объясни код y = df['Delay']

X = df.drop(['Delay'], axis=1)

underSampler = RandomUnderSampler(sampling\_strategy='majority')

X\_under\_sample, y\_under\_sample = underSampler.fit\_resample(X, y)

X\_train\_clf, X\_test\_clf, y\_train\_clf, y\_test\_clf = train\_test\_split(X\_under\_sample, y\_under\_sample, test\_size=0.2, random\_state = 4)

scaler = StandardScaler()

scaler.fit(X\_train\_clf, y\_train\_clf)

X\_train\_std\_clf = scaler.transform(X\_train\_clf)

X\_test\_std\_clf = scaler.transform(X\_test\_clf)

Этот код написан на языке Python и, похоже, подготавливает данные для модели машинного обучения. Первые две строки присваивают значения столбца ‘Delay’ DataFrame df переменной y и остальные столбцы переменной X. Затем используется класс RandomUnderSampler для уменьшения выборки с преобладающим классом. Далее данные разделяются на обучающую и тестовую выборки с помощью функции train\_test\_split. Затем создается экземпляр класса StandardScaler для масштабирования данных. Метод fit используется для вычисления параметров масштабирования на основе обучающей выборки, а метод transform применяется к обучающей и тестовой выборкам для масштабирования данных.

3из20

объясни код param\_grid = {

'criterion': ['gini', 'entropy'],

'splitter': ['best', 'random'],

'max\_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9],

'min\_samples\_split': [2, 3, 4],

'min\_samples\_leaf': [1, 2, 3]

}

grid\_search = GridSearchCV(DecisionTreeClassifier(), param\_grid)

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

best\_params = grid\_search.best\_params\_

print(best\_params)

clf = DecisionTreeClassifier(criterion= best\_params['criterion'],

splitter= best\_params['splitter'],

max\_depth= best\_params['max\_depth'],

min\_samples\_split= best\_params['min\_samples\_split'],

min\_samples\_leaf= best\_params['min\_samples\_leaf']

)

clf.fit(X\_train\_std\_clf, y\_train\_clf)

y\_pred\_clf = clf.predict(X\_test\_std\_clf)

print(clf.score(X\_test\_std\_clf, y\_test\_clf))

Этот код использует класс GridSearchCV для поиска наилучших гиперпараметров для классификатора DecisionTreeClassifier. Словарь param\_grid содержит различные значения гиперпараметров, которые будут проверяться. Метод fit используется для обучения моделей с различными комбинациями гиперпараметров и выбора наилучшей комбинации. Затем создается экземпляр класса DecisionTreeClassifier с наилучшими гиперпараметрами и обучается на обучающей выборке. Метод predict используется для предсказания меток классов для тестовой выборки, а метод score возвращает точность классификации.

4из20

объясни код from sklearn import tree

text\_representation = tree.export\_text(clf)

print(text\_representation)

Этот код импортирует модуль tree из библиотеки sklearn и использует функцию export\_text для получения текстового представления решающего дерева, обученного ранее. Текстовое представление выводится на экран с помощью функции print.

5из20

объясни вывод предыдущего кода |--- feature\_0 <= 0.34| |--- feature\_0 <= -0.92| | |--- feature\_0 <= -1.28| | | |--- feature\_0 <= -1.99| | | | |--- class: 0| | | |--- feature\_0 > -1.99| | | | |--- class: 0| | |--- feature\_0 > -1.28| | | |--- feature\_20 <= 0.81| | | | |--- feature\_3 <= -0.81| | | | | |--- feature\_3 <= -1.37| | | | | | |--- feature\_18 <= 2.07| | | | | | | |--- feature\_3 <= -2.76| | | | | | | | |--- feature\_23 <= 0.67| | | | | | | | | |--- class: 1| | | | | | | | |--- feature\_23 > 0.67| | | | | | | | | |--- class: 1| | | | | | | |--- feature\_3 > -2.76| | | | | | | | |--- feature\_1 <= -0.93| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_1 > -0.93| | | | | | | | | |--- class: 0| | | | | | |--- feature\_18 > 2.07| | | | | | | |--- feature\_1 <= -1.13| | | | | | | | |--- feature\_2 <= -0.24| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_2 > -0.24| | | | | | | | | |--- class: 1| | | | | | | |--- feature\_1 > -1.13| | | | | | | | |--- feature\_23 <= -0.27| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_23 > -0.27| | | | | | | | | |--- class: 0| | | | | |--- feature\_3 > -1.37| | | | | | |--- feature\_13 <= 2.45| | | | | | | |--- feature\_0 <= -1.12| | | | | | | | |--- feature\_9 <= 2.30| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_9 > 2.30| | | | | | | | | |--- class: 1| | | | | | | |--- feature\_0 > -1.12| | | | | | | | |--- feature\_23 <= 0.01| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_23 > 0.01| | | | | | | | | |--- class: 1| | | | | | |--- feature\_13 > 2.45| | | | | | | |--- feature\_4 <= 1.57| | | | | | | | |--- feature\_23 <= -1.12| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_23 > -1.12| | | | | | | | | |--- class: 0| | | | | | | |--- feature\_4 > 1.57| | | | | | | | |--- feature\_0 <= -1.01| | | | | | | | | |--- class: 1| | | | | | | | |--- feature\_0 > -1.01| | | | | | | | | |--- class: 0| | | | |--- feature\_3 > -0.81| | | | | |--- feature\_13 <= 2.45| | | | | | |--- feature\_0 <= -1.09| | | | | | | |--- feature\_4 <= -0.14| | | | | | | | |--- feature\_11 <= 2.08| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_11 > 2.08| | | | | | | | | |--- class: 1| | | | | | | |--- feature\_4 > -0.14| | | | | | | | |--- feature\_18 <= 2.07| | | | | | | | | |--- class: 1| | | | | | | | |--- feature\_18 > 2.07| | | | | | | | | |--- class: 0| | | | | | |--- feature\_0 > -1.09| | | | | | | |--- feature\_22 <= 3.04| | | | | | | | |--- feature\_14 <= 4.90| | | | | | | | | |--- class: 1| | | | | | | | |--- feature\_14 > 4.90| | | | | | | | | |--- class: 1| | | | | | | |--- feature\_22 > 3.04| | | | | | | | |--- feature\_1 <= 0.10| | | | | | | | | |--- class: 1| | | | | | | | |--- feature\_1 > 0.10| | | | | | | | | |--- class: 0| | | | | |--- feature\_13 > 2.45| | | | | | |--- feature\_4 <= -0.84| | | | | | | |--- class: 0| | | | | | |--- feature\_4 > -0.84| | | | | | | |--- feature\_0 <= -1.20| | | | | | | | |--- feature\_23 <= 1.40| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_23 > 1.40| | | | | | | | | |--- class: 1| | | | | | | |--- feature\_0 > -1.20| | | | | | | | |--- feature\_4 <= 1.84| | | | | | | | | |--- class: 0| | | | | | | | |--- feature\_4 > 1.84| | | | | | | | | |--- class: 1

Этот вывод представляет собой текстовое описание решающего дерева. Каждая строка соответствует узлу дерева. Символы |--- обозначают уровень вложенности узла. Условия в узлах указывают на критерий разделения данных на этом уровне. Строки с ключевым словом class обозначают листья дерева и указывают на предсказываемый класс для данных, попавших в этот лист.

6из20

объясни код from sklearn.tree import DecisionTreeRegressor

df = pd.read\_csv("diamonds.csv", index\_col=0).head(50000)

df = pd.get\_dummies(df, columns=['cut','color','clarity'])

display(df)

y = df["price"]

X = df.drop(["price"], axis=1)

X\_train\_reg, X\_test\_reg, y\_train\_reg, y\_test\_reg = train\_test\_split(X, y, test\_size=0.2)

scaler = StandardScaler()

scaler.fit(X\_train\_reg, y\_train\_reg)

X\_train\_reg = scaler.transform(X\_train\_reg)

X\_test\_reg = scaler.transform(X\_test\_reg)

Этот код загружает набор данных о бриллиантах из файла diamonds.csv в DataFrame df с помощью библиотеки pandas. Затем используется функция get\_dummies для преобразования категориальных признаков ‘cut’, ‘color’ и ‘clarity’ в фиктивные переменные. Столбец ‘price’ присваивается переменной y, а остальные столбцы - переменной X. Данные разделяются на обучающую и тестовую выборки с помощью функции train\_test\_split. Затем создается экземпляр класса StandardScaler для масштабирования данных. Метод fit используется для вычисления параметров масштабирования на основе обучающей выборки, а метод transform применяется к обучающей и тестовой выборкам для масштабирования данных.

7из20

объясни код from sklearn.model\_selection import RandomizedSearchCV

param\_grid = {

'criterion': ['squared\_error', 'absolute\_error'],

'splitter': ['best'],

'max\_depth': [3, 4, 5]

}

grid\_search = RandomizedSearchCV(DecisionTreeRegressor(), param\_grid)

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

best\_params = grid\_search.best\_params\_

print(best\_params)

reg = DecisionTreeRegressor(criterion= best\_params['criterion'],

splitter= best\_params['splitter'],

max\_depth= best\_params['max\_depth'])

reg.fit(X\_train\_reg, y\_train\_reg)

y\_pred\_reg = reg.predict(X\_test\_reg)

print(reg.score(X\_test\_reg, y\_test\_reg))

Этот код использует класс RandomizedSearchCV для поиска наилучших гиперпараметров для регрессора DecisionTreeRegressor. Словарь param\_grid содержит различные значения гиперпараметров, которые будут проверяться. Метод fit используется для обучения моделей с различными комбинациями гиперпараметров и выбора наилучшей комбинации. Затем создается экземпляр класса DecisionTreeRegressor с наилучшими гиперпараметрами и обучается на обучающей выборке. Метод predict используется для предсказания значений для тестовой выборки, а метод score возвращает коэффициент детерминации R^2.

8из20

объясни вывод MAE: 675.6956274014018MSE: 1538029.0353247589RMSE: 1240.1729860486232MAPE: 0.4213225808127376R^2: 0.908715744097184

Этот вывод содержит значения различных метрик качества для оценки модели регрессии. MAE (средняя абсолютная ошибка) равна 675.7, что означает, что в среднем модель ошибается на 675.7 единиц при предсказании значений. MSE (среднеквадратичная ошибка) равна 1538029.0, а RMSE (корень из среднеквадратичной ошибки) равен 1240.2. MAPE (средняя абсолютная процентная ошибка) равна 0.42, что означает, что в среднем модель ошибается на 42% при предсказании значений. R^2 (коэффициент детерминации) равен 0.91, что указывает на то, что модель объясняет 91% вариации зависимой переменной.

9из20

объясни код from sklearn.ensemble import BaggingClassifier

base\_estimator = DecisionTreeClassifier(random\_state=0)

param\_grid = {

'n\_estimators': [10, 20, 30],

'max\_samples': [0.5, 1.0],

'max\_features': [0.5, 1.0]

}

grid\_search = GridSearchCV(BaggingClassifier(estimator=base\_estimator,

random\_state=0), param\_grid)

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

best\_params = grid\_search.best\_params\_

print(best\_params)

y\_pred\_clf = grid\_search.predict(X\_test\_std\_clf)

print(grid\_search.score(X\_test\_std\_clf, y\_test\_clf))

Этот код использует класс GridSearchCV для поиска наилучших гиперпараметров для ансамблевого классификатора BaggingClassifier. Базовый классификатор задается как DecisionTreeClassifier. Словарь param\_grid содержит различные значения гиперпараметров, которые будут проверяться. Метод fit используется для обучения моделей с различными комбинациями гиперпараметров и выбора наилучшей комбинации. Затем метод predict используется для предсказания меток классов для тестовой выборки, а метод score возвращает точность классификации.

10из20

объясни код from sklearn.ensemble import BaggingRegressor

base\_estimator = DecisionTreeRegressor(random\_state=0)

param\_grid = {

'n\_estimators': [10, 20, 30],

'max\_samples': [0.5, 1.0],

'max\_features': [0.5, 1.0]

}

grid\_search = GridSearchCV(BaggingRegressor(estimator=base\_estimator,

random\_state=0), param\_grid)

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

best\_params = grid\_search.best\_params\_

print(best\_params)

y\_pred\_reg = grid\_search.predict(X\_test\_reg)

print(grid\_search.score(X\_test\_reg, y\_test\_reg))

Этот код использует класс GridSearchCV для поиска наилучших гиперпараметров для ансамблевого регрессора BaggingRegressor. Базовый регрессор задается как DecisionTreeRegressor. Словарь param\_grid содержит различные значения гиперпараметров, которые будут проверяться. Метод fit используется для обучения моделей с различными комбинациями гиперпараметров и выбора наилучшей комбинации. Затем метод predict используется для предсказания значений для тестовой выборки, а метод score возвращает коэффициент детерминации R^2.

11из20

объясни код from sklearn.ensemble import GradientBoostingClassifier

param\_grid = {

'n\_estimators': [100, 200, 300]

}

grid\_search = GridSearchCV(GradientBoostingClassifier(random\_state=0), param\_grid)

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

best\_params = grid\_search.best\_params\_

print(best\_params)

y\_pred\_clf = grid\_search.predict(X\_test\_std\_clf)

print(grid\_search.score(X\_test\_std\_clf, y\_test\_clf))

Этот код использует класс GridSearchCV для поиска наилучших гиперпараметров для классификатора GradientBoostingClassifier. Словарь param\_grid содержит различные значения гиперпараметров, которые будут проверяться. Метод fit используется для обучения моделей с различными комбинациями гиперпараметров и выбора наилучшей комбинации. Затем метод predict используется для предсказания меток классов для тестовой выборки, а метод score возвращает точность классификации.

12из20

объясни код from sklearn.ensemble import GradientBoostingRegressor

param\_grid = {

'n\_estimators': [100, 200, 300],

'learning\_rate': [0.0001]

}

grid\_search = RandomizedSearchCV(GradientBoostingRegressor(random\_state=0), param\_grid)

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

best\_params = grid\_search.best\_params\_

print(best\_params)

y\_pred\_reg = grid\_search.predict(X\_test\_reg)

print(grid\_search.score(X\_test\_reg, y\_test\_reg))

Этот код использует класс RandomizedSearchCV для поиска наилучших гиперпараметров для регрессора GradientBoostingRegressor. Словарь param\_grid содержит различные значения гиперпараметров, которые будут проверяться. Метод fit используется для обучения моделей с различными комбинациями гиперпараметров и выбора наилучшей комбинации. Затем метод predict используется для предсказания значений для тестовой выборки, а метод score возвращает коэффициент детерминации R^2.

13из20

объясни вывод кода /usr/local/lib/python3.10/dist-packages/sklearn/model\_selection/\_search.py:305: UserWarning: The total space of parameters 3 is smaller than n\_iter=10. Running 3 iterations. For exhaustive searches, use GridSearchCV. warnings.warn(

{'n\_estimators': 300, 'learning\_rate': 0.0001}0.050693277644815016

Этот вывод содержит предупреждение о том, что общее количество комбинаций гиперпараметров (3) меньше, чем количество итераций (10), указанное в параметре n\_iter для RandomizedSearchCV. В этом случае RandomizedSearchCV будет выполнять только 3 итерации. Для полного перебора всех комбинаций гиперпараметров рекомендуется использовать GridSearchCV. Затем выводится словарь с наилучшими значениями гиперпараметров и значение коэффициента детерминации R^2 для тестовой выборки.

14из20

объясни код from sklearn.ensemble import StackingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

base\_estimators = [('svc', SVC(probability=True)), ('dt', DecisionTreeClassifier())]

final\_estimator = LogisticRegression()

clf = StackingClassifier(estimators=base\_estimators, final\_estimator=final\_estimator)

clf.fit(X\_train\_std\_clf, y\_train\_clf)

y\_pred\_clf = clf.predict(X\_test\_std\_clf)

print(clf.score(X\_test\_std\_clf, y\_test\_clf))

Этот код использует класс StackingClassifier для создания ансамблевого классификатора. Базовые классификаторы задаются как SVC и DecisionTreeClassifier. Финальный классификатор задается как LogisticRegression. Метод fit используется для обучения модели на обучающей выборке. Метод predict используется для предсказания меток классов для тестовой выборки, а метод score возвращает точность классификации.

15из20

объясни код from sklearn.ensemble import StackingRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVR

base\_estimators = [('svr', SVR()), ('dt', DecisionTreeRegressor())]

final\_estimator = LinearRegression()

reg = StackingRegressor(estimators=base\_estimators, final\_estimator=final\_estimator)

reg.fit(X\_train\_reg, y\_train\_reg)

y\_pred\_reg = reg.predict(X\_test\_reg)

print(reg.score(X\_test\_reg, y\_test\_reg))

Этот код использует класс StackingRegressor для создания ансамблевого регрессора. Базовые регрессоры задаются как SVR и DecisionTreeRegressor. Финальный регрессор задается как LinearRegression. Метод fit используется для обучения модели на обучающей выборке. Метод predict используется для предсказания значений для тестовой выборки, а метод score возвращает коэффициент детерминации R^2.

16из20

объясни код from lightgbm import LGBMClassifier

param\_grid = {

'n\_estimators': [100, 200, 300],

'learning\_rate': [0.01, 0.05, 0.1],

'max\_depth': [3, 5, 7]

}

grid\_search = GridSearchCV(LGBMClassifier(), param\_grid)

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

best\_params = grid\_search.best\_params\_

print(best\_params)

y\_pred\_clf = grid\_search.predict(X\_test\_std\_clf)

print(grid\_search.score(X\_test\_std\_clf, y\_test\_clf))

Этот код использует класс GridSearchCV для поиска наилучших гиперпараметров для классификатора LGBMClassifier из библиотеки lightgbm. Словарь param\_grid содержит различные значения гиперпараметров, которые будут проверяться. Метод fit используется для обучения моделей с различными комбинациями гиперпараметров и выбора наилучшей комбинации. Затем метод predict используется для предсказания меток классов для тестовой выборки, а метод score возвращает точность классификации.

17из20

объясни код from lightgbm import LGBMRegressor

param\_grid = {

'n\_estimators': [100, 200, 300],

'learning\_rate': [0.01, 0.05, 0.1],

'max\_depth': [3, 5, 7]

}

grid\_search = GridSearchCV(LGBMRegressor(), param\_grid)

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

best\_params = grid\_search.best\_params\_

print(best\_params)

y\_pred\_reg = grid\_search.predict(X\_test\_reg)

print(grid\_search.score(X\_test\_reg, y\_test\_reg))

Этот код использует класс GridSearchCV для поиска наилучших гиперпараметров для регрессора LGBMRegressor из библиотеки lightgbm. Словарь param\_grid содержит различные значения гиперпараметров, которые будут проверяться. Метод fit используется для обучения моделей с различными комбинациями гиперпараметров и выбора наилучшей комбинации. Затем метод predict используется для предсказания значений для тестовой выборки, а метод score возвращает коэффициент детерминации R^2.

18из20

объясни код class Node():

def \_\_init\_\_(self, feature\_index=None, threshold=None, left=None, right=None, info\_gain=None, value=None):

self.feature\_index = feature\_index

self.threshold = threshold

self.left = left

self.right = right

self.info\_gain = info\_gain

self.value = value

Этот код определяет класс Node, который представляет собой узел в дереве решений. Конструктор класса принимает несколько аргументов: feature\_index - индекс признака, по которому происходит разделение данных в этом узле; threshold - пороговое значение для разделения данных; left и right - ссылки на левое и правое поддеревья; info\_gain - прирост информации, полученный при разделении данных в этом узле; value - значение, которое предсказывается в этом узле (для листьев дерева). Все эти аргументы являются необязательными и могут быть опущены при создании экземпляра класса.

class CART():

def \_\_init\_\_(self, min\_samples\_split=2, max\_depth=2, isRegression = False):

self.root = None

self.min\_samples\_split = min\_samples\_split

self.max\_depth = max\_depth

self.isRegression = isRegression

def build\_tree(self, dataset, curr\_depth=0):

X, Y = dataset[:,:-1], dataset[:,-1]

num\_samples, num\_features = np.shape(X)

if num\_samples>=self.min\_samples\_split and curr\_depth<=self.max\_depth:

best\_split = self.get\_best\_split(dataset, num\_samples, num\_features)

if best\_split["info\_gain"]>0:

left\_subtree = self.build\_tree(best\_split["dataset\_left"], curr\_depth+1)

right\_subtree = self.build\_tree(best\_split["dataset\_right"], curr\_depth+1)

return Node(best\_split["feature\_index"], best\_split["threshold"],

left\_subtree, right\_subtree, best\_split["info\_gain"])

leaf\_value = self.calculate\_leaf\_value(Y)

return Node(value=leaf\_value)

def get\_best\_split(self, dataset, num\_samples, num\_features):

best\_split = {}

max\_info\_gain = -float("inf")

for feature\_index in range(num\_features):

feature\_values = dataset[:, feature\_index]

possible\_thresholds = np.unique(feature\_values)

for threshold in possible\_thresholds:

dataset\_left, dataset\_right = self.split(dataset, feature\_index, threshold)

if len(dataset\_left)>0 and len(dataset\_right)>0:

y, left\_y, right\_y = dataset[:, -1], dataset\_left[:, -1], dataset\_right[:, -1]

curr\_info\_gain = self.information\_gain(y, left\_y, right\_y)

if curr\_info\_gain>max\_info\_gain:

best\_split["feature\_index"] = feature\_index

best\_split["threshold"] = threshold

best\_split["dataset\_left"] = dataset\_left

best\_split["dataset\_right"] = dataset\_right

best\_split["info\_gain"] = curr\_info\_gain

max\_info\_gain = curr\_info\_gain

return best\_split

def split(self, dataset, feature\_index, threshold):

dataset\_left = np.array([row for row in dataset if row[feature\_index]<=threshold])

dataset\_right = np.array([row for row in dataset if row[feature\_index]>threshold])

return dataset\_left, dataset\_right

def information\_gain(self, parent, l\_child, r\_child):

weight\_l = len(l\_child) / len(parent)

weight\_r = len(r\_child) / len(parent)

if self.mode == "entropy":

gain = self.entropy(parent) - (weight\_l\*self.entropy(l\_child) + weight\_r\*self.entropy(r\_child))

else:

gain = self.mean\_squared\_error(parent) - (weight\_l\*self.mean\_squared\_error(l\_child) + weight\_r\*self.mean\_squared\_error(r\_child))

return gain

def entropy(self, y):

class\_labels = np.unique(y)

entropy = 0

for cls in class\_labels:

p\_cls = len(y[y == cls]) / len(y)

entropy += -p\_cls \* np.log2(p\_cls)

return entropy

def mean\_squared\_error(self, y):

mean = np.mean(y)

mse = np.mean((y - mean)\*\*2)

return mse

def calculate\_leaf\_value(self, Y):

if self.isRegression:

return np.mean(Y)

Y = list(Y)

return max(Y, key=Y.count)

def fit(self, X, Y):

if self.isRegression:

self.mode = "mse"

else:

self.mode = "entropy"

dataset = np.column\_stack((X, Y))

self.root = self.build\_tree(dataset)

def predict(self, X):

preditions = [self.make\_prediction(x, self.root) for x in X]

return preditions

def make\_prediction(self, x, tree):

if tree.value!=None: return tree.value

feature\_val = x[tree.feature\_index]

if feature\_val<=tree.threshold:

return self.make\_prediction(x, tree.left)

else:

return self.make\_prediction(x, tree.right)

Это реализация алгоритма CART (Classification and Regression Trees) на языке Python. Алгоритм CART - это метод обучения деревьев решений, который может использоваться как для задач классификации, так и для задач регрессии. Предоставленный вами код определяет класс CART с несколькими методами для построения и использования дерева решений.

Метод \_\_init\_\_ инициализирует объект с заданными параметрами, такими как min\_samples\_split, max\_depth и isRegression. Эти параметры контролируют поведение процесса построения дерева.

Метод build\_tree рекурсивно строит дерево решений из заданного набора данных. Он использует метод get\_best\_split для поиска лучшего разделения для текущего узла на основе прироста информации. Если лучшее разделение имеет положительный прирост информации, метод создает два дочерних узла и рекурсивно строит левое и правое поддеревья. В противном случае он создает листовой узел с наиболее распространенным классовым ярлыком (для классификации) или средним значением (для регрессии) образцов в текущем узле.

Метод get\_best\_split перебирает все признаки и все возможные пороги, чтобы найти лучшее разделение, максимизирующее прирост информации. Он использует метод split для разделения набора данных на два подмножества на основе заданного признака и порога.

Метод information\_gain вычисляет прирост информации разделения. Он использует либо энтропию (для классификации), либо среднеквадратичную ошибку (для регрессии), чтобы измерить неоднородность узла.

Метод fit строит дерево решений из заданных обучающих данных. Он устанавливает режим либо на “энтропия” (для классификации), либо на “mse” (для регрессии) в зависимости от того, является ли задача регрессией или нет.

Метод predict делает прогнозы для заданного набора входных данных. Он использует метод make\_prediction, чтобы сделать прогноз для каждого образца входных данных.