Prédiction des feux de forêt grâce au Machine Learning

Yann Millet - 43_923

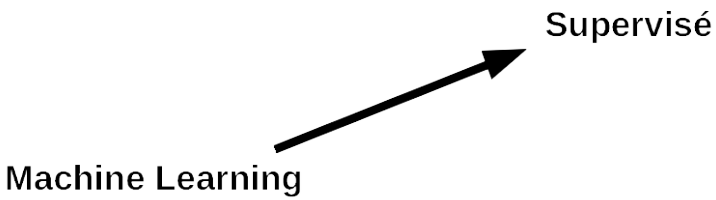
juin/juillet 2021

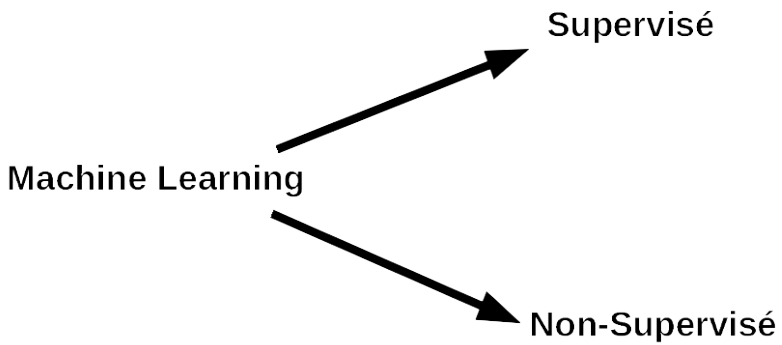
I] Présentation du projet

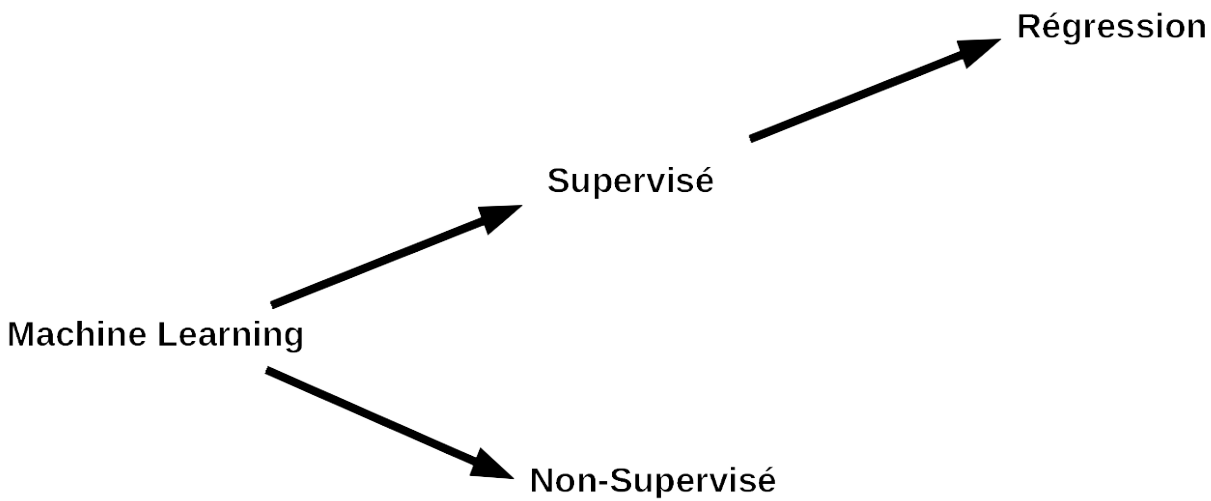
I] Présentation du projet
II] Les SVR à noyau

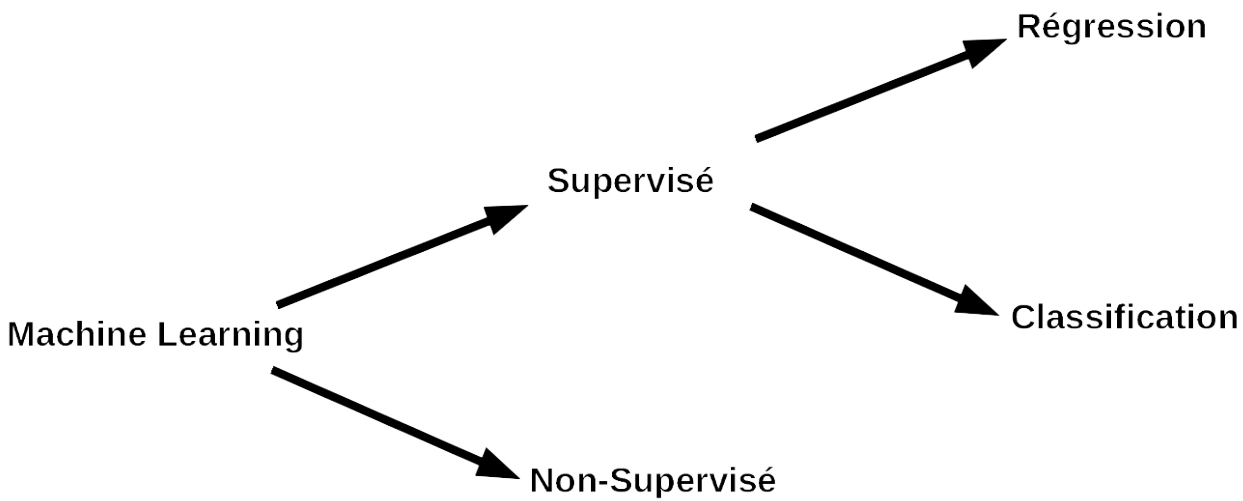
I] Présentation du projet
II] Les SVR à noyau
III] Mise en pratique

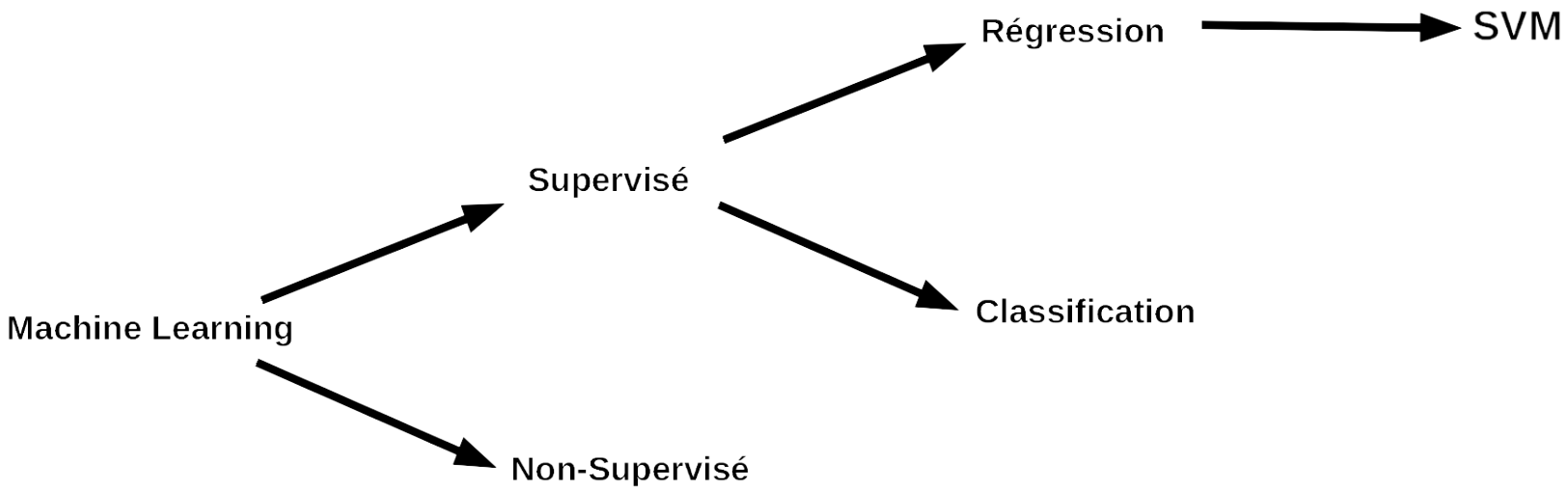
Machine Learning


Supervisé
Machine Learning


Supervisé
Machine Learning
Non-Supervisé


Régression
Supervisé
Machine Learning
Non-Supervisé


Régression
Supervisé
Classification
Machine Learning
Non-Supervisé


Machine Learning
Supervisé
Non-Supervisé
Régression
Classification
SVM

$$h(x) = \langle \omega | x \rangle + \beta$$

x vecteur d'entrée
ω poids de l'hyperplan
β intercept (ordonnée à l'origine)

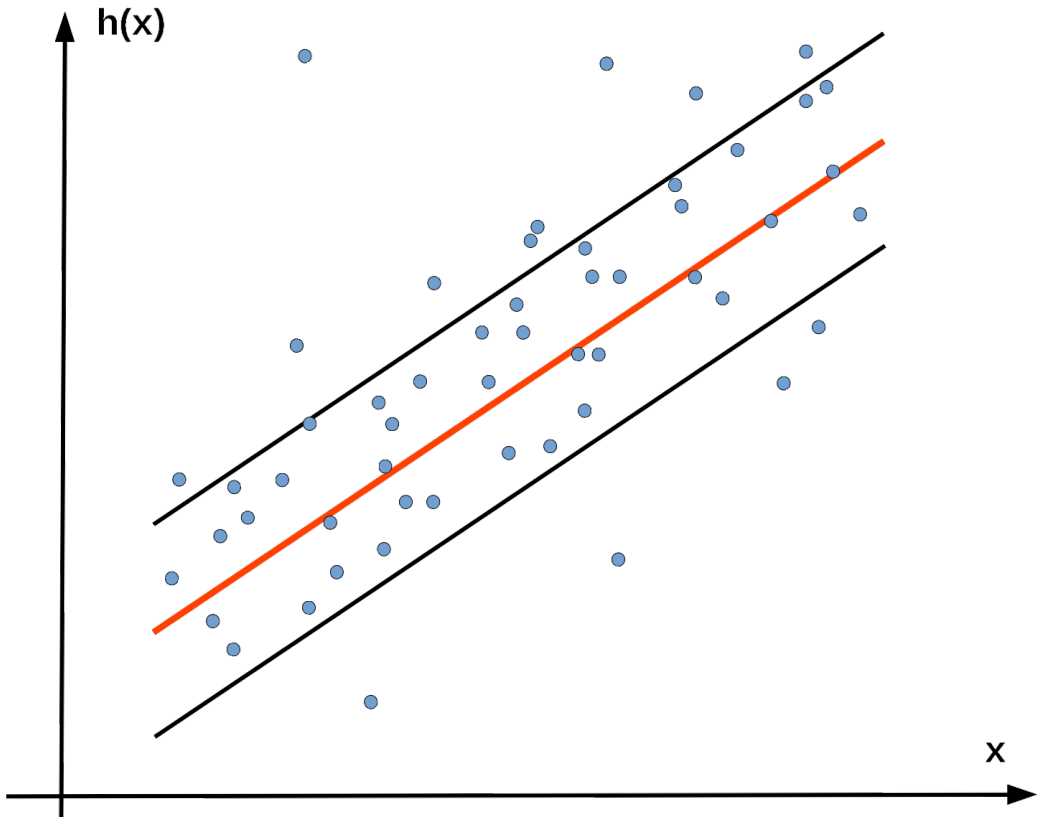
$$h(x) = \langle \omega | x \rangle + \beta$$

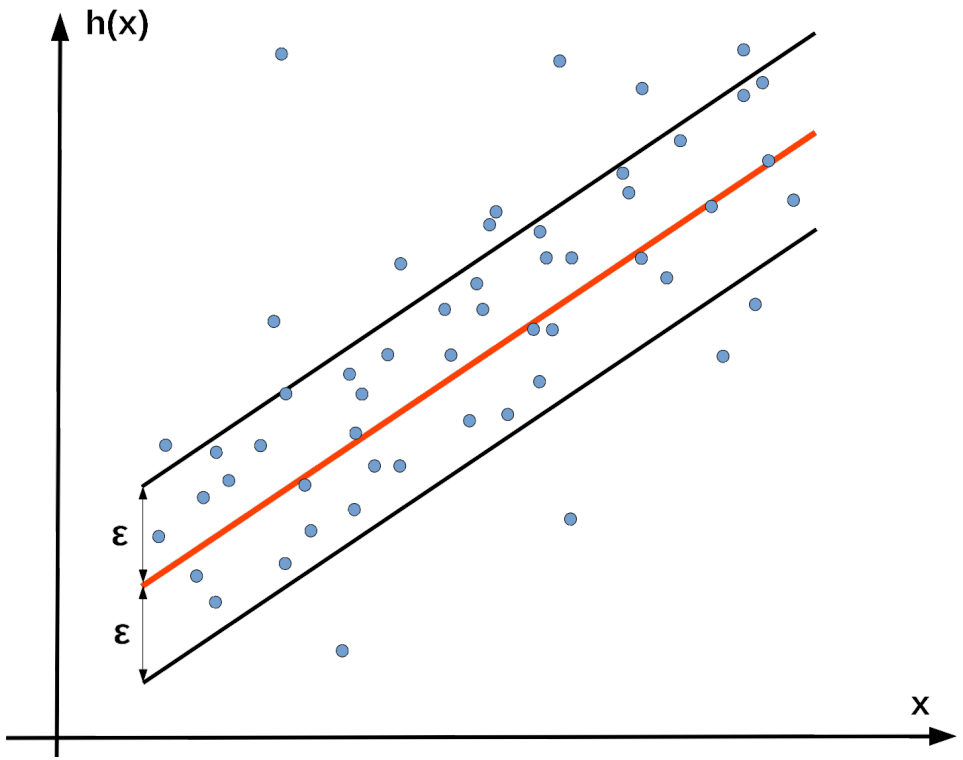
x vecteur d'entrée
ω poids de l'hyperplan
β intercept (ordonnée à l'origine)

Erreur quadratique :

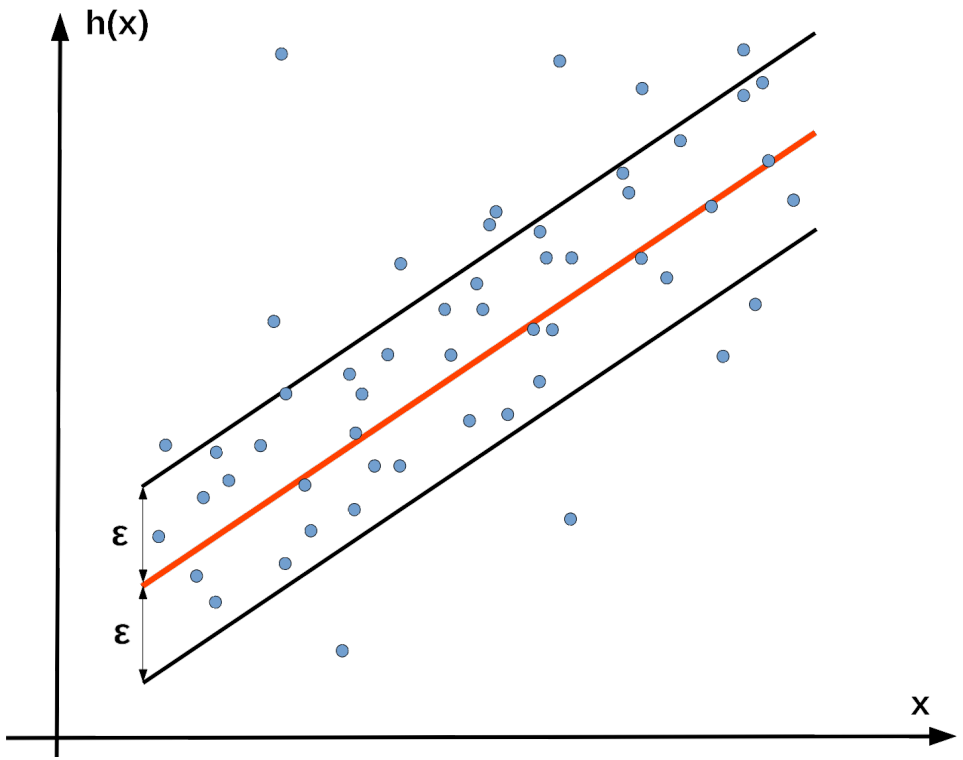
$$\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$h(x) = \langle \omega | x \rangle + \beta$$

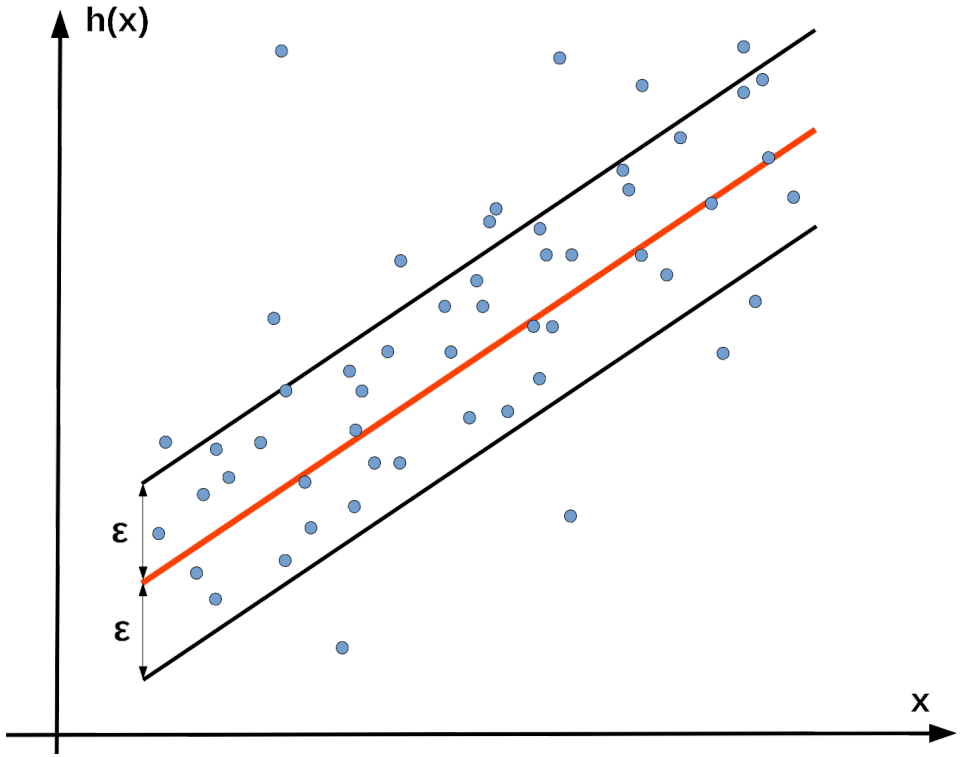


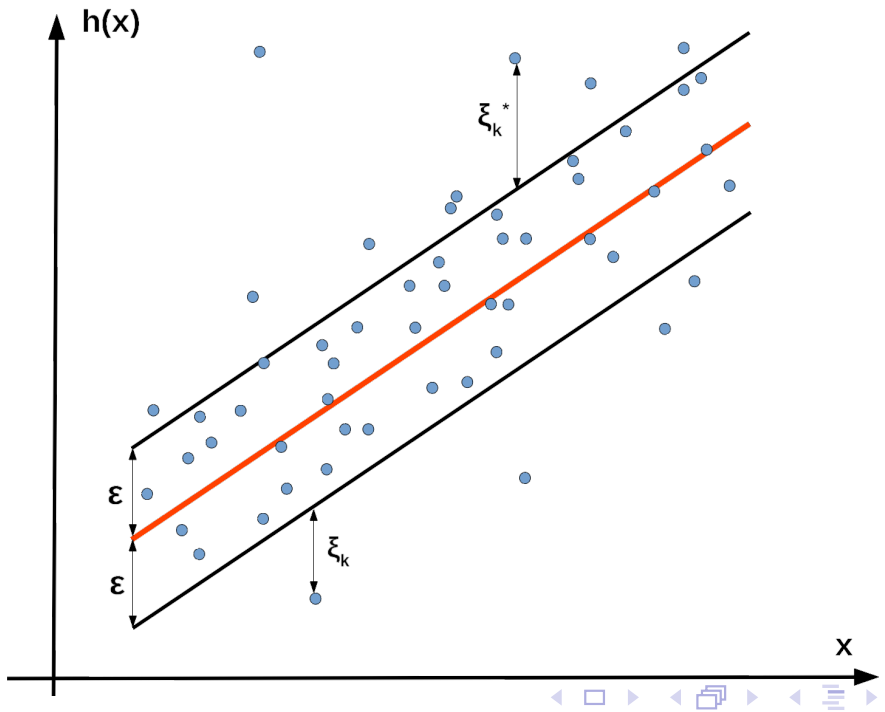

h(x)
ε
ε
x

$$\min_{\omega \in \mathbb{R}^n} \quad \frac{1}{2}\|\omega\|^2$$

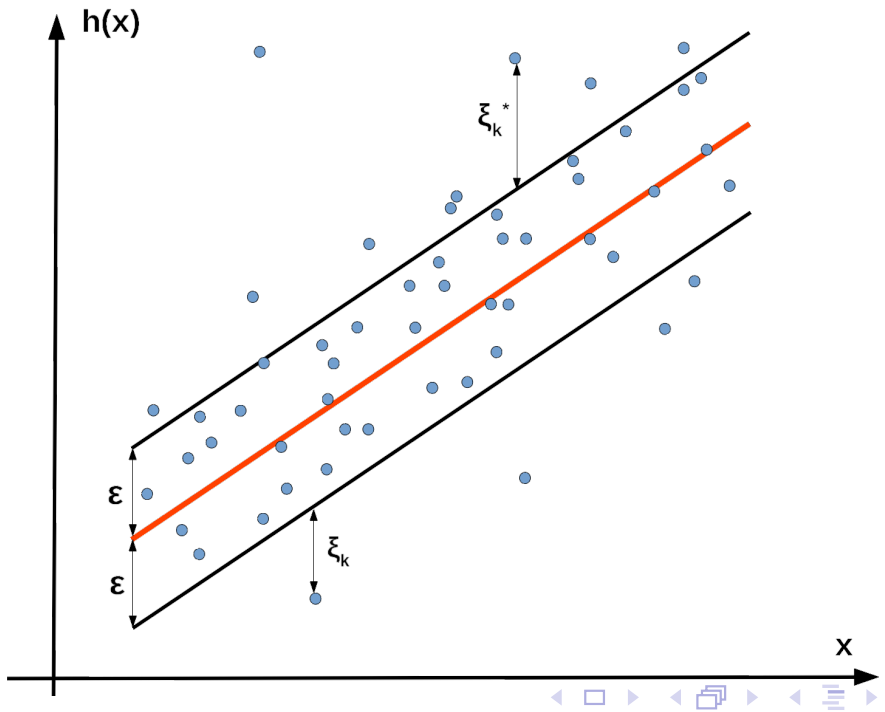


$$\min_{\omega \in \mathbb{R}^n} \quad \frac{1}{2}\|\omega\|^2 \qquad \begin{cases} \forall k \in [\![1, N]\!], & y_k - \hat{y}_k \leqslant \varepsilon \\ \forall k \in [\![1, N]\!], & \hat{y}_k - y_k \leqslant \varepsilon \end{cases}$$



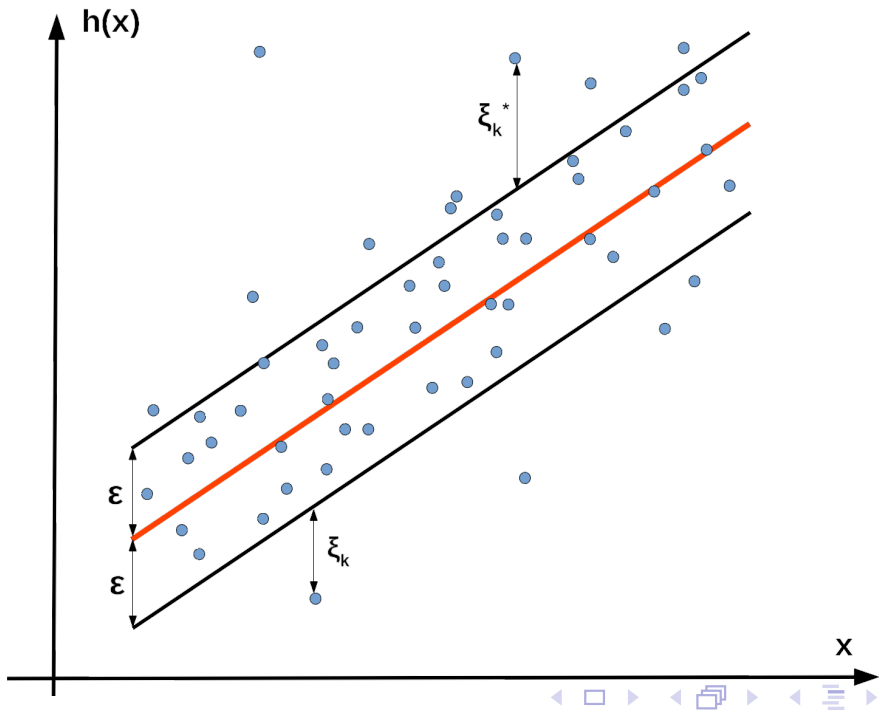

h(x)
ξ_k^*
ε
ε
ξ_k
x

$$\min_{\omega \in \mathbb{R}^n} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{N} \xi_k + \xi_k^*$$



$$\min_{\omega \in \mathbb{R}^n} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{N} \xi_k + \xi_k^*$$

$$\begin{cases} \forall k \in [\![1, N]\!], & y_k - \hat{y}_k \leqslant \varepsilon + \xi_k \\ \forall k \in [\![1, N]\!], & \hat{y}_k - y_k \leqslant \varepsilon + \xi_k^* \\ \forall k \in [\![1, N]\!], & \xi_k, \xi_k^* \geqslant 0 \end{cases}$$



$$\begin{aligned} F : \mathbb{R}^n &\longrightarrow \mathbb{R}^M \\ v &\longmapsto (F_i(v))_{1 \leqslant i \leqslant M} \end{aligned}$$

$$\begin{aligned} F : \mathbb{R}^n &\longrightarrow \mathbb{R}^M \\ v &\longmapsto (F_i(v))_{1 \leqslant i \leqslant M} \end{aligned}$$

$$K = \{v \in V \mid F_i(v) \leqslant 0 \ \ \forall i \in [\![1, M]\!]\}$$

$$\begin{aligned} F : \mathbb{R}^n &\longrightarrow \mathbb{R}^M \\ v &\longmapsto (F_i(v))_{1 \leqslant i \leqslant M} \end{aligned}$$

$$K = \{v \in V \mid F_i(v) \leqslant 0 \ \ \forall i \in [\![1, M]\!]\}$$

$$\forall (v, q) \in V \times \mathbb{R}^M , \quad \mathscr{L}(v, q) = J(v) + \langle q | F(v) \rangle$$

$$\begin{aligned} F : \mathbb{R}^n &\longrightarrow \mathbb{R}^M \\ v &\longmapsto (F_i(v))_{1 \leqslant i \leqslant M} \end{aligned}$$

$$K = \{v \in V \mid F_i(v) \leqslant 0 \ \ \forall i \in [\![1, M]\!]\}$$

$$\forall (v, q) \in V \times \mathbb{R}^M , \quad \mathscr{L}(v, q) = J(v) + \langle q | F(v) \rangle$$

$$\forall (v, q) \in V \times \mathbb{R}^M \ , \ \mathscr{L}(u, q) \leqslant \mathscr{L}(u, p) \leqslant \mathscr{L}(v, p)$$

$$\forall (v,q) \in V \times \mathbb{R}^M \ , \ \mathscr{L}(u,q) \leqslant \mathscr{L}(u,p) \leqslant \mathscr{L}(v,p)$$

$$\forall (v,q) \in V \times \mathbb{R}^M \ , \ \mathscr{L}(u,q) \leqslant \mathscr{L}(u,p) \leqslant \mathscr{L}(v,p)$$

$$\boxed{\mathscr{I}(v) = \sup_{q \in \mathbb{R}_+^M} \mathscr{L}(v,q)}$$

$$\forall (v,q) \in V \times \mathbb{R}^M \ , \ \mathscr{L}(u,q) \leqslant \mathscr{L}(u,p) \leqslant \mathscr{L}(v,p)$$

$$\boxed{\mathscr{I}(v) = \sup_{q \in \mathbb{R}_+^M} \mathscr{L}(v,q)} \qquad \boxed{\mathscr{G}(q) = \inf_{v \in U} \mathscr{L}(v,q)}$$

$$\forall (v,q) \in V \times \mathbb{R}^M \ , \ \mathscr{L}(u,q) \leqslant \mathscr{L}(u,p) \leqslant \mathscr{L}(v,p)$$

$$\boxed{\mathscr{I}(v) = \sup_{q \in \mathbb{R}^M_+} \mathscr{L}(v,q)} \qquad \boxed{\mathscr{G}(q) = \inf_{v \in U} \mathscr{L}(v,q)}$$

Theorem

(De dualité) Le couple (u,p) est un point-selle de $\mathscr{L}$ sur $U \times \mathbb{R}^M_+ \subset V \times \mathbb{R}^M$ si et seulement si :

$$\mathscr{I}(u) = \min_{v \in U} \left(\sup_{q \in \mathbb{R}^M_+} \mathscr{L}(v,q) \right) = \max_{q \in \mathbb{R}^M_+} \left(\inf_{v \in U} \mathscr{L}(v,q) \right) = \mathscr{G}(p)$$

$$K = \prod_{i=1}^{n} [\![a_i, b_i]\!]$$

$$K = \prod_{i=1}^{n} [\![a_i, b_i]\!]$$

$$\begin{cases} \mathscr{L}(u_n, p_n) = \inf_{v \in V} \mathscr{L}(v, p_n) \\ \\ p_{n+1} = P_{\mathbb{R}_+^M}(p_n + \mu\, F(u_n)) \end{cases}$$

Algorithme d'Uzawa : méthode du gradient avec projection pour le dual.

$$\mathscr{G}(q) = \inf_{v \in V} \mathscr{L}(v, q)$$

$$p_{n+1} = P_{\mathbb{R}_+^M}(p_n + \mu\, \mathscr{G}'(p_n))$$

$$primal : \begin{cases} \min\left(\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{N}\xi_k + \xi_k^*\right) \\ \forall k \in [\![1, N]\!], \quad y_k - \langle\omega|x_k\rangle - \beta \leqslant \varepsilon + \xi_k \\ \forall k \in [\![1, N]\!], \quad \langle\omega|x_k\rangle + \beta - y_k \leqslant \varepsilon + \xi_k^* \\ \forall k \in [\![1, N]\!], \quad \xi_k, \xi_k^* \geqslant 0 \ , \ \xi_k\xi_k^* = 0 \end{cases}$$

$$primal : \begin{cases} \min \left(\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{N} \xi_k + \xi_k^* \right) \\ \forall k \in [\![1, N]\!], \quad y_k - \langle \omega | x_k \rangle - \beta \leqslant \varepsilon + \xi_k \\ \forall k \in [\![1, N]\!], \quad \langle \omega | x_k \rangle + \beta - y_k \leqslant \varepsilon + \xi_k^* \\ \forall k \in [\![1, N]\!], \quad \xi_k, \xi_k^* \geqslant 0 \ , \ \xi_k \xi_k^* = 0 \end{cases}$$

$$dual : \begin{cases} \max \left(\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i | x_j \rangle \right. \\ \qquad \left. -\varepsilon \sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*) \right) \\ \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \\ \\ \forall i \in [\![1, N]\!] \ , \ 0 \leqslant \alpha_i, \alpha_i^* \leqslant C \ , \ \alpha_i \alpha_i^* = 0 \end{cases}$$

ϕ

$$\phi : \mathbb{R}^n \longmapsto$$

$$\phi : \mathbb{R}^n \longmapsto \mathbb{R}^{n+m}$$

$$\phi : \mathbb{R}^n \longmapsto \mathbb{R}^{n+m}$$

$$primal : \begin{cases} \min \left(\frac{1}{2}\|\omega\|^2 + C \sum_{i=1}^{N} \xi_k + \xi_k^* \right) \\ \forall k \in [\![1, N]\!], \quad y_k - \langle \omega | \phi(x_k) \rangle - \beta \leqslant \varepsilon + \xi_k \\ \forall k \in [\![1, N]\!], \quad \langle \omega | \phi(x_k) \rangle + \beta - y_k \leqslant \varepsilon + \xi_k^* \\ \forall k \in [\![1, N]\!], \quad \xi_k, \xi_k^* \geqslant 0 \ , \ \xi_k \xi_k^* = 0 \end{cases}$$

$$dual : \begin{cases} \max \left(\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle \phi(x_i) | \phi(x_j) \rangle \right. \\ \qquad \left. -\varepsilon \sum_{i=1}^{N} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i (\alpha_i - \alpha_i^*) \right) \\ \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0 \\ \forall i \in [\![1, N]\!] \ , \ 0 \leqslant \alpha_i, \alpha_i^* \leqslant C \ \ , \ \ \alpha_i \alpha_i^* = 0 \end{cases}$$

$$primal : \begin{cases} \min \left(\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{N} \xi_k + \xi_k^* \right) \\ \forall k \in [\![1, N]\!], \quad y_k - \langle w | \phi(x_k) \rangle - \beta \leqslant \varepsilon + \xi_k \\ \forall k \in [\![1, N]\!], \quad \langle w | \phi(x_k) \rangle + \beta - y_k \leqslant \varepsilon + \xi_k^* \\ \forall k \in [\![1, N]\!], \quad \xi_k, \xi_k^* \geqslant 0 \ , \ \xi_k \xi_k^* = 0 \end{cases}$$

$$dual : \begin{cases} \max \left(\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle \phi(x_i) | \phi(x_j) \rangle \right. \\ \qquad \left. -\varepsilon \sum_{i=1}^{N} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i (\alpha_i - \alpha_i^*) \right) \\ \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0 \\ \forall i \in [\![1, N]\!] \ , \ 0 \leqslant \alpha_i, \alpha_i^* \leqslant C \ , \ \alpha_i \alpha_i^* = 0 \end{cases}$$

Theorem

(de Mercer) Si K est une fonction continue, symétrique et semi-définie positive sur un espace E, alors K s'exprime comme un produit scalaire d'un espace de grande dimension.

Theorem

(de Mercer) Si K est une fonction continue, symétrique et semi-définie positive sur un espace E, alors K s'exprime comme un produit scalaire d'un espace de grande dimension.

$$\forall (x_1, x_2) \in E \ , \ K(x_1, x_2) = \langle \phi(x_1) | \phi(x_2) \rangle$$

Theorem

(de Mercer) Si K est une fonction continue, symétrique et semi-définie positive sur un espace E, alors K s'exprime comme un produit scalaire d'un espace de grande dimension.

$$\forall (x_1, x_2) \in E \ , \ K(x_1, x_2) = \langle \phi(x_1) | \phi(x_2) \rangle$$

$$\forall (x_1, x_2) \in (\mathbb{R}^n)^2 \ , \ K(x_1, x_2) = e^{-\gamma \|x_1 - x_2\|^2}$$

Complexité :

$$\Theta(n_{caracteristique} \times n_{echantillon}^2) \leqslant C(n_c, n_e) \leqslant \Theta(n_{caracteristique} \times n_{echantillon}^3)$$

Prédiction :

$$\hat{y}(x) = \sum_{i \in SV} (\alpha_i - \alpha_i^*) \langle \phi(x_i) | \phi(x) \rangle + \beta$$

1. 12 caractéristiques

1. 12 caractéristiques (position

1. 12 caractéristiques (position , date

1. 12 caractéristiques (position , date , données météorologiques)

1. 12 caractéristiques (position , date , données météorologiques)
2. 1 résultat

1. 12 caractéristiques (position , date , données météorologiques)
2. 1 résultat (surface brûlée en hectares)

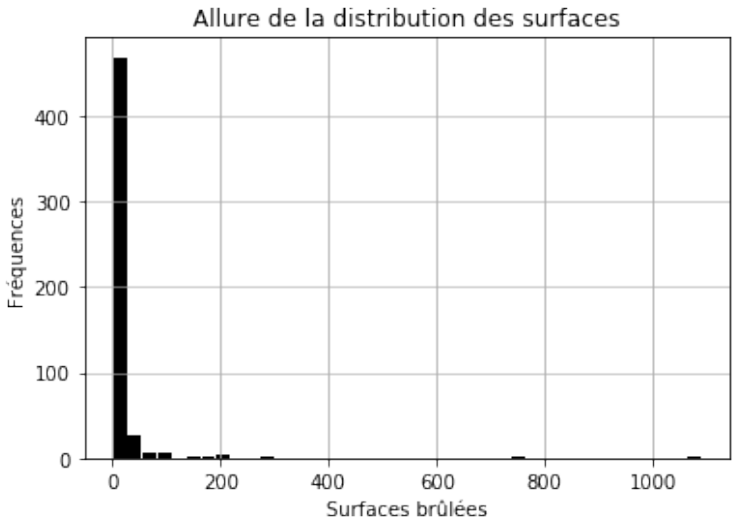
1. 12 caractéristiques (position , date , données météorologiques)
2. 1 résultat (surface brûlée en hectares)
3. 517 échantillons

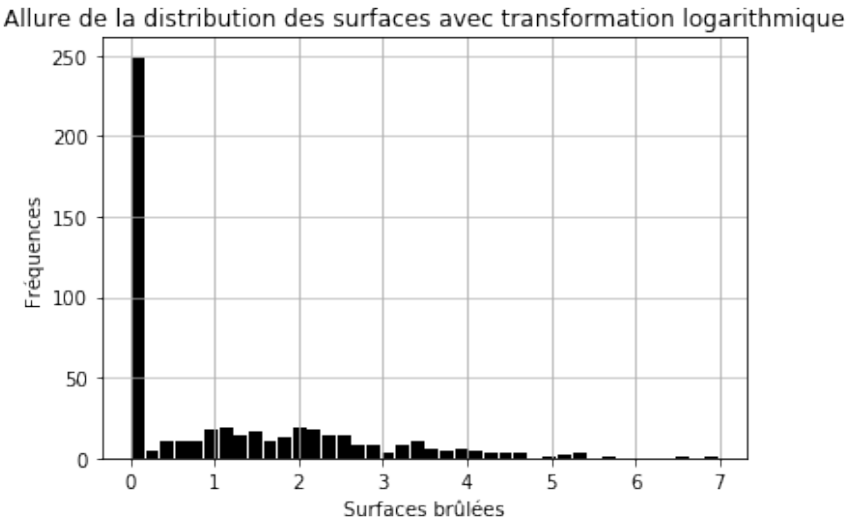
$$(\text{lundi}, \ldots, \text{dimanche}) \Longleftrightarrow [\![0, 6]\!]$$

$$(\text{lundi, ..., dimanche}) \iff [\![0, 6]\!]$$
$$(\text{janvier, ..., décembre}) \iff [\![0, 11]\!]$$

$$(\text{lundi, ..., dimanche}) \Longleftrightarrow [\![0, 6]\!]$$
$$(\text{janvier, ..., décembre}) \Longleftrightarrow [\![0, 11]\!]$$
$$X \longmapsto ln(1 + X)$$

$$(\text{lundi, ..., dimanche}) \Longleftrightarrow [\![0, 6]\!]$$
$$(\text{janvier, ..., décembre}) \Longleftrightarrow [\![0, 11]\!]$$
$$X \longmapsto ln(1 + X)$$





```python
# fabrication d'un second set pour un second modèle sur de petits incendies
# proportion d'échantillons de target < = 15 hectares (se donner une idée)
n = len(target)
s = 0
for i in range(n):
    if target_temp[i] <= 15:
        s += 1
# qui représentent +85% des échantillons (vu avec)
print(s/n)
```

```
0.8665377176015474
```

```
# fabrication d'un second set pour un second modèle sur de petits incendies
# proportion d'échantillons de target < = 15 hectares (se donner une idée)
n = len(target)
s = 0
for i in range(n):
    if target_temp[i] <= 15:
        s += 1
# qui représentent +85% des échantillons (vu avec)
print(s/n)
```

```
0.8665377176015474
```

```
# dans le module model_selection, train_test_split pour diviser les data
from sklearn.model_selection import train_test_split

# séparation en 2 jeux (un de train et un de test)
data_split = train_test_split(data, target, train_size=0.75, test_size = 0.25,
                              random_state = 70, shuffle=False)
data_train, data_test, target_train, target_test = data_split
```

1. moyenne = 0

1. moyenne = 0
2. variance = 1

1. moyenne = 0
2. variance = 1

```
from sklearn.preprocessing import StandardScaler

scalerS = StandardScaler()  # pour toutes les données
scalerS2 = StandardScaler()  # les données ayant target <= 15 ha

scalerS.fit(data_train)
scalerS2.fit(dat_tra)

data_train_tr = scalerS.transform(data_train)
data_test_tr = scalerS.transform(data_test)
dat_tra_tr = scalerS2.transform(dat_tra)
dat_tst_tr = scalerS2.transform(dat_tst)
```

RMSE

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

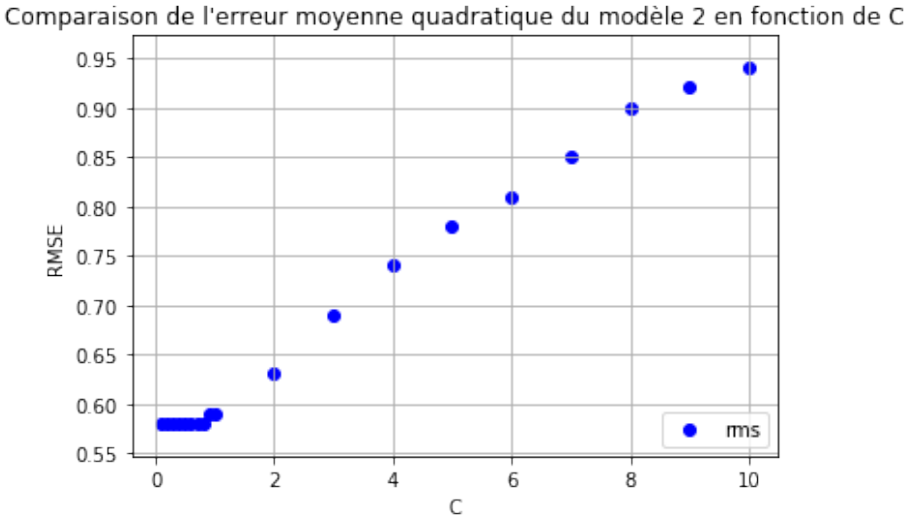
RMSE

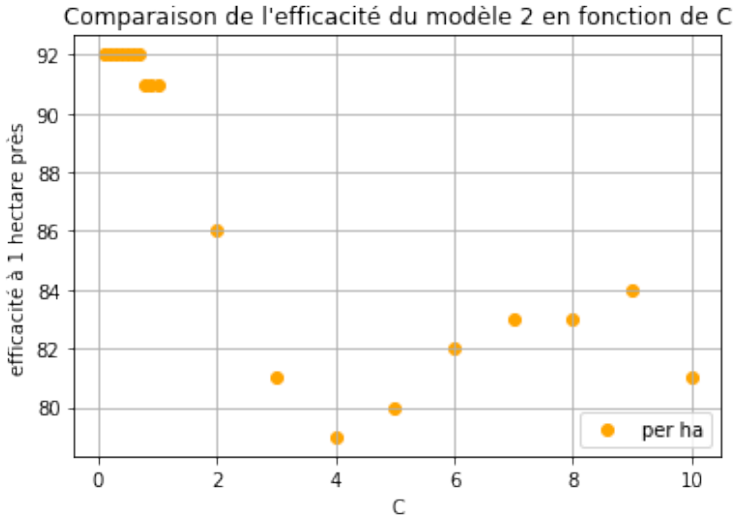
$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

```
def score(vrais_resultats, predictions, eps=10):
    # renvoie RMSE et % de points tels que abs(prédiction-résultat) <= eps
    ecart = np.abs(vrais_resultats - predictions).ravel()
    # ravel pour aplatir un éventuel tableau de plusieurs dimensions

    RMSE, s, n = 0, 0, len(ecart)
    for i in range(n):
        RMSE += ecart[i]*ecart[i]
        if ecart[i] <= eps:
            s += 1
    return np.sqrt(RMSE/n), 100*(round(100*s/n)/100)
```

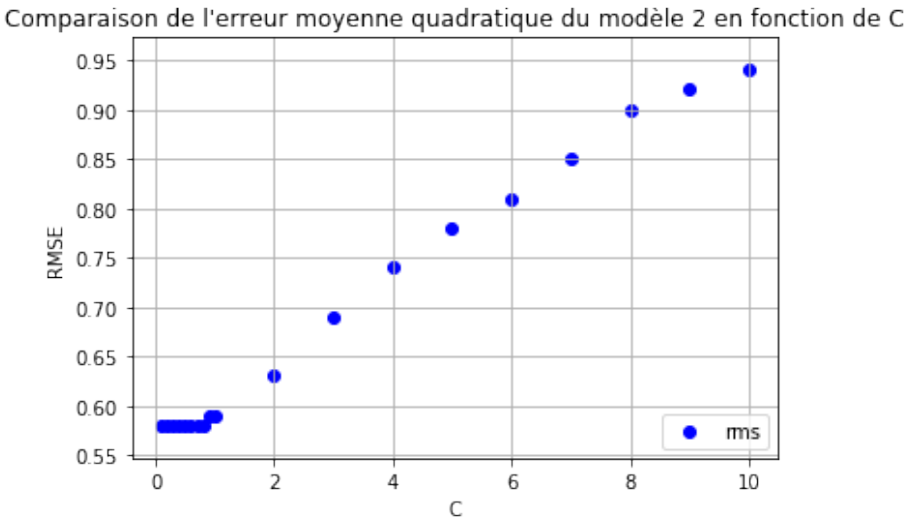
$C = 0.2$ (par sérendipité puis confirmé par une heuristique (de 0.1 à 10))

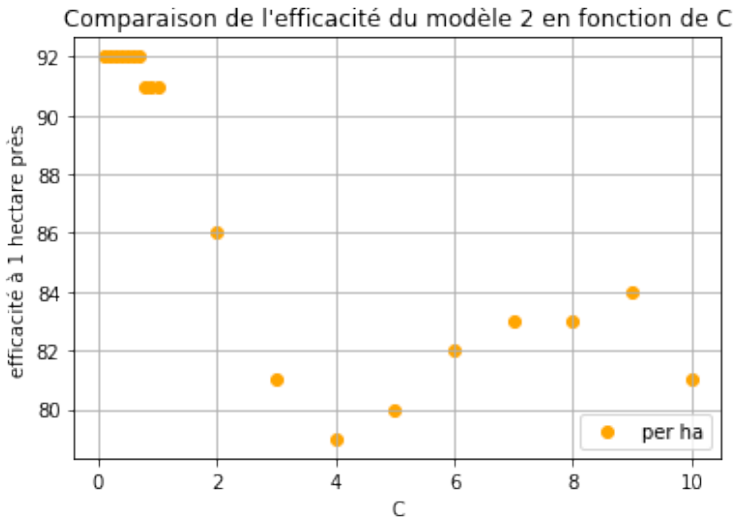
$C = 0.2$ (par sérendipité puis confirmé par une heuristique (de 0.1 à 10))





$C = 0.2$ (par sérendipité puis confirmé par une heuristique (de 0.1 à 10))





$$\varepsilon \sim \sqrt{\frac{ln(N)}{N}} \sim 10^{-1} \text{ (d'après le } \textit{théorème de Vapnìk}\text{)}$$

$$\varepsilon \sim \sqrt{\frac{ln(N)}{N}} \sim 10^{-1} \text{ (d'après le } \textit{théorème de Vapnìk}\text{)}$$

Theorem

(Approximation) Si la dimension VC (∼ complexité du modèle) d'un modèle est une petite fraction du nombre N de données, alors : erreur test $\leqslant$ *erreur train* $+ \sqrt{\frac{VC}{N}}$

$\varepsilon \sim \sqrt{\frac{ln(N)}{N}} \sim 10^{-1}$ (d'après le *théorème de Vapnìk*)

Theorem

(Approximation) Si la dimension VC ($\sim$ complexité du modèle) d'un modèle est une petite fraction du nombre N de données, alors : erreur test $\leqslant$ erreur train $+ \sqrt{\frac{VC}{N}}$

Estimation : $VC \leqslant ln(N)$

Paramètre gamma du noyau :

$$\gamma = \frac{1}{n_c}$$

Paramètre gamma du noyau :

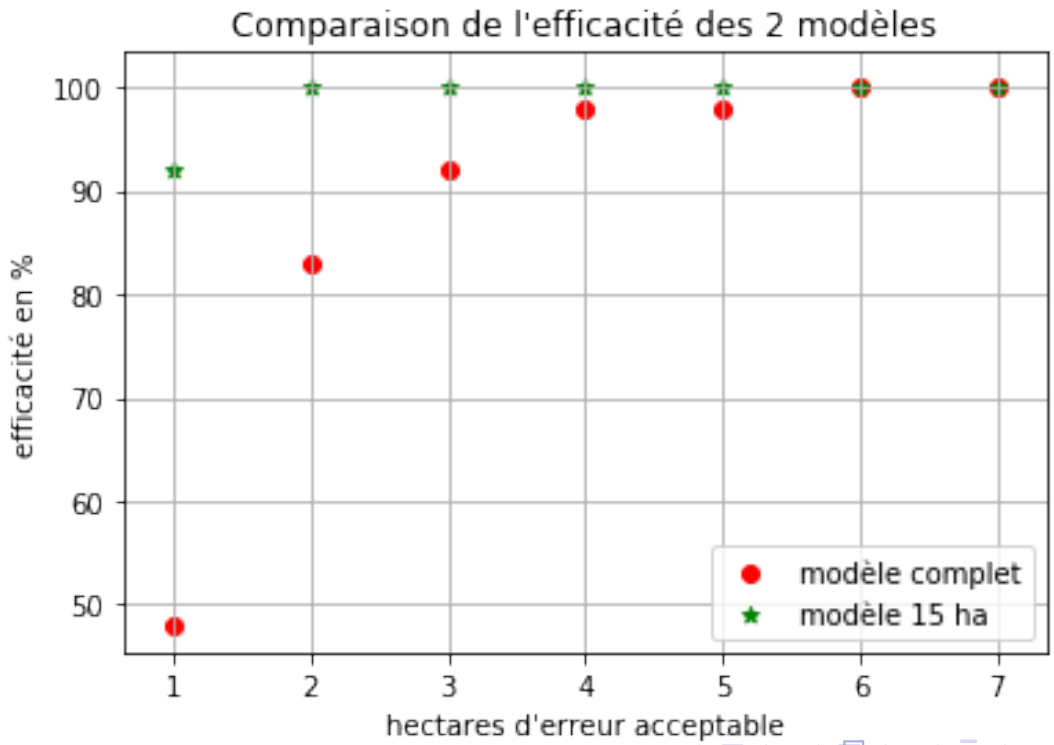
$$\gamma = \frac{1}{n_c}$$

```
### Partie initialisation de l'algorithme

from sklearn.svm import SVR
C= 0.2
eps = 1e-1

model = SVR(C=C, gamma='auto', kernel='rbf', epsilon= eps)
model.fit(data_train_tr, target_train)

model2 = SVR(C=C, gamma='auto', kernel='rbf', epsilon=eps)
model2.fit(dat_tra_tr, targ_tra)
```

```python
# prédictions model (encore sous forme logarithmique)
log_predictions = model.predict(data_test_tr)
predictions = [np.exp(X)-1 for X in log_predictions]

# prédictions model2 (encore sous forme logarithmique)
log_predictions2 = model2.predict(dat_tst_tr)
predictions2 = [np.exp(X)-1 for X in log_predictions2]
```

Résultats :

1. Modèle 1 : RMSE = 1.68421 (max err 5.58 ha)($< 2\%$ err rel)
2. Modèle 2 : RMSE = 0.57849 (max err 1.16 ha)($< 1\%$ err rel)



$$R^2 = 1 - \frac{u}{v}$$

u : RMSE sur les points extérieurs

v : RMSE sur tous les points

$$R^2 = 1 - \frac{u}{v}$$

u : RMSE sur les points extérieurs

v : RMSE sur tous les points

Premier modèle :

```
0.9992233792590851
```

$$R^2 = 1 - \frac{u}{v}$$

u : RMSE sur les points extérieurs

v : RMSE sur tous les points

Premier modèle :

```
0.9992233792590851
```

Second modèle :

```
0.9950255184086423
```

En conclusion

Annexes

Conditions nécessaires KKT :
sous hypothèses de continuité et de dérivabilité de J et F, u est un *extremum*

$$u \in K \ , \ J'(u) + \sum_{i=1}^{M} p_i F_i'(u) = 0$$

Annexes

Conditions nécessaires KKT :
sous hypothèses de continuité et de dérivabilité de J et F, u est un *extremum*

$$u \in K \ , \ J'(u) + \sum_{i=1}^{M} p_i F_i'(u) = 0$$

Suffisantes sous hypothèse supplémentaire de convexité de J et F.

Annexes

Si (u, p) point-selle de $\mathscr{L}$, alors :

$$(u, p) \in V \times \mathbb{R}_+^M \ , \ \langle p | F(u) \rangle = 0$$

et

$$J'(u) + \langle p | F'(u) \rangle = 0$$

Annexes

Gradient avec projection :

$$\forall \mu > 0 \ , \ \forall v \in K \ , \ \langle u - (u - \mu \, J'(u)) | v - u \rangle \geqslant 0$$

Annexes

Gradient avec projection :

$$\forall \mu > 0 \ , \ \forall v \in K \ , \ \langle u - (u - \mu \ J'(u)) | v - u \rangle \geqslant 0$$

$$u = P_K(u - \mu J'(u))$$

Annexes

Semi définie-positivité d'une fonction K (sur un espace mesurable/probabilisable) :

$$(x_1, x_2, \ldots, x_N), (\mu_1, \mu_2, \ldots, \mu_N) \in \mathbb{R} \ , \ \sum_{i,j}^{N} K(x_i, x_j)\mu_i\mu_j \geqslant 0$$

Annexes

Theorem

(de Vapnìk) pour p la probabilité d'obtenir un jeu d'entraînement non représentatif des données générales, N le nombre de données du jeu d'entraînement et VC la dimension VC du modèle entraîné, alors avec une probabilité $1 - p$ *l'erreur de prédiction du modèle sur les données de test ne diffère de celle sur les données d'entraînement que d'une marge*

$$\sqrt{\frac{VC(1 + log(\frac{N}{VC})) + log(\frac{1}{4p})}{N}}$$

```
### Partie extraction des données

# import bibliothèque traitement des fichiers tableur + ouverture du document
import xlrd
doc = xlrd.open_workbook('C:/Users/Yann/Desktop/prepa/20_21/tipe/forestfires.csv')

# extraction première (et seule) feuille du fichier + ses dimensions
feuille = doc.sheet_by_index(0)
lignes, colonnes = feuille.nrows, feuille.ncols

import numpy as np

# extraction noms des caractéristiques
feature_names = []
for i in range(colonnes-1): # -1 pour enlever la dernière colonne (résultats)
    # on prend pas les coordonnées (colonnes 0 et 1)
    if i >= 2:
        feature_names.append(feuille.cell_value(rowx = 0, colx = i))

# convertir les mois et les jours de chaîne en entiers
month = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']
day = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun']

def month_2_int(i):
    n = len(month)
    for k in range(n):
        if feuille.cell_value(rowx = i, colx = 2) == month[k]:
            return k

def day_2_int(i):
    n = len(day)
    for k in range(n):
        if feuille.cell_value(rowx = i, colx = 3) == day[k]:
            return k
```

```
# extraction données dans tableau numpy (scikit avec type Bunch)
data_list = []
for i in range(1, lignes):
    elem = []
    for j in range(colonnes-1):
        # omission volontaire des colonnes de coordonnées 0 et 1
        if j >= 4:
            elem.append(float(feuille.cell_value(rowx = i, colx = j)))
        elif j == 2: # mois transformés
            elem.append(month_2_int(i))
        elif j == 3: # jours transformés
            elem.append(day_2_int(i))
    data_list.append(elem)
    elem = []
data = np.array(data_list)

# mettre les targets dans un tableau numpy de 1 colonne target
target_temp = []
for i in range(1, lignes):
    target_temp.append(float(feuille.cell_value(rowx=i, colx= 12)))

target = np.array([np.log(1+X) for X in target_temp]) # transfo logarithmique
# rapprocher valeurs des abscisses (sans ça: imprécisions pour grandes valeurs)

fire= [data, feature_names, target]
```

```
# fabrication d'un second set pour un second modèle sur de petits incendies
# proportion d'échantillons de target < = 15 hectares (se donner une idée)
n = len(target)
s = 0
for i in range(n):
    if target_temp[i] <= 15:
        s += 1
# qui représentent +85% des échantillons (vu avec)
print(s/n)
```

```
0.8665377176015474
```

```
# dans le module model_selection, train_test_split pour diviser les data
from sklearn.model_selection import train_test_split

# séparation en 2 jeux (un de train et un de test)
data_split = train_test_split(data, target, train_size=0.75, test_size = 0.25,
                              random_state = 70, shuffle=False)
data_train, data_test, target_train, target_test = data_split
```

```
from sklearn.preprocessing import StandardScaler

scalerS = StandardScaler()  # pour toutes les données
scalerS2 = StandardScaler()  # les données ayant target <= 15 ha

scalerS.fit(data_train)
scalerS2.fit(dat_tra)

data_train_tr = scalerS.transform(data_train)
data_test_tr = scalerS.transform(data_test)
dat_tra_tr = scalerS2.transform(dat_tra)
dat_tst_tr = scalerS2.transform(dat_tst)
```

```
def score(vrais_resultats, predictions, eps=10):
    # renvoie RMSE et % de points tels que abs(prédiction-résultat) <= eps
    ecart = np.abs(vrais_resultats - predictions).ravel()
    # ravel pour aplatir un éventuel tableau de plusieurs dimensions

    RMSE, s, n = 0, 0, len(ecart)
    for i in range(n):
        RMSE += ecart[i]*ecart[i]
        if ecart[i] <= eps:
            s += 1
    return np.sqrt(RMSE/n), 100*(round(100*s/n)/100)
```

```
# méthode de recherche heuristique de C (ici pour le modèle 2)

Y3, Y4 = [], []
CC = [i/10 for i in range(1,10)]+[i for i in range(1,11)]

for C in CC:
    model2 = SVR(C=C, gamma='auto', kernel='rbf', epsilon=eps)
    model2.fit(dat_tra_tr, targ_tra)
    log_predictions2 = model2.predict(dat_tst_tr)
    predictions2 = [np.exp(X)-1 for X in log_predictions2]
    rms, per = score(targ_tst, predictions2, eps=1)

    Y3.append(round(100*rms)/100), Y4.append(per)


import matplotlib.pyplot as plt

rms_graph = plt.scatter(CC,Y3, color='blue', marker='o')
plt.grid()
plt.xlabel("C")
plt.ylabel("RMSE")
plt.title("Comparaison de l'erreur moyenne quadratique du modèle 2 en fonction de C")
plt.legend([rms_graph], ['rms'], loc='lower right')
plt.show()

per_graph = plt.scatter(CC,Y4, color='orange')
plt.grid()
plt.xlabel("C")
plt.ylabel("efficacité à 1 hectare près")
plt.title("Comparaison de l'efficacité du modèle 2 en fonction de C")
plt.legend([per_graph], ['per ha'], loc='lower right')
plt.show()
```

```
### Partie initialisation de l'algorithme

from sklearn.svm import SVR
C= 0.2
eps = 1e-1

model = SVR(C=C, gamma='auto', kernel='rbf', epsilon= eps)
model.fit(data_train_tr, target_train)

model2 = SVR(C=C, gamma='auto', kernel='rbf', epsilon=eps)
model2.fit(dat_tra_tr, targ_tra)
```

```
# prédictions model (encore sous forme logarithmique)
log_predictions = model.predict(data_test_tr)
predictions = [np.exp(X)-1 for X in log_predictions]

# prédictions model2 (encore sous forme logarithmique)
log_predictions2 = model2.predict(dat_tst_tr)
predictions2 = [np.exp(X)-1 for X in log_predictions2]
```

```
import random as rd
# ce qu'on peut retrouver à propos du modèle (ici model) après l'apprentissage
# attributs appelables: support_ , support_vectors_ , dual_coef_ , intercept_
n_sv = len(model.support_)
k = rd.randint(0, n_sv -1)
print('\n')
print('- Le {}ème vecteur de support est le vecteur numéro {} du jeu de départ.'
      .format(k, model.support_[k]))
sv_k = model.support_vectors_[k]
print("- Les {} coordonnées de ce vecteur dans l'espace de départ sont \n {} "
      .format(len(sv_k), sv_k))
print('- Son coefficient dual est {}'.format(model.dual_coef_[0,k]))
```

```
- Le 230ème vecteur de support est le vecteur numéro 248 du jeu de départ.
- Les 10 coordonnées de ce vecteur dans l'espace de départ sont
 [ 0.17497563 -0.59350883  0.46212341  1.19072374  0.48809286  1.03737799
  2.02855139 -0.96914545 -0.68367344 -0.06864065]
- Son coefficient dual est -0.2
```

```
X, Y1, Y2 = [], [], []
for i in range(1,8):
    Y1.append(score(target_test, predictions, eps=i)[1])
    Y2.append(score(targ_tst, predictions2, eps=i)[1])
    X.append(i)
```

```
mod1 = plt.scatter(X,Y1, color='red', marker='o')
mod2 = plt.scatter(X,Y2, color='green', marker='*')
plt.grid()
plt.xlabel("hectares d'erreur acceptable")
plt.ylabel("efficacité en %")
plt.title("Comparaison de l'efficacité des 2 modèles")
plt.legend([mod1,mod2], ['modèle complet','modèle 15 ha'], loc='lower right')
```

```python
### Partie visualisation de la précision des prédictions

def visu_diff_abs(pred, target):
    # visualiser la différence absolue entre deux tableaux
    tst = len(target)
    return [round(abs(target[k]-pred[k])*100)/100 for k in range(tst)]

def visu_diff_rel(pred, target):
    # visualiser la différence relative entre deux tableaux
    tst = len(target)
    L = []
    for i in range(tst):
        if target[i] == 0:
            L.append((-1)*round(abs(target[i]-pred[i])*100)/100)
            continue
        L.append(round(abs((target[i]-pred[i])/target[i])*10000)/100)
    return L

diff_abs = visu_diff_abs(predictions, target_test)
diff_abs2 = visu_diff_abs(predictions2, targ_tst)
diff_rel = visu_diff_rel(predictions, target_test)
diff_rel2 = visu_diff_rel(predictions2, targ_tst)
```

```python
print(sorted(diff_abs))
```

```
[0.02, 0.06, 0.07, 0.11, 0.13, 0.13, 0.14, 0.16, 0.16, 0.17, 0.25, 0.27, 0.28, 0.29, 0.31, 0.31, 0.32, 0.32, 0.32, 0.32, 0.34,
0.37, 0.38, 0.4, 0.43, 0.43, 0.44, 0.45, 0.45, 0.46, 0.47, 0.48, 0.51, 0.53, 0.55, 0.56, 0.57, 0.59, 0.62, 0.62, 0.67, 0.68, 0.
68, 0.69, 0.69, 0.69, 0.7, 0.71, 0.72, 0.77, 0.77, 0.78, 0.8, 0.81, 0.81, 0.81, 0.82, 0.84, 0.86, 0.88, 0.94, 0.95, 0.96, 1.02,
1.02, 1.04, 1.04, 1.06, 1.06, 1.06, 1.12, 1.13, 1.14, 1.15, 1.19, 1.2, 1.24, 1.3, 1.3, 1.31, 1.33, 1.33, 1.33, 1.37, 1.38, 1.3
9, 1.42, 1.46, 1.52, 1.55, 1.58, 1.61, 1.67, 1.69, 1.72, 1.75, 1.75, 1.76, 1.81, 1.87, 1.89, 1.89, 1.89, 1.9, 1.9, 1.91, 1.92,
1.93, 2.0, 2.0, 2.11, 2.12, 2.18, 2.4, 2.71, 2.79, 2.87, 2.89, 2.91, 3.0, 3.31, 3.38, 3.59, 3.8, 3.82, 3.92, 3.98, 4.05, 5.29,
5.58]
```

```
print(sorted(diff_abs2))
```

```
[0.0, 0.08, 0.08, 0.09, 0.09, 0.09, 0.1, 0.1, 0.1, 0.11, 0.11, 0.11, 0.11, 0.11, 0.11, 0.11, 0.12, 0.12, 0.13, 0.13, 0.13, 0.1
3, 0.13, 0.14, 0.14, 0.14, 0.15, 0.15, 0.16, 0.17, 0.18, 0.18, 0.18, 0.18, 0.19, 0.2, 0.2, 0.2, 0.2, 0.2, 0.23, 0.24, 0.24, 0.2
4, 0.26, 0.27, 0.27, 0.27, 0.28, 0.28, 0.28, 0.28, 0.29, 0.3, 0.31, 0.33, 0.35, 0.36, 0.39, 0.41, 0.42, 0.43, 0.43, 0.46, 0.48,
0.5, 0.52, 0.54, 0.54, 0.54, 0.58, 0.58, 0.59, 0.61, 0.67, 0.68, 0.75, 0.76, 0.78, 0.78, 0.78, 0.78, 0.78, 0.78, 0.79, 0.79, 0.
82, 0.82, 0.83, 0.83, 0.85, 0.85, 0.86, 0.89, 0.89, 0.89, 0.92, 0.95, 0.95, 0.96, 0.96, 0.98, 0.99, 1.0, 1.0, 1.04, 1.05, 1.05,
1.1, 1.11, 1.16, 1.16]
```

```
print(sorted(diff_rel))
```

```
[-1.89, -1.89, -1.81, -1.76, -1.75, -1.75, -1.69, -1.61, -1.58, -1.46, -1.42, -1.39, -1.38, -1.33, -1.33, -1.31, -1.3, -1.3, -
1.14, -1.13, -1.12, -1.06, -1.06, -1.02, -1.02, -0.86, -0.82, -0.81, -0.78, -0.77, -0.72, -0.7, -0.68, -0.67, -0.57, -0.55, -0.
51, -0.45, -0.43, -0.38, -0.34, -0.32, -0.32, -0.32, -0.32, -0.29, -0.27, -0.25, -0.17, -0.16, -0.16, -0.13, -0.11, -0.07, -0.0
6, 0.03, 0.09, 0.11, 0.21, 0.22, 0.25, 0.25, 0.25, 0.26, 0.33, 0.34, 0.37, 0.37, 0.39, 0.4, 0.43, 0.45, 0.48, 0.5, 0.51, 0.52,
0.52, 0.53, 0.55, 0.57, 0.59, 0.6, 0.6, 0.6, 0.61, 0.62, 0.64, 0.65, 0.66, 0.66, 0.69, 0.7, 0.73, 0.73, 0.75, 0.75, 0.76, 0.76,
0.76, 0.79, 0.79, 0.8, 0.8, 0.82, 0.83, 0.83, 0.84, 0.84, 0.86, 0.87, 0.87, 0.87, 0.87, 0.88, 0.88, 0.91, 0.91, 0.92, 0.93, 0.9
3, 0.94, 0.95, 0.95, 0.96, 0.96, 0.98, 0.98, 1.59, 1.68, 1.9]
```

```
print(sorted(diff_rel2))
```

```
[-0.54, -0.46, -0.43, -0.39, -0.35, -0.31, -0.3, -0.29, -0.28, -0.28, -0.28, -0.27, -0.27, -0.27, -0.24, -0.24, -0.24, -0.23, -
0.2, -0.2, -0.2, -0.2, -0.2, -0.19, -0.18, -0.18, -0.18, -0.18, -0.17, -0.15, -0.14, -0.14, -0.14, -0.13, -0.13, -0.13, -0.13,
-0.13, -0.12, -0.12, -0.11, -0.11, -0.11, -0.11, -0.11, -0.11, -0.11, -0.1, -0.1, -0.1, -0.09, -0.09, -0.09, -0.08, -0.08, 0.0
1, 0.38, 0.43, 0.45, 0.48, 0.52, 0.54, 0.59, 0.59, 0.6, 0.7, 0.7, 0.7, 0.7, 0.71, 0.72, 0.73, 0.73, 0.74, 0.75, 0.75, 0.75, 0.7
5, 0.76, 0.76, 0.77, 0.77, 0.77, 0.78, 0.81, 0.82, 0.84, 0.84, 0.85, 0.85, 0.85, 0.86, 0.86, 0.87, 0.87, 0.87, 0.87, 0.87, 0.8
7, 0.88, 0.88, 0.9, 0.9, 0.9, 0.9, 0.91, 0.91, 0.91, 0.91, 0.91, 0.92, 0.93]
```

Pour le premier modèle :

```
# calcul R^2 premier modèle (sur les données d'entraînement)

p = [np.exp(X)-1 for X in model.predict(data_train_tr)]
n = len(model.support_) # nb de SV

u, v = 0, 0
for i in range(n):
    v += (target_train[i]-p[i])**2
    if not i in model.support_:
        u += (target_train[i]-p[i])**2

print(1-u/v)
```

```
0.9992233792590851
```

Pour le second modèle :

```
0.9950255184086423
```