DATA SCIENTEST

MODÈLE DE PRÉDICTION DES RECORDS D'ATHLÉTISME

JEUX OLYMPIQUES 2024

Auteurs :
Armen ASATRYAN
Yannick OREAL
Yves-Marie SALAUN
Xavier GERVAIS

Managers :
Jérémy ROBERT
Raja AARAB

RAPPORT PRELIMINAIRE DE MODÉLISATIONS BASELINE

Projet réalisé dans le cadre de la formation DATA SCIENTIST BOOTCAMP d'avril 2024

ATHLÉTISME DU 01 AU 11 AOÛT 2024

Classification

COMPARAISON des MODELES

Dataset Temps

Le Random Forest est le plus approprié

```
[253]: #knn
       #Random forest
       #LR

       clf1 = KNeighborsClassifier(n_neighbors=3)
       clf2 = RandomForestClassifier(random_state=123)
       clf3 = LogisticRegression(solver='saga', max_iter=10000)

       vclf = VotingClassifier(estimators=[('knn', clf1), ('rf', clf2), ('lr', clf3)], voting='hard')

       cv3 = KFold(n_splits=3, random_state=111, shuffle=True)

       for clf, label in zip([clf1, clf2, clf3, vclf], ['KNN', 'Random Forest', 'Logistic Regression', 'Voting Classifier']):
           scores = cross_validate(clf, X_train, y_train, cv=cv3, scoring=['accuracy', 'f1', 'precision', 'recall'], error_score='raise')
           print("{%s}: \n Accuracy: %0.2f (+/- %0.2f)" % (label, scores['test_accuracy'].mean(), scores['test_accuracy'].std()),
                 "F1 score: %0.2f (+/- %0.2f)" % (scores['test_f1'].mean(), scores['test_f1'].std()),
                 "Precision: %0.2f (+/- %0.2f)" % (scores['test_precision'].mean(), scores['test_precision'].std()),
                 "Recall: %0.2f (+/- %0.2f)" % (scores['test_recall'].mean(), scores['test_recall'].std()))

       [KNN]:
        Accuracy: 0.99 (+/- 0.00) F1 score: 0.44 (+/- 0.01) Precision: 0.63 (+/- 0.01) Recall: 0.34 (+/- 0.01)
       [Random Forest]:
        Accuracy: 1.00 (+/- 0.00) F1 score: 0.87 (+/- 0.01) Precision: 0.98 (+/- 0.01) Recall: 0.78 (+/- 0.02)
       [Logistic Regression]:
        Accuracy: 0.99 (+/- 0.00) F1 score: 0.42 (+/- 0.01) Precision: 0.88 (+/- 0.05) Recall: 0.27 (+/- 0.01)
       [Voting Classifier]:
        Accuracy: 0.99 (+/- 0.00) F1 score: 0.61 (+/- 0.02) Precision: 0.99 (+/- 0.01) Recall: 0.44 (+/- 0.02)
```

ISOF, avec une réduction de nombre d'entrée à 20k)

```
Entrée [22]: print(classification_report(y_test_isof, y_pred_isof))
```

	precision	recall	f1-score	support
-1	0.18	0.10	0.13	273
1	0.94	0.97	0.95	4018
accuracy			0.91	4291
macro avg	0.56	0.53	0.54	4291
weighted avg	0.89	0.91	0.90	4291

Out[21]:

Classe prédite	-1	1
Classe réelle		
-1	27	246
1	126	3892

Le modèle prédit mal les records (target -1)

Classification - Temps

CHOIX DES HYPERPARAMETRES

```
[255]: #grid search

from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier()
clf3 = SVC(probability=True)
vclf = VotingClassifier(estimators=[('knn', clf1), ('rf', clf2), ('svm', clf3)], voting='soft')

params = {
    'knn__n_neighbors': [1, 3, 5],
    'rf__n_estimators': [10, 20, 30],
    'svm__C': [0.005, 0.01, 0.015],
}

grid = GridSearchCV(estimator=vclf, param_grid=params, cv=5)
grid.fit(X_train, y_train)

print(grid.best_params_)
```

rf_n_estimators = 10

```
{'knn__n_neighbors': 1, 'rf__n_estimators': 10, 'svm__C': 0.015}
```

```
[270]: # best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Reports
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9982733206990946
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     23429
           1       1.00      0.87      0.93       316

    accuracy                           1.00     23745
   macro avg       1.00      0.94      0.96     23745
weighted avg       1.00      1.00      1.00     23745
```

