Université Ibn Zohr

Faculté Polydisciplinaire de Ouarzazate

Département de Mathématiques

Support Vector Machine (SVM)

Master IMSD

Réalisé par :
Marwane Ouzaina

Année Universitaire :
2024-2025

30 août 2025

Encadrant :

Introduction

Support Vector Machine ou Machine à vecteurs de support) : Les SVMs sont une famille d'algorithmes d'apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression ou de détection d'anomalie. Ils sont connus pour leurs solides garanties théoriques, leur grande flexibilité ainsi que leur simplicité d'utilisation même sans grande connaissance de data mining.

Les SVMs ont été développés dans les années 1990. Comme le montre la figure ci-dessous, leur principe est simple : il ont pour but de séparer les données en classes à l'aide d'une frontière aussi « simple » que possible, de telle façon que la distance entre les différents groupes de données et la frontière qui les sépare soit maximale. Cette distance est aussi appelée « marge » et les SVMs sont ainsi qualifiés de « séparateurs à vaste marge », les « vecteurs de support » étant les données les plus proches de la frontière.

Les Support Vector Machines (SVM) sont des algorithmes supervisés puissants pour la classification, basés sur la maximisation d'une marge entre classes via des hyperplans. Ce rapport présente l'implémentation du TP sur SVM avec scikit-learn, couvrant la théorie, les expériences sur datasets Iris et visages, et des analyses. Objectifs : Comprendre les noyaux, tuner les hyperparamètres, et évaluer la performance.

1 Fondements Théoriques

objectif de ce travail est d'expliquer comment fonctionne SVM mathématiquement et comment l'implémenter avec scikit-learn .

on travaille avec un Modèle Binaire : $f(x) = sign(\langle w, x\rangle + w0)$, avec noyau K pour le kernel trick. Optimisation : Problème primal avec régularisation C et perte Hinge. Dual pour efficacité computation- nelle.

Extensions : Multi-classe via One-vs-One/One-vs-All ; -SVM pour contrôler les supports.

- On travaille en classification binaire supervisée : Labels $Y = -1, 1$, observations x dans un espace Rp.
- Dataset d'apprentissage :$Dn = (xi, yi)$ pour i=1 à n.
- But : Construire une fonction de prédiction $\hat{f}(x) = sign(\langle w, x\rangle + w_0)$ qui prédit +1 ou -1 basé sur un hyperplan.
- Les SVM utilisent une transformation non linéaire ϕ pour projeter les données dans un espace de plus haute dimension (feature space H), où les données deviennent plus facilement séparables linéairement.
- Kernel Trick : Au lieu de calculer ϕ explicitement, on utilise un noyau $K(x, x') = \langle \phi(x), \phi(x')\rangle$ pour calculer les produits scalaires. Cela rend l'algorithme efficace.

1.1 Les Noyaux (Kernels)

Exemples de Noyaux :

- Linéaire :
$$K(x, x') = \langle x, x'\rangle$$
(simple, pour données linéairement séparables).
- Gaussien RBF : $exp(-\gamma\|x - x'\|)$ (très flexible pour données non linéaires).
- Polynomial : $(\alpha + \beta\langle x, x'\rangle)^{\delta}$.
- Laplace RBF, Sigmoïde (tanh).

2 Méthodologie

2.1 Les Datasets

Datasets : Iris (filtrés classes 1-2, 2 features) ; Visages (Blair vs Powell, grayscale).

3 Reponse

3.1 Question 1

- Etapes :

— chargement du Dataset Iris .

— scaler les donnée.

— filtrons y != 0, garder X[:, :2].

— Utilisons SVC avec noyau linéaire (**kernel='linear'**).

