TP APSTA

Diane CHOUNLAMOUNTRY
Orane VERNET

Décembre 2023

1 Introduction

Nous utilisons la base de données "animals10classes" de Caltech 101. Avec ces données, nous allons nous entraîner à utiliser des classifieurs linéaires ainsi que des machines à vecteur de support. Nous cherchons également à comparer les différents types de classfieurs et à effectuer des mesures de performance.

2 Préparation : téléchargement et séparation des données

On dispose de plusieurs images d'animaux, ces dernières sont réparties selon 10 classes en fonction de l'animal présent dessus.

On trouve les 10 classes suivantes : 'ant', 'butterfly', 'crayfish', 'crocodile', 'dragonfly', 'flamingo', 'lobster', 'octopus', 'sea horse', 'starfish'.

La fonction *buildDataset* permet de réduire la base données à un nombre K de classes. Dans ce TP, on s'intéresse à une classification binaire on choisit donc $K = 2$ et on garde les classes "octopus" et "dragonfly" (on peut commenter la commande 'random.seed(a=seed)' dans la fonction buildDataset si on veut que les classes soient tirées aléatoirement).

On sépare ensuite les données : 80% sont utilisés pour l'entraînement (train) et 20% pour tester (test). Nous avons utilisé un random.sample dans le but de ne pas avoir que la classe 1 dans le set des tests. Initialement nous avions pris les 80 premiers pourcents de X pour l'entraînement et les 20 pourcents restants pour le test ; et donc le test ne contenait que le label 1, ce qui n'est pas forcément la meilleure façon de tester.

3 SVM (Machine à vecteur de support)

3.1 SVM linéaire

On va s'appuyer sur un modèle SVM linéaire (SVC). On prédit un Y_pred à partir de X_test puis on évalue la performance en calculant l'erreur effectuée par rappport à Y_test. Le nombre d'erreur est égal au nombre de 1 dans le vecteur $err = abs(Y\_pred - Y\_test)$ puisqu'on travaille uniquement avec les classes 0 et 1. On trouve souvent entre 7 et 8 erreurs sur 21, ce qui est assez conséquent.
On utilise également la fonction *SVC.score* qui renvoie la précision moyenne (mean accuracy) qui vaut dans notre cas 1. C'est une mesure courante pour les modèles de classification et représente le rapport entre les prédictions correctes et le nombre total de prédictions.

Ensuite, nous faisons une prédiction probabiliste en rajoutant l'argument de probabilité ($svMLin = SVC(kernel =' linear', probability = True)$. On crée donc le vecteur de prédictions probabilistes Y_pred_proba. C'est un vecteur à deux colonnes avec autant de lignes qu'il y a de données "test". A la ligne i, on trouve sur la première colonne la probabilité que la i-ème donnée test appartienne à la classe 0, et sur la deuxième colonne la probabilité qu'elle appartienne à la classe 1.
Le modèle de probabilité est basée sur la cross-validation, les résultats peuvent donc être légèrement différents que ceux fait par prédiction. Par ailleurs, sur une base de données avec peu d'éléments, cette méthode n'est pas assez précise.

3.2 Vecteurs de support

Le modèle a 10 000 paramètres ; c'est la taille du vecteur svmLin.coef_. Pour un modèle linéaire, le seul hyperparamètre est C. Il y a 64 vecteurs de support. Le nombre de vecteurs support pour la classe 0 est 25 et pour la classe 1 est 37.

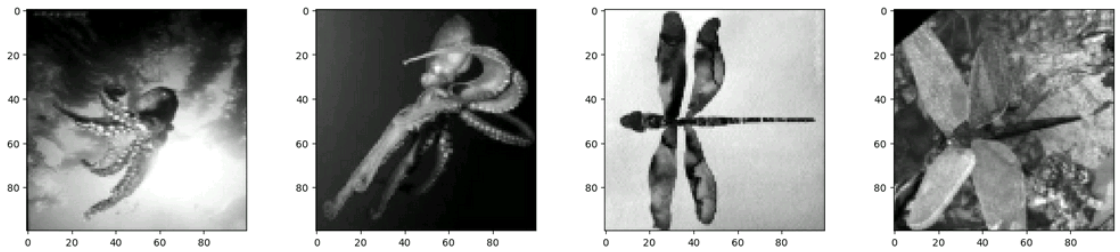

Figure 1: Quelques vecteurs de support

3.3 Des SVMs avec d'autres kernels

Maintenant on va entraîner trois autres types de modèles SVC en utilisant les kernels suivants : 'rbf', 'poly' et 'sigmoid'.

