Attribution de poids

import numpy as np

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import make\_scorer, f1\_score

# Créez un modèle de classification (par exemple, RandomForestClassifier)

model = RandomForestClassifier(random\_state=42)

# Définissez les poids de classe à tester

class\_weights\_to\_test = [{0: 1, 1: 1}, {0: 1, 1: 2}, {0: 1, 1: 5}, {0: 1, 1: 10}]

# Définissez la grille de recherche

param\_grid = {'class\_weight': class\_weights\_to\_test}

# Définissez la métrique que vous souhaitez optimiser (dans cet exemple, le F1-score)

scorer = make\_scorer(f1\_score)

# Créez un objet GridSearchCV pour la recherche en grille avec validation croisée

grid\_search = GridSearchCV(estimator=model, param\_grid=param\_grid, scoring=scorer, cv=5)

# Entraînez le modèle avec toutes les combinaisons de poids de classe

grid\_search.fit(X\_train, y\_train)

# Affichez les meilleurs paramètres

print("Meilleurs paramètres :", grid\_search.best\_params\_)

# Affichez la meilleure métrique de performance

print("Meilleur F1-score :", grid\_search.best\_score\_)

# Obtenez le modèle avec les meilleurs paramètres

best\_model = grid\_search.best\_estimator\_

# Évaluez le modèle sur l'ensemble de test

y\_pred = best\_model.predict(X\_test)

f1 = f1\_score(y\_test, y\_pred)

print("F1-score sur l'ensemble de test :", f1)

Yyyuuuuiiiiiiiitt

import pandas as pd

# Créez un DataFrame exemple

data = {'Feature1': [1, 2, 3, 4, 5],

'Feature2': [2, 4, 6, 8, 10],

'Feature3': [3, 6, 9, 12, 15]}

df = pd.DataFrame(data)

# Définissez un seuil de corrélation (par exemple, 0.7)

threshold = 0.7

# Créez une matrice de corrélation

corr\_matrix = df.corr().abs()

# Sélectionnez la partie supérieure de la matrice de corrélation (hors diagonale)

upper = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(np.bool))

# Trouvez les paires de caractéristiques corrélées à supprimer

to\_drop = [column for column in upper.columns if any(upper[column] > threshold)]

# Supprimez les caractéristiques corrélées du DataFrame

df.drop(to\_drop, axis=1, inplace=True)

# Affichez le DataFrame après la suppression

print(df) import pandas as pd

# Créez un DataFrame exemple

data = {'Feature1': [1, 2, 3, 4, 5],

'Feature2': [2, 4, 6, 8, 10],

'Feature3': [3, 6, 9, 12, 15]}

df = pd.DataFrame(data)

# Définissez un seuil de corrélation (par exemple, 0.7)

threshold = 0.7

# Créez une matrice de corrélation

corr\_matrix = df.corr().abs()

# Sélectionnez la partie supérieure de la matrice de corrélation (hors diagonale)

upper = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(np.bool))

# Trouvez les paires de caractéristiques corrélées à supprimer

to\_drop = [column for column in upper.columns if any(upper[column] > threshold)]

# Supprimez les caractéristiques corrélées du DataFrame

df.drop(to\_drop, axis=1, inplace=True)

# Affichez le DataFrame après la suppression

print(df)

Jjjjjgggffgggggferfr

import pandas as pd

import statsmodels.api as sm

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import LogisticRegression # Vous pouvez utiliser votre modèle préféré

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

def evaluate\_model(features):

model = LogisticRegression() # Remplacez par votre modèle

scores = cross\_val\_score(model, X\_train[features], y\_train, cv=5, scoring='accuracy')

return scores.mean()

best\_features = []

best\_score = 0

for feature in X\_train.columns:

current\_features = best\_features + [feature]

score = evaluate\_model(current\_features)

if score > best\_score:

best\_score = score

best\_features = current\_features

final\_model = LogisticRegression() # Remplacez par votre modèle

final\_model.fit(X\_train[best\_features], y\_train)