TP : Classification SVM RBF sur le jeu de données Iris

Dans ce travail pratique, nous allons mettre en place un modèle de **SVM (Support Vector Machine)** avec un **noyau RBF (Radial Basis Function)** afin de classer des fleurs d'iris. Nous n'utiliserons que les **deux premières caractéristiques** du jeu de données (longueur et largeur des sépales) pour simplifier la visualisation.

Nous allons procéder comme suit :

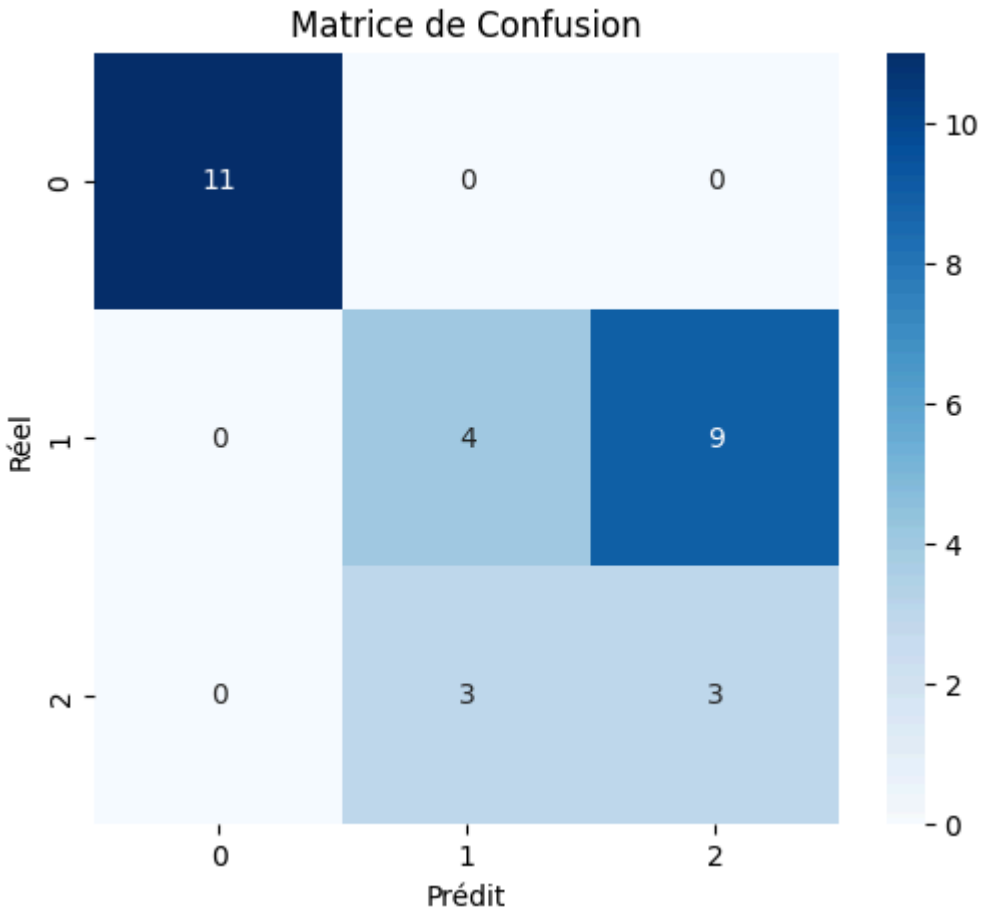
1. **Charger et préparer** le jeu de données Iris.
2. **Normaliser** les données pour améliorer les performances du modèle.
3. **Séparer** nos données en un ensemble d'entraînement et un ensemble de test.
4. **Entraîner** le SVM avec un noyau RBF.
5. **Évaluer** les performances du modèle via la matrice de confusion.
6. **Visualiser** la frontière de décision.

