Apprentissage Statistique

TP : SVM

Kaoutar SARIH
Doha EL QEMMAH

Montpellier, le 3 octobre 2025

STATISTIQUE
SCIENCE DES DONNÉES MIND
UNIVERSITÉ DE MONTPELLIER

Contents

Objectif du TP

Ce TP a pour objectif de mettre en œuvre les Support Vector Machines (SVM) sur des jeux de données simulées (gaussiennes, Iris) et réelles (visages LFW). Les points étudiés sont :

- l'effet du noyau (linéaire, polynomial),
- l'influence du paramètre de régularisation C et du paramètre γ,
- l'impact des données déséquilibrées et des variables de nuisance,
- l'amélioration possible via la réduction de dimension par PCA.

Jeu de données simulé : deux gaussiennes

Nous commençons par générer un jeu de données simple composé de deux nuages gaussiens bidimensionnels. La figure ci-dessous illustre la distribution : chaque nuage correspond à une classe différente et est représenté par une couleur contrastée. On observe ainsi que l'ensemble est formé de deux sous-nuages distincts, issus de vecteurs gaussiens générés aléatoirement.

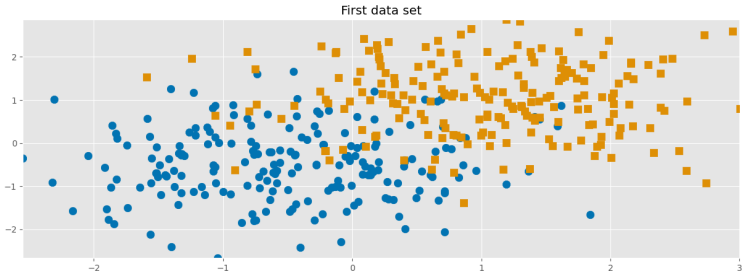


Figure 1: Premier jeu de données simulé : deux gaussiennes distinctes.

Nous discernons bien une séparation entre la majorité des points de chaque distribution gaussienne. Cependant, le choix d'un noyau linéaire implique qu'il reste des observations mal classées de part et d'autre de la frontière.

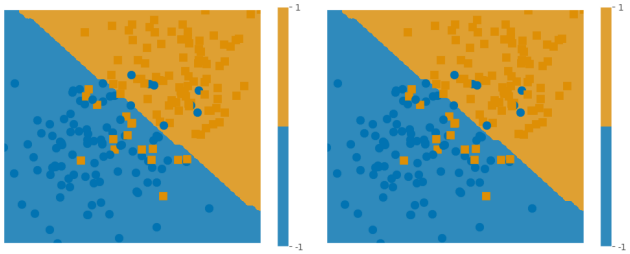


Figure 2: Frontière de décision d'un SVM à noyau linéaire sur les données gaussiennes.

Il apparaît que plusieurs valeurs du paramètre C donnent un score d'environ 0.875, ce qui semble correspondre à une valeur optimale. Néanmoins, le graphe obtenu n'est pas particulièrement plus convaincant que le précédent, ce qui suggère que le jeu de données choisi n'est pas idéal pour mettre en évidence les différences de performance entre paramètres et noyaux. Un jeu de données plus complexe permettrait sans doute une meilleure illustration.

1 Question 1 et 2 : Iris avec noyau linéaire et polynomial

Les étapes suivantes seront réalisées :

- Normalisation des données et sélection des deux classes.
- Découpage en apprentissage/test.
- Recherche du meilleur C avec GridSearchCV.
- Présenter les scores obtenus (train/test).

1.1 Chargement et normalisation des données

Nous utilisons le jeu de données `Iris` disponible dans la bibliothèque `scikit-learn`. Afin de simplifier le problème, nous nous limitons aux deux classes, et nous ne considérons que les deux premières caractéristiques (longueur et largeur du sépale).
L'étape de normalisation est indispensable en SVM et une variable ayant une échelle beaucoup plus grande que les autres aurait une influence disproportionnée sur la frontière de décision. Nous présentons ci-dessous un extrait des jeux de données d'apprentissage et de test utilisés pour l'entraînement du SVM.

Aperçu X_train:

	sepal length (cm)	sepal width (cm)
0	1.038005	0.098217
1	-0.416010	-1.052767
2	0.553333	-1.282963
3	-0.779513	-0.822570
4	2.249683	-0.592373

Figure 3: Aperçu du jeu de train X_{train}.

Aperçu X_test:

	sepal length (cm)	sepal width (cm)
0	0.553333	-0.592373
1	0.553333	-0.362176
2	1.280340	0.328414
3	-0.173674	-0.131979
4	-0.294842	-0.822570

Figure 4: Aperçu du jeu de test X_{test}.

