Différentes implémentations du SVM

(SVC, nuSvc, linéarSVC, SVR, ,nuSvr, linéarSVR)

Présenté par:

Sow *Aoua*

Sanogo *Ousmane*

Diallo *Aboubacar*

Bah *Sidi*

Sommaire:

- Hypothèse
- objectif
- Importation et Exploration des données
- Tracé Des Graphiques

Hypothèses:

Les outils requis sont:

- SDK Python 3
- Un environnement de développement Python. Jupyter notebook (application web utilisée pour programmer en python) fera bien l'affaire
- les librairies requises :***matplotlib, numpy, pandas, sklearn***
- Jeu de Donnés : ***IRIS***

Objectifs:

- Observer la différence entre les différentes variantes du SVM dans un cas de régression et de classification;
- Jouer sur les paramètres de l'algorithme afin de voir leurs effets (Tuning);

Améliorer les performances avec l'algorithme Réglage (Algorithm Tuning)

Les modèles d'apprentissage automatique sont paramétrés afin que leur comportement puisse être ajusté pour un problème. Les modèles peuvent avoir de nombreux paramètres et trouver la meilleure combinaison de paramètres peut être traité comme un problème de recherche.

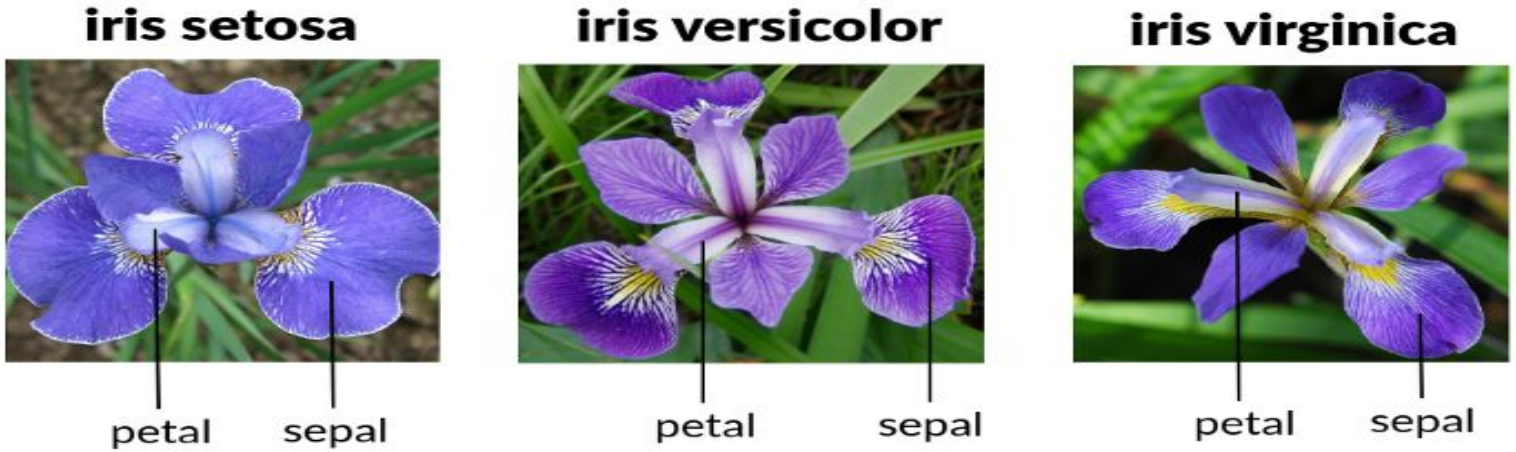
Améliorer les performances avec l'algorithme Réglage (Algorithm Tuning)

Le réglage de l'algorithme est une étape finale dans le processus d'apprentissage automatique appliqué avant de finaliser et valider notre modèle

Elle est parfois appelée
“***Optimisation d'hyper paramètres”***
où les paramètres de l'algorithme
sont appelés hyperparamètres.

Jeu de données

Dans ce cas pratique, on utilisera le célèbre jeu de données **IRIS.** Ce dernier est une base de données regroupant les caractéristiques de trois espèces de fleurs d'Iris, à savoir ***Setosa***, ***Versicolour*** et ***Virginica*** représentés dans le dataset dans la série target par 0, 1 et 2. Chaque ligne de ce jeu de données est une observation des caractéristiques d'une fleur d'Iris. Ce *dataset* décrit les espèces d'Iris par quatre propriétés : longueur et largeur de sépales ainsi que longueur et largeur de pétales. La base de données comporte 150 observations (50 observations par espèce)



Jeu de données

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Importation et Exploration de données

Importations des Bibliothèques:

Premièrement, nous importons les bibliothèques numpy, pyplot et sklearn.

Scikit-Learn vient avec un ensemble de jeu de données prêt à l'emploi pour des fins d'expérimentation. Ces dataset sont regroupés dans le package sklearn.datasets.

