Nom et Prénom : BATTACHE Nassim, DJOUAD Iskander.

Master 1 OIVM - Optique Image Vision MultiMedia

Apprentissage supervisé

Abstract :

Ce rapport résume SVM linéairespour l'apprentissage supervisé .

Introduction

Jeu de données Iris

Jeu de données Digits

Conclusion

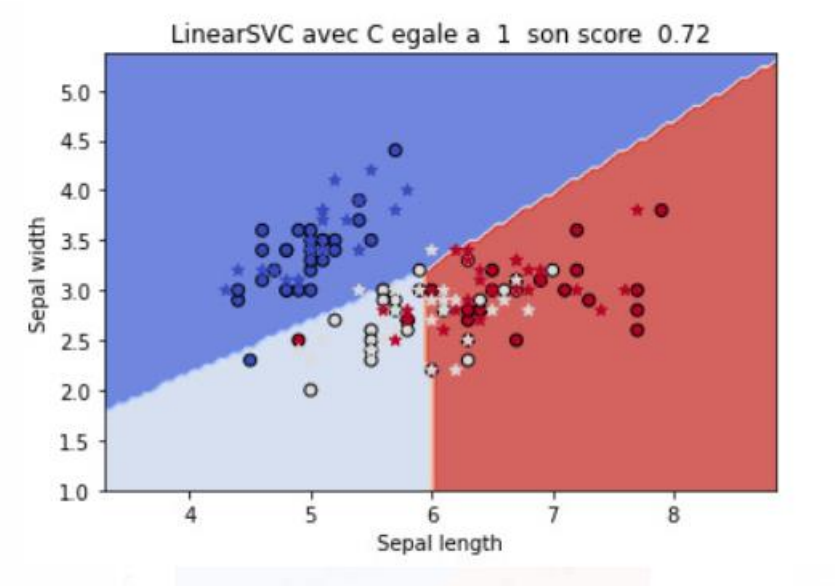
Introduction :

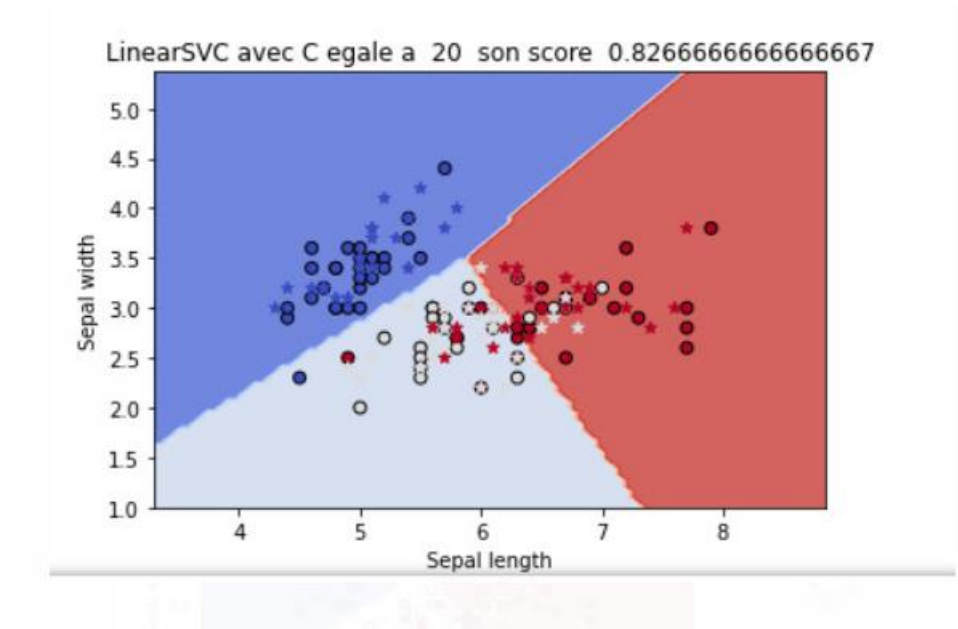
Les machines à vecteurs de support (SVM : *Support Vector Machines*) sont une classe de méthodes d'apprentissage statistique basées sur le principe de la maximisation de la marge (séparation des classes). Il existe plusieurs formulations (linéaires, versions à noyaux) qui peuvent s'appliquer sur des données séparables (linéairement) mais aussi sur des données non séparables.
Les machines à vecteurs de support ou séparateurs à vaste marge (en anglais support-vector machine, SVM) sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression. Les SVM sont une généralisation des classifieurs linéaires.

