TP 3 : Support Vector Machine (SVM)

Hachem Reda Riwa El Mazzouji Wahel

Introduction

Les machines à vecteurs de support (SVM) constituent une méthode de classification supervisée largement utilisée pour leur capacité à séparer efficacement des classes, même dans des espaces de grande dimension. Leur principe repose sur la recherche d'un hyperplan séparateur maximisant la marge entre les classes, avec la possibilité d'utiliser des fonctions noyau afin de gérer des données non linéairement séparables.

Le but de ce TP est de mettre en pratique les SVM sur des données simulées et des jeux de données réels, à l'aide du package scikit-learn. Nous cherchons notamment à comprendre comment contrôler les paramètres influençant leur flexibilité, tels que les hyperparamètres et le choix du noyau.

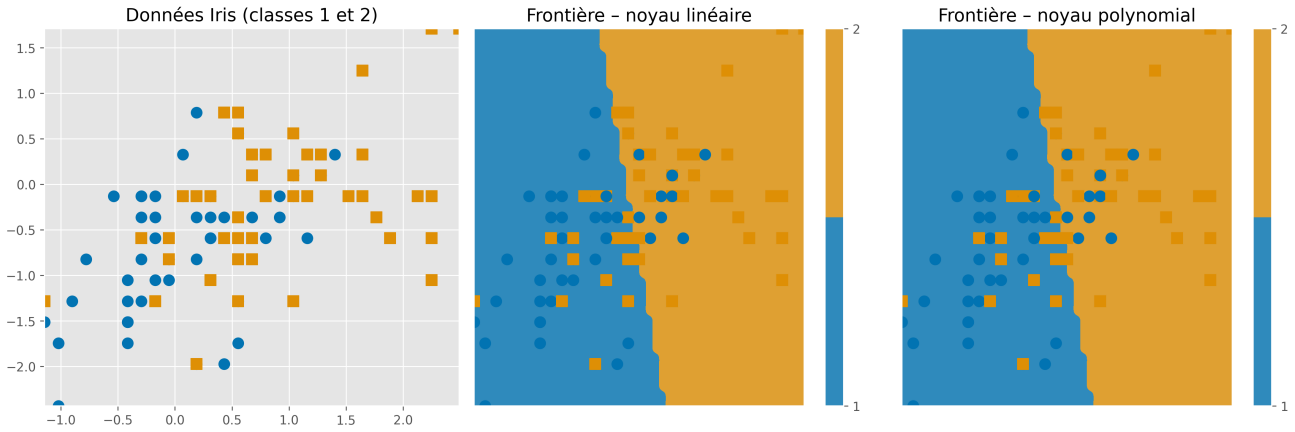
Question 1

```
Meilleur paramètre C : 8.310 (kernel = linéaire)
Score entraînement : 0.733
Score test : 0.72
```

Avec un SVM à noyau linéaire entraîné sur les classes 1 et 2 du jeu de données Iris, le meilleur paramètre trouvé est C 8.31. Le modèle atteint un score d'entraînement de 73% et un score de test de 72%.

Question 2

```
Meilleurs paramètres (poly) : C = 1.000, degré = 1, gamma = 10.0
Score entraînement : 0.720
Score test : 0.720
```



Un SVM avec noyau polynomial a été testé en faisant varier les paramètres C, gamma et le degré du polynôme.

Le meilleur modèle polynomial obtenu correspond en réalité à un polynôme de degré 1, ce qui revient à un noyau linéaire.

On observe que les scores sont identiques (72%) et en particulier très proches de ceux obtenus avec le SVM linéaire.

La frontière de décision est donc la même qu'avec le noyau linéaire, ce qui montre que, dans ce cas, l'utilisation d'un noyau polynomial n'apporte pas d'amélioration.

SVM GUI

Question 3

Dans cette partie nous avons utilisé le script `svm_gui.py`, pour étudier le paramètre de régularisation C.

On observe que lorsque C est grand, l'algorithme cherche à classer correctement tous les points, quitte à réduire la marge et à ajuster fortement la frontière de décision.

Lorsque C est petit, les erreurs sont davantage tolérées, la marge s'élargit et la frontière se déplace au profit de la classe majoritaire.

Nous avons généré un jeu de données fortement déséquilibré, composé de 90 points appartenant à une classe et seulement 10 à l'autre. Nous avons ensuite étudié l'effet du paramètre C avec un noyau linéaire.

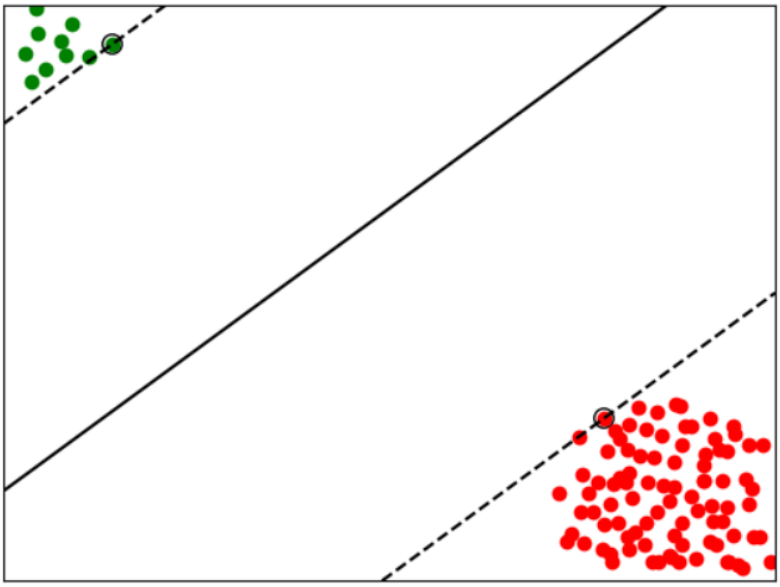


Figure 1 – Visualisation de la frontière de décision pour $C = 1$

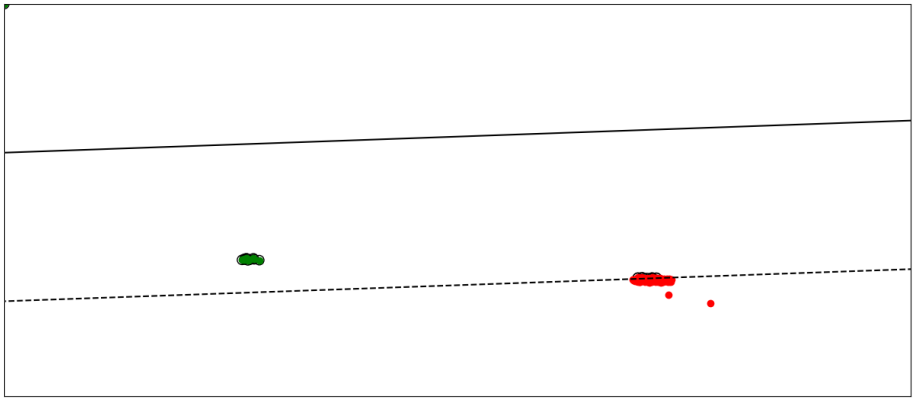


Figure 2 – Visualisation de la frontière de décision pour $C = 0.01$

On observe qu'avec C = 1, le SVM maintient une séparation correcte entre les deux classes, y compris pour la minorité, tout en élargissant légèrement la marge.

En revanche, avec C = 0.01, certains points verts franchissent l'hyperplan et sont mal classés, la frontière devient moins stricte et privilégie clairement la classe majoritaire.

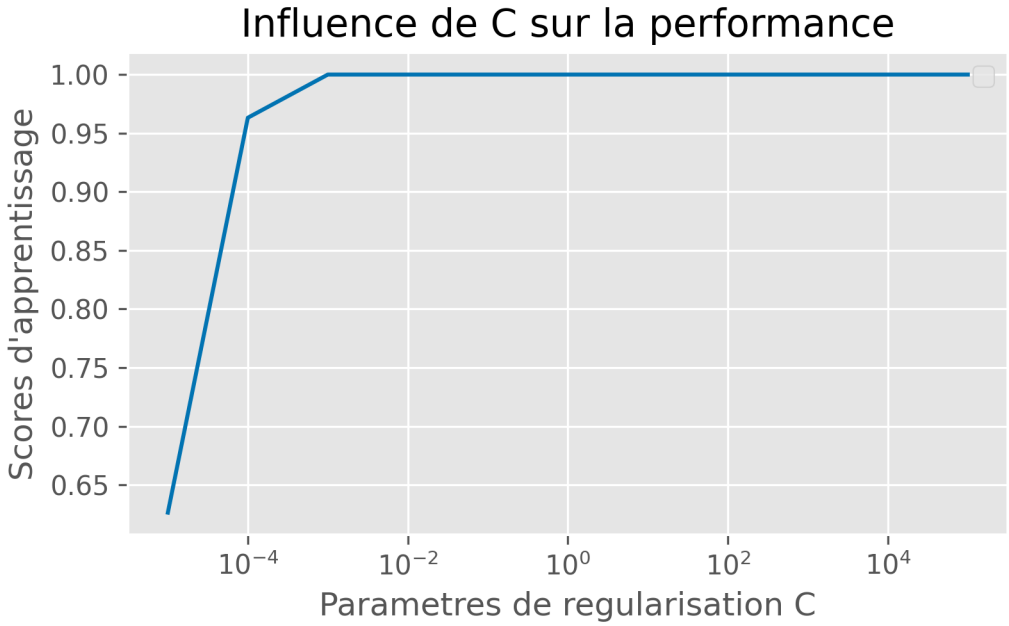
Ainsi, la diminution de C élargit les marges et accentue le biais en faveur de la classe majoritaire, car le modèle tolère davantage d'erreurs sur la minorité.

Classification de visages

Dans cette section, nous appliquons les SVM à un problème de reconnaissance faciale à partir de la base Labeled Faces in the Wild. L'objectif est de distinguer deux individus en extrayant des caractéristiques des images et en entraînant un classifieur SVM.

Question 4

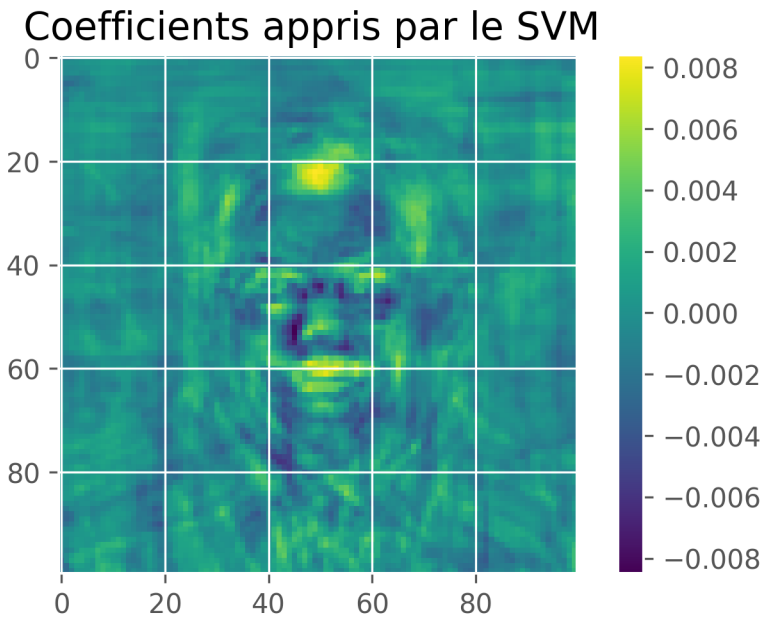
```
=== SVM linéaire : influence de C ===
Meilleur C : 1.0e-03
Score test optimal : 1.000
```




```
Prédiction des noms des personnes sur l'échantillon de test
Réalisé en 0.329s
Niveau de hasard (majorité) : 0.621
Taux de précision final sur le test : 0.884
```

Lorsque C est très petit, le modèle est trop régularisé et n'arrive pas à bien séparer les classes (sous-apprentissage). Autour de C 1.0×10^{-3} , la performance devient optimale et se stabilise. Pour des valeurs trop grandes de C, le modèle risque de sur-apprendre sans gain significatif. Le meilleur compromis est obtenu avec C 1.0×10^{-3}, qui donne un taux de précision de 91% sur le test, bien supérieur au hasard (62%).

La prédiction sur l'échantillon de test montre que le modèle arrive à bien distinguer les deux individus.



La visualisation des coefficients montre que le SVM capte les traits distinctifs des visages, confirmant l'efficacité de cette approche pour la classification faciale.

Question 5

```
Score sans variable de nuisance
[Données originales] Score entraînement : 1.000 | Score test : 0.911
Score avec variable de nuisance
[Données bruitées] Score entraînement : 1.000 | Score test : 0.516
```

Sans ajout de variables de nuisance, le SVM linéaire obtient un taux de précision parfait à l'entraînement (1.0) et une très bonne performance en test : 0.911 . En revanche, lorsque l'on ajoute 300 variables bruitées, la performance en test chute fortement (0.516), tandis que l'entraînement reste à 1.0.

Cela montre que l'ajout de dimensions non informatives dégrade la capacité de généralisation : le classifieur se retrouve à sur-apprendre sur des données bruitées, ce qui entraîne une baisse nette de la précision sur de nouvelles données.

Question 6

```
Score apres reduction de dimension
[] Score entraînement : 0.816 | Score test : 0.595
```

Avec la réduction de dimension par PCA, le modèle conserve un taux de précision parfait sur l'entraînement mais la performance en test reste faible (0.595 pour 200 composantes). Cela s'explique par le fait qu'un trop grand nombre de composantes conserve encore beaucoup de bruit.

Question 7

Dans le script fourni, la normalisation est effectuée avant la séparation entre apprentissage et test, comme l'illustre l'extrait de code ci-dessous.

```
X = (np.mean(images, axis=3)).reshape(n_samples, -1)

X -= np.mean(X, axis=0)
X /= np.std(X, axis=0)

indices = np.random.permutation(X.shape[0])
train_idx, test_idx = indices[:X.shape[0] // 2], indices[X.shape[0] // 2:]
X_train, X_test = X[train_idx, :], X[test_idx, :]
y_train, y_test = y[train_idx], y[test_idx]
images_train, images_test = images[
    train_idx, :, :, :], images[test_idx, :, :, :]
```

Cette approche engendre une fuite d'information, dans la mesure où les statistiques issues du jeu de test (moyenne et variance) interviennent dans la transformation des données d'apprentissage. Une telle erreur introduit un biais et conduit à une surestimation de la performance du modèle, puisque l'indépendance du jeu de test n'est plus respectée. La procédure correcte consiste à ajuster la normalisation uniquement sur l'échantillon d'apprentissage, puis à appliquer la transformation ainsi obtenue aux données de test, garantissant ainsi une évaluation fidèle de la capacité de généralisation du classifieur.