On charge le package datasets pour retrouver le jeu de données IRIS.

```
1 #import des librairies l'environnement
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import pandas as pd
5 from sklearn import svm, datasets
6 from sklearn.inspection import DecisionBoundaryDisplay
```

Importation et Exploration de données

Importations du jeu de données:

Pour charger le jeu de données Iris, on utilise la méthode **load_iris()** du package *datasets et ensuite créer un dataframe avec*

```
14 iris = datasets.load_iris()
15
16 # Create a dataframe
17 df = pd.DataFrame(iris.data, columns = iris.feature_names)
18 df['target'] = iris.target
```

Importation et Exploration de données

Pour simplifier le travail, on n'utilisera que les deux premières features à savoir : Sepal_length *et* Sepal_width.

Également, le jeu IRIS se compose de trois classes, les étiquettes peuvent donc appartenir à l'ensemble {0, 1, 2}. Il s'agit donc d'une classification Multi-classes.

Stockons également "targe" dans une séries Y

```
20 X = np.array(df.loc[:, ["sepal length (cm)","sepal width (cm)"] ])
21 y = iris.target
```

Importation et Exploration de données

Visualisation des données:

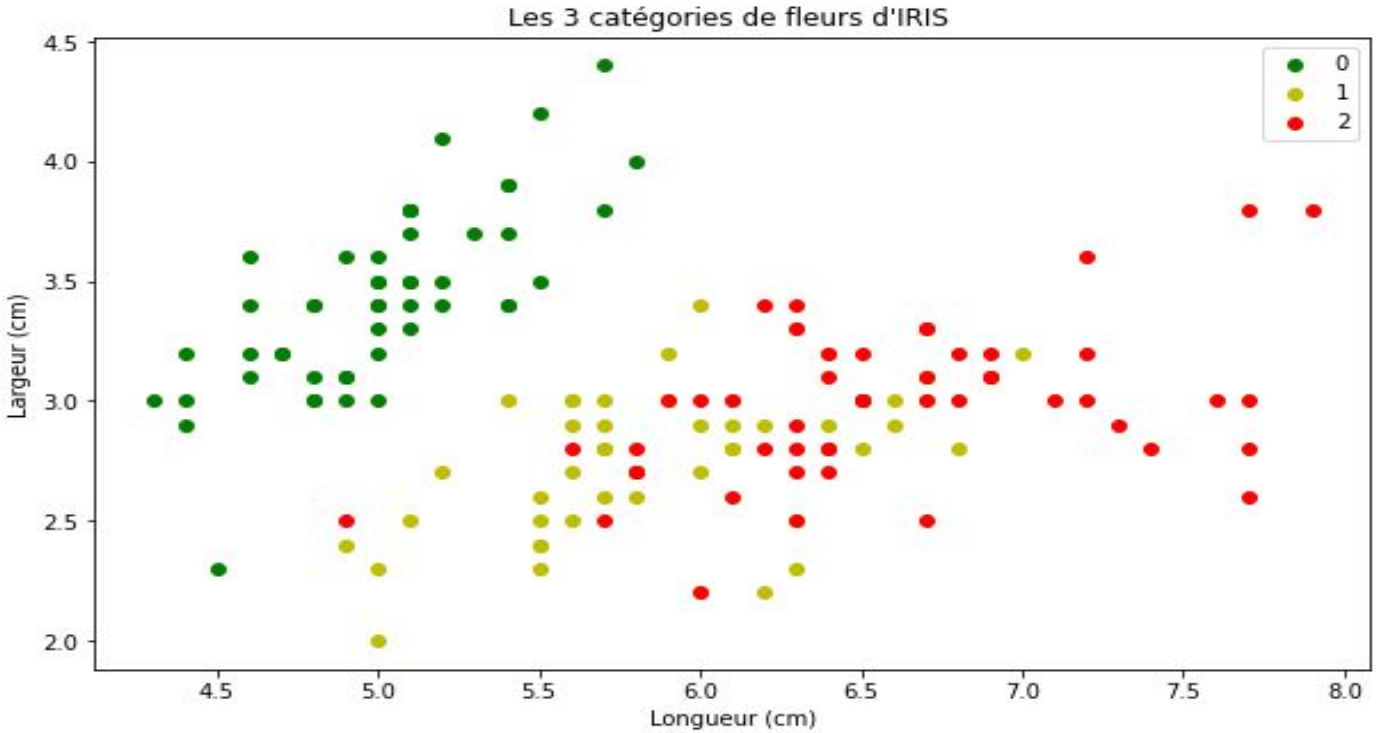
Voyons comment sont disposés nos données dans un plan

Afin de mieux comprendre notre jeu de données, il est judicieux de le visualiser.

```
#visualisation des données
plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='g', label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='y', label='1')
plt.scatter(X[y == 2][:, 0], X[y == 2][:, 1], color='r', label='2')
plt.xlabel("Longueur (cm)")
plt.ylabel("Largeur (cm)")
plt.title("Les 3 catégories de fleurs d'IRIS ")
plt.legend();
```

Importation et Exploration de données

Visualisation des données:



Importation et Exploration de données

Interprétation des données:

Les données sont disposés de telle sorte que la 1ère classe de données et l'ensemble de la 2ème et 3ème sont linéairement séparable, alors que les 2 derniers ne le sont pas.