- Modèle RBF : Score SVC = 0,857 ; Accuracy = 0,619 ; Hyperparamètres : C et γ
- Modèle poly : Score SVC = 1 ; Accuracy = 0,714 ; Hyperparamètres : C, degré, coef0
- Modèle sigmoid : Score SVC = 1 ; Accuracy = 0,619 ; Hyperparamètres : C, coef0

Commentaire : Le modèle poly semble être le plus efficace. On remarque aussi que le Y_{pred} du modèle RBF ne prédit que la classe 1, nous ne savons pas l'expliquer.

3.4 Réglage des hyperparamètres

Le réglage des hyperparamètres est très important, il implique de trouver les meilleures valeurs pour les hyperparamètres du modèle afin d'optimiser ses performances. Nous avons utilisé la fonction 'GridSearchCV' qui prend en argument le modèle et une grille de paramètres (dans notre cas, la grille de paramètres était donnée).

Les meilleurs paramètres retournés par la fonction sont affichés sur la figure suivante :

```
Meilleurs paramètres (linéaire) : {'C': 0.1, 'degree': 2, 'gamma': 1}
Meilleurs paramètres (RBF) : {'C': 10, 'degree': 2, 'gamma': 0.001}
Meilleurs paramètres (Poly) : {'C': 0.1, 'degree': 2, 'gamma': 1}
Meilleurs paramètres (Sigmoid) : {'C': 100, 'degree': 2, 'gamma': 0.01}
```

Figure 2: Meilleurs paramètres

On peut également observer l'évolution de l'accuracy après réglage des hyperparamètres sur la figure qui suit. On note que la précision a augmenté pour les modèles linéaire et RBF, qu'elle a baissé pour le modèle poly et qu'elle a stagné pour le modèle sigmoid. Cela parait bizarre et n'est pas très cohérent, on va retrouver ces évolutions sur les courbes de ROC dans la partie qui suit (une baisse d'efficacité pour le modèle poly, et une amélioration pour les modèles linéaires et rbf).

```
Précision du modèle linéaire:  0.8095238095238095
Précision du modèle RBF:  0.8095238095238095
Précision du modèle poly:  0.6190476190476191
Précision du modèle sigmoid:  0.6190476190476191
```

Figure 3: Evolutions de l'accuracy après réglage des hyperparamètres

4 Mesure de performance

4.1 Mesures d'évaluation

Nous allons maintenant effectuer des mesures d'évaluations. On crée une fonction 'compute_measures' qui va renvoyer différents types d'évaluations (l'accuracy, la spécificité, la précision, etc...). On commence par créer la matrice d'erreur en utilisant le module 'confusion_matrix. Notons que pour éviter une division par zéro, nous rajoutons à tous les dénominateurs un 'zéro numérique' qui correspond à un $eps = 10^{-12}$.
On peut ensuite comparer avec les mesures d'évaluations sorties par le module spécifique de sklearn 'classification_report'.

```
Ma fonction
TP  13 TN  4 FP 4 FN 0 Total 21
Accuracy  0.809523809523771
 Precision 0.7647058823528963
 Recall 0.9999999999999231
 Specificity  0.4999999999999375
 F-measure 0.8666666666661177
 NPV 4.0
 FPV 0.4999999999999375

Fonction de sklearn
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.50      0.67         8
         1.0       0.76      1.00      0.87        13

    accuracy                           0.81        21
   macro avg       0.88      0.75      0.77        21
weighted avg       0.85      0.81      0.79        21
```

Figure 4: Mesures d'évaluation faites par notre fonction et par le module classification_report pour le modèle linéaire

Les résultats entre les deux méthodes de mesures sont similaires. On obtient une accuracy de 0,81, une précision de 0,76, un score F1 de 0,67 pour le modèle linéaire.

4.2 Courbes de ROC

Nous avons utilisé la fonction 'roc_curve' de sklearn pour afficher les courbes de ROC. Dans un premier temps, on a affiché les courbes avant le réglage des hyperparamètres. Les résultats ne sont pas très satisfaisants : les courbes s'écartent à peine de la courbe sans aucun entraînement. Notamment pour le modèle RBF où il semble qu'aucun entraînement n'a été fait. La meilleure courbe est celle avec le modèle 'poly'.

Après le réglage des hyperparamètres, certaines courbes de ROC sont mieux notamment avec le modèle linéaire et le modèle RBF. En revanche, celle du modèle Poly est moins satisfaisante et nous ne comprenons pas vraiment pourquoi.

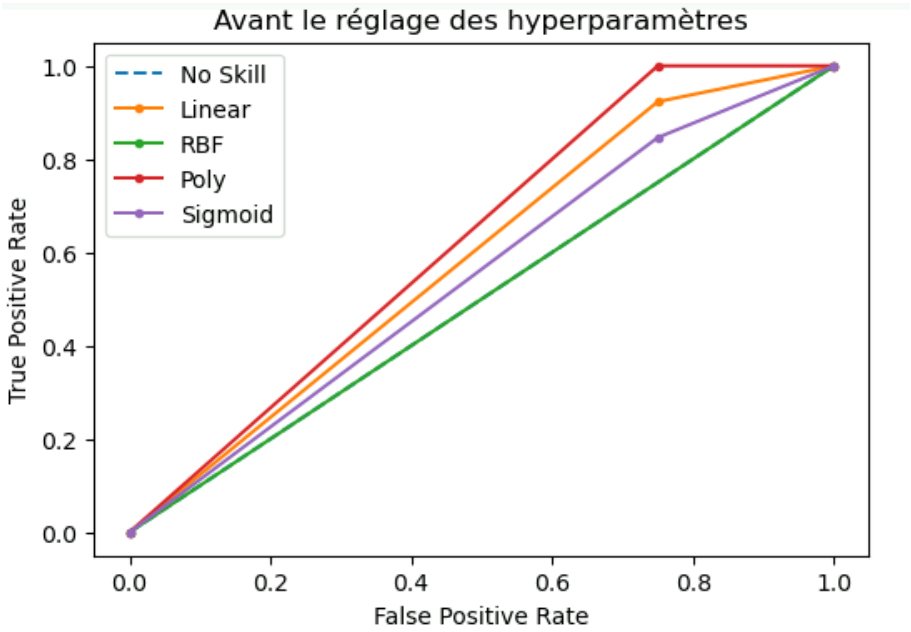

Figure 5: Courbes de ROC avant le réglage des hyperparamètres

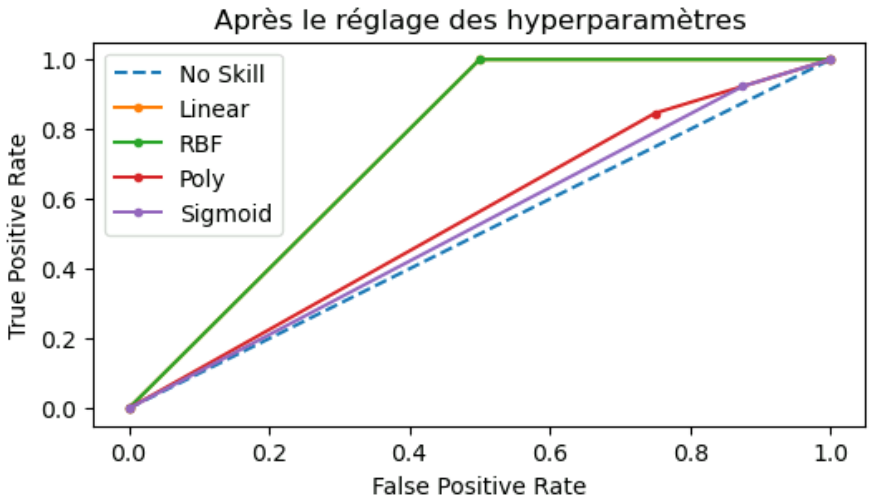

Figure 6: Courbes de ROC après le réglage des hyperparamètres

Remarque : nous n'avons pas réussi à faire le modèle avec régression logistique.

