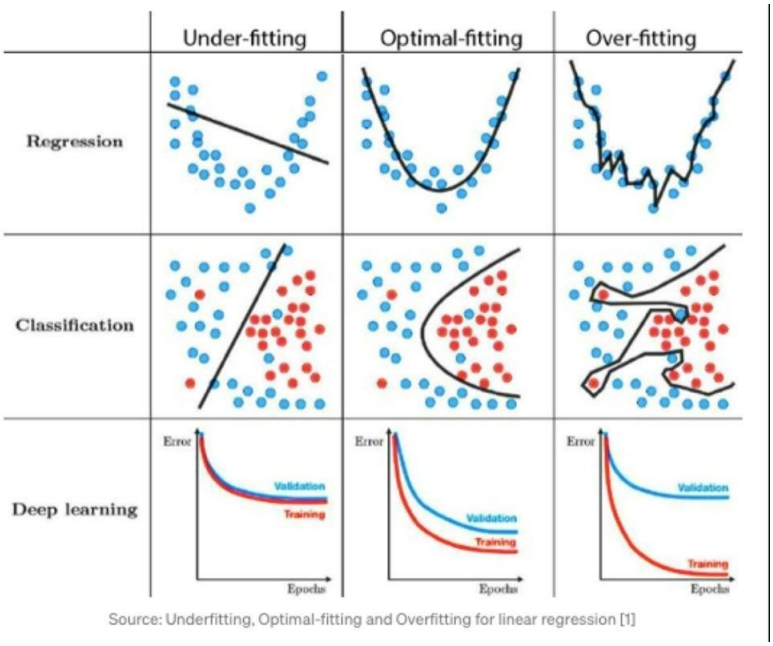

Figure 1 : Représentation de la régression et classification d'un modèle dans le cas de sous-apprentissage, de sur-apprentissage et d'optimalité d'apprentissage

2.2 La multicollinéarité

Cela intervient principalement lorsqu'il existe une forte dépendance statistiquement entre presque toutes les variables explicatives, ce qui peut générer une grande variance des coefficients et les rendre instables. Interpréter de tels coefficients peut se révéler bien ardu. Il s'agit d'un problème de conditionnement d'une matrice symétrique en analyse numérique. (cf la formule de conditionnement donnée par : $cond(X^T X) = \dfrac{\lambda_{max}}{\lambda_{min}}$)

3 Régression linéaire pénalisée

3.1 Principe de la régression Ridge et Lasso

Dans l'optique d'ajuster les coefficients de la régression multiple, on applique ce qu'on appelle la régularisation pénalisée de Ridge ou Lasso. Focalisons nous sur le cas de la régression ridge : On pénalise la somme des coefficients au carré i.e la norme euclidienne au carré des coefficients. On multiplie ce terme de pénalité par un paramètre λ qui canalise ce même terme.

La régression Lasso fonctionne de la même façon à la seule différence que le terme de pénalité sera en norme l_1 (là où dans le cas de Ridge, il s'agit de la norme l_2). Elle nous permet d'avoir une sélection de variables pertinentes en annulant les autres variables explicatives par souci de dimensionnalité.

3.2 Formulation de la solution du problème des moindres carrés (cas de la régression Ridge)

Soit le problème d'optimisation suivant :

$$F(\beta) = ||Y - X\beta||_2^2 + \lambda||\beta||_2^2$$

On rappelle que la dérivation est un opérateur linéaire, à cet effet, on a bien :

$$\forall u \in \mathbb{K}^n \ \ \forall f, g \in C(\mathbb{K}) \quad \nabla(\alpha_1 f + \alpha_2 g)(u) = \alpha_1 \nabla f(u) + \alpha_2 \nabla g(u)$$

En particulier :

$$f(\beta) = ||Y - X\beta||_2^2 \ ; \ g(\beta) = \lambda||\beta||_2^2$$

En reprenant le résultat du minimiseur du problème d'optimisation précédent :

$$\forall u \in \mathbb{K}^n \ \ \nabla f(u) = 0 \ \Leftrightarrow \ \hat{\beta} = (X^T X)^{-1} X^T Y$$

Et du fait que :

$$\nabla f(\beta) = -2X^T(Y - X\beta) \ ; \nabla g(\beta) = 2\lambda\beta$$

En effet, pour la dérivée partielle i-ème de g :

$$\frac{\partial g}{\partial \beta_i}(\beta) = 2\lambda\beta_i$$

Le résultat est immédiat :

$$\nabla F(u) = 0 \ \Leftrightarrow \ -2X^T(Y - X\beta) + 2\lambda\beta = 0 \ \Leftrightarrow \ \beta(XX^T + \lambda I_{n,p}) = X^T Y$$

Ainsi :

$$\hat{\beta}_\lambda = (X^T X + \lambda I_{n,p})^{-1} X^T Y$$

3.3 Application de la régression Ridge sur 3 jeux de données (estimation des coefficients, sommaire des informations sur les output du modèle)

3.3.1 Jeu de données : Pokemon

Avant toute chose, nous devons définir les deux variables quantitatives continues dans notre modèle à savoir la variable explicative x et la variable à expliquer y. Dans notre jeu de données, plusieurs variables à expliquer sont exploitables à partir des étiquettes de données mises en lumière par notre dataframe.

L'objectif est bien de prédire la variable à expliquer y au profit de la variable explicative x.
Un choix judicieux nous a paru d'opter pour les stats d'attaque, défense, hp (comprenant aussi bien les attaques et défenses spéciales) comme variable explicative afin d'expliquer la variable vitesse.

https://www.kaggle.com/rounakbanik/pokemon

```
# Le jeu de données pokemon
pokemon <- read.csv("pokemon.csv")
View(pokemon)

# noms des colonnes
colnames(pokemon)

# variable a definir
y=pokemon$speed

# matrice des variables predictives
x = data.matrix(pokemon[,c("attack","defense","hp","sp_attack","sp_defense")])
```

Figure 1 : Chargement du jeu de données et choix adéquat de la variable explicative x et de la variable à expliquer y

Nous effectuerons à partir du package glmnet le calcul des coefficients de notre modèle linéaire régularisé.

```
# estimation des coefficients du modele de regression
model = glmnet(x, y, alpha=0, standardize = F)
```

Figure 2 : Application de la régularisation de notre modèle via la commande glmnet

Une fois notre modèle établi, nous nous pencherons sur les valeurs calculées de λ grâce à l'information donnée par les déviances résiduelles.

	Df	%Dev	Lambda
1	5	0.00	410100
2	5	4.94	373600
3	5	5.33	340400
4	5	5.75	310200
5	5	6.19	282600
6	5	6.65	257500
7	5	7.13	234600
8	5	7.64	213800
9	5	8.17	194800
10	5	8.73	177500
78	5	29.36	318
79	5	29.36	289
80	5	29.36	264
81	5	29.36	240
82	5	29.36	219
83	5	29.36	199
84	5	29.36	182
85	5	29.37	166
86	5	29.37	151
87	5	29.37	137
88	5	29.37	125
89	5	29.37	114

Figure 3 : Description des 10 premières et dernières valeurs de λ, des déviations résiduelles (ou nulles) de notre modèle

A partir du sommaire des sorties de notre modèle, on remarque que le facteur λ tend à être minimisé au fur et à mesure qu'on réalise plusieurs déviances résiduelles.
Rappelons tout d'abord que la qualité d'un modèle se fait principalement à partir de la définition de la déviance.
Une telle quantité se calcule à partir de la notion de log-vraisemblance vue en cours (pas très différent du critère AIC) :

$$D(y, \hat{\mu}) = 2(\log\,(p(y \mid \hat{\theta_s}) - \log\,(p(y \mid \hat{\theta_0}))$$

avec θ_s et θ_0 représentant respectivement : le modèle entrainé et le modèle nul.

