ESIEA

RUDY LOMBARD – NONO ARMEL TCHIASSO – JOLLIVET JULIEN

2021-2022

TP 1

MACHINE LEARNING

SUPPORT VECTOR MACHINES & DECISION TREE

A– GENERATION DES DONNEES

Afin de pouvoir générer une première base de données synthétiques (B1), nous allons utiliser la fonction suivante :

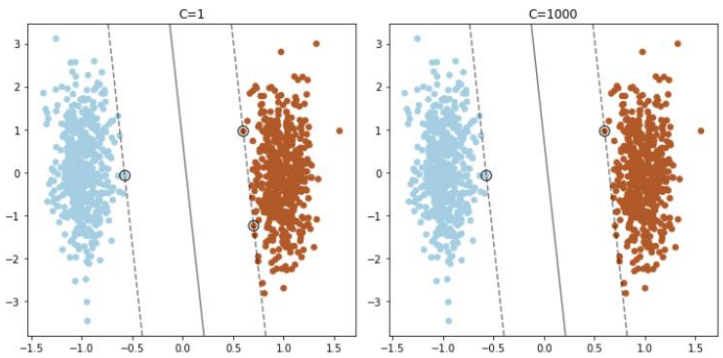
```
X,y = make_blobs(n_samples=1000, n_features=2, centers=2, cluster_std=1,
center_box=(-10.0, 10.0))
```

B– SVM LINEAIRE

a. Données linéairement séparables :

1) Tout d'abord, nous savons que les données sont linéairement séparables. Donc on définit notre SVM à marge stricte grâce à la fonction LinearSVC(). Dans cette fonction, nous définissons un paramètre C qui permet de réduire ou augmenter la marge maximale. Ici, nous allons mettre une valeur élevée pour avoir une marge stricte et donc un nombre de supports vecteur minimales en réduisant la marge.

2) On constate un taux d'erreur nul avec une matrice de confusion qui nous indique que chaque exemple prédit est bien dans sa classe réelle. On voit visuellement que les données sont très bien séparées (voir graphe ci-dessous). Sur une marge souple (C=1), le nombre de supports vecteur est supérieur à celle d'une marge stricte. On cherche à minimiser le nombre de supports vecteur donc on préfère la seconde représentation.



3) Les SVM permettent d'avoir une convergence assurée vers le minimum global et limite le risque de sur-apprentissage. Pour les réseaux de neurones, un seul neurone permettrait de bien séparer nos données.

4) Par conséquent, le SVM permet de définir la frontière de décision et de séparer à vaste marge les vecteurs support. L'utilisation d'un neurone aurait les mêmes performances que l'utilisation d'un SVM dans ce cas.

b. Données non linéairement séparables

1)
```
#On augmente l'écart-type au sein de notre distribution
X,y = make_blobs(n_samples=1000, n_features=2, centers=2, cluster_std =5.0, center_box=(-10.0,10.0))

X0 = (X[:,0])
X1 = (X[:,1])

#Normalisation des données
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

#Séparation des données en train et en test
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
```

2) Afin d'entrainer un modèle, nous devons évaluer ses performances sur des données qu'il n'a jamais vues, or il est nécessaire de pouvoir modifier nos hyper paramètres en se basant sur une base de test, la cross-validation intervient à ce niveau en implémentant à l'intérieur de la base d'apprentissage une base nommée base de validation qui aura pour but d'évaluer le modèle et de modifier ses hyper paramètres afin de d'avoir une meilleure accuracy. La base de test sera par conséquent inutilisée jusqu'à la fin et nous permettra d'évaluer notre modèle sur des données qu'il n'aura jamais vues.
Il faut donc entrainer notre modèle sur la base d'apprentissage, évaluer ses performances sur la base de validation, rectifier les hyperparamètres et enfin évaluer le modèle sur la base de test.

Nous choisissons 5 splits, cela signifie que nous prenons 30% de notre base d'apprentissage et que nous évaluons le model sur la base de validation, nous répétons l'opération 5 fois de manière à tester le model sous toutes ses configurations possibles, l'évaluation se fera sur la moyenne des 5 résultats obtenus.

3)

	Accuracy_train	Accuracy_test
C=1	70	68
C=100	70	70
C=1000	59	66
C=10000	59	66
C=100000	59	66

4) On remarque que la capacité des SVM linéaires à séparer des données non linéairement séparables est limité, nos valeurs ici manquent de features pertinentes afin de pouvoir les classifier avec un classifieur SVM linéaire.
On remarque que pour une marge souple comme pour une marge complexe, l'accuracy en test n'est pas élevée due à des hyperparamètres non significatifs pour la séparation de nos données.

