

TP 1 - Part. 2

Support Vector Machines & Decision Trees

MADAD Sarra

NOUAR Manelle

BERNARDOU Eliott

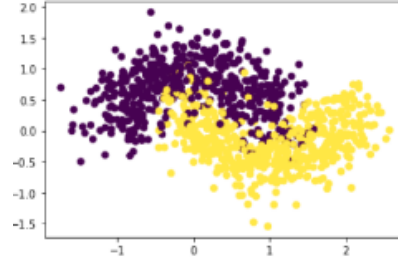
Enseignant : M. Lionel PREVOST

Matière : Machine Learning & Application

Introduction

Nous commençons par créer un jeu de données aléatoires binaires : nous avons 1000 observations pour 2 features. On ajoute du bruit car si on ne le met pas ou si on le laisse à 0, les données sont trop bien séparées (même non linéairement). Il sera donc difficile pour un SVM non linéaire de ne pas bien généraliser.

Elles sont donc non linéairement séparables, avec un bruit de 0.28.



Pas besoin de normaliser nos données (les centrer-réduire) ni de faire une PCA puisque nous n'avons que 2 descripteurs.

Le but ici est d'essayer de bien séparer ce problème bi-classes, à l'aide de modèles non linéaires adéquates.

Modèles utilisés :

- SVM non linéaire à noyau gaussien

Ce modèle a pour vocation de séparer les données en déterminant un hyperplan optimisé à l'aide de marges. Ici, il fait intervenir un kernel non linéaire capable de séparer notre type de données, ainsi que **2 paramètres à optimiser : C et Gamma**.

Le paramètre C de la classe SVC est le paramètre qui régit le degré de souplesse d'une marge. Par défaut, il a une valeur de 1.

- Plus la valeur de C est petite, plus les marges sont larges - ce qui peut conduire à plus de mauvaises classifications.
- Inversement, plus la valeur de C est grande, plus les marges du classificateur sont étroites - ce qui peut conduire à moins d'erreurs de classification mais une mauvaise généralisation.

En décidant de fixer C, on fait varier Gamma. On remarque que plus Gamma augmente, plus on passe en surapprentissage et par conséquent l'erreur en training tend vers 0 et **l'erreur en test décolle lorsque Gamma est trop haut (et celui en apprentissage plafonne au maximum)**.

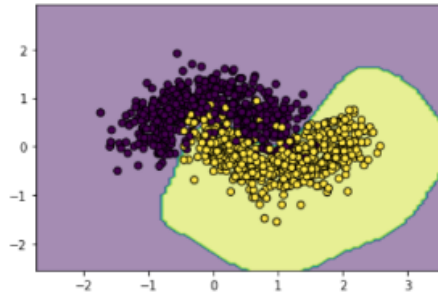
Gamma définit dans quelle mesure le modèle s'adapte aux données d'apprentissage, ce qui permet d'éviter l'overfitting. Le gamma définit la portée de l'influence d'un seul exemple de formation. Plus la valeur est faible, plus la portée d'un point de formation est grande.

Inversement, plus la valeur est grande, plus la portée du point de formation est faible.

Le nombre de vecteurs-supports (ce sont les points les plus proches de la séparatrice) joue un rôle important : il ne faut pas qu'il y ait trop de vecteurs-supports sinon cela voudrait dire que un trop grand nombre de points sont proches des frontières, ce qui peut mener à une mauvaise généralisation.

En utilisant une optimisation par recherche en grille, les paramètres optimaux et leur performance sont :

- **C = 1**
- **Gamma = 2**
 - **Accuracy en apprentissage = 0.86**
 - **Accuracy en test = 0.87**
 - **Temps d'inférence/exécution = 9e-06**
 - **Temps d'apprentissage = 10e-06**
 - **Nombre de vecteurs supports = 176**



- **Décision Trees**

L'arbre de décision est un diagramme de flux, et peut aider à prendre des décisions sur la base d'expériences passées. On doit chercher un attribut discriminant, lui attribuer un seuil et mettre en place un critère d'arrêt (indice de Gini, entropie...).