Plus la déviance résiduelle est faible, plus on a de fortes chances que notre modèle soit approprié dans la mesure où les données expliquent parfaitement notre variable à prédire.

Qu'en est-il des coefficients de notre modèle linéaire pénalisé ?
Pour ce faire, on réalise un affichage des coefficients pour une valeur λ donnée :

```
# Test des coefficients avec un lambda a 0.1
coef(model, s = 0.1)
```

Figure 4 : Commande permettant de déterminer le vecteur coefficients du modèle

```
> coef(model, s = 0.1)
6 x 1 sparse Matrix of class "dgCMatrix"
                    s1
(Intercept) 36.0134361
attack       0.3308899
defense     -0.2816550
hp          -0.1019833
sp_attack    0.2972188
sp_defense   0.1545776
```

Figure 5 : Valeur des coefficients pour chaque variable explicative

3.3.2 Jeu de données : Fifa 19

Dans ce jeu de données, on reprend les mêmes étapes parcourues dans le jeu de données précédents.
On tâchera de préparer et de nettoyer le dataset des éventuels valeurs redondantes ou aberrantes et on charge le fichier csv.
Notre variable y à expliquer n'est autre que l'âge des joueurs à partir des variables explicatives x agrégeant les stats globales, potentiel, réputation internationale de chaque joueur.

https://www.kaggle.com/datasets/karangadiya/fifa19

```
# Le jeu de données fifa
fifa <- read.csv("~/L3_MIASHS_IDS/Modele Lineaire/fifa_data.csv")
fifa = na.omit(fifa)
View(fifa)

# variable a definir
y = fifa$Age

# matrice des variables predictives
x = data.matrix(fifa[,c("Overall","Potential","Special","International.Reputation")])
View(x)
```

Figure 1 : Chargement et préparation des données du dataframe Fifa 19

La régularisation de notre modèle se fait à nouveau comme suit :

```
# Model d'estimation
model = glmnet(x, y, alpha=0, standardize = F)
print(model)
```

Figure 2 : Régularisation de notre modèle linéaire

Et ses informations sommaires nous en disent long sur l'évolution du paramètre de pénalité λ en contre-partie de la déviance résiduelle :

```
> print(model)

Call:  glmnet(x = x,

    Df  %Dev Lambda
1    4  0.00 302000
2    4  4.55 275200
3    4  4.67 250700
4    4  4.77 228500
5    4  4.87 208200
6    4  4.96 189700
7    4  5.05 172800
8    4  5.13 157500
9    4  5.20 143500
10   4  5.26 130700
90   4 53.47     77
91   4 55.22     70
92   4 56.91     64
93   4 58.53     58
94   4 60.07     53
95   4 61.53     48
96   4 62.90     44
97   4 64.17     40
98   4 65.36     36
99   4 66.45     33
100  4 67.46     30
```

Figure 3 : Description des 10 premières et dernières valeurs des déviations résiduelles (ou nulles) de notre modèle

Les 100 itérations de calcul de déviance résiduelle nous laisse augurer d'ores et déjà que $\lambda_{min} \simeq 30$. Cette information nous confortera davantage à l'idée d'obtenir le même résultat par la méthode de cross-validation dans la prochaine section.

3.3.3 Jeu de données : Consommation d'alcool des étudiants

A nouveau, on fusionne nos deux datasets et on prépare les données en les nettoyant des valeurs aberrantes.
Il s'ensuivra par la suite le choix des variables explicatives qui constitueront la matrice design X ainsi que le vecteur Y à prédire.

https://www.kaggle.com/datasets/uciml/student-alcohol-consumption

```
# Le jeu de données student
student_mat <- read.csv("~/L3_MIASHS_IDS/Modele Lineaire/student-mat.csv")
student_por <- read.csv("~/L3_MIASHS_IDS/Modele Lineaire/student-por.csv")
student <- merge(student_mat,student_por,by=c("school","sex","age","address"
      ,"famsize","Pstatus","Medu","Fedu","Mjob","Fjob","reason","nursery","internet"))
student = na.omit(student)
View(student)

colnames(student)

# variable a definir
y = student$age

# matrice des variables predictives
x = data.matrix(student[,c("Dalc.x","Walc.x","absences.x","health.x")])
```

Figure 1 : Chargement et préparation des données du dataframe Alcohol Student

La régularisation de notre modèle est donné comme suit :

```
# Model d'estimation
model = glmnet(x, y, alpha=0, standardize = F)
print(model)
```

Figure 2 : Régularisation de notre modèle linéaire

Pour plus d'informations, inspectons les valeurs actualisées du paramètre de pénalité λ en parallèle des valeurs de la déviation résiduelle :

```
Call:  glmnet(x = x,

    Df %Dev  Lambda
1    4 0.00 1124.00
2    4 0.20 1024.00
3    4 0.22  933.40
4    4 0.24  850.40
5    4 0.26  774.90
6    4 0.29  706.10
7    4 0.31  643.30
8    4 0.33  586.20
9    4 0.36  534.10
10   4 0.39  486.70
90   4 3.87    0.28
91   4 3.87    0.26
92   4 3.88    0.24
93   4 3.88    0.22
94   4 3.88    0.20
95   4 3.88    0.18
96   4 3.89    0.16
97   4 3.89    0.15
98   4 3.89    0.14
99   4 3.89    0.12
100  4 3.89    0.11
```

Figure 3 : Description des 10 premières et dernières valeurs des déviations résiduelles (ou nulles) de notre modèle

Voilà qui est rassurant ! La déviation résiduelle demeure bien faible, ce qui nous donne un aperçu du fait que notre modèle performera bien probablement en terme de prédiction. D'autre part, on remarque bien que $\lambda_{min} \approx 0$, ce qui nullifiera sans doute la somme des carrés des pondérations.
La régression obtenue sera alors identique à une régression linéaire simple.

3.4 Choix optimal du paramètre λ par la méthode k-fold cross-validation

3.4.1 Principe de la méthode k-fold cross-validation

Nous avions extrapolé dans l'un des paragraphes précédents les nombreuses contraintes que nous génèrent le modèle linéaire multiple entre autres le sur-apprentissage et la multicolinéarité, ce qui rendait notre modèle instable et bien poreux en terme de prédiction.
Ceci s'expliquait le plus souvent à cause des variances du modèle.
Et quand il est question de variance, nous pouvons alors inspecter l'erreur quadratique moyenne MSE des estimateurs ridge, qui au bout du compte se révèle bien plus faible que l'erreur quadratique moyenne des estimateurs des moindres carrées.

L'erreur quadratique moyenne s'exprime canoniquement comme suit :

$$MSE(\hat{\beta}) = Var(\hat{\beta}) + Biais(\hat{\beta})^2$$

Plus généralement, dans le cas du modèle linéaire multivarié de Ridge, nous avons bien :

$$\begin{aligned} MSE(\hat{\beta}_\lambda) &= \mathbb{E}[\ ||\hat{\beta}_\lambda - \beta_\lambda||^2 \mid X] \\ &= \ \mathrm{Tr}(Var[\ \hat{\beta}_\lambda \mid X\ ]) + ||Biais(\hat{\beta}_\lambda \mid X)||^2 \end{aligned}$$

De plus, l'espérance de l'estimateur ridge est sans biais et sa matrice de variance/covariance s'exprime comme suit :

$$\mathbb{E}[\ \hat{\beta}_\lambda \mid X] = \hat{\beta}_\lambda$$

$$Var[\ \hat{\beta}_\lambda \mid X] = \sigma^2 (X^T X + \lambda I_{n,p})^{-1} X^T X (X^T X + \lambda I_{n,p})^{-1}$$

Une fois les paramètres de l'estimateur établis, on peut remanier l'erreur quadratique moyenne de cet estimateur de telle sorte à ce qu'il existe un λ qui satisfait de telles conditions.

