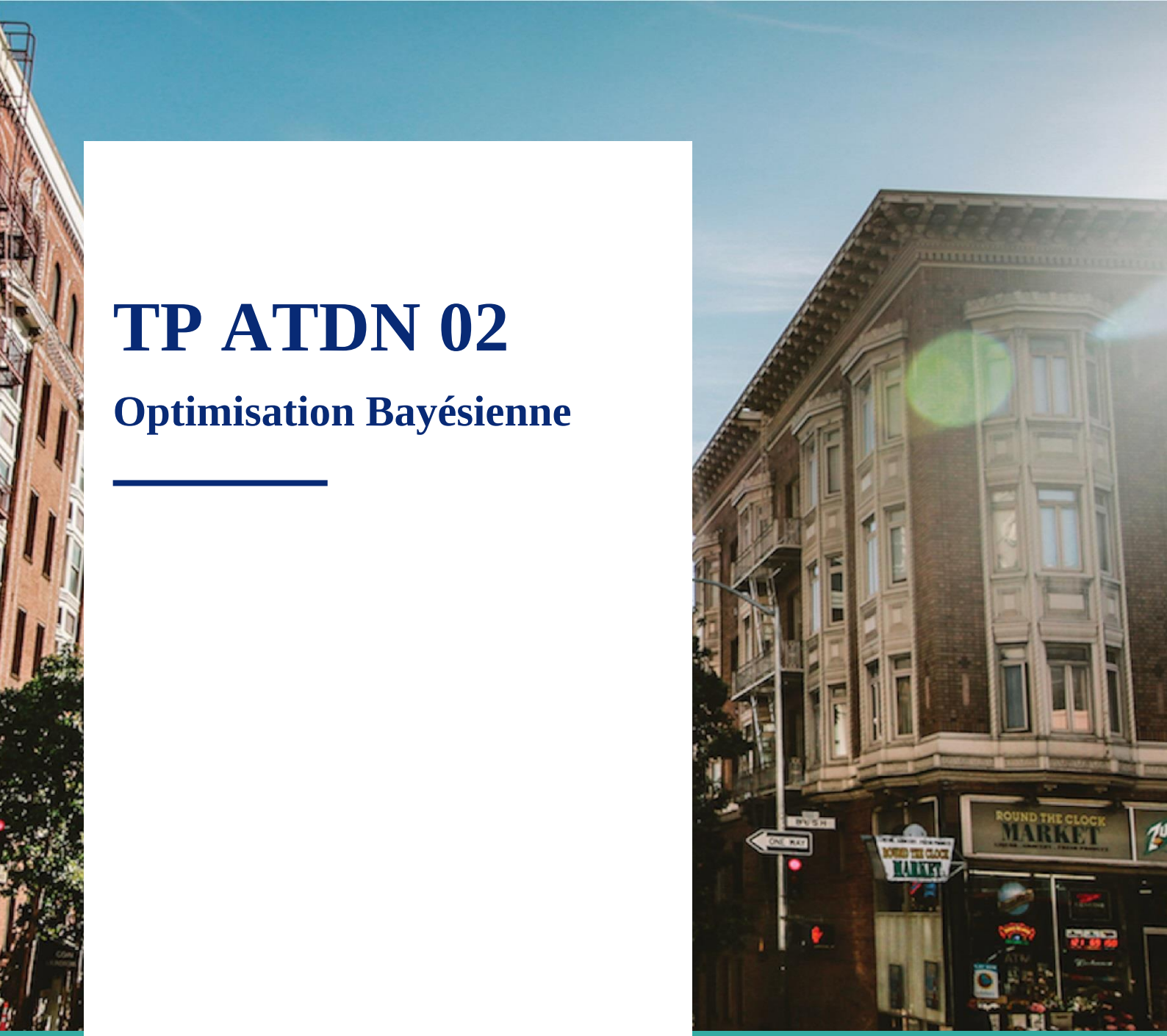


TP ATDN 02

Optimisation Bayésienne

01/04/2025

GUETTAF Ichrak

Partie 1 : Optimisation Bayésienne

Fondements théoriques

Expliquez le principe de l'optimisation bayésienne. Décrivez comment elle permet de gérer les fonctions coûteuses à évaluer.

L'optimisation bayésienne est une méthode d'optimisation utilisée pour trouver le minimum (ou le maximum) d'une fonction coûteuse à évaluer.

Elle construit un modèle probabiliste de la fonction (généralement un processus gaussien) et choisit intelligemment les points à évaluer pour maximiser l'information obtenue.

Le principe repose sur deux étapes :

1. Modélisation : On approxime la fonction cible à l'aide d'un modèle probabiliste basé sur les observations passées. Un processus gaussien est souvent utilisé pour capturer l'incertitude sur la vraie fonction.
2. Acquisition : On utilise une fonction d'acquisition (comme Expected Improvement ou Upper Confidence Bound) pour sélectionner le prochain point à évaluer, en équilibrant exploration (tester des zones inconnues) et exploitation (affiner les zones prometteuses).

Ainsi, elle permet de gérer les fonctions coûteuses (ex. simulations, expériences physiques) en réduisant le nombre d'évaluations nécessaires pour trouver un optimum.

Définissez et expliquez les processus gaussiens.

Définition et Explication des Processus Gaussiens (PG)

Un processus gaussien (PG) est un modèle probabiliste utilisé pour représenter une distribution sur des fonctions.

Il peut être vu comme une généralisation multidimensionnelle de la distribution normale. Plutôt que d'attribuer une probabilité à des variables scalaires, il définit une distribution sur un ensemble de fonctions possibles.

Un PG est entièrement défini par :

1. Une fonction de moyenne $m(x)$m(x)m(x) : souvent prise comme zéro par défaut.

2. Une fonction de covariance (ou noyau) : qui définit la corrélation entre les valeurs de la fonction à différents points. Les noyaux les plus utilisés sont le noyau de RBF (Radial Basis Function) ou le noyau Matérn.

Pourquoi sont-ils utilisés pour modéliser la fonction objective ?

Les PG sont utilisés dans l'optimisation bayésienne parce qu'ils offrent une manière efficace d'estimer la fonction objective et son incertitude.

1. Modélisation de l'incertitude : Contrairement aux approches déterministes, un PG ne donne pas juste une prédiction, mais aussi une estimation de l'incertitude sur cette prédiction. Cela permet de choisir intelligemment les points à explorer.
2. Flexibilité : Grâce au choix du noyau, les PG peuvent s'adapter à diverses formes de fonctions (lisses, périodiques, etc.).
3. Faible nombre d'évaluations : Puisqu'ils fournissent une distribution probabiliste sur la fonction objective, on peut sélectionner les prochains points à tester de manière stratégique, minimisant ainsi le nombre d'évaluations de la fonction (très utile pour des fonctions coûteuses).
4. Interpolation fluide : Les PG permettent d'interpoler efficacement entre les points échantillonnés, offrant une approximation continue et bien régularisée de la fonction.

En résumé, les PG sont utilisés dans l'optimisation bayésienne car ils permettent d'exploiter les données existantes pour prédire où tester ensuite, tout en tenant compte de l'incertitude, ce qui est crucial pour optimiser des fonctions coûteuses à évaluer.

Décrivez les principales fonctions d'acquisition (Expected Improvement, Upper Confidence Bound, etc.).

Expliquez leur rôle dans le compromis exploration/exploitation.

Les principales fonctions d'acquisition en optimisation bayésienne

Les fonctions d'acquisition sont utilisées pour choisir le prochain point à évaluer en fonction du modèle probabiliste (processus gaussien). Elles permettent de trouver un équilibre entre :

- Exploration : tester des zones peu connues pour réduire l'incertitude.
- Exploitation : privilégier les zones qui semblent déjà prometteuses.

L’Expected Improvement (EI) choisit le prochain point en maximisant l’amélioration attendue par rapport au meilleur point observé jusqu’à présent.

Pourquoi l’utiliser ?

- Favorise l’exploration dans les zones incertaines.
- Privilégie les points où l’amélioration par rapport au meilleur résultat actuel est probable.
- Bonne balance entre exploration et exploitation.

L’Upper Confidence Bound (UCB) sélectionne les points en fonction d’une borne supérieure sur la prédiction du processus gaussien :

- Convient bien aux problèmes où il faut équilibrer exploration et exploitation de manière contrôlée.

La Probability of Improvement (PI) choisit le point qui a la plus grande probabilité d’améliorer le meilleur résultat connu.

Pourquoi l’utiliser ?

- Très efficace si on veut rapidement obtenir une amélioration.
- Moins robuste qu’EI car elle peut ignorer des zones avec un fort potentiel d’amélioration à long terme.