On choisit ici le critère "Entropy" et on l'entraîne selon deux schémas, avec **2 paramètres à optimiser (max_depth et min_samples_split)** :

- Faire varier max_depth et noter les performances
- Faire varier min_samples_split et noter les performances

L'entropie caractérise le degré de désorganisation, ou d'imprédictibilité, du contenu en information d'un système. Si on continue jusqu'à une entropie à 0 (cf. arbre complet) on est en sur-apprentissage. On va donc prendre des critères d'arrêt lors de l'apprentissage comme une entropie à une valeur minimum.

La "maximum tree depth" qui signifie profondeur maximale de l'arbre, il s'agit d'arrêter le développement de l'arbre une fois qu'il a atteint une certaine profondeur, cela évitera que l'arbre ne construise des branches avec trop peu d'exemples et donc permettra d'éviter un sur apprentissage (les noeuds sont développés jusqu'à ce que toutes les feuilles soient pures ou jusqu'à ce que toutes les feuilles contiennent moins de min_samples_split échantillon).

L'erreur en test décolle lorsque ce paramètre est trop haut (et celui en apprentissage plafonne au maximum).

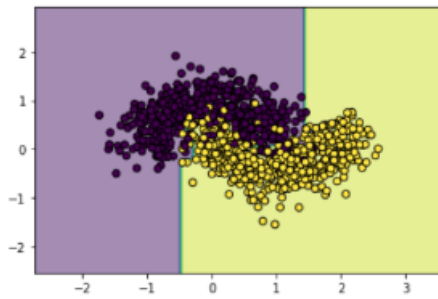
Le "minimum sample split" ou encore nombre d'exemples minimum pour un split consiste à ne pas splitter une branche si la décision concerne trop peu d'exemples. Cela permet également d'empêcher le surapprentissage.

L'erreur en test décolle lorsque ce paramètre est trop bas (et celui en apprentissage plafonne au maximum).

En utilisant une optimisation par recherche en grille, les paramètres optimaux et leur performance sont :

- **max_depth = 6**
- **min_samples_split = 20**
 - **Accuracy en apprentissage = 0.92**

- **Acuracy en test = 0.91**
- **Temps d'apprentissage = 3e-06**



- Extra Trees

Les Extra-Trees divisent en choisissant des nœuds seuils entièrement au hasard. Le principal avantage des arbres supplémentaires est la réduction du biais et du temps de calcul. Imaginons que l'on a un dataset de 1000 descripteurs, c'est compliqué/long de trouver le meilleur descripteur avec le meilleur seuil !

On utilise ici deux schémas :

- Lancement d'un seul extra trees 20 fois
- Lancement de 10, 20 et 50 extra-trees

On peut comparer le temps d'apprentissage et en combinant cela à la précision, on peut jauger si notre arbre de décision optimal ou nos extra-trees sont les plus performants.

Pour les Extra-Trees lancé en même temps : **la performance en test la plus intéressante est celle de 10 Extra-Trees avec un score de 90%. Le temps d'apprentissage associé est 0.000014.**

Pour notre Extra-Trees lancé 20 fois : **au bout de la 15ème fois, la performance en test la plus intéressante est à 90%. Le temps d'apprentissage associé est 9.663105e-07.**

Nous pouvons donc conclure qu'il est plus intéressant d'utiliser une combinaison d'Extra-Trees plutôt que de lancer plusieurs fois un même Extra-Tree (temps d'apprentissage moins long).

Conclusion

A l'issue de ce TP, nous pouvons dire que les 3 modèles sont efficaces pour traiter un modèle de classification non linéaire bi-classes, mais le plus performant au regard de la performance en test et du temps de calcul semble l'Extra-Tree.

Vous trouverez en annexe le notebook avec le code et les résultats.

In [48]:

	Itération	Performance en training	Performance en test	Temps d'apprentissage
0	1	1.0	0.866667	3.984928e-06
1	2	1.0	0.856667	2.990961e-06
2	3	1.0	0.823333	1.994133e-06
3	4	1.0	0.823333	2.002478e-06
4	5	1.0	0.840000	1.990080e-06
5	6	1.0	0.863333	1.994610e-06
6	7	1.0	0.846667	1.998425e-06
7	8	1.0	0.860000	1.988649e-06
8	9	1.0	0.870000	1.991749e-06
9	10	1.0	0.830000	1.036167e-06
10	11	1.0	0.873333	9.999275e-07
11	12	1.0	0.860000	9.999275e-07
12	13	1.0	0.836667	9.756088e-07
13	14	1.0	0.866667	1.004699e-06
14	15	1.0	0.900000	9.663105e-07
15	16	1.0	0.860000	9.949207e-07
16	17	1.0	0.846667	9.973049e-07
17	18	1.0	0.860000	9.989738e-07
18	19	1.0	0.856667	9.965897e-07
19	20	1.0	0.880000	1.018286e-06