3.4.2 Procédure de choix du paramètre de pénalité λ minimal et régularisé

Retrouver un paramètre de pénalité à variance minimal se fait en suivant la démarche suivante :

- Tirer un échantillon de p-valeurs $\lambda_1, ..., \lambda_p$ comme paramètre de pénalité

- Pour $i = 1, ..., N$ on exclue la $i^{ème}$ observation (x_i, y_i) et on calcule parallèlement p-estimateurs de ridge notés $\hat{\beta}_{\lambda_p, i}$ relatifs aux $N - 1$ observations restantes

- Déterminer par calcul la prédiction des p-variables à expliquer :

$$\hat{y}_{\lambda_p,i} = x_i \hat{\beta}_{\lambda_p,i}$$

- Calculer l'erreur quadratique moyenne des p-estimateurs ridge :

$$MSE = \frac{1}{n} \sum_{i=1}^{N} (y_i - \hat{y}_{\lambda_p,i})^2$$

- Choisir un paramètre de pénalité λ optimal en minimisant l'erreur quadratique moyenne des prédictions :

$$\lambda^* = \underset{\lambda_p \in \mathbb{K}}{\arg\min}\, MSE$$

(Plusieurs algorithmes numériques de recherche d'optimiseurs peuvent entrer en jeu : descente de gradient, méthode de Newton-Raphson...)

3.5 Analyse des résultats finaux

3.5.1 Jeu de données : Pokémon

La commande cv.glmnet() nous permet de présenter en aval les résultats de la procédure cross-validation expliquée précédemment.
On obtient sans peine les valeurs du paramètre de pénalité λ minimal ainsi que λ régularisé.

```
# objectif : trouver le lambda optimal tel qu'il
# minimise l'Erreur Quadratique Moyenne

# grace a la procedure dite "validation croisee"
# on arrive a faire la "Moyenne des Erreurs de prediction
# sur une nouvelle donneee" :
cv_model = cv.glmnet(x, y, alpha=0, standardize = F)
```

Figure 1 : Application de la cross-validation sur le paramètre de pénalité λ

```
> cv_model

Call:  cv.glmnet(x = x, y = y, alpha = 0, standardize = F)

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min    420    75   599.4 27.65       5
1se  11953    39   625.1 23.58       5
```

Figure 2 : Résultats de la cross-validation sur le paramètre de pénalité λ

Quelle différence foncière existe-elle entre λ_{min} et λ_{1se} ?
λ_{min} représente bien effectivement la valeur obtenue en minimisant l'erreur quadratique moyenne des prédictions tandis que λ_{1se} fait office du paramètre de pénalité correspondant au modèle le plus "régularisé".

Illustrons ceci par la figure suivante :

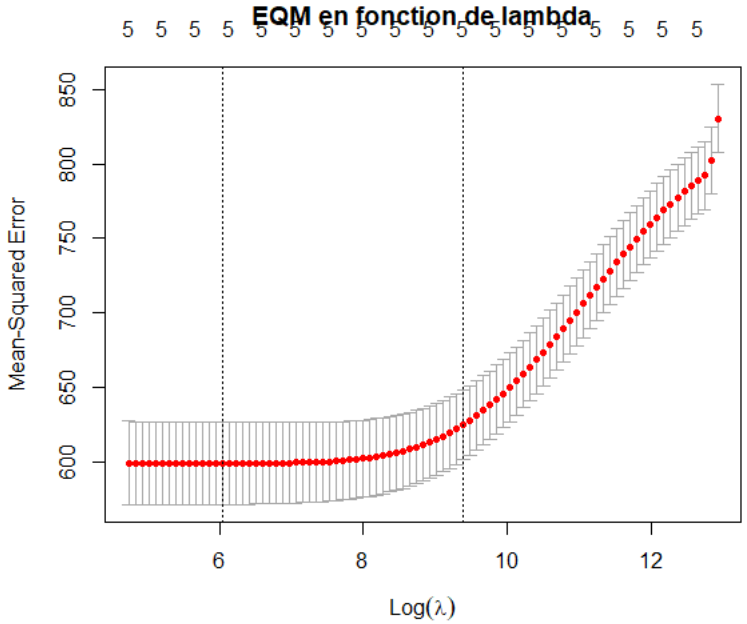

Figure 3 : Représentation de l'erreur quadratique moyenne MSE en fonction de λ

Il s'agit de la courbe de cross-validation délimitée par l'écart-type le plus bas ainsi que l'écart-type le plus haut. Comme prévu, on y retrouve également les deux valeurs λ_{min} et λ_{1se}.

Nous pouvons à présent obtenir le vecteur coefficients relatif au meilleur λ calculé à partir de la démarche de cross-validation :

```
> coef(best_model, s = "best_lambda")
6 x 1 sparse Matrix of class "dgCMatrix"
                      s1
(Intercept) 60.594213019
attack       0.014115478
defense     -0.003161633
hp           0.005166500
sp_attack    0.017998435
sp_defense   0.008737927
```

Figure 3 : Coefficients du modèle suite au choix du "meilleur" λ

D'autre part, à partir de λ_{min}, on peut sans peine prédire une valeur $\hat{y}_i$:

```
> p_conf = predict(best_model, newx = z, s = "lambda.min", interval = "confidence")
> p_conf
           s1
[1,] 63.26931
```

Figure 4 : Prédiction d'une valeur $\hat{y}_i$ à partir du paramètre de pénalité λ_{min}

En passant par une estimation par intervalle de confiance plutôt qu'un intervalle de prédiction, on obtient à peu près la même valeur prédite :

```
> # prediction des valeurs de y disponible avec predict (intervalle : predict) :
> p_pred = predict(best_model, newx = z, s = "best_lambda", interval = "prediction") ; p_pred
           s1
[1,] 63.18005
```

Figure 5 : Estimation d'une valeur prédite $\hat{y}_i$ par intervalle de confiance à partir du paramètre de pénalité λ_{min}

Quoiqu'il en soit, ses deux intervalles nous donnent bien la valeur peu ou prou prédite à une légère différence près : Notons bien qu'une estimation par intervalle de confiance néglige l'incertitude sur la valeur prédite et offre un intervalle moins large qu'un intervalle de prédiction.

Pour rappel, la construction d'un intervalle de prédiction d'une valeur à expliquer se fait dans le cas d'une régression ridge en considérant l'hypothèse :

$$\hat{y}_{pred} \sim \mathcal{N}(y_{pred}, \sigma^2 x_{pred}^t (X^T X + \lambda I_{n,p})^{-1} x_{pred})$$

De ce faisant, on centre et on réduit notre variable à prédire.
On remplace la variance par son estimateur s^2 et notre statistique de test est donc donnée par :

$$\frac{\hat{y}_{pred} - y_{pred}}{s\sqrt{x_{pred}^t (X^T X + \lambda I_{n,p})^{-1} x_{pred}}} \sim T_{n-p-1}$$

On en déduit sans peine l'intervalle de prédiction suivant :

$$IC(y_{pred}) = \left[\hat{y}_{pred} \pm t_{1-\frac{\alpha}{2}, n-p-1} s\sqrt{x_{pred}^t (X^T X + \lambda I_{n,p})^{-1} x_{pred}}\right]$$

3.5.2 Jeu de données : Fifa 19

L'objectif de la cross-validation étant de régulariser et de minimiser le paramètre de pénalité λ, on obtient alors :

```
> cv_model = cv.glmnet(x, y, alpha=0, standardize = F)
> print(cv_model)

Call:  cv.glmnet(x = x, y = y, alpha = 0, standardize = F)

Measure: Mean-Squared Error

    Lambda Index Measure      SE Nonzero
min   30.2   100    7.11 0.09966       4
1se   30.2   100    7.11 0.09966       4
```

Figure 1 : Obtention de λ_{min} et λ_{1se} de notre régression ridge

Le paramètre de pénalité minimal λ_{min} étant exactement le même que le paramètre de pénalité régularisé λ_{1se}, la fonction MSE en fonction de l'échelle logarithmique de λ sera strictement croissante dans tout son domaine.