C– SVM NON LINEAIRE A NOYAU GAUSSIEN

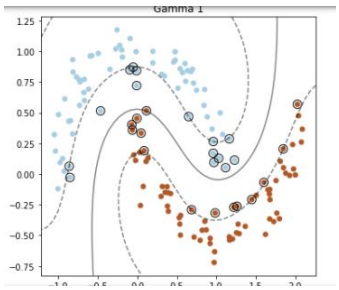
1)

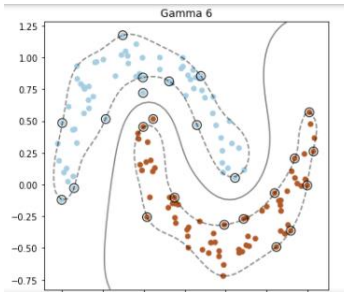
```
X, y = make_moons(noise=0.1, random_state=1, n_samples=200)

X0 = (X[:,0])
X1 = (X[:,1])


#Normalisation des données
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

#Séparation des données en 70% train et 30% test
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.3, random_state=0)

plt.scatter(X_norm[:,0], X_norm[:,1] , c=y, s=30, cmap=plt.cm.Paired)
```

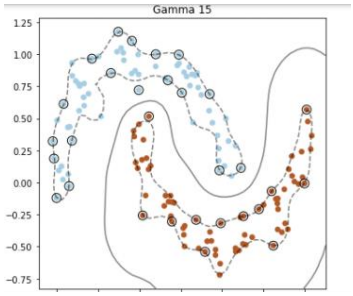
2)a)

```
#Initialisation de gamma en échelle linéaire
gamma = [1,3,6,9,12,15]

#On fixe C à 1
for i, C in enumerate([1]):
    #On itère pour nos valeur de gamma
    for j in gamma:

        clf = svm.SVC(kernel = "rbf", gamma=j, C=C).fit(X_train, y_train)
        decision_function = clf.decision_function(X_train)
```

b)





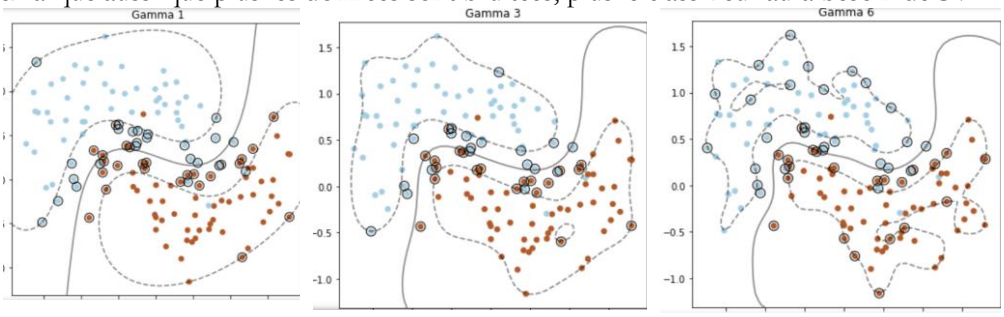


	Taux erreur Train	Taux erreur Test	Nombre de SV
Gamma = 1	0	1	29
Gamma = 3	0	0	22
Gamma = 6	0	0	24
Gamma = 9	0	1	25
Gamma = 12	0	1	26
Gamma = 15	0	1	30

(Taux erreur est en pourcentage)

c) Ici, on s'aperçoit que nos exemples sont très bien classifiés avec une dispersion du gamma allant de 1 à 15, les supports vecteurs augmentent avec le nombre de gamma, ici on préfèrera limiter le nombre de SV en choisissant un gamma = 3 ou 6. On observe que plus le gamma augmente, plus nos frontières se rapprochent de nos points et les augmentent, nous risquons le sur-apprentissage en augmentant gamma. On peut en conclure que le plus le gamma augmente, plus le biais diminue mais la variance augmentera (overfitting), et plus celui-ci est faible, plus le biais sera élevé et la variance sera faible (underfitting).

3) On observe là aussi que plus gamma augmente, plus nos frontières se rapprochent des points ce qui signifie que le biais diminue mais que la variance augmente (overfiting). On remarque aussi que plus les données sont bruitées, plus le classifieur aura besoin de SV.