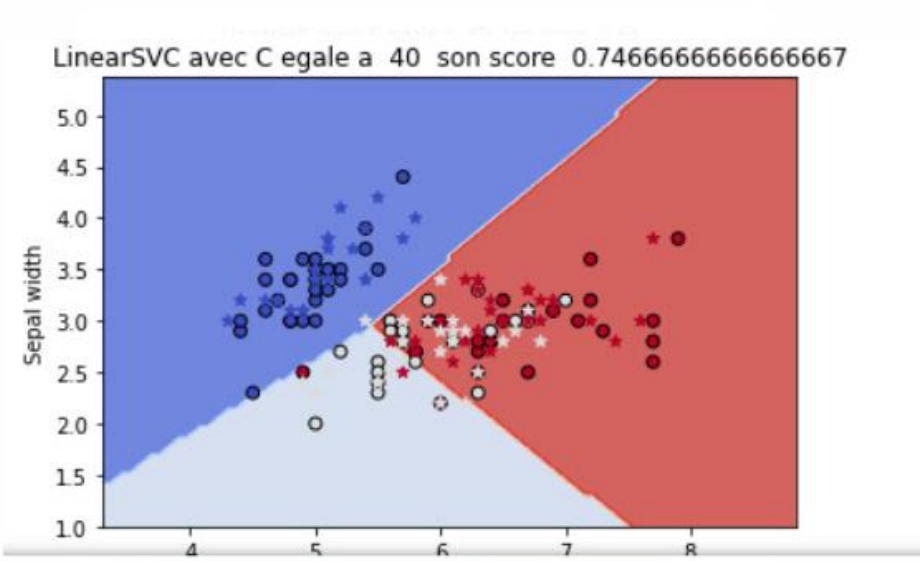
Jeu de données Iris

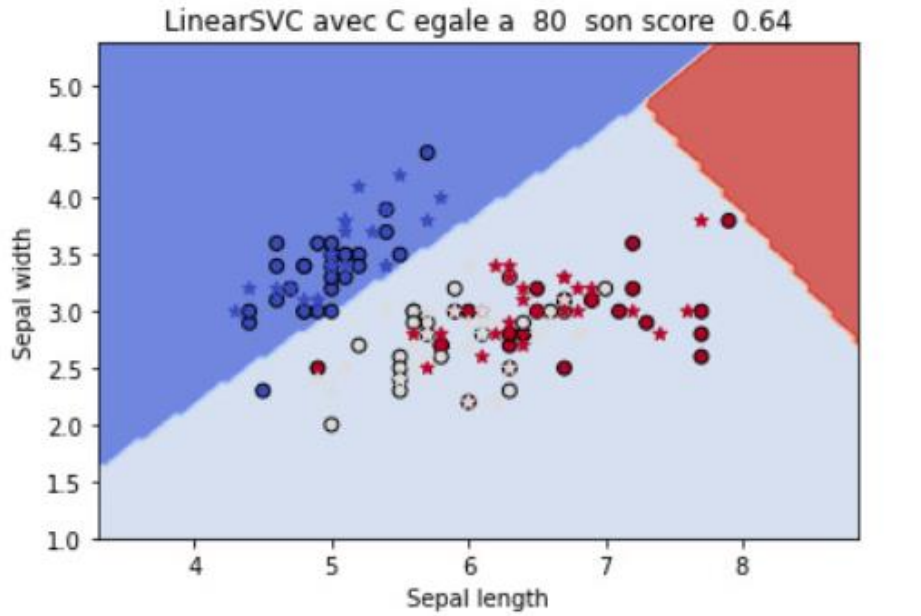
Calculez le score d'échantillons bien classifiés sur le jeu de données de test.