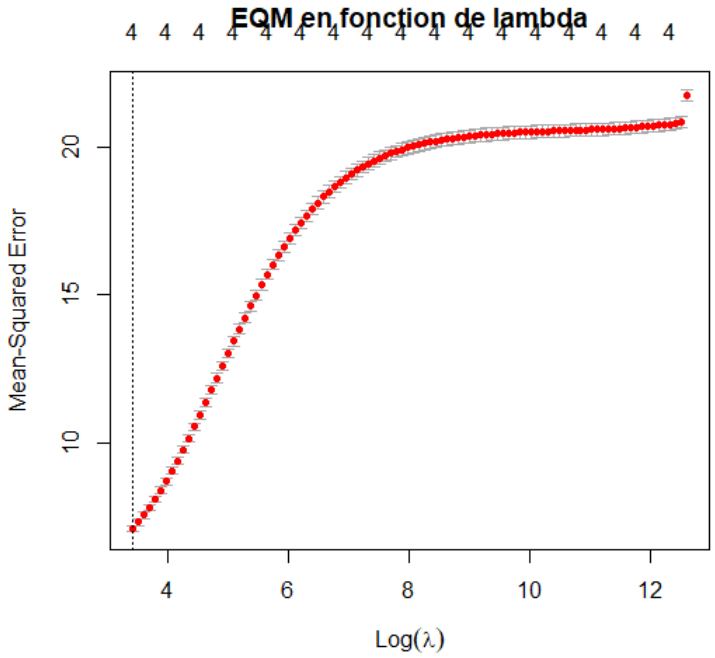

Figure 2 : Représentation graphique de l'erreur quadratique moyenne MSE en fonction du paramètre de pénalité λ

Le chevauchement de l'écart-type le plus bas et le plus haut tend à se confondre exactement avec le tracé de chaque valeur discrétisée de l'erreur quadratique moyenne MSE en remarquant aussi bien que graphiquement $\lambda_{min} = \lambda_{1se}$. On peut enfin opter pour notre paramètre de pénalité λ optimal.

En revanche pour un tel λ adéquat, la déviance résiduelle doit être faible (dans l'ordre de 10 comme valeur). Notre modèle est bien en mesure de prédire une valeur quelconque de notre variable à expliquer à partir de nouvelles valeurs de variables explicatives.

```
> # calcul des coefficients du modele par regression ridge
> best_model = glmnet(x, y, alpha = 0, lambda = best_lambda) ; best_model

Call:  glmnet(x = x, y = y, alpha = 0, lambda = best_lambda)

  Df %Dev Lambda
1  4 9.84   30.2
```

Figure 3 : Informations sur notre modèle performant

On en tire par la suite le vecteur coefficients associé à un tel modèle :

```
> # Coefficients des variables de x correspondant au lambda optimal
> coef(best_model, s = "best_lambda")
5 x 1 sparse Matrix of class "dgCMatrix"
                                    s1
(Intercept)              23.5476605610
Overall                   0.0405968794
Potential                -0.0320451845
Special                   0.0004758343
International.Reputation  0.3673480567
```

Figure 4 : Valeurs du vecteur coefficient de notre modèle performant

La prédiction d'une valeur quelconque $\hat{y}_{pred}$ à partir des valeurs des variables explicatives suivantes :

```
# Nouvelles donnees sur lesquelles on va predire y :
Z=c(84,85,1850,4)
```

Figure 5 : Valeurs des variables explicatives qui prédiront une valeur quelconque de notre variable à expliquer

Le résultat obtenu est donc :

```
> p_pred = predict(best_model, newx = z, s = "best_lambda", interval = "prediction") ; p_pred
          s1
[1,] 26.58364
```

Figure 6 : Valeur prédite de la variable à expliquer y_{pred}

3.5.3 Jeu de données : Consommation d'alcool des étudiants

Remarquons tout d'abord que la valeur λ_{1se} semble beaucoup trop grande en comparaison avce λ_{min} :

```
> cv_model = cv.glmnet(x, y, alpha=0, standardize = F)
> print(cv_model)

Call:  cv.glmnet(x = x, y = y, alpha = 0, standardize = F)

Measure: Mean-Squared Error

    Lambda Index Measure      SE Nonzero
min    3.2    64   1.364 0.08530       4
1se 1124.2     1   1.378 0.08557       4
```

Figure 1 : Obtention de λ_{min} et λ_{1se} de notre régression ridge

Une fonction convexe de MSE est alors exprimée en fonction de λ :

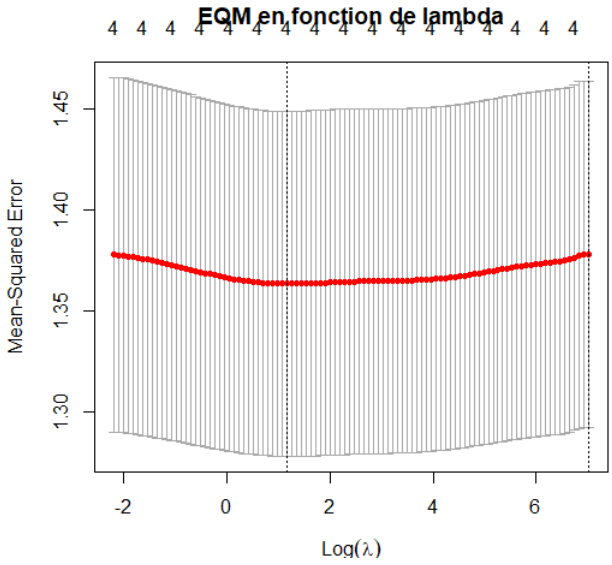

Figure 2 : Représentation graphique de l'erreur quadratique moyenne MSE en fonction du paramètre de pénalité λ

```
> # Coefficients des variables de x correspondant au lambda optimal
> coef(best_model, s = "best_lambda")
5 x 1 sparse Matrix of class "dgCMatrix"
                    s1
(Intercept) 16.48916457
Dalc.x       0.03634604
Walc.x       0.03167839
absences.x   0.00471532
health.x    -0.01499422
```

Figure 3 : Valeurs du vecteur coefficient de notre modèle performant

```
# Nouvelles donnees sur lesquelles on va predire y :
z=c(3,5,10,2)
```

Figure 4 : Valeurs des variables explicatives qui prédiront une valeur quelconque de notre variable à expliquer

```
> # prediction des valeurs de y disponible avec predict (intervalle de confiance) :
> p_conf = predict(best_model, newx = z, s = "best_lambda", interval = "confidence") ; p_conf
          s1
[1,] 16.77376
```

Figure 5 : Valeur prédite de la variable à expliquer y_{pred} graphique de l'erreur quadratique moyenne MSE en fonction du paramètre de pénalité λ

4 Régression à noyau

4.1 Principe de la régression à noyau

Dans cette partie, nous imbriquons une autre façon de rendre la régression linéaire classique plus "flexible" en opérant un changement de variable de type $\phi(x)$.
Etant donné que nous nous plaçons dans le cas où nos variables explicatives sont de dimension infinie et donc ne sont pas nécessairement des éléments de $\mathbb{R}^d$ mais bel et bien d'autres fonctions définies dans un espace E différent de $\mathbb{R}^d$ avec E un espace préhilbertien (espace vectoriel muni d'un produit scalaire à la seule différence qu'il n'est pas de dimension finie).
Pour en revenir sur des modèles de régressions à noyaux, l'une des plus édifiantes que l'on peut déclarer en particularisant le type de transformation de ϕ :

- $\phi_k(x) = x^k$, fonction intervenante dans la régression polynomiale.
- $\phi(x) = \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$, largement utilisé dans les noyaux gaussiens.

A titre d'exemple, représentons l'image des points (1, 2, 3), (3, 2, 0) et (2, 3, 2) par les transformations suivantes :

On désigne respectivement :
- l'application linéaire de type $\phi : x \mapsto AX$
- l'application gaussienne décrite par $\psi : x \mapsto \frac{1}{2\sqrt{2\pi}}\exp(-\frac{||z-x||_2^2}{2.2^2})$
- Pour $A = \begin{pmatrix} 1 & 0 & -3 \\ 3 & -2 & 1 \end{pmatrix}$; $X = \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix}$:

$$\phi(x) = AX = \begin{pmatrix} 1 & 0 & -3 \\ 3 & -2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix} = \begin{pmatrix} -8 \\ 10 \end{pmatrix}$$

$$\psi(x) = \frac{1}{2\sqrt{2\pi}} \exp(-\frac{||z-x||_2^2}{2.2^2}) = \frac{1}{2\sqrt{2\pi}} \exp(-\frac{\sum_{i=1}^{3}(z-x_i)^2}{2.2^2}) = \frac{1}{2\sqrt{2\pi}} \exp(-\frac{3z^2-4z+14}{2.2^2})$$

- Pour $A = \begin{pmatrix} 1 & 0 & -3 \\ 3 & -2 & 1 \end{pmatrix}$; $X = \begin{pmatrix} -3 \\ 2 \\ 0 \end{pmatrix}$:

$$\phi(x) = AX = \begin{pmatrix} 1 & 0 & -3 \\ 3 & -2 & 1 \end{pmatrix} \begin{pmatrix} -3 \\ 2 \\ 0 \end{pmatrix} = -\begin{pmatrix} 3 \\ 13 \end{pmatrix}$$

$$\psi(x) = \frac{1}{2\sqrt{2\pi}} \exp(-\frac{\sum_{i=1}^{3}(z-x_i)^2}{2.2^2}) = \frac{1}{2\sqrt{2\pi}} \exp(-\frac{3z^2+2z+13}{2.2^2})$$

- Pour $A = \begin{pmatrix} 1 & 0 & -3 \\ 3 & -2 & 1 \end{pmatrix}$; $X = \begin{pmatrix} 2 \\ 3 \\ 2 \end{pmatrix}$:

$$\phi(x) = AX = \begin{pmatrix} 1 & 0 & -3 \\ 3 & -2 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} -4 \\ 2 \end{pmatrix}$$

$$\psi(x) = \frac{1}{2\sqrt{2\pi}} \exp(-\frac{\sum_{i=1}^{3}(z - x_i)^2}{2.2^2}) = \frac{1}{2\sqrt{2\pi}} \exp(-\frac{3z^2 - 14z + 17}{2.2^2})$$

Pour avoir plus de visibilité d'un point de vue analytique, représentons l'ensemble de l'image de ces points à l'aide de fonctions sur R comme suit :

Pour la transformation linéaire, on obtient tout compte fait un point dans l'espace (ou plutôt un vecteur de $\mathbb{R}^2$).

```r
# Fonction Transformation Linéaire
transf_lineaire <- function(x){
  matrix(c(1,0,-3,3,-2,1),nrow=2,ncol=3,byrow=TRUE) %*% x
}
```

Figure 1 : Fonction permettant d'implémenter le calcul de l'image par l'application linéaire $\phi = AX$

```r
P1 = as.vector(transf_lineaire(c(1,-2,3)))
P2 = as.vector(transf_lineaire(c(-3,2,0)))
P3 = as.vector(transf_lineaire(c(2,3,2)))
plot(P1[1],P1[2],xlim=c(-15,15),ylim=c(-15,15),col=1, main="Représentation de l'image de chaque point
     par une transformation linéaire")
points(P2[1],P2[2],col=2)
points(P3[1],P3[2],col=3)
legend("topright",legend=c("c(1,-2,3)","c(-3,2,0)","c(2,3,2)"),pch=1,col=c(1,2,3))
```

Figure 2: Déclaration du code de représentation de l'image des 3 points par ϕ

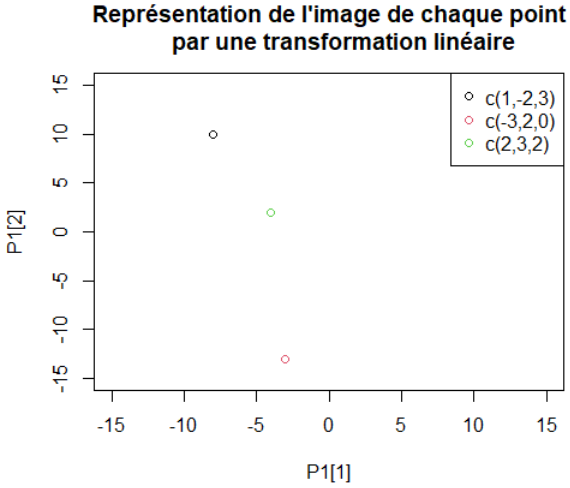

Figure 3: Représentation graphique de l'image des 3 points par ϕ

Pour la transformation gaussienne, nous obtenons en sortie une fonction gaussienne de paramètre z représentée dans le repère orthornormé via le programme R suivant :

```
f1 = function(x){(1/sqrt(2*pi*2**2))*exp(-(3*x**2-4*x+14)/(2*2**2))}
f2 = function(x){(1/sqrt(2*pi*2**2))*exp(-(3*x**2+2*x+13)/(2*2**2))}
f3 = function(x){(1/sqrt(2*pi*2**2))*exp(-(3*x**2-14*x+17)/(2*2**2))}
c3 = curve(f3,from=-5, to=7,col=3, main="Représentation de l'image de chaque fonction
     par une transformation gaussienne")
c2 = curve(f2,from=-5, to=7, add=T,col=2)
c1 = curve(f1,from=-5, to=7, add=T,col=1)
legend("topright",legend=c("c(1,-2,3)","c(-3,2,0)","c(2,3,2)"),pch=1,col=c(1,2,3))
```

Figure 4 : Fonction permettant d'implémenter le calcul de l'image par l'application gaussienne ψ

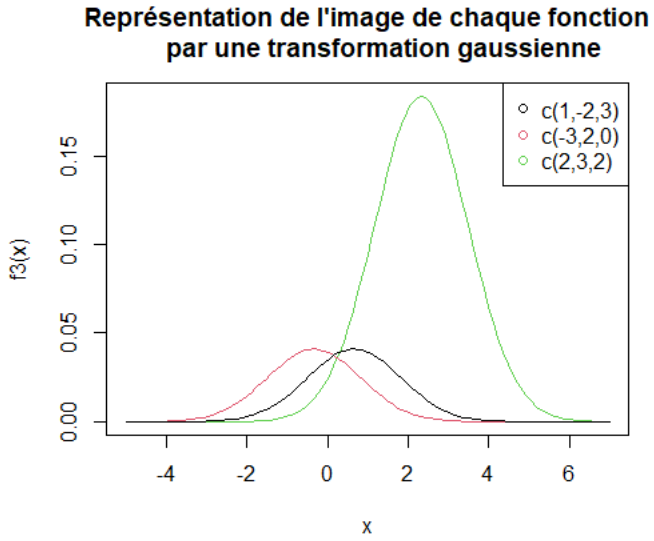

Figure 5: Représentation graphique de l'image des 3 points par ψ

De façon générale, nous pouvons générer autant de points que l'on veut à partir de ces transformations décrites.
Le code R suivant permet de retranscrire ceci :

```
# Matrice de résultats par transformation linéaire
ML <- matrix(rep(NA,2*length(1:3)^3),nrow=2)

# Matrice de résultats par transformation gaussienne
MG <- rep(NA,27)

col <- 0
for(i in 1:3){
  for(j in 1:3){
    for(k in 1:3){
      col <- col + 1
      ML[,col] <- transf_lineaire(c(i,j,k))
    }
  }
}
plot(as.vector(ML),main="Représentation de différentes images de points
     (c(1,1,1) : c(3,3,3))")
```

Figure 4 : Fonction générant un ensemble de points par l'application ϕ et ψ

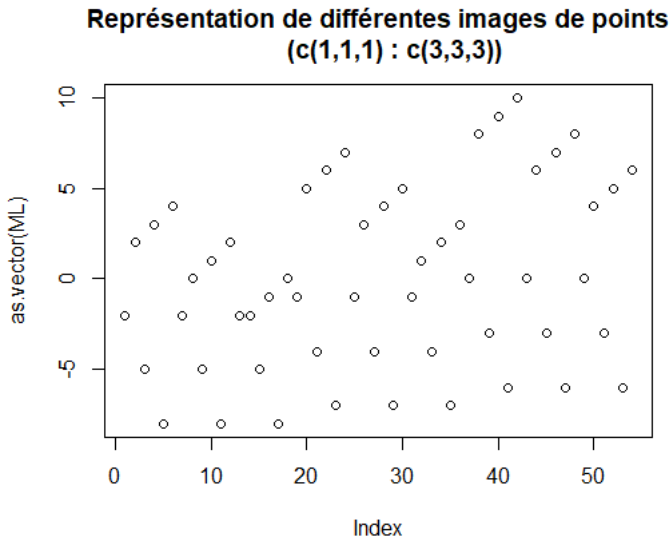

Figure 5: Représentation graphique de l'ensemble des points par ϕ

4.2 Formulation du problème de moindres carrés dans le cas d'une régression non linéaire

Après ce changement de variable, la régression linéaire peut désormais se réécrire à partir des valeurs prises par le noyau $K(x, x') = \langle \phi(x), \phi(x') \rangle$ sur les données d'apprentissage.
Ainsi, la régression qui minimise l'erreur empirique se calcule par la résolution d'un problème d'optimisation convexe.

Cette régression s'écrit sous la forme :

$$y = \langle \beta, \phi(x) \rangle + \epsilon$$

Ce qui permet alors de démarquer sans peine le même problème dual que le précédent :

$$\min_{b \in \mathbb{K}^n} \sum_{i=1}^{n} (y_i - b^T \phi(x_i))^2$$

On obtient tout compte fait la solution au même problème d'optimisation :

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

4.3 Formulation du problème de ridge dans le cas d'une régression non linéaire

Qu'en est-il d'une version améliorée de la solution au problème de Ridge ?

Pour répondre à cette question, nous introduisons le théorème du représentant qui énonce que pour tout noyau K symétrique défini positif avec H un espace de Hilbert à noyau reproducteur correspondant, il est possible de représenter le problème dual régularisé défini sur un noyau de reproduction par le produit scalaire de ce noyau du même espace.

Autrement dit, la formulation suivante du problème dual de la fonction ψ strictement croissante et convexe :

$$\arg\min_{f \in H} \psi(f(x_1), ..., f(x_n), ||f||_H)$$

admet pour représentation :

$$f(x) = \sum_{i=1}^{n} \alpha_i \, \phi(x_i)$$

Preuve :

Nous définissons l'espace vectoriel H_s engendré par $(f(x_1), ..., f(x_n))$:

$$H_s = Vec(f(x_1), ..., f(x_n)) = \left\{ f(x) = \sum_{i=1}^{n} \alpha_i \, \phi(x_i) \ / \ \alpha_1, ..., \alpha_n \in \mathbb{R} \right\}$$

et $H_s^{\perp}$ l'espace orthogonal de H_s dans H.

H_s est un sous-espace vectoriel de dimension finie, donc toute fonction $f \in H_s$ est décomposable de manière unique :

$$f = f_s + f_{\perp}$$

Cette définition est tirée de la notion de supplémentarité entre H_s et $H_s^{\perp}$ d'où la décomposition unique de H en ces deux espaces :

$$H = H_s + H_s^{\perp}$$

Ou encore à partir d'une somme directe :

$$H = H_s \oplus H_s^{\perp}$$

Le théorème de Pythagore permet sans peine de vérifier que :

$$||f||_H^2 = ||f||_{H_s}^2 + ||f||_{\perp}^2$$

Ceci est dû au fait que :

$$\forall i = 1, ..., n \ \ ; \ \ f_\perp(x_i) = \langle f_\perp, \phi(x_i) \rangle_H = 0$$

En injectant cette simplification dans la fonction à minimiser, cette nouvelle expression est bien inférieure par rapport à l'expression précédente, ce qui achève que la solution au problème dual peut bien être représentée comme suit :

$$f \in H_s \ \ ; \ \ f(x) = \sum_{i=1}^{n} \alpha_i \ \phi(x_i)$$

Nous pourrons retravailler la formulation du problème de Ridge autrement en homologuant un problème sur $b \in H$ en un problème de $\alpha \in H_s$.

Le paramètre à minimiser ne sera donc plus explicitement le vecteur b mais bel et bien un vecteur $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{R}^n$.

Nous réécrivons le problème dual comme suit :

$$\begin{aligned}
\sum_{i=1}^{n} (y_i - \langle b, \phi(x_i) \rangle)^2 + \lambda ||b||^2 &= \sum_{i=1}^{n} \left(y_i - \left\langle \sum_{i'=1}^{n} \alpha_{i'} \ \phi(x_{i'}), \phi(x_i) \right\rangle \right)^2 + \lambda \left|\left| \sum_{i'=1}^{n} \alpha_{i'} \ \phi(x_{i'}) \right|\right|_2^2 \\
&= \sum_{i=1}^{n} \left(y_i - \sum_{i'=1}^{n} \langle \alpha_{i'} \ \phi(x_{i'}), \phi(x_i) \rangle \right)^2 + \lambda \left|\left| \sum_{i'=1}^{n} \alpha_{i'} \ \phi(x_{i'}) \right|\right|_2^2 \\
&= \sum_{i=1}^{n} \left(y_i - \sum_{i'=1}^{n} \langle \alpha_{i'} \ \phi(x_{i'}), \phi(x_i) \rangle \right)^2 + \lambda \sum_{i',i''=1}^{n} \alpha_{i'} \alpha_{i''} \ \langle \phi(x_{i'}), \phi(x_{i''}) \rangle
\end{aligned}$$

A-t-on réellement l'égalité suivante :

$$\left|\left| \sum_{i'=1}^{n} \alpha_{i'} \ \phi(x_{i'}) \right|\right|_2^2 = \sum_{i',i''=1}^{n} \alpha_{i'} \alpha_{i''} \ \langle \phi(x_{i'}), \phi(x_{i''}) \rangle$$

Pour ce faire, il suffira d'exploiter les propriétés de la bilinéarité du produit scalaire :

$$\begin{aligned}
\left|\left| \sum_{i'=1}^{n} \alpha_{i'} \ \phi(x_{i'}) \right|\right|_2^2 &= \left\langle \sum_{i'=1}^{n} \alpha_{i'} \ \phi(x_{i'}), \sum_{i''=1}^{n} \alpha_{i''} \ \phi(x_{i''}) \right\rangle \\
&= \sum_{i'=1}^{n} a_{i'} \left\langle \phi(x_{i'}), \sum_{i''=1}^{n} \alpha_{i''} \ \phi(x_{i''}) \right\rangle \\
&= \sum_{i'=1}^{n} \sum_{i''=1}^{n} a_{i'} a_{i''} \ \langle \phi(x_{i'}), \phi(x_{i''}) \rangle
\end{aligned}$$

$$= \sum_{i',i''=1}^{n} a_{i'} a_{i''} \, \langle \phi(x_{i'}), \alpha_{i''} \, \phi(x_{i''}) \rangle$$