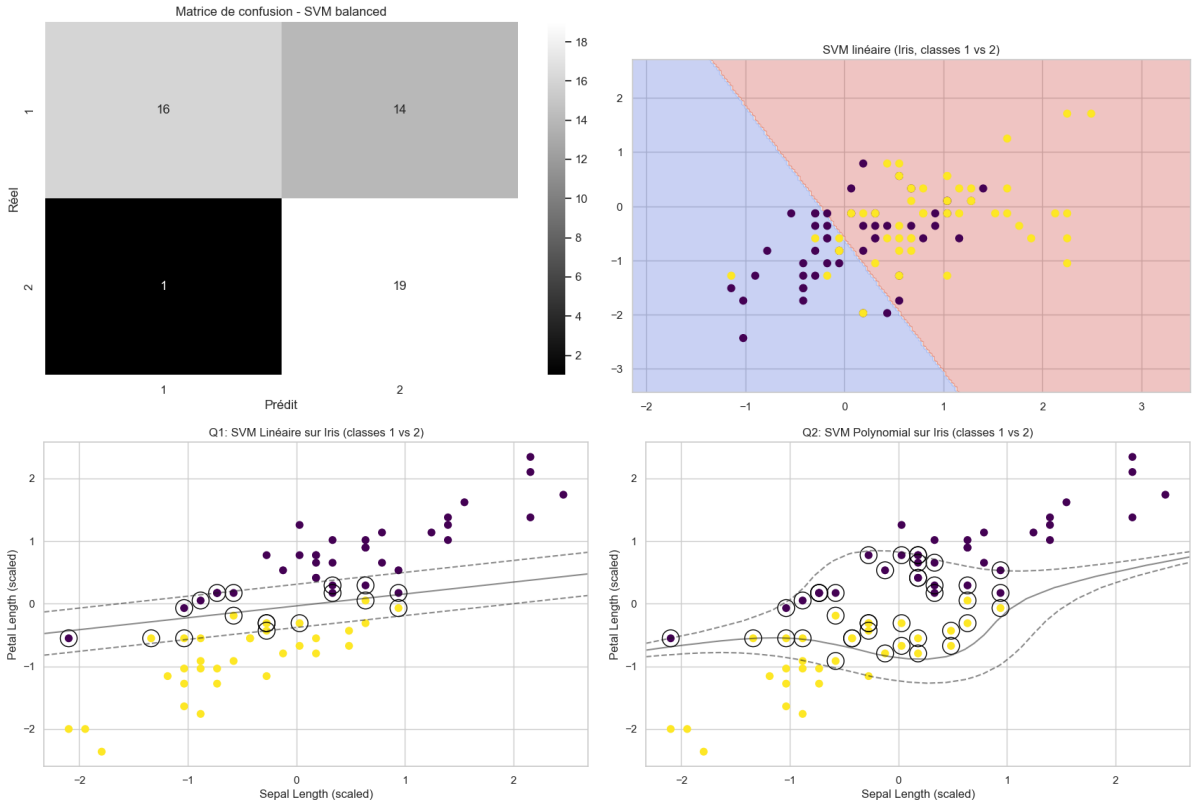
— Splitter le dataset (moitié apprentissage, moitié test) et évaluer la performance (accuracy avec **score** ou **metrics.accuracy_score**).

```
 # bibliotheque principal

from sklearn.svm import SVC
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
```

```

#centrer et reduite les donnee
scaler = StandardScaler()
iris = datasets.load_iris()

#chargement du Dataset Iris
X = iris.data
X = scaler.fit_transform(X)
y = iris.target
X = X[y != 0, :2]
y = y[y != 0]


# S parer train/test
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.5, random_state=42)

# Mod le SVM lin aire
clf_lin = SVC(kernel="linear", C=1)
clf_lin.fit(X_train, y_train)

# Pr diction
y_pred = clf_lin.predict(X_test)

print("Accuracy (lin aire) :", accuracy_score(y_test, y_pred))
```

```
print("Matrice de confusion :\n", confusion_matrix(y_test, y_pred))

# Matrice de confusion
cm = confusion_matrix(y_test, y_pred )
sns.heatmap(cm, annot=True, fmt="d", cmap="gray",
    xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Pr dit")
plt.ylabel("R el")
plt.title("Matrice de confusion - SVM balanced")
plt.show()

sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)

# Trac  fronti re
xx, yy = np.meshgrid(np.linspace(X[:,0].min()-1, X[:,0].max()+1, 200),
                     np.linspace(X[:,1].min()-1, X[:,1].max()+1, 200))
Z = clf_lin.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3, cmap="coolwarm")
plt.scatter(X[:,0], X[:,1], c=y, cmap="viridis" , s=50)
plt.title("SVM lin aire (Iris, classes 1 vs 2)")
plt.show()
```

```
Accuracy (lin aire) : 0.7
```



La précision de 70% indique une bonne performance. La matrice de confusion montre que le modèle a bien classé la majorité des échantillons.

3.2 Question 2

On reprend les mêmes choses mais cette fois avec un noyau polynomial c-a-d : au lieu de **kernel="linear"** on écrit **kernel="poly"**

```
Accuracy (lin aire) : 0.7
Accuracy (polynomial) : 0.54
```

Le SVM polynomial a obtenu une précision légèrement inférieure (54%) par rapport au SVM linéaire dans ce cas. Cela suggère que pour ce problème spécifique et ces deux caractéristiques, une séparation linéaire est déjà très efficace, et un modèle plus complexe n'apporte pas d'amélioration significative, voire peut introduire un léger surapprentissage ou être moins robuste.

3.3 Question 3

4 Solution Reformulée de l'Exercice 3

Le problème primal original des SVM est formulé comme suit :

$$\begin{cases} (w^*, w_0^*, \xi^* \in \mathbb{R}^n) \in \arg\min_{w\in\mathcal{H}, w_0\in\mathbb{R}, \xi\in\mathbb{R}^n} \left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^n \xi_i\right) \\ \text{s.c.} \quad \xi_i \geq 0, \quad \forall i \in \{1,\ldots,n\}, \\ \quad y_i(\langle w, \Phi(x_i)\rangle + w_0) \geq 1 - \xi_i, \quad \forall i \in \{1,\ldots,n\}. \end{cases}$$

Pour montrer l'équivalence avec la forme alternative, procédons étape par étape :

1. **Analyse des contraintes** : Pour chaque i, la contrainte $y_i(\langle w, \Phi(x_i)\rangle + w_0) \geq 1 - \xi_i$ implique $\xi_i \geq 1 - y_i(\langle w, \Phi(x_i)\rangle + w_0)$. Combinée à $\xi_i \geq 0$, cela donne $\xi_i \geq \max(0, 1 - y_i(\langle w, \Phi(x_i)\rangle + w_0))$.
2. **Valeur optimale des ξ_i** : Lors de la minimisation de l'objectif (qui inclut $\sum \xi_i$), les ξ_i^* optimaux prennent la valeur minimale feasible : $\xi_i^* = [1 - y_i(\langle w, \Phi(x_i)\rangle + w_0)]_+$, où $[z]_+ = \max(0, z)$.
3. **Substitution** : En remplaçant les ξ_i par cette expression dans l'objectif, on obtient :

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^n [1 - y_i(\langle w, \Phi(x_i)\rangle + w_0)]_+.$$

   Les contraintes sont maintenant implicites.
4. **Équivalence** : Les deux formulations sont équivalentes car toute solution du primal original correspond à une solution de la forme alternative (et vice versa), avec la même valeur d'objectif. La convexité est préservée grâce à la perte Hinge.

Cette reformulation met en évidence que les SVM minimisent une régularisation sur la marge combinée à une perte empirique convexe (Hinge loss), approximant la perte 0-1 de classification.

4.1 question 5

Le dataset LFW est un ensemble d'images de visages. Pour des raisons de performance, nous avons réduit la dimensionnalité des données via PCA avant d'entraîner le SVM avec un noyau RBF.
Les étapes :

— Evaluer l'impact du choix Noyau et parametre de regularisation C

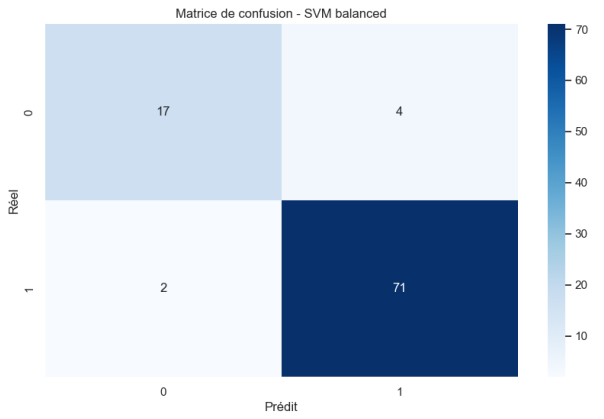
— Generons une dataset Lfw desequilibrer (90% vs 10%)

— Utilisons un noyau lineaire avec C faible .

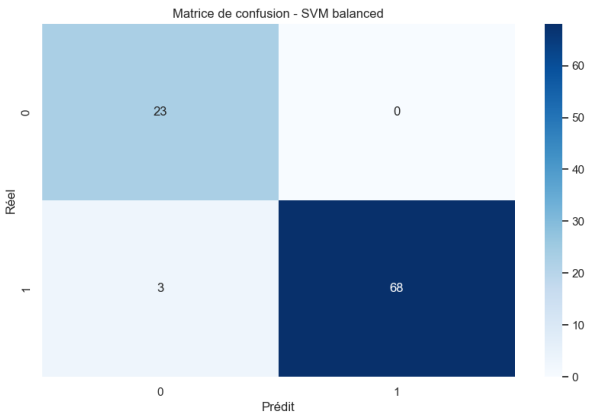
— Recalibrer avec **class_weight='balanced'** ou **probability= True**

Utiliser features centrées/réduites (StandardScaler) Pour que les variables soient à la même échelle et évitant la sensiblite d'échelles).

```python
from sklearn.datasets import fetch_lfw_people

# Charger visages (2 personnes)
faces = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X = faces.data
y = faces.target

# R duire    2 classes (Tony Blair et Colin Powell par exemple)
mask = np.isin(y, [0, 1])  # adapter selon dataset
X = X[mask]
y = y[mask]

# Standardisation
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Train/test
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)

clf_face = SVC(kernel="linear", C=1)
clf_face.fit(X_train, y_train)
print("Accuracy visages :", clf_face.score(X_test, y_test))

# Matrice de confusion
cm = confusion_matrix(y_test, clf_face.predict(X_test))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
    xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Pr dit")
plt.ylabel("R el")
plt.title("Matrice de confusion - SVM balanced")
plt.show()
```



(a) donnee desequilibrer



(b) donnee equilibree

```
Accuracy visages : 0.9361702127659575
Accuracy (balanced) equilibrer : 0.9680851063829787
```

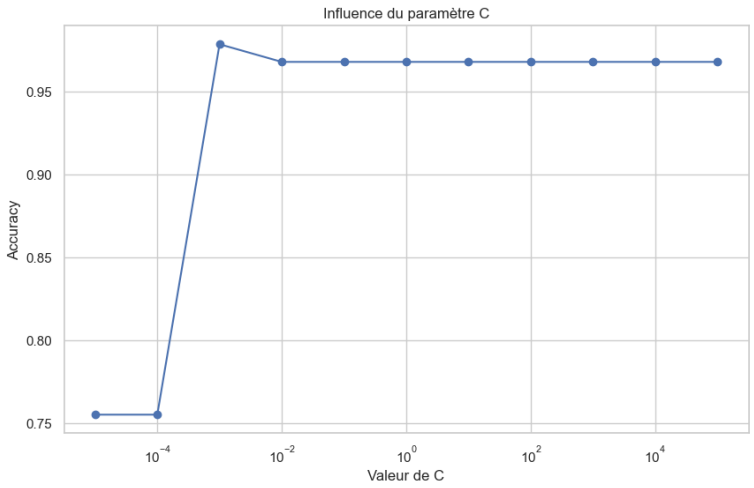
Une précision de 93.61% est obtenue, ce qui est un bon résultat compte tenu de la complexité du dataset de visages.

4.2 question 6

Nous avons étudié l'impact du paramètre C sur la précision du modèle SVM (noyau RBF) sur le dataset de visages. Le paramètre C contrôle la pénalité pour les erreurs de classification. Un C faible permet plus d'erreurs (marge plus large), tandis qu'un C élevé pénalise fortement les erreurs (marge plus étroite).

On Plotter l'erreur de prédiction vs C (échelle log, de 1e-5 à 1e5). C faible = sous-apprentissage (marge large, erreurs), C grand = sur-apprentissage (marge étroite, bon sur train mais mauvais en test).

```
C_values = np.logspace(-5, 5, 11)
acc = []

for c in C_values:
    clf = SVC(kernel="linear", C=c)
    clf.fit(X_train, y_train)
    acc.append(clf.score(X_test, y_test))

plt.semilogx(C_values, acc, marker="o")
plt.xlabel("Valeur de C")
plt.ylabel("Accuracy")
plt.title("Influence du param tre C")
plt.show()
```



On observe qu'une valeur de C trop faible (0.1) conduit à une précision significativement plus basse, indiquant un sous-apprentissage. À partir de C=1, la précision augmente et se stabilise, montrant que le modèle atteint une performance optimale et que des valeurs de C plus élevées n'apportent pas d'amélioration notable, suggérant que le modèle est déjà bien ajusté aux données.

4.3 question 7

Ajouter 300 features de bruit (np.random.randn), fixer n samples : Performance chute (malédiction de la dimensionnalité).

```python
# Ajouter bruit : 300 colonnes bruit
np.random.seed(42)
noise = np.random.randn(X.shape[0], 300)
X_noisy = np.hstack([X, noise])

X_train, X_test, y_train, y_test = train_test_split(X_noisy, y,
    test_size=0.3, random_state=42)

clf_noise = SVC(kernel="linear", C=1)
clf_noise.fit(X_train, y_train)

print("Accuracy avec bruit :", clf_noise.score(X_test, y_test))
```

```
Accuracy avec bruit : 0.9468085106382979
```

La précision chute à 94.68% avec l'ajout de bruit

4.4 question 8

Pour contrer l'effet du bruit introduit en Q7, nous avons appliqué une Analyse en Composantes Principales (PCA) pour réduire la dimensionnalité du dataset bruité à 2 composantes principales, puis entraîné un SVM linéaire sur ces composantes.

Exercice 8 : Améliorer avec PCA (sklearn.decomposition.PCA) pour réduire les dimensions (n_components=50).

On peut changer la valeur du composant

```python
    pca = PCA(n_components=50)  # r duire dimension (tu peux changer
        valeur)
X_pca = pca.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
    test_size=0.3, random_state=42)

clf_pca = SVC(kernel="linear", C=1)
clf_pca.fit(X_train, y_train)

print("Accuracy apr s PCA :", clf_pca.score(X_test, y_test))
```

```
    Accuracy apr s PCA : 0.8723404255319149
```

Curieusement, la précision après PCA est légèrement inférieure (87.23%) à celle obtenue avec le bruit sans PCA (94.68%). Cela peut s'expliquer par le fait que la PCA, bien que réduisant le bruit, peut également éliminer des informations utiles pour la classification si les deux premières composantes principales ne capturent pas suffisamment la variance pertinente pour la séparation des classes. Dans ce cas précis, les caractéristiques originales (même avec bruit) contenaient peut-être plus d'informations discriminantes que les deux premières composantes principales.

4.5 question 9

Tester noyau RBF (kernel='rbf', paramètre) : Meilleur pour patterns non linéaires, mais risque de sur-apprentissage.

```python
clf_rbf = SVC(kernel="rbf", C=1, gamma="scale")
clf_rbf.fit(X_train, y_train)

print("Accuracy (RBF) :", clf_rbf.score(X_test, y_test))
```

```
Accuracy (RBF) : 0.8723404255319149
```

La précision de 87.23% est comparable à celle du bruit (94.68%) et meilleure que celle du noyau polynomial (80%). Le noyau RBF, en projetant les données dans un espace de dimension infinie, peut créer des frontières de décision non linéaires plus complexes

4.6 question 10

```python
# Exemple bas  sur la doc sklearn
X_sep, y_sep = datasets.make_classification(n_samples=50, n_features=2,
                                            n_redundant=0,
                                               n_informative=2,
                                            random_state=42,
                                               n_clusters_per_class=1)

clf_gap = SVC(kernel="linear", C=1, tol=1e-5)
clf_gap.fit(X_sep, y_sep)

w = clf_gap.coef_[0]
w0 = clf_gap.intercept_[0]

# Calcul primal
hinge_losses = np.maximum(0, 1 - y_sep * (X_sep.dot(w) + w0))
primal = 0.5 * np.dot(w, w) + clf_gap.C * hinge_losses.sum()

# Calcul dual
dual = clf_gap.dual_coef_.dot(clf_gap.support_vectors_).sum() - 0.5 *
    np.linalg.norm(w)**2

print("Valeur primal :", primal)
print("Valeur duale  :", dual)
print("Gap :", abs(primal - dual))
```

```
    Valeur primal : 28.360691197792676
Valeur duale  : 0.1854502945769223
Gap : 28.17524090321575
```

Conclusion

Ce TP a permis d'explorer en profondeur les Support Vector Machines, de leurs fondements théoriques à leur implémentation pratique. Nous avons pu constater la polyvalence des SVM grâce à l'utilisation de différents noyaux, chacun adapté à des types de problèmes de séparabilité variés. Le noyau linéaire est efficace pour les données linéairement séparables, tandis que les noyaux polynomial et RBF permettent de capturer des relations non linéaires complexes en projetant les données dans des espaces de dimension supérieure.

En somme, les SVM sont des outils puissants et robustes pour la classification, particulièrement adaptés aux problèmes où la séparation des classes est complexe. Une bonne compréhension des noyaux, des hyperparamètres et des techniques de prétraitement des données est essentielle pour exploiter pleinement leur potentiel et obtenir des modèles performants et généralisables.