

PROJET STATISTIQUE EN GRANDE DIMENSION

Solène LESAGE - 24/01/23



Plan

I. Exercice 1

- 1 - Méthode Gradient Boosting
- 2 - Méthode XGBoost
- 3 - Optimisation des paramètres/hyperparamètres

II. Exercice 2

- 1 - Contexte
- 2 - Méthode PCR
- 3 - Méthode PLS
- 4 - Méthode LASSO
- 5 - Méthode Sparse PCA
- 6 - Méthode Kernel PCA
- 7 - Méthode SVM
- 8 - Méthode Random Forest

III. Exercice 3

- 1 - Contexte
- 2 - Méthode PLS
- 3 - Méthode LASSO

Exercice 1

1- Méthode Gradient Boosting

- Principe

Le Gradient Boosting est une méthode non linéaire dont **l'apprentissage se fait de manière séquentielle en boostant la précision du précédent modèle** (*non pas les données mais les résidus*)

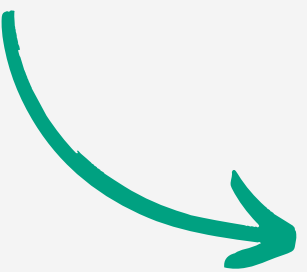
Après chaque itération, le but est de se **rapprocher du modèle final** et de les écarter petit à petit des prédictions du modèle de la moyenne pour les rapprocher de la réalité.

Exercice 1

1- Méthode Gradient Boosting

- **Algorithme :**

- Le premier apprenant faible est initialisé à la moyenne des observations.
- L'écart entre cette moyenne et la réalité appelé premier **résidu**.
- Le second apprenant faible est entraîné pour prédire le résidu du premier. Les prédictions du second apprenants faibles sont ensuite multipliés par un facteur η (learning rate) **pour réduire la taille du pas et donc pour augmenter la précision**.
- Les erreurs sont minimisées par l'algorithme de **descente de gradient**. On corrige seulement les prédictions pour lesquelles les résidus sont élevés, c'est ce que fait le prochain apprenant faible.



Pour connaître la prédiction du gradient boosting sur une observation, on interroge chaque apprenant faible et on somme toutes les réponses obtenues pour former l'apprenant fort final. **Le boosting convertit un système d'apprenants faibles en un système unique d'apprentissage fort.**

Exercice 1

1- Méthode Gradient Boosting

- Utilisation d'une partie du jeu de données MNIST

- Implémentation

```
gradient_booster = GradientBoostingClassifier(loss='deviance')
gradient_booster.fit(X_train, y_train)
```

- Résultats

```
Accuracy score (train): 0.99
Accuracy score (validation): 0.90
```

+ Recherche des paramètres optimaux GridSearch CV

```
gradient_booster_best = GradientBoostingClassifier(loss='deviance',
learning_rate=0.1, max_depth=9, max_features=4, n_estimators=100,
random_state=0)
```

```
Accuracy score (validation): 0.93
```

-> meilleur que le premier modèle

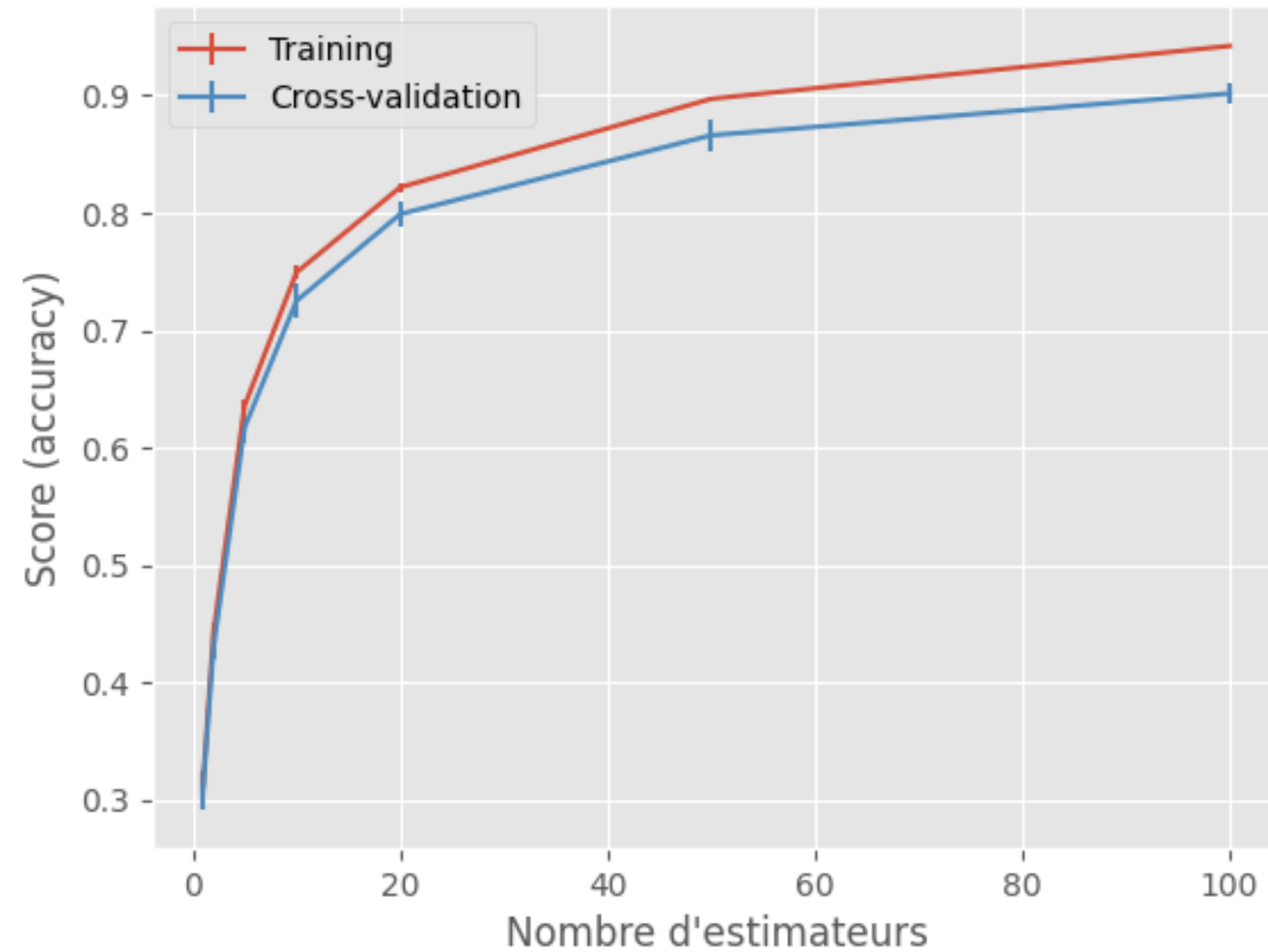
Métriques

	precision	recall	f1-score	support
0	0.95	0.99	0.97	175
1	0.97	0.99	0.98	234
2	0.93	0.92	0.93	219
3	0.87	0.92	0.89	207
4	0.92	0.92	0.92	217
5	0.95	0.92	0.93	179
6	0.97	0.93	0.95	179
7	0.94	0.88	0.91	204
8	0.93	0.86	0.90	192
9	0.85	0.91	0.88	194
accuracy			0.93	2000
macro avg	0.93	0.93	0.93	2000
weighted avg	0.93	0.93	0.93	2000

