TP n° 3 : Support Vector Machine (SVM)

Rivaldi Tristan

2024-05-29

Table des matières

Introduction

Ce TP porte sur les **Support Vector Machines (SVM)**, une méthode d'apprentissage supervisé populaire pour la classification binaire. Les SVM cherchent à séparer les données en utilisant un hyperplan optimal dans un espace à haute dimension, souvent transformé à l'aide d'une fonction noyau. Grâce à l'implémentation dans la bibliothèque **scikit-learn**, nous allons explorer l'utilisation des SVM sur des jeux de données. L'objectif est de comprendre comment ajuster les hyperparamètres et choisir le noyau adapté afin d'optimiser les performances du modèle, tout en examinant les enjeux liés à la régularisation.

Le code utiliser dans se projet est disponible dans le référentiel https://github.com/TristanRivaldi/HAX907X_APP_STAT. J'ai fixé la graine dans le code pour que celui-ci soit reproductible.

Question 1

Intéressons-nous aux performances des SVM sur un dataset bien connu en machine learning : le dataset **Iris**. Ce dataset décrit les caractéristiques de trois espèces de fleurs à partir de quatre variables quantitatives représentant divers aspects physiques de chaque plantes.

Dans cette expérience, nous allons restreindre notre analyse aux **classes 1 et 2** du dataset pour rester dans un cadre de classification binaire. Nous utiliserons uniquement les deux premières variables explicatives (longueur et largeur des sépales) pour simplifier notre modèle.

Le classifieur SVM que nous allons utiliser sera entraîné avec un noyau linéaire.

Nous n'allons pas fixer manuellement le paramètre de régularisation **C** mais plutôt utiliser une validation croisée avec la classe **GridSearchCV** afin de déterminer la meilleure valeur de **C** pour notre modèle.

Une fois le modèle entraîné avec les données d'entraînement, nous évaluons ses performances sur un ensemble de test (50 % des données totales).

Après la validation croisée, le meilleur paramètre de régularisation trouvé pour le noyau linéaire est le suivant : **C**= 0.29

Le score de prédiction sur les données d'entraînement et de test est de 0.66, ce qui signifie que le modèle est capable de prédire correctement dans 66% des cas, à la fois sur les données d'entraînement et sur les données de test.

Question 2

Après avoir testé un SVM à noyau linéaire, nous avons maintenant ajusté un SVM avec un noyau polynomial afin de comparer les performances des deux approches.

Le modèle polynomial a été ajusté à l'aide de la classe **GridSearchCV** pour déterminer les meilleurs hyperparamètres, notamment le degré du polynôme, le paramètre de régularisation C, ainsi que le paramètre γ. La recherche a été effectuée sur les valeurs suivantes :

- C : testé sur une échelle logarithmique allant de 10^{-3} à 10^{3}.
- γ : testé avec la valeur 10.
- Degré : testé avec les valeurs 1, 2 et 3.

Après l'entraînement, les meilleurs paramètres trouvés pour le SVM à noyau polynomial sont : {'C': 0.001, 'degree': 2, 'gamma': 10.0, 'kernel': 'poly'}

Le SVM à noyau polynomial a atteint les résultats suivants :

- **Score d'entraînement** : 0.64
- **Score de test** : 0.44

Cela signifie que le modèle classifie correctement 64% des données d'entraînement, mais seulement 44% des données de test.

Ces résultats montrent que le modèle linéaire a une **meilleure stabilité** entre les ensembles d'entraînement et de test.

Le **noyau linéaire**, avec un score constant de 0.66 pour l'entraînement et le test, montre que la simplicité de la frontière de décision linéaire est plus efficace pour ces données. Cela indique que les classes 1 et 2 sont suffisamment bien séparées par une frontière linéaire, et qu'une complexité supplémentaire, comme celle introduite par le noyau polynomial, n'est pas nécessaire.

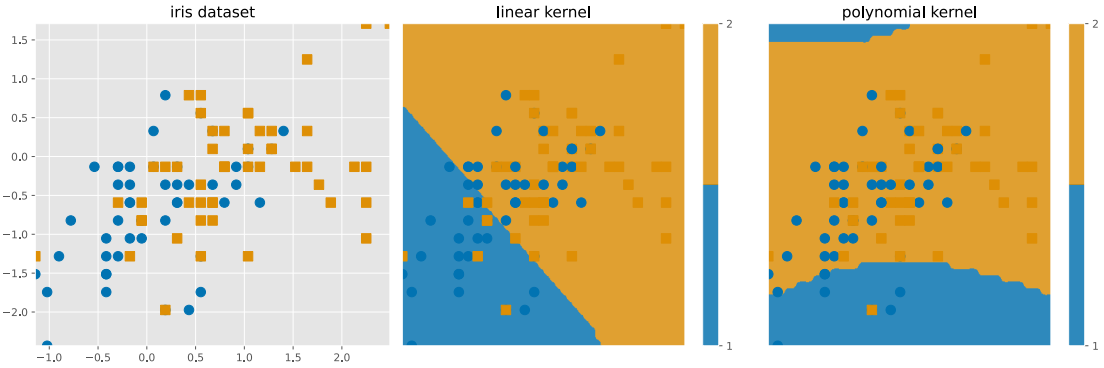


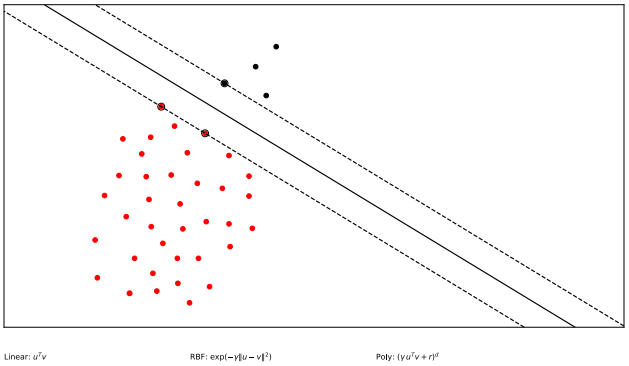
Figure 1: Comparaison des noyaux

En observant la Figure 1 on voit que les frontières données par le modèle avec noyau polynômiale sont plus complexes que celles données par le noyau linéaire. On remarque aussi que ce n'est pas forcément nécessaire de l'utiliser dans cette exemple.

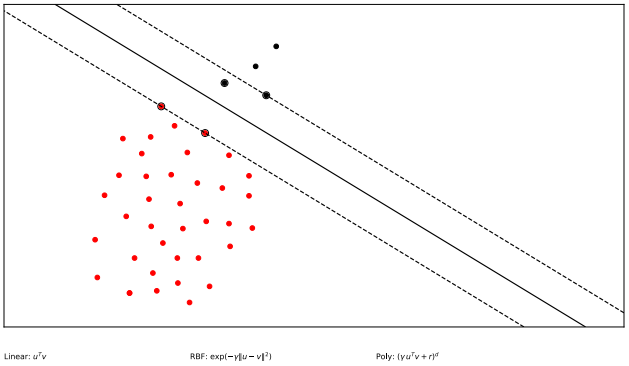
J'ai effectué de nombreux essais et, à chaque fois, l'utilisation du noyau polynomial n'a pas été nécessaire.

Question 3

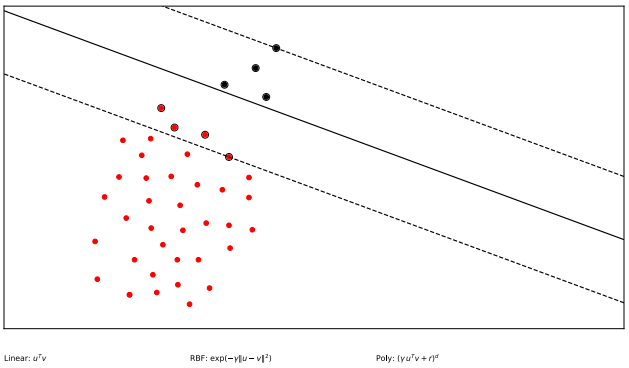
Dans cette question, nous allons analyser l'impact du paramètre de régularisation **C** sur les performances d'un SVM à noyau linéaire. Pour cela, nous avons généré un jeu de données très déséquilibré avec 90 % des données appartenant à une classe et seulement 10 % à l'autre. Ensuite, nous avons évalué le SVM a noyau linéaire pour différentes valeurs de **C**. Voici les résultats obtenues :



(a) C=1



(b) C=0.01



(c) C=0.001

Figure 2: Evaluations pour différentes valeur de C

En observant la Figure 2 on voit que le paramètre de régularisation **C** joue un rôle crucial dans la manière dont le SVM ajuste la marge de séparation entre les classes. Une valeur élevée de **C** donne une marge plus étroite et une séparation plus stricte, tandis qu'une faible valeur de **C** élargit la marge.

Question 4

Nous allons maintenant étudier le cas de classifications de visages avec la base de données **fetch_lwd_people** disponible aussi dans la librarie **sklearn.datasets**. Elle contient des images de personnalités politiques américaine.

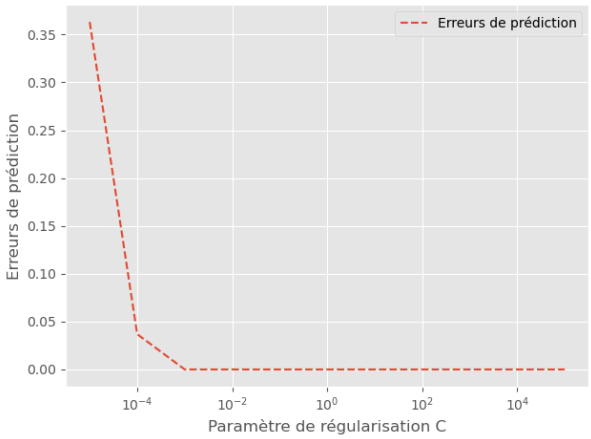
Pour rester dans un cadre simple, nous allons nous contenter de classifier deux classes d'images : celles de Tony Blair et celles de Colin Powell. Voci un extrait de leurs photos disponible dans le jeu de donnée (voir Figure 3).

Figure 3: Extrait du jeu de donnée

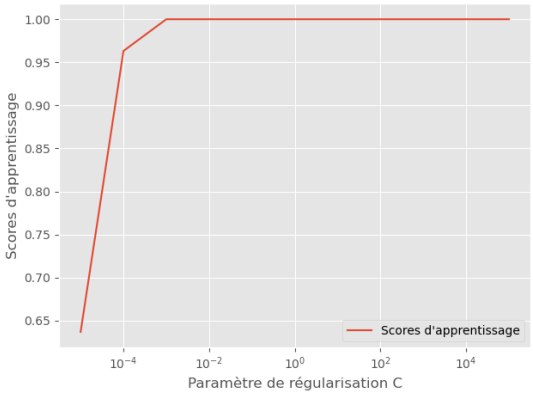
Influence du paramètre de régularisation

Le paramètre **C** dans un modèle SVM contrôle la régularisation. Une régularisation forte (**C** petit) pénalise fortement les grands coefficients, favorisant un modèle plus simple et régulier. Un **C** élevé

permet au modèle de mieux s'ajuster aux données d'entraînement (risque de surapprentissage).Le code teste plusieurs valeurs de **C** sur une échelle logarithmique allant de 1e-5 à 1e5.



(a) Erreur de prédiction



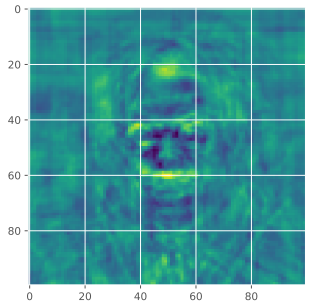
(b) score d'apprentissatge

Figure 4: Influence du paramètre C sur l'erreur d'apprentissage et le score d'apprenstissage

D'apres les Figure 4a et Figure 4b on voit que le **C** optimal vaut 10^{-3} c'est le **C** le plus petit qui permet d'atteindre une erreur de prédiction sur l'ensemble d'apprentissage minimal et un score d'apprentissage maximal.

Pour les petites valeurs de **C** (infèrieur à 10^{-3}) on remarque que l'erreur de prédiction est relativement élevée et que le score d'apprentissage est faible. Cela peut s'explique par le fait que le SVM devient alors très tolérant aux erreurs de classification, ce qui conduit à un sous-apprentissage, c'est-à-dire que le modèle est trop simple pour capturer les motifs dans les données d'entraînement.

(a) Exemple de prédictions



(b) Représentation de coefficients

Figure 5: Extrait de prédictions et représentations des coefficients

On va ensuite regarder les performances de notre modèle avec ce $\mathbf{C}$ optimal qui vaut 10^{-3} on obtient une précision (Accuracy) qui vaut 0.94 sur l'ensemble de test ce qui veut dire que le modèle donne la bonne classification dans 94% des cas. La Figure 5a permet de voir quelques prédictions du modèle, on voit que dans ce petit extrait le modèle c'est tromper une seule fois.

On obtient comme "niveau de chance" 62% qui représente la précision obtenue en prédisant toujours la classe majoritaire. Notre modèle a donc réussi à dépasser largement le niveau de chance, ce qui montre qu'il a bien appris à reconnaître les différents patterns dans les données pour faire des prédictions correctes.

La Figure 5b permet de visualiser les coefficients du SVM sous forme d'image. Cette représentation montre les zones de l'image utilisées par le modèle pour différencier les deux classes de visages (Colin Powell et Tony Blair). On observe que les contours des yeux, de la bouche et des cheveux semblent être les facteurs principaux qui ont permis au classifieur de distinguer ces deux personnes.

Question 5

Nous allons voir ici que la présence de variables de nuissances fait drastiquement chuter la performance d'un classifieur SVM.

Pour le montrer, nous allons rajouter à nos données d'apprentissage 300 pixels de bruits, suivant une loi normale centrée réduite avec le code suivant :

```
sigma = 1
noise = sigma * np.random.randn(n_samples, 300)  # 300 variables de nuisance
X_noisy = np.concatenate((X, noise), axis=1)
X_noisy = X_noisy[np.random.permutation(X.shape[0])]
```

Maintenant, nous pouvons évaluer les performances de prédiction de ce modèle avec des variables de nuisance et les comparer à celles du modèle simple. Les calculs seront effectués à l'aide de la fonction **run_svm_cv** disponible dans le fichier **script_svm.py**. On peut voir les performaces des modèles dans la Table 1.

Table 1: Résultats

Modèles	Sans variables de bruit	Avec variables de bruit
Score apprentissage	1.0	0.93
Score test	1.0	0.49

Les résultats de la Table 1 montrent que, sans variables de bruit, le modèle atteint une performance parfaite sur l'ensemble d'entraînement (1.0) et une bonne performance sur l'ensemble de test (0.93). Cependant, lorsque des variables de bruit sont ajoutées, bien que la performance en apprentissage reste élevée (1.0), la performance sur l'ensemble de test chute drastiquement à 0.49. Cela indique que l'ajout de bruit dégrade considérablement la capacité du modèle à faire de la prédiction.

Question 6

Une façon de contourner ce problème est d'appliquer une réduction de dimension. Une méthode bien connue consiste à sélectionner les directions contenant le plus d'informations dans le sous-espace généré par les variables : c'est l'analyse en composantes principales (PCA). Pour cela, nous allons utiliser **sklearn.decomposition.PCA(svd_solver='randomized'**.

```
# Réduction de dimension avec PCA
print("Score après réduction de dimension")
n_components = 150  # Nombre de composantes, peut être ajusté

# Appliquer la PCA
pca = PCA(n_components=n_components, svd_solver='randomized', whiten=True)
#whiten=True normalise les composantes principales
X_noisy_pca = pca.fit_transform(X_noisy)

# Passer les données réduites à la fonction run_svm_cv
run_svm_cv(X_noisy_pca, y)
```

Après avoir appliqué la réduction de dimension avec PCA (avec 150 composantes principales), le score d'apprentissage diminue à 0.63, indiquant que le modèle n'est plus capable de parfaitement classer les données d'entraînement. Cependant, le score de test s'améliore considérablement par rapport au cas précédent, atteignant 0.61. Cela montre que la PCA a permis de réduire l'effet du bruit en conservant les directions les plus importantes, ce qui améliore la capacité du modèle à généraliser sur les données de test.

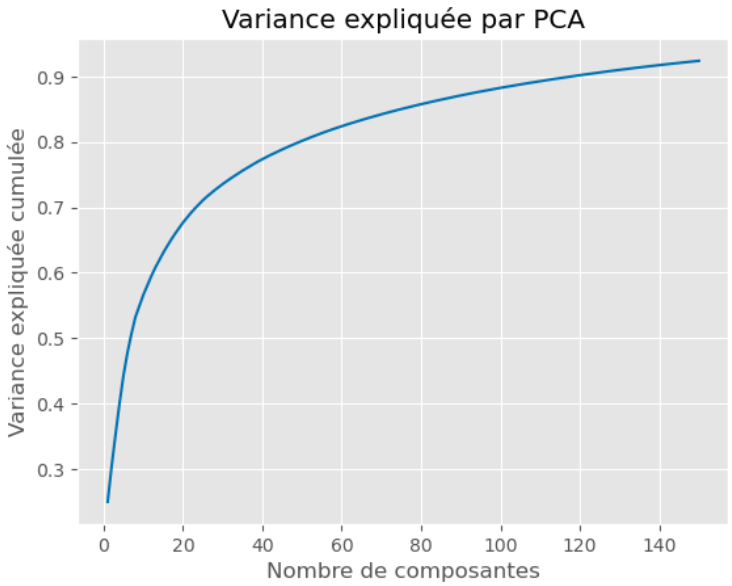


Figure 6: Variance expliquée

La Figure 6 montre la variance expliquée cumulée en fonction du nombre de composantes principales sélectionnées lors de l'analyse en composantes principales. Ce graphique montre que, grâce à la PCA, il est possible de réduire significativement la dimensionnalité des données tout en conservant la majeure partie de l'information. Car ici les 120 premières composantes principales expliquent environ 90% de la variance totale du modèle.

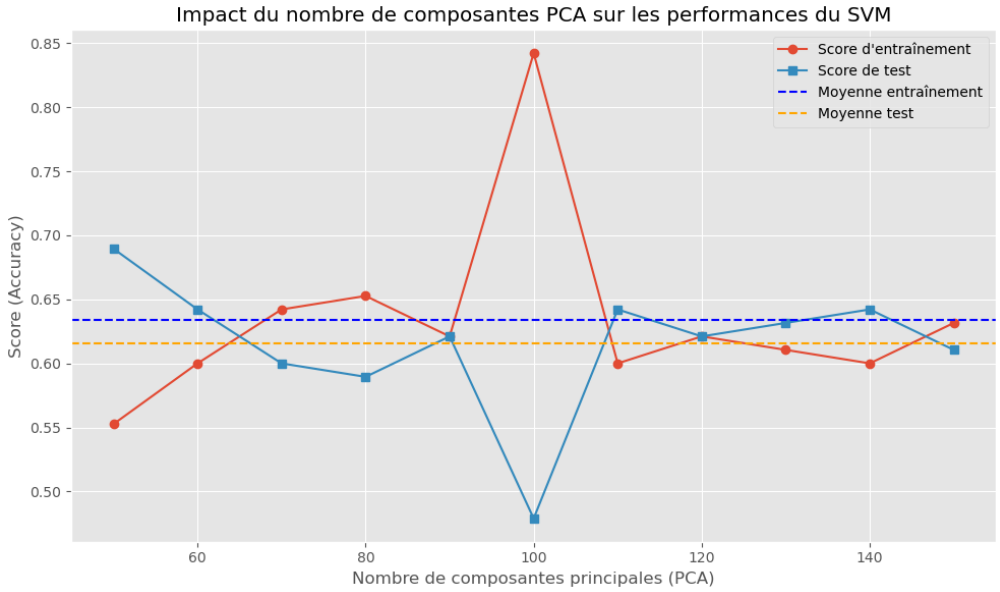


Figure 7: Impact du nombre de composantes PCA

La Figure 7 montre l'impact du nombre de composantes principales (PCA) sur les performances du SVM, évaluées à travers les scores d'entraînement et de test. Le nombre de composantes varie de 50 à 150 avec un pas de 10. On observe de grandes fluctuations dans les résultats en fonction du nombre de composantes choisies. Cependant, en moyenne, quelle que soit le nombre de composantes choisie, la PCA améliore les scores de prédiction sur les données bruitées. En effet, le score moyen sur l'ensemble de test est d'environ 0,62 ce qui est nettement meilleur que le score de 0,49 obtenu sans PCA.

On observe également que le score d'apprentissage est régulièrement inférieur au score de test, ce qui semble être dû à l'utilisation de l'option **whiten=True**. Cette commande normalise les composantes principales en réduisant la variance de chaque dimension à une unité, ce qui peut effectivement améliorer la généralisation (meilleur score sur les données de test). Cependant, cette normalisation peut aussi entraîner une perte d'information importante. En conséquence, le modèle SVM peut avoir des difficultés à bien s'ajuster sur l'ensemble d'apprentissage, ce qui se traduit par un score d'apprentissage plus faible.

Il aurait été utile de tester un nombre de composantes inférieur à 50, mais les performances de mon ordinateur ne le permettent pas, et avec un choix de 150 composantes, les résultats étaient déjà meilleurs que sans l'utilisation de la PCA.