27-Random_Forest_Regressor

October 20, 2024

1 Random Forest Regressor

A **Random Forest Regressor** is an ensemble learning method that combines the predictions of multiple decision trees to improve predictive performance for regression tasks. Unlike a single decision tree, which might overfit or have high variance, a random forest aggregates the outputs of many trees to make the final prediction, thus reducing overfitting and variance while improving generalization.

It works by constructing several decision trees during training and averaging their predictions. This process of averaging multiple trees helps to smooth out the predictions and gives more robust results.

1.0.1 Why is Random Forest Regressor Important?

- **Reduces Overfitting**: Individual decision trees tend to overfit, especially when the trees are deep. By averaging the predictions of many trees, Random Forest mitigates overfitting, making the model more generalizable to unseen data.
- **Handles Non-linear Relationships**: Decision trees naturally handle non-linear relationships. A random forest regressor benefits from this, enabling it to capture complex patterns in the data.
- **Robustness to Noise and Outliers**: Because Random Forest averages the predictions of many trees, it is less sensitive to outliers and noise than individual decision trees.
- **Feature Importance**: Random Forest provides feature importance scores, which can be useful in understanding which features have the most predictive power.

1.0.2 How does Random Forest Regressor work?

Random Forest Regressor is based on an ensemble of decision trees. Here's a step-by-step explanation of how it works:

1. Bagging (Bootstrap Aggregating): The first key idea behind Random Forest is bagging. Random subsets (bootstrapped samples) of the original training data are created. Each decision tree in the forest is trained on a different random sample of the data. This technique helps in reducing the variance and overfitting that can occur with individual trees.

2. Random Feature Selection**: During the construction of each tree, Random Forest also selects a random subset of features to consider at each split. This random selection of features ensures that the trees in the forest are more diverse, further reducing correlation between the trees.

3. Tree Construction: Each tree is built independently by splitting the data at different nodes based on feature values (as in any decision tree). The process is recursive, and the tree grows until a stopping criterion is met (e.g., a minimum number of samples per leaf or a maximum tree depth).

4. Prediction: After all the trees are trained, predictions are made by each tree, and the final prediction for regression is the average of the predictions from all the trees.

1.0.3 When should you use Random Forest Regressor?

- **Non-linear and complex datasets**: Random Forest works well when your dataset has complex, non-linear relationships between features and target values.
- **High-dimensional data**: It handles high-dimensional datasets with many features and can reduce the risk of overfitting by selecting random subsets of features.
- **When interpretability is not the primary goal**: Random Forest is less interpretable than simpler models like linear regression, so it's best used when predictive accuracy is more important than understanding the model's inner workings.
- **When there are many categorical and continuous variables**: It can handle mixed-type data without much preprocessing.

1.0.4 Who uses Random Forest Regressor?

- **Data Scientists**: Random Forest is widely used in industries like finance, healthcare, and marketing to solve regression problems where complex patterns need to be captured.
- **Business Analysts**: It is often used to predict continuous outcomes such as sales forecasts, customer retention rates, and other business metrics.
- **Researchers**: In research fields like genomics or environmental science, Random Forest is applied to make predictions based on numerous features, and it's useful for feature selection.

1.0.5 Key Points to Remember:

- **Ensemble of Decision Trees**: Random Forest is a collection of decision trees, each trained on a different bootstrap sample of the data.
- **Random Feature Selection**: At each split, a random subset of features is chosen, making the trees less correlated with one another.
- **Averaging Predictions**: For regression tasks, the final prediction is the average of the predictions made by all the trees.
- **Feature Importance**: Random Forest naturally provides feature importance scores, which can help identify the most relevant features.

1.0.6 Advantages of Random Forest Regressor:

- **Reduced Overfitting**: By averaging multiple trees, Random Forest smooths out the predictions and reduces overfitting compared to a single decision tree.
- **Handles Missing Data**: Random Forest can handle missing values relatively well by filling in missing data based on correlations.

- **Robust to Outliers**: Due to the averaging process, Random Forest is less sensitive to outliers in the data.
- **No need for feature scaling**: Unlike models like SVM or linear regression, Random Forest doesn't require feature scaling (e.g., normalization or standardization).

[1]:
```
import pandas as pd
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, root_mean_squared_error,␣
 ↪r2_score

healthexp = sns.load_dataset('healthexp')
healthexp
```

[1]:
```
     Year        Country  Spending_USD  Life_Expectancy
0    1970        Germany       252.311             70.6
1    1970         France       192.143             72.2
2    1970  Great Britain       123.993             71.9
3    1970          Japan       150.437             72.0
4    1970            USA       326.961             70.9
..    …            …             …               …
269  2020        Germany      6938.983             81.1
270  2020         France      5468.418             82.3
271  2020  Great Britain      5018.700             80.4
272  2020          Japan      4665.641             84.7
273  2020            USA     11859.179             77.0

[274 rows x 4 columns]
```

[2]:
```
healthexp = pd.get_dummies(healthexp)
healthexp
```

[2]:
```
     Year  Spending_USD  Life_Expectancy  Country_Canada  Country_France  \
0    1970       252.311             70.6           False           False
1    1970       192.143             72.2           False            True
2    1970       123.993             71.9           False           False
3    1970       150.437             72.0           False           False
4    1970       326.961             70.9           False           False
..    …           …                …              …               …
269  2020      6938.983             81.1           False           False
270  2020      5468.418             82.3           False            True
271  2020      5018.700             80.4           False           False
272  2020      4665.641             84.7           False           False
273  2020     11859.179             77.0           False           False
```

```
     Country_Germany  Country_Great Britain  Country_Japan  Country_USA
0               True                  False          False        False
1              False                  False          False        False
2              False                   True          False        False
3              False                  False           True        False
4              False                  False          False         True
..               …                    …             …           …
269             True                  False          False        False
270            False                  False          False        False
271            False                   True          False        False
272            False                  False           True        False
273            False                  False          False         True

[274 rows x 9 columns]
```

[3]:
```
X = healthexp.drop(['Life_Expectancy'], axis=1)
y = healthexp['Life_Expectancy']

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
 ↪random_state=19)

rfr = RandomForestRegressor(random_state=13)
rfr.fit(X_train, y_train)
```

[3]:
```
RandomForestRegressor(random_state=13)
```

[4]:
```
y_pred = rfr.predict(X_test)
y_pred
```

[4]:
```
array([79.705, 81.089, 74.833, 79.357, 71.152, 79.186, 82.417, 81.7  ,
       78.368, 71.48 , 76.446, 75.79 , 82.694, 78.601, 76.848, 77.614,
       71.838, 78.91 , 72.414, 82.033, 71.816, 78.415, 79.937, 78.513,
       78.265, 72.639, 75.759, 81.337, 81.408, 82.269, 74.003, 74.395,
       78.677, 74.98 , 71.777, 74.347, 74.581, 76.581, 81.142, 80.895,
       74.979, 81.68 , 77.489, 78.219, 80.888, 83.572, 78.392, 73.976,
       77.236, 80.114, 74.728, 76.782, 78.031, 78.778, 77.519])
```

[5]:
```
mean_absolute_error(y_test, y_pred)
```

[5]:
```
0.25916363636361917
```

[6]:
```
root_mean_squared_error(y_test, y_pred)
```

[6]:
```
0.31970520512155615
```

[7]:
```
r2_score(y_test, y_pred)
```

[7]:
```
0.9910457602615238
```

[8]:
```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10,20,30],
    'min_samples_split': [2,5,10],
    'min_samples_leaf': [1,2,4]
}

rfr_cv = GridSearchCV(rfr, param_grid, cv=3, n_jobs=-1)
rfr_cv.fit(X_train, y_train)
```

[8]:
```
GridSearchCV(cv=3, estimator=RandomForestRegressor(random_state=13), n_jobs=-1,
             param_grid={'max_depth': [10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [100, 200, 300]})
```

[9]:
```
y_pred_cv = rfr_cv.predict(X_test)
y_pred_cv
```

[9]:
```
array([79.71933333, 81.09833333, 74.83733333, 79.40433333, 71.159     ,
       79.232     , 82.41166667, 81.686     , 78.43533333, 71.404     ,
       76.47266667, 75.904     , 82.69633333, 78.64833333, 76.843     ,
       77.58633333, 71.715     , 78.93433333, 72.42533333, 82.016     ,
       71.731     , 78.45666667, 80.        , 78.529     , 78.31966667,
       72.51466667, 75.78433333, 81.29033333, 81.392     , 82.281     ,
       73.972     , 74.404     , 78.67233333, 74.99466667, 71.68766667,
       74.34333333, 74.56433333, 76.604     , 81.14366667, 80.91      ,
       74.97633333, 81.66533333, 77.59366667, 78.17166667, 80.87033333,
       83.52633333, 78.43366667, 73.93433333, 77.149     , 80.104     ,
       74.718     , 76.741     , 78.136     , 78.70566667, 77.53133333])
```

[10]:
```
mean_absolute_error(y_test, y_pred_cv)
```

[10]:
```
0.25089696969702713
```

[11]:
```
root_mean_squared_error(y_test, y_pred_cv)
```

[11]:
```
0.31042865768991584
```

[12]:
```
r2_score(y_test, y_pred_cv)
```

[12]:
```
0.9915578528520343
```