Donc les classes 2 et 3 sont plus similaires comparé à la 1ère (en considérant que leur taille en longueur et largeur)

Importation et Exploration de données

Spécification des hyper-paramètre c et gamma

C:

C est le paramètre de pénalité, qui représente une mauvaise classification ou un terme d'erreur. Le terme de mauvaise classification ou d'erreur indique à l'optimisation SVM le degré d'erreur supportable. C'est ainsi que vous pouvez contrôler le compromis entre la limite de décision et le terme de mauvaise classification

NB: Lorsque C est élevé, il classera correctement tous les points de données, il y a également une chance de surajustement

Importation et Exportation de données

Spécification des hyper-paramètre c et gamma

gamma:

Il définit dans quelle mesure influence le calcul de la ligne de séparation plausible

Importation et Exploration de données

La SVM

Ici, nous ferons le tuning sur 4 variantesdu SVM :

a) Pour la Classification:

   SVC(kernel=linéaire), linearSVC, SVC(kernel=rbf) et SVC(kernel=poly)

b) De même pour la régression

   SVR(kernel=linéaire), linearSVR, SVR(kernel=rbf) et SVR(kernel=poly)

Pour chaque implementation, on fixera le paramètre gamma pour varier c et inversement (dans la pratique)

Importation et Exploration de données

Pour des fins de structurations, on organisera nos 4 différences instance du SVM (implémentation) dans une structure “n-uplet” à part; de même pour le titre de chaque figure

Et enfin entraîner notre modèle pour chaque instance du SVM

```
45 models = (
46     svm.SVC(kernel="linear", C=C),
47     svm.LinearSVC(C=C, max_iter=10000),
48     svm.SVC(kernel="rbf", gamma=0.7, C=C),
49     svm.SVC(kernel="poly", degree=3, gamma="auto", C=C),
50 )
51
52 # title for the plots
53 titles = (
54     "SVC with linear kernel",
55     "LinearSVC (linear kernel)",
56     "SVC with RBF kernel",
57     "SVC with polynomial (degree 3) kernel",
58 )
59 models = (clf.fit(X, y) for clf in models)
```

Tracé des graphiques en changeant à chaque fois un paramètre

La fonction ci-dessous nous permet d'afficher les graphes SVM de chaque cas:

```
X0, X1 = X[:, 0], X[:, 1]

for clf, title, ax in zip(models, titles, sub.flatten()):
    disp = DecisionBoundaryDisplay.from_estimator(
        clf,
        X,
        response_method="predict",
        cmap=plt.cm.coolwarm,
        alpha=0.8,
        ax=ax,
        xlabel=iris.feature_names[0],
        ylabel=iris.feature_names[1],
    )
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)

plt.show()
```