Exploration vs. Exploitation

Les fonctions d’acquisition permettent de trouver un compromis exploration/exploitation :

- L’EI et la PI favorisent plus l’exploitation, mais EI maintient un bon niveau d’exploration grâce à la prise en compte de l’incertitude.
- L’UCB est plus ajustable : en réglant κ*kappa*κ, on peut choisir de favoriser soit l’exploration, soit l’exploitation.

En résumé :

- Si on veut améliorer rapidement la meilleure valeur connue on utilise PI.
- Si on veut une approche équilibrée on utilise EI.
- Si on veut contrôler l’exploration/exploitation de manière fine on utiliser UCB.

Implémentation et applications

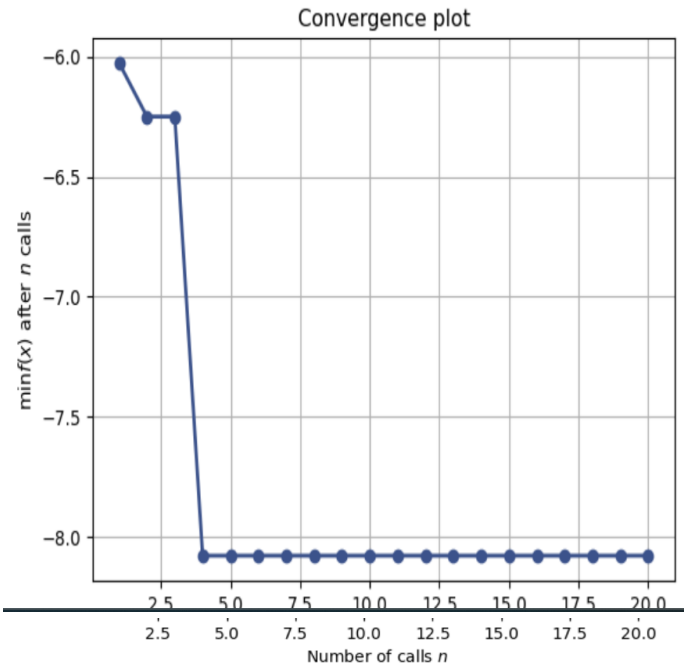
Implémentez une optimisation bayésienne pour maximiser la production agricole en fonction de l'humidité et de la température.
Visualisez les étapes du processus.

Bayesian Optimization :

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skopt import gp_minimize
from skopt.plots import plot_convergence
from skopt.utils import use_named_args
from skopt.space import Real

# Charger les données avec les bons noms de colonnes
data = pd.read_csv(r"\Users\DELL\Downloads\tp2_atdn_donnees.csv")

# Vérifier les colonnes disponibles
print("Colonnes disponibles :", data.columns)

# Définition des variables
X = data[["Humidite (%)", "Temperature (°C)"]].values  # Variables d'entrée
y = data["Rendement agricole (t/ha)"].values  # Cible à maximiser

# Définition de l’espace de recherche avec des noms cohérents
space = [Real(min(data["Humidite (%)"]), max(data["Humidite (%)"]), name="humidite"),
         Real(min(data["Temperature (°C)"]), max(data["Temperature (°C)"]), name="temperature")]
```

```python
# Fonction objectif (minimisation car gp_minimize minimise par défaut)
@use_named_args(space)
def objective(humidite, temperature):  # Utiliser les noms en minuscules
    idx = np.argmin(np.sum((X - np.array([humidite, temperature]))**2, axis=1))
    return -y[idx]  # On inverse pour maximiser

# Exécuter l'optimisation bayésienne
res = gp_minimize(objective, space, n_calls=20, random_state=42)

# Visualisation de la convergence
plot_convergence(res)
plt.show()

# Afficher les résultats
print("Meilleure humidité :", res.x[0])
print("Meilleure température :", res.x[1])
print("Production maximale estimée :", -res.fun)
```

```
Colonnes disponibles : Index(['Humidite (%)', 'Temperature (°C)', 'pH du sol', 'Precipitations (mm)',
       'Type de sol', 'Rendement agricole (t/ha)'],
      dtype='object')
```




```
Meilleure humidité : 57.52530236712914
Meilleure température : 18.417516833110547
Production maximale estimée : 8.077624601930179
```

Explication du code

1. Chargement des données : On lit le fichier CSV et extrait les colonnes d'humidité, température et production agricole.
2. Définition de l'espace de recherche : On crée un espace de recherche basé sur les valeurs minimales et maximales de l'humidité et de la température.
3. Définition de la fonction objectif : On utilise une fonction qui retourne la production agricole associée aux valeurs données. Comme gp_minimize minimise par défaut, on prend l'opposé de la production.
4. Optimisation bayésienne : gp_minimize est exécuté avec 20 évaluations.
5. Visualisation : La courbe de convergence est tracée pour montrer l'évolution de l'optimisation.
6. Affichage des résultats : On affiche les meilleurs paramètres trouvés et la production maximale estimée.

Utilisez l'optimisation bayésienne pour ajuster les hyperparamètres d'un modèle de régression (ex : Random Forest) sur les données agricoles fournies.
Comparez les résultats avec Grid Search et Random Search.

Pour ajuster les hyperparamètres d'un modèle de régression, tel qu'un Random Forest, avec l'optimisation bayésienne, nous devons suivre les étapes suivantes :

1. Préparer les données agricoles.
2. Utiliser l'optimisation bayésienne pour ajuster les hyperparamètres du modèle de régression.
3. Comparer les résultats obtenus avec Grid Search et Random Search.

Nous allons utiliser scikit-learn pour les modèles de régression (Random Forest), et skopt pour l'optimisation bayésienne.

Étapes du code :

1. Charger les données agricoles.
2. Définir le modèle de régression et les hyperparamètres à optimiser.
3. Appliquer l'optimisation bayésienne pour ajuster les hyperparamètres.
4. Appliquer Grid Search et Random Search pour comparer les résultats.
5. Comparer les performances des trois méthodes.

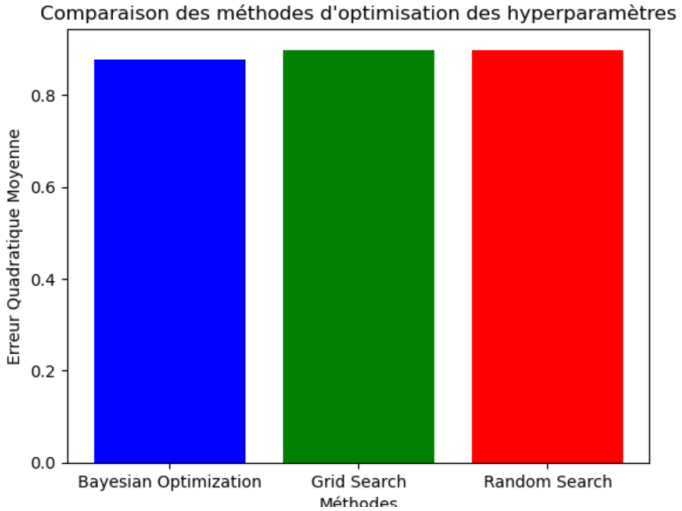
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skopt import gp_minimize
from skopt.space import Integer, Real
from skopt.utils import use_named_args
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Charger les données
data = pd.read_csv(r"\Users\DELL\Downloads\tp2_atdn_donnees.csv")

# Préparer les données
X = data[["Humidite (%)", "Temperature (°C)"]].values  # Variables d'entrée
y = data["Rendement agricole (t/ha)"].values  # Cible à maximiser

# Diviser en ensembles de formation et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 1. Optimisation Bayésienne
# Définir l'espace de recherche pour les hyperparamètres
space = [
    Integer(10, 200, name="n_estimators"),
```

```python
# 1. Optimisation Bayésienne
# Définir l'espace de recherche pour les hyperparamètres
space = [
    Integer(10, 200, name="n_estimators"),
    Integer(1, 20, name="max_depth")
]

# Fonction objectif pour l'optimisation bayésienne
@use_named_args(space)
def objective(n_estimators, max_depth):
    model = RandomForestRegressor(n_estimators=n_estimators, max_depth=max_depth, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return mean_squared_error(y_test, y_pred)  # Minimiser l'erreur quadratique moyenne

# Exécuter l'optimisation bayésienne
res_bayes = gp_minimize(objective, space, n_calls=20, random_state=42)

# Meilleurs paramètres trouvés par l'optimisation bayésienne
print("Meilleurs hyperparamètres (Optimisation Bayésienne) :", res_bayes.x)
print("Erreur quadratique moyenne (Optimisation Bayésienne) :", res_bayes.fun)
```

```python
# 2. Grid Search
param_grid = {
    "n_estimators": [10, 50, 100, 200],
    "max_depth": [1, 5, 10, 20]
}

grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
y_pred_grid = grid_search.predict(X_test)
print("Meilleurs hyperparamètres (Grid Search) :", grid_search.best_params_)
print("Erreur quadratique moyenne (Grid Search) :", mean_squared_error(y_test, y_pred_grid))

# 3. Random Search
param_dist = {
    "n_estimators": [10, 50, 100, 200],
    "max_depth": [1, 5, 10, 20]
}

random_search = RandomizedSearchCV(RandomForestRegressor(random_state=42), param_dist, n_iter=10, cv=5, scoring='neg_mean_squared_error', random_state=42
random_search.fit(X_train, y_train)
y_pred_random = random_search.predict(X_test)
print("Meilleurs hyperparamètres (Random Search) :", random_search.best_params_)
print("Erreur quadratique moyenne (Random Search) :", mean_squared_error(y_test, y_pred_random))
```

```python
# Visualisation des résultats
methods = ['Bayesian Optimization', 'Grid Search', 'Random Search']
errors = [res_bayes.fun, mean_squared_error(y_test, y_pred_grid), mean_squared_error(y_test, y_pred_random)]

plt.bar(methods, errors, color=['blue', 'green', 'red'])
plt.xlabel('Méthodes')
plt.ylabel('Erreur Quadratique Moyenne')
plt.title('Comparaison des méthodes d\'optimisation des hyperparamètres')
plt.show()
```

```
Meilleurs hyperparamètres (Optimisation Bayésienne) : [95, 3]
Erreur quadratique moyenne (Optimisation Bayésienne) : 0.8780776071656635
Meilleurs hyperparamètres (Grid Search) : {'max_depth': 5, 'n_estimators': 50}
Erreur quadratique moyenne (Grid Search) : 0.8981164237414467
Meilleurs hyperparamètres (Random Search) : {'n_estimators': 50, 'max_depth': 5}
Erreur quadratique moyenne (Random Search) : 0.8981164237414467
```