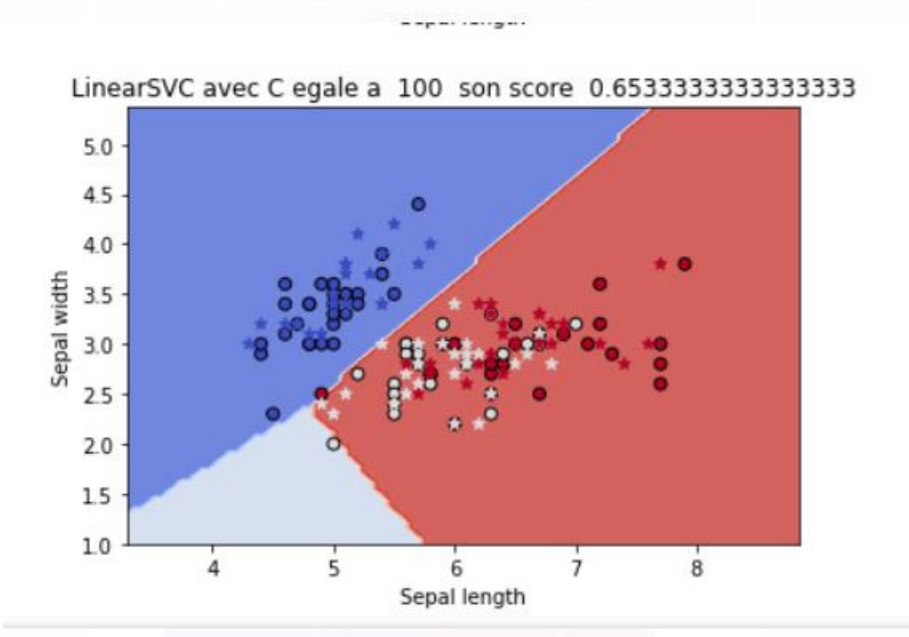
question 1 :






LinearSVC avec C egale a 40 son score 0.7466666666666667
Sepal width
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


LinearSVC avec C egale a 80 son score 0.64
Sepal width
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8
Sepal length



Le paramètre C indique à l'optimisation SVM dans quelle mesure vous voulez éviter de mal classer chaque exemple de formation. Pour les grandes valeurs de C, l'optimisation choisira un hyperplan à marge réduite si cet hyperplan permet de mieux classer correctement tous les points de formation. À l'inverse, une très petite valeur de C amènera l'optimiseur à rechercher un hyperplan de séparation à plus grande marge, même si cet hyperplan classe mal davantage de points. Pour de très petites valeurs de C, vous devriez obtenir des exemples mal classés, souvent même si vos données d'apprentissage sont linéairement séparables.

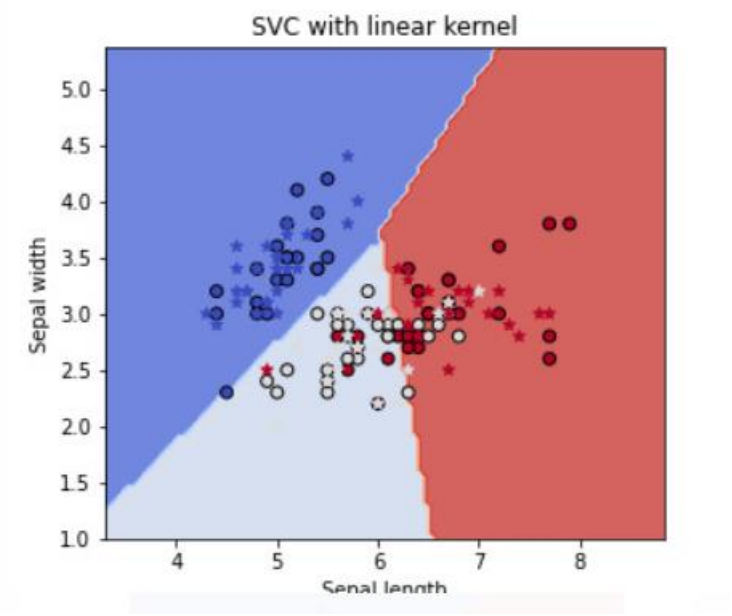
Testez différentes valeurs pour le paramètre C. Comment la frontière de décision évolue en fonction de C ?

