Introduction à l'intelligence artificielle - groupe A

Viet NGUYEN 20006303

22/12/2022

Rapport du projet

1) Présentation du Dataset:

- J'ai beaucoup été exposé sur la type classification dans le cours IIA, donc dans ce projet, je veux en savoir plus sur les modèles d'algorithmes de regression par moi-meme. La régression est un processus de recherche des corrélations entre les variables dépendantes et indépendantes. Il aide à prédire les variables continues telles que la prédiction des tendances du marché, la prédiction des prix des maisons, etc.

- Dans ce cahier, je veux bien à présenter rapidement le jeu de données connu sous le nom de **« California housing dataset ».** C'est les statistiques sur les prix des maisons concernant chaque district de Californie. La raison pour laquelle je le préfère est qu'il est disponible dans sckikit, et toutes les valeurs sont des float, donc c'est parfait pour moi pour commencer avec Regession. Cet ensemble de données peut être récupéré sur Internet à l'aide de scikit-learn.

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	PRICE
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Informations sur les attributs:

- **MedInc** : revenu médian en groupe bloc.
- **HouseAge** : âge médian de la maison dans le groupe d'îlots.
- **AveRooms** : nombre moyen de pièces par foyer.

- **AveBedrms** : nombre moyen de chambres par ménage.
- **Population** : population du groupe bloc.
- **AvOccup** : nombre moyen de membres du ménage.
- **Latitude** : latitude du groupe de blocs.
- **Longitude** : longitude du groupe de blocs.
- **PRICE** : prix des maisons.

2) Description du modèle:

- J'utilise les modèles Regression qui ont cours dans scikit.
- J'ai créé un nom de dictionnaire 'modèle' qui contient tous les modèles que je connais à l'intérieur:

```
models = {
    'linear': LinearRegression(),
    'ridge': RidgeCV(alphas = np.linspace(1e-3,10)),
    'lasso': LassoCV(alphas = np.linspace(1e-3,10)),
    'SVR': SVR(C = 0.5),
    'knn': KNeighborsRegressor(n_neighbors = 5),
    'dt' : DecisionTreeRegressor(max_depth = 5),
    'rf' : RandomForestRegressor(),
    'et' : ExtraTreesRegressor(),
    'gbm': GradientBoostingRegressor(),
    'histgbm': HistGradientBoostingRegressor()
}
```

- Cependant, comme il n'est pas possible de déterminer les propiétés exactes des algorithmes à l'intérieur Dictionaire 'models', donc je laisse ces propriétés avec la valeur par defaut. La validation se fera dans la validation croisée pour comparer les résultats.

- Il existe 10 medèles dans cette Dictionaire:
    +) Linear Regression
    +) Ridge
    +) Lasso
    +) SVR (Super Vector Regression)
    +) KNN (K Nearest Neighbors)
    +) Decision Tree
    +) Random Forest
    +) Extra Tree
    +) GBM (Gradient Boosting)
    +) Hist Gradient Boosting

3) Formation – Test:

- Il y a toujours 3 étapes de base pour entraîner les modèles :

1/ Appelez le nom de la classe de ce modèle après l'avoir importé dans scikit.

```
histgbm = HistGradientBoostingRegressor(max_leaf_nodes = 16,
                                        max_depth = 7,
                                        max_iter = 1000,
                                        learning_rate = 0.05)
```

2/ Utilisation de la fonction **fit(X_train, y_test)**.

Exemple:

```
histgbm.fit( X_train, y_train )
```

3/ Imprimez la **précision**, à quelle fréquence la régression est-elle correcte?

```
print('R2 for Train)', r2_score(y_train, histgbm.predict(X_train)))

print('R2 for Test (validation)', r2_score(y_test,
                                           histgbm.predict(X_test)))
```

```
R2 for Train) 0.9989455225197804
R2 for Test (validation) 0.7966756319744817
```

- Mais dans mon cas, au lieu d'écrire un par un, j'ai choisi d'utiliser une boucle pour faire tous les modèles dans la Dictionnaire '**models**' à la fois.

```
scores_results = {k: np.array([0,0])for k in models.keys()}
```

```
for k, model in models.items():
    model.fit(X_train, y_train)
    scores_results[k] = np.array([r2_score(y_train,model.predict(X_train)),
                                  r2_score(y_test,model.predict(X_test))])
```

- Et pour imprimez la **précision** sous la forme d'une tableau, j'ai utilisé **panda.DataFrame.from_dict** comme ça:

```
df_scores = pd.DataFrame.from_dict(scores_results,
                                   orient='index',
                                   columns = ['Train Score', 'Test Score'])
```

- Et voici le resutat:

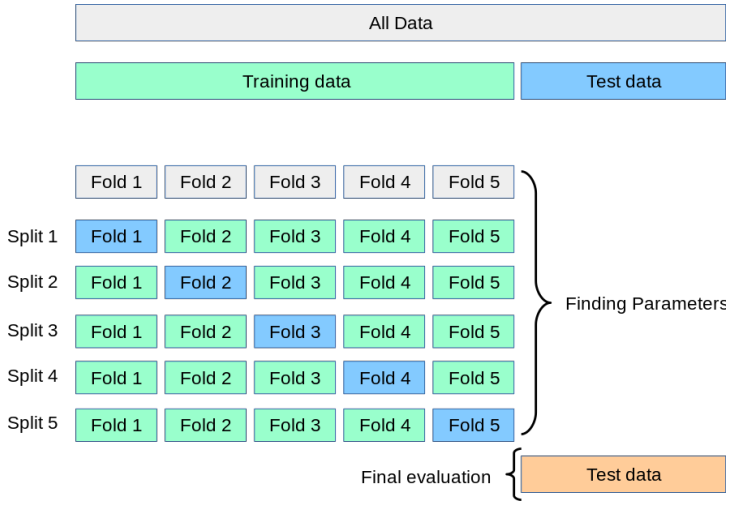
	Train Score	Test Score
linear	0.611294	0.592609
ridge	0.611289	0.592612
lasso	0.611257	0.592629
SVR	-0.038183	-0.038798
knn	0.438298	0.140096
dt	0.631082	0.596863
rf	0.972422	0.791590
et	1.000000	0.803486
gbm	0.807847	0.782294
histgbm	0.883238	0.832029

- Dans le tableau des résultats ci-dessus, l'algorithme le plus puissant est l'algorithme Extra Tree Regression avec le score d'entrainement le plus élevé, mais après le test, l'algorithme Hist Gradient Boosting a le plus test score. Ce qui me déroute, c'est qu'à propos de l'algorithme SVR (Super Vector Regression), il est très fort, mais ses deux colonnes de score sont négatives.

→ La valeur d'apprentisage d'un algorithme est souvent supérieureà la valeur après test. Cela montre que peu importe la hauteur du score d'entrainement, si le score du test est faible, cela montre que l'algorithme est trop mémorisé et difficile à improviser dans la situation de test.

4) Validation Croisée:

- C’est l’étape pour confirmer le score d’attribut exact requis pour les algorithmes en les essayant un par un et donner le plus approriédans un montant fixe, comme indiqué dans le diagramme ci-dessous:



- Et la classe **GridSearchCV** est la solution que je trouve la plus OK de le faire, par exemple:

Decision Tree

```
grid_dt = GridSearchCV(models['dt'],
                       cv = 5,
                       n_jobs = 4,
                       param_grid = {'max_depth' : np.linspace(2,50,num = 15).astype(int)}
                      )
```

```
grid_dt.fit(X_train, y_train)
```

GridSearchCV
estimator: DecisionTreeRegressor
DecisionTreeRegressor

```
pd.DataFrame(grid_dt.cv_results_).sort_values('rank_test_score')[['param_max_depth',*score_cols]]
```

	param_max_depth	mean_test_score	rank_test_score
2	8	0.665432	1
3	12	0.641301	2
4	15	0.608608	3
1	5	0.607672	4
12	43	0.590179	5
13	46	0.589862	6
6	22	0.586469	7
5	19	0.586396	8
8	29	0.586105	9
10	36	0.584260	10
9	32	0.583259	11
14	50	0.582565	12
7	26	0.582141	13
11	39	0.581091	14
0	2	0.446610	15

5) Rapportez les résultats et comparez la force des modèles:

-

	Train Score	Test Score
tuned_histgbm	0.906846	0.835539
histgbm	0.883238	0.832029
tuned_gbm	0.855144	0.811253
tuned_et	0.961004	0.808369
tuned_rf	0.954766	0.807331
et	1.000000	0.803486
rf	0.972422	0.791590
gbm	0.807847	0.782294
tuned_dt	0.755842	0.667557
dt	0.631082	0.596863
lasso	0.611257	0.592629
ridge	0.611289	0.592612
linear	0.611294	0.592609
tuned_knn	0.344870	0.156911
knn	0.438298	0.140096
tuned_SVR	-0.038183	-0.038798
SVR	-0.038183	-0.038798