(Cas avec gamma = 1, gamma = 3 et gamma = 6)

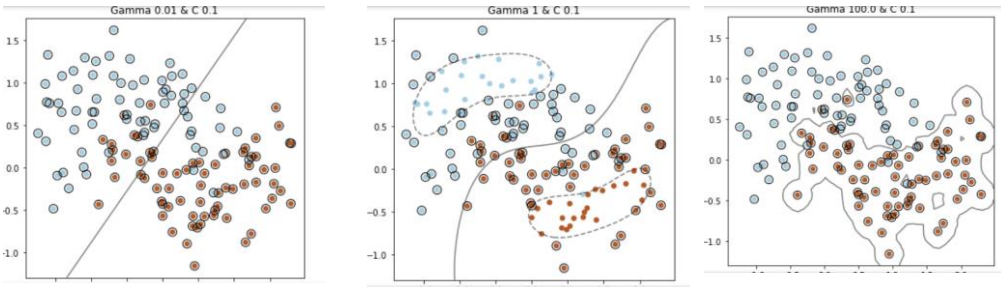
Nous observons ici que plus le gamma est élevé plus nos frontières se rapprochent des points, on augmente donc le sur-apprentissage : biais faible mais variance élevée. D'autre part, on remarque que plus Gamma augmente, plus le nombre de support vectors augmente aussi (corrélation positive 90%) ce qui s'explique par la complexité des frontières avec l'augmentation de Gamma.

	Taux erreur Train	Taux erreur Test	Nombre de SV
Gamma = 1	8	8	49
Gamma = 3	6	6	39
Gamma = 6	5	6	49
Gamma = 9	5	6	52
Gamma = 12	5	6	58
Gamma = 15	5	1	71

(Taux erreur en pourcentage)

4)

a)




```
Matrice de confusion pour la base tr
[[56 14]
 [14 56]]

Matrice de confusion pour la base t
[[22  8]
 [10 20]]
```

```
Matrice de confusion pour la base tra
[[62  8]
 [ 9 61]]

Matrice de confusion pour la base
[[26  4]
 [ 6 24]]
```

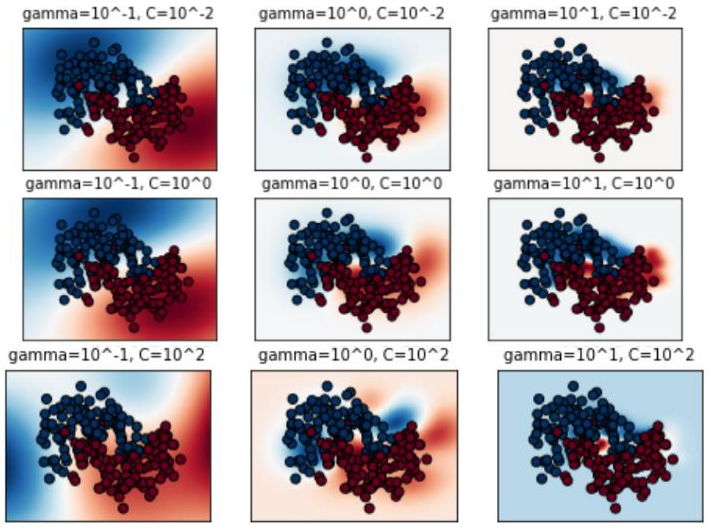
```
Matrice de confusion pour la base tra
[[67  3]
 [ 1 69]]

Matrice de confusion pour la base t
[[28  2]
 [ 7 23]]
```

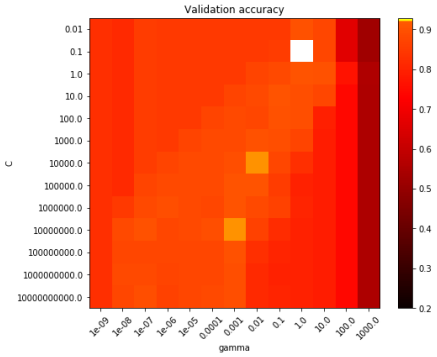
	G = 0.01 & C = 0.1	G = 1 & C = 0.1	G = 100 & C = 0.1	G = 0.01 & C = 1	G = 1 & C = 1	G = 0.01 & C = 10	G = 1 & C = 10	G = 100 & C = 10
% Taux erreur Train	49	9	0	18	5	17	5	0
% Taux erreur Test	51	13	16	12	8	12	9	23
Support Vectors	138	103	79	93	42	58	27	87

