STATISTIQUE
SCIENCE DES DONNÉES
UNIVERSITÉ DE MONTPELLIER

Master 2 Statistiques et Sciences des Données

Rapport de TP :

Support Vector Machines (SVM)

BOULAND Guillaume

Année 2024 - 2025

Table des matières

Le code complet est disponible sur notre Github dans le fichier `svm_script.py`, dans le dossier `src`. Il est important de ne pas oublier que les résultats que nous allons présenter peuvent dépendre d'une simulation à une autre. Nous avons fait des moyennes sur quelques valeurs pour avoir des résultats plus sûrs.

I Question 1

Nous allons reproduire une expérience similaire à ce que nous pouvons retrouver dans le fichier `svm_script.py` présent dans notre dépôt, en utilisant le jeu de données `iris` de `scikit-learn`. En effet, nous allons classifier la classe 1 contre la classe 2 en utilisant les deux premières variables du dataset (comme nous pouvons le voir sur la Figure 1) et un **noyau linéaire**.

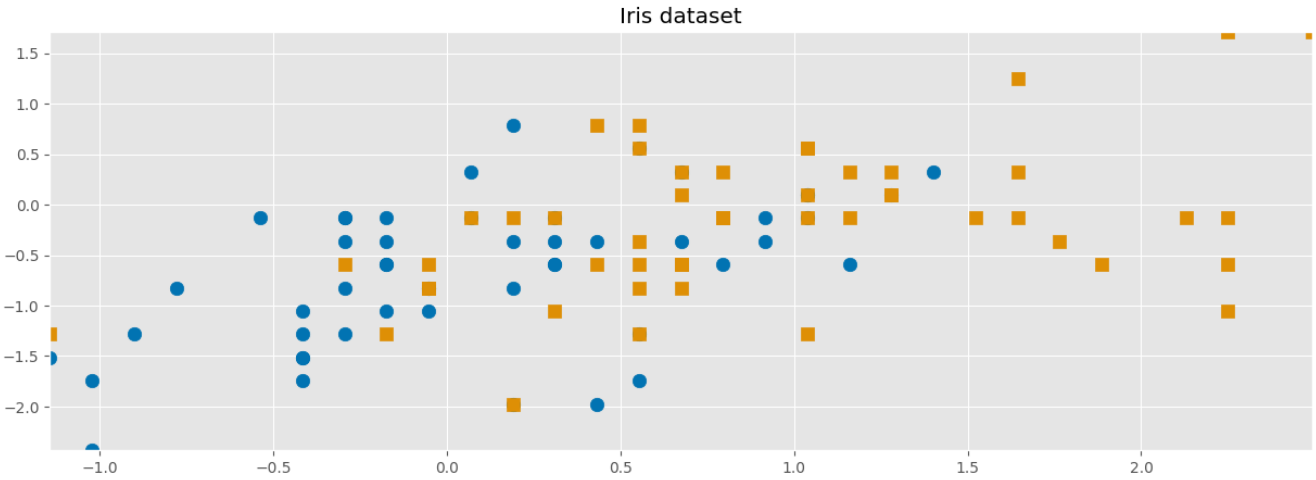


Figure 1: Représentation de l'échantillon du dataset iris retenu.

Nous allons à présent séparer notre jeu de données en deux afin de mettre des données de côté, caractérisant l'échantillon de test. Puis, en utilisant un noyau linéaire et à l'aide de la fonction `SVC` et `GridSearchCV` (pour le choix des paramètres par validation croisée) de `scikit-learn`. Nous obtenons une séparation que nous pouvons observer dans la Figure 2.

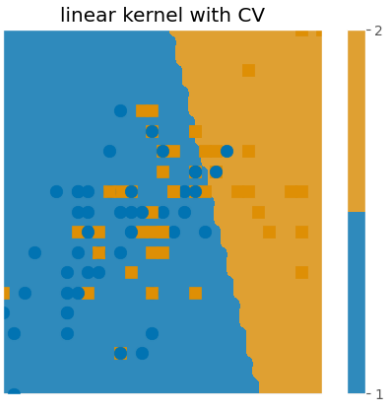


Figure 2: Représentation de la frontière du classifieur à noyau linéaire en utilisant la validation croisée

Nous calculons ensuite les scores de précision du modèle à noyau linéaire sur les données X et y, à la fois sur les données d'entraînement mais aussi et surtout sur les données de test. Il est à noter que ces scores peuvent dépendre d'une simulation à un autre.
En faisant une moyenne sur plusieurs simulations (30), nous obtenons un score sur les données d'apprentissage égal à 0.74 et 0.6 pour les données de test. Le score d'entraînement est plutôt bon, il représente la précision du modèle sur les données d'entraînement. Le score de test, ou score de généralisation, montre comment le modèle s'adapte à des données qu'il n'a pas vu auparavant (capacité de généralisation) . Les scores sont relativement proches, mais un score de test de 0.6 peut soulever quelques préoccupations, notamment le fait que les données ne sont pas

parfaitement linéaires, le modèle a du mal à séparer les données avec une simple droite affine. Cela implique que le modèle a besoins de certaines améliorations.

