

Démonstration KNN 2 paramètres en utilisant la base IRIS

Importation des Bibliothèques:

```
In [21]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_openml
from sklearn.metrics import accuracy_score, classification_report
```

1 Charger le dataset MNIST

```
In [22]: mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data.astype(np.float32), mnist.target.astype(int)
```

2 Normalisation des données (important pour PCA & SVM)

```
In [23]: X /= 255.0
```

3 Réduire la dimension avec PCA

```
In [24]: pca = PCA(n_components=50) # Réduire à 50 dimensions (optimisé pour classificat
X_pca = pca.fit_transform(X)
```

4 Séparer en données d'entraînement et de test

```
In [25]: X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, ran
```

5 Entraîner un classifieur SVM

```
In [27]: svm_model = SVC(kernel='rbf', C=1) # RBF est souvent plus performant sur MNIST
svm_model.fit(X_train, y_train)
```

Out[27]:

```
SVC
SVC(C=1)
```

6 Prédictions et évaluation

```
In [28]: y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy (SVM après PCA) C=1, n_components=50: {accuracy * 100:.2f}")
print(classification_report(y_test, y_pred))
```

Test Accuracy (SVM après PCA) C=1, n_components=50: 98.07%

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1343
1	0.99	0.99	0.99	1600
2	0.97	0.99	0.98	1380
3	0.98	0.97	0.97	1433
4	0.98	0.98	0.98	1295
5	0.98	0.97	0.98	1273
6	0.99	0.99	0.99	1396
7	0.98	0.98	0.98	1503
8	0.98	0.97	0.98	1357
9	0.98	0.97	0.97	1420
accuracy			0.98	14000
macro avg	0.98	0.98	0.98	14000
weighted avg	0.98	0.98	0.98	14000

Test Accuracy (SVM après PCA) C=0.1, n_components=50: 98.07%

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1343
1	0.99	0.99	0.99	1600
2	0.97	0.99	0.98	1380
3	0.98	0.97	0.97	1433
4	0.98	0.98	0.98	1295
5	0.98	0.97	0.98	1273
6	0.99	0.99	0.99	1396
7	0.98	0.98	0.98	1503
8	0.98	0.97	0.98	1357
9	0.98	0.97	0.97	1420
accuracy			0.98	14000
macro avg	0.98	0.98	0.98	14000
weighted avg	0.98	0.98	0.98	14000

Test Accuracy (SVM après PCA) C=0.1, n_components=10: 91.55%					
	precision	recall	f1-score	support	
0	0.94	0.94	0.94	1343	
1	0.96	0.98	0.97	1600	
2	0.93	0.92	0.93	1380	
3	0.90	0.88	0.89	1433	
4	0.87	0.90	0.88	1295	
5	0.90	0.91	0.90	1273	
6	0.94	0.96	0.95	1396	
7	0.95	0.93	0.94	1503	
8	0.90	0.87	0.89	1357	
9	0.85	0.85	0.85	1420	
accuracy			0.92	14000	
macro avg	0.91	0.91	0.91	14000	
weighted avg	0.92	0.92	0.92	14000	

7 Visualisation en 2D si PCA réduit à 2 composants

```
In [29]: pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X)

plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca_2d[:, 0], y=X_pca_2d[:, 1], hue=y, palette="tab10", alpha=0.5)
plt.xlabel("Composante Principale 1")
plt.ylabel("Composante Principale 2")
plt.title("Visualisation des chiffres MNIST en 2D après PCA (50 composantes, C=1)")
plt.legend(title="Chiffres")
plt.show()
```