5 Comparaison des classifieurs (TP3)

Dans cette partie nous travaillons sur la même dataset que précédemment. Cependant, nous allons d'avantage nous concentrer sur la tâche de classification multi-classe en comparant les performances les trois modèles de classification suivants : SVM, Single Decision Tree and Random Forest.

5.1 Entraînement des 3 modèles de classification

Les modèles de classification 'Logistic regression' et 'SVM' sont des modèles de classification binaires. Ils ne sont donc pas nativement multiclasse. En revanche, les modèles basés sur des arbres de décision le sont.

Dans un premier temps nous chargeons et séparons le dataset grâce aux fonctions loadImagesAndLabels() et buildDataset() construites précédemment. Nous prenons 5 classes.

```
The chosen label  0  is  crayfish
It will be read in //data-pfe/overnet2022/EI2/APSTA/TP3/animals10classes/crayfish
The chosen label  1  is  octopus
It will be read in //data-pfe/overnet2022/EI2/APSTA/TP3/animals10classes/octopus
The chosen label  2  is  lobster
It will be read in //data-pfe/overnet2022/EI2/APSTA/TP3/animals10classes/lobster
The chosen label  3  is  flamingo
It will be read in //data-pfe/overnet2022/EI2/APSTA/TP3/animals10classes/flamingo
The chosen label  4  is  sea_horse
It will be read in //data-pfe/overnet2022/EI2/APSTA/TP3/animals10classes/sea_horse
Total number of samples 270
Used labels ['crayfish', 'octopus', 'lobster', 'flamingo', 'sea_horse']
Size of data matrix (270, 10000)
```

Figure 7: Classes retenues

Nous entraînons ensuite nos 3 modèles de classifieur et imprimons leur accuracy.
Pour le modèle SVM, nous obtenons une valeur d'accuracy sur le set de test de 0,314814.
Pour le modèle Decision Tree de 0,351851.
Et pour le modèle de Random Forest de 0,44444.
Pour les trois modèles, la valeur d'accuracy sur les set d'entraînement sont égales à 1.

On constate que le modèle de Random Forest donne le meilleur résultat sur les données test et que le modèle SVM donne le moins bon, cela est sans doute dû au fait qu'il n'est pas nativement multiclasse et est donc moins adapté à notre problème.

5.2 ROC curves

Les courbes ROC sont utilisées pour des problèmes de classification binaires. On ne peut donc pas les utiliser directement pour les problèmes multiclasses. Pour cela, il faut utiliser une stratégie de 'one-vs-all' en désignant une classe à laquelle on attribuera la valeur 'vraie' (ou égale à 1) et en attribuant la valeur 'fausse' (ou égale à 0) à toutes les autres classes.
Malheureusement, nous ne sommes pas parvenues à tracer ces courbes.

5.3 Grid Search

Finalement, nous effectuons une recherche en grille afin de déterminer les paramètres optimaux pour les trois modèles de classification. Cela consiste à essayer toutes les combinaisons possibles des hyperparamètres en entraînant le modèle pour chaque combinaison et en choisissant la combinaison qui donne les meilleurs résultats.
On obtient les résultats suivant :

```
Meilleurs paramètres pour SVM: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
Meilleurs paramètres pour Decision Tree: {'max_depth': 5, 'max_features': 45, 'min_samples_leaf': 5}
Meilleurs paramètres pour Random Forest: {'max_depth': 10, 'max_features': 15, 'min_samples_leaf': 3, 'n_estimators': 75}
```

Figure 8: Paramètres optimaux obtenus par recherche en grille

Les valeurs d'accuracy obtenues en utilisant ces valeurs pour les hyperparamètres sont les même que précedemment.