Une fois en ayant posé la définition du noyau K défini par $K(x, x') = \langle \phi(x), \phi(x') \rangle$, le lagrangien s'écrira à nouveau comme suit :

$$\min_{\alpha \in \mathbb{K}^n} ||y - K\alpha||_2^2 + \lambda \, \alpha^T K \alpha$$

avec K une matrice de Gram $n \times n$ donnée par :

$$K = \begin{pmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \dots & \langle \phi(x_1), \phi(x_n) \rangle \\ \vdots & \ddots & \vdots \\ \langle \phi(x_n), \phi(x_1) \rangle & \dots & \langle \phi(x_n), \phi(x_n) \rangle \end{pmatrix}$$

et $||f||_{H_s}^2$ une forme quadratique définie par : $||f||_{H_s}^2 = \alpha^T K \alpha$

On pose alors le problème d'optimisation suivant ;

$$f(\alpha) = ||y - K\alpha||_2^2 + \lambda \, \alpha^T K \alpha$$

On y retrouvera de façon immédiate la solution au problème de Ridge comme préconisé dans la première partie :

$$\nabla f(\alpha) = 0 \;\Leftrightarrow\; -2K^T(y - K\alpha) + 2\lambda K\alpha = 0 \;\Leftrightarrow\; \hat{\alpha_\lambda} = (K^T K + \lambda K)^{-1} K^T y$$

4.4 Mise en pratique de la régression non linéaire à base de noyaux

Une fois que l'on s'est targué à choisir le noyau K de notre régression non linéaire, nous pouvons alors représenter cette régression selon les hypothèses données supra.
Simulons tout d'abord notre jeu de données Pokémon en élaborant une régression non linéaire suivant la fonction ϕ définie par :

$$\phi(x) = x - 1.2x^2 - 0.8x^3 + 0.6\cos(2\pi x)$$

On y représentera parallèlement le nuage de points dont les perturbations sont de nature gaussienne $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

```
# Simulation des donnees du modele
fstar <- function(x){
  return(x-1.2*x^2-0.8*x^3+0.6*cos(2*pi*x))
}

sgma <- 0.1
n <- 80
x <- runif(n)
y <- fstar(x)+sgma*rnorm(n)
plot(x,y)

gridx=0:200/200
fgridx <- fstar(gridx)
lines(gridx,fgridx,col='red') # fonction y
```

Figure 1 : Fonction permettant de simuler les données de notre modèle

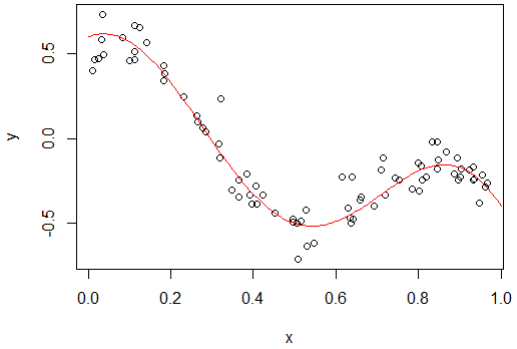

Figure 2: Représentation graphique des données de notre modèle

Puis on procède par l'ajustement de la régression de la crète du noyau standard par un noyau gaussien K.

```
# Ajustement de la regression de la crete du noyau avec un noyau gaussien
install.packages("CVST")
library(CVST)
krr <- constructKRRLearner()

dat <- constructData(x,y)
dat_tst <- constructData(gridx,0)
```

Figure 3 : Mise en exergue du noyau gaussien dans la régression non linéaire

Pour différents λ, nous élaborons un modèle avec un σ et un paramètre de régularisation λ correspondant.

Nous effectuons l'apprentissage puis la prédiction de ce modèle sur les données de test. Et pour finir, nous représentons le nuage de points, le noyau gaussien K ainsi que la régression non linéaire de la prédiction.

```
# Simulation de la regression avec sigma fixe et lambda qui varie
par(mfrow=c(3,3),oma = c(5,4,0,0) + 0.1,mar = c(0,0,1,1) + 0.1)
lambdas= 10^(-8:0)
for(lambda in lambdas){
  param <- list(kernel="rbfdot", sigma=50, lambda=lambda)
  krr.model <- krr$learn(dat,param)
  pred <- krr$predict(krr.model,dat_tst)
  plot( x, y, xaxt='n', yaxt='n', main=paste('lambda =',signif(lambda,digits=3)) )
  lines(gridx,fgridx,col='red')
  lines(gridx,pred,col='blue') # fonction de regression ajustee
}
# On remarque que :
#     - lambda petit = sur-ajustement et pas assez de regularisation
#     - lambda grand = sous-ajustement et trop de regularisation
```

Figure 4 : Fonction permettant de simuler différentes régressions non linéaires avec des λ variables

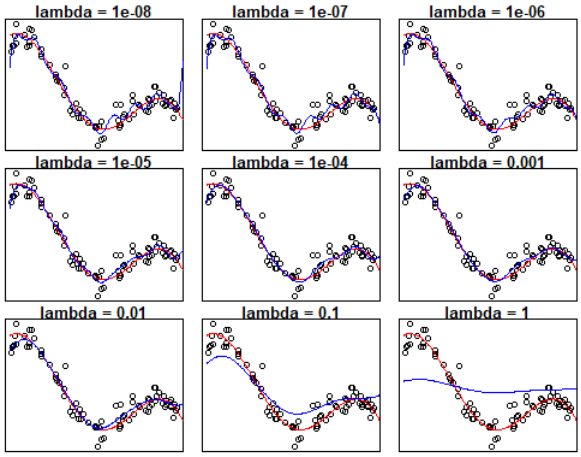

Figure 5 : Représentation graphique des différentes régressions non linéaires avec des λ variables

De même, on remanie notre modèle comme le précédent à la seule différence que cette fois-ci, il s'agira du paramètre σ que l'on variera au fur et à mesure en fixant le paramètre de régularisation λ

```r
# Simulation de la regression avec sigma qui varie et lambda fixe
par(mfrow=c(3,3),oma = c(5,4,0,0) + 0.1,mar = c(0,0,1,1) + 0.1)
sigmas=10^((1:9)/3)
for(sigma in sigmas){
  param <- list(kernel="rbfdot", sigma=sigma, lambda=0.01)
  krr.model <- krr$learn(dat,param)
  pred <- krr$predict(krr.model,dat_tst)
  plot(x,y,xaxt='n',yaxt='n',main=paste('sigma =',signif(sigma,digits=3)))
  lines(gridx,fgridx,col='red')
  lines(gridx,pred,col='blue')
}
# On remarque que :
#     - sigma petit = sous-ajustement
#     - sigma grand = sur-ajustement
```

Figure 6 : Fonction permettant de simuler différentes régressions non linéaires avec des σ variables

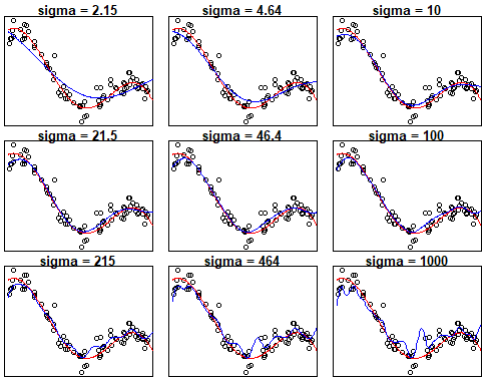

Figure 7 : Représentation graphique des différentes régressions non linéaires avec des σ variables

Nous avons préconisé ci-haut dans le cas de la régression de Ridge la méthode de cross-validation nous assurant un choix optimal du paramètre de régularisation λ en séparant notre jeu de données en un substrat de jeu de test et de jeu d'entrainement, deux à deux. L'idée de la régression à noyau est de raffuter encore le problème de Ridge sous une autre forme (cf. la partie précédente).