Exercice 1

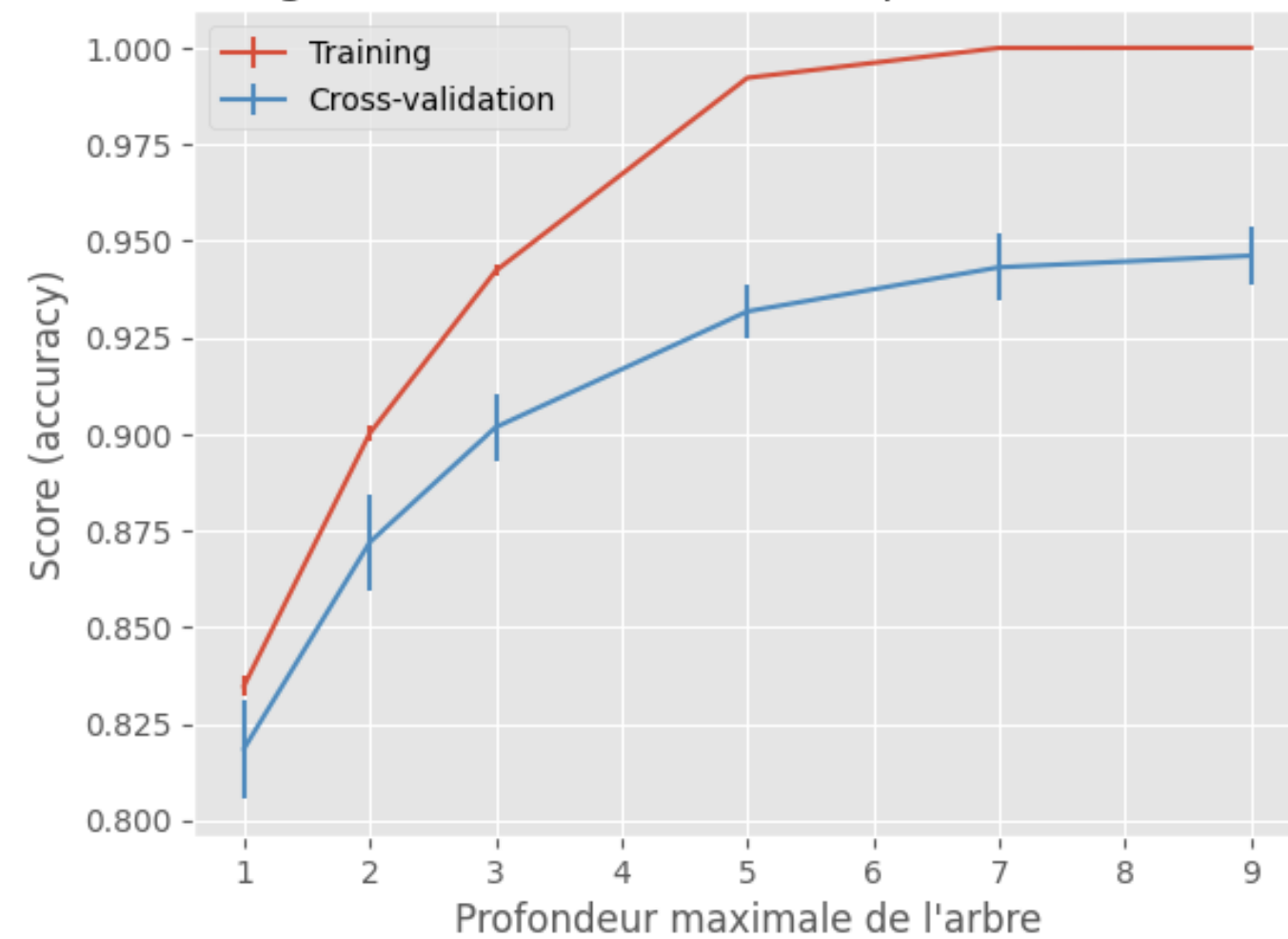
1- Méthode Gradient Boosting

Gradient Boosting - Score en fonction du nombre d'estimateurs



Plus le nombre d'estimateurs (d'arbres) augmentent, plus les scores train et test sont meilleurs et tendent vers 1.

Gradient Boosting - Score en fonction de la profondeur maximale de l'arbre



On remarque que l'erreur de validation stagne. Pour éviter le sur-apprentissage, on choisit de prendre une profondeur maximale de 2 ou 3.

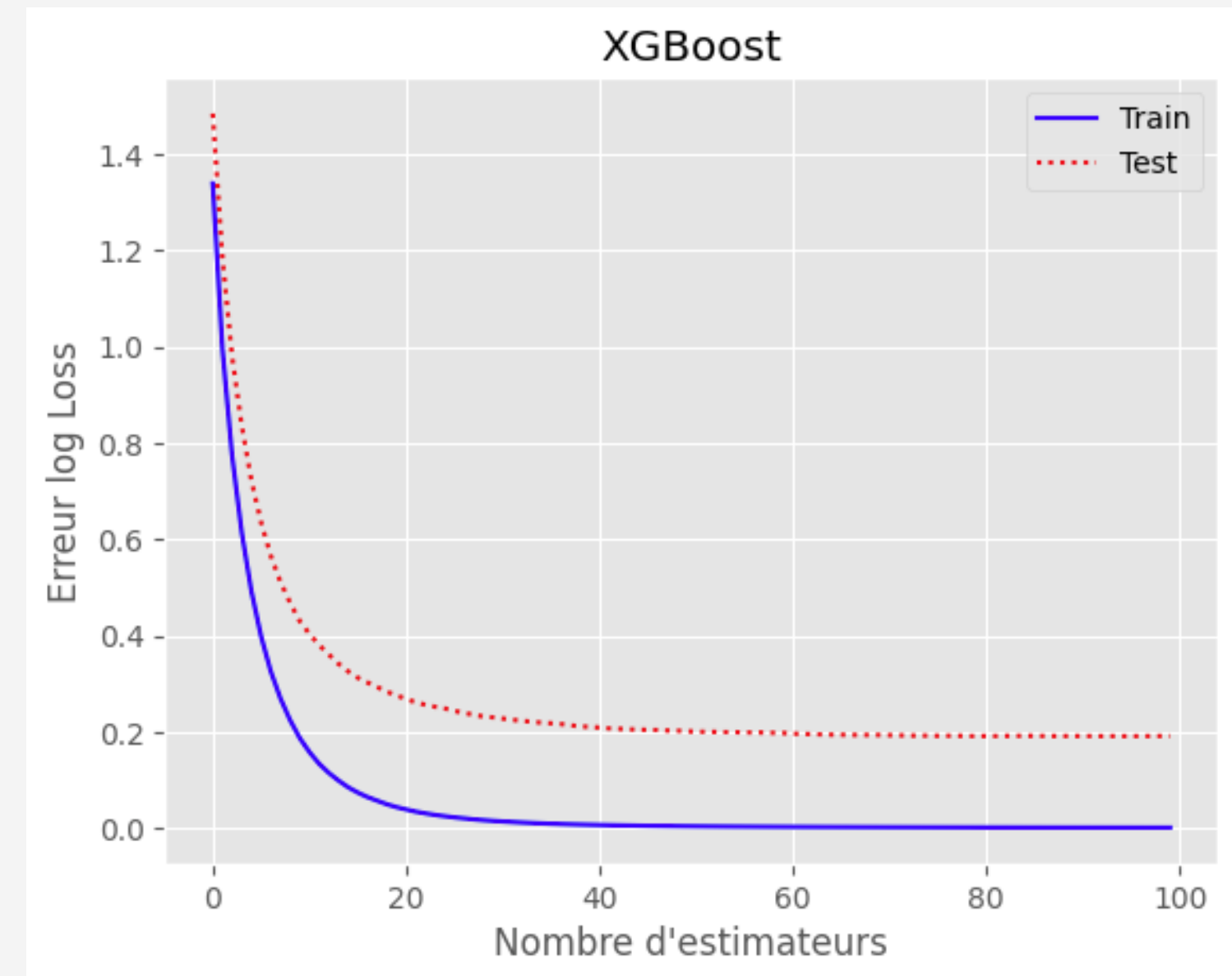
Exercice 1

2- Méthode XGBoost

XGBoost est une **variante de l'algorithme du Gradient Boosting**. La particularité de cette méthode réside dans le type d'apprenant faible.

XGBoost permet de **sélectionner des features suivant le gain**. Cette méthode implique un tri pour chaque *feature* et un calcul du gain pour chaque critère de séparation. En conséquence, c'est une approche qui demande beaucoup de temps de calcul et qui est applicable que pour des jeux de données de taille raisonnable.

XGBoost s'assure de ne garder que des bons apprenants faibles.



Accuracy score (validation) : 0.88

Exercice 1

3- Optimisation des paramètres/hyperparamètres

- **Paramètres :**
 - Profondeur maximale de l'arbre
 - Nombre d'estimateur
 - *Learning rate* (impact sur la valeur prédite)
- **Paramètres de régularisation**
 - *lambda* (impact calcul du gain et sur la prédiction)
 - *gamma* (impact calcul du gain)
 - *alpha* (impact sur la prédiction)
- **Variantes**
 - *LightGBM* (voir code)
 - *Catboost*
- **Early-stopping** (stop l'ajout de nouveaux arbres si l'ensemble n'apporte aucun gain)

Exercice 2

1- Contexte

- **Jeu de données (cancer du sein) :**
 - 278 individus et 22299 variables (contexte en grande dimension) + sélection de toutes les variables
 - 63 valeurs manquantes
 - supprime les individus et les variables non importantes
 - chercher à remplacer les valeurs manquantes si possible
 - supprimer 10 individus et 8 variables au total
 - transforme les variables catégorielles en variables numériques
- **But :**
 - Prédire la réaction à un traitement (pCR ou RD)
- **Données déséquilibrées** (213 pour le RD et 55 pour le pCR)
- **Normalisation des données**
- **Application de plusieurs méthodes d'apprentissage statistique et de réduction de dimension**
 - Utilisation de la validation croisée 5 folds en général avec le découpage : 70% train & 30% test
 - Recherche d'un potentiel meilleur modèle (compromis biais/variance)

Exercice 2

2- Méthode PCR

- **But :**

Projeter les données dans un sous-espace de dimension réduit tout en conservant un maximum d'informations.

Cette méthode permet de comprendre et d'expliquer les données pour en tirer de l'information.

