Partie 1 : calcul du gradient :

Annale 2023-2024 :

$J_1(\vec{w}) = w_0 + x_0 + w_1 x_1 + w_2^2 x_2 + w_3 w_4 x_3^2 - w_4 x_4$

$\nabla_{\vec{w}} J_1(\vec{w})$ en fonction de $\vec{w}$ et des constantes $(x_0, x_1, x_2, x_3, x_4)$:

```
def gradient_de_J1(w,x):
    return np.array([1 , x[1], 2*w[2]*x[2], w[4]*x[3]**2 , w[3]*x[3]**2 - x[4]])
```

* rq : normalement w est de dimension d
x est de dimension (n, d)] → donc x@d
D = X.shape[1]
N = X.shape[0]

```
def gradient_de_J2(w,x):
    return x   # où J2(w) = w.x
```

Annale 2024 - 2025 :

$J_1(w) = 1 + x_0 + w_0^2 + x_1^2 w_1 - 2x_2 w_2^2 + x_3^2 w_3 w_4 + x_4^4$

```
def gradient_de_J1(w,x):
    return ( 2*w[0], x[1]**2, -4*x[2]*w[2], x[3]**2*w[4], x[3]**2*w[3])
np.round(gradient_de_J1(w,x), 3)
```

$\ell(w,x,y) = \frac{1}{2}(w.x - y)^2$ (x : shape N)

$\frac{\partial}{\partial w} \ell_i = (w.x - y)(x)$ (D : (N,D) ≈ x@w ; ↑ * de la dérivée)

```
def gradient_de_ell(w,x,y):
    return (x@w - y)*x
```

Cours / TDs :

I. Qq. notions de base

→ Types d'apprentissage

- supervisé : on connaît la 'vraie' sortie y_n pour chaque entrée x_n (ex : regression, classification)
- non supervisé : pas de sortie y (on cherche une structure dans P(X))

→ représentation des données :

données. $X := \{\vec{x}^{(1)}, \_\_\_, \vec{x}^{(n)}\} \in \mathbb{R}^{N \times D}$

N. nombre d'échantillons (ligne de X)

D : nombre de features (colonnes de X)

$x_{n,d}$: valeur en ligne n, colonne d

sorties : $Y = (y_1, \_\_\_, y_n)$

GT = valeur réelle de y_n

1) Regression linéaire :

$f_w(\vec{x}) = w_0 + \sum_{d=1}^{D} w_{1,d} x_d = w_0 + \vec{w_1} \cdot \vec{x}$

prédiction : $\hat{y} = f_w(\vec{x})$

pour un même x, y varie : bruit ou manque d'information

→ on modélise $y = f_w(x) + \varepsilon$ ou $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ (↓ $1/\sqrt{D}$)

2) Fonction coût et optimisation :

$J(w,X,Y) = \frac{1}{N} \sum_{n=1}^{N} (f_w(x_n) - y_n)^2$

fit = entraîner le modèle = on cherche les param qui minimisent l'erreur de prédiction.

Ex 2 du TD

1) $J_1(\vec{\theta}) = \frac{1}{2}\|\vec{\theta}\|^2 = \frac{1}{2}\vec{\theta}.\vec{\theta} = \frac{1}{2}(\theta_1^2 + \_\_\_ + \theta_D^2)$

$\frac{\partial}{\partial \theta_i} J_1 = \theta_i$

$\Rightarrow \boxed{\nabla_{\vec{\theta}} J_1 = \vec{\theta}}$

2) $J_2(\vec{\theta}) = \frac{1}{4}\|\vec{\theta}\|_4^4 = \frac{1}{4}(\theta_1^4 + \_\_\_ + \theta_D^4)$

$\frac{\partial J_2}{\partial \theta_i} = \theta_i^3$

$\Rightarrow \boxed{\vec{\nabla} J_2(\vec{\theta}) = (\theta_1^3, \_\_\_, \theta_D^3)}$

3) $J_3(\vec{\theta}) = \frac{1}{2}\frac{1}{N}\sum_n^N (\vec{\theta}.\vec{x_n} - y_n)^2$

$= \frac{1}{2N}\sum_n^N \left(\sum_j^D \theta_j x_{nj}\right) - y_n)^2$ ← fonction de coût pour la regression linéaire

