randomForest

June 30, 2021

[1]:
```
# Make needed imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import graphviz
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error
```

[2]:
```
# Load train and testing data
training_data = pd.read_pickle("train.pkl")
testing_data = pd.read_pickle("test.pkl")
```

[3]:
```
# Seperate training X and Y values
train_X = training_data.iloc[:, 2:10].values
train_Y = training_data.iloc[:, 1].values
```

[4]:
```
regressor = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 150,␣
 ↪bootstrap = True)

# Fit the regressor with x and y data
regressor.fit(train_X, train_Y)

# Load testing values
test_X = testing_data.iloc[:, 2:10].values
test_Y = testing_data.iloc[:, 1].values

# Predict with testing X
predict_Y = regressor.predict(test_X)

# Compute loss
loss = []
for i in range(len(test_Y)):
    dif = (abs(float(test_Y[i] - predict_Y[i])) / float(test_Y[i])) * 100 # we␣
 ↪want to calculate the percentage.
    loss.append(dif)
```

```
def total(input):
    res = 0
    for value in input:
        res += value
    return value
```

[5]:
```
print("Loss percentage on testing set: " + str(total(loss) / len(loss)))
print("MSE loss on testing set: " + str(mean_squared_error(test_Y, predict_Y)))
```

```
Loss percentage on testing set: 0.03350179242734122
MSE loss on testing set: 0.19376031604155886
```

[6]:
```
features = ["Log GDP per capita","Social support","Healthy life expectancy at␣
 ↪birth","Freedom to make life choices","Generosity","Perceptions of␣
 ↪corruption","Positive affect","Negative affect"]

# Create tree dot data

decision_tree = (tree.export_graphviz(regressor[0], out_file=None,␣
 ↪feature_names = features,
                                      filled=True))
```

[7]:
```
# Render a single decision tree
graph = graphviz.Source(decision_tree, format="png")
graph.format = 'svg'
graph.render("decision_tree")
```

[7]:
```
'decision_tree.svg'
```

[8]:
```
world_happiness = pd.read_csv ('../data/world-happiness-report.csv')
world_average = world_happiness.mean()[2:]
print(world_average)
print("Predicted world happiness: " + str(regressor.
 ↪predict([world_average])[0]))
print("Actual world happiness: " + str(world_happiness.get("Life Ladder").
 ↪mean()))
```

```
Log GDP per capita                  9.368453
Social support                      0.812552
Healthy life expectancy at birth    63.359374
Freedom to make life choices        0.742558
Generosity                          0.000103
Perceptions of corruption           0.747125
Positive affect                     0.710003
Negative affect                     0.268544
dtype: float64
Predicted world happiness: 5.250852308009663
Actual world happiness: 5.46670548999487
```

[9]:
```
# Improvement via 1%
max_value = 0
index = 0
for i in range(len(world_average)):
    new_world_average = world_average.copy()
    new_world_average[i] *= 1.01
    new_happiness = regressor.predict([new_world_average])[0]
    if new_happiness > max_value:
        max_value = new_happiness
        index = i
print("Best way to improve the world by 1%: " + world_happiness.
 ↪columns[3+index])
```

```
Best way to improve the world by 1%: Social support
```

[10]:
```
# Improvement via 10%
max_value = 0
index = 0
for i in range(len(world_average)):
    new_world_average = world_average.copy()
    new_world_average[i] *= 1.1
    new_happiness = regressor.predict([new_world_average])[0]
    if new_happiness > max_value:
        max_value = new_happiness
        index = i
print("Best way to improve the world by 10%: " + world_happiness.
 ↪columns[3+index])
```

```
Best way to improve the world by 10%: Log GDP per capita
```

[11]:
```
# Improvement via 50%
max_value = 0
index = 0
for i in range(len(world_average)):
    new_world_average = world_average.copy()
    new_world_average[i] *= 1.5
    new_happiness = regressor.predict([new_world_average])[0]
    if new_happiness > max_value:
        max_value = new_happiness
        index = i
print("Best way to improve the world by 50%: " + world_happiness.
 ↪columns[3+index])
```

```
Best way to improve the world by 50%: Log GDP per capita
```

[12]:
```
# Improvement via 100%
max_value = 0
index = 0
for i in range(len(world_average)):
```

```
    new_world_average = world_average.copy()
    new_world_average[i] *= 2
    new_happiness = regressor.predict([new_world_average])[0]
    if new_happiness > max_value:
        max_value = new_happiness
        index = i
print("Best way to improve the world by 100%: " + world_happiness.
 ↪columns[3+index])
```

```
Best way to improve the world by 100%: Log GDP per capita
```

[13]:
```
target_happiness = 5.3
feature_improvement_need = []
improvement = 1
prev_happiness = 0
iters = 0

for i in range(len(world_average) - 2):
    new_world_average = world_average.copy()
    new_happiness = regressor.predict([new_world_average])[0]
    prev_happiness = new_happiness
    while (new_happiness < target_happiness and iters < 100):
        new_world_average[i] *= improvement
        # Scale up 1% each time
        improvement += 0.01
        new_happiness = regressor.predict([new_world_average])[0]
        new_world_average = world_average.copy()
        prev_happiness = new_happiness
        iters += 1
    feature_improvement_need.append(improvement)
    improvement = 1
    iters = 0
for i in range(len(feature_improvement_need) - 2):
    print("We need to increase " + world_happiness.columns[3+i] + " by " +␣
 ↪str(int((feature_improvement_need[i]-1)*100)) + "%")
```

```
We need to increase Log GDP per capita by 6%
We need to increase Social support by 8%
We need to increase Healthy life expectancy at birth by 9%
We need to increase Freedom to make life choices by 100%
```