- **Résultats :**

Variance expliquée avec 1 composante = 17%

Variance expliquée avec 2 composantes = 21%

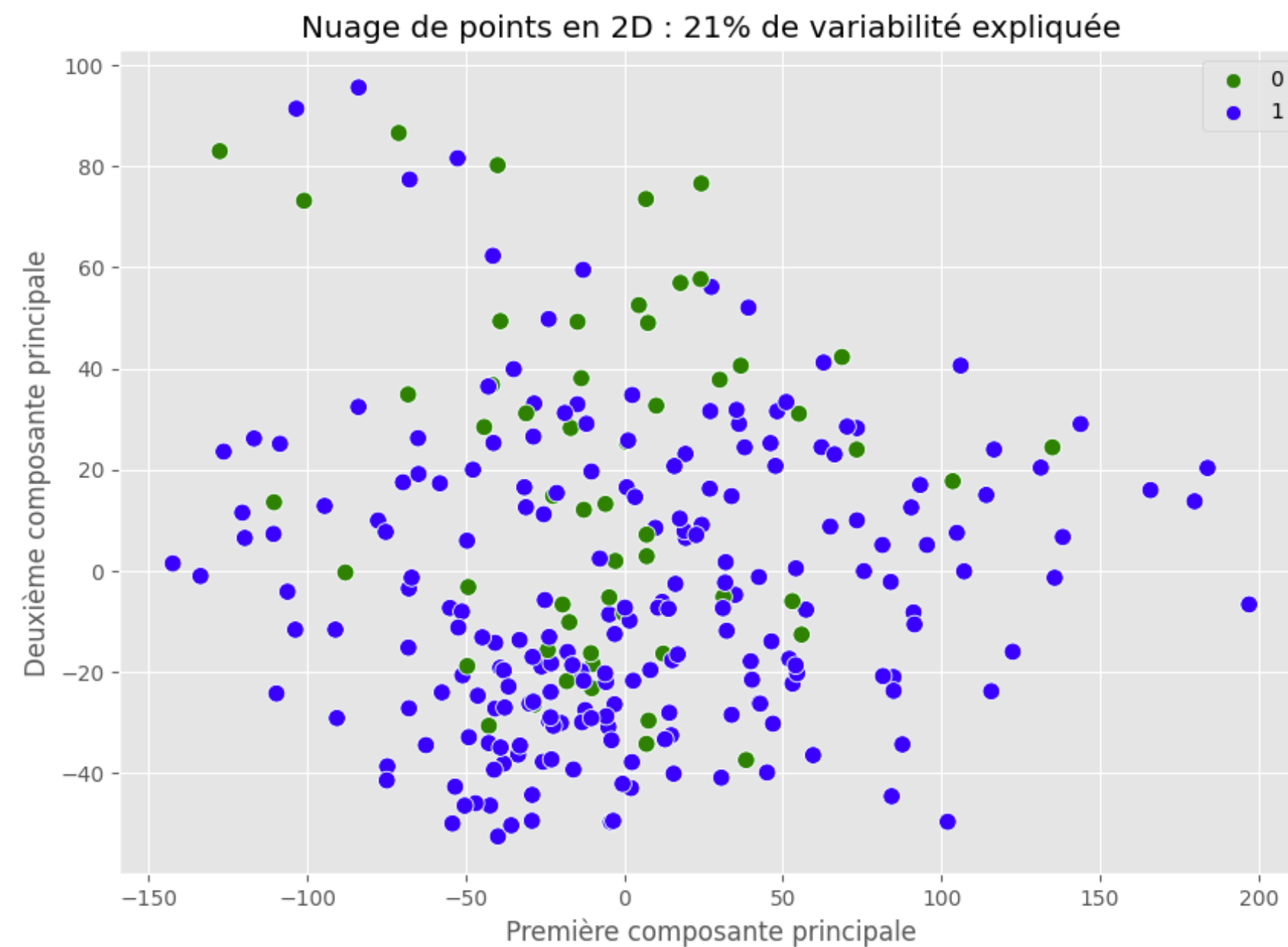


Modèle qui n'explique quasiment rien

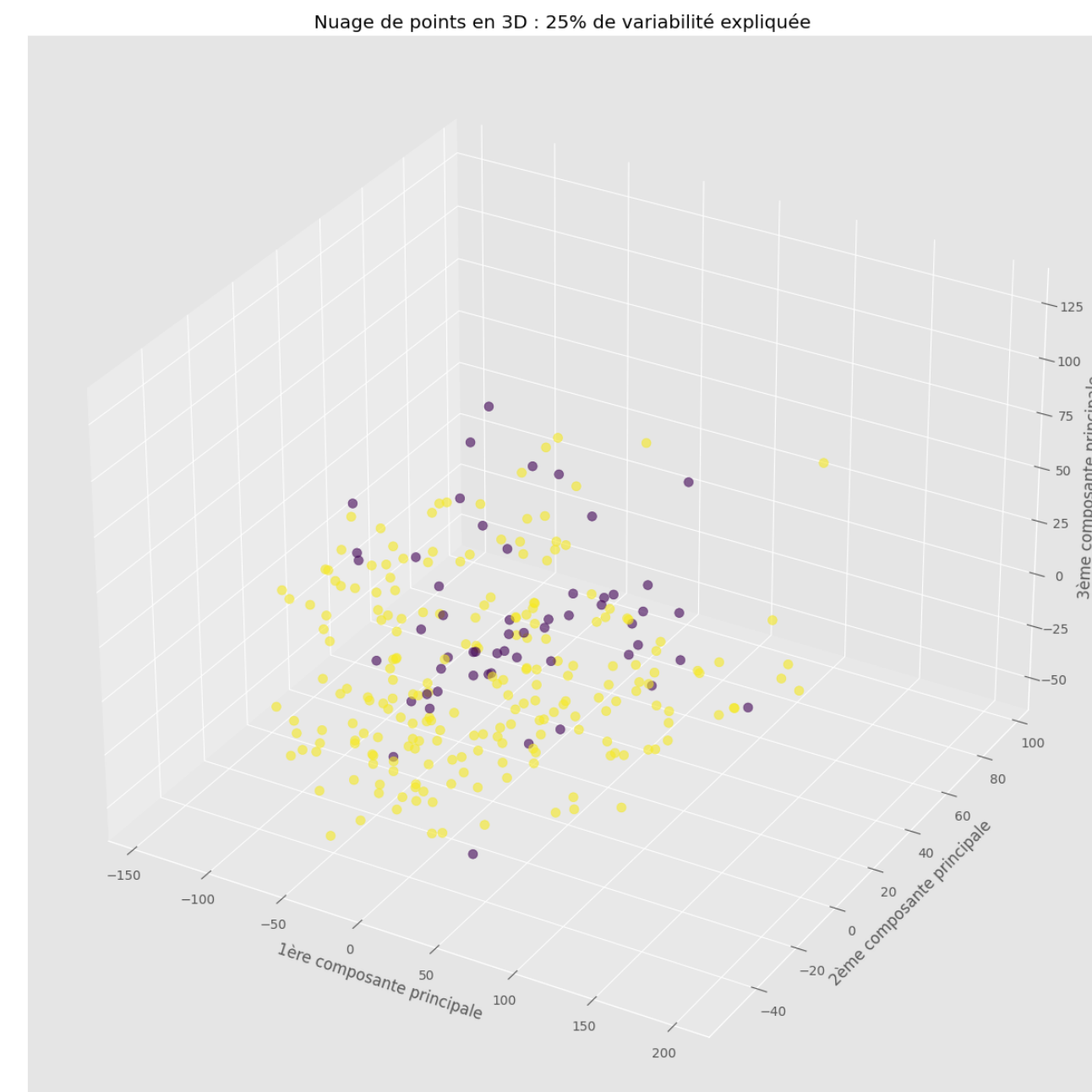
Exercice 2

2- Méthode PCR : Visualisation des données en 2D et en 3D

- En 2D

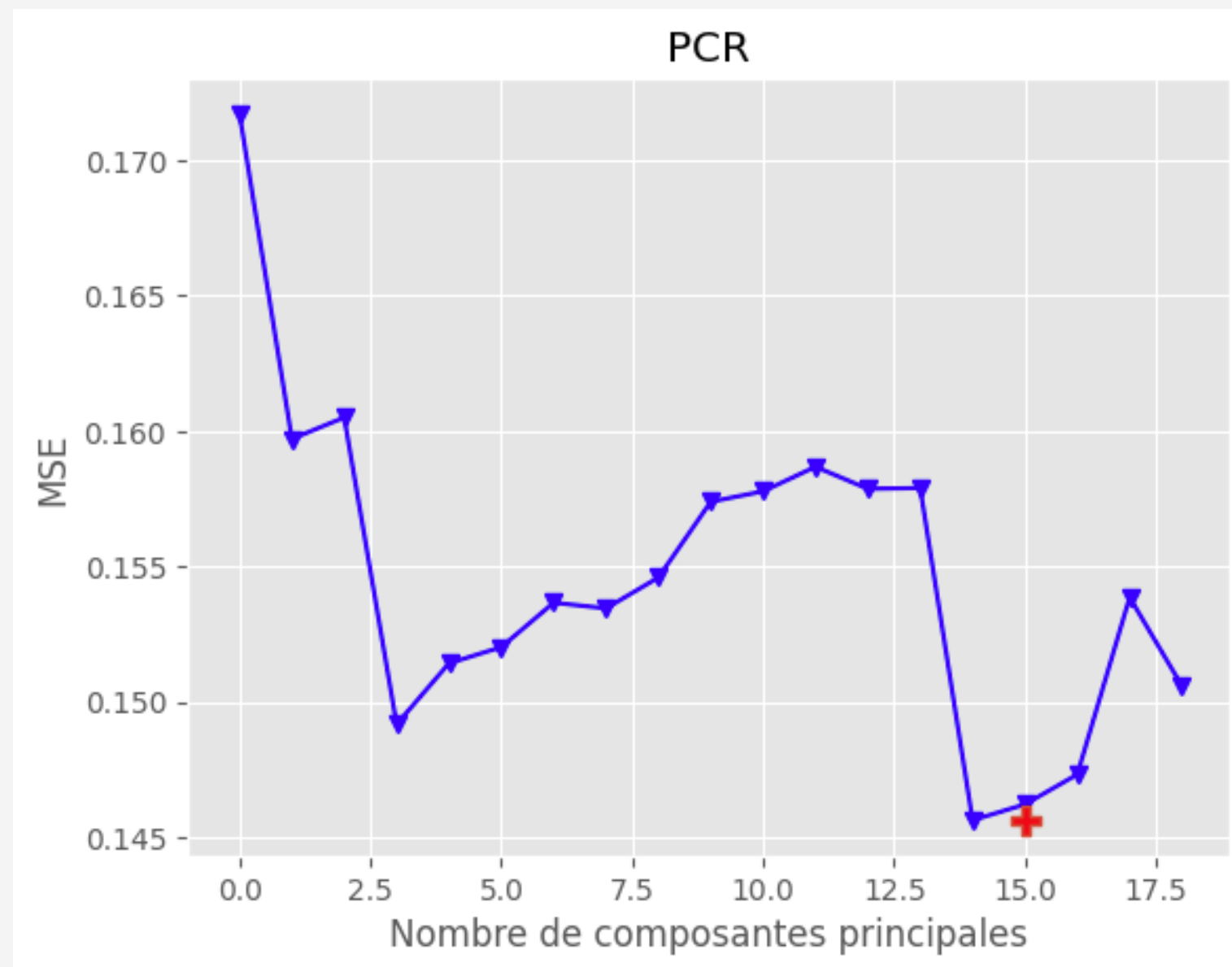


- En 3D



Exercice 2

2- Méthode PCR : Nombre de composantes principales minimales suivant l'erreur MSE minimale



- Résultats :

Nombre de composantes suggérées : 15

MSE (train) : 0.13

MSE (test) : 0.12

Exercice 2

3- Méthode PLS

- But :

Méthode de régression par les moindres carrés partiels basés sur un critère de minimisation des covariances

- Résultats :

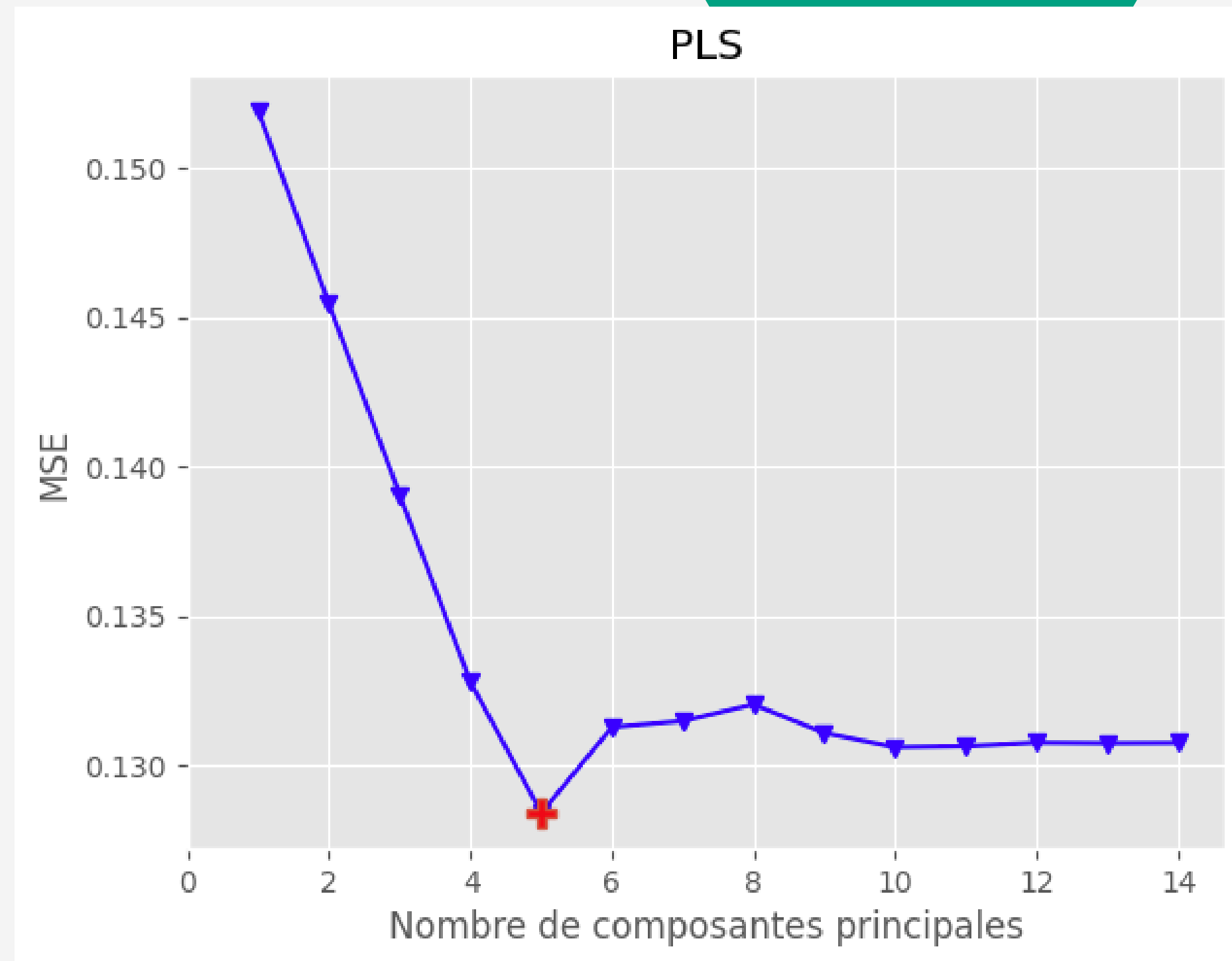
Nombre de composantes suggérées : 5

MSE (train) : 0.006

MSE (test) : 0.114

-> erreur MSE plus faible que PCR

-> nécessite que 5 composantes principales



Exercice 2

4- Méthode LASSO

- But :

Permet de **contraindre les coefficients à se rapprocher de 0 en utilisant la régularisation L1.**

Certains coefficients valent exactement 0. Cela signifie que certaines variables sont totalement ignorées par le modèle.



Cette méthode permet de rendre **plus simple** le modèle à interpréter en **sélectionnant des variables**.

Exercice 2

4- Méthode LASSO : Résultats

1ère version :

```
lasso = Lasso(alpha=1)
lasso.fit(X_train, y_train)
```

R2 : data train : 0.0

R2 : data test : -0.01

MSE data train : 0.17

MSE data test : 0.15

-> sous-apprentissage

2ème version : optimisation du paramètre alpha (en le diminuant). + augmenter le nombre d'itérations

```
model = LassoCV(cv=5, random_state=0, max_iter=10000)
model.fit(X_train, y_train)
```

```
lasso_best = Lasso(alpha=model.alpha_)
lasso_best.fit(X_train, y_train)
Lasso(alpha=0.045)
```

R2 : data train : 0.68

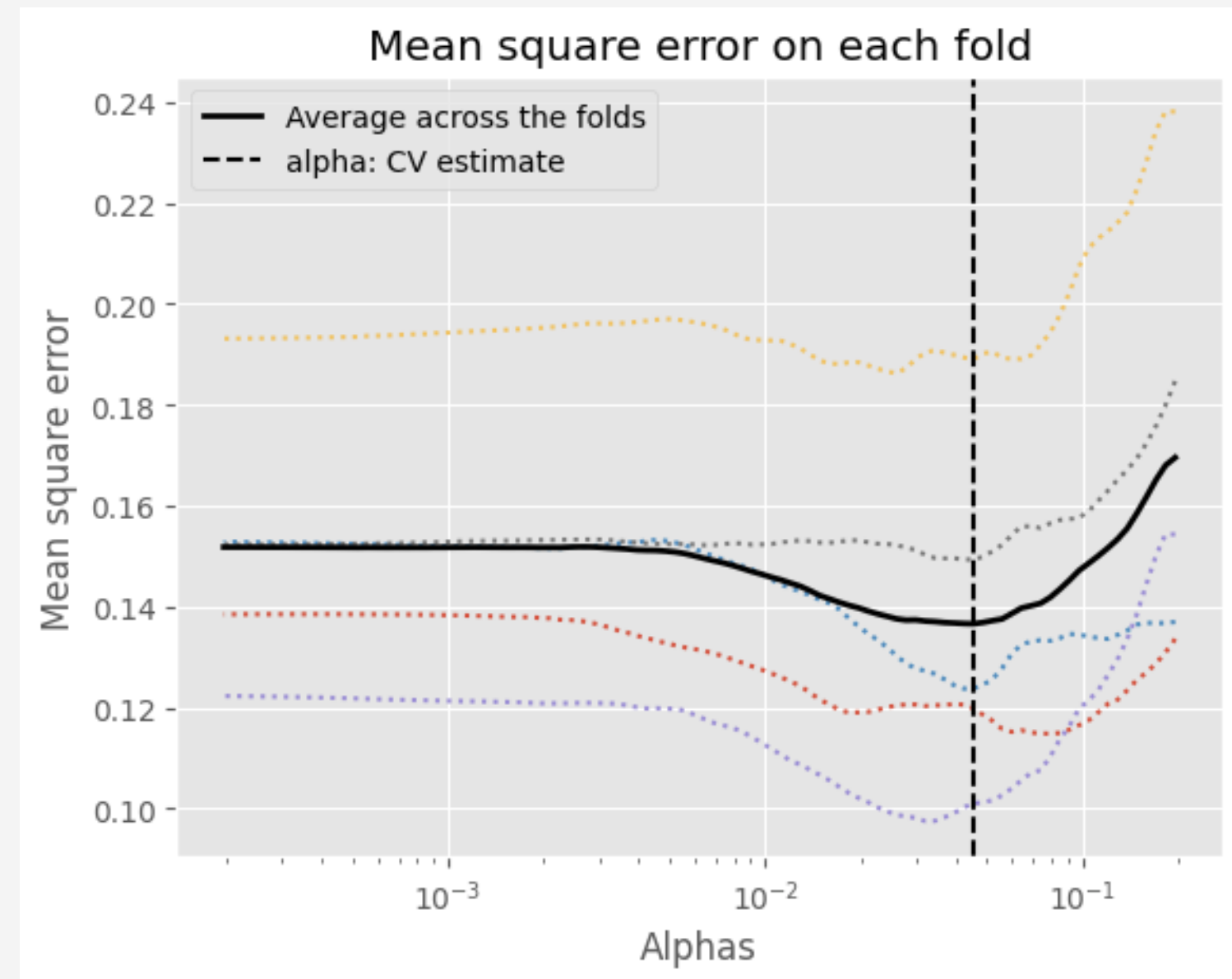
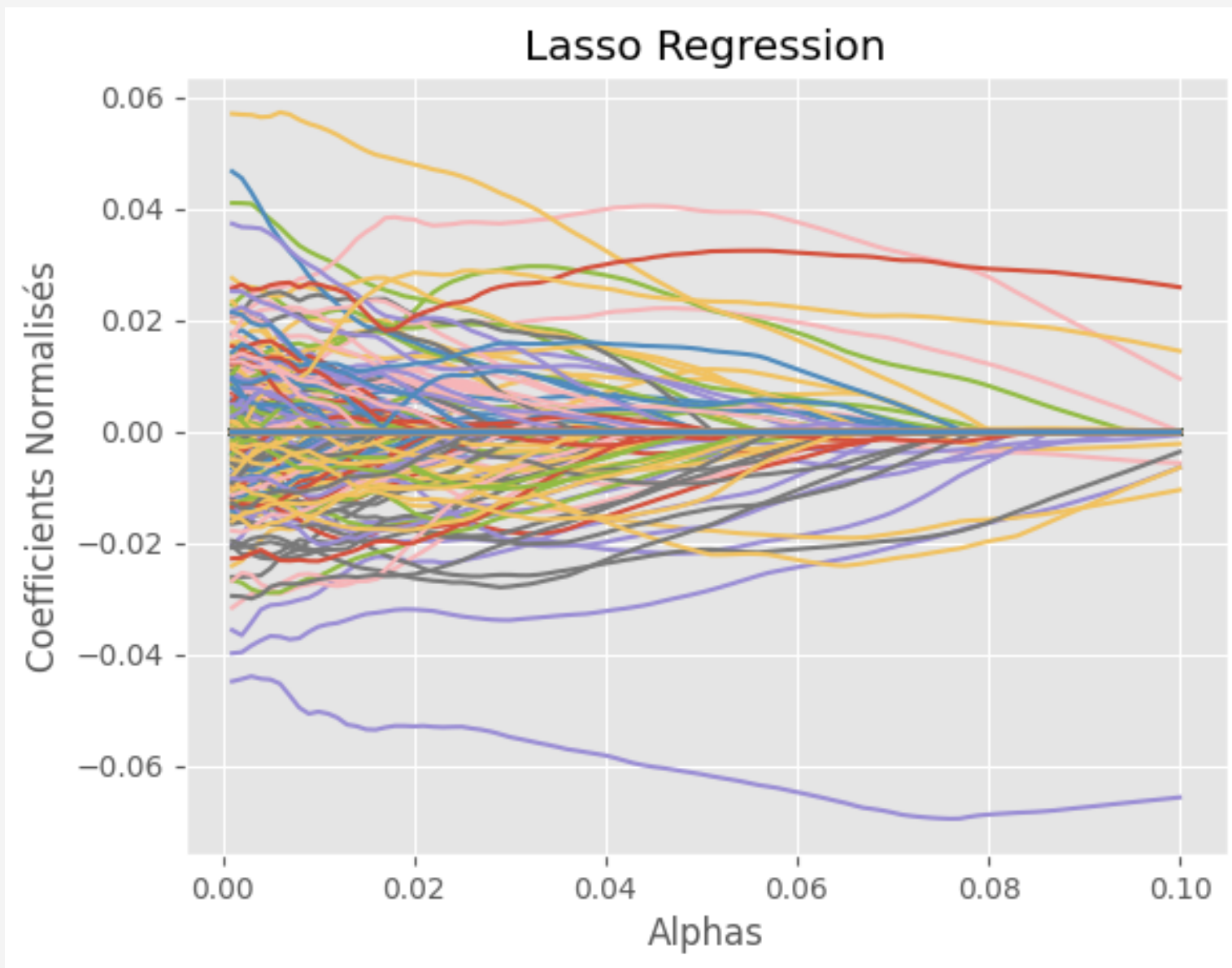
R2 : data test : 0.27 -> R2 non satisfaisant mais meilleur que la 1ère version

MSE data train : 0.05

MSE data test : 0.11

Exercice 2

4- Méthode LASSO : Résultats



Exercice 2

5- Méthode Sparse PCA

But :

Un inconvénient particulier de l'ACP est que les composants principaux sont généralement des combinaisons linéaires de toutes les variables d'entrée. Sparse PCA surmonte cet inconvénient en **trouvant des combinaisons linéaires qui ne contiennent que quelques variables d'entrée**.

La quantité de parcimonie peut être contrôlée à l'aide du paramètre *alpha*, où des **valeurs alpha élevées conduisent à des résultats plus clairs**.

Nous pouvons utiliser un nombre limité de composants sans la limitation donnée par une matrice de projection dense. Cela peut être fait en utilisant une matrice creuse, où le nombre d'éléments non nuls est assez faible.

Exercice 2

6- Méthode Kernel PCA

- But :

La méthode Kernel PCA permet de traiter des modèles de données plus complexes pour **faire face à la présence non linéaire des données**.

L'idée est d'**utiliser une fonction de noyau** pour projeter un ensemble de données dans un espace de caractéristiques de dimension supérieure, où il est linéairement séparable (la frontière de décision devient linéaire).

Exercice 2

6- Méthode Kernel PCA :

- Résultats :

Implémentation du meilleur modèle

```
kpca_best= KernelPCA(n_components=2, kernel =  
'rbf',gamma=0.02)
```

```
X_train = kpca_best.fit_transform(X_train)
```

```
X_test = kpca_best.transform(X_test)
```

```
model = LogisticRegression()
```

```
model.fit(X_train, y_train)
```

Matrice de confusion :