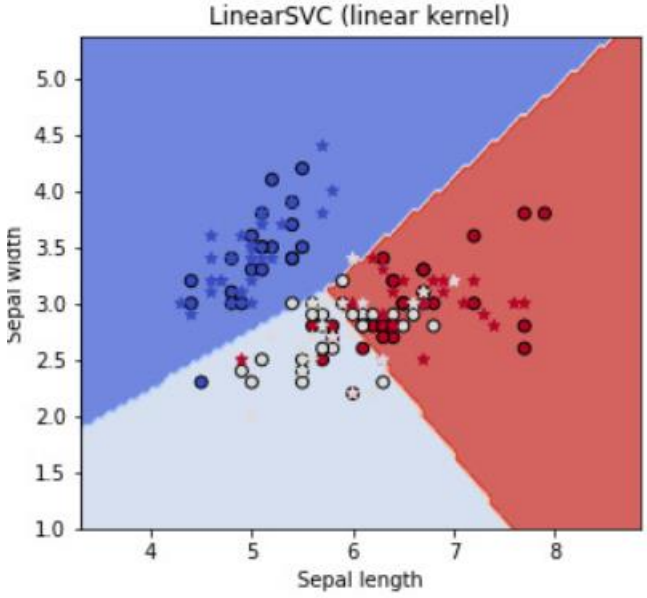
Dans les titres des figures précédentes on a les différentes valeurs .

question 2 :

Le classifieur produit des frontières de décision linéaire. Cela suffit à séparer une des trois classes des deux autres, toutefois en ne considérant que les deux premiers attributs, les deux autres classes ne semblent pas linéairement séparables.

Il faudrait soit utiliser un modèle non linéaire, soit ajouter des attributs supplémentaires en espérant qu'ils permettront de séparer linéairement les deux classes restantes.



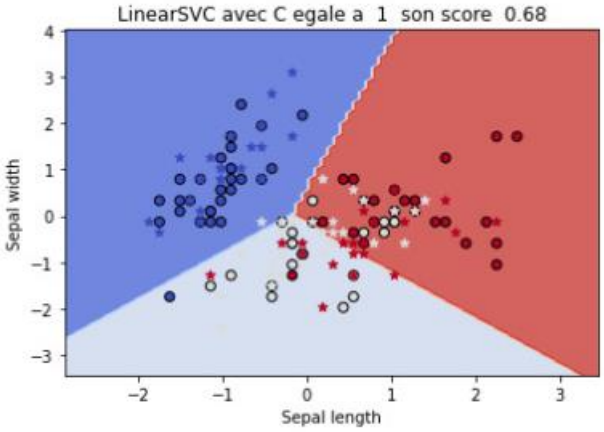


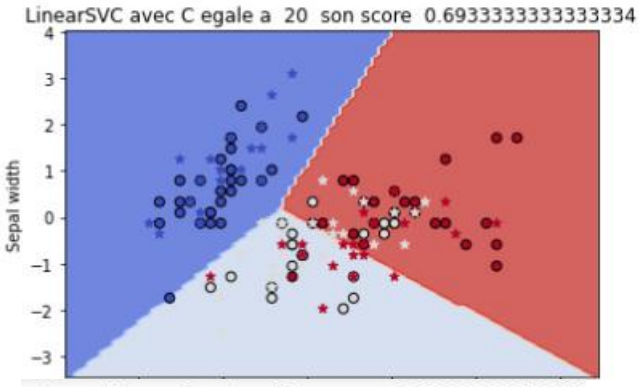
Iris après standardisation :

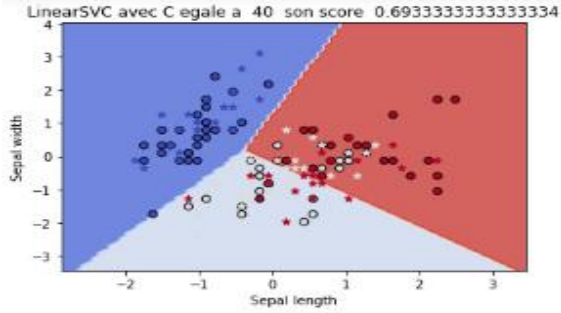
```
C = [1,20,40,80,100]
for C in C:
    lin_svc = svm.LinearSVC(C=C,random_state=0)
    lin_svc.fit(X_train, y_train)
    lin_svc.score(X_test, y_test)
    plt.figure(C)
    # Créer la surface de décision discretisée
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    # Pour afficher la surface de décision on va discrétiser l'espace avec unpas h
    h = max((x_max - x_min) / 100, (y_max - y_min) / 100)
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Surface de décision
    Z = lin_svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    # Afficher aussi les points d'apprentissage
    plt.scatter(X_train[:, 0], X_train[:, 1], label="train", edgecolors='k',
    c=y_train, cmap=plt.cm.coolwarm)
    plt.scatter(X_test[:, 0], X_test[:, 1], label="test", marker='*', c=y_test,
    cmap=plt.cm.coolwarm)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.title("LinearSVC avec C egale a  "+str(C)+"  son score  "+str(lin_svc.score(X_test, y_test)))

    print(lin_svc.score(X_test, y_test))
```

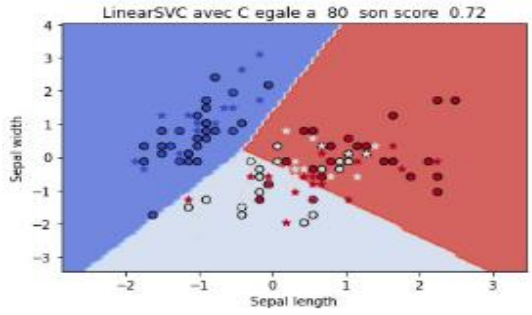
```
X = iris.data
y = iris.target
scaler = StandardScaler()
scaler.fit(X)
X=scaler.transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
C = [1,20,40,80,100]
for C in C:
 lin_svc = svm.LinearSVC(C=C)
 lin_svc.fit(X_train, y_train)
 lin_svc.score(X_test, y_test)
 lin_svc.fit(X_train, y_train)
 print("avec C = "+str(C) +"  on obtien le score suivent  "+str(lin_svc.score(X_test, y_test)))
```

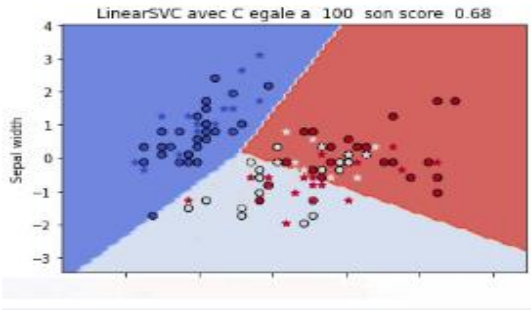
```
avec C = 1  on obtien le score suivent  0.9333333333333333
avec C = 20  on obtien le score suivent  0.9733333333333334
avec C = 40  on obtien le score suivent  0.9733333333333334
avec C = 80  on obtien le score suivent  0.9733333333333334
avec C = 100  on obtien le score suivent  0.9733333333333334
```


LinearSVC avec C egale a 1 son score 0.68
Sepal width
Sepal length


LinearSVC avec C egale a 20 son score 0.6933333333333334
Sepal width


LinearSVC avec C egale a 40 son score 0.6933333333333334
Sepal width
Sepal length


LinearSVC avec C egale a 80 son score 0.72
Sepal width
Sepal length


LinearSVC avec C egale a 100 son score 0.68
Sepal width

Le score augmente de 0,85 à 0,96 car les nouveaux attributs que nous avons ajoutés ont permet de donner un meilleur score en mieux séparant les trois classes.

On remarque qu'à chaque fois que le nombre de features augmente le SVM fonctionne mieux

Après la standardisation le score augmente sauf pour c=20.

Jeu de données Digits

Utilisez les données Digits pour construire un classifieur LinearSVC et évaluez-le. Si le temps d'apprentissage est trop long, sélectionnez une partieplus petite de la base d'apprentissage (par exemple 10000 échantillons).

Pour quelle valeur de C on obtient le meilleur résultat de généralisation ?

```
from sklearn.datasets import load_digits
digits = load_digits()
X, y = digits.data, digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,random_state=0)
C = [1,20,40,80,100]
for C in C:
    lin_svc = svm.LinearSVC(C=C)
    lin_svc.fit(X_train, y_train)
    print("avec C = "+str(C) +"  on obtien le score suivent  "+str(lin_svc.score(X_test, y_test)))
```

```
avec C = 1  on obtien le score suivent  0.92880978865406
avec C = 20  on obtien le score suivent  0.9210233592880979
avec C = 40  on obtien le score suivent  0.9221357063403782
avec C = 80  on obtien le score suivent  0.9199110122358176
avec C = 100  on obtien le score suivent  0.917686318131257
```

On ajoute le paramètre avec loss = 'hinge' :

```
avec C = 1  on obtien le score suivent  0.914349276974416
avec C = 20  on obtien le score suivent  0.9243604004449388
avec C = 40  on obtien le score suivent  0.9254727474972191
avec C = 80  on obtien le score suivent  0.9254727474972191
avec C = 100  on obtien le score suivent  0.9254727474972191
```

On remarque une légère variation de dans les scores.

Maintenant on va standardiser.

```
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
digits = load_digits()

X, y = digits.data, digits.target

scaler = StandardScaler()
scaler.fit(X)
X=scaler.transform(X)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,random_state=0)
C = [1,20,40,80,100]
for C in C:
 scaler.fit(X_train, y_train)

 lin_svc = svm.LinearSVC(C=C)
 lin_svc.fit(X_train, y_train)
 lin_svc.score(X_test, y_test)
 print("avec C = "+str(C) +"  on obtien le score suivent  "+str(lin_svc.score(X_test, y_test)))
```

avec C = 1 on obtien le score suivent 0.946607341490545

avec C = 20 on obtien le score suivent 0.9210233592880979

avec C = 40 on obtien le score suivent 0.9098998887652948

avec C = 80 on obtien le score suivent 0.9187986651835373

avec C = 100 on obtien le score suivent 0.9154616240266963

avec loss = 'hinge'

avec C = 1 on obtien le score suivent 0.946607341490545

avec C = 20 on obtien le score suivent 0.9243604004449388

avec C = 40 on obtien le score suivent 0.9199110122358176

avec C = 80 on obtien le score suivent 0.9165739710789766

avec C = 100 on obtien le score suivent 0.9154616240266963

CONCLUSION :

Ont conclue qu'en standardisant les données, le score d'exactitude évolue.

Cependant, il faut faire attention au cas particulier où le nombre de » features » est beaucoup plus grand que le nombre d'échantillons, car un mauvais le choix des fonctions de noyau peut entrainer l'over-fitting.

SVM est un modèle de machine learning très puissant et polyvalent, capable d'effectuer une classification linéaire ou non linéaire, une régression et même une détection des outliers. C'est l'un des modèles les plus populaires de l'apprentissage automatique et toute personne intéressée par l'apprentissage automatique devrait l'avoir dans sa boîte à outils. Dans cet article, nous ferons une introduction aux SVM et implémenterons un SVM en python.