IP_5th_Random_Forest_ensemble_methods

December 30, 2023

0.0.1 How To Learn Machine Learning Algorithms For Interviews

Random Forest Classifier And Regresor

11. Ensemble Techniques(Boosting And Bagging)
12. Working of Random Forest Classifier
13. Working of Random Forest Regresor
14. Hyperparameter Tuning(Grid Search And RandomSearch)

Xgboost Classifier And Regressor, GB Algorithm, Adaboost Decision Tree Theoretical Understanding:

1. Tutorial 37:Entropy In Decision Tree https://www.youtube.com/watch?v=1IQOtJ4NI_0
2. Tutorial 38:Information Gain https://www.youtube.com/watch?v=FuTRucXB9rA
3. Tutorial 39:Gini Impurity https://www.youtube.com/watch?v=5aIFgrrTqOw
4. Tutorial 40: Decision Tree For Numerical Features: https://www.youtube.com/watch?v=5O8HvA9pMew
5. How To Visualize DT: https://www.youtube.com/watch?v=ot75kOmpYjI

Theoretical Understanding:

1. Ensemble technique(Bagging): https://www.youtube.com/watch?v=KIOeZ5cFZ50
2. Adaboost(Boosting Technique):https://www.youtube.com/watch?v=NLRO1-jp5F8
3. Gradient Boosting In Depth Intuition Part 1: https://www.youtube.com/watch?v=Nol1hVtLOSg
4. Gradient Boosting In Depth Intuition Part 2: https://www.youtube.com/watch?v=Oo9q6YtGzvc
5. Xgboost Classifier Indepth Intuition: https://www.youtube.com/watch?v=gPciUPwWJQQ
6. Xgboost Regression Indpeth Intuition: https://www.youtube.com/watch?v=w-_vmVfpssg
7. Implementation of Xgboost: https://youtu.be/9HomdnM12o4

1 Random Forest

Important properties of Random Forest Classifiers

1. Decision Tree—Low Bias And High Variance
2. Ensemble Bagging(Random Forest Classifier)–Low Bias And Low Variance

[ ]:
```

```

1. What Are the Basic Assumption? There are no such assumptions

```
[ ]:
```

2. Advantages Advantages of Random Forest

1. Doesn't Overfit
2. Favourite algorithm for Kaggle competition
3. Less Parameter Tuning required
4. Decision Tree can handle both continuous and categorical variables.
5. No feature scaling required: No feature scaling (standardization and normalization) required in case of Random Forest as it uses DEcision Tree internally
6. Suitable for any kind of ML problems

```
[ ]:
```

3. Disadvantages Disadvantages of Random Forest

1.Biased With features having many categories

2. Biased in multiclass classification problems towards more frequent classes.

```
[ ]:
```

4. Whether Feature Scaling is required? No

6. Impact of outliers? Robust to Outliers

```
[ ]:
```

4. Whether Feature Scaling is required? No

6. Impact of outliers? Robust to Outliers

```
[ ]:
```

Types of Problems it can solve(Supervised)

1. Classification
2. Regression

```
[ ]:
```

2 Practical implementation:

https://www.geeksforgeeks.org/random-forest-regression-in-python/

[119]:
```python
import pandas as pd
import sklearn
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

[120]:
```python
df=pd.read_csv("dataset/Employee_Salary_Dataset.csv")
```

[121]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35 entries, 0 to 34
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                35 non-null     int64
 1   Experience_Years  35 non-null     int64
 2   Age               35 non-null     int64
 3   Gender            35 non-null     object
 4   Salary            35 non-null     int64
dtypes: int64(4), object(1)
memory usage: 1.5+ KB
```

[122]:
```python
df.shape
```

[122]: (35, 5)

[123]:
```python
df
```

[123]:

	ID	Experience_Years	Age	Gender	Salary
0	1	5	28	Female	250000
1	2	1	21	Male	50000
2	3	3	23	Female	170000
3	4	2	22	Male	25000
4	5	1	17	Male	10000
5	6	25	62	Male	5001000
6	7	19	54	Female	800000
7	8	2	21	Female	9000
8	9	10	36	Female	61500
9	10	15	54	Female	650000
10	11	4	26	Female	250000
11	12	6	29	Male	1400000
12	13	14	39	Male	6000050
13	14	11	40	Male	220100
14	15	2	23	Male	7500
15	16	4	27	Female	87000

```
16  17                  10   34  Female    930000
17  18                  15   54  Female   7900000
18  19                   2   21    Male     15000
19  20                  10   36    Male    330000
20  21                  15   54    Male   6570000
21  22                   4   26    Male     25000
22  23                   5   29    Male   6845000
23  24                   1   21  Female      6000
24  25                   4   23  Female      8900
25  26                   3   22  Female     20000
26  27                   1   18    Male      3000
27  28                  27   62  Female  10000000
28  29                  19   54  Female   5000000
29  30                   2   21  Female      6100
30  31                  10   34    Male     80000
31  32                  15   54    Male    900000
32  33                  20   55  Female   1540000
33  34                  19   53  Female   9300000
34  35                  16   49    Male   7600000
```

[ ]:

[124]:
```
l=LabelEncoder()
```

[125]:
```
df["Gender"]=l.fit_transform(df["Gender"])
```

[126]:
```
x=df.iloc[:,0:4].values
x
```

[126]:
```
array([[ 1,  5, 28,  0],
       [ 2,  1, 21,  1],
       [ 3,  3, 23,  0],
       [ 4,  2, 22,  1],
       [ 5,  1, 17,  1],
       [ 6, 25, 62,  1],
       [ 7, 19, 54,  0],
       [ 8,  2, 21,  0],
       [ 9, 10, 36,  0],
       [10, 15, 54,  0],
       [11,  4, 26,  0],
       [12,  6, 29,  1],
       [13, 14, 39,  1],
       [14, 11, 40,  1],
       [15,  2, 23,  1],
       [16,  4, 27,  0],
       [17, 10, 34,  0],
       [18, 15, 54,  0],
```

```
       [19,  2, 21,  1],
       [20, 10, 36,  1],
       [21, 15, 54,  1],
       [22,  4, 26,  1],
       [23,  5, 29,  1],
       [24,  1, 21,  0],
       [25,  4, 23,  0],
       [26,  3, 22,  0],
       [27,  1, 18,  1],
       [28, 27, 62,  0],
       [29, 19, 54,  0],
       [30,  2, 21,  0],
       [31, 10, 34,  1],
       [32, 15, 54,  1],
       [33, 20, 55,  0],
       [34, 19, 53,  0],
       [35, 16, 49,  1]], dtype=int64)
```

[127]:
```
y=df.iloc[:,-1].values
y
```

[127]:
```
array([  250000,    50000,   170000,    25000,    10000,  5001000,
         800000,     9000,    61500,   650000,   250000,  1400000,
        6000050,   220100,     7500,    87000,   930000,  7900000,
          15000,   330000,  6570000,    25000,  6845000,     6000,
           8900,    20000,     3000, 10000000,  5000000,     6100,
          80000,   900000,  1540000,  9300000,  7600000], dtype=int64)
```

[128]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)
```

[129]:
```
y_train
```

[129]:
```
array([ 7600000,  6845000,   900000,  9300000,  1540000,   250000,
          25000,   800000,   650000, 10000000,  5001000,   220100,
        1400000,    15000,     9000,   250000,  5000000,    87000,
           3000,  6000050,    80000,  7900000,    10000,   930000,
           8900,  6570000], dtype=int64)
```

[130]:
```
s=StandardScaler()
s.fit(x_train)
s.transform(x_train)
s.transform(x_test)
```

[130]:
```
array([[ 0.34904982, -0.91792842, -0.87060341,  1.        ],
       [-0.33403693, -1.18019369, -1.07210806,  1.        ],
       [-1.6026266 , -1.31132632, -1.20644449,  1.        ],
       [ 1.12972039, -1.18019369, -1.20644449, -1.        ],
```