Gradient BOOSTING

```
Accuracy: 0.9964624131396084
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     23465
           1       0.91      0.78      0.84       280

    accuracy                           1.00     23745
   macro avg       0.95      0.89      0.92     23745
weighted avg       1.00      1.00      1.00     23745
```

Classification

COMPARAISON des MODELES

Dataset Distance

```
[186]: #knn
       #Random forest
       #LR

       clf1 = KNeighborsClassifier(n_neighbors=3)
       clf2 = RandomForestClassifier(random_state=12)
       clf3 = LogisticRegression(max_iter=10000)

       vclf = VotingClassifier(estimators=[('knn', clf1), ('rf', clf2), ('lr', clf3)], voting='hard')

       cv3 = KFold(n_splits=3, random_state=111, shuffle=True)

       for clf, label in zip([clf1, clf2, clf3, vclf], ['KNN', 'Random Forest', 'Logistic Regression', 'Voting Classifier']):
           scores = cross_validate(clf, X_train, y_train, cv=cv3, scoring=['accuracy','f1', 'precision', 'recall'])
           print("[%s]: \n Accuracy: %0.2f (+/- %0.2f)" % (label, scores['test_accuracy'].mean(), scores['test_accuracy'].std())
                 "F1 score: %0.2f (+/- %0.2f)" % (scores['test_f1'].mean(), scores['test_f1'].std()),
                 "Precision: %0.2f (+/- %0.2f)" % (scores['test_precision'].mean(), scores['test_precision'].std()),
                 "Recall: %0.2f (+/- %0.2f)" % (scores['test_recall'].mean(), scores['test_recall'].std()))

[KNN]:
 Accuracy: 0.99 (+/- 0.00) F1 score: 0.40 (+/- 0.03) Precision: 0.48 (+/- 0.02) Recall: 0.36 (+/- 0.07)
[Random Forest]:
 Accuracy: 1.00 (+/- 0.00) F1 score: 0.89 (+/- 0.04) Precision: 0.97 (+/- 0.02) Recall: 0.82 (+/- 0.05)
[Logistic Regression]:
 Accuracy: 1.00 (+/- 0.00) F1 score: 0.76 (+/- 0.01) Precision: 0.99 (+/- 0.01) Recall: 0.62 (+/- 0.02)
[Voting Classifier]:
 Accuracy: 1.00 (+/- 0.00) F1 score: 0.75 (+/- 0.06) Precision: 0.98 (+/- 0.01) Recall: 0.62 (+/- 0.08)
```

ISOF, avec une réduction de nombre d'entrée à 20k)

```
Entrée [57]: print(classification_report(y_test_isof, y_pred_isof))
```

	precision	recall	f1-score	support
-1	0.28	0.34	0.31	103
1	0.98	0.98	0.98	4001
accuracy			0.96	4104
macro avg	0.63	0.66	0.65	4104
weighted avg	0.97	0.96	0.96	4104

Out[56]:

Classe prédite	-1	1
Classe réelle		
-1	35	68
1	88	3913

Le modèle prédit mal les records (target -1), idem Temps

Classification

CHOIX DES HYPERPARAMETRES

Dataset Distance

```
[189]: #grid search

from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier()
clf3 = SVC(probability=True)
vclf = VotingClassifier(estimators=[('knn', clf1), ('rf', clf2), ('svm', clf3)], voting='soft')

params = {
    'knn__n_neighbors': [1, 3, 5],
    'rf__n_estimators': [10, 20, 30],
    'svm__C': [0.005, 0.01, 0.015],
}


grid = GridSearchCV(estimator=vclf, param_grid=params, cv=5)
grid.fit(X_train, y_train)

print(grid.best_params_)
```

rf_n_estimators = 30

```
{'knn__n_neighbors': 1, 'rf__n_estimators': 30, 'svm__C': 0.01}
```

```
# Reports
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9991564295882321
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18854
           1       0.99      0.87      0.92       113

    accuracy                           1.00     18967
   macro avg       0.99      0.93      0.96     18967
weighted avg       1.00      1.00      1.00     18967
```

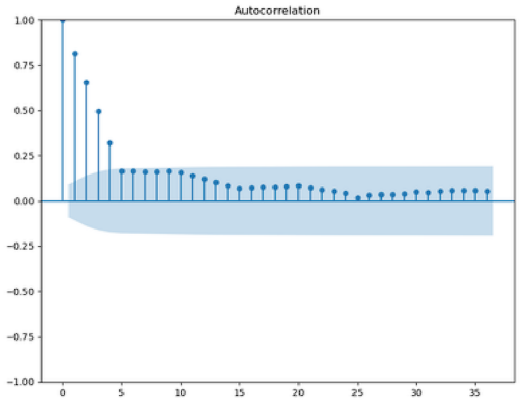
Gradient BOOSTING

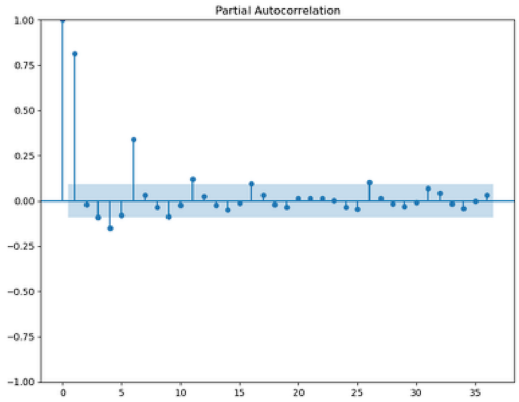
```
Accuracy: 0.9984183054779353
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18854
           1       0.87      0.86      0.87       113

    accuracy                           1.00     18967
   macro avg       0.94      0.93      0.93     18967
weighted avg       1.00      1.00      1.00     18967
```

SARIMAX

SERIES TEMPORELLES

Dataset Distance





SARIMAX Results

Dep. Variable:	mark	No. Observations:	465
Model:	SARIMAX(4, 0, 1)	Log Likelihood	283.596
Date:	Wed, 29 May 2024	AIC	-555.192
Time:	12:20:59	BIC	-530.340
Sample:	0	HQIC	-545.410
	- 465		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.8615	2.058	0.419	0.676	-3.173	4.896
ar.L2	0.1304	1.883	0.069	0.945	-3.559	3.820
ar.L3	0.0497	0.156	0.319	0.750	-0.255	0.355
ar.L4	-0.0416	0.115	-0.361	0.718	-0.267	0.184
ma.L1	0.0636	2.068	0.031	0.975	-3.989	4.116
sigma2	0.0170	0.000	65.243	0.000	0.016	0.017

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	93700.66
Prob(Q):	0.97	Prob(JB):	0.00
Heteroskedasticity (H):	4.67	Skew:	0.54
Prob(H) (two-sided):	0.00	Kurtosis:	72.53

Régression linéaire

ELASTIC NET

Dataset Distance

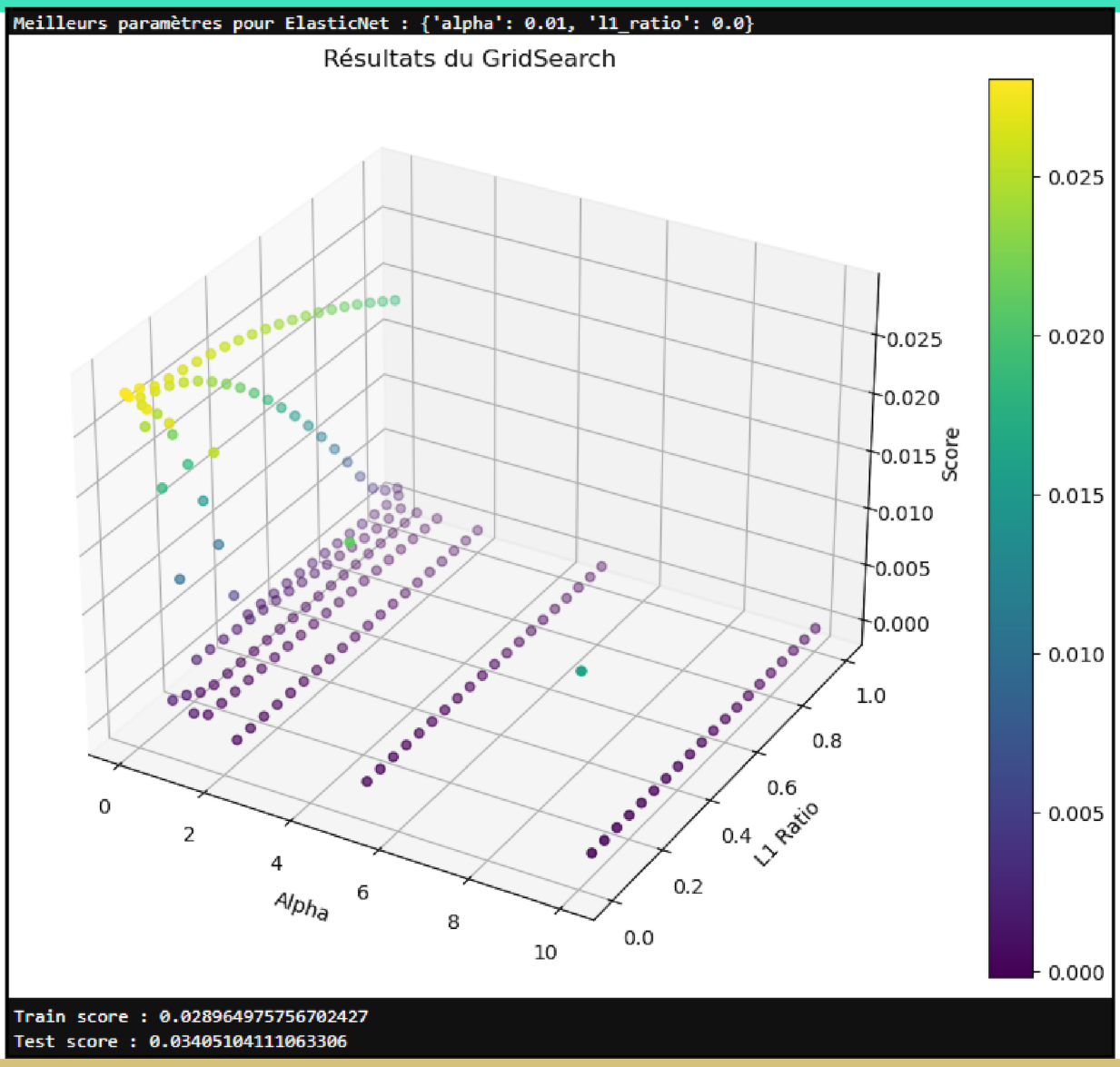
```
# Import des librairies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn import model_selection, preprocessing
from sklearn.model_selection import cross_val_predict, cross_val_score, cross_validate, train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, LassoCV, RidgeCV, ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import OneClassSVM

import warnings
warnings.filterwarnings("ignore")


# Loading du Dataset
df = pd.read_csv('WA_dist_classif_290524.csv')


# Séparation des data et de la target
target = df.record_battu
data = df.drop('record_battu', axis = 1)


# Séparation des données de test et d'entrainement
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size = 0.2, random_state = 123)


# Création des paramètres
params = {'alpha' : [0.01, 0.02, 0.05, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0],
          'l1_ratio' : np.arange(0.0, 1.01, 0.05)}
elastic_grid = GridSearchCV(ElasticNet(), params, cv = 3)
elastic_grid.fit(X_train, y_train)


# Affichage des meilleurs paramètres
print('Meilleurs paramètres pour ElasticNet :', elastic_grid.best_params_)
# Affichage graphique des paramètres
alphas = pd.DataFrame(elastic_grid.cv_results_['param_alpha'])
l1_ratios = pd.DataFrame(elastic_grid.cv_results_['param_l1_ratio'])
scores = pd.DataFrame(elastic_grid.cv_results_['mean_test_score'])


fig = plt.figure(figsize = (12, 8))
ax = fig.add_subplot(111, projection = '3d')

colors = scores

scatter = ax.scatter(alphas, l1_ratios, scores, c=colors, cmap = 'viridis')
fig.colorbar(scatter)
ax.set_xlabel('Alpha')
ax.set_ylabel('L1 Ratio')
ax.set_zlabel('Score')
ax.set_title('Résultats du GridSearch')

plt.show();


# Création du modèle avec les meilleurs paramètres

elastic = ElasticNet(alpha = 0.01, l1_ratio = 0.0)
elastic.fit(X_train, y_train)
elastic_pred = elastic.predict(X_train)
elastic_pred_test = elastic.predict(X_test)


# Affichage des scores du modèle
print('Train score :', elastic.score(X_train, y_train))
print('Test score :', elastic.score(X_test, y_test))
```

Régression linéaire

ELASTIC NET

Dataset Distance

```
Meilleurs paramètres pour ElasticNet : {'alpha': 0.01, 'l1_ratio': 0.0}
```




```
Train score : 0.028964975756702427
Test score : 0.03405104111063306
```