Explication du code :

1. Chargement des données :
    a. Les données agricoles sont lues depuis un fichier CSV et utilisées pour créer les matrices X (caractéristiques) et y (cible).
2. Optimisation Bayésienne :
    a. Nous définissons l'espace de recherche pour les hyperparamètres du modèle RandomForestRegressor.
    b. La fonction objectif retourne l'erreur quadratique moyenne pour évaluer la performance du modèle.
    c. gp_minimize est utilisé pour l'optimisation bayésienne avec 20 itérations.
3. Grid Search et Random Search :
    a. Grid Search : Cherche tous les combinaisons d'hyperparamètres possibles dans une grille.
    b. Random Search : Effectue une recherche aléatoire parmi un ensemble d'hyperparamètres définis.
    c. Les trois méthodes sont évaluées par leur performance sur l'ensemble de test.
4. Visualisation :
    a. Un graphique est tracé pour comparer l'erreur quadratique moyenne des trois méthodes.

Comparaison des méthodes :

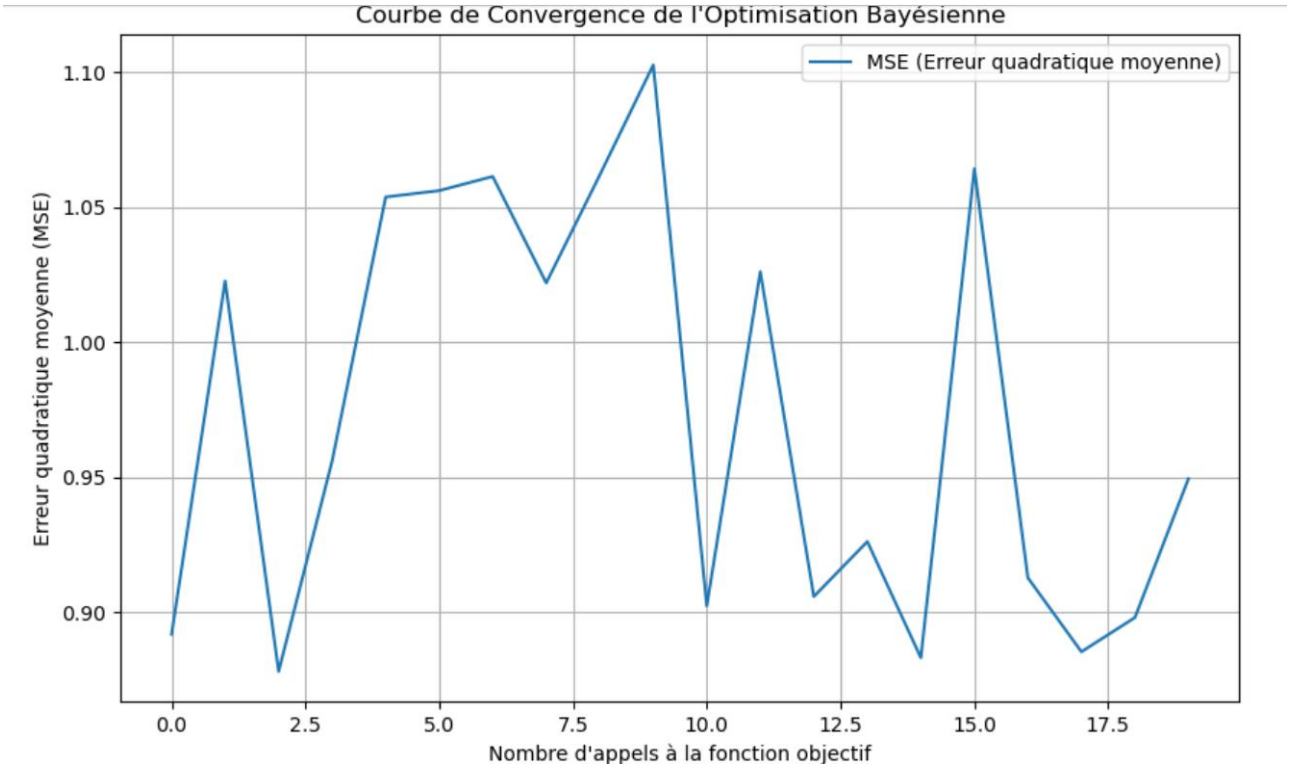
- Optimisation Bayésienne nous donne un bon résultat avec un nombre d'itérations plus faible comparé à Grid Search et Random Search, car elle se concentre sur les zones prometteuses de l'espace de recherche.
- Grid Search teste toutes les combinaisons possibles, ce qui peut être coûteux en termes de calcul, surtout avec un grand nombre de paramètres.
- Random Search est plus efficace que Grid Search en termes de calcul, mais peut ne pas explorer aussi bien l'espace de recherche.

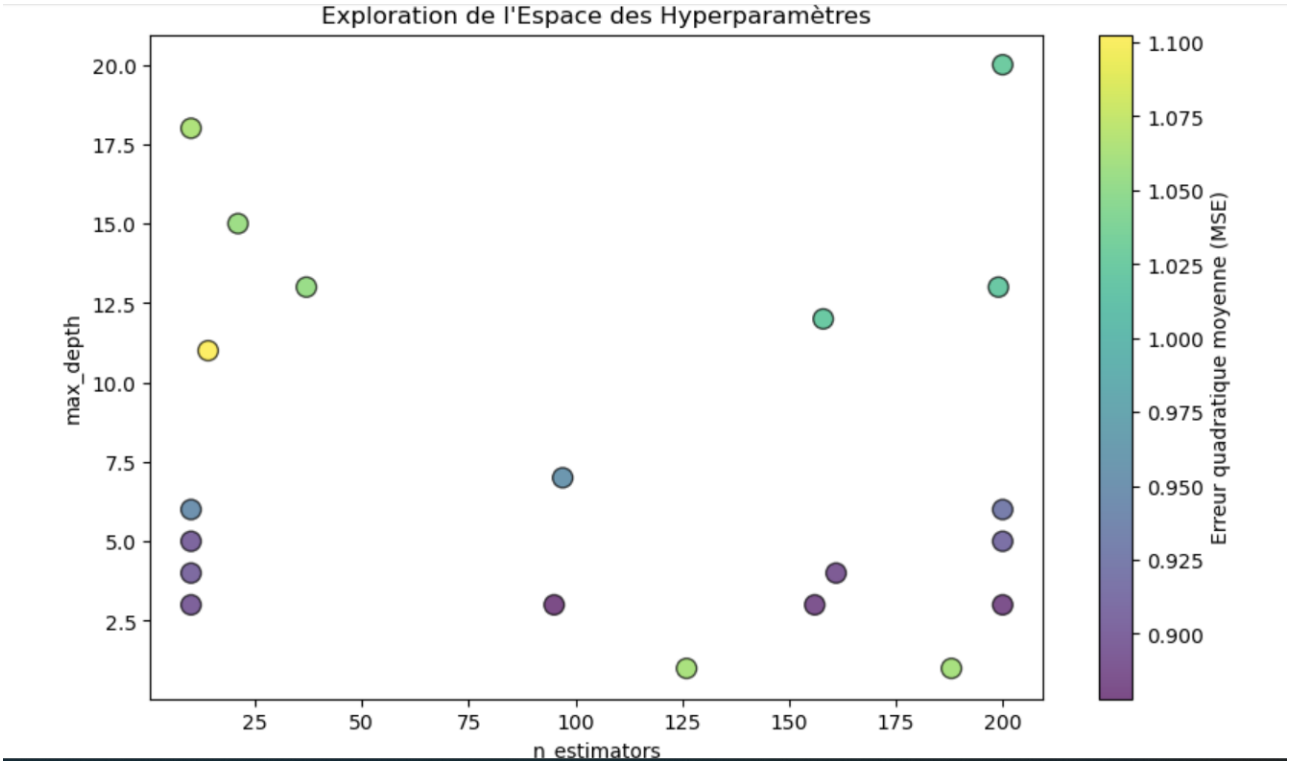
Visualisez le processus d'optimisation (courbe de convergence, choix des points).