```
[ 0.7393851 , -1.04906106, -1.13927628, -1.        ],
[ 0.54421746, -1.31132632, -1.20644449, -1.        ],
[-1.50504278, -1.04906106, -1.07210806, -1.        ],
[-0.91953985, -0.13113263, -0.19892126, -1.        ],
[ 0.15388218, -0.13113263, -0.19892126,  1.        ]])
```

[ ]:

[ ]:

[131]:
```
re=RandomForestRegressor(oob_score=True)
```

[132]:
```
re.fit(x_train,y_train)
```

[132]:
```
RandomForestRegressor(oob_score=True)
```

[133]:
```
re.oob_score_
```

[133]:
```
0.09668858029680272
```

[134]:
```
from sklearn.metrics import mean_squared_error
```

[135]:
```
y_pred=re.predict(x_test)
```

[136]:
```
mean_squared_error(y_test,y_pred)
```

[136]:
```
976521079719.1389
```

[137]:
```
re.score(x_test,y_test)
```

[137]:
```
-92.0546149055722
```

[138]:
```
#overfitted model
```

[ ]:

3 Ada boost

[139]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
```

[140]:
```
dt=DecisionTreeRegressor(max_depth=1)
```

[141]:
```
ada=AdaBoostRegressor(dt, n_estimators=50, random_state=42)
```

[142]:
```
ada.fit(x_train,y_train)
```

[142]:
```
AdaBoostRegressor(estimator=DecisionTreeRegressor(max_depth=1), random_state=42)
```

[143]:
```python
# Make predictions on the test set
predictions = ada.predict(x_test)

# Evaluate Mean Squared Error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 1792370368390.9731
```

[144]:
```python
ada.score(x_train,y_train)
```

[144]:
```
0.5001169877588378
```

[145]:
```python
ada.score(x_test,y_test)
```

[145]:
```
-169.79849873466245
```

[ ]:
```python

```

[ ]:
```python

```

[146]:
```python
x_train
```

[146]:
```
array([[35, 16, 49,  1],
       [23,  5, 29,  1],
       [32, 15, 54,  1],
       [34, 19, 53,  0],
       [33, 20, 55,  0],
       [ 1,  5, 28,  0],
       [ 4,  2, 22,  1],
       [ 7, 19, 54,  0],
       [10, 15, 54,  0],
       [28, 27, 62,  0],
       [ 6, 25, 62,  1],
       [14, 11, 40,  1],
       [12,  6, 29,  1],
       [19,  2, 21,  1],
       [ 8,  2, 21,  0],
       [11,  4, 26,  0],
       [29, 19, 54,  0],
       [16,  4, 27,  0],
       [27,  1, 18,  1],
       [13, 14, 39,  1],
       [31, 10, 34,  1],
       [18, 15, 54,  0],
       [ 5,  1, 17,  1],
       [17, 10, 34,  0],
       [25,  4, 23,  0],
```

```
        [21, 15, 54,  1]], dtype=int64)
```

4 GBoost

[147]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

[148]:
```
s=StandardScaler()
s.fit(x_train)
x_train=s.transform(x_train)
x_test=s.transform(x_test)
```

[ ]:
```
```

[ ]:
```
```

[149]:
```
# Create Gradient Boosting regressor
gb_regressor = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
 ↪max_depth=1, random_state=42)

# Train the Gradient Boosting regressor
gb_regressor.fit(x_train, y_train)

# Make predictions on the test set
predictions = gb_regressor.predict(x_test)

# Evaluate Mean Squared Error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 2120741249121.087
```

[150]:
```
gb_regressor.score(x_train,y_train)
```

[150]:
```
0.7924476779635051
```

[151]:
```
gb_regressor.score(x_test,y_test)
```

[151]:
```
-201.08960600019404
```

5 i have to do feature engineering,hyper parameter tunning

[ ]:
```
```

6 XG Boost

[152]:
```
import xgboost
```

[153]:
```
from xgboost import XGBRegressor
```

[156]:
```
# Create XGBoost regressor
xgb_regressor = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3,␣
↪random_state=42)

# Train the XGBoost regressor
xgb_regressor.fit(x_train, y_train)

# Make predictions on the test set
predictions = xgb_regressor.predict(x_test)

# Evaluate Mean Squared Error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 976058570083.7365
```

[158]:
```
xgb_regressor.score(x_train,y_train)
```

[158]: 0.99743131653229

[159]:
```
xgb_regressor.score(x_test,y_test)
```

[159]: -92.01054145246775

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: