TP Support Vector Machine

Mario LAPI

Septembre 2024

Contents

introduction

Dans ce TP, nous explorons l'application des Support Vector Machine (SVM) à travers plusieurs approches, en étudiant leur capacité à classifier des données complexes. En utilisant différents noyaux (linéaire et polynomial), nous commençons par entraîner un SVM sur le célèbre jeu de données Iris, avant de nous intéresser à des problématiques plus avancées comme la classification d'images de visages. L'objectif est d'analyser les performances des modèles, notamment en jouant sur des paramètres comme le coefficient de régularisation C, et en introduisant des variables de nuisance pour observer l'impact du bruit sur la prédiction. Enfin, nous utilisons des techniques de réduction de dimension telles que la PCA pour simplifier le modèle.

1 Première Partie : dataset Iris

L'objectif ici est d'entraîner un SVM avec un noyau linéaire pour classifier les classes 1 et 2 du dataset Iris, en utilisant uniquement les deux premières variables. Ensuite, on évalue la performance du modèle en termes de généralisation, c'est-à-dire sa capacité à faire des prédictions correctes sur des données qu'il n'a pas vues lors de l'entraînement.

1.1 Question 1 : Noyau linéaire

Nous avons rédigé un code qui réalise cette classification à l'aide d'un noyau linéaire.
Nous allons ensuite évaluer les scores d'entraînement et de prédiction du modèle. Nous allons répéter cette opération 50 fois puis faire les moyenne des scores obtenus afin d'avoir un résultat plus robuste.

On obtient les résultats suivants:

Score d'entraînement	Score de prédiction
0.7128	0.6860

Table 1: Scores d'un SVM avec noyau linéaire

Des résultats de envirron 0.7 montrent que le modèle fait quelques erreurs de classification, mais dans l'ensemble, il parvient à séparer les classes de manière correcte.
Le score de prédiction (0.6860) est très légèrement plus faible que le score d'entraînement (0.7128), ce qui est normal, il est plus difficile pour le modèle de généraliser à de nouvelles données qu'il ne connaît pas.
Ce résultat est même plutôt satisfaisant car la différence entre les deux scores est plutôt faible, ce qui peut signifier que le modèle généralise correctement.

Il est important de noter que le modèle se base uniquement sur deux variables, il est possible qu'une meilleure performance puisse être atteinte en utilisant plus de variables (comme les dimensions des pétales) ou en utilisant un noyau non-linéaire, ce que nous allons essayer dans la suite de cette étude.

1.2 Question 2: Noyau Polynomial

Nous avons procédé de la même facon que dans la question précédente, mais en choisissant cette fois un noyau polynomial au lieu d'un noyau linéaire, le but étant de voir si un noyau plus complexe permet de mieux classer nos données. Nous obtenons alors les résultats suivants:

Score d'entraînement	Score de prédiction
0.6752	0.5848

Table 2: Scores d'un SVM avec noyau polynomial

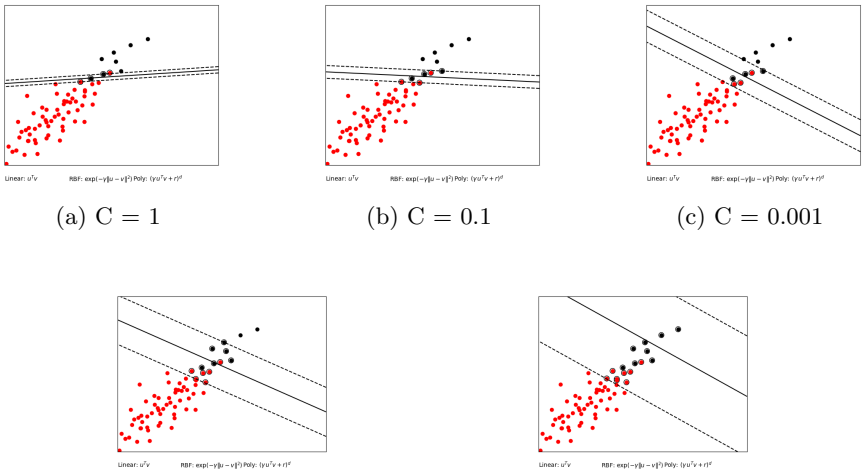
On remarque que nos deux scores ont baissé, de plus, la différence entre les deux scores à augmenté. Ainsi on peut dire que le SVM avec noyau polynomial n'améliore pas la performance par rapport au noyau linéaire, elle en est même grandement déteriorée.

Il semblerait donc que la relation entre les sépales de nos iris soient plutôt de nature linéaire que polynomiale.

1.3 Question 3: SVM GUI

Le SVM cherche à maximiser la distance entre la frontière de décision et les points de données les plus proches (vecteurs supports). En même temps, il essaie de minimiser les erreurs de classification (les points mal classés ou trop proches de la frontière).

Le paramètre de régularisation C joue un rôle essentiel dans les SVM en contrôlant le compromis entre la maximisation de la marge du séparateur et la minimisation de l'erreur de classification. Nous allons illustrer cela en jouant avec le paramètre C dans à l'interface générée par le code **svm_gui.py**
On commence par générer un jeu de données très déséquilibré, ici 90% de points rouges contre 10% de points noirs. On obtient alors les résultats suivants:



(a) C = 1 (b) C = 0.1 (c) C = 0.001

(d) C = 0.0001 (e) C = 0.00002

Figure 1: Séparation des classes en fonction du paramètre C

- Avec un C élevé, le modèle essaye de classer correctement la minorité. Par exemple sur la Figure (a) où C = 1, un seul point rouge est mal classé, en revanche tous les points noirs sont bien classés.
L'inconvénient est que la marge à la frontière est très petite. Ainsi une légère perturbation sur une donnée proche de la frontière pourrait avoir un impact sur le SVM.

- Avec un C plus petit le modèle devient plus tolérant aux erreurs de classification pour la classe minoritaire, car il privilégie la marge plutôt que la classification exacte de chaque point. On remarque même dans le cas extrême (Figure (e) avec C = 0.00002) que 5 points noirs sur 7 sont du mauvais côté de

la frontière et donc mal classés.
À noter que l'augmentation de la marge à la frontière se fait toujours au détriment de la classe minoritaire.

2 Deuxième Partie: Classification de visages

Dans cette partie, nous allons faire de la classification sur un jeu de données contenant des images de visages.

2.1 Question 4: Choix de la constante de régularisation

Le but de cette question est de mesurer l'impact du paramètre de régularisation C sur la classification proposée par le SVM. Pour cela nous allons évaluer le score d'apprentissage en fonction de C.
On obtient alors le graphique suivant:

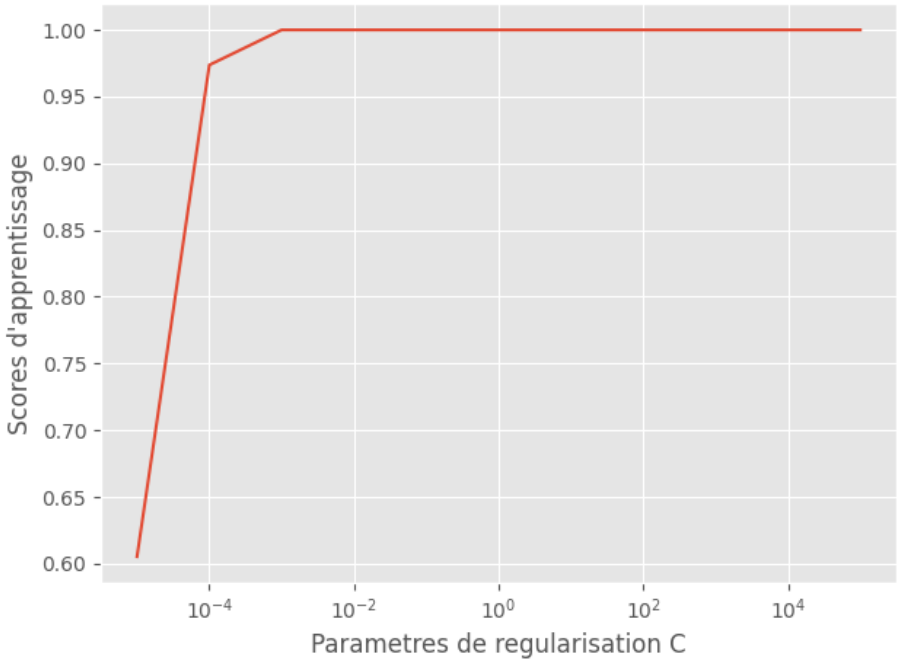


Figure 2: Évolution du score d'apprentissage en fonction du paramètre de régularisation C

On observe sur la Figur 2 que le score d'apprentissage augmente très vite pour atteindre une valeur de 1 dès que C est plus grand que 10^{-3}. Un score d'apprentissage de 1 signifie qu'il n'y a plus d'erreur de classification. Pour trouver la valeur optimale de C il suffit donc de prendre le plus petit C permettant

un score de 1 afin d'avoir une distance entre les vecteurs supports et la frontière qui soit la plus grande possible.
On trouve alors une valeur optimale pour ce paramètre de $C = 0.001$

On peut maintenant mettre à l'épreuve notre SVM sur un exemple concret.
Après avoir entraîné notre SVM avec des images associées à un nom (ici "Blair" ou "Powell").
On le teste ensuite sur 12 photos sans lui dire qui est qui, le modèle doit ensuite reconnaître la personne. Voici le resultat:

Figure 3: Résultats de classification du SVM

On remarque que les résultats sont plutôt bons, en effet sur les 12 images, notre SVM ne s'est trompé qu'une seule fois.

Après cette opération, on peut même observer les zones les plus utiles au SVM pour la classification grâce à la Figure 4. Dans ce dernier, plus la zone est claire, plus elle est déterminante dans la reconnaissance d'une personne:

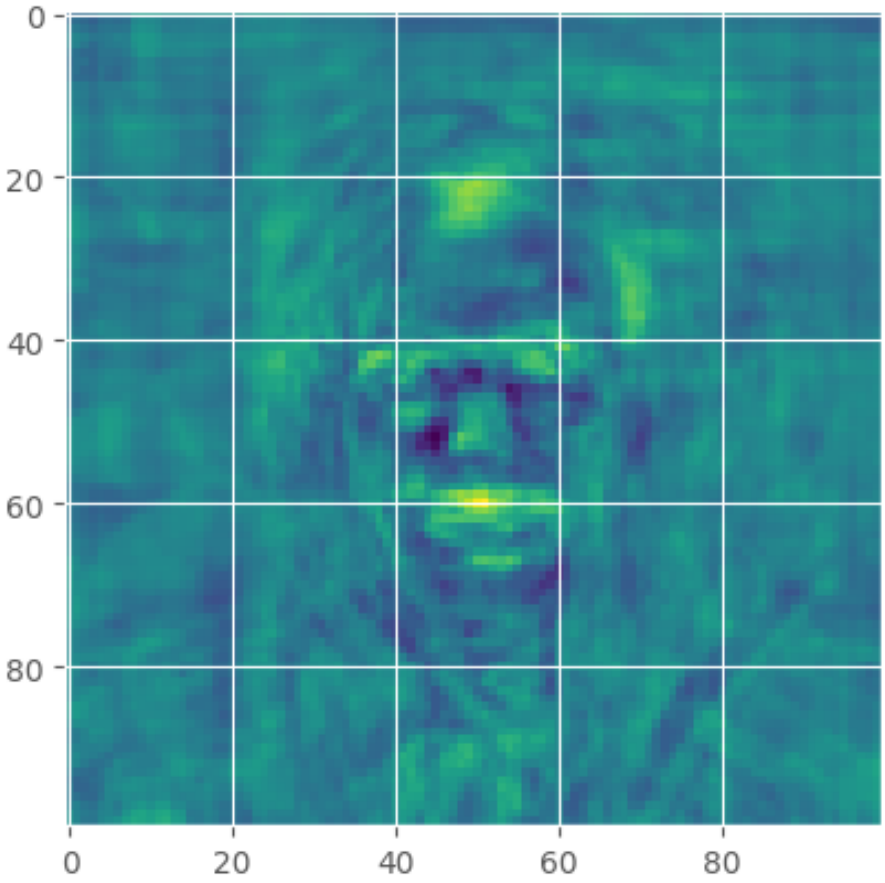


Figure 4: Graphique des zones importantes pour le SVM

Sans surprise, on remarque que les zones les plus utiles pour reconnaître une personne sont sur le visage notemment les yeux et la bouche.

2.2 Question 5: Comparaison des scores d'un SVM avec et sans bruit ajoutés

Nous allons maintenant regarder l'impact que pourrait avoir l'ajout de variables de bruit sur le score de prediction.
Nous avons donc crée deux jeux de données, un premier auquel nous avons ajouté 300 variables de nuisance (gaussiennes) au données initiales, et un deuxième que l'on n'a pas touché qui nous servira de témoin pour comparer avec le modèle bruité.
Nous allons ensuite regarder les scores de prédiction de chacun et les comprarer. Pour plus de précision on répète cette opération 50 fois afin d'obtenir des moyennes de scores.

	Sans variables de nuisances	Avec variables de nuisances
Score de prédiction moyen	0.90410	0.54401

Table 3: Comparaison des scores d'un SVM avec et sans bruit

En ajoutant du bruit inutile, la performance chute.
Ceci n'est pas surprenant, en effet, les variables de nuisance augmentent la dimensionnalité du problème sans ajouter d'information utile, ce qui complique la tâche pour le SVM, et finit par diminuer le score de prédiction moyen.

2.3 Question 6: Réduction de la dimension

Après avoir augmenté artificiellement le nombre de variables en y ajoutant du bruit, nous allons la réduire à l'aide d'une PCA (Principal Components analysis) et observer l'évolution des scores d'apprentissages et de test.
Ne sachant pas à l'avance le bon nombre de composantes à conserver, nous allons faire varier ce paramètre.

Nombre de composantes	20	50	80	130	200
Score d'apprentissage	0.66315	0.8405	0.81210	0.9947	1.0
Score de prédiction	0.58368	0.5768	0.55947	0.5315	0.5157
Variance expliquée	68%	80%	86%	0.91%	0.95%

Table 4: Tableau des performances en fonction du nombre de composantes conservées

Une première remarque est que le pourcentage de variance expliquée est globalement très bon. Ceci est normal, 300 variables ne contiennent que du bruit Au fur et à mesure que la dimension diminue, on observe plusieurs phénomènes intéressant:

- **Le score d'apprentissage diminue.** Le modèle est plus simple et a moins de liberté pour capturer tous les moindres détails des données d'entraînement, y compris le bruit. C'est plutôt bon signe, cela signifie que ce dernier évite le surapprentissage.
- **Le score de prédiction augmente.** La réduction de la dimension rend le modèle moins complexe, il est capable de mieux généraliser sur de nouvelles données.

Cela illustre bien le compromis biais-variance :

Avec plus de variables, le modèle est flexible et peut bien s'ajuster aux données d'entraînement, cependant il risque de surapprendre (forte variance). Il est donc moins performant sur de nouvelles données. En réduisant la dimension, le modèle devient plus simple, ce qui augmente le biais (il capture moins de détails), mais il est plus robuste et généralise mieux sur des données inconnues (faible variance)

En résumé, réduire la dimension permet de mieux équilibrer biais et variance, en évitant le surapprentissage tout en améliorant la capacité de prédiction.