Nous faisons ensuite la même chose, mais cette fois sans utiliser la validation croisée, à l'aide de seulement la fonction `SVC` et un noyau linéaire. Nous obtenons un score moyen d'apprentissage égal à 0.74, et un score de test égal à 0.68. Nous observons la classification dans la Figure 3 suivante.

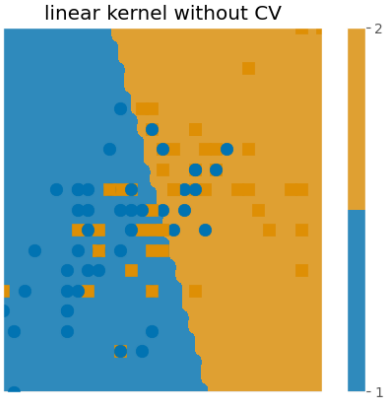
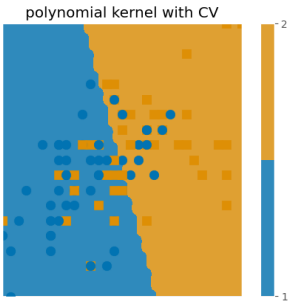


Figure 3: Représentation de la frontière du classifieur à noyau linéaire sans utiliser la validation croisée

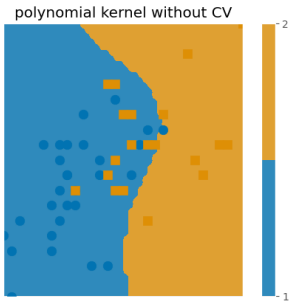
Il est à noter que sans validation croisée, le score de généralisation est en moyenne meilleur que lorsqu'on l'utilise. Le modèle a cependant toujours besoin d'améliorations si nous souhaitons avoir une généralisation satisfaisante et efficace.

II Question 2

Nous allons essayer de trouver une amélioration, notamment en changeant le noyau. En effet, nous allons utiliser un noyau polynomial, puis comparer les scores avec ceux obtenus dans la question 1.
Après avoir calculé les scores d'entraînement et de test en faisant une moyenne sur 10 simulations (moindre à cause du temps de calcul), nous obtenons un score d'entraînement égal à 0.74, ce qui n'a pas changé, et un score de généralisation égal à 0.68, qui a lui augmenté. Pour le cas sans validation croisée, nous obtenons un score d'apprentissage moyen égal à 0.64 et un score de généralisation égal à 0.62.



(a) Avec validation croisée



(b) Sans validation croisée

Figure 4: Représentation des deux frontières à noyau polynomial

Comme nous pouvons les voir dans les Figures 4a et 4b, quand on n'utilise pas la validation croisée, nous observons bien une frontière avec une allure polynomiale.

Le fait que le score de généralisation augmente en changeant le noyau (kernel) (seulement avec l'utilisation de la validation croisée) signifie que le noyau polynomial permet au modèle de s'adapter d'une meilleure manière aux données étrangères que le noyau linéaire le lui permet. Cette légère augmentation de score signifie que les données contiennent certaines relations non-linéaires, mais celle-ci reste relative et ces relations ne semblent pas être très fortes. Quand nous ne l'utilisons pas, nous nous rendons compte que le modèle a toujours besoin d'améliorations.