1.2 Classification avec noyau linéaire

Les données sont séparées aléatoirement en deux sous-ensembles : un ensemble d'apprentissage (`train`) et un ensemble de test (`test`). Nous entraînons ensuite un SVM avec noyau linéaire, en optimisant le paramètre de régularisation C à l'aide d'une validation croisée.
Le meilleur paramètre trouvé est :

$$C = \texttt{0.8407}, \quad \text{noyau} = \text{linéaire}$$

Les scores de généralisation obtenus sont :

$$\text{Train} = 0.747 \qquad \text{Test} = 0.680$$

Il est logique que le score obtenu sur l'ensemble d'apprentissage soit plus élevé que celui mesuré sur l'ensemble de test. Cela traduit le fait que le modèle s'adapte davantage aux données d'entraînement, tout en conservant une capacité correcte de généralisation. Avec le noyau linéaire et un paramètre C optimal, la précision atteint environ 70%.

```
Generalization score for linear kernel: 0.7466666666666667, 0.68
```

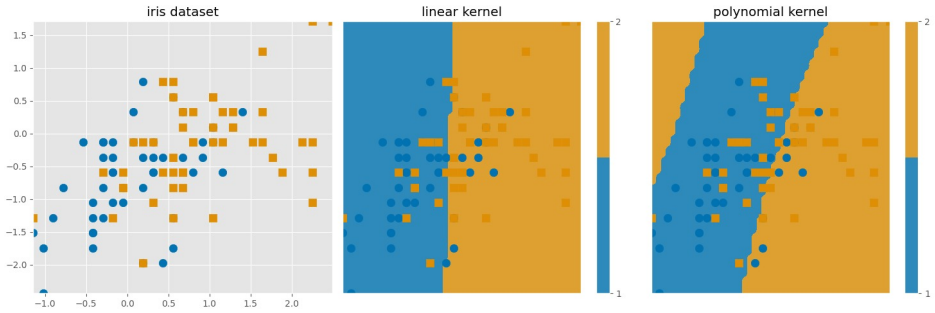
1.3 Classification avec noyau polynomial

Passons maintenant au noyau polynomial :
Pour le noyau polynomial, l'optimisation a sélectionné le degré 1 comme valeur optimale, ce qui conduit à un modèle identique au cas linéaire.

```
{'C': np.float64(0.3009), 'kernel': 'linear'}
Generalization score for polynomial kernel: 0.7466666666666667, 0.68
```

Le degré 1 ayant été sélectionné initialement (cas linéaire), nous avons volontairement exclu cette valeur et limité la recherche aux degrés 2 et 3 pour observer les performances du noyau polynomial dans un cadre non linéaire.

```
Generalization score for polynomial kernel: 0.6533333333333333, 0.52
```



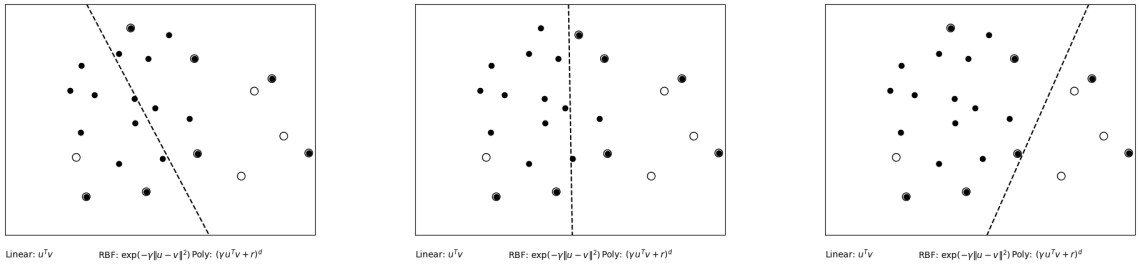
La représentation graphique montre que le noyau linéaire fournit une séparation simple et robuste, mais imparfaite en raison du chevauchement naturel des classes. Le noyau polynomial introduit une frontière plus flexible, qui peut mieux séparer certaines observations mais au prix d'une complexité accrue et d'un risque de surapprentissage. Ce comportement correspond exactement à ce qui est vu dans le cours : le choix du noyau doit refléter la nature de la distribution des données, en recherchant un compromis entre biais (modèle trop simple) et variance (modèle trop complexe).

2 Question 3 : SVM Gui

Nous avons suivi les indications du site scikit-learn svm_gui qui fournit le code source disponible dans `python_script/svm_gui.py`. Ce script est lancé en console.
Dans cette expérience, nous générons un jeu de données déséquilibré constitué de 20 points de la classe 1 (représentés en noir) et 4 points de la classe 2 (représentés en blanc). L'objectif est d'ajuster une frontière de séparation en testant différents noyaux et paramètres.
Nous explorons le noyau linéaire avec les paramètres :

- de régularisation C, qui détermine l'influence des erreurs de classification .
- γ, qui contrôle la distance d'influence d'un point donné .

Dans notre étude, nous avons principalement modifié le paramètre C afin d'observer son impact sur la frontière de décision et la précision du modèle.



(a) Noyau linéaire $C = 0.01$ (b) Noyau linéaire $C = 1$ (c) Noyau linéaire $C = 7$

Figure 5: Influence du paramètre de régularisation C sur la frontière de décision (noyau linéaire).

Interprétation. En observant les trois figures, on constate que la valeur de C influence directement la position et l'inclinaison de la frontière de séparation.

- Pour $C = 0.01$, la frontière est très approximative : elle ne cherche pas à bien séparer tous les points et tolère davantage d'erreurs de classification.
- Pour $C = 1$, la séparation devient plus équilibrée. La frontière s'ajuste mieux à la répartition des données.
- Pour $C = 7$, la frontière est beaucoup plus contrainte par les données. Le modèle cherche à corriger davantage de points mal classés.

En résumé, plus C est petit, plus le modèle tolère les erreurs individuelles; à l'inverse, plus C est grand, plus le modèle essaie de classer correctement chaque point.

3 Classification des visages :

3.1 Question 4:

Classification de visages avec noyau linéaire

Nous utilisons la base de données LFW (Labeled Faces in the Wild) en restreignant le problème à deux individus : *Tony Blair* et *Colin Powell*.
Les données sont centrées et réduites puis divisées en deux ensembles de taille égale : apprentissage (50%) et test (50%). Un SVM à noyau linéaire est entraîné en testant différents paramètres de régularisation C sur une échelle logarithmique entre 10^{-5} et 10^{5}.

Figure 6: Exemples d'images extraites du dataset LFW (Tony Blair).

Le graphique ci-dessous montre l'évolution des scores et de l'erreur de test en fonction de C:

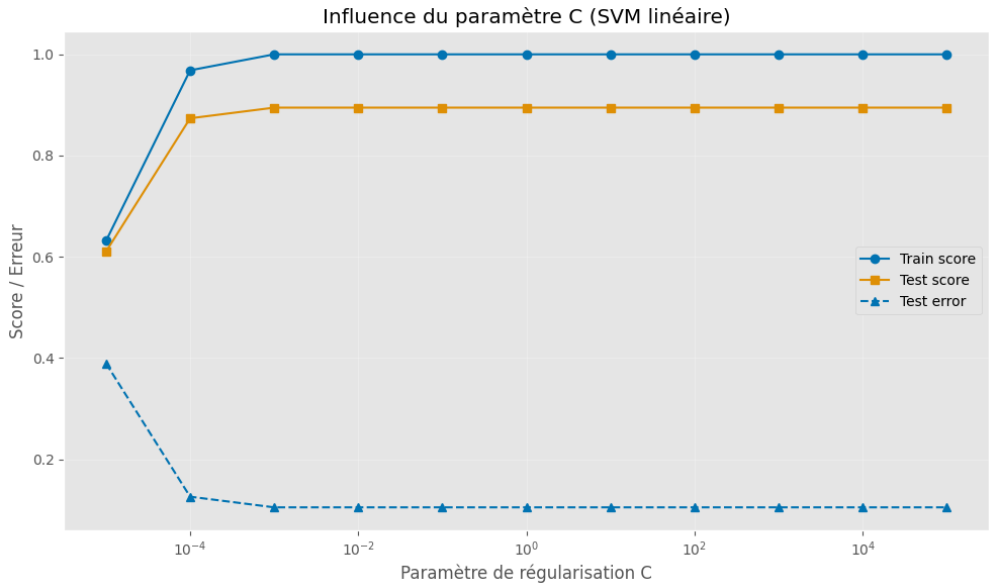


Figure 7: Influence du paramètre de régularisation C.

Le meilleur paramètre obtenu est $C = 0.001$. Avec ce choix, le modèle atteint une précision parfaite sur l'ensemble d'apprentissage (100%), et une précision de à peu près 91% sur l'ensemble de test. L'écart entre apprentissage et test montre un léger sur-apprentissage, mais les performances restent excellentes, nettement supérieures au niveau de hasard (62%).

En plus de l'évaluation quantitative, nous avons analysé les résultats de la classification sur l'ensemble de test. La figure suivante illustre quelques prédictions réalisées par le modèle, avec comparaison entre l'étiquette prédite et l'étiquette réelle voir figure 8.

Nous observons que la plupart des visages sont correctement classés, avec néanmoins quelques erreurs (3 mal prédits vs 9 correctement prédits).

Nous avons également visualisé la carte des coefficients du SVM linéaire entraîné avec le meilleur paramètre C voir figure 9.
Les zones les plus intenses dans la figure correspondent aux parties de l'image ayant le plus d'influence dans la décision. Cette représentation met en évidence que les zones centrales du visage (yeux, nez, bouche) contribuent le plus à la classification.

Figure 8: nom prédit vs nom réel

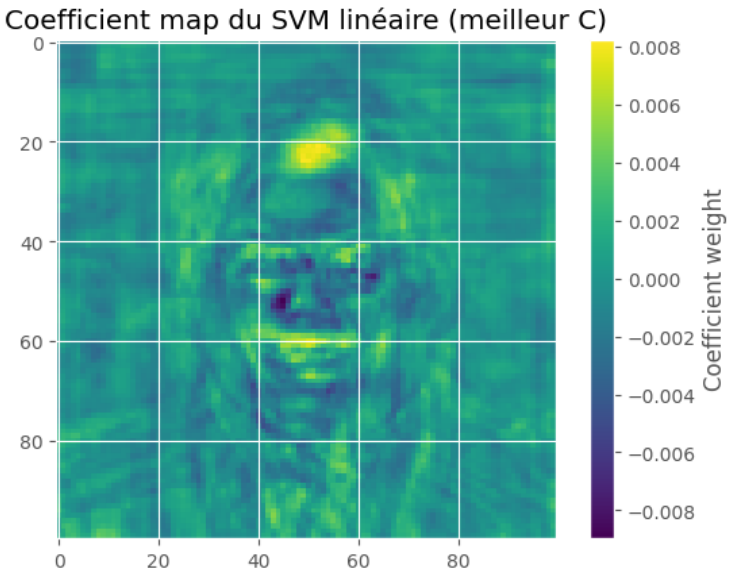


Figure 9: Carte des coefficients du SVM linéaire.

3.2 Question 5 :

Pour évaluer la robustesse du modèle, nous avons ajouté 300 variables de nuisance générées aléatoirement (bruit gaussien) aux caractéristiques originales. Le SVM linéaire est ensuite ré-entraîné et évalué.

- Sans variables de nuisance : *Score entraînement* = 1.0, *Score test* = 0.879
- Avec variables de nuisance : *Score entraînement* = 1.0, *Score test* = 0.584

Les résultats montrent que l'ajout de bruit n'affecte pas l'entraînement (le modèle parvient toujours à séparer les données d'apprentissage), mais entraîne une chute importante de la performance en test.
Ce phénomène s'explique par le surapprentissage : le modèle utilise les variables parasites pour améliorer artificiellement son score d'entraînement, mais ces dimensions ne contiennent aucune information utile pour la classification, ce qui dégrade fortement sa capacité de généralisation.

3.3 Question 6 : Amélioration par réduction de dimension (PCA)

Afin de limiter l'effet des variables de nuisance ajoutées dans la question précédente, nous appliquons une réduction de dimension par Analyse en Composantes Principales (PCA) en utilisant `svd_solver='randomized'`.
Avec $n\_components = 10$, les résultats obtenus sont :

$$\text{Score train} = 0.60, \quad \text{Score test} = 0.64$$

Comparé au cas précédent (sans PCA), où le score de test tombait à 0.58 malgré un score d'entraînement parfait, la PCA améliore la capacité de généralisation du modèle.
La réduction de dimension agit comme un filtre qui supprime le bruit et empêche le surapprentissage. Toutefois, un choix trop faible du nombre de composantes peut entraîner une perte d'information et donc un score global plus bas.
Nous avons toutefois rencontré un problème : le temps d'exécution de la PCA devient très important lorsque le nombre n augmente. Néanmoins, cela n'empêche pas de confirmer que la PCA joue bien son rôle de compromis entre réduction du bruit et conservation de l'information utile.

3.4 Question 7 : Biais dans le prétraitement des données

Dans notre pipeline initial dans le dataset d'images, nous avons normalisé les données (en centrant et réduisant chaque variable) avant la séparation entre apprentissage (train) et test. Ce choix introduit un biais, car les statistiques (moyenne et écart-type) sont calculées sur l'ensemble complet, incluant des informations provenant du jeu de test. Ce phénomène correspond à une *fuite d'information* : le modèle bénéficie indirectement d'informations sur le test, ce qui contredit la démarche que nous devons poursuir pour une réussite de prédiction et une dévaluation de ses performances. La bonne pratique consiste à ajuster la normalisation uniquement sur l'ensemble train, puis à appliquer cette transformation au jeu de test.

Annexe

Le code complet utilisé pour les expériences est disponible à l'adresse suivante : https://github.com/ksarih/TP_SVM.