Rapport de TP

M2 AppStat

Damien Mariac

2 octobre 2025

Introduction

On souhaite mettre en pratique une technique de classification sur données réelles et simulées au moyen du package scikit-learn et d'apprendre à contrôler les paramètres garantissant leur flexibilité (hyper-paramètres, noyau).

On va tester dans une première partie cette méthode sur des données simulées et sur le jeu de données Iris de Fisher. Puis dans une deuxième partie, nous utiliserons cette technique sur de la classification de visages provenant du module de Scikit-learn.

Partie 1 - Jeux de données numériques

Jeu de données simulées (gaussien)

On génère deux échantillons de données de taille 200 suivant une loi gaussienne bi-dimensionnelle. Les deux gaussiennes sont centrées en (1,1) et (-1/2,-1/2) et ont pour matrice de covariance $0.9 * I$.

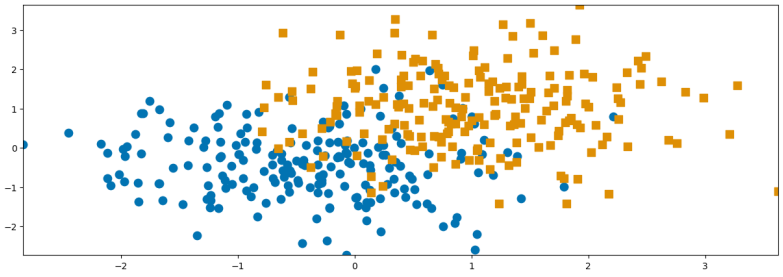

FIGURE 1 – Données gaussiennes simulées

On décide de séparer les données en deux ensembles : un ensemble d'apprentissage et un ensemble de test. On utilise 50% des données pour l'apprentissage et 50% pour le test.

Avec l'ensemble d'apprentissage, on cherche à ajuster un SVM avec un noyau **linéaire**. On utilise la classe SVC de Scikit-learn avec le paramètre C=1.0 (paramètre de régularisation). Puis on évalue la performance de ce modèle sur l'ensemble de test. Ceci nous donne un score de 0.89 ce qui est plutôt bon.

On peut visualiser la frontière de décision du SVM sur la figure ci-dessous.

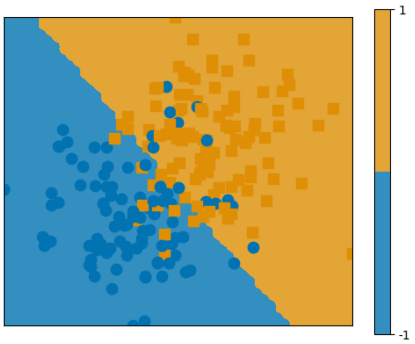

FIGURE 2 – Frontière de décision SVM avec noyau linéaire

Il existe quelques points mal classés mais la frontière de décision est globalement correcte. On peut chercher à améliorer le modèle en cherchant un paramètre C plus adapté. Pour cela, on peut utiliser la validation croisée (cross-validation) pour tester plusieurs valeurs de C et choisir celle qui donne le meilleur score sur l'ensemble de validation.

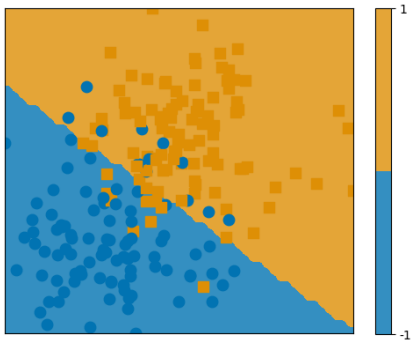

FIGURE 3 – Frontière de décision SVM avec noyau linéaire et C par validation croisée

On observe un score relativement similaire de 0.89 lorsque nous avons un paramètre $C = 0.1$.

Il est rassurant de voir que si on génère un nouveau jeu de données avec une gaussienne centrée en (5,5) et l'autre en (-1/2,-1/2), le SVM avec un noyau linéaire s'adapte très bien et obtient un score de 1.0 sur l'ensemble de test. Il sépare parfaitement les deux classes comme on peut le voir sur la figure ci-dessous.

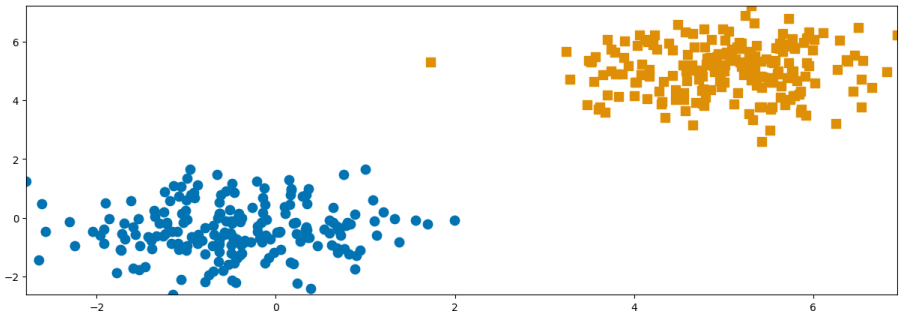

FIGURE 4 – Gaussienne centrée en (5,5) et (-1/2,-1/2)

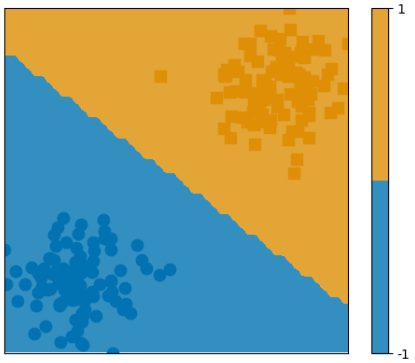

FIGURE 5 – Frontière de décision SVM avec noyau linéaire (gaussienne centrée en (5,5) et (-1/2,-1/2))

Jeu de données Iris de Fisher

On utilise le jeu de données Iris de Fisher disponible dans Scikit-learn. On décide de ne garder que les classes 1 et 2 (resp. versicolor et virginica) pour faire un problème de classification binaire. On utilise seulement les deux premières caractéristiques (longueur et épaisseur des sépales) pour pouvoir visualiser les données en 2D. On sépare les données en un ensemble d'apprentissage qui correspond à 25% des données.

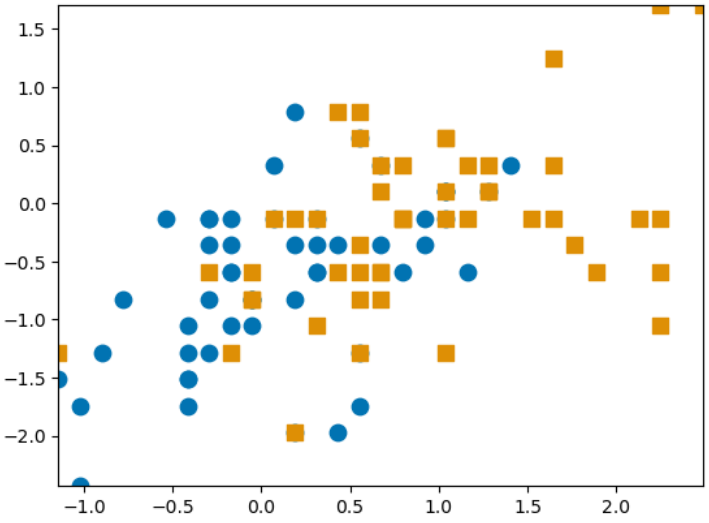

FIGURE 6 – Données Iris (classes 1 et 2)

Après entraînement du modèle SVM avec un noyau linéaire, on obtient un score de **0.747 sur l'ensemble d'entraînement**, et un score de **0.68 sur l'ensemble de test**. Ce qui est moyennement satisfaisant mais normal vu que les données ne sont pas « vraiment » visuellement linéairement séparables au vu de la figure 6.
On peut visualiser la frontière de décision du SVM sur la figure ci-dessous.

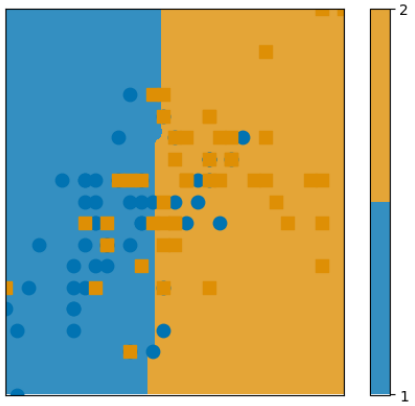

FIGURE 7 – Frontière de décision SVM avec noyau linéaire

Il est naturel de se demander si un noyau linéaire est adapté à ce jeu de données. On essaye donc avec un noyau polynomial de degré 2. On obtient un score de **0.653 sur l'ensemble d'entraînement**, et un score de **0.52 sur l'ensemble de test**. Ce qui est moins bon qu'avec le noyau linéaire. On peut visualiser la frontière de décision du SVM sur la figure ci-dessous.

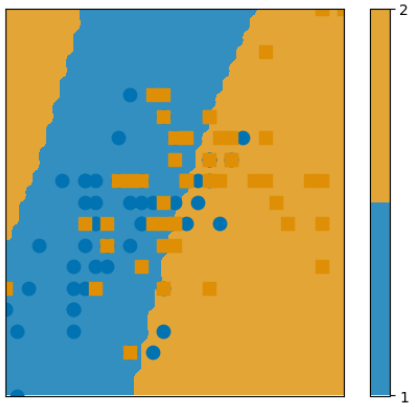

FIGURE 8 – Frontière de décision SVM avec noyau polynomial de degré 2

Partie 2 - Classification des visages

Dans cette partie, on s'intéresse à la classification de visages. Autrement dit, étant donné une image de visage, on cherche à déterminer à quelle personne elle appartient. On utilise le jeu de données *"Labeled Faces in the Wild" (LFW)* disponible dans **Scikit-Learn**. On se limite à la classification de 2 personnes (Tony Blair et Colin Powell) afin de faire un problème de classification binaire. On effectue une division aléatoire des données en ensembles d'entraînement et de test répartis en parts égales.

Impact du paramètre C :

On commence par utiliser un SVM avec un noyau linéaire en faisant varier le paramètre de régularisation C afin de voir son impact sur la frontière de décision.
On affiche sur la figure ci-dessous l'erreur de prédiction en fonction de C sur une échelle logarithmique entre **1e-5** et **1e5**.

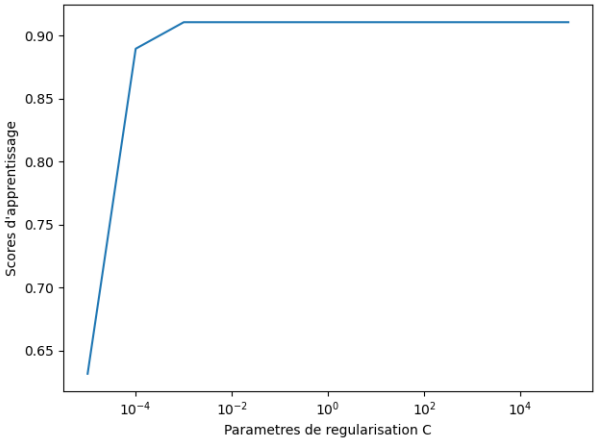

FIGURE 9 – Erreur de prédiction en fonction de C (noyau linéaire)

On obtient numériquement un score maximale de **0.91** pour $C = 0.001$ **sur l'ensemble de test**. C'est un très bon score. De ce qui est de l'allure de la courbe, on observe que pour des valeurs de C très petites (inférieures à 1e-3), l'erreur de prédiction est élevée. Cela s'explique par le fait que le modèle est trop régularisé et ne s'adapte pas bien aux données d'entraînement (sous-apprentissage). Pour des valeurs de C très grandes (supérieures à 1e3), la pénalisation des erreurs augmente et donne une meilleure adaptation aux données d'entraînement. Une fois que le meilleur hyperplan linéaire qui généralise bien est trouvé, changer C ne modifie plus les vecteurs supports actifs ni la frontière (conditions KKT inchangées) ce qui forme ce plateau à partir de $C = 10^{-2}$.

FIGURE 10 – Prédiction des visages (C=0.001, noyau linéaire)

On observe que le modèle est capable de bien séparer les deux classes de visages. Bien que certaines images soient mal classées (1 fois sur 10), la majorité des visages sont correctement identifiés.

On peut tester la robustesse du modèle en ajoutant des variables de nuisance. Cela permet de tester la sensibilité du modèle aux données bruitées. Pour cela on ajoute 300 variables de nuisance suivant une loi normale centrée réduite à chaque échantillon. On réentraîne le modèle SVM avec un noyau linéaire et on évalue la performance sur l'ensemble de test. Bien qu'on obtienne un score de 1 sur les données d'entraînement, on obtient avec les données bruitées un score de **0.5 sur l'ensemble de test**, ce qui est fortement moins bon qu'avec les données non bruitées.

Afin de réduire l'impact des variables de nuisance, on procède par une réduction de dimension sur un sous ensemble d'entrainement des données bruités. On utilise pour cela l'ACP classique. Mais reste à choisir le nombre de composantes principales à garder.

Pour identifier un petit nombre de composantes, on décide d'utiliser la méthode du coude (elbow method) pour choisir le nombre optimal de composantes principales.

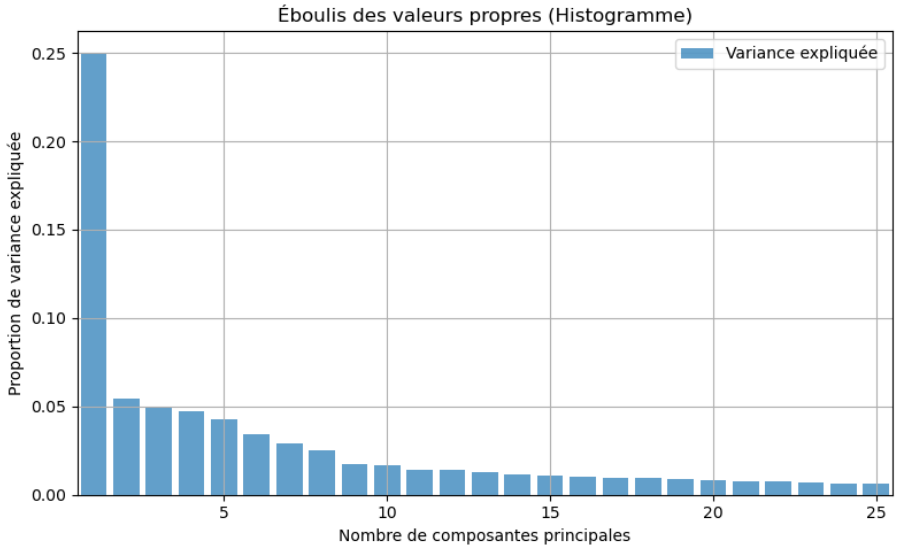

FIGURE 11 – Méthode du coude pour choisir le nombre de composantes principales

Il apparaît plusieurs coudes sur la figure ci-dessus. On se contente de 5 composantes principales pour réduire la dimension des données, ce qui correspond à environ **40 %** . On réentraîne le modèle SVM avec un noyau linéaire. On obtient un score de **0.579** sur **les données d'entrainements** et un score de **0.600 sur les données de test** ce qui est légèrement mieux.

On peut augmenter le nombre de composantes afin d'améliorer la performance du modèle en prenant un nombre arbitraire et grand de composantes. On décide de prendre 50 et 75. On obtient respectivement des scores de **0.626** et **0.815 sur les données d'entraînement**. Mais on observe que les scores sur les **données de test** sont respectivement de **0.511** et **0.495**. On observe que la performance du modèle diminue sur les données de test lorsque le nombre de composantes augmente bien que celui-ci augmente sur les données d'entrainemet. Cela s'explique par le fait que le modèle s'adapte trop aux données qui l'entraîne (sur-apprentissage) et le généralise mal aux nouvelles données.

On pourrait considérer un plus grand nombre de composantes mais ca serait trop coûteux en temps de calcul et n'améliorerait pas forcément la performance du modèle vu que la performance diminue déjà au bout de 75.

Il existe un biais dans le prétraitement des données. En effet, il survient une **fuite** de données (data leakage). Plus précisément, il survient dans le script entre la ligne 215 et 241 :

```
#####################################################################
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, :, :, :]
#####################################################################
```

On normalise les données avant de les séparer en ensembles d'entraînement et de test. Cela introduit un biais car les statistiques (moyenne et écart-type) utilisées pour la normalisation sont calculées sur l'ensemble complet des données et non seulement sur l'ensemble d'entraînement. Les données de test influencent donc la normalisation des données d'entraînement, ce qui peut fausser l'évaluation de la performance du modèle lorsqu'on le teste sur l'ensemble de test.

Ce biais à aussi une repercussion sur l'ACP. Comme on a normalisé les données avant de les séparer puis bruité puis renormalisé encore une fois (mais cette fois que par rapport aux données de train), les composantes principales sont influencées par l'ensemble complet des données, y compris les données de test. Cela peut conduire à une sélection de composantes principales qui ne sont pas représentatives uniquement des données d'entraînement, faussant ainsi la réduction de dimension et la performance du modèle SVM entraîné sur ces données réduites.