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# Étape 1 : Chargement du jeu de données Iris
iris = datasets.load_iris()

# Nous prenons uniquement les deux premières caractéristiques (longueur et large
X = iris.data[:, :2]
y = iris.target

# Étape 2 : Normalisation des données
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Étape 3 : Séparation en train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0
)

# Étape 4 : Entraînement du modèle SVM avec noyau RBF
svm = SVC(kernel='rbf', C=10, gamma=1)
svm.fit(X_train, y_train)

# Étape 5 : Prédictions et évaluation
y_pred = svm.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
```

Matrice de Confusion

Cette matrice indique les performances de notre classifieur. Les lignes représentent les **classes réelles** et les colonnes représentent les **classes prédites**.

In [4]:
```
def afficher_matrice_confusion(cm, labels):
    """
    Affiche la matrice de confusion en utilisant une heatmap.
    """
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.xlabel('Prédit')
    plt.ylabel('Réel')
    plt.title('Matrice de Confusion')
    plt.show()

# Affichage de la matrice de confusion
afficher_matrice_confusion(cm, np.unique(y))
```

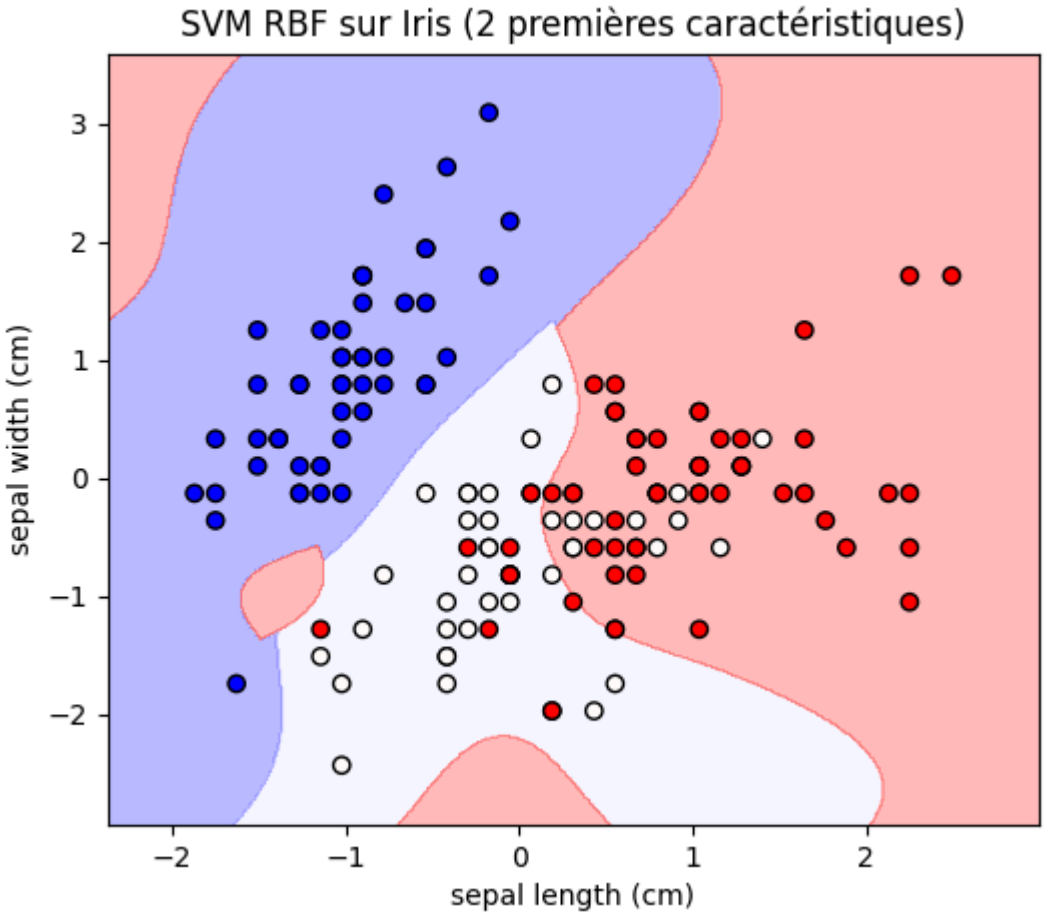


Visualisation de la Frontière de Décision

Nous allons tracer la zone de décision du SVM pour les deux premières caractéristiques.
Chaque point du plan 2D représente un échantillon (une fleur d'iris).
La couleur correspond à la classe prédite ou réelle.

In [5]:
```
def afficher_frontiere_decision(X, y, model, feature_names):
    """
    Affiche la frontière de décision du modèle SVM entraîné
    sur deux caractéristiques (longueur et largeur de sépale).
```