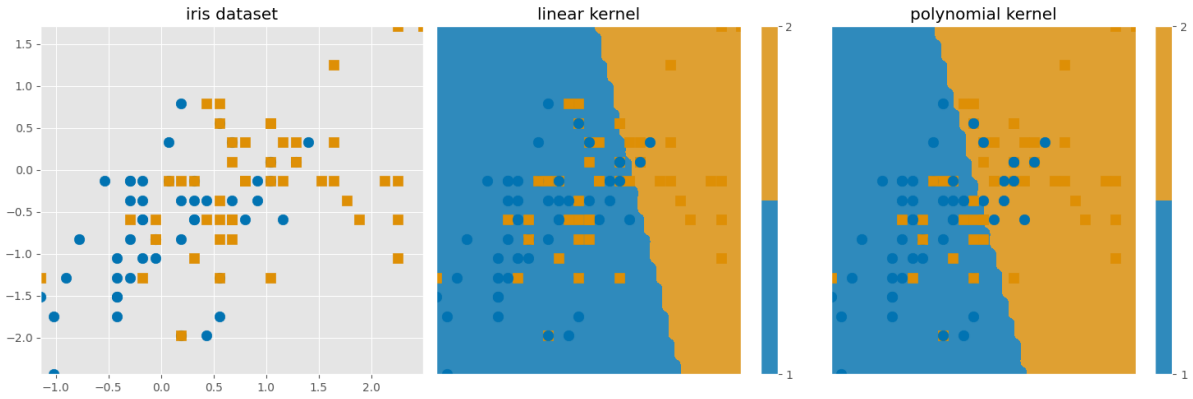


Figure 5: Comparaison des classifications selon les noyaux, avec validation croisée

Comme nous pouvons le voir sur la Figure 5, la différence de classification entre le noyau linéaire et polynomial n'est pas flagrante, tout comme les scores de généralisation. (Cela peut être dû au fait que pour le noyau polynomial, nous ajoutons des dimensions dans l'espace pour trouver un meilleur séparateur linéaire, puis pour des questions de visualisation, nous retournons en deux dimensions pour visualiser cette séparation.)

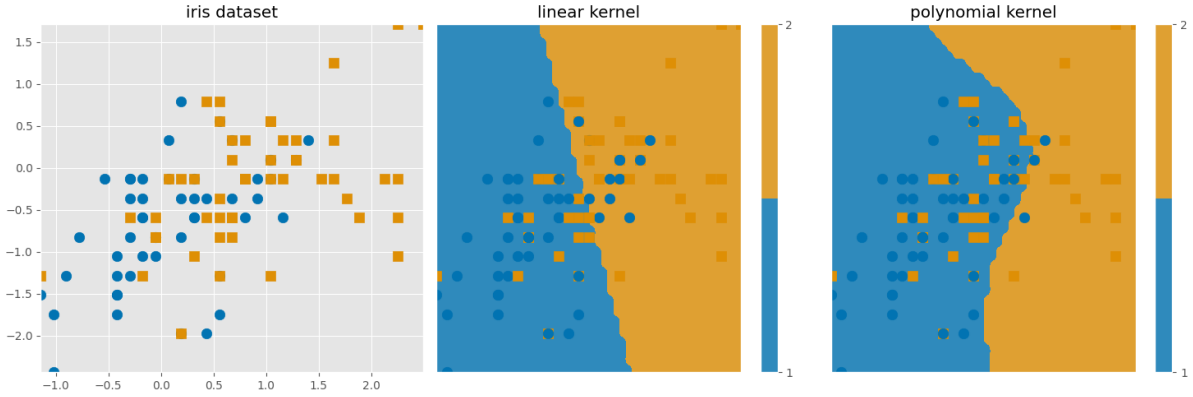
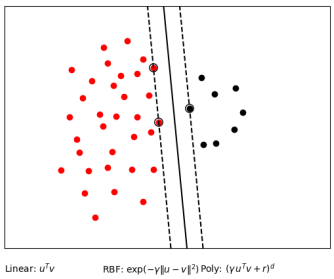


Figure 6: Comparaison des classifications selon les noyaux, sans validation croisée

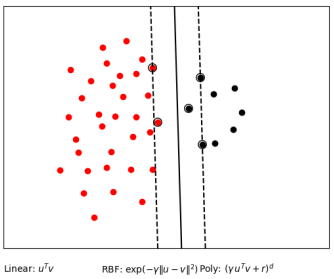
Nous pouvons bien observer, dans la Figure 6, la différence de classification entre un noyau linéaire et un noyau polynomial. Cela est dû au fait que lorsque l'on n'utilise pas la validation croisée, la fonction `SVC` choisit automatiquement un degré égal à 3 pour le polynôme caractérisant le noyau.

III Question 3

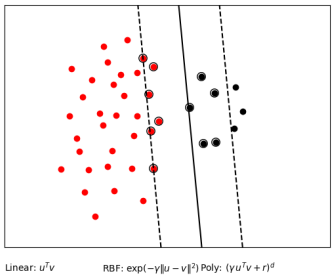
Nous allons utiliser une application interactive permettent de comprendre l'impact des différents paramètres des SVM. Pour cela, nous allons lancer le script `svm_gui.py`, que l'on peut retrouver à l'adresse suivante : `https://scikit-learn.org/1.2/auto_examples/applications/svm_gui.html`, ou alors directement sur notre page Github, dans le dossier `src`.
Nous plaçons des points en les répartissant de manière à ce que le proportion de points d'une classe soit très supérieure à celle de l'autre. En faisant varier le paramètre C, de régularisation, nous obtenons les représentations suivantes.



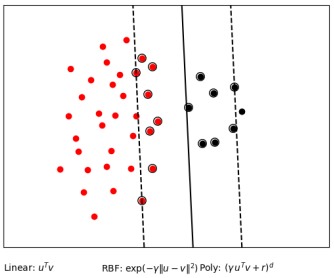
(a) Avec $C = 1$



(b) Avec $C = 0.01$



(c) Avec $C = 0.001$



(d) Avec $C = 0.0005$

Figure 7: Représentation des différents résultats de classification en faisant varier le paramètre de régularisation C

Comme nous pouvons le voir dans la Figure 7 ci-dessus, la variation de C influe sur la taille de la fenêtre de séparation des classes, de taille 2γ, où γ est la marge maximale. Nous pouvons donc observer, pour toutes les valeurs de C, les différents vecteurs supports. C étant le paramètre de régularisation de la marge souple, plus il est petit et plus le nombre de points entre les vecteurs supports ($\xi \in [0, 1]$, avec ξ une variable ressort), ou mal classés ($\xi > 1$, pas présent dans notre exemple), est important.
De plus, plus la valeur de C est faible, et plus le classifieur va dépendre de beaucoup de données, et donc le temps de calcul sera plus important, bien que les performances peuvent être améliorées.

Nous pouvons aussi remarquer que le classifieur ne prends pas en compte la proportion de données présentes dans chaque classe, omettant donc le poids de chaque nuage de points. Nous pouvons corriger ce phénomène en pondérant davantage les erreurs sur la classe la moins présente, ou alors par re-calibration.

IV Question 4

Nous allons à présent utiliser une banque d'images constituée de visages photographiés, nommée *"Labeled Faces in the Wild"* (`http://vis-www.cs.umass.edu/lfw/`).

Parmi toutes les personnes présentes dans cette base de données, nous nous restreignons à deux personnées (variables) : Tony Blair et Colin Powell. Nous pouvons le voir sur la Figure 8 suivante.

Figure 8: Exemple d'images présentes dans la base de données (Tony Blair)

Dans cette question, nous allons chercher à montrer l'influence de ce paramètre de régularisation C. Pour cela, nous n'allons pas utilisé la validation croisée comme dans les deux premières question, mais simplement la fonction `SVC` du module `scikit-learn.svm` (si nous l'avions utilisé, nous aurions eu une droite constante). En faisant varier ce paramètre, nous allons chercher à retenir la meilleur valeur afin de prédire au mieux sur les données de test.

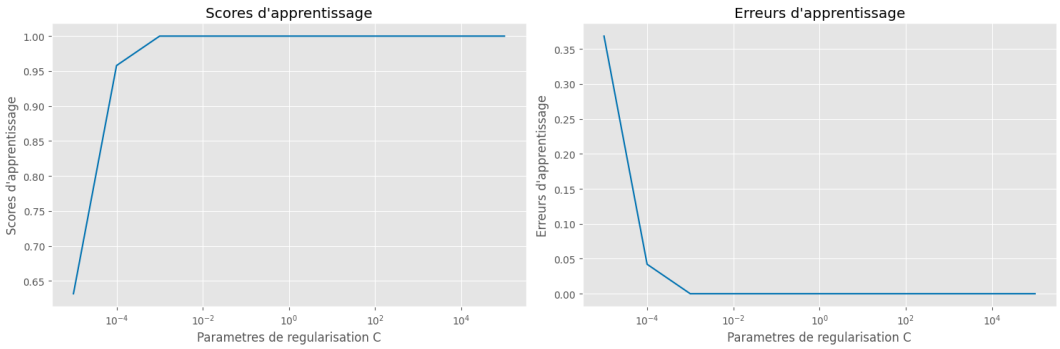
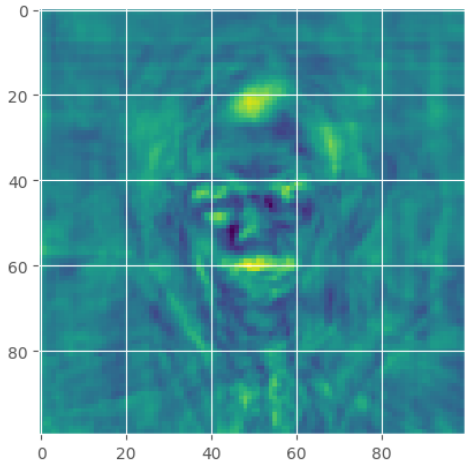


Figure 9: Scores et erreurs d'apprentissage selon la valeur de C.