In [35]:

```
import matplotlib.pyplot as plt
import numpy as np

df2 = pd.DataFrame(list(zip(perf_app, perf_test)), columns = ['Erreur en training', 'Erreur en test'])

fig = plt.figure(figsize = (10, 7))

#boite à moustache
plt.boxplot(df2)
plt.show()
```



Si on interprète les boxplot :

- erreur en training: maximum, 3ème quartile, médiane, 1er quartile et le minimum sont à 1 => on a une seule droite à une accuracy de 1
- erreur en test:
 - maximum : environ 0.9
 - 3ème quartile : environ 0.88
 - Médiane : environ 0.86
 - 1er quartile : environ 0.85
 - minimum : environ 0.83

In [51]:

```
print(df2["Erreur en test"].min())
print(df2["Erreur en test"].max())
print(df2["Erreur en test"].median())
print(np.quantile(df2["Erreur en test"], 0.25)) #1er quantile
print(np.quantile(df2["Erreur en test"], 0.75)) #3eme quantile

0.8366666666666667
0.8933333333333333
0.8583333333333334
0.8525
0.8808333333333334
```

4. Lancer l'apprentissage avec 10, puis 20 puis 50 extra-trees. Comment est réalisée la combinaison de leur prédiction ? Rassembler les performances en apprentissage, en test dans un tableau. Comparer le modèle optimal avec celui trouvé en 2).

In [63]:

```
from sklearn.ensemble import ExtraTreesClassifier
import time

total = t1-t0

precisionTrain = []
precisionTest = []
nbr_arbre = []
tps_apprentissage = []

for i in ([1, 10, 20, 50]):
    clf = ExtraTreesClassifier(n_estimators=i)#, splitter="random")

    t0 = time.time()
    clf.fit(X_train, y_train)
    t1 = time.time()

    predictTrain = clf.predict(X_train)
    predictTest = clf.predict(X_test)
    predictionTrain = metrics.accuracy_score(predictTrain, y_train)
    predictionTest = metrics.accuracy_score(predictTest, y_test)
    precisionTrain.append(predictionTrain)
    precisionTest.append(predictionTest)
    tps_apprentissage.append((t1-t0)/len(X))

    nbr_arbre.append(i)

colonnes = ["Nombre d'arbres", "Précision Train", "Précision Test"]
df = pd.DataFrame(columns=colonnes)
df["Nombre d'arbres"] = nbr_arbre
df["Précision Train"] = precisionTrain
df["Précision Test"] = precisionTest
df["Temps d'apprentissage"] = tps_apprentissage
df
```

Out [63]:

	Nombre d'arbres	Précision Train	Précision Test	Temps d'apprentissage
0	1	1.0	0.843333	0.000003
1	10	1.0	0.900000	0.000014
2	20	1.0	0.893333	0.000024
3	50	1.0	0.893333	0.000052

On peut comparer le temps d'apprentissage et en combinant cela à la précision, on peut jauger si notre arbre de décision optimal ou nos extratrees sont les plus performants.

- Pour les Extra-Trees lancé en même temps : la performance en test la plus intéressante est celle de 10 Extra-Trees avec un score de 90%. Le temps d'apprentissage associé est 0.000014.
- Pour notre Extra-Trees lancé 20 fois : au bout de la 15ème fois, la performance en test la plus intéressante est à 90%. Le temps d'apprentissage associé est 9.663105e-07.

Nous pouvons donc conclure qu'il est plus intéressant d'utiliser une combinaison d'ExtraTrees plutôt que de lancer plusieurs fois un même ExtraTree (temps d'apprentissage moins long).

Pour des différences plus probantes, nous pourrions tester avec un dataset beaucoup plus large que nos 1000 données de base.

In []:

```
#sur le anaconda prompt, taper :
#python nbconvert --to PDFviaHTML TP2_SocialNetwork_MADAD_NOUAR.ipynb
```