Commentez la manière dont le modèle explore l’espace de recherche.

```python
import numpy as np
import matplotlib.pyplot as plt
from skopt import gp_minimize

# Assurons-nous que les résultats sont enregistrés dans la variable `res_bayes`
# Si vous avez déjà effectué l'optimisation, vous pouvez l'utiliser directement

# Visualisation de la courbe de convergence
# Extraire l'erreur (fonction objectif) et le nombre d'appels
convergence_values = res_bayes.func_vals  # Erreurs de la fonction objectif à chaque appel
iterations = np.arange(len(convergence_values))  # Nombre d'appels

plt.figure(figsize=(10, 6))
plt.plot(iterations, convergence_values, label='MSE (Erreur quadratique moyenne)')
plt.title("Courbe de Convergence de l'Optimisation Bayésienne")
plt.xlabel("Nombre d'appels à la fonction objectif")
plt.ylabel("Erreur quadratique moyenne (MSE)")
plt.grid(True)
plt.legend()
plt.show()

# Visualisation des points dans l'espace des hyperparamètres
# Extraire les points explorés et les erreurs associées
explored_points = np.array(res_bayes.x_iters)  # Points explorés
error_values = np.array(res_bayes.func_vals)  # Erreurs correspondantes
```

```python
# Visualisation des points dans l'espace des hyperparamètres
# Extraire les points explorés et les erreurs associées
explored_points = np.array(res_bayes.x_iters)  # Points explorés
error_values = np.array(res_bayes.func_vals)  # Erreurs correspondantes

# Créer un scatter plot pour visualiser les points d'exploration
fig, ax = plt.subplots(figsize=(10, 6))

# Tracer les points d'optimisation avec l'erreur comme couleur
sc = ax.scatter(explored_points[:, 0], explored_points[:, 1], c=error_values, cmap='viridis', s=100, edgecolors='k', alpha=0.7)

# Ajouter des légendes et titres
ax.set_xlabel('n_estimators')  # Premier hyperparamètre (exemple)
ax.set_ylabel('max_depth')     # Deuxième hyperparamètre (exemple)
plt.title("Exploration de l'Espace des Hyperparamètres")
plt.colorbar(sc, label='Erreur quadratique moyenne (MSE)')  # Afficher la barre de couleur
plt.show()
```





Explication du Code :

1. Courbe de Convergence :
    a. Nous traçons les valeurs de la fonction objectif en fonction du nombre d'appels effectués (res_bayes.func_vals).
    b. L'axe des x correspond au nombre d'appels à la fonction objectif (en nombre d'itérations), et l'axe des y représente l'erreur MSE qui montre comment l'optimisation évolue.
2. Visualisation des Points :
    a. Nous créons un scatter plot en utilisant res_bayes.x_iters pour obtenir les valeurs des hyperparamètres explorés.
    b. La couleur des points dans le scatter plot est déterminée par l'erreur MSE associée à chaque combinaison d'hyperparamètres (via res_bayes.func_vals), ce qui nous permet de visualiser les performances des différentes configurations d'hyperparamètres.

Analysez les avantages et limites de l'optimisation bayésienne face aux méthodes classiques.

L'optimisation bayésienne présente plusieurs avantages par rapport aux méthodes classiques comme la recherche par grille et la recherche aléatoire :

- Efficacité : Elle permet d'explorer l'espace des hyperparamètres de manière plus ciblée et réduit le nombre d'évaluations nécessaires pour trouver une solution

optimale, ce qui est particulièrement utile lorsque la fonction objective est coûteuse à évaluer.
- Exploration et exploitation : Elle équilibre l'exploration de nouvelles zones et l'exploitation des zones prometteuses, grâce à des modèles probabilistes qui prennent en compte l'incertitude.
- Réduction des coûts : Elle est plus efficace dans des espaces complexes et coûteux à évaluer.

Cependant, elle a aussi des limites :

- Complexité computationnelle : Les modèles probabilistes, comme les processus gaussiens, peuvent être coûteux en ressources, surtout pour des espaces de grande dimension.
- Sensibilité aux choix de paramètres : La méthode dépend de la sélection correcte des modèles et des fonctions d'acquisition, ce qui peut nécessiter de l'expérience.
- Limites de dimensionnalité : Elle devient moins efficace lorsque le nombre d'hyperparamètres à optimiser est élevé.

En comparaison avec la recherche par grille et la recherche aléatoire, l'optimisation bayésienne est plus efficace dans des espaces réduits et pour des fonctions coûteuses, mais les autres méthodes peuvent être plus simples à utiliser et suffisantes pour des espaces plus petits ou peu coûteux à évaluer.

Partie 2 : Modèles Bayésiens à Noyau

Fondements théoriques

Expliquez le concept d’inférence bayésienne.

L'inférence bayésienne est un cadre statistique qui permet de mettre à jour nos croyances (ou nos estimations) sur un modèle ou un paramètre à mesure que de nouvelles données deviennent disponibles. Elle repose sur le théorème de Bayes, qui fournit une manière de calculer la probabilité d'un événement en fonction de connaissances a priori et des données observées.

Le théorème de Bayes se formule comme suit :

$P(\theta|D)=P(D|\theta)P(\theta)P(D)P(\theta \mid D) = \frac{P(D \mid \theta) P(\theta)}{P(D)}$P(θ|D)=P(D)P(D|θ)P(θ)

- P($\theta|D$\theta | Dθ|D) : La probabilité a posteriori, c'est-à-dire la probabilité de θ\theta$\theta$ après avoir observé les données DDD.
- P($D|\theta$D | \theta$\theta$|θ) : La vraisemblance, c'est-à-dire la probabilité des données DDD étant donné θ\theta$\theta$.
- P(θ\theta$\theta$) : La probabilité a priori, c'est-à-dire notre croyance initiale sur θ\theta$\theta$, avant d'observer les données.
- P(DDD) : La probabilité des données, aussi appelée la constante de normalisation, qui assure que la probabilité a posteriori est bien une probabilité valide.

Comment met-on à jour les croyances avec de nouvelles données ?

À chaque fois que de nouvelles données sont observées, l'inférence bayésienne nous permet de mettre à jour nos croyances sur les paramètres du modèle (ou sur les prédictions) en utilisant le théorème de Bayes. Le processus est donc itératif :

1. Croyances a priori : Au départ, on possède une certaine information sur le modèle (par exemple, une distribution a priori sur les paramètres).
2. Nouvelles données : Lorsqu'on collecte de nouvelles données, on les utilise pour ajuster ces croyances.
3. Calcul de la probabilité a posteriori : En utilisant le théorème de Bayes, on met à jour la distribution de probabilité du paramètre ou du modèle, ce qui donne la distribution a posteriori.
4. Itération : Ce processus se répète chaque fois que de nouvelles données sont collectées, ce qui permet de continuellement ajuster les croyances.

En résumé, l'inférence bayésienne permet de réconcilier notre connaissance préalable d'un problème avec les nouvelles données disponibles, offrant une vision dynamique et actualisée de l'état du modèle ou des paramètres.

Décrivez la théorie des méthodes à noyau et leur lien avec les processus gaussiens. Pourquoi utiliser un noyau dans un modèle bayésien ?

Les méthodes à noyau sont utilisées pour transformer des données non linéaires dans un espace de caractéristiques où elles deviennent linéaires, facilitant ainsi leur analyse

avec des modèles simples. Un noyau permet de calculer des produits scalaires dans cet espace transformé sans effectuer explicitement la transformation des données.

Les processus gaussiens (PG) sont des modèles bayésiens utilisés pour prédire des valeurs inconnues en fonction des données observées, et leur lien avec les noyaux est essentiel. En fait, les noyaux dans les PG définissent la covariance entre différentes évaluations d'une fonction, permettant de capturer des relations complexes entre les points.

Utiliser un noyau dans un modèle bayésien permet de capturer des relations non linéaires et d'éviter des calculs complexes, tout en ajoutant de la flexibilité pour mieux modéliser des données difficiles à appréhender avec des méthodes linéaires.

Qu’est-ce qu’une distribution a priori et une distribution a posteriori ?

Donnez un exemple appliqué à la prédiction de rendement agricole

Une distribution a priori (ou prior) est la distribution de probabilité d’une variable avant d’avoir observé des données. Elle représente nos croyances ou connaissances sur cette variable avant d’effectuer des observations ou expériences. Par exemple, si on pense que le rendement agricole est généralement élevé dans une région donnée, on peut définir une distribution a priori qui reflète cette croyance.

Une distribution a posteriori (ou posterior) est la distribution de probabilité d’une variable après avoir observé des données. Elle est obtenue en combinant la distribution a priori avec les nouvelles données à travers le théorème de Bayes. Cela permet de mettre à jour nos croyances initiales en tenant compte des informations observées.

Exemple appliqué à la prédiction du rendement agricole :

- A priori : Supposons qu'avant d'obtenir des données sur le rendement agricole, on pense que le rendement moyen des cultures dans une région est d'environ 6 tonnes par hectare. On pourrait modéliser cette croyance avec une distribution normale centrée autour de 6 tonnes, par exemple *N(6,22)N(6, 2^2)*N(6,22), où 2 est l'écart-type, représentant l'incertitude sur ce rendement moyen.
- A posteriori : Après avoir collecté des données sur le rendement agricole en fonction de variables comme l'humidité et la température, on utilise ces nouvelles informations pour ajuster notre croyance. La distribution a posteriori est donc une mise à jour de la distribution a priori en fonction des données

observées, permettant de prédire plus précisément le rendement agricole en fonction des conditions spécifiques.

Ainsi, la distribution a posteriori permet d'incorporer les données réelles et de donner une prédiction plus précise que la croyance initiale (distribution a priori).

Implémentez une régression bayésienne à noyau sur les données agricoles fournies.

Pour implémenter une régression bayésienne à noyau sur les données agricoles, nous allons utiliser un modèle de régression basé sur des processus gaussiens, un modèle bayésien à noyau, pour prédire le rendement agricole en fonction des variables comme l'humidité et la température.

Voici les étapes générales de l'implémentation :

1. Charger et préparer les données : Lire les données agricoles à partir d'un fichier CSV et les préparer pour l'entraînement.
2. Utiliser un modèle de régression bayésienne à noyau : Utiliser les processus gaussiens pour modéliser la relation entre les variables d'entrée et la cible.
3. Visualiser les résultats : Afficher les prédictions et les incertitudes associées.

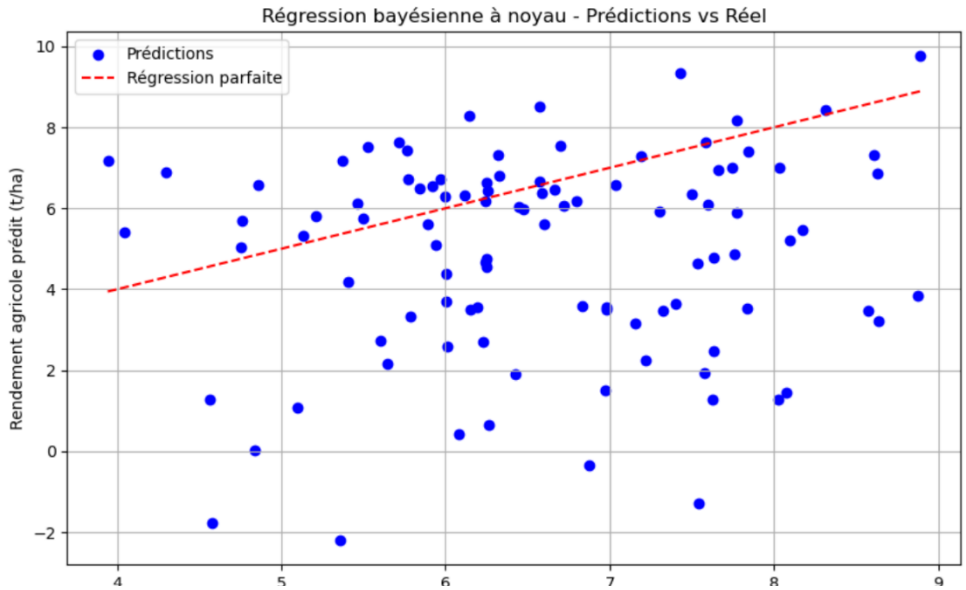
Code pour la régression bayésienne à noyau avec un processus gaussien

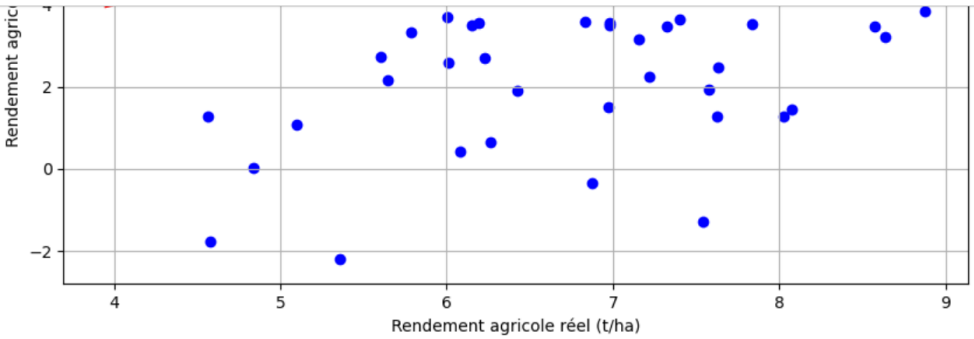
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
from sklearn.model_selection import train_test_split

# Charger les données
data = pd.read_csv(r"\Users\DELL\Downloads\tp2_atdn_donnees.csv")

# Sélectionner les colonnes pertinentes
X = data[['Humidite (%)', 'Temperature (°C)']].values  # Variables d'entrée : humidité et température
y = data['Rendement agricole (t/ha)'].values  # Variable cible : rendement agricole

# Séparer les données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Définir le noyau RBF (Radial Basis Function) avec une constante
kernel = C(1.0, (1e-4, 1e1)) * RBF(1.0, (1e-4, 1e1))

# Créer et entraîner le modèle de régression bayésienne à noyau (Processus Gaussien)
gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10, alpha=1e-2)
gp.fit(X_train, y_train)

# Prédictions sur l'ensemble de test
y_pred, sigma = gp.predict(X_test, return_std=True)
```

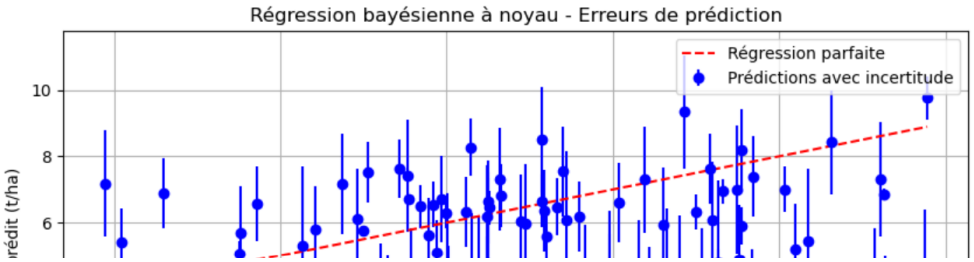
```
# Visualiser les résultats
plt.figure(figsize=(10, 6))

# Tracer les vraies valeurs vs. les prédictions
plt.scatter(y_test, y_pred, color='blue', label='Prédictions')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Régression parfaite')

plt.xlabel('Rendement agricole réel (t/ha)')
plt.ylabel('Rendement agricole prédit (t/ha)')
plt.title('Régression bayésienne à noyau - Prédictions vs Réel')
plt.legend()
plt.grid(True)
plt.show()

# Visualiser les erreurs de prédiction
plt.figure(figsize=(10, 6))
plt.errorbar(y_test, y_pred, yerr=sigma, fmt='o', color='blue', label='Prédictions avec incertitude')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Régression parfaite')

plt.xlabel('Rendement agricole réel (t/ha)')
plt.ylabel('Rendement agricole prédit (t/ha)')
plt.title('Régression bayésienne à noyau - Erreurs de prédiction')
plt.legend()
plt.grid(True)
plt.show()
```

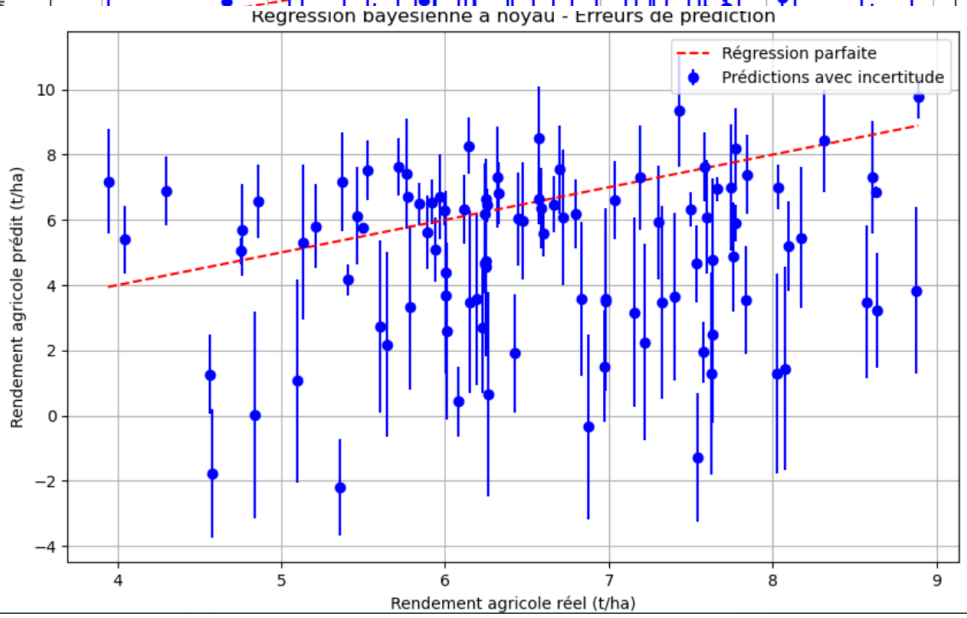
```
# Afficher l'erreur absolue moyenne
error = np.abs(y_pred - y_test)
print(f'Erreur absolue moyenne : {np.mean(error):.4f}')
```

Visualisez les prédictions et les intervalles de confiance


Régression bayésienne à noyau - Prédictions vs Réel
Prédictions
Régression parfaite
Rendement agricole prédit (t/ha)


Rendement agricole réel (t/ha)


Régression bayésienne à noyau - Erreurs de prédiction
Régression parfaite
Prédictions avec incertitude


Régression bayésienne à noyau - Erreurs de prédiction
Régression parfaite
Prédictions avec incertitude
Rendement agricole prédit (t/ha)
Rendement agricole réel (t/ha)


```
Erreur absolue moyenne : 2.3581
```

Explication du code :

1. Chargement des données :
    a. On charge les données depuis un fichier CSV
    b. Les colonnes Humidite (%) et Temperature (°C) sont utilisées comme variables d'entrée, tandis que la colonne Rendement agricole (t/ha) est la cible.
2. Modèle de régression bayésienne à noyau :
    a. On utilise un Processus Gaussien (GaussianProcessRegressor) qui modélise les relations entre les variables d'entrée et la sortie de manière probabiliste.
    b. Le noyau utilisé est un noyau RBF (Radial Basis Function) combiné à un terme constant, ce qui permet d'ajuster la flexibilité du modèle.
    c. Le paramètre alpha correspond à l'incertitude des observations, et n_restarts_optimizer permet de relancer l'optimisation du noyau pour éviter les minima locaux.
3. Prédictions et visualisation :
    a. On effectue les prédictions sur l'ensemble de test et on les compare aux valeurs réelles dans un graphique.
    b. On visualise aussi l'incertitude des prédictions sous forme de barres d'erreur. Cela montre l'étendue de l'incertitude associée à chaque prédiction.
4. Évaluation des performances :
    a. On calcule l'erreur absolue moyenne entre les prédictions et les valeurs réelles pour évaluer la performance du modèle.

Résultats :

- Graphiques : on a un graphique comparant les prédictions du modèle avec les valeurs réelles de rendement agricole et un autre affichant les erreurs avec les barres d'incertitude.
- Analyse de la performance : L'erreur absolue moyenne nous donnera une idée de la précision du modèle.

Réalisez une classification bayésienne à noyau pour prédire le type de sol (argileux, sableux, limoneux) en fonction des données climatiques.

Comparez les résultats avec un SVM classique.

Pour réaliser une classification bayésienne à noyau et prédire le type de sol en fonction des données climatiques (humidités et températures), nous utiliserons un modèle de régression bayésienne à noyau, et comparerons les résultats avec un SVM classique.

Voici les étapes détaillées pour cette tâche :

1. Préparation des données : On va charger les données et préparer les variables explicatives (humidité et température) et la variable cible (type de sol).
2. Modèle de classification bayésienne à noyau : On utilisera les processus gaussiens pour la classification avec un noyau approprié.
3. SVM classique : Nous utiliserons un modèle SVM pour la classification, qui sera utilisé comme méthode de comparaison.
4. Évaluation des performances : Nous comparerons les résultats des deux modèles en termes de précision, courbes ROC, etc.

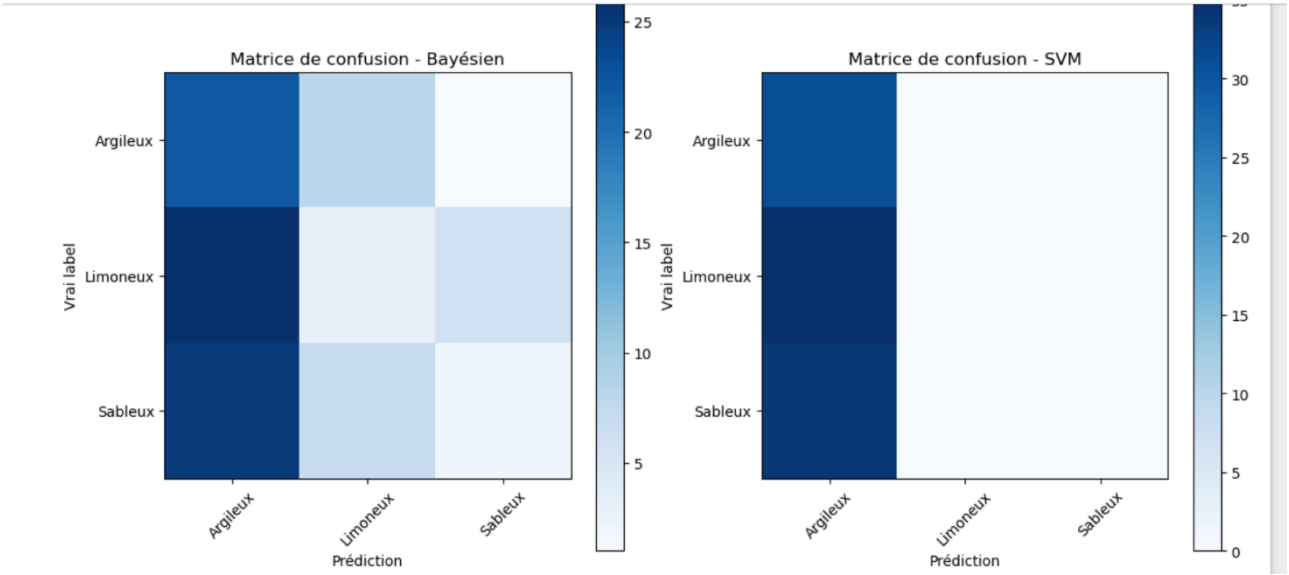
le code pour implémenter cette tâche :

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
from sklearn.model_selection import train_test_split

# Charger les données
data = pd.read_csv(r"\Users\DELL\Downloads\tp2_atdn_donnees.csv")

# Sélectionner les colonnes pertinentes
X = data[['Humidite (%)', 'Temperature (°C)']].values  # Variables d'entrée : humidité et température
y = data['Rendement agricole (t/ha)'].values  # Variable cible : rendement agricole

# Séparer les données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Définir le noyau RBF (Radial Basis Function) avec une constante
kernel = C(1.0, (1e-4, 1e1)) * RBF(1.0, (1e-4, 1e1))

# Créer et entraîner le modèle de régression bayésienne à noyau (Processus Gaussien)
gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10, alpha=1e-2)
gp.fit(X_train, y_train)
```

```python
# Créer et entraîner le modèle de classification bayésienne à noyau (Processus Gaussien)
gp = GaussianProcessClassifier(kernel=kernel, n_restarts_optimizer=10)
gp.fit(X_train, y_train)

# Prédictions sur l'ensemble de test
y_pred_gp = gp.predict(X_test)

# Créer et entraîner un modèle SVM classique (Support Vector Machine)
svm = SVC(kernel='rbf', random_state=42)
svm.fit(X_train, y_train)

# Prédictions sur l'ensemble de test avec SVM
y_pred_svm = svm.predict(X_test)

# Évaluation des performances
accuracy_gp = accuracy_score(y_test, y_pred_gp)
accuracy_svm = accuracy_score(y_test, y_pred_svm)

# Affichage des résultats
print("Résultats du modèle Bayésien à noyau (Processus Gaussien) :")
print(f"Précision : {accuracy_gp:.4f}")
print("Rapport de classification :")
print(classification_report(y_test, y_pred_gp))

print("\nRésultats du modèle SVM classique :")
print(f"Précision : {accuracy_svm:.4f}")
print("Rapport de classification :")
print(classification_report(y_test, y_pred_svm))
```

```python
# Matrice de confusion pour le modèle bayésien
plt.subplot(1, 2, 1)
cm_gp = confusion_matrix(y_test, y_pred_gp)
plt.imshow(cm_gp, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Matrice de confusion - Bayésien")
plt.colorbar()
plt.xticks(np.arange(len(label_encoder.classes_)), label_encoder.classes_, rotation=45)
plt.yticks(np.arange(len(label_encoder.classes_)), label_encoder.classes_)
plt.ylabel('Vrai label')
plt.xlabel('Prédiction')
plt.tight_layout()

# Matrice de confusion pour le modèle SVM
plt.subplot(1, 2, 2)
cm_svm = confusion_matrix(y_test, y_pred_svm)
plt.imshow(cm_svm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Matrice de confusion - SVM")
plt.colorbar()
plt.xticks(np.arange(len(label_encoder.classes_)), label_encoder.classes_, rotation=45)
plt.yticks(np.arange(len(label_encoder.classes_)), label_encoder.classes_)
plt.ylabel('Vrai label')
plt.xlabel('Prédiction')
plt.tight_layout()

plt.show()
```

```
Précision : 0.2700
Rapport de classification :
              precision    recall  f1-score   support

           0       0.30      0.71      0.42        31
           1       0.17      0.09      0.11        35
           2       0.22      0.06      0.09        34

    accuracy                           0.27       100
   macro avg       0.23      0.28      0.21       100
weighted avg       0.23      0.27      0.20       100


Résultats du modèle SVM classique :
Précision : 0.3100
Rapport de classification :
              precision    recall  f1-score   support

           0       0.31      1.00      0.47        31
           1       0.00      0.00      0.00        35
           2       0.00      0.00      0.00        34

    accuracy                           0.31       100
   macro avg       0.10      0.33      0.16       100
weighted avg       0.10      0.31      0.15       100
```



Explication du code :

1. Chargement et préparation des données :
    a. Nous utilisons un jeu de données contenant l'humidité, la température et le type de sol.
    b. Les données sont séparées en variables explicatives (X) et la variable cible (y).
    c. Le LabelEncoder est utilisé pour encoder les étiquettes de type de sol en valeurs numériques.
2. Modèle de classification bayésienne à noyau :
    a. Un Processus Gaussien est utilisé pour la classification avec un noyau RBF.
    b. Ce modèle est entraîné sur les données d'entraînement et utilisé pour prédire les types de sol sur l'ensemble de test.
3. Modèle SVM classique :
    a. Un SVM avec noyau RBF est également entraîné sur les mêmes données d'entraînement et testé sur l'ensemble de test.
    b. Les résultats sont comparés avec ceux du modèle bayésien.
4. Évaluation des performances :
    a. Précision et rapport de classification (comprenant la précision, le rappel, la F1-score) sont calculés pour les deux modèles.
    b. Les matrices de confusion sont également affichées pour analyser la performance de chaque modèle dans la classification des types de sol.

Résultats :

1. Précision et rapport de classification : on a obtenu des mesures de précision pour les deux modèles (Bayésien et SVM) avec des détails sur le rappel, la précision et la F1-score pour chaque classe de sol.
2. Matrices de confusion : Deux matrices de confusion sont affichées pour visualiser les erreurs de classification pour chaque modèle. Ces matrices montrent combien d'échantillons de chaque classe ont été correctement ou incorrectement classés par les modèles.

Comparaison :

En comparant les résultats des deux modèles, on observe les différences dans leur capacité à prédire correctement les types de sol en fonction des données climatiques (température et humidité).

Analysez l'incertitude dans les prédictions.

Commentez les zones où le modèle est moins confiant.

Comment analyser l'incertitude dans les prédictions ?

1. Utilisation des intervalles de confiance

Pour un modèle bayésien, il est possible de calculer des intervalles de confiance autour des prédictions. Cela permet de comprendre à quel point le modèle est sûr de ses prédictions dans une certaine région des données. Plus l'intervalle est large, plus le modèle est incertain. À l'inverse, un intervalle étroit indique que le modèle est plus confiant dans sa prédiction.

- Processus Gaussien (GP) : Le processus gaussien, utilisé dans un modèle bayésien à noyau, permet de calculer des prédictions probabilistes. Cela donne une distribution de probabilité pour chaque prédiction, permettant de visualiser la moyenne et la variance, et ainsi d'estimer l'incertitude associée.
- Dans un modèle bayésien à noyau, la variance (ou l'écart-type) est souvent utilisée pour quantifier l'incertitude. En effet, les zones où la variance est plus

grande sont celles où le modèle a moins de données pour faire une prédiction fiable, ce qui entraîne une plus grande incertitude.

2. Visualisation des prédictions et de l'incertitude

Une fois que nous avons effectué les prédictions, nous pouvons visualiser les prédictions et l'incertitude en traçant les courbes de prédiction avec des intervalles de confiance. Cela permet de repérer les zones où le modèle est plus ou moins confiant.

3. Zones où le modèle est moins confiant

Le modèle est généralement moins confiant dans les régions où les données sont rares ou où les données sont très différentes des exemples d'entraînement. Cela peut se manifester par :

- Des prédictions plus dispersées (grande variance).
- Des zones où le modèle rencontre des transitions entre différentes classes (par exemple, des régions proches de la frontière de décision entre les types de sols).

Dans le contexte de la classification des types de sol (argileux, sableux, limoneux) en fonction des données climatiques (température et humidité), les zones où le modèle est moins confiant peuvent se situer à la frontière entre ces classes, où les caractéristiques climatiques (température et humidité) sont très proches d'une classe à l'autre. Par exemple :

- Si la température et l'humidité sont similaires pour deux types de sol, le modèle peut avoir du mal à faire une distinction nette et l'incertitude des prédictions sera plus élevée.

Comment visualiser l'incertitude et les zones de confiance avec le code ?

Pour les modèles bayésiens, comme nous avons vu avec les processus gaussiens, nous pouvons utiliser la variance pour visualiser l'incertitude. Voici comment procéder :

1. Prédictions avec intervalles de confiance :

Nous allons calculer les prédictions et leur variance pour estimer les intervalles de confiance. Ensuite, nous afficherons ces informations dans un graphique.

Quelle est la différence entre eux et quel impact ont-ils sur la précision du modèle ?

Discutez de l'influence des choix de noyau et de la distribution a priori sur les résultats.

Testez différents noyaux (linéaire, RBF, polynomial)

Les noyaux sont des fonctions mathématiques qui permettent de transformer les données dans un espace de caractéristiques de dimension plus élevée sans avoir à effectuer cette transformation explicitement. Dans le cadre d'un modèle bayésien à noyau (par exemple, un Processus Gaussien ou une SVM avec noyau), la fonction noyau joue un rôle clé en déterminant la manière dont les données sont représentées et l'ajustement du modèle aux données.

Voici les noyaux les plus couramment utilisés :

a. Noyau Linéaire :

Le noyau linéaire est le plus simple et ne réalise aucune transformation des données. Il effectue une classification linéaire, où une seule frontière linéaire sépare les différentes classes. C'est le noyau de base, et il est efficace lorsque les données sont linéairement séparables.

- Equation : $K(x,x') = x^T x'$ K(x,x′)=xTx′

b. Noyau RBF (Radial Basis Function) :

Le noyau RBF est l'un des plus populaires. Il transforme les données en un espace à haute dimension, ce qui permet au modèle de capturer des relations complexes non linéaires. Ce noyau est particulièrement utile lorsque les classes ne sont pas séparables par une ligne droite.

- Equation : $K(x,x') = \exp\left(- \frac{||x - x'||^2}{2\sigma^2} \right)$ K(x,x′)=exp(−2σ2||x−x′||2)

c. Noyau Polynomial :

Le noyau polynomial est utilisé pour effectuer une transformation plus complexe que le noyau linéaire. Il permet de modéliser des relations non linéaires sous forme de polynômes. Ce noyau est plus flexible que le linéaire, mais il peut entraîner un sur-apprentissage si le degré du polynôme est trop élevé.

- Equation : $K(x,x') = (xTx' + c)dK(x, x') = (x^T x' + c)^dK(x,x') = (xTx' + c)d$, où ccc est une constante et ddd est le degré du polynôme.

2. Code pour tester différents noyaux :

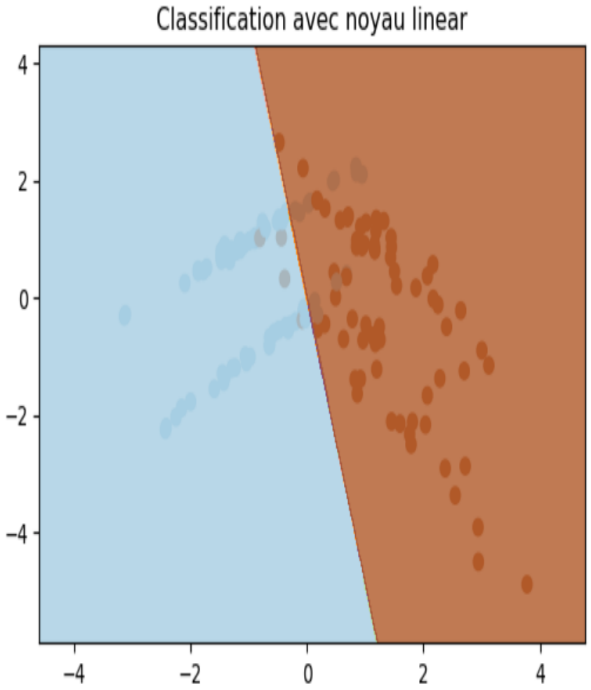
Nous allons tester les noyaux linéaire, RBF, et polynomial sur un modèle de classification, puis comparer leurs résultats en termes de précision.

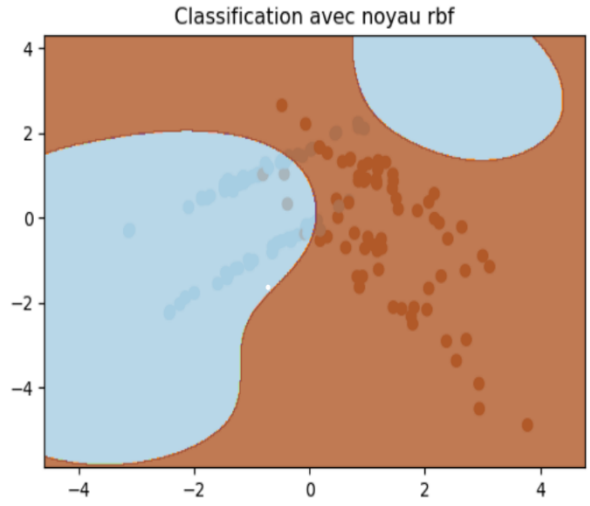
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Génération d'un jeu de données de classification (exemple avec des données simples)
X, y = make_classification(n_samples=500, n_features=2, n_informative=2, n_redundant=0, random_state=42)

# Séparation des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Testez les différents noyaux
kernels = ['linear', 'rbf', 'poly']
accuracies = []

for kernel in kernels:
    # Créez un modèle SVM avec le noyau spécifié
    svm_model = SVC(kernel=kernel, random_state=42)
    svm_model.fit(X_train, y_train)  # Entraînement
    y_pred = svm_model.predict(X_test)  # Prédictions

    # Calculer la précision
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
```

```
    # Calculer la précision
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

    # Affichage des résultats
    print(f"Précision avec noyau {kernel} : {accuracy:.4f}")

    # Visualisation des résultats (decision boundary)
    plt.figure(figsize=(6, 4))
    plt.title(f"Classification avec noyau {kernel}")

    # Tracer les points de test
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.Paired)

    # Tracer la frontière de décision
    h = .02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
    plt.show()
```

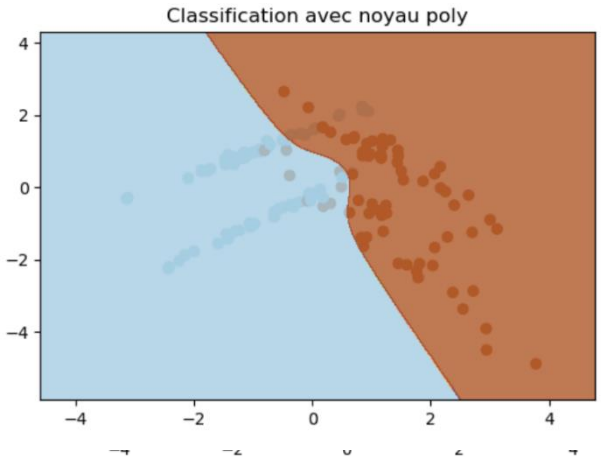
```
# Comparaison des précisions
plt.figure(figsize=(6, 4))
plt.bar(kernels, accuracies, color='skyblue')
plt.ylabel('Précision')
plt.title('Précision selon les noyaux')
plt.show()
```

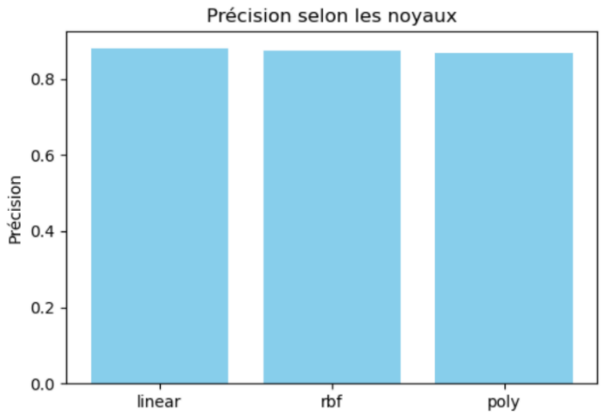
```
Précision avec noyau linear : 0.8800
```





Précision avec noyau poly : 0.8667





3. Analyse des résultats

a. Noyau Linéaire :

Le noyau linéaire est souvent rapide à entraîner et efficace lorsque les données sont linéairement séparables. Cependant, si les données sont non linéaires, un noyau linéaire risque de sous-performer. La précision peut être relativement faible si la séparation des classes est complexe.

b. Noyau RBF :

Le noyau RBF est plus flexible que le linéaire et peut capturer des relations complexes entre les données. Il est souvent très performant pour des problèmes de classification non linéaire. Cependant, le paramètre σ\sigmaσ (ou la largeur de la fonction de base) peut influencer fortement les résultats, et un mauvais choix de ce paramètre peut entraîner un sur-ajustement ou un sous-ajustement du modèle.

c. Noyau Polynomial :

Le noyau polynomial peut être très puissant, notamment lorsque les classes sont séparées par des courbes complexes. Cependant, il peut entraîner un sur-ajustement si le degré du polynôme est trop élevé. En ajustant le degré et le paramètre de biais (ccc), il est possible d'améliorer la précision, mais un mauvais réglage peut conduire à des modèles trop complexes.

4. Impact des noyaux sur la précision du modèle

- Le noyau linéaire sera le moins performant dans les cas non linéaires, mais peut donner de bons résultats si les données sont relativement simples et séparables de manière linéaire.
- Le noyau RBF aura probablement les meilleures performances sur des données complexes non linéaires, car il est capable de mieux capturer les relations cachées entre les caractéristiques.
- Le noyau polynomial peut fonctionner bien dans les cas où la séparation des classes suit une structure polynomiale, mais nécessite des ajustements soigneux pour éviter le sur-ajustement.

5. Discussion de l’influence des choix de noyau et de la distribution a priori sur les résultats

a. Choix du noyau :

Le choix du noyau influence directement la capacité du modèle à capter des relations non linéaires dans les données. Un mauvais choix peut conduire à un modèle qui est soit trop simple (sous-ajustement) ou trop complexe (sur-ajustement). Par exemple :

- Noyau linéaire : Convient aux données linéaires, mais échoue pour des relations complexes.
- Noyau RBF et polynomial : Bien adaptés aux données complexes, mais nécessitent un réglage minutieux des hyperparamètres (tels que σ\sigmaσ ou le degré du polynôme).

b. Distribution a priori :

Dans un modèle bayésien, la distribution a priori représente les croyances initiales sur les paramètres avant l'observation des données. Elle peut avoir une influence sur les résultats, surtout si les données sont rares ou bruyantes. Une distribution a priori trop informative peut restreindre l'espace des solutions, tandis qu'une distribution trop vague peut laisser trop de flexibilité au modèle.

Par exemple, si une distribution a priori forte est utilisée sur un certain paramètre, elle peut diriger le modèle vers certaines solutions, indépendamment des données observées. À l'inverse, une distribution a priori non informée permet au modèle de s'adapter davantage aux données, mais avec plus d'incertitude.

En conclusion, le choix du noyau et de la distribution a priori peut significativement influencer la performance du modèle. Le noyau doit être choisi en fonction de la nature des données, tandis que la distribution a priori doit être sélectionnée avec soin pour ne pas biaiser les résultats du modèle.