$\sum_j^D \frac{\partial}{\partial \theta_i}(\theta_j x_{nj}) - y_n$

$= x_i$

$\boxed{\nabla_{\vec{\theta}} J_3 = \frac{1}{N}\sum_n^N (\vec{\theta}.\vec{x_n} - y_n)\vec{x_n}}$

$X \in M_{N,D}(\mathbb{R}) = \begin{pmatrix} x_{1,1} & \_\_\_ & x_{1,D} \\ x_{n,1} & & \vdots \\ x_N & \_\_\_ & x_{ND} \end{pmatrix}$ $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$

4) $J_4(\vec{\theta}) = \frac{1}{4}.\frac{1}{N}\sum_n^N (\vec{\theta}.\vec{x_n} - y_n)^4$

$= \frac{1}{4}.\frac{1}{N}\sum_{n=1}^{N} \frac{\partial}{\partial \theta_i}\left[\left(\left(\sum_{j=1}^{D} \theta_j x_{nj}\right) - y_n\right)^2\right]$

$\vec{\nabla}_{\vec{\theta}} J_4 = \frac{1}{4N}\sum_n^N 4(\vec{\theta}.\vec{x_n} - y_n)^3 \vec{x_n}$

$\boxed{= \frac{1}{N}\sum_n^N (\vec{\theta}.\vec{x_n} - y_n)^3 \vec{x_n}}$

optionnel : $J_5(\vec{\theta}) = \|\vec{\theta}\|_1 = |\theta_1| + \_\_\_ + |\theta_D|$

$\frac{\partial J_5}{\partial \theta_i} = \begin{cases} 1 \text{ si } \theta_i > 0 \\ -1 \text{ si } \theta_i < 0 \end{cases} = \text{sign}(\theta_i)$

$\Rightarrow \vec{\nabla}_{\vec{\theta}} J_5 = \text{sign}(\vec{\theta})$

Partie II : descente de gradient

Annale 2023-2024 :

$f_{\vec{w}}(\vec{x_n}) = \vec{w} \cdot \vec{x_n}$

```
def fw(w,X):
    return X@w
```

~> regression car les valeurs prises par Y sont continues et non discrètes

~> modèle avec biais (caché dans la colonne 1 de X) ~> c'est un modèle affine

(si 1e colonne = 1 de pour tous les échantillons, alors c'est un modèle affine ($f(x) = \vec{w}X$ avec $\vec{w} = (b, w_1, w_2, \dots)$

```
# calcule le terme (w.xn - yn)^3 xn pour n fixé
def calcul_intermediaire(X,Y,n,w):
    a = (w@X[n] - Y[n])**3 * X[n]
```

(ou : N = X.shape[0]

a = ((X@w - Y)**3).reshape(N,1) * X)[n]) ~> resultat de shape (D,) à la fin (= shape que w toujours !)

~> resultat de shape (N, D)

→ Calcul du gradient : ici, on code $\nabla_w J(w,X,Y) = -\alpha w + \frac{1}{N}\sum_n (w.x_n - y_n)^3 x_n$

```
def gradient_de_J(w,X,Y,alpha):
    N = X.shape[0]
    D = X.shape[1]
    g = np.zeros(D)
    for n in range(N):
        g += calcul_intermediaire(X,Y,n,w)
    g /= N
    g -= alpha*w
    return g
```

```
N = X.shape[0]
g = (((X@w - Y)**3).reshape(N,1) * X).sum(axis=0) / N - alpha*w
return g

g = np.mean(((X@w - Y)**3).reshape(N,1) * X), axis=0)
g -= alpha*w
```

Application descente du gradient :

```
def fit(X,Y,w0,eta,ITERMAX,alpha):
    w = w0.copy()  # ici, c'était déjà initialisé
    for i in range(ITERMAX):
        w -= eta * gradient_de_J(w,X,Y,alpha)
    return w
```

II. Partage des données (train / validation split)

```
X_train, X_valid, Y_train, Y_valid = sklearn.model_selection.train_test_split(X,Y,test_size=0.2, random_state=6)
```

```
# fonction pour Mean Absolute error  MAE = 1/N Σ_{n=1}^{N} |w.xn - yn|
def fonction_MAE(w,X,Y):
    MAE = np.abs(X@w - Y)
    MAE = np.mean(MAE)
    return MAE.
```

```
# intégration dans la fonction fit
(ajout des param : X_valid, Y_valid
def fit_amelioree(X,Y,X_valid,Y_valid,w0,eta,
ITERMAX,alpha):
    MAE_train = np.zeros(ITERMAX)
    MAE_valid = np.zeros(ITERMAX)
    w = w0.copy()
    for i in range(ITERMAX):
        w -= eta * gradient_de_J(w,X,Y,alpha)
        MAE_train[i] = fonction_MAE(w,X,Y)
        MAE_valid[i] = fonction_MAE(w,X_valid,Y_valid)
return w, MAE_train, MAE_valid.
```

* Analyse graphes + (notes personnelles)

- η trop petit - descente lente
- η bien choisi - convergence stable
- η légèrement trop grand - oscillations
- η trop grand - divergence
- ITERMAX trop petit - arrêt prématuré
- overfitting : train ↓ valid ↑

en général :
fonction fw
calcul du gradient
descente de gradient
entraînement avec le modèle crée.

Annale 2022-2023 :

```
def fw(w,x):
    return x@w
```

pour $\vec{\nabla}_{\vec{w}} J(\vec{w},X,Y) = \frac{1}{N}\sum_n (fw(x_n)-y_n)^3 \vec{x_n}$

```
def gradient_de_J(w,X,Y):
    N = X.shape[0]
    return (1/N) * (((fw(w,X)-Y)) **3) @X
```

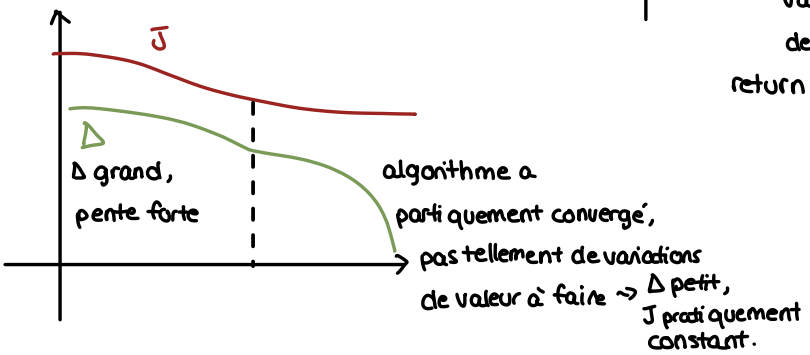
ou : `grad = (((fw(w,X)-Y)**3).reshape(N,1) * X).mean(axis=0)`

initialisation de w avec la loi normale :

```
def initialiser_w(D):
    return np.random.normal(0, D**-0.5, D)
```

fonction fit

```
def fit(X,Y,eta,ITERMAX):
    D = X.shape[1]
    w = initialiser_w(D)
    for i in range(ITERMAX):
        w -= eta * gradient_de_J(w,X,Y)
    return w
```

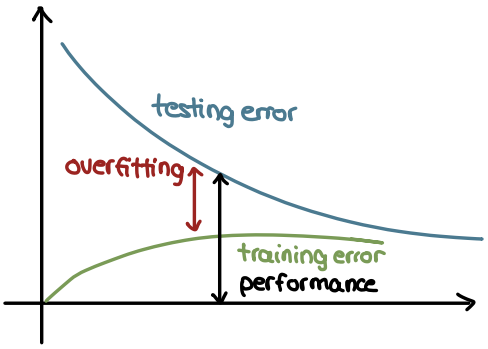


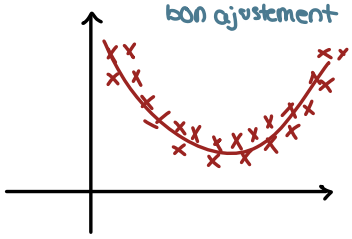
fit, avec les valeurs de J et $\eta \|\vec{\nabla}_{\vec{w}} J(\vec{w})\|$

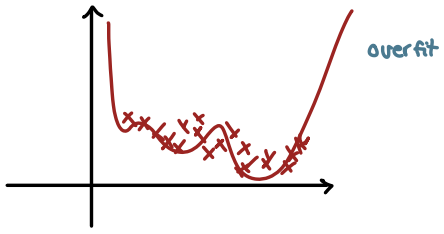
$J(\vec{w},X,Y) = \frac{1}{4N}\sum_n (fw(x_n)-y_n)^4$

```
def fit2(X,Y,eta,ITERMAX):
    D = X.shape[1]
    w = initialiser_w(D)
    valeurs_de_J = np.zeros(ITERMAX)
    deplacements_de_J = np.zeros(ITERMAX)
    for i in range(ITERMAX):
        delta = eta * gradient_de_J(w,X,Y)
        w -= delta
        valeurs_de_J[i] = valeur de J(w,X,Y)
        deplacements_de_J[i] = np.linalg.norm(delta)
    return w, valeurs_de_J, deplacements de J
```

Quelques graphes :







(+ perceptron)

TP : `def perceptron_FullBatch_version_minimale (X,Y,eta,maxITER=20, plot=None, verbose=None, Loss=None):`

```
    w = w0.copy()
    for itera in range(maxIter):
        ListeDesMalClasses = ((X@w) * Y <= 0)
        if ListeDesMalClasses.sum()==0:
            return w
        w = w+ eta * 1/N * X[ListeDesMalClasses].T @ Y[ListeDesMalClasses]
    return w
```

I : l'ensemble des indices des exemples mal classés

$J(\vec{w},X,Y) = \frac{1}{N}\sum_{n=1} \max(0, -(\vec{w}.\vec{x}^{(n)})y_n) = \frac{-1}{N}\sum_{n\in I} (\vec{w}.\vec{x}^{(n)})$

↝ Mise à jour des paramètres par descente de gradient

η taux d'apprentissage

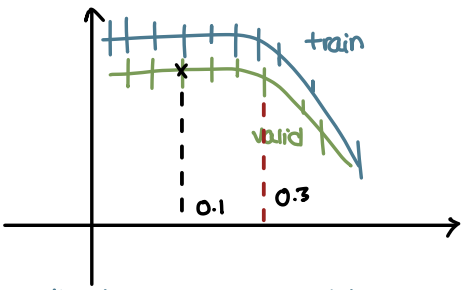
$\vec{w} \leftarrow \vec{w} - \eta \vec{\nabla}_{\vec{w}} J(\vec{w},X,Y)$

+ voir 2022-S4
+ TP dernier

Partie III : optimisation de 1 hyper-paramètre

Annale 2024-2025 :

```
# chargement des données (donné)
X, y = sklearn.datasets.load_digits(n_class=10, return_X_y=True)
X_trainval, x_test, y_trainval, y_test = sklearn.model_selection.train_test_split(x, y, test_size=0.2, random_state=4)

# plage de valeurs du paramètre
plage_de_valeurs_pour_alpha = np.logspace(-2, 4, num=20)   [entre -2 et 4]
monModele = sklearn.linear_model.Ridge(alpha=10)

# cross_validation:                                  ~> renvoi un dictionnaire contenant 4 tableaux :
nombredePlisDeCV = 5                                    fit_time ~> temps d'entraînement par plis
sklearn.model_selection.cross_validate(monModele, x_trainval, y_trainval,   score_time ~> temps d'évaluation
                cv=nombredeplisDeCV, return_train_score=true)               train_score ~> score sur l'ensemble d'apprentissage du pli
                                                                            test_score ~> score de validation
                                                                        Chaque tableau contient une valeur par pli.

# train_score et test_score pour chaque α dans la plage et les 5 plis     α1 ..... αn
nb plis CV = 5
trainScore = np.zeros((plage_de_valeurs_pour_alpha.shape[0], nbdeplisCV))
validScore = _____
for i, alpha in enumerate(plage_de_valeurs_pour_alpha):
    monModele = sklearn.linear_model.Ridge(alpha=alpha)
    myCV = sklearn.model_selection.cross_validate(monModele, x_trainval, y_trainval, cv=nbdePlisCV, return_train_score=true)
    trainScore[i] = myCV['train_score']
    validScore[i] = myCV['test_score']

# plot des barres d'erreurs
plt.figure()
plt.errorbar(plage_de_valeurs_pour_alpha, np.mean(trainScore, axis=1), label="train score", yerr=np.std(trainScore, axis=1))
plt.errorbar(plage_de_valeurs_pour_alpha, np.mean(validScore, axis=1), label="valid score", yerr=np.std(validScore, axis=1))
plt.xlabel("alpha")        plt.legend()
plt.ylabel("scores")       plt.semilogx()

# affichage du meilleur point :
linear_valid_score = np.mean(validScore, axis=1)
bestIndex = np.argmax(linear_valid_score)
best NC = plage_de_valeurs_pour_alpha[bestIndex]
plt.plot(best NC, linear_valid_score[bestIndex], marker='X', color='green')
```



~> on peut prendre le meilleur validation score
~> si les données sont petites, α contrôle le degré de régularisation,
10^3 : meilleure généralisation (on regularize plus), overfit (espace entre train et valid) est plus petit, barre d'erreur plus petite.

```
# entrainement du modèle sur le meilleur paramètre trouvé :
monModele = sklearn.linear_model.Ridge(alpha=1000)
monModele.fit(x_trainval, y_trainval)
monModele.score(x_trainval, y_trainval), monModele.score(x_test, y_test)
```

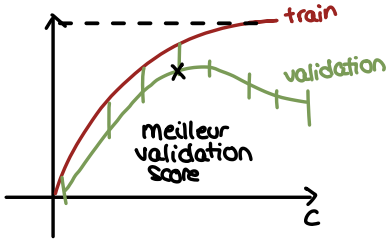
```
plt.scatter(y_test, monModele.predict(x_test))
```
~> x vraie valeur, y valeur predite

```
np.unique(y_trainval)
```
~> entiers
~> classification...

Annale 2022-2023 :

```
X, Y = sklearn.datasets.load_digits(n_class=10, return_X_y=True)
X_trainval, X_test, Y_trainval, Y_test = sklearn.model_selection.train_test_split(X, Y, test_size=0.2, random_state=7)
# exemple pour C=1
C = 1
monModele = sklearn.svm.LinearSVC(C=C, max_iter=1000, dual=False)
monModele.fit(X_trainval, Y_trainval)
monModele.score(X_trainval, Y_trainval)

# faire varier C puis crossfit avec les train et validScore
plage_de_valeurs_pour_C = np.logspace(-6, 2, num=9)   # (entre 10^-6 et 10^2)
nbplisCV = 5
trainScore = np.zeros((plage_de_valeurs_pour_C.shape[0], nbplisCV))
validScore = ____________________________
for i, C in enumerate(plage_de_valeurs_pour_C):
    mon_modele = sklearn.svm.LinearSVC(C=C, max_iter=1000, dual=False)
    myCV = sklearn.model_selection.cross_validate(monModele, X_trainval, Y_trainval, cv=nbplisCV, return_train_score=True)
    trainScore[i] = myCV['train_score']
    validScore[i] = myCV['test_score']
```

```
# plot
plt.figure()
plt.errorbar(plage_de_valeurs_pour_C, np.mean(trainScore, axis=1), label="trainscore", yerr=np.std(trainScore, axis=1))
plt.errorbar(plage_de_valeurs_pour_C, np.mean(validScore, axis=1), label="validscore", yerr=np.std(validScore, axis=1))
plt.axhline(1, c='k', ls='--') + # affichage du meilleur point
```



intervalle attendue : ↓ testscore dans la barre d'erreur

```
C_choisi = 10**-3
validScore[plagede_valeurs_pour_C == C_choisi]
(validScore[plagede_valeurs_pour_C == C_choisi].min(),
                                validScore[plagede_valeurs_pour_C == C_choisi].max())]
```

intervalle attendue

```
C = C_choisi
monModele = sklearn.svm.LinearSVC(C=C, max_iter=1000, dual=False)
monModele.fit(X_trainval, Y_trainval)
monModele.score(X_trainval, Y_trainval), monmodele.score(X_test, Y_test)
```

---

* gradient cheat sheet :

let $X \in \mathbb{R}^{N \times D}$, $Y \in \mathbb{R}^{N}$, $\omega \in \mathbb{R}^{D}$, $f_\omega(X) = X\omega$, $N = X.shape[0]$

1) MSE : $J = \frac{1}{2N} \| X\omega - Y \|^2$

$\nabla_\omega J = \frac{1}{N} X^T (X\omega - Y)$

matrix: grad = (X.T @ (X @ ω - Y)) / N

mean: grad = np.mean((X @ ω - Y).reshape(-1, 1) * X, axis=0) (N,1)

2) Ridge : $J = \frac{1}{2N} \| X\omega - Y \|^2 + \frac{\alpha}{2} \| \omega \|^2$

$\nabla_\omega J = \frac{1}{N} X^T (X\omega - Y) + \alpha\omega$

matrix: grad = (X.T @ (X @ ω - Y)) / N + alpha * ω

mean: grad = np.mean((X @ ω - Y).reshape(-1, 1) * X, axis=0) + alpha * ω

3) MAE : $J = \frac{1}{N} \sum_i |(X\omega - Y)_i|$

$\nabla_\omega J = \frac{1}{N} X^T \, sign(X\omega - Y)$

Matrix: grad = (X.T @ np.sign(X @ ω - Y)) / N

mean : grad = np.mean(np.sign(X @ ω - Y).reshape(-1, 1) * X, axis=0)

4) Logistic (BCE) : $J = -\frac{1}{N} \sum_i [y \log \sigma_i + (1 - y_i) \log(1 - \sigma_i)]$

$\sigma(X\omega) = \frac{1}{1 + e^{-(X\omega)}}$

$\nabla_\omega J = \frac{1}{N} X^T (\sigma(X\omega) - Y)$

matrix: y_pred = 1/(1 + np.exp(-(X @ ω)))

grad = (X.T @ (y_pred - Y)) / N

mean: grad = np.mean((y_pred - Y).reshape(-1, 1) * X, axis=0)

5) Lasso (L1) : $J = \frac{1}{2N} \|Xw - y\|^2 + \alpha \|w\|_1$

$\nabla_w J = \frac{1}{N} X^T (Xw - y) + \alpha \, \text{sign}(w)$

matrix: grad = (X.T @ (X @ w - y)) / N + alpha * np.sign(w)

6) Huber (δ)

$r = Xw - y \quad g_i = \begin{cases} r_i & |r_i| < \delta \\ \delta \, \text{sign}(r_i) & \text{sinon} \end{cases} \quad \nabla_w J = \frac{1}{N} X^T g$

Matrix: grad = (X.T @ g) / N Mean: grad = np.mean(g.reshape(-1,1) * X, axis=0)

Partie IV : exemples d'erreurs dans le code.

Annale 2024-2025:

```
clf = sklearn.svm.SVC(C=1, kernel='poly', degree=4)
clf.fit(X_train_transformed, Ytrain)
trainscore = clf.score(X_train_transformed, Ytrain)
clf.fit(X_valid_Transformed, Yvalid)
```

X il faut pas faire fit deux fois
+ il faut pas fit avec les données de validation

Annale 2022-2023:

```
prePoc = sklearn.decomposition.PCA(n_components = 10)
prePoc.fit(X_train)
X_transformed = preProc.transform(X_train)
X_valid_Transformed = preProc.transform(X_valid)
clf = sklearn.svm.SVC(C=1, kernel='poly', degree=3)
clf.fit(X_transformed, Y)
```

il faut entrainer le modèle sur les données transformées

⇒ clf.fit(X_train_Transformed, Y_train)

~> l'erreur était que on entrainait sur toutes les données (train + val) donc il n'y avait pas en fait de validation set.
Donc la performance de "validation" était la même qu'en train. Avec un split, comme attendu, la perf en validation est légèrement moins bonne.

NOTES. bien transformer le valid comme le train, only fit train, ne pas fit sur les données valides.

autres exemples :

type d'erreur	code erroné	correction	conséquence / explication
1. fit du modèle sur tout le jeu	clf.fit(X, Y)	clf.fit(X_train, Y_train)	Data Leakage : le modèle 'voit' les données de test ~> score irréaliste (trop bon)
2. fit du transformeur sur tout X	pca.fit(X)	pca.fit(X_train) puis pca.transform (X_valid) / X_test	fuite d'information : les composantes principales sont influencées par le test ~> biais dans la validation
3. Mauvais ordre fit / transform	X_train = pca.transform (X_train) pca.fit(X_train)	pca.fit(X_train) X_train = pca.transform (X_train)	erreur logique : le transform précède le calcul des composantes ~> résultats incohérents
4. Double fit du modèle	clf.fit(X_trainval) puis clf.fit(X_trainval)	faire un seul fit	le second fit écrase le 1er ~>incohérence, ou apprentissage sur le mauvais split
5. Evaluation biaisée	clf.score(X_trainval, Y_trainval)	clf.score(X_test, Y_test)	test non indépendant ~> on mesure la performance sur des données déjà vues
6. Transform après fit global	pca.fit_transform(X) puis split train/test	Split avant, puis fit_transform sur train	Les infos du test influencent la base du modèle ~> fuite de données
7. Transform seulement sur le train	pca.fit(X_train) mais X_test inchangé	X_test = pca.transform(X_test)	Le modèle apprend dans un espace transformé. Δ! les données de validation / test sont dans un espace différent ~> prédictions incohérentes

*train dans cross_validate