On fait évoluer gamma de 0.01 à 100 sans changer le paramètre C qui est égal à 0.1. On observe que quand le paramètre gamma augmente, la frontière de décision gagne en précision avec un risque de sur-apprentissage lorsque celui est élevé (car frontière trop précise).

b) Sur la grille, on observe que les meilleures valeurs de Gamma et C en Validation sont G = 1 et C = 0.1, on en déduit qu'il n'est pas nécessaire d'appliquer une marge stricte afin de séparer des données bruitées, il n'est pas nécessaire d'appliquer un grand Gamma aussi. On remarque que plus Gamma et C sont grands, plus notre modèle est en sur-apprentissage (biais faible et variance élevée). Nous observons aussi que plus la marge est large, plus le nombre de SV est important, ce qui est normal selon nous car plus le classifieur utilise des données éloignées afin de séparer des données, plus celui-ci prendra en compte des points qui se rapprochent de la marge.

On en conclut que dans le cas d'un schéma de données en projection « croissant de lune » dont le but est de séparer 2 classes avec un bruit élevé, l'utilisation d'un SVM Gaussien est efficace lorsque la marge est souple avec une faible valeur de gamma, ces paramètres n'ont pas besoin d'être très grands pour ce cas.





D– ARBRES DE DECISION

1)

```
#Charger les données
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

#Normalisation des données
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

plt.scatter(X_norm[:,0], X_norm[:,1],X_norm[:,2] ,c=y, cmap=plt.cm.Paired)
#Séparation des données en train et en test
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
```

5

2)

b) max_depth : Ce paramètre permet d'approfondir les couches de l'arbre de décision, plus il sera haut, plus les séparations entre features sera possible.

	Max_depth = 1	Max_depth = 2	Max_depth = 3	Max_depth = 4	Max_depth = 5	Max_depth = 6
% Accuracy en Train	65	90	94	93	91	94
% Accuracy en Test	64	98	95	95	95	95

Lorsqu'on autorise l'arbre de décision à n'avoir qu'une seule couche, on s'aperçoit que les résultats sont faibles, avec peu de variance et un haut biais -> Sous-apprentissage.

Lorsqu'on autorise l'arbre de décision à avoir deux et trois couches, on s'aperçoit que les résultats de classifications sont meilleurs avec peu de variance entre nos résultats et peu de biais -> Bon apprentissage.

Lorsqu'on autorise l'arbre de décision à avoir plus de 3 couches, on s'aperçoit que les résultats de classifications sont corrects mais avec un peu plus de variance, le biais augmente -> Risque de sur-apprentissage difficile à conclure ici.

c) min_samples_split : Ce paramètre permet de mettre un critère sur la séparation des données entre 2 couches, si le nombre d'échantillon est inférieur au critère, il n'y a pas de classification par la suite.

	Min_samples_split = 2	Min_samples_split = 5	Min_samples_split = 10	Min_samples_split = 20	Min_samples_split = 30	Min_samples_split = 35
% Accuracy en Train	94	93	92	92	90	91
% Accuracy en Test	95	95	95	98	64	28

On observe qu'avec un haut critère (30) dans notre cas, que les scores d'accuracy sont très faibles. On observe aussi une faible variance dans ce cas, on en déduit qu'un haut critère permet de réduire la variance.

On observe de meilleurs résultats avec un critère fixé à 20. On observe qu'avec un plus petit critère, il est possible d'augmenter le nombre de couches car nous nous retrouvons dans certains cas avec des samples faibles.

d) On remarque dans notre cas que pour limiter le sur-apprentissage, il est nécessaire d'avoir une profondeur d'arbre pas très élevée avec un critère de split plus haut.

Avec un faible critère de split et une grande profondeur d'arbre nous risquons le sur-apprentissage.

Avec un critère de split trop haut et un nombre de couche trop bas, le modèle est en sous-apprentissage.

Afin d'avoir un bon modèle, il est nécessaire d'équilibrer notre critère de split et la profondeur d'arbre dans le but de limiter sous et sur-apprentissage.

3) On observe que pour 6 couches de profondeurs et 2 échantillons de split minimum, les résultats en validation sont de 95% d'accuracy contre 97% en test. Par conséquent, nous avons un modèle bien entraîné et bien évalué.