```
    """
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 500),
        np.linspace(y_min, y_max, 500)
    )

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(6, 5))
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='bwr')
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', edgecolors='k')
    plt.xlabel(feature_names[0])
    plt.ylabel(feature_names[1])
    plt.title("SVM RBF sur Iris (2 premières caractéristiques)")
    plt.show()

# Affichage de la frontière de décision
afficher_frontiere_decision(X, y, svm, iris.feature_names[:2])
```



Interprétation

1. **Matrice de confusion**

   - Les valeurs diagonales indiquent le nombre de fleurs correctement classées pour chaque variété d'iris.

- Une matrice de confusion proche de la diagonale signifie un bon taux de classification.

2. **Frontière de décision**

- Les zones colorées montrent comment le modèle sépare les classes dans le plan défini par (longueur_sépale, largeur_sépale).
- On observe clairement les régions prédominantes de chaque classe.

3. **Paramètres du SVM**

- **C** : Contrôle la pénalisation des erreurs. Plus la valeur est élevée, plus le modèle essaie de classer parfaitement chaque point, au risque de surapprendre.
- **Gamma** : Contrôle l'influence d'un exemple d'entraînement. Plus gamma est élevé, moins la distance aura d'effet (frontières plus complexes).

4. **Utilisation de deux caractéristiques**

- Nous aurions pu utiliser les quatre caractéristiques (longueur_sépale, largeur_sépale, longueur_pétale, largeur_pétale) pour de meilleures performances.
- Ici, nous limitons volontairement le jeu de données à deux dimensions pour visualiser la frontière de décision.

In [9]:
```
# Test 1 : SVM avec C=1 et gamma=0.1
svm_test1 = SVC(kernel='rbf', C=1, gamma=0.1)
svm_test1.fit(X_train, y_train)
y_pred_test1 = svm_test1.predict(X_test)

# Calcul de l'accuracy et de la matrice de confusion
acc_test1 = accuracy_score(y_test, y_pred_test1)
cm_test1 = confusion_matrix(y_test, y_pred_test1)

print("=== Test 1 : C=1, gamma=0.1 ===")
print(f"Accuracy : {acc_test1:.3f}")
print("Matrice de Confusion :")
print(cm_test1)
```

```
=== Test 1 : C=1, gamma=0.1 ===
Accuracy : 0.700
Matrice de Confusion :
[[11  0  0]
 [ 0  8  5]
 [ 0  4  2]]
```

Analyse du Test 1 – C=1, gamma=0.1 :

- **C=1** : Une valeur modérée qui offre un compromis entre l'importance des erreurs de classification et la marge.
- **gamma=0.1** : Un faible gamma qui produit une frontière de décision plus lisse et moins sensible aux variations locales des données.

Observations attendues :
La matrice de confusion devrait être relativement diagonale, indiquant que le modèle généralise bien sur cet ensemble de données simple.

In [10]:
```
# Test 2 : SVM avec C=1 et gamma=1
svm_test2 = SVC(kernel='rbf', C=1, gamma=1)
svm_test2.fit(X_train, y_train)
y_pred_test2 = svm_test2.predict(X_test)

# Calcul de l'accuracy et de la matrice de confusion
acc_test2 = accuracy_score(y_test, y_pred_test2)
cm_test2 = confusion_matrix(y_test, y_pred_test2)

print("=== Test 2 : C=1, gamma=1 ===")
print(f"Accuracy : {acc_test2:.3f}")
print("Matrice de Confusion :")
print(cm_test2)
```

```
=== Test 2 : C=1, gamma=1 ===
Accuracy : 0.667
Matrice de Confusion :
[[11  0  0]
 [ 0  5  8]
 [ 0  2  4]]
```

Analyse du Test 2 – C=1, gamma=1 :

- **C=1** reste modéré.
- **gamma=1** : L'augmentation de gamma rend la frontière de décision plus complexe et plus sensible aux points proches.

Observations attendues :
Cette configuration peut permettre de capturer davantage de détails dans la répartition des classes. Vérifiez si la matrice de confusion reste bien structurée et si l'accuracy s'améliore ou se dégrade par rapport au Test 1.

In [11]:
```
# Test 3 : SVM avec C=1 et gamma=10
svm_test3 = SVC(kernel='rbf', C=1, gamma=10)
svm_test3.fit(X_train, y_train)
y_pred_test3 = svm_test3.predict(X_test)

# Calcul de l'accuracy et de la matrice de confusion
acc_test3 = accuracy_score(y_test, y_pred_test3)
cm_test3 = confusion_matrix(y_test, y_pred_test3)

print("=== Test 3 : C=1, gamma=10 ===")
print(f"Accuracy : {acc_test3:.3f}")
print("Matrice de Confusion :")
print(cm_test3)
```

```
=== Test 3 : C=1, gamma=10 ===
Accuracy : 0.567
Matrice de Confusion :
[[11  0  0]
 [ 0  3 10]
 [ 0  3  3]]
```

Analyse du Test 3 – C=1, gamma=10 :

- **C=1** reste constant.

- **gamma=10** : Un gamma très élevé induit une frontière de décision très complexe, adaptée de près aux points d'entraînement.

Observations attendues :
Cette configuration risque de surajuster les données d'entraînement. Vérifiez la matrice de confusion pour repérer d'éventuelles erreurs de classification et observez si l'accuracy baisse.

In [12]:
```
# Test 4 : SVM avec C=10 et gamma=0.1
svm_test4 = SVC(kernel='rbf', C=10, gamma=0.1)
svm_test4.fit(X_train, y_train)
y_pred_test4 = svm_test4.predict(X_test)

# Calcul de l'accuracy et de la matrice de confusion
acc_test4 = accuracy_score(y_test, y_pred_test4)
cm_test4 = confusion_matrix(y_test, y_pred_test4)

print("=== Test 4 : C=10, gamma=0.1 ===")
print(f"Accuracy : {acc_test4:.3f}")
print("Matrice de Confusion :")
print(cm_test4)
```

```
=== Test 4 : C=10, gamma=0.1 ===
Accuracy : 0.633
Matrice de Confusion :
[[11  0  0]
 [ 0  5  8]
 [ 0  3  3]]
```

Analyse du Test 4 – C=10, gamma=0.1 :

- **C=10** : Une valeur plus élevée de C cherche à réduire fortement les erreurs sur l'ensemble d'entraînement.
- **gamma=0.1** : Un faible gamma maintient une frontière lisse.

Observations attendues :
Cette configuration peut améliorer la précision si les données ne sont pas bruyantes, tout en évitant le surajustement. Vérifiez que la matrice de confusion est satisfaisante.

In [13]:
```
# Test 5 : SVM avec C=10 et gamma=1
svm_test5 = SVC(kernel='rbf', C=10, gamma=1)
svm_test5.fit(X_train, y_train)
y_pred_test5 = svm_test5.predict(X_test)

# Calcul de l'accuracy et de la matrice de confusion
acc_test5 = accuracy_score(y_test, y_pred_test5)
cm_test5 = confusion_matrix(y_test, y_pred_test5)

print("=== Test 5 : C=10, gamma=1 ===")
print(f"Accuracy : {acc_test5:.3f}")
print("Matrice de Confusion :")
print(cm_test5)
```

```
=== Test 5 : C=10, gamma=1 ===
Accuracy : 0.600
Matrice de Confusion :
[[11  0  0]
 [ 0  4  9]
 [ 0  3  3]]
```

Analyse du Test 5 – C=10, gamma=1 :

- **C=10** : L'augmentation de C réduit les erreurs sur l'entraînement.
- **gamma=1** : Un gamma intermédiaire permet une bonne flexibilité de la frontière sans trop de sensibilité.

Observations attendues :
Cette combinaison tend à trouver un bon compromis entre biais et variance. Vérifiez si la matrice de confusion est bien orientée et si l'accuracy atteint un bon niveau.

In [14]:
```
# Test 6 : SVM avec C=10 et gamma=10
svm_test6 = SVC(kernel='rbf', C=10, gamma=10)
svm_test6.fit(X_train, y_train)
y_pred_test6 = svm_test6.predict(X_test)

# Calcul de l'accuracy et de la matrice de confusion
acc_test6 = accuracy_score(y_test, y_pred_test6)
cm_test6 = confusion_matrix(y_test, y_pred_test6)

print("=== Test 6 : C=10, gamma=10 ===")
print(f"Accuracy : {acc_test6:.3f}")
print("Matrice de Confusion :")
print(cm_test6)
```

```
=== Test 6 : C=10, gamma=10 ===
Accuracy : 0.533
Matrice de Confusion :
[[11  0  0]
 [ 0  3 10]
 [ 0  4  2]]
```

Analyse du Test 6 – C=10, gamma=10 :

- **C=10** : Le modèle est très pénalisé pour toute erreur sur l'entraînement.
- **gamma=10** : Un gamma très élevé produit une frontière très complexe, susceptible de surajuster les données.

Observations attendues :
Cette configuration est la plus risquée en termes de surajustement. Vérifiez si la matrice de confusion montre des erreurs significatives et si l'accuracy est inférieure par rapport aux autres tests.

Conclusion Finale

Au terme de ces différents tests, nous avons observé l'impact des paramètres **C** et **gamma** sur les performances d'un SVM à noyau RBF appliqué au jeu de données Iris (en utilisant uniquement les deux premières caractéristiques pour la visualisation).

Observations clés :

- **Impact de C :**
  - Une valeur faible de C (par exemple, 0.1) tolère davantage les erreurs d'entraînement et peut mener à un sous-ajustement (underfitting).
  - Une valeur élevée de C (par exemple, 10) force le modèle à minimiser les erreurs sur l'entraînement, ce qui peut améliorer l'accuracy mais augmenter le risque de surajustement (overfitting).
- **Impact de gamma :**
  - Un faible gamma (par exemple, 0.1) produit une frontière de décision lisse, moins sensible aux variations locales.
  - Un gamma élevé (par exemple, 10) rend la frontière de décision très complexe et sensible aux données d'entraînement, avec un risque accru de surajustement.
- **Compromis entre biais et variance :**
  Le choix optimal de ces paramètres doit équilibrer la capacité du modèle à généraliser (faible variance) et sa capacité à capturer la structure des données (faible biais). Par exemple, des valeurs intermédiaires comme **C=10 et gamma=1** ou **C=5 et gamma=0.5** se révèlent souvent être des compromis satisfaisants.

En conclusion, ces expérimentations montrent qu'une recherche minutieuse des hyperparamètres est cruciale pour optimiser la performance d'un modèle SVM. Pour des applications réelles, il est recommandé d'utiliser des techniques d'optimisation automatisées telles que **GridSearchCV** ou **RandomizedSearchCV** afin de trouver le meilleur couple de paramètres pour le problème spécifique à traiter.

In [ ]: