Day49_Random_Forest_Regressor

July 18, 2025

Today, we are learning about the **Random Forest Regression model** — an ensemble method that combines multiple Decision Trees to make better predictions. It is known for its accuracy, robustness, and ability to handle both regression and classification problems.

In this notebook, we'll explore:

- What is Random Forest?
- How it works
- Key features and use cases
- Python implementation with visualization

What is Random Forest?

Random Forest is an **ensemble learning method** primarily used for classification and regression tasks. It builds multiple decision trees and merges them together to get a more accurate and stable prediction.

How it works:

- Builds multiple decision trees (hence "forest").
- Uses **bagging** (Bootstrap Aggregating): random sampling with replacement.
- Final prediction is based on majority vote (classification) or average (regression).

Key Features:

- Reduces overfitting (compared to Decision Trees).
- Works well with both categorical and numerical data.
- Handles missing values.
- More accurate than a single Decision Tree.

Random Forest Use Cases:

- **Medical Diagnosis** (disease prediction)
- **Banking** (loan default prediction)
- **Marketing** (customer churn, recommendation systems)
- **Finance** (stock market prediction)
- **E-commerce** (product classification)

1 Importing Required Libraries

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Load Dataset

[3]:
```
dataset = pd.read_csv(r"C:\Users\Lenovo\Downloads\emp_sal.csv")
dataset
```

[3]:

	Position	Level	Salary
0	Jr Software Engineer	1	45000
1	Sr Software Engineer	2	50000
2	Team Lead	3	60000
3	Manager	4	80000
4	Sr manager	5	110000
5	Region Manager	6	150000
6	AVP	7	200000
7	VP	8	300000
8	CTO	9	500000
9	CEO	10	1000000

2 Prepare Features and Target

[4]:
```
X = dataset.iloc[:, 1:2].values  # Level column
y = dataset.iloc[:, 2].values   # Salary column
```

3 Train Random Forest Regressor

[5]:
```
from sklearn.ensemble import RandomForestRegressor

# Create model instance
rf_reg = RandomForestRegressor()
rf_reg.fit(X, y)
```

[5]:
```
RandomForestRegressor()
```

4 Predicting Salaries

[6]:
```
rf_pred = rf_reg.predict([[6]])
print(rf_pred)
```

```
[139000.]
```

> **Note**: This prediction will change every time model run because Random Forest uses random sampling (bagging).

[8]:
```
rf_reg = RandomForestRegressor()
rf_reg.fit(X, y)
rf_pred = rf_reg.predict([[6]])
print(rf_pred)
```

```
[145800.]
```

[10]:
```
rf_reg = RandomForestRegressor()
rf_reg.fit(X, y)
rf_pred = rf_reg.predict([[6]])
print(rf_pred)
```

```
[143400.]
```

5 Make Predictions Consistent using random_state

[11]:
```
rf_reg = RandomForestRegressor(random_state=0)
rf_reg.fit(X, y)
rf_pred = rf_reg.predict([[6]])
print(rf_pred)
```

```
[142600.]
```

[12]:
```
# Run 2nd time with same output
rf_reg = RandomForestRegressor(random_state=0)
rf_reg.fit(X, y)
rf_pred = rf_reg.predict([[6]])
print(rf_pred)
```

```
[142600.]
```

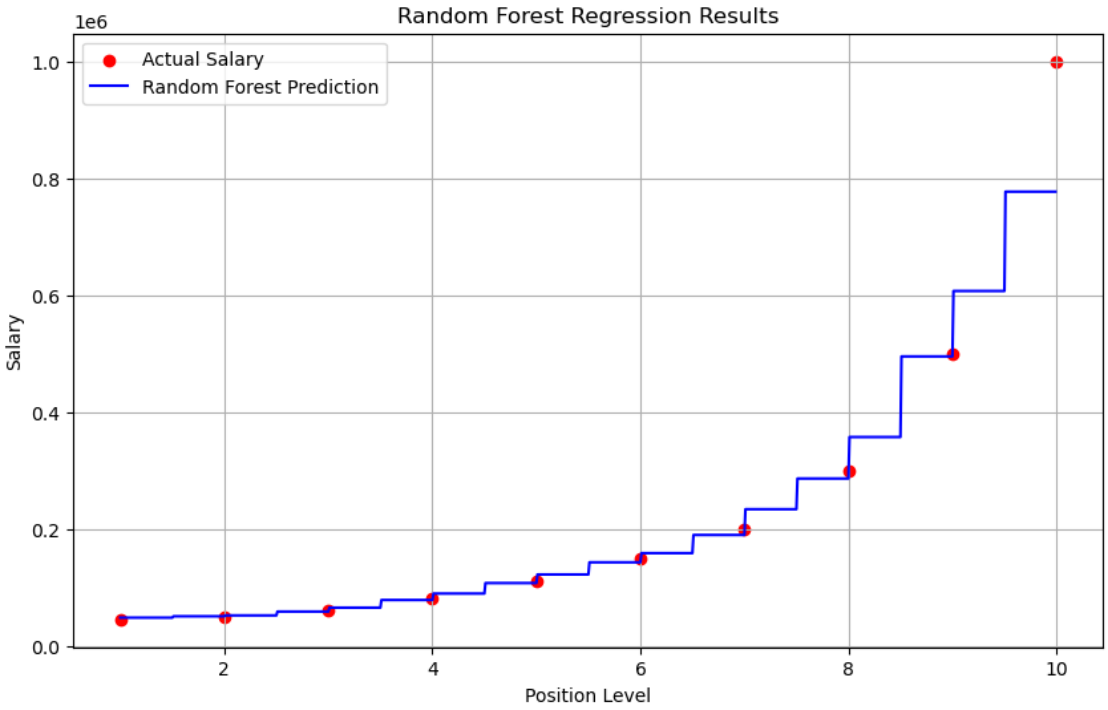
Visualization

[13]:
```
X_grid = np.arange(min(X), max(X), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))

plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='red', label='Actual Salary')
plt.plot(X_grid, rf_reg.predict(X_grid), color='blue', label='Random Forest␣
 ↪Prediction')
plt.title('Random Forest Regression Results')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_13680\2586575323.py:1:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is
```

```
deprecated, and will error in future. Ensure you extract a single element from
your array before performing this operation. (Deprecated NumPy 1.25.)
  X_grid = np.arange(min(X), max(X), 0.01)
```



6 Summary

- We learned what a **Random Forest** is and how it improves over single Decision Trees.
- Implemented the model using `sklearn.ensemble.RandomForestRegressor`.
- Visualized the results to understand its performance.

Random Forest is a powerful and accurate model that performs well even with noisy and complex datasets.