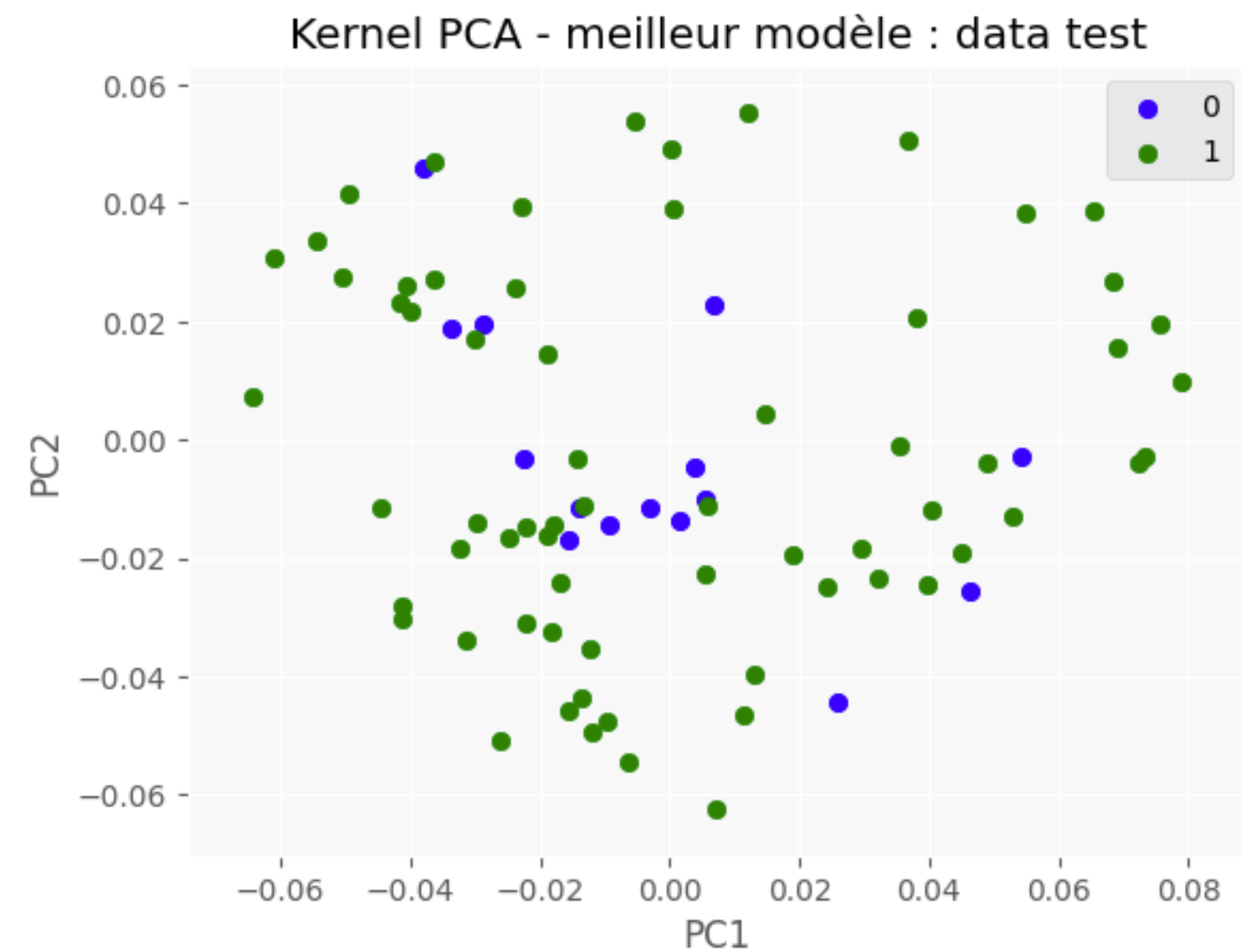
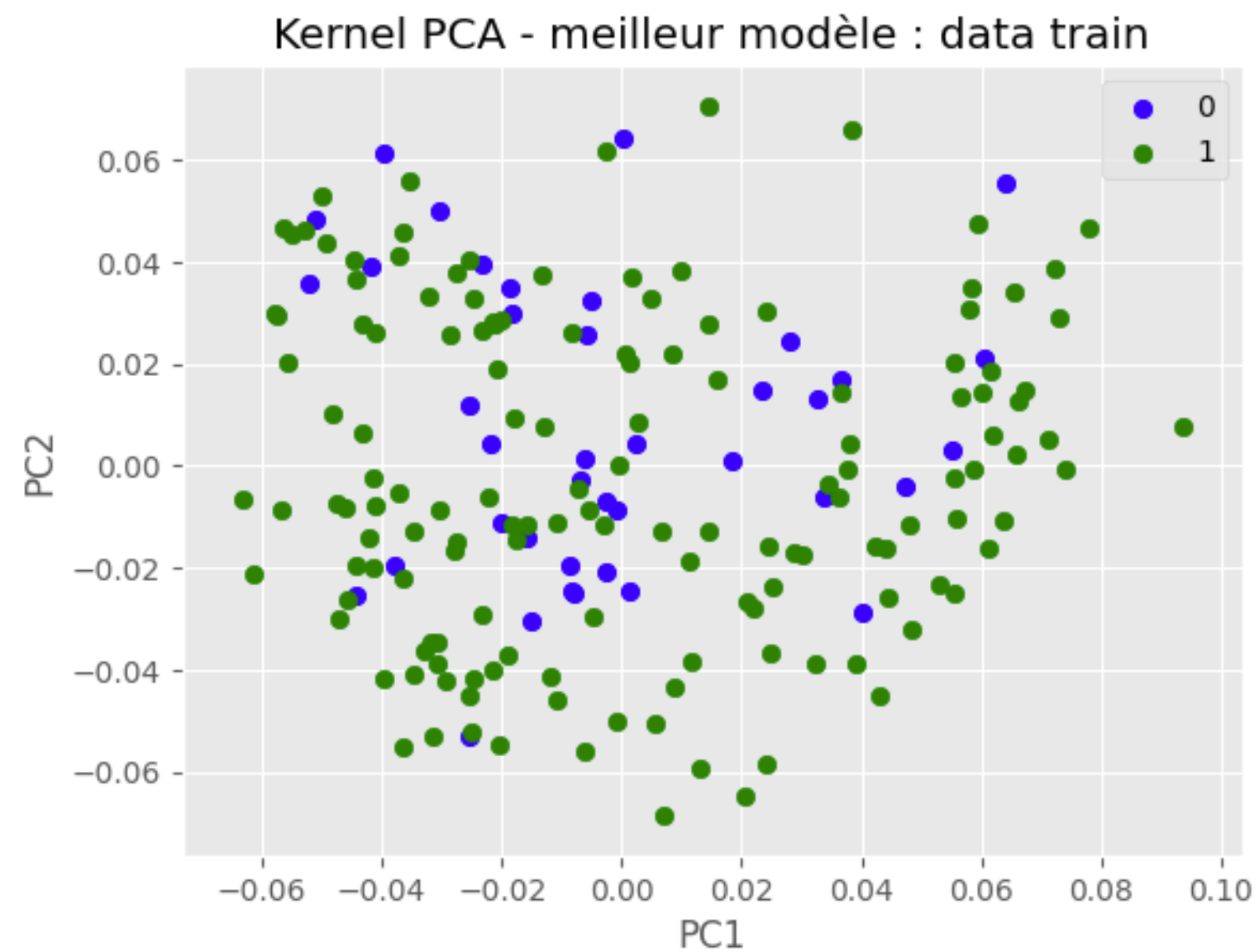
```
[[ 0 15]  
 [ 0 66]]
```

Métriques :

	precision	recall	f1-score	support
0	0.00	0.00	0.00	15
1	0.81	1.00	0.90	66
accuracy			0.81	81
macro avg	0.41	0.50	0.45	81
weighted avg	0.66	0.81	0.73	81

Exercice 2

6- Méthode Kernel PCA : Visualisation des données



Exercice 2

7- Méthode SVM :

- But

L'algorithme SVM recherche un **hyperplan le plus loin possible des observations** des deux classes. Il cherche à **accroître la marge séparatrice entre ces deux classes** et classer correctement un maximum de points de données d'entraînement.

Deux paramètres sont importants à ajuster :

- **l'hyperparamètre C** (pénalité associée à chaque observation mal classées)
- **γ** utilisé avec des fonctions de noyaux non linéaires qui permet de contrôler le niveau de courbure de l'hyperplan séparant les deux classes

Exercice 2

7- Méthode SVM :

- Résultats :

1ère version : noyau linéaire

```
svm = SVC(kernel='linear')  
svm.fit(X_train, y_train)
```

-> 155 vecteurs supports
(40 pour la classe 0 et 115 pour la 1)

-> très bons résultats de métriques

Métriques

	precision	recall	f1-score	support
0	0.73	0.53	0.62	15
1	0.90	0.95	0.93	66
accuracy			0.88	81
macro avg	0.81	0.74	0.77	81
weighted avg	0.87	0.88	0.87	81

Matrice de confusion

```
[[ 8  7]  
 [ 3 63]]
```

Exercice 2

7- Méthode SVM :

- Résultats :

2ème version : optimisation des paramètres sur le critère recall

```
svm_best2 = SVC(kernel='rbf', C=1, gamma=0.01)
svm_best2.fit(X_train, y_train)
```

-> on préfère les résultats du premier modèle suivant le critère de l'accuracy (linéaire et C=1)

Métriques

	precision	recall	f1-score	support
0	0.00	0.00	0.00	15
1	0.81	1.00	0.90	66
accuracy			0.81	81
macro avg	0.41	0.50	0.45	81
weighted avg	0.66	0.81	0.73	81

Exercice 2

8- Méthode Random Forest :

- But :

L'algorithme Random Forest **construit plusieurs arbres de décision** à partir d'un ensemble d'échantillons. Pour chaque échantillon tiré, un arbre est construit.

Plusieurs paramètres sont à prendre en compte :

- le **nombre d'arbres** à construire
- la **profondeur de l'arbre** de décision
- le **choix des variables**

Exercice 2

8- Méthode Random Forest :

- Résultats :

Implémentation du meilleur modèle

```
rfc_best=RandomForestClassifier(  
random_state=0, max_features='log2', n_estimators= 5,  
max_depth=7, criterion='gini')
```

```
rfc_best.fit(X_train, y_train)
```

Matrice de confusion

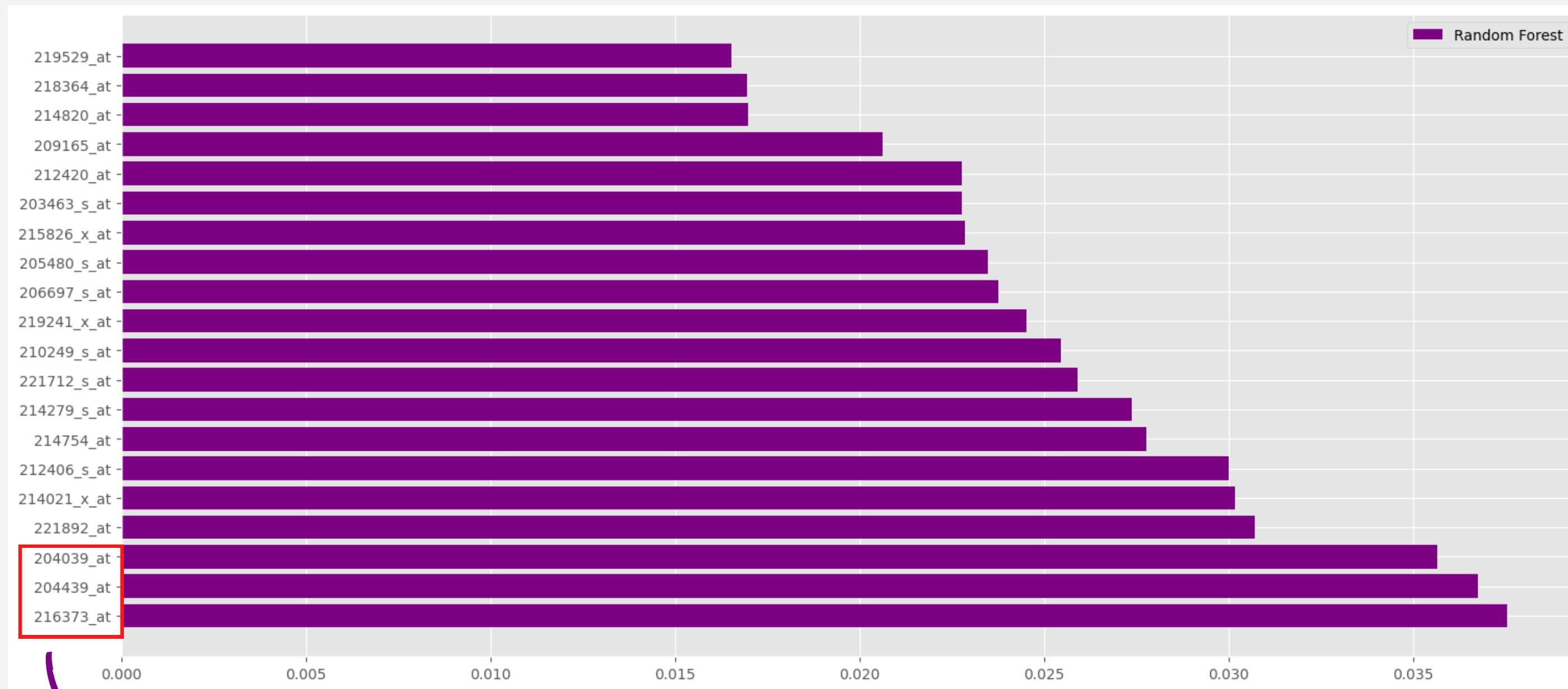
```
[[ 5 10]  
 [ 5 61]]
```

Métriques

	precision	recall	f1-score	support
0	0.50	0.33	0.40	15
1	0.86	0.92	0.89	66
accuracy			0.81	81
macro avg	0.68	0.63	0.65	81
weighted avg	0.79	0.81	0.80	81

Exercice 2

8- Méthode Random Forest :



Corrélation très faible avec la variable réponse mais 3 semblent plus importantes

Exercice 3

Chloé Douarec & Meriem Bencheikh

1- Contexte

- **But :**

Ce challenge a pour but de **prévoir les surges dans deux villes côtières d'Europe occidentale**.

Connaissant les valeurs de *surge* et le champ de pression au niveau de la mer des 5 derniers jours, nous voulons prédire les valeurs de *surge* des 5 prochains jours.

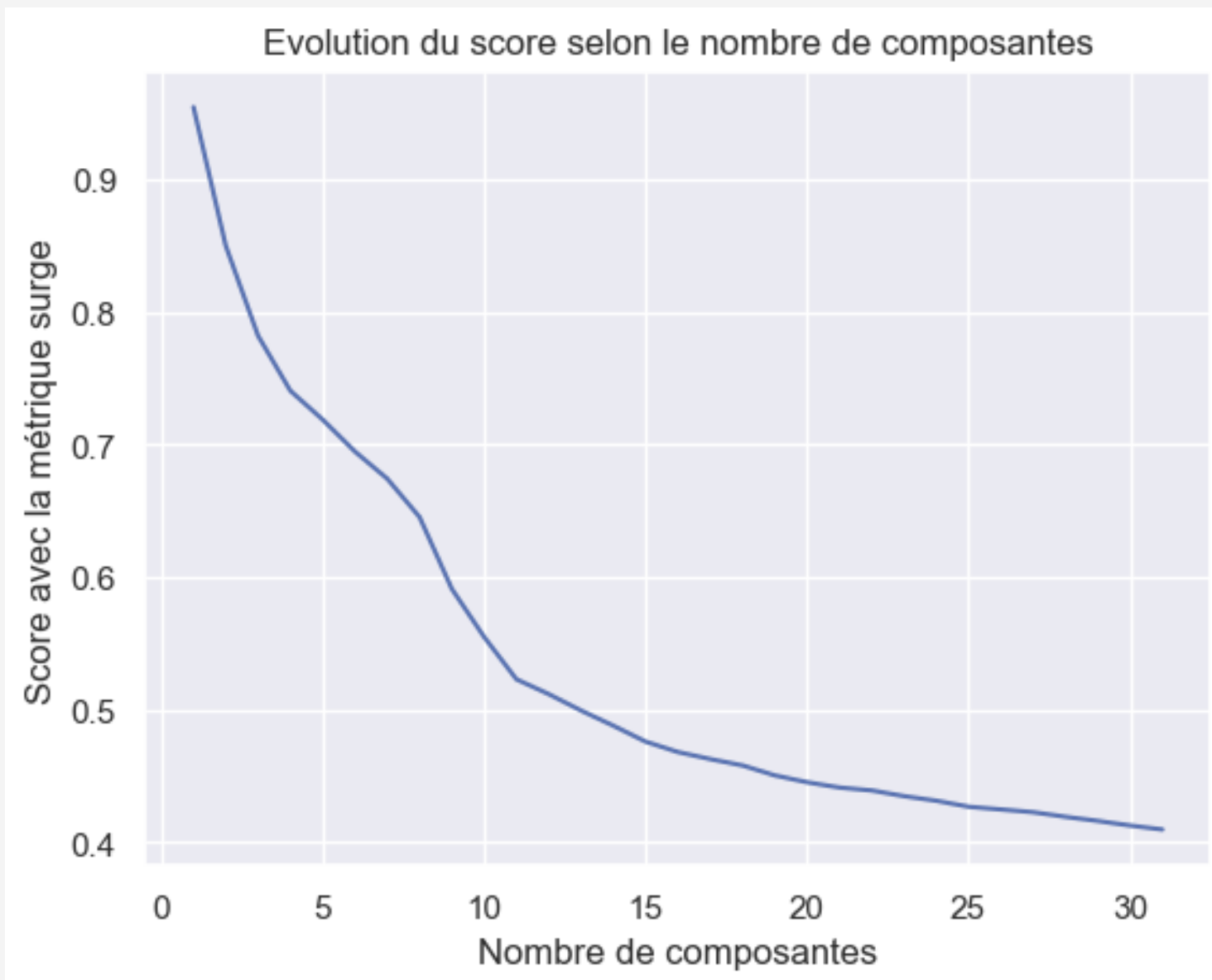
Il s'agit donc d'un problème de prédiction de séries temporelles.

- Construction de *slp_test* et *slp_train*
- Normalisation des données

Exercice 3

2 - Méthode PLS :

- Résultats :



Nombre de composantes PLS	Erreur d'entrainement	Score sur le site du challenge
19	0.47	0.5419
24	0.429	0.5240
26	0.4247	0.5210
28	0.4192	
30	0.4126	0.520
31	0.4096	0.5185
32	0.4078	0.5238

Exercice 3

3 - Méthode LASSO :

- Résultats :

Alpha	Erreur d'entraînement	Score sur le site du challenge
0.008		0.4886
0.01	0.38	0.4871
0.011	0.3887	0.4866
0.018	0.4	0.4866
0.02		0.4873

- Conclusion :

Nous avons réussi à trouver une méthode : **LASSO meilleure que la méthode PLS.**

Les méthodes de régression non linéaire nous ont permis d'obtenir de très bons résultats.

A decorative graphic in the top right corner consisting of two overlapping hexagons. The front hexagon is a light lime green, and the back hexagon is a darker teal color.

Merci pour votre
attention