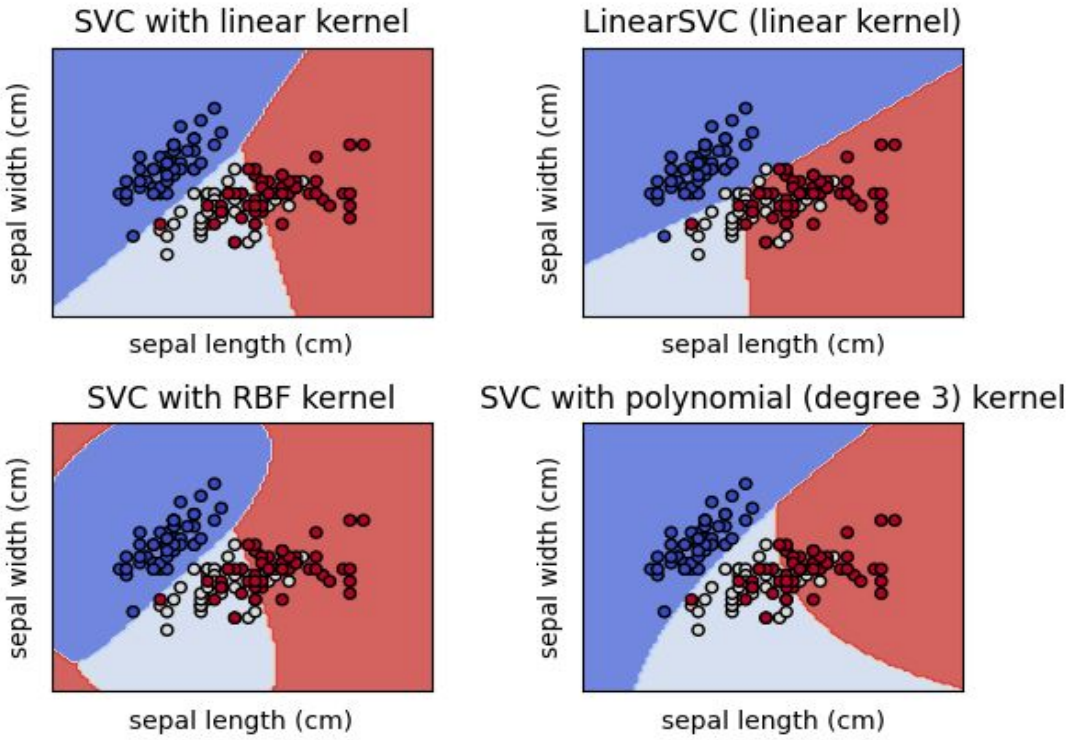
Cas de la Classification

Tracé des graphiques en changeant à chaque fois un paramètre

Fixons gamma à 0.7 et varions c

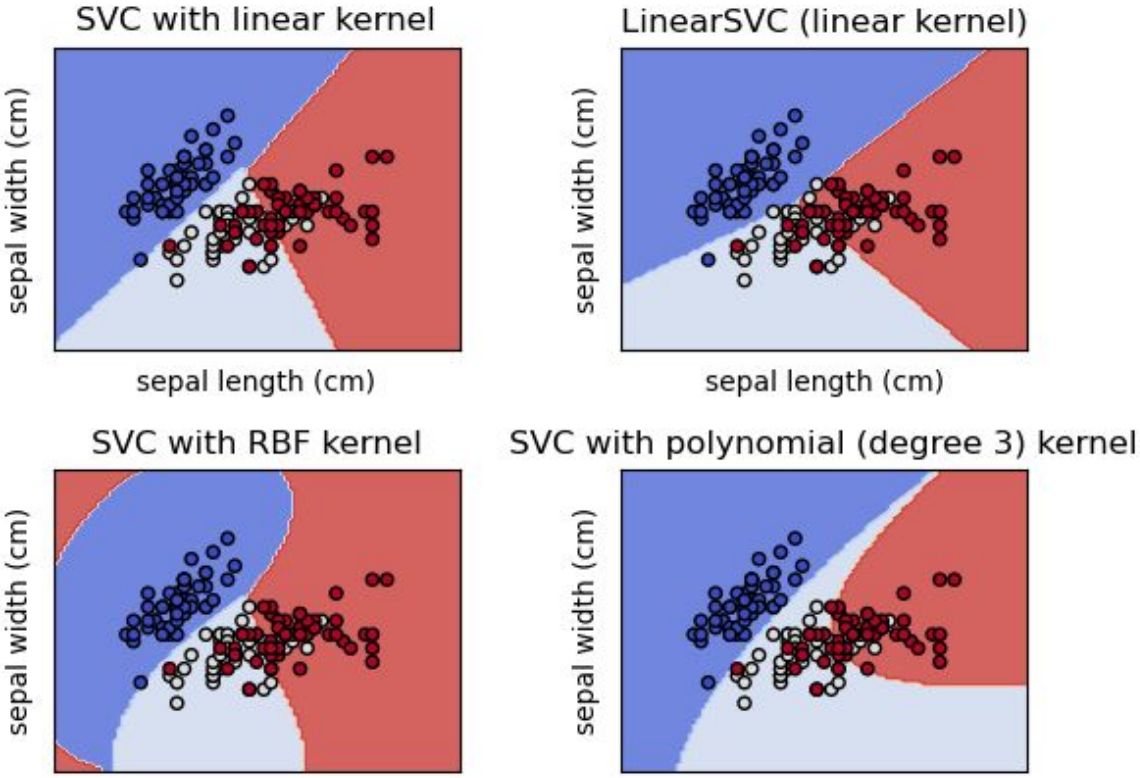
<u>Pour un 1er round de tuning</u>:

c=0.1



Cas de la Classification

Tracé des graphiques en changeant à chaque fois un paramètre



Pour le 2er round :

c=1

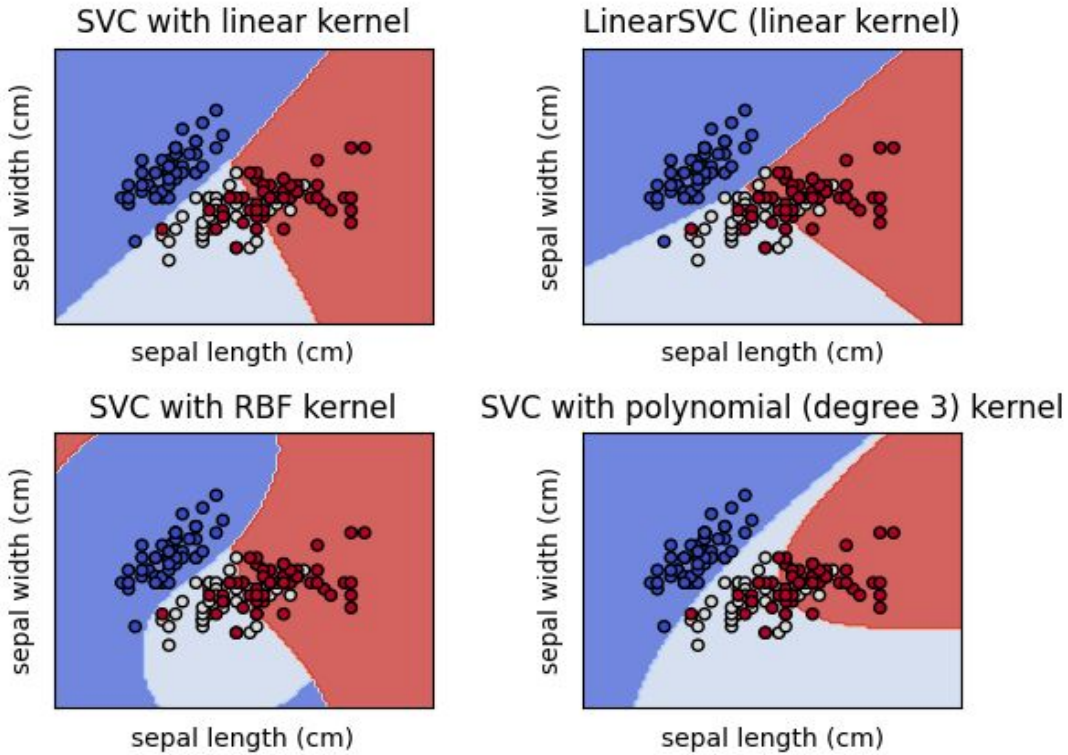
Cas de la Classification

Tracé des graphiques en changeant à chaque fois un paramètre

Fixons gamma à 0.7 et varions c

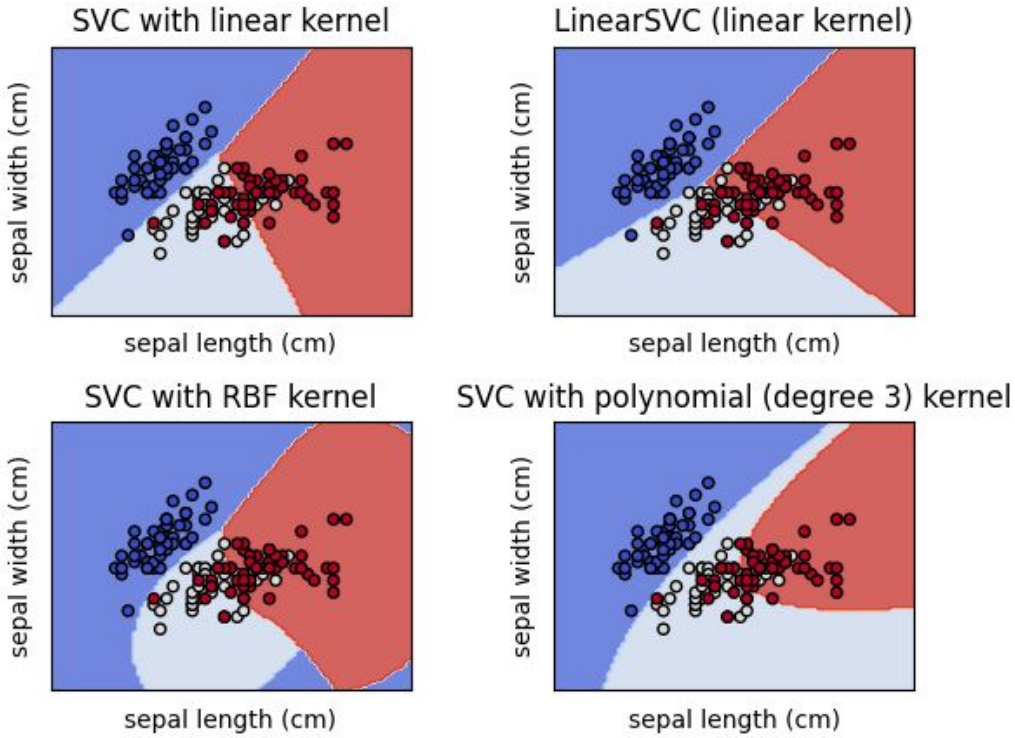
<u>Pour le 3ème round de tuning</u>:

c=3



<u>Cas de la Classification</u>

Tracé des graphiques en changeant à chaque fois un paramètre



Pour le 4ème round:

c=10

Cas de la Classification

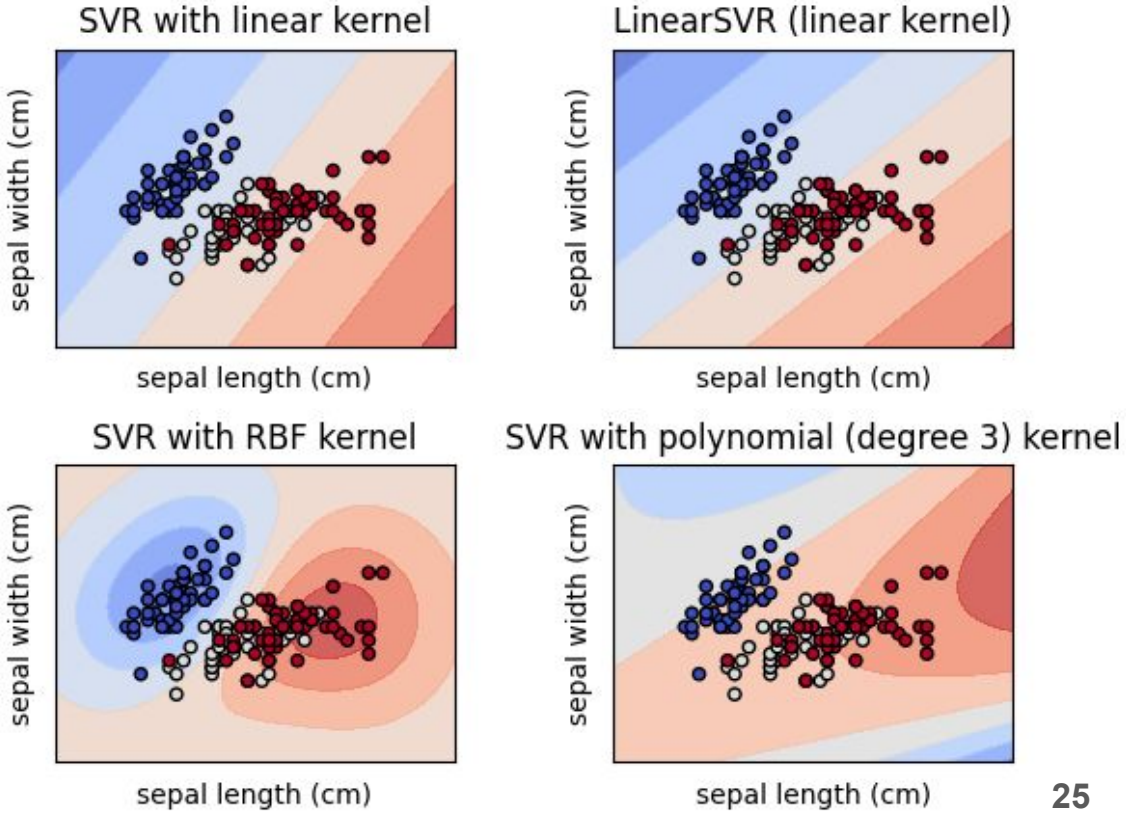
Cas de la Régression

Tracé des graphiques en changeant à chaque fois un paramètre

Fixons gamma à 0.7 et varions c

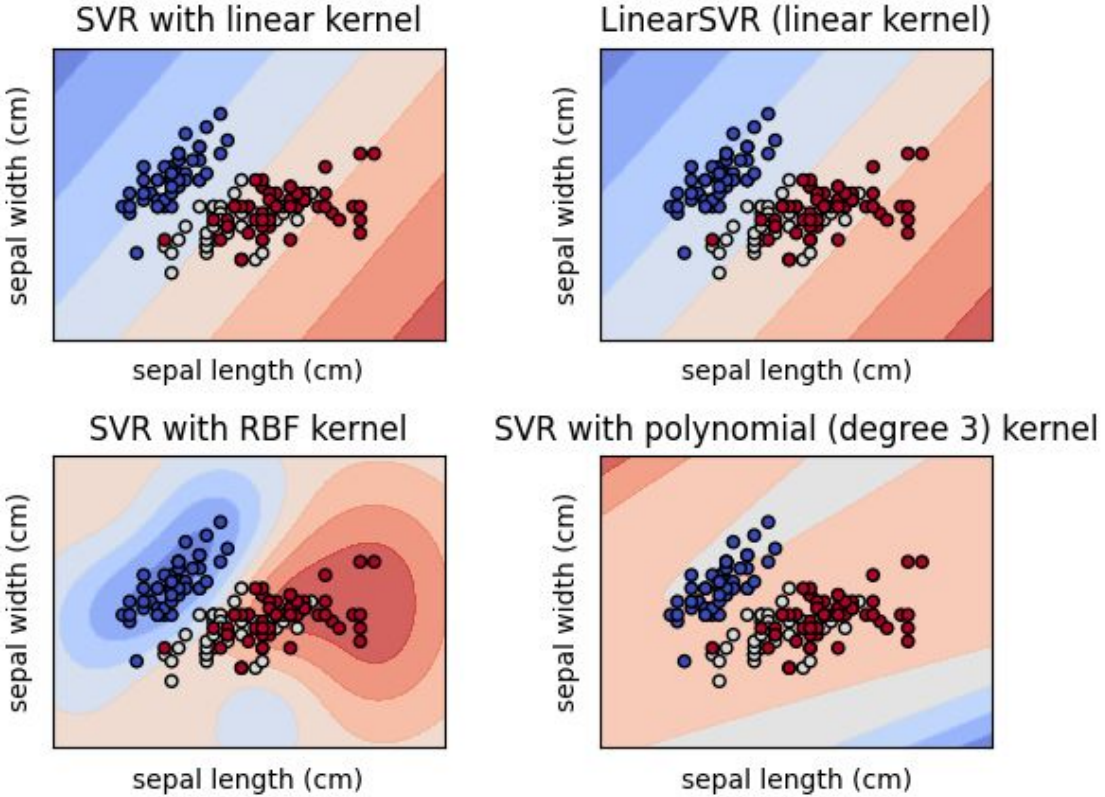
Pour un 1er round de tuning:

c=0.1



Cas de la Régression

Tracé des graphiques en changeant à chaque fois un paramètre



Pour le 2er round :

c=1

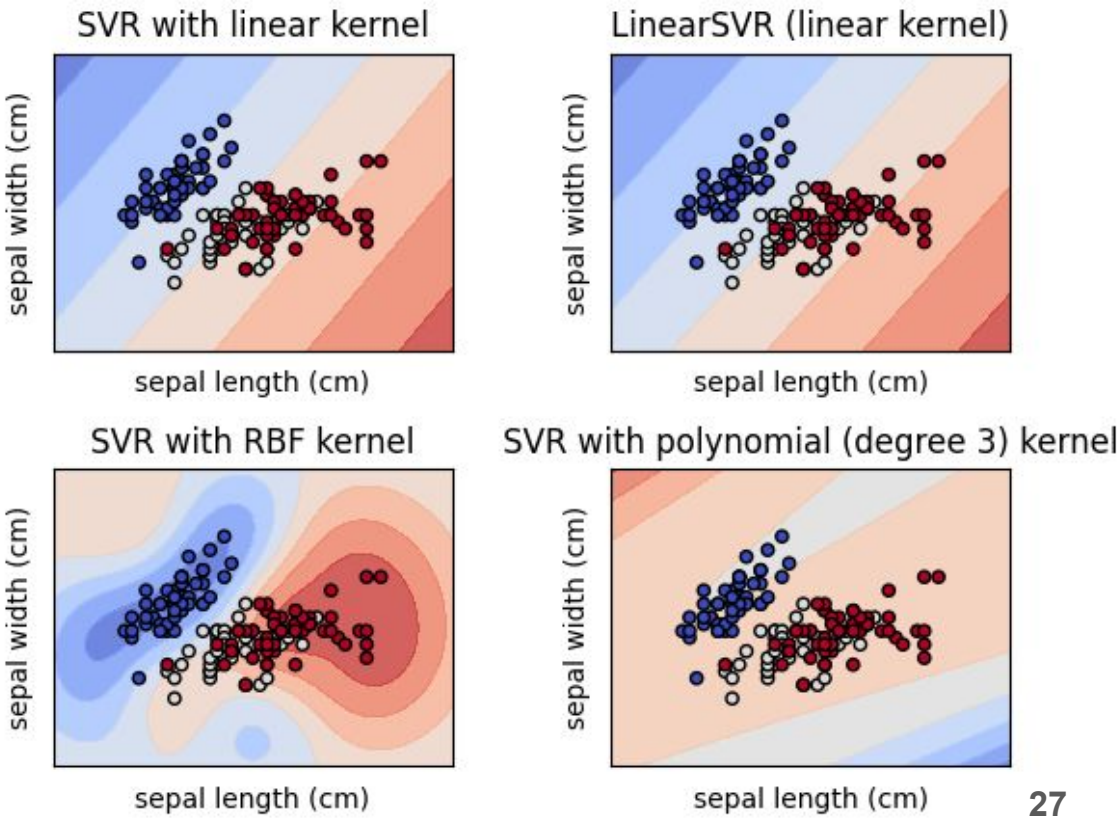
Cas de la Régression

Tracé des graphiques en changeant à chaque fois un paramètre

Fixons gamma à 0.7 et varions c

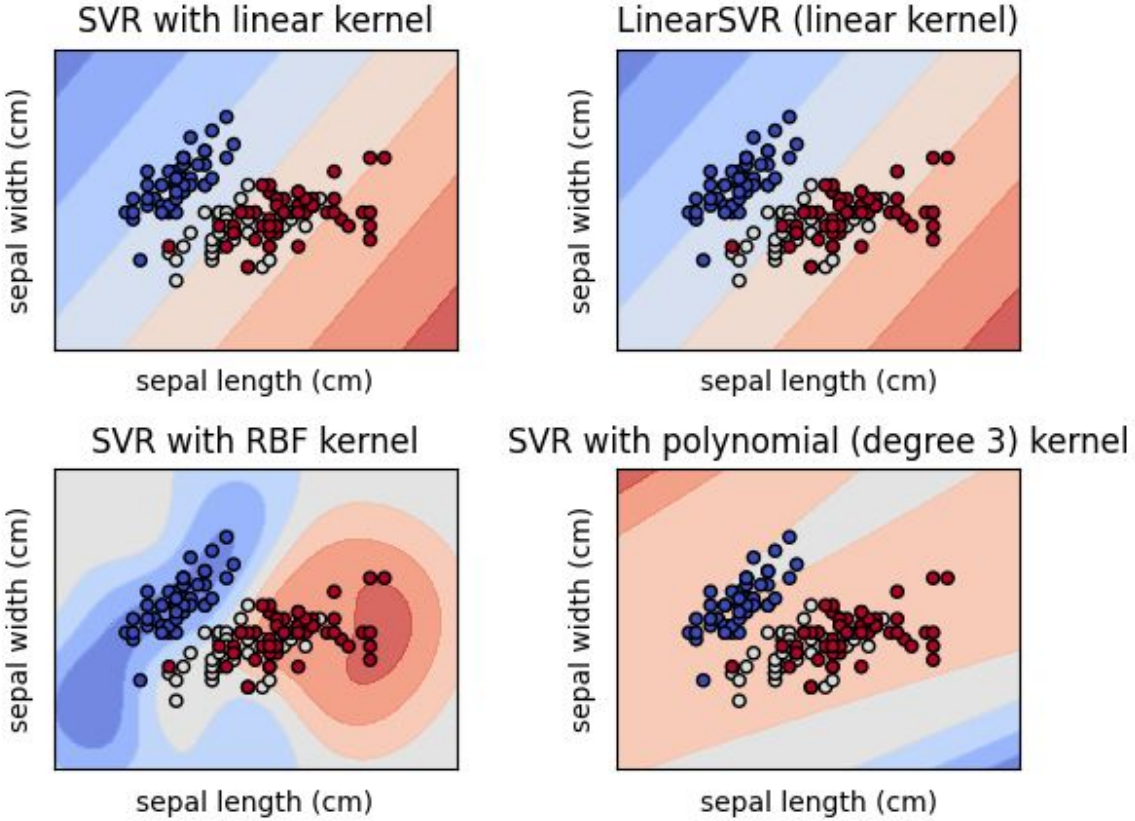
<u>Pour le 3ème round de tuning</u>:

c=3



<u>Cas de la Régression</u>

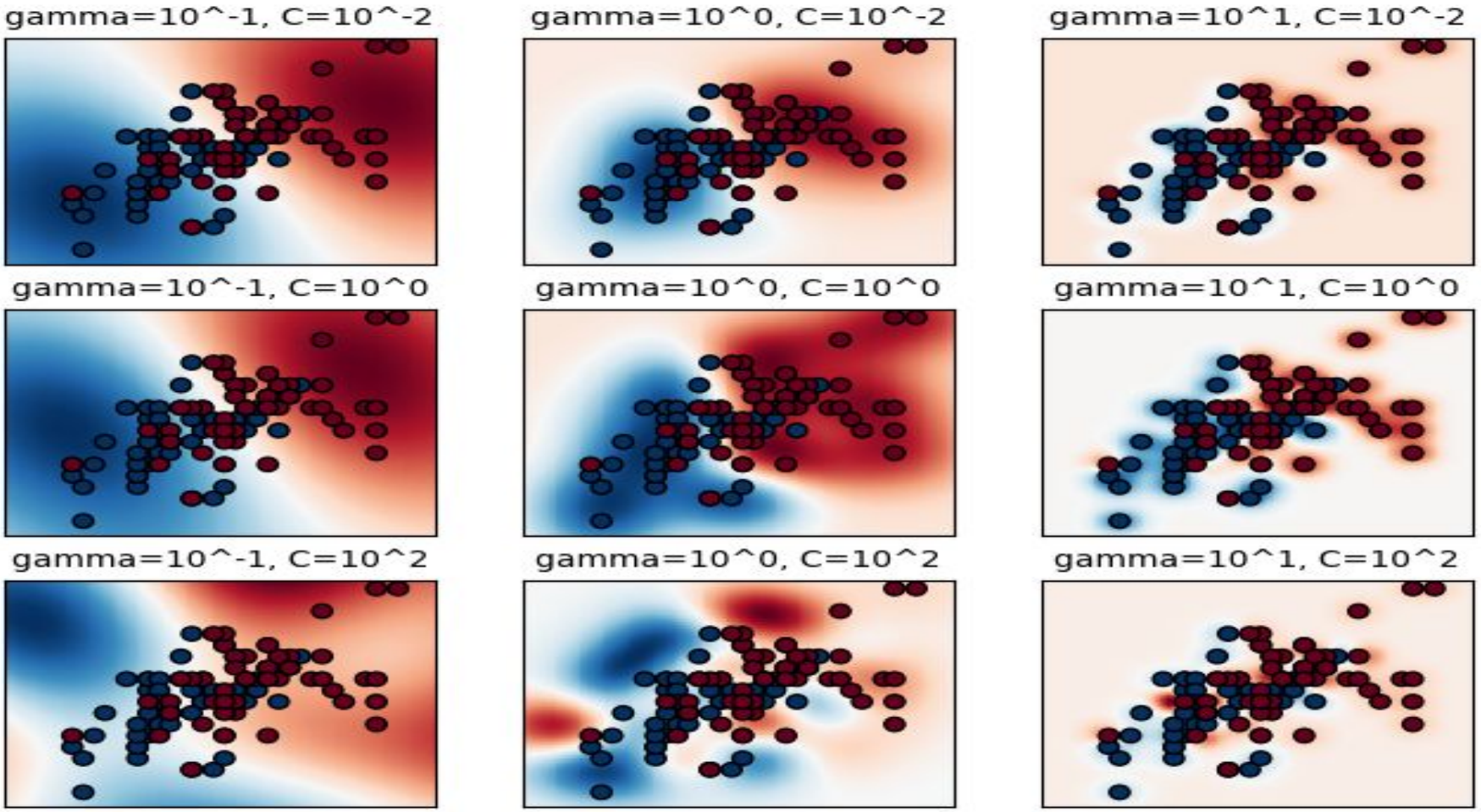
Tracé des graphiques en changeant à chaque fois un paramètre



Pour le 4ème round:

c=10

Cas de la Régression


gamma=10^-1, C=10^-2
gamma=10^0, C=10^-2
gamma=10^1, C=10^-2
gamma=10^-1, C=10^0
gamma=10^0, C=10^0
gamma=10^1, C=10^0
gamma=10^-1, C=10^2
gamma=10^0, C=10^2
gamma=10^1, C=10^2

MERCI!!!