Comme nous pouvons l'observer dans la Figure 9 ci-dessus, la meilleure valeur du paramètre C de régularisation est 10^{-3} puisque la valeur de score d'apprentissage est de 1, et celle d'erreur d'apprentissage est de 0.

Nous allons donc utiliser cette valeur pour recréer un classifieur linéaire, l'entraîner avec les données d'apprentissage, et le tester sur les données de test, en prédisant les valeurs y de test à l'aide des valeurs X de test et du modèle entraîné sur les données d'apprentissage.
Nous obtenons ainsi une précision de 0.9, avec un niveau de chance (si le modèle ne fait rien de particulier pour apprendre ou généraliser, il se contente de prédire la classe majoritaire) de 0.62. Nous avons donc une prédiction nettement meilleure que l'aléatoire, ce qui signifie que notre modèle a bien appris à classifier les données correctement (90% de précision).

(a) Prédictions



(b) Coefficients

Figure 10: Représentation des prédiction et des coefficients de différenciation.

Nous pouvons donc vérifier si les prédictions sont vraiment précises à l'aide de la Figure 10a. Dans l'échantillon présenté, toutes les prédictions sont justes, mais il ne faut pas oublier qu'il y a une précision de 90% avec ce classifieur.
De plus, nous pouvons aussi regarder une représentation des coefficients qui ont permis au classifieur d'atteindre une telle précision, dans la Figure 10b. En effet, ce sont les zones en jaune qui ont le plus permis d'établir une différence entre les deux personnes que nous avons traitées. Le contour des yeux, de la bouche, et les cheveux semblent être les facteurs qui ont permis au classifieur de distinguer ces deux personnes.

V Question 5

Nous allons à présent rajouter des variables de bruit dans notre modèle afin d'observer dans quelle mesure la performance de celui-ci est impactée. Tout d'abord, à l'aide de la fonction `run_svm_cv`, nous allons utiliser la validation croisée. Nous obtenons alors, sans ajouter de variables de nuisance, un score de généralisation (de test) égal à 0.9 (moyenne sur 10 simulations). Lorsque l'on rajoute des variables de nuisance, nous obtenons un score de généralisation égal à 0.54. La performance de notre modèle chute drastiquement. Nous observons aussi un grand écart entre les deux score, significatif d'un sur-ajustement caractérisé par l'ajout des variables de bruit.

VI Question 6

Nous pouvons améliorer cette prédiction à l'aide d'une réduction de dimensions en utilisant l'objet `sklearn.decomposition.PCA(n_components, svd_solver='randomized')`. On remarque qu'en faisant varier la valeur de `n_components`, le temps d'exécution peut changer du tout au tout. En effet, pour $n\_components = 100$, le code s'exécute en 1 seconde, mais quand on diminue la valeur de ce paramètre, le temps d'exécution augmente. Cela est dû au fait que ce paramètre représente le nombre de composantes que l'on souhaite garder après la réduction de dimension.
Pour $n\_components = 80$, nous trouvons un score de généralisation égal à 0.58. Nous pouvons remarquer que l'amélioration reste relative, et qu'il faudrait baisser le nombre de dimensions à conserver pour augmenter ce score. Pour ces valeurs de composantes à conserver dans l'ACP nous observons toujours du sur-ajustement mais pour des valeurs plus basses (que nous n'avons pas effectuées mais un camarade l'a fait (Quentin FESTOR)), il y a une amélioration. En effet, nous obtenons les valeurs suivantes :

Scores / *n_components*	Apprentissage	Test
200	0.9263157894736842	0.5210526315789473
120	1.0	0.5263157894736842
80	0.7473684210526316	0.48947368421052634
25	0.6947368421052632	0.5842105263157895
20	0.6578947368421053	0.5894736842105263
15	0.6526315789473685	0.5894736842105263
10	0.6052631578947368	0.6368421052631579
5	0.6052631578947368	0.6157894736842106

Table 1: Scores d'appretissage et de test selon le nombre de dimensions conservées ans la réduction de dimensions

Ce code a été réalisé en 11 heures (sur une seule machine). Nous pouvons observer dans le Tableau 1 que plus la valeurs de $n\_components$ diminue et plus le score de test augmente, et l'écart entre les deux scores diminue, ce qui signifie qu'on corrige le sur-ajustement par la réduction de dimensions.