TP3 - Support Vector Machine (SVM)

Kenewy Diallo

Introduction

Ce TP a pour objectif de pratiquer les techniques de classification SVM en utilisant le package `scikit-learn`, basé sur la bibliothèque `libsvm`. Nous allons explorer comment ajuster les hyperparamètres et les fonctions noyaux pour optimiser la flexibilité du modèle.

Les morceaux de code ainsi que mes ajouts sont insérés tout au long du rapport.

Partie 1 : SVM sans optimisation

Question 1

Nous avons d'abord séparé le jeu de données `iris` en deux parties, une pour l'entraînement et une pour les tests :

```
X, y = shuffle(X, y)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.5)
```

Listing 1 – Initialisation et entraînement du modèle SVM avec noyau linéaire

```
# Initialisation du mod le SVM avec noyau lin aire
clf_linear = SVC(kernel='linear', C=1.0)

# Entra nement du mod le
clf_linear.fit(X_train, y_train)
```

Le modèle a bien appris à classifier les instances dans les données d'entraînement avec une précision de 72%. Toutefois, il généralise avec une précision de 68,6%, indiquant une légère sur-adaptation. Cela pourrait être amélioré en ajustant le paramètre C.

Données	Précision
Entraînement	72%
Test	68,6%

TABLE 1 – Précision du modèle sur les données d'entraînement et de test.

Question 2

Nous nous retrouvons maintenant avec un score de 0.7 avec un noyau polynomial pour les données d'entraînement et un score de 0.76 pour les données de tests. C'est mieux que pour le noyau linéaire pour les données de tests mais moins bon pour les données d'entraînement.

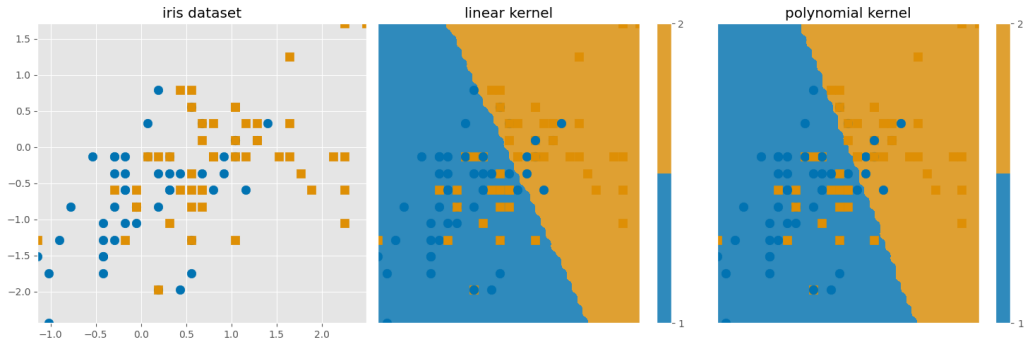

FIGURE 1 – Comparaison entre un noyau linéaire et un noyau polynomial.

Comparaison des noyaux

Il est possible de comparer différents noyaux pour la classification. Ici, nous avons comparé un noyau linéaire avec un noyau polynomial de degrés 1, 2 et 3.

Listing 2 – Comparaison des noyaux linéaire et polynomial

```
degrees = list(map(int, np.r_[1, 2, 3]))
# D finition du grid de param tres
parameters = {'kernel': ['poly'], 'C': Cs, 'gamma': gammas, 'degree': degrees}
# Utilisation de GridSearchCV
clf_poly = GridSearchCV(SVC(), param_grid=parameters, n_jobs=-1)
clf_poly.fit(X_train, y_train)
```

On obtient ainsi les meilleurs paramètres :

- C : 0.0316
- Degree : 1
- Gamma : 10.0
- Kernel : polynomial

Le polynome dégré 1 est le meilleur aprés validation croisée

Partie 3 : Influence du paramètre C

Question 3

Nous avons étudié l'effet de la constante de régularisation C sur l'accuracy du modèle. Voici les résultats obtenus pour différentes valeurs de C :

Valeur de C	1	0.1	0.01	0.001
Accuracy	96	96	96	92

TABLE 2 – Variation de l'accuracy en fonction de C.

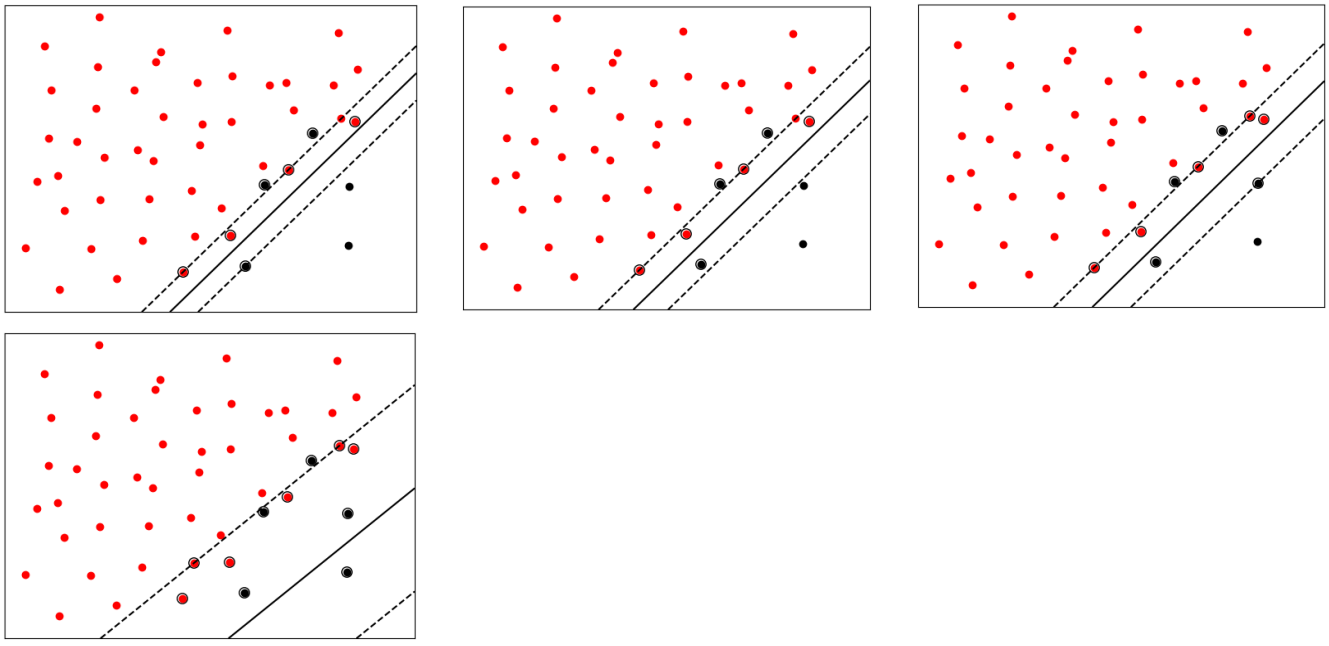

FIGURE 2 – Classification avec noyau linéaire pour différentes valeurs de C.

On constate qu'en diminuant le paramètre C, la marge se déplace jusqu'à devenir incorrecte. En effet, dans la dernière image, on remarque que des points noirs se situent dans la même région que les points rouges, ce qui induit une mauvaise classification.

Question 4 : Classification de visages

Le score d'apprentissage augmente avec la valeur de C jusqu'à atteindre un palier à partir de $C = 10^{-3}$ que l'on va donc considérer comme notre meilleur paramètre.

Listing 3 – Fit d'un classifieur linéaire et test sur différentes valeurs de C

```
Cs = 10. ** np.arange(-5, 6)
scores = []
for C in Cs:
    clf = SVC(kernel='linear', C=C)
    clf.fit(X_train, y_train)
    scores.append(clf.score(X_train, y_train))
```

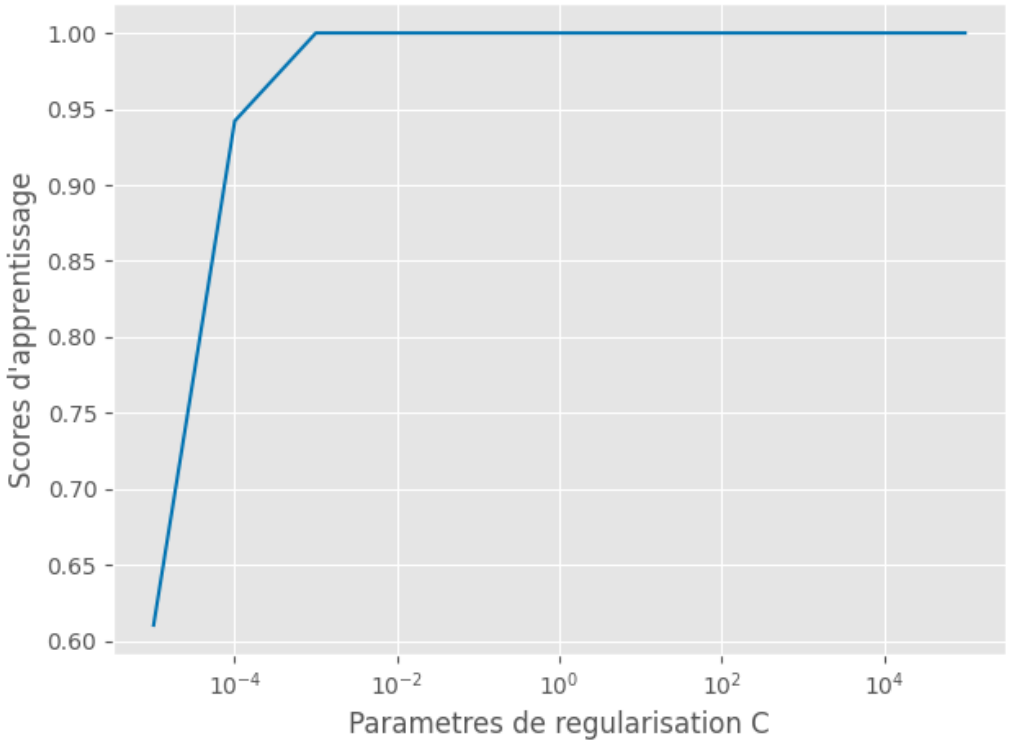

FIGURE 3 – Évolution du score en fonction du paramètre de régularisation C.

On obtient alors les résultats suivants pour la valeur de $C = 10^{-3}$:

- Niveau de chance : 0.6211
- Précision : 0.9105

Listing 4 – Prédiction des étiquettes pour les images de test

```
clf = SVC(kernel='linear', C=Cs[ind])
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

On pourrait aussi tracer la courbe de l'erreur par validation croisée sur le paramètre C.

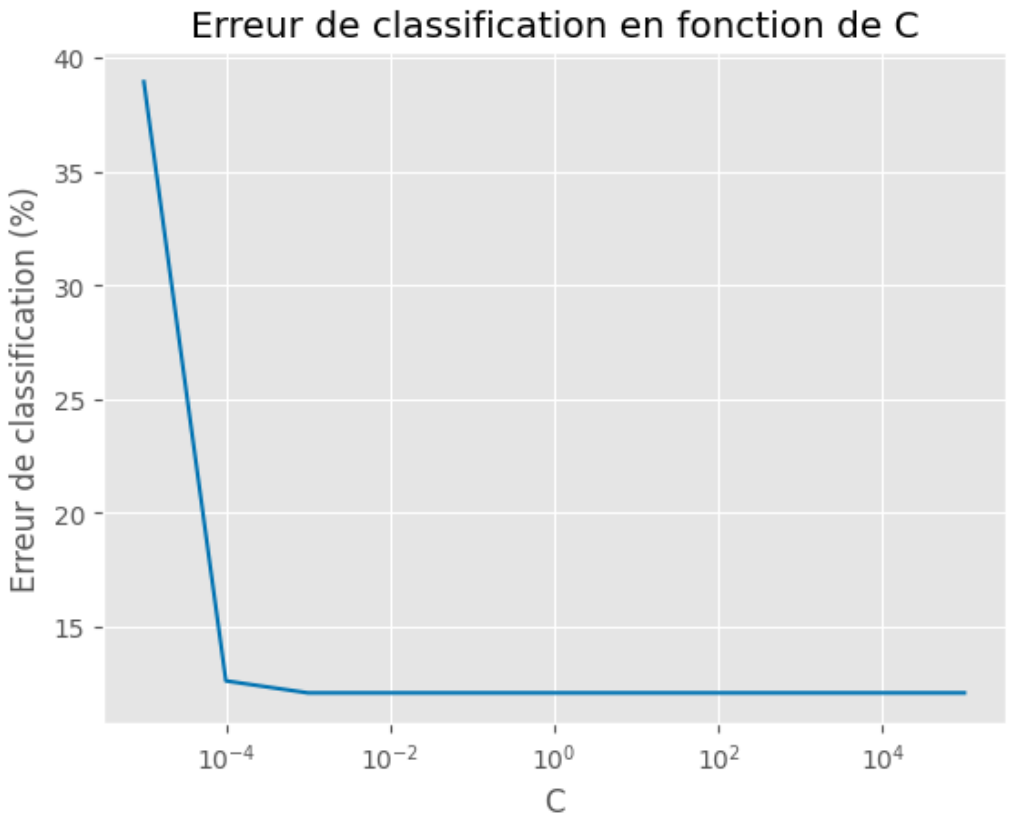

FIGURE 4 – Erreur de prédictions en fonction de C obtenue par validation croisée.

On retrouve la valeur de $C = 10^{-3}$ qui va minimiser l'erreur de prédiction.

Listing 5 – Courbe de l'erreur par validation croisée

```
# courbe de l'erreur par cross validation
from sklearn.model_selection import cross_val_score
err = []
for C in Cs:
    clf = SVC(kernel='linear', C=C)
    scores = cross_val_score(clf, X_train, y_train, cv=5)
    err.append((1 - scores.mean()) * 100)
```

Et pour les images on'a :

FIGURE 5 – Prédiction entre Colin Powell et Tony Blair.

On observe en conséquence sur le jeu de données des images une erreur de une prédiction sur les 12 ce qui correspond bien à une Accuracy de 0, 91.

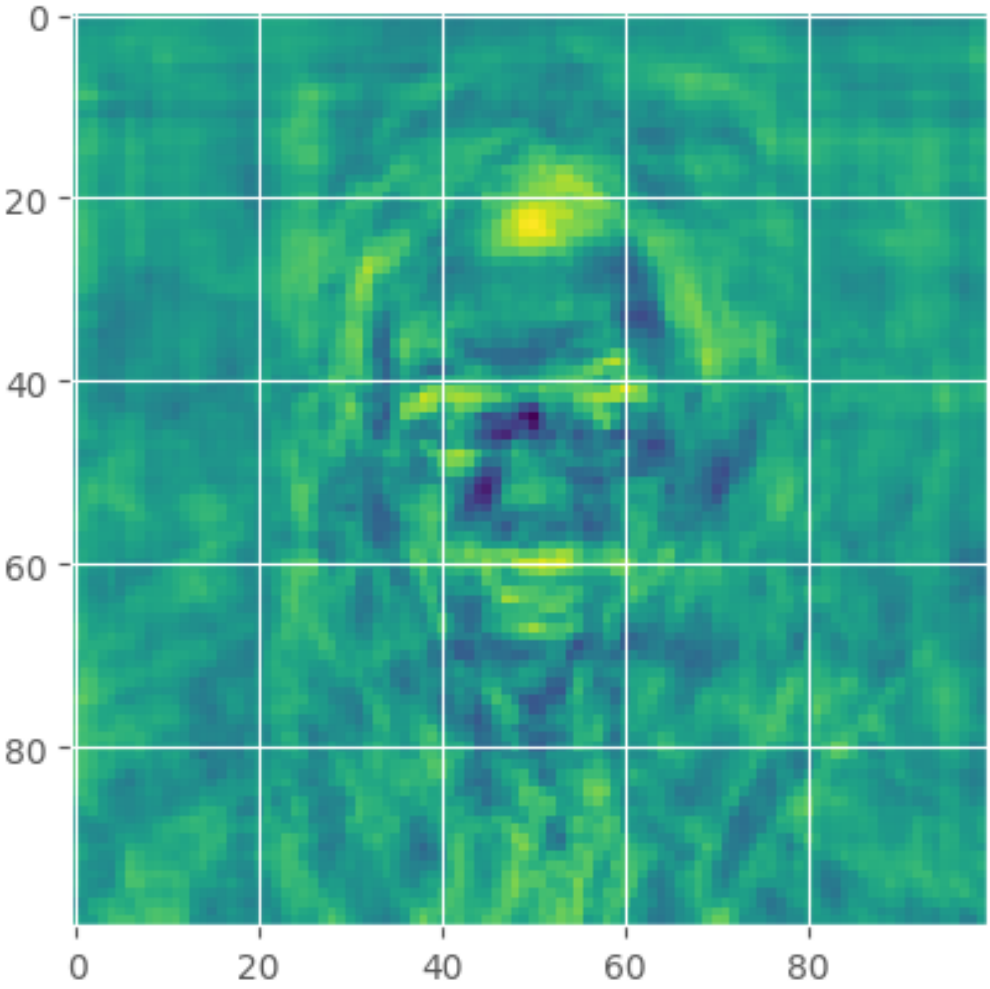

FIGURE 6 – Importance des différents pixels dans la décision.

Sur la figure 6, les pixels aux couleurs les plus vives, qu'elles soient claires ou foncées, se distinguent nettement. Ce sont ces pixels, représentant les variables, qui contribuent le plus à la prédiction lors de la classification.

Question 5 : Introduction de bruit

En introduisant du bruit suivant une loi normale centrée réduite sur notre échantillon, nous constatons que le score diminue drastiquement :

Listing 6 – Impact du bruit sur le score

```
run_svm_cv(X, y)
run_svm_cv(X_noisy, y)
```

— Score sans variable de nuisance : 0.9
— Score avec variable de nuisance : 0.5

Nous ajoutons des variables de bruit au modèle pour évaluer l'impact sur ses performances. Sans nuisance, le score de généralisation est de 0.9 (moyenne de 10 simulations). Avec des variables de nuisance, le score chute à 0.54, indiquant une dégradation significative des performances.
il peut etre represnte graphiquement

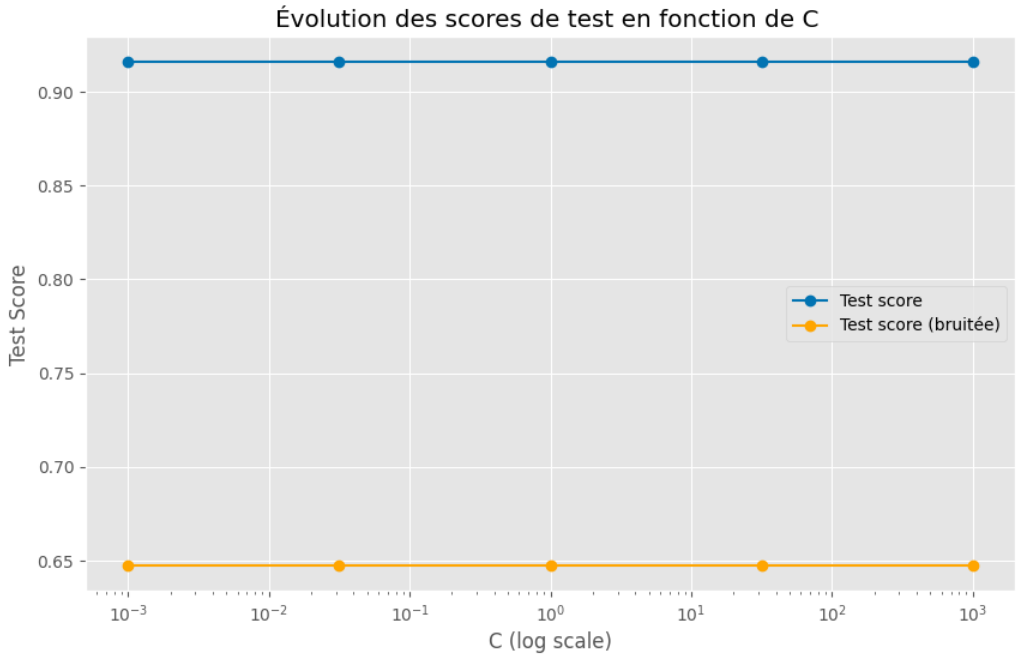

FIGURE 7 – Évolution du score en fonction du paramètre de régularisation C.

Question 6 : Réduction de dimensions
Nous pouvons réduire les dimensions pour améliorer la prédiction en ajustant le paramètre ncAdhérents = 20. On applique ensuite l'ACP sur les données bruitées avec PCA(n$_c omponents = ncAdhrents, svd_s olver =$ "$randomized$"), $puisontransformelesdonnesbruitesavecpca.fit_t ransform(X_n oisy)$
il peut etre represente graphiquement

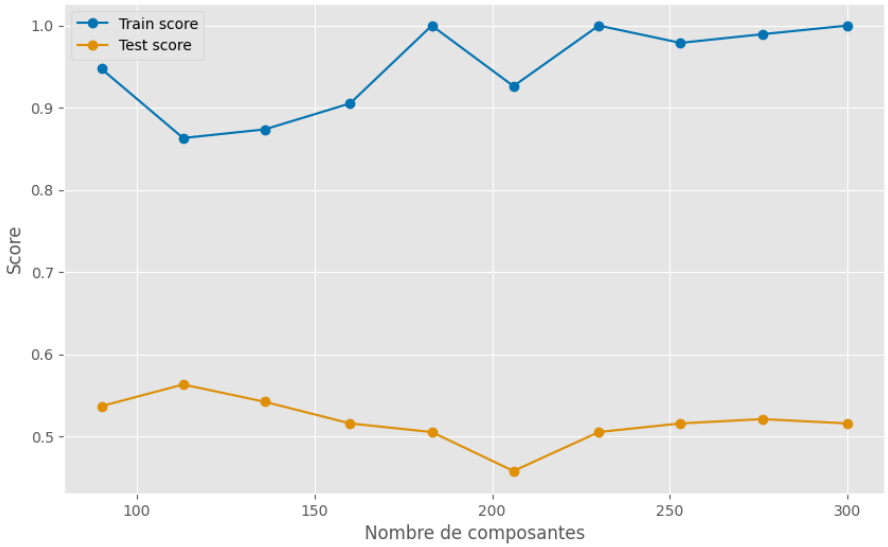

FIGURE 8 – Score en fonction du nombre de composantes de l'ACP

Dans ce cas de figure on arrive pas tirer une ou des conclusions mais neamoins on peut utliser une autre methode c'st a dire calculer le score en jouant sur le nombre de paramétre

Réduction de dimensions avec ACP

En diminuant la dimension de l'espace de départ en cherchant les composantes principales par ACP, voici les résultats obtenus :

Listing 7 – Réduction de dimensions avec ACP

Nombre de composantes	5	10	20
Score d'apprentissage	0.6059	0.6043	0.6570
Score de test	0.6153	0.6364	0.5899

TABLE 3 – Scores d'apprentissage et de test pour des variables bruitées en fonction du nombre de composantes principales.

```
n_components = 20 # jouer avec ce parametre
print(f'Score apres reduction de dimension avec {n_components} composantes principales')
pca = PCA(n_components=n_components).fit(X_noisy)
X_pca = pca.transform(X_noisy)
run_svm_cv(X_pca, y)
```

L'erreur diminue avec certaines composantes principales, mais au-delà de certains rangs , elles semblent ajouter du bruit et dégradent le score. Faut noter qu'aussi ça demande un temps de calcul trés long (donc c'est a utitiser avec precaution)

Conclusion

Ce travail pratique sur les Machines à Vecteurs de Support (SVM) a permis d'explorer différentes approches de classification supervisée en utilisant divers types de noyaux et des techniques d'optimisation des hyperparamètres, telles que la validation croisée. L'étude a montré l'efficacité des noyaux linéaires et polynomiaux, tout en soulignant l'importance du paramètre C dans l'équilibrage entre la précision du modèle et sa capacité de généralisation.
L'ajout de variables de nuisance a entraîné une diminution marquée des performances prédictives, ce qui a été partiellement compensé par l'utilisation de l'Analyse en Composantes Principales (ACP) pour réduire la dimensionnalité. Cependant, cette réduction des dimensions n'a pas totalement résolu le problème de surapprentissage, ce qui souligne la nécessité d'explorer d'autres méthodes plus adaptées aux données bruitées.