Machine Learning and Data Mining (Module 2)

Matteo Francia
DISI — University of Bologna
m.francia@unibo.it

Open in Colab

The `California Housing` case study

Our task is to use California census data to forecast housing prices given the population, median income, and median housing price for each block group in California. Block groups are the smallest geographical unit for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). We will just call them “districts” for short.

Solution of the previous Hands on

▼ Code

```python
def preprocess(normalize=True):
    df = pd.read_csv("https://raw.githubusercontent.com/w4bo/handsOnDataPipelines/main/materials/datasets/housing.csv", delimiter=",")
    num_df = df.drop(columns=["ocean_proximity", "median_house_value"])
    # Filling in (i.e., impute) missing values with the median value
    num_df["total_bedrooms"] = num_df["total_bedrooms"].fillna(num_df["total_bedrooms"].median())
    # Add a new column: population_per_household = population / households
    num_df["population_per_household"] = num_df["population"] / num_df["households"]
    # Add a new column: rooms_per_household = total_rooms / households
    num_df["rooms_per_household"] = num_df["total_rooms"] / num_df["households"]
    # Add a new column: bedrooms_per_room = total_bedrooms / total_rooms
    num_df["bedrooms_per_room"] = num_df["total_bedrooms"] / num_df["total_rooms"]
    if normalize:
        # Apply standardization to all the numeric columns
        num_df = (num_df - num_df.mean()) / num_df.std()
    # One hot encode `ocean_proximity` since it is a categorical attribute
    cat_df = pd.get_dummies(df["ocean_proximity"], prefix='ocean_proximity')
    # Join all the dataframes
    return pd.concat([num_df, cat_df, df[["median_house_value"]]], axis=1)


df = preprocess()
df
```

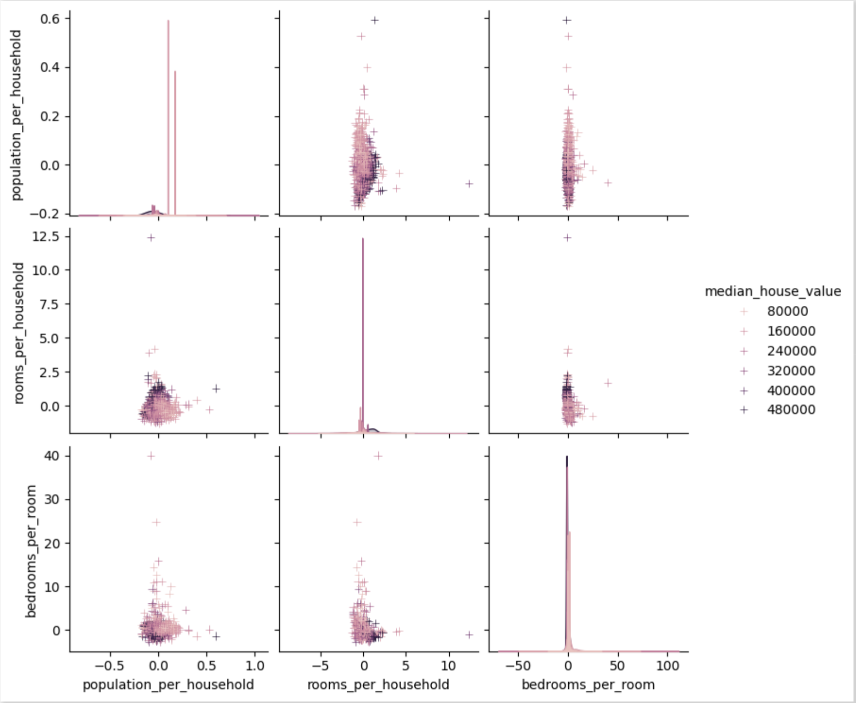
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	population_per_household	rooms_per_household	bedrooms_per_room	ocean_proximity O
0	-1.327803	1.052523	0.982119	-0.804800	-0.972453	-0.974405	-0.977009	2.344709	-0.049595	0.628544	-1.029963	False
1	-1.322812	1.043159	-0.607004	2.045841	1.357111	0.861418	1.669921	2.332181	-0.092510	0.327033	-0.888876	False
2	-1.332794	1.038478	1.856137	-0.535733	-0.827004	-0.820757	-0.843616	1.782656	-0.025842	1.155592	-1.291654	False
3	-1.337785	1.038478	1.856137	-0.624199	-0.719706	-0.766010	-0.733764	0.932945	-0.050328	0.156962	-0.449602	False
4	-1.337785	1.038478	1.856137	-0.462393	-0.612408	-0.759828	-0.629142	-0.012881	-0.085614	0.344702	-0.639071	False
...	...	...	...	...	...	...	...	...	...	...	...	...
20635	-0.758808	1.801603	-0.289180	-0.444974	-0.388274	-0.512579	-0.443438	-1.216099	-0.049109	-0.155020	0.165990	False
20636	-0.818702	1.806285	-0.845373	-0.888682	-0.922380	-0.944382	-1.008396	-0.691576	0.005021	0.276874	0.021670	False
20637	-0.823693	1.778194	-0.924829	-0.174991	-0.123605	-0.369528	-0.174037	-1.142566	-0.071733	-0.090316	0.021134	False
20638	-0.873605	1.778194	-0.845373	-0.355591	-0.304820	-0.604415	-0.393743	-1.054557	-0.091223	-0.040210	0.093464	False
20639	-0.833676	1.750104	-1.004285	0.068407	0.188792	-0.053976	0.079630	-0.780111	-0.043681	-0.070441	0.113272	False

20640 rows × 17 columns

Checking feature correlations

▼ Code

```
sns.pairplot(
    df[["population_per_household", "rooms_per_household", "bedrooms_per_room", "median_house_value"]].sample(n=1000, random_state=42),
    hue='median_house_value', markers='+'
)
```



Splitting training and test data

▼ Code

```python
from sklearn.model_selection import train_test_split
y = df["median_house_value"]  # labels
X = df.drop(columns=["median_house_value"])  # input data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"X_train: {X_train.shape}")
print(f"X_test: {X_test.shape}")
print(f"y_train: {y_train.shape}")
print(f"y_test: {y_test.shape}")
```

```
X_train: (16512, 16)
X_test: (4128, 16)
y_train: (16512,)
y_test: (4128,)
```

▼ Code

```python
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16512 entries, 14196 to 15795
Data columns (total 16 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   longitude                   16512 non-null  float64
 1   latitude                    16512 non-null  float64
 2   housing_median_age          16512 non-null  float64
 3   total_rooms                 16512 non-null  float64
 4   total_bedrooms              16512 non-null  float64
 5   population                  16512 non-null  float64
 6   households                  16512 non-null  float64
 7   median_income               16512 non-null  float64
 8   population_per_household    16512 non-null  float64
 9   rooms_per_household         16512 non-null  float64
 10  bedrooms_per_room           16512 non-null  float64
 11  ocean_proximity_<1H OCEAN   16512 non-null  bool   
 12  ocean_proximity_INLAND      16512 non-null  bool   
 13  ocean_proximity_ISLAND      16512 non-null  bool   
 14  ocean_proximity_NEAR BAY    16512 non-null  bool   
 15  ocean_proximity_NEAR OCEAN  16512 non-null  bool   
dtypes: bool(5), float64(11)
memory usage: 1.6 MB
```

Regression

LinearRegression fits a linear model with coefficients $w = (w_1, ..., w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

▼ Code

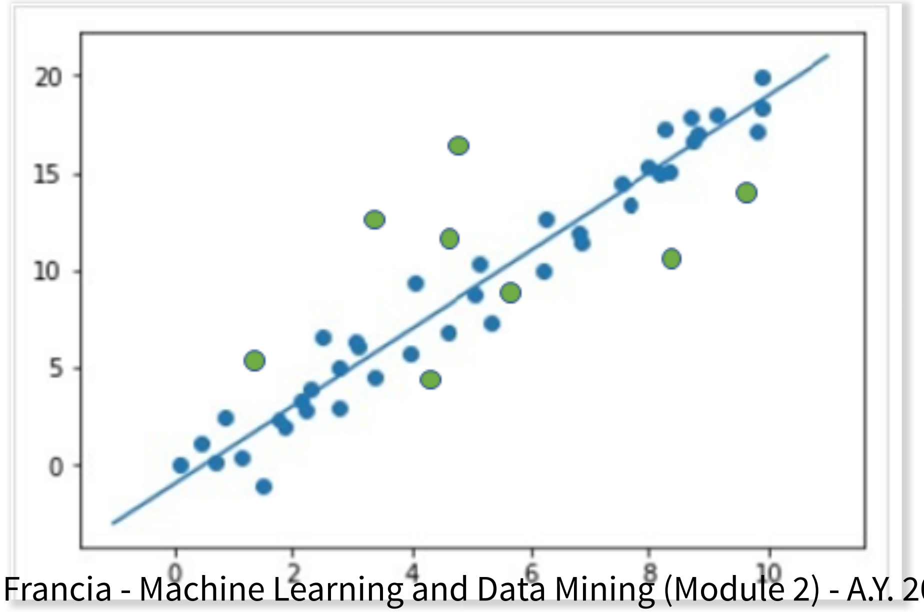
```
from sklearn.linear_model import LinearRegression  # choose and import the model
lin_reg = LinearRegression(fit_intercept=True)  # choose model hyperparameters and initialize the model (i.e., the estimator)
lin_reg.fit(X_train, y_train)  # model fitting
lin_reg.coef_  # return the learned parameters
```

```
array([-56218.2257606 , -56621.04836643,  14103.74699543,   5900.12924673,
         5312.69799925, -46134.52612316,  40378.76502969,  78721.51735298,
          673.09498837,   7979.62051402,  18911.78683053, -18666.47855331,
       -53617.84037612, 112060.2444519 , -24109.18513167, -15666.74039081])
```

▼ Code

```
housing_predictions = lin_reg.predict(X_test)  # predict the cost of the houses in the test set
housing_predictions
```

```
array([ 61463.74466641, 121631.21809998, 267594.25389058, ...,
       447837.04647878, 117275.9214608 , 185597.46125194])
```

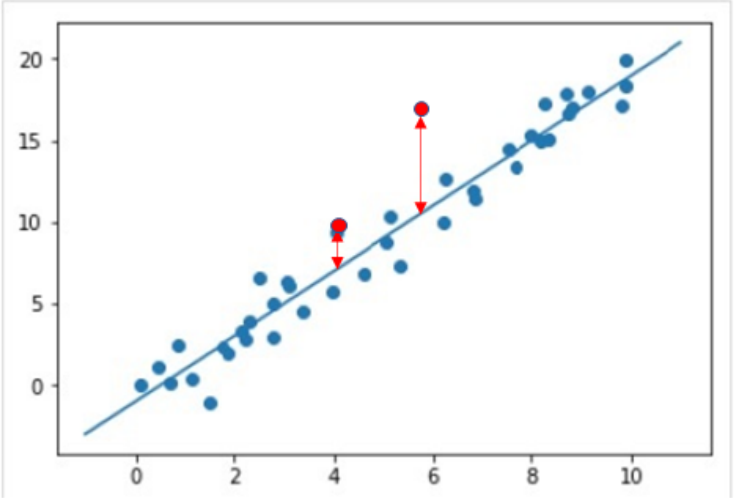


image

Measuring performance

We are facing a *regression problem*

- A typical performance measure for regression problems is the *Root Mean Square Error (RMSE)*
- RMSE is the standard deviation of the residuals (prediction errors)
- Residuals measures how far from the regression line data points are; RMSE is a measure of how spread out these residuals are



$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

Computing the RMSE

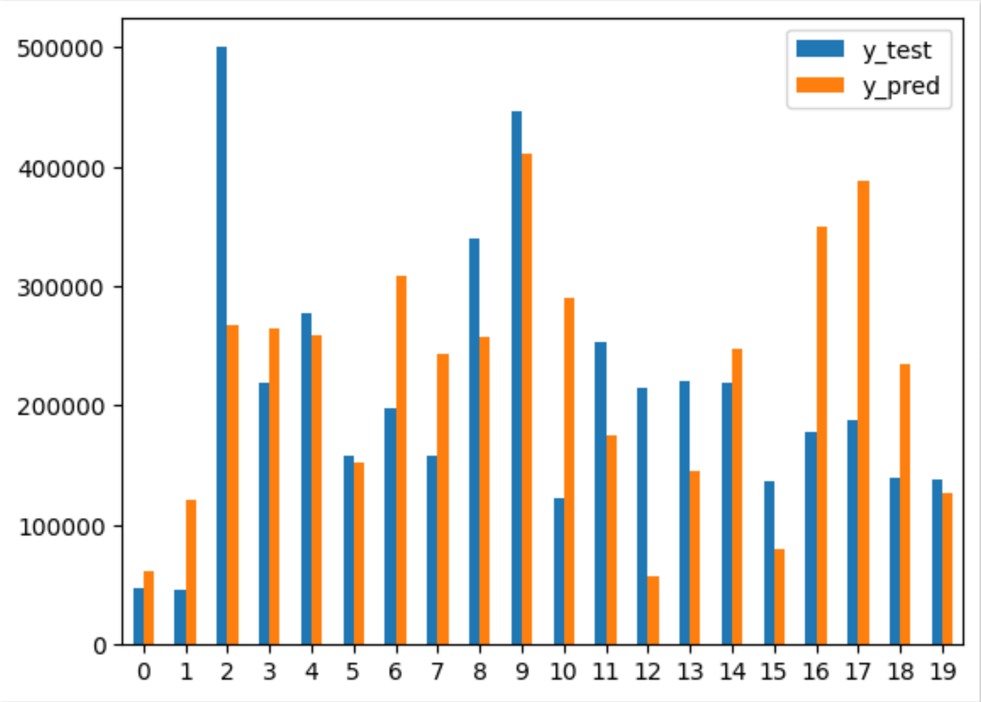
▼ Code

```
from sklearn.metrics import root_mean_squared_error
from sklearn.metrics import accuracy_score

def plot_predictions(y_test, housing_predictions, show=20, accuracy=False, plot=True):
    if not accuracy:
        print(f"RMSE: {root_mean_squared_error(y_test, housing_predictions)}")  # check the error
    else:
        print(f"Accuracy: {accuracy_score(y_test, housing_predictions)}")  # check the error
    if plot: pd.DataFrame({'y_test': y_test[:show].to_numpy(), 'y_pred': housing_predictions[:show]}, index=[x for x in range(show)]).plot.bar(rot=0)  # visualize some predictions

plot_predictions(y_test, housing_predictions)
```

```
RMSE: 72668.53837868225
```



Are we satisfied?

This is better than nothing, but clearly not a great score: most districts' `median_housing_values` range between 120K USD and 265K USD, so a typical prediction error of ~70K USD is not very satisfying.

This is an example of a model underfitting the training data.

Random forest regressor

A random forest fits decision trees on sub-samples of the dataset and uses averaging to improve accuracy and control over-fitting.

The sample size is controlled with `max_samples` if `bootstrap=True`, otherwise the whole dataset is used to build each tree.

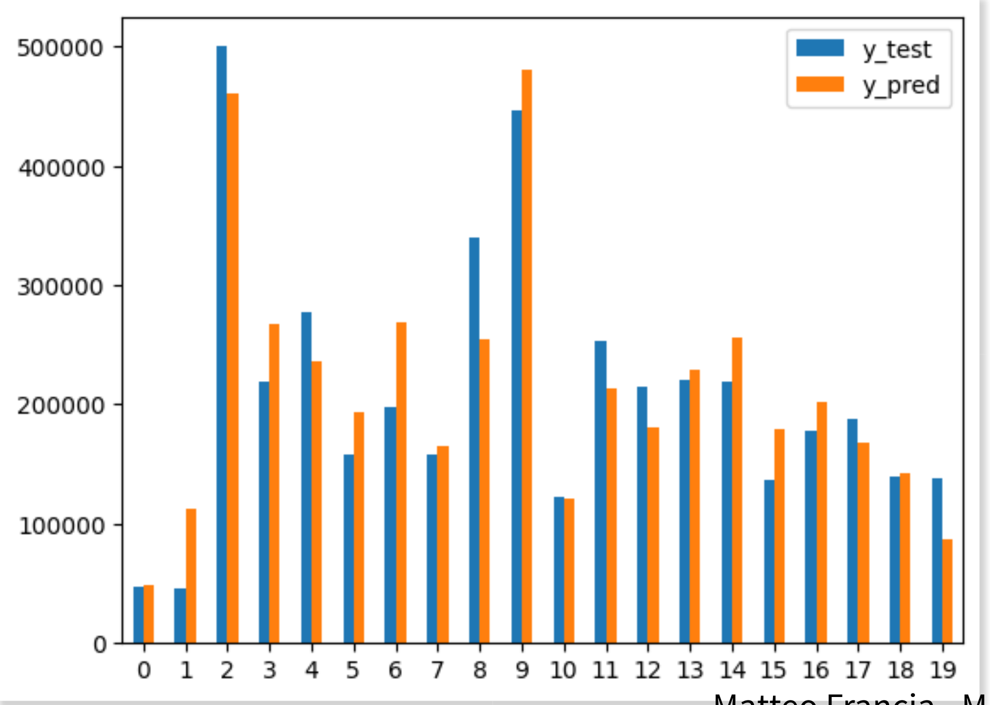
▼ Code

```
from sklearn.ensemble import RandomForestRegressor  # import the model

def run_forest(n_estimators, max_features, min_samples_split=5):
    forest_reg = RandomForestRegressor(n_estimators=n_estimators, max_features=max_features, random_state=42, bootstrap=True)  # initialize the model (i.e., the estimator)
    forest_reg.fit(X_train, y_train)  # train it
    housing_predictions = forest_reg.predict(X_test)  # predict the cost of houses in the test set
    plot_predictions(y_test, housing_predictions)
    return forest_reg

forest_reg = run_forest(100, 1.0)
```

```
RMSE: 50402.56458314594
```



Feature importance

▼ Code

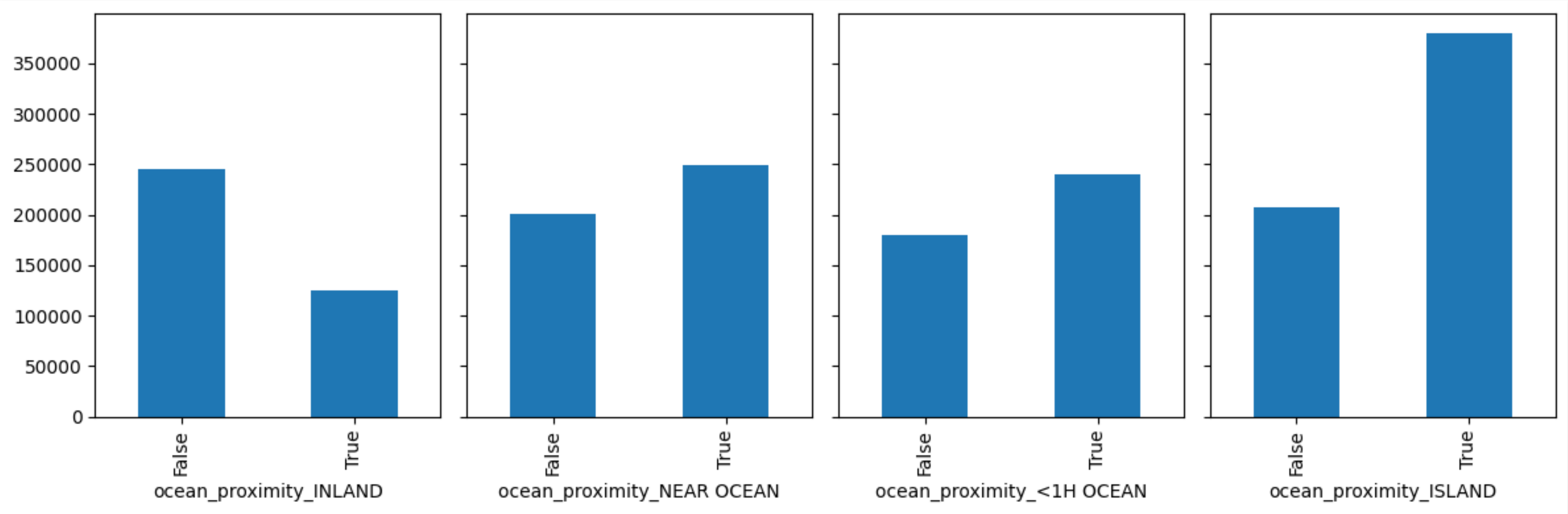
```python
feature_importance = forest_reg.feature_importances_
pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importance}).sort_values(by='Importance', ascending=False)
```

	Feature	Importance
7	median_income	0.481621
12	ocean_proximity_INLAND	0.137495
8	population_per_household	0.121431
0	longitude	0.057869
1	latitude	0.056177
2	housing_median_age	0.044215
9	rooms_per_household	0.025773
10	bedrooms_per_room	0.024335
3	total_rooms	0.012644
4	total_bedrooms	0.012243
5	population	0.011547
6	households	0.010305
15	ocean_proximity_NEAR OCEAN	0.002166
11	ocean_proximity_<1H OCEAN	0.001226
14	ocean_proximity_NEAR BAY	0.000817
13	ocean_proximity_ISLAND	0.000136

Calculate the average `median_house_income` by `ocean_proximity`

▼ Code

```
cur_df = preprocess(normalize=False)
fig, axs = plt.subplots(1, 4, figsize=(12, 4), sharex=True, sharey=True)
for i, c in enumerate(['INLAND', 'NEAR OCEAN', '<1H OCEAN', 'ISLAND']):
    cur_df.groupby(f'ocean_proximity_{c}')['median_house_value'].mean().plot(kind="bar", ax=axs[i])
fig.tight_layout()
```



Hyperparameters

Look at parameters used by our current forest

▼ Code

```
forest_reg.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 1.0,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```