Université de Montpellier
Département de Mathématiques Appliquées

TP3 : Support Vector Machine

Apprentissage Statistique — HAX907X

Réalisé par :
ATTOUMANI Ibrahim

STATISTIQUE
SCIENCE DES DONNÉES
UNIVERSITÉ DE MONTPELLIER

Année Universitaire 2025 – 2026

Table des matières

1. Introduction

Les **SVM** (Support Vector Machines), introduits par Vapnik, sont des méthodes de classification très utilisées, en particulier pour la classification binaire. Elles reposent sur la recherche d'une règle de décision linéaire sous la forme d'un **hyperplan séparateur**. Pour traiter des problèmes plus complexes, cette recherche est effectuée non pas directement dans l'espace des données initiales, mais dans un **espace de caractéristiques** de grande dimension, obtenu grâce à une transformation non linéaire.

L'objectif de ce TP est d'appliquer les SVM sur des données réelles et simulées à l'aide de la librairie `scikit-learn` (qui s'appuie sur `libsvm`). Nous apprendrons également à ajuster les **hyperparamètres** et le **choix du noyau** afin de mieux contrôler la flexibilité du modèle.

2. Classification sur les donnée Iris

Dans cette section, nous allons mettre en œuvre un **SVM linéaire** afin de distinguer les classes **1 et 2** du jeu de données *iris*. Pour simplifier le problème, seules les **deux premières variables** seront utilisées. Nous conserverons la moitié des données pour l'**apprentissage** et l'autre moitié pour le **test**, afin d'évaluer la capacité du modèle à bien généraliser.

2.1. SVM linéaire : discrimination entre les classes 1 et 2 d'Iris

Ici nous allons sélectionner les classes 1 et 2 du dataset *iris*, puis conserver uniquement les deux premières variables (longueur et largeur des sépales) afin de simplifier la visualisation et la classification.

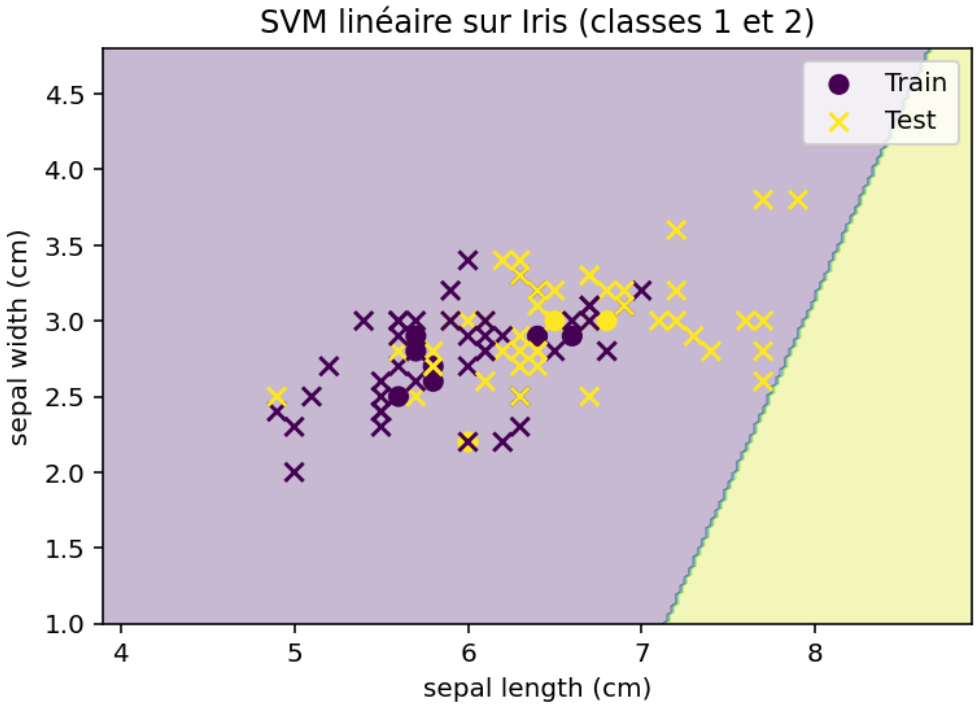

Figure 1: SVM linéaire avec 90% des données réservées au test

Ci-dessus, on a dédiées 90% des données pour le test, ce qui ne laisse qu'environ 10% pour l'entraînement (soit seulement 10 points). Avec si peu d'exemples, le SVM ne dispose pas d'assez d'informations pour bien apprendre. On a alors que la frontière de décision est très approximative et la précision sur le test chute à **0.48**, proche du hasard.

En utilisant 40% des données pour le test, nous obtenons une précision de 7%. Comparée à la **figure 1**, la frontière de décision sépare mieux les points, bien que quelques erreurs de prédiction subsistent. La **figure 2** ci-dessous illustre cette visualisation.

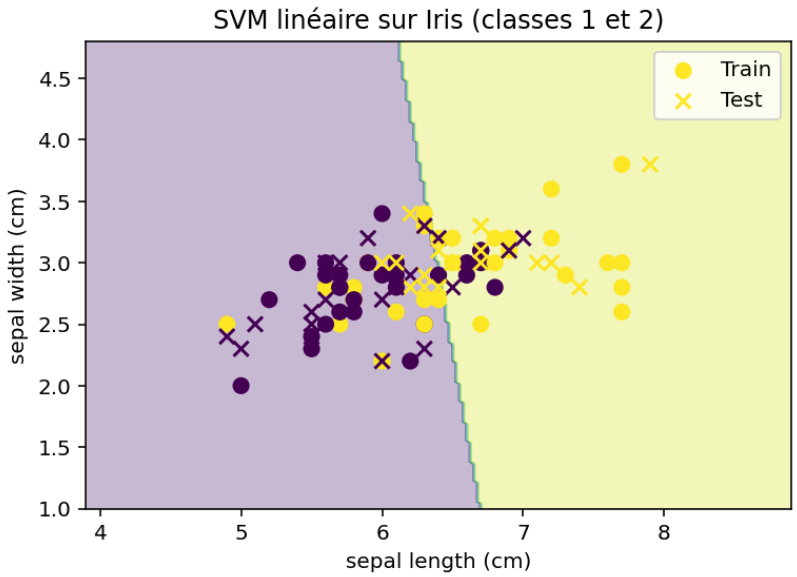

Figure 2: SVM linéaire avec 40% des données réservées au test

2.2. SVM polynomial : discrimination entre les classes 1 et 2 d'Iris

Dans cette section, nous entraînons un SVM avec noyau polynomial pour discriminer les classes 1 et 2 d'Iris. Ce noyau permet de modéliser des frontières de décision non linéaires, plus flexibles que celles du SVM linéaire. Nous comparerons ensuite ses performances et sa frontière de décision avec celles du SVM linéaire présenté précédemment.

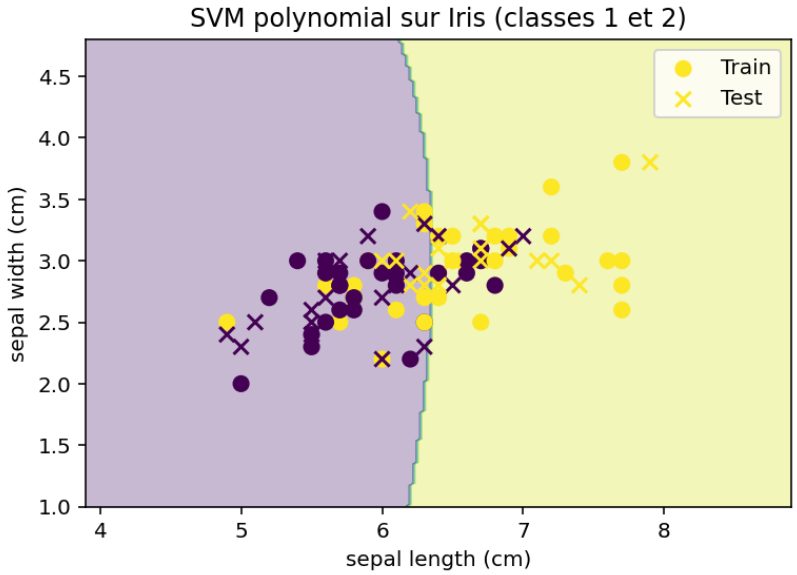

Figure 3: SVM polynomial avec 40% des données réservées au test

À présent, nous comparons la classification linéaire à la classification polynomiale. Cette

dernière ajuste mieux la frontière de décision, avec une précision de 0.75 sur le jeu de test. Comme pour la classification linéaire, 40% des données ont été utilisées pour l'évaluation.

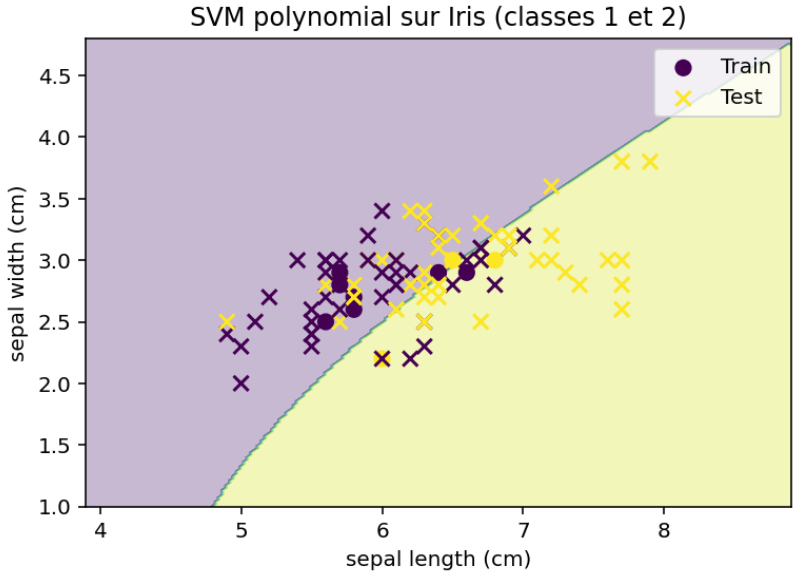

Figure 4: SVM polynomial avec 90% des données réservées au test

Nous avons testé deux modèles SVM (linéaire et polynomial) sur les classes 1 et 2 de l'iris, en utilisant les caractéristiques *sepal length* et *sepal width*.

Avec une grande proportion de données en test (90%), les deux modèles donnent de faibles scores, mais le SVM polynomial s'adapte mieux grâce à une frontière de décision non linéaire.

Avec une répartition plus équilibrée (60% train / 40% test), les performances s'améliorent. Le SVM polynomial reste légèrement meilleur, car il capture mieux la séparation non linéaire entre les classes.

2.3. SVM GUI: données simulées déséquilibré

Cette application svm_gui.py(https://scikit-learn.org/1.2/auto_examples/applications/svm_gui.html) permet en temps réel d'évaluer l'impact du noyaux et du paramètre de régularisation C.

Dans ce qui suit, nous allons générer un jeu de données très déséquilibré avec beaucoup plus de points dans une classe que dans l'autre. Ensuite, nous utiliserons un noyau linéaire tout en réduisant le paramètre C et observerons le résultat.

Lorsque l'on diminue la valeur du paramètre $C = 0,001$, le SVM devient plus tolérant aux erreurs. La frontière de décision se simplifie et ne cherche plus à bien séparer toutes les classes.

Dans un jeu de données très déséquilibré (90% vs 10%), cela entraîne un fort biais vers la classe majoritaire. La classe minoritaire est alors mal reconnue, voire ignorée. La figure ci-dessous illustre la frontière de décision d'un problème déséquilibré:

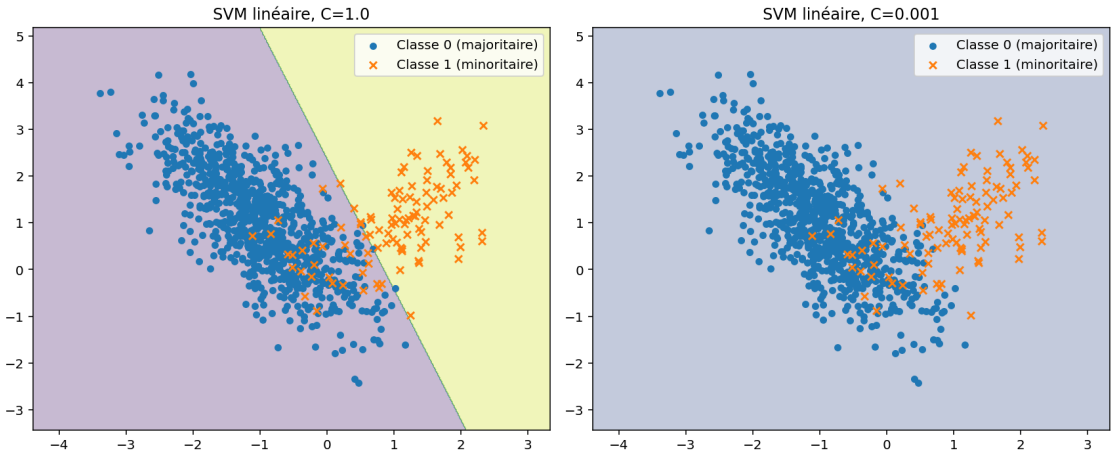

Figure 5: SVM linéaire avec $C = 0.001$ sur données déséquilibrées.

Pour résoudre le problème de données déséquilibrées, nous pouvons utiliser l'option `class_weight="balanced"` dans le SVM. Cette approche permet d'ajouter une correction par pondération, en donnant davantage de poids à la classe minoritaire afin d'éviter qu'elle soit ignorée par le modèle.

L'illustration de ce phénomène, avec et sans correction par pondération, est présentée en annexe.

3. Classification de visage

Dans cette partie, nous allons étudier un problème de classification appliqué à des visages. La base de données utilisée est disponible à l'adresse suivante : https://github.com/atttoum8643f/tp_svm/tree/main/data/olivetti_visages.

L'objectif est d'entraîner un SVM afin de distinguer différents individus, et d'analyser l'impact du paramètre de régularisation C sur les performances du modèle.

3.1. Évaluation de l'influence du paramètre C

La figure ci-dessous illustre l'évolution de l'erreur de prédiction en fonction du paramètre de régularisation C dans un SVM linéaire.
Pour des valeurs très faibles ($C \leq 10^{-4}$), l'erreur est élevée (autour de 45–60%), ce qui traduit un modèle trop régularisé et sous-apprenant.
À partir de $C \approx 10^{-3}$, l'erreur chute fortement et atteint environ 15%.
Ensuite, pour des valeurs plus grandes de C (entre 10^{-2} et 10^{5}), l'erreur se stabilise, indiquant que l'augmentation de C n'apporte plus de gain significatif.

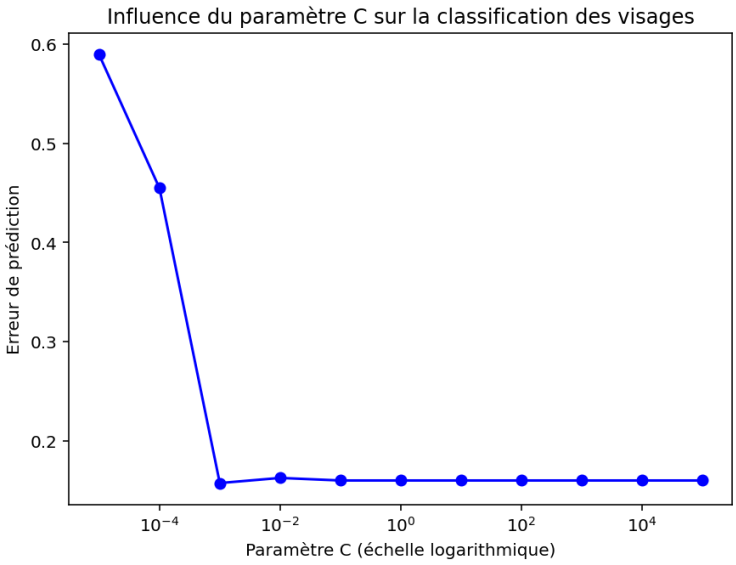

Figure 6: Influence du paramètre C sur la classification de visages

L'introduction de variables parasites dégrade la qualité de la classification. En effet, les résultats obtenus mettent en évidence l'impact des variables de nuisance sur les performances du classifieur.
Sans ajout de bruit, le taux d'erreur est d'environ 0.152, soit une précision proche de 85%. En revanche, lorsque l'on ajoute 500 variables de nuisance, l'erreur augmente à 0196 (soit une précision d'environ 80%).

Néanmoins, le SVM reste relativement robuste puisque la performance globale demeure correcte (voir la fonction **erreur_svm** se trouvant dans **influ_erreur_c.py**).

3.2. Amélioration de la prédiction via un ACP

Afin d'améliorer les performances du classifieur, nous utilisons une réduction de dimension par *Analyse en Composantes Principales* (ACP), implémentée dans `sklearn.decomposition.PCA` avec l'option `svd_solver='randomized'`.

L'ACP avec le critère de Kaiser (**garder que les composantes qui ont des valeurs propres supérieures à un**) a permis de réduire fortement la dimension des données : on passe de **1850 variables initiales à seulement 137 composantes principales**. Cela correspond à une réduction de plus de **90%** de la dimension.

Malgré cette réduction importante, le modèle conserve une **bonne précision (0.770)** et explique une grande partie de l'information contenue dans les données, avec une **inertie expliquée de 93.9%**.

En d'autres termes, on simplifie fortement les données tout en gardant l'essentiel de leur structure et de leur pouvoir explicatif.

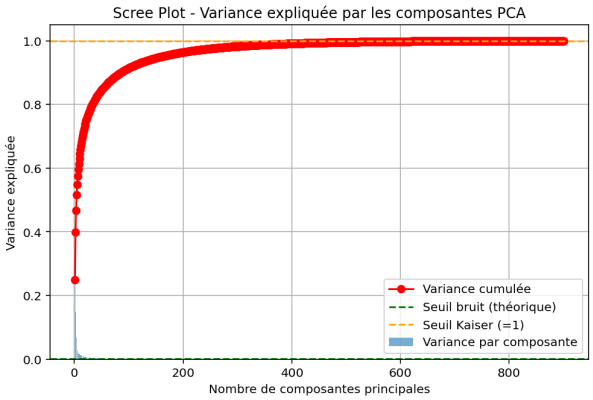

Figure 7: Variance expliquée

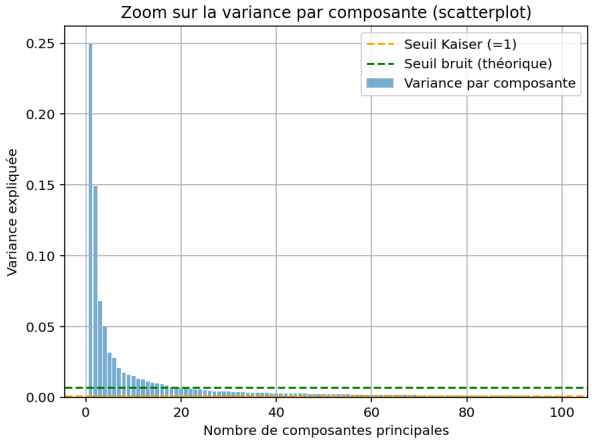

Figure 8: Scatter plot des composantes

Il existe un biais dans notre prétraittement des données de **svm_script.py**, en effet, la normalisation (centrage-réduction) est appliquée **avant** la séparation entre les ensembles d'apprentissage et de test.

```
# Scale features (lignes 224-226)
X -= np.mean(X, axis=0)
X /= np.std(X, axis=0)

# Split data into a half training and half test set
# X_train, X_test, y_train, y_test, images_train, images_test = \
#     train_test_split(X, y, images, test_size=0.5, random_state=0)
# X_train, X_test, y_train, y_test = \
#     train_test_split(X, y, test_size=0.5, random_state=0)
```

Ici, la moyenne et l'écart-type sont calculés sur toutes les données (train + test confondus), ce qui introduit une fuite d'information. La normalisation doit être faite après le split, en utilisant uniquement les statistiques de l'ensemble d'apprentissage.

4. Annexe

Ce code génère un jeu de données binaire fortement déséquilibré (90,% vs 10,%). Deux modèles SVM linéaires sont ensuite entraînés : l'un sans pondération et l'autre avec l'option `class_weight="balanced"`. Enfin, la frontière de décision de chaque modèle est tracée afin de comparer l'effet de la pondération sur la classe minoritaire.

Le code complet utilisé pour générer les comparaisons de frontières de décision est disponible sur GitHub à l'adresse suivante :

https://github.com/atttoum8643f/tp_svm/blob/main/code_py/corr_ponderation.py

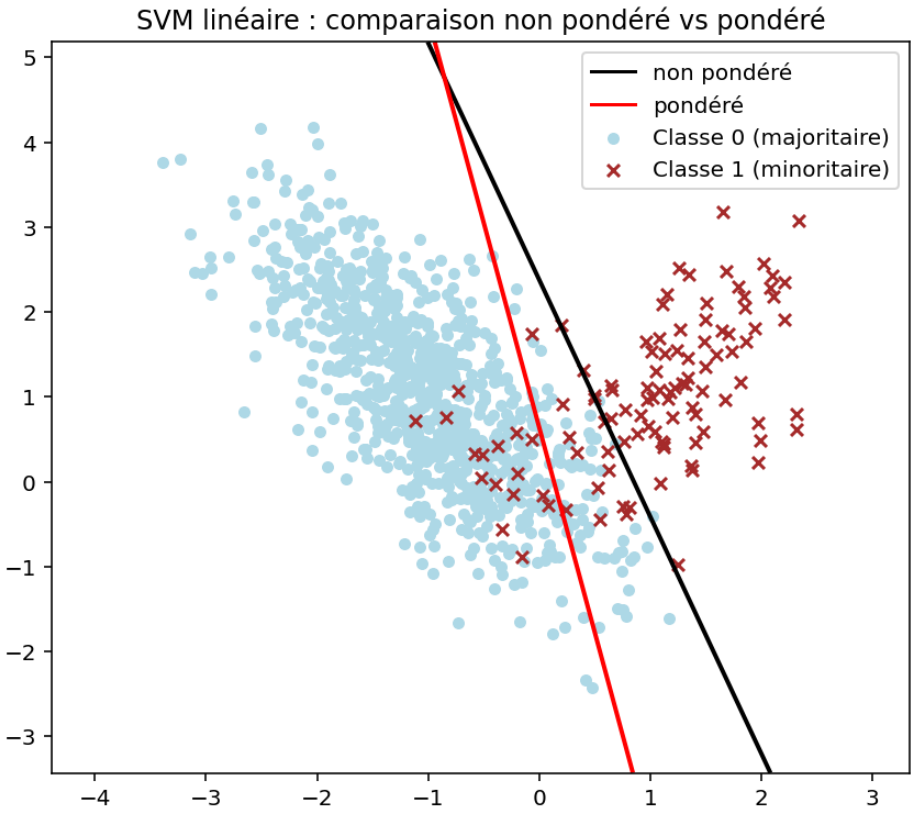

Figure 9: Frontière de décision d'un SVM linéaire avec et sans correction par pondération

References

[1] scikit-learn developers. *Support Vector Machines — scikit-learn documentation*. http://scikit-learn.org/stable/modules/svm.html. Consulté le 24 septembre 2025.

[2] Wikipedia. *Support vector machine*. http://en.wikipedia.org/wiki/Support_vector_machine. Consulté le 24 septembre 2025.

[3] Wikipédia. *Machine à vecteurs de support*. http://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support. Consulté le 24 septembre 2025.