Machine Learning: Supervised

Exercice 3: Prédiction du gagnant d'un match de NBA

Afin de prédire avec la plus grande certitude quelle équipe remportera un match de la NBA, nous disposons de deux ensembles de données. Le premier est un fichier inputs.npy qui contient les données d'un match à la mi-temps, prétraitées pour être compatibles avec notre modèle. Le second est un fichier labels.npy qui contient les résultats de ces matchs, permettant ainsi de former notre intelligence artificielle.

Tout d'abord nous allons diviser nos dataset afin d'avoir des échantillons tests, de la façon suivante :

- 90% sera consacré à l'entraînement (soit 450 lignes)
- 10% sera consacré aux tests (soit 50 lignes)

Maintenant, nous pouvons procéder à des tests en utilisant différents modèles et ajuster les hyperparamètres pour obtenir la meilleure précision possible.

Logistic Regression

Dans le cas du modèle "Régression logistique", nous allons tester trois hyperparamètres qui ont le plus d'impact sur les performances du modèle, tandis que les autres ont un impact négligeable dans notre cas. Voici la liste de ces hyperparamètres avec leurs options :

- Solver : Optimisation du model selon le dataset
    - Lbfgs : options de base, bonne performance et utilise peu de mémoire cependant produit parfois des erreurs de convergence
    - Newton-cg : coûteux en calcul dû à l'utilisation de la matrice hessienne
    - Sag : option la plus rapide pour de gros dataset avec un nombre de caractéristique et d'échantillons élevé
    - Saga : option faite pour la régression logistique multinomial, également correct pour de gros dataset

- Penalty : Vise à réduire l'erreur de généralisation du modèle et minimise et réglemente le surajustement

	Solvers				
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty ('none')	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

- C : Fonctionne avec l'option « Penalty » pour réguler le surajustement, une faible valeur implique une minimisation forte tandis qu'une forte valeur donne un poids élevé aux données d'apprentissage pour le model

Afin d'obtenir la meilleur combinaison d'hyperparamètre possible nous allons tout simplement réaliser une liste avec toutes les combinaisons possibles. Nous aurons alors un tableau nous indiquant les résultats des performances de chaque combinaison et nous donner les hyperparamètres de la combinaison la plus performante.

Support Vector Machine (SVM)

Dans le cas du model SVM, il y a, la encore, 3 hyperparamètre sur lesquels nous allons nous pencher car ce sont eux qui affecte le plus le model dans notre cas. Les autres hyperparamètres seront eux négligeable. Voici la liste de ces 3 hyperparamètres :

- Kernel : Transforme le dataset en augmentant la dimension pour le séparé linéairement
  - Rbf : Option par défaut du Kernel
  - Poly : Option cherchant a combiner les caractéristiques en plus de chercher leur similarité
- C : Régularise le surajustement selon la valeur affecté, une valeur faible impliquera une forte minimisation du surajustement tandis qu'une valeur forte laissera aux données un poids plus important dans l'apprentissage
- Gamma : Détermine le coefficient kernel, si la valeur est faible le rayon d'influence du vecteur support est élevé et inversement

Afin d'obtenir la meilleur combinaison d'hyperparamètre, nous allons cette fois utilisé la méthode « struct Search », qui consiste à définir les valeurs des hyperparamètres dans une grid puis, par la suite de tester toutes les combinaisons depuis cette grid.

```
parameters_struct = {
    "C": c_range,
    "kernel": ['rbf', 'poly'],
    "gamma": gamma_value.tolist()+['scale', 'auto']
}
scoring = ['accuracy']

kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=0)
struct_search = GridSearchCV(estimator=SVC(),
                             parameters_struct=parameters_struct,
                             scoring=scoring,
                             refit='accuracy',
                             n_jobs=1,
                             cv=kfold,
                             verbose=0)
struct_result = struct_search.fit(x_train, y_train)
```

Source:

Model:
https://moncoachdata.com/blog/modeles-de-machine-learning-expliques/

LR hyperparamètre:
https://medium.com/codex/do-i-need-to-tune-logistic-regression-hyperparameters-1cb2b81fca69

SVM:
https://www.mltut.com/svm-implementation-in-python-from-scratch/

SVM hyperparamètre :
https://medium.com/grabngoinfo/support-vector-machine-svm-hyperparameter-tuning-in-python-a65586289bcb