Pour ce faire, la fonction CV() prend le soin de minimiser le paramètre λ :

```
# Validation croisee
params <- constructParams(kernel="rbfdot", sigma=sigmas, lambda=lambdas)
opt <- CV(dat, krr, params, fold=10, verbose=FALSE)

par(mfrow=c(1,1), mar = c(0,0,0,0))
param <- list(kernel="rbfdot", sigma=opt[[1]]$sigma, lambda=opt[[1]]$lambda)
krr.model <- krr$learn(dat,param)
pred <- krr$predict(krr.model,dat_tst)
plot(x,y,xaxt='n',yaxt='n',main=paste("selected values: sigma=",signif(param$sigma,digits=3),
                                      ", lambda=",signif(param$lambda,digits=3)))
lines(gridx,fgridx,col='red')
lines(gridx,pred,col='blue')
```

Figure 8 : Fonction permettant de faire intervenir la cross-validation pour le choix optimal de λ

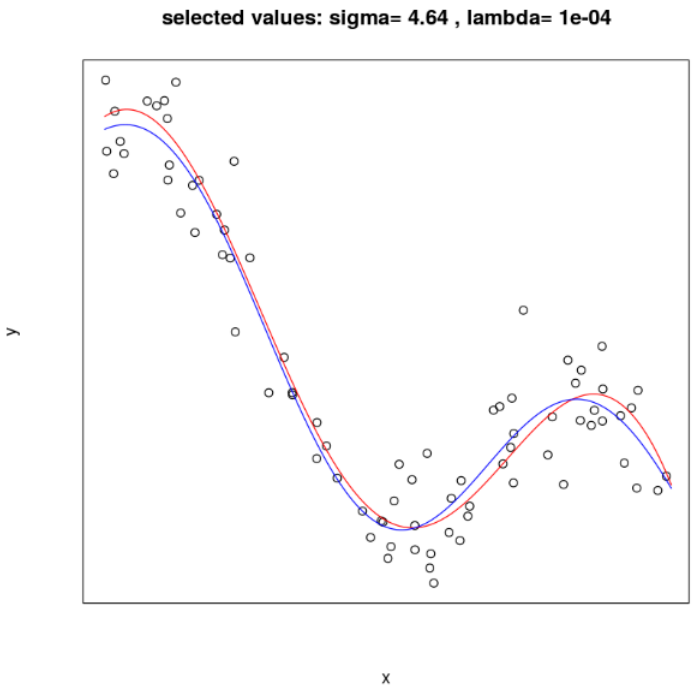

Figure 9 : Représentation graphique du modèle optimal calculé grâce à la procédure de cross-validation

4.5 Modélisation de la régression à noyau sur le jeu de données Pokémon

On restitue la même démarche que la procédure suivie dans la partie précédente. Il sera donc plus question de particulariser et circonstancier notre jeu de données en fonction de ce que l'on dispose à partir du jeu de données Pokémon.
Dans un premier temps, une récupération et un chargement de données est nécessaire pour définir notre variable à expliquer ainsi que le chargement et la construction de notre modèle.

```
# Rappel des variables de pokemon
pokemon <- na.omit(read.csv("~/L3_MIASHS_IDS/Modele Lineaire/pokemon.csv"))
pokemon <- pokemon[,c("attack","defense","hp","sp_attack","sp_defense","speed")]
attach(pokemon)
a = pokemon[order(speed),]     # Tri par la speed pour avoir un nuage de points cohérent
x <- 1:684
y <- a$speed                   # Extraction de la variable à estimer
```

Figure 1 : Récupération/Nettoyage des données Pokemon

```
# Ajustement de la regression de la crete du noyau avec un noyau gaussien
install.packages("CVST")
library(CVST)
krr <- constructKRRLearner()
dat <- constructData(x,y)
dat_tst <- constructData(x,0)
```

Figure 2 : Mise en exergue du noyau gaussien dans la régression non linéaire

Pour obtenir une sous-régularisation, on itère à nouveau la prédiction sur notre modèle en ajustant les bons paramètres λ et σ.
Ceci aura pour but de remédier à un modèle beaucoup trop overfitted après maintes manipulations sur les paramètres du modèle.

```
# Simulation de la regression avec sigma et lambda qui varient
sigmas <- 10^(-15:15)                                          # les différentes valeurs de sigma
lambdas <- 10^(-15:15)                                         # les différentes valeurs de lambda
params <- constructParams(kernel="rbfdot", sigma=sigmas, lambda=lambdas)# les paramètres de la validation croisée
opt <- CV(dat, krr, params, fold=10, verbose=FALSE)            # valeurs optimales pour la validation croisée

param <- list(kernel="rbfdot", sigma=opt[[1]]$sigma, lambda=opt[[1]]$lambda)# les paramètres optimaux
krr.model <- krr$learn(dat,param)                              # le modèle d'apprentissagese servant de nos données et des paramètres optimaux
pred <- krr$predict(krr.model,dat_tst)                         # on prédit la regression sur nos data test (vierge)
plot(x,y,xaxt='n',yaxt='n',main=paste("selected values: sigma=",signif(param$sigma,digits=3),", lambda=",signif(param$lambda,digits=3)))
lines(x,pred,col='blue', lwd=2)
```

Figure 3 : Fonction permettant de simuler la régression non linéaire optimale

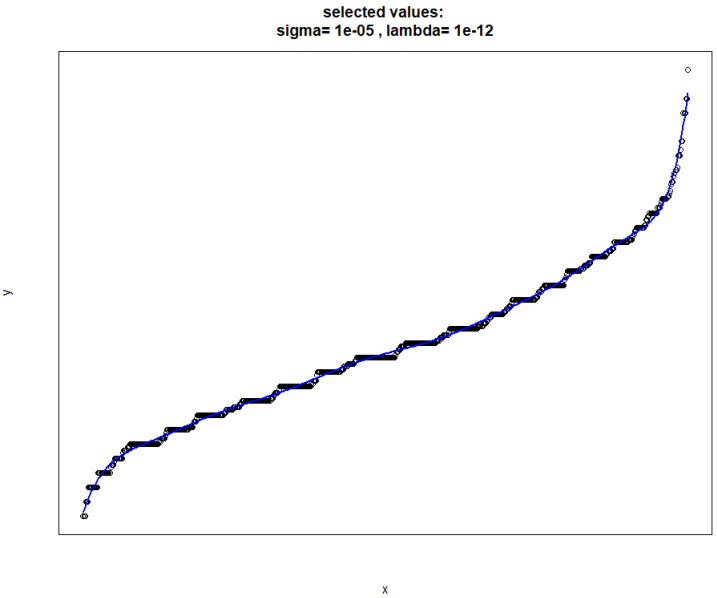

Figure 4 : Représentation graphique du modèle optimal calculé grâce à la procédure de cross-validation

5 Conclusion

En définitive, plusieurs recherches tendant à pâlier aux problèmes inhérents de la régression linéaire classique se sont mobilisées notamment pour estomper la variance des données. La régularisation à partir des régressions pénalisées a marqué un tournant dans l'apprentissage statistique de façon générale.
Cette même pénalisation de Ridge a été manoeuvrée à partir de la technique de kernelisation de chaque régression, méthode célèbre qui non seulement s'affaire aux problèmes de régression mais également dans la non-linéarité de l'algorithme des SVM (Support Vector Machine) et dans la réduction de dimensionalité par le truchement de l'ACP. D'autres algorithmes d'apprentissage supervisés et non supervisés qui sont bien encore en vogue dans le diapason du machine learning.