Using random forest regression to determine what factors have influence on life expectancy

Team member: **Jiahua Zhang , Bin Xu, Di Kang, Yiwei Zhang**

Abstract

By using the Life Expectancy dataset provided by Kaggle, we wanted to see what the world needed to improve on the Life Expectancy. We used a random forest regressor to determine the effects of scaling certain features and their correlation to Life Expectancy.

1 Introduction

The dataset contains different countries with different life expectancy. Some variables have a very weak relationship with life expectancy or even cannot affect life expectancy. The life expectancy Report is a survey of the state of global life expectancy, reporting these 18 factors: Country, Year, Status, Life expectancy, Adult Mortality, infant deaths, Alcohol, percentage expenditure, Hepatitis B, Measles, BMI, under-five deaths, Polio, Total expenditure, Diphtheria, HIV/AIDS, GDP, Population, thinness 1-19 years, thinness 5-9 years, Income composition of resources, Schooling. In the project, we need to discover which factor has more influence on life expectancy.

2 Dataset preparation

2.1 Data overview

There are currently 150 countries and regions included within. The first element records the name of the country, the second element indicates the year of the data. In our project, we will drop the status column from the data. Our Y value is "Life Expectancy".

```
life_expectancy = pd.read_csv ('../data/Life_Expectancy_Data.csv')
life_expectancy = life_expectancy.drop('Status', axis=1)
life_expectancy = life_expectancy.drop('Year', axis=1)
life_expectancy = life_expectancy.drop('Country', axis=1)
life_expectancy_shape = life_expectancy.shape
print(life_expectancy)
```

```
      Life expectancy   Adult Mortality  infant deaths  Alcohol  \
0                65.0              263             62     0.01
1                59.9              271             64     0.01
2                59.9              268             66     0.01
3                59.5              272             69     0.01
4                59.2              275             71     0.01
...               ...              ...            ...      ...
1644             44.3              723             27     4.36
1645             44.5              715             26     4.06
1646             44.8               73             25     4.43
1647             45.3              686             25     1.72
1648             46.0              665             24     1.68
```

figure 1

2.2 Data processing

We extract 80% of data as our training data and the remaining 20% as our testing data.

```
life_expectancy['split'] = np.random.randn(life_expectancy.shape[0], 1)
msk = np.random.rand(len(life_expectancy)) <= 0.8
train = life_expectancy[msk]
test = life_expectancy[~msk]
print(train.shape)
print(test.shape)
```

```
(1310, 20)
(339, 20)
```

figure 2

Next, we using minmaxscaler to normalize the testing data and training data, then we can convert the data to a pickle file.

```
x = train.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
normalized_training_data = pd.DataFrame(x_scaled)
print(normalized_training_data)
```

	0	1	2	3	4	5	6
0	0.466667	0.362881	0.038750	0.000000	0.003759	0.649485	0.008780
1	0.353333	0.369806	0.041250	0.000000	0.003861	0.639175	0.003271
2	0.337778	0.379501	0.044375	0.000000	0.000374	0.680412	0.022923
3	0.328889	0.385042	0.046250	0.000000	0.004202	0.659794	0.015132
4	0.324444	0.387812	0.048125	0.000000	0.002994	0.628866	0.021766
...	...	...	...	...	...	...	...
1324	0.006667	1.000000	0.016875	0.243561	0.000000	0.680412	0.000236
1325	0.011111	0.988920	0.016250	0.226764	0.000000	0.051546	0.007593
1326	0.017778	0.099723	0.015625	0.247480	0.000000	0.731959	0.002313
1327	0.028889	0.948753	0.015625	0.095745	0.000000	0.762887	0.004025
1328	0.044444	0.919668	0.015000	0.093505	0.000000	0.793814	0.011283

figure 3

3 Experimentation

3.1 Overview

We plan to find the optimal value from the decision tree. Using a random forest regressor to decide which factor will have a bigger influence on life expectancy.

3.2 Algorithm decision

1. According to the characteristics of our data set: there are many attributes. This means that our regression is high-dimensional, but if the data is not preprocessed to reduce the dimension, some models can not fit the results well, but the regression tree model used by random forest has a good effect on high-dimensional data processing.
2. The data structure used in the regression tree model helps calculate the importance of each feature, which is what we want to know.
3. The regression model has good anti-noise performance.

3.3 Algorithm background

Random Forest

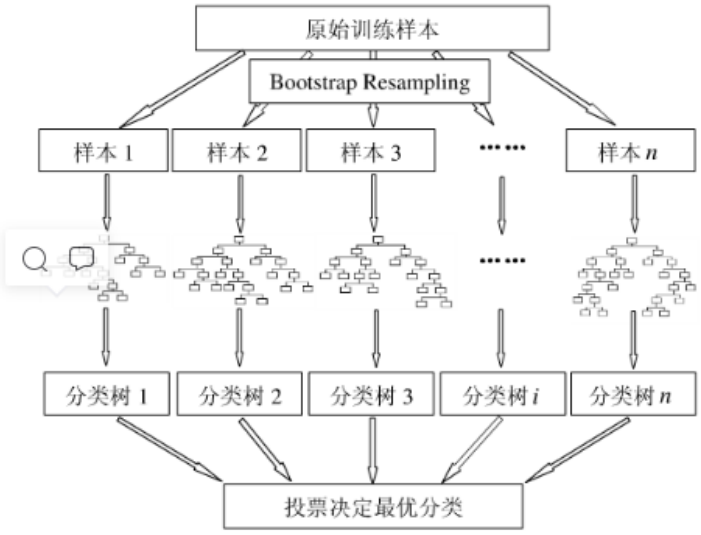

figure 4

1. N sample subsets are randomly selected from the dataset.
2. Build a decision tree based on this N sample subset.
3. Select the number of trees you want in the algorithm, and repeat steps 1 and 2.
4. The final value is calculated by taking the average of the predicted values of all decision trees in the forest.

Regression Tree

Regression tree is actually a variant of the decision tree.
The difference between these two are:

1. In the classification problem, the Regression tree uses Gini index as the basis for feature selection and split. In regression problems, the Regression tree uses MSE (Mean Square Error) or MAE (Mean Absolute error) as criteria for selecting features and splits.

2. In the classification problem, each leaf of the Regression tree represents a class; In regression problems, each leaf of the Regression tree represents a predicted value, which is continuous.

3.4 Approach

Because the random forest regression model does not need to do too much pre-processing of data, and theoretically, tree model does not need normalization, because they do not care about the value of variables, but the distribution of variables and conditional probability between variables, we didn't normalize the data at first.
But when we calculate accuracy, we found that when we normalize the data, the accuracy will be improved. It is interesting found maybe can be extension topic to discuss.
Second, we optimized the interface parameters of random forest, which are n-Estimators and Max-Leaf-Nodes, in order to resist overfitting.

4 Results

4.1 Set Baseline

Before we can make and evaluate predictions, we need to set up the baseline. In this way, we can measure if our model can beat after training.

```
#basline estimate
#getting life_expectancy from the oroginal testing data
baseline_preds = testing_data.iloc[:, 1:2].values
# print(baseline_preds)
# Baseline errors, and display average baseline error
base_errors = abs(baseline_preds - test_Y)
print('Average baseline error: ', round(np.mean(base_errors), 2))

 Average baseline error:  0.19
```

figure 5

4.2 Train model

Using scikit learn, instantiate the model. We import the random first regression model, and use training data.

```
#Training Model using RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 300, bootstrap = True
# Fit the regressor with x and y data
regressor.fit(train_X, train_Y)
# Predict with testing X
predict_Y = regressor.predict(test_X)
print(predict_Y)
```

figure 6

4.3 make prediction on the test set and determine the performance

We have got the relationships between the feature and target. Then we need to figure out if our result is good enough. In order to make predictions, we need to use the absolute error. We compare the predictions to the known answer. We want to know how far the prediction value is from the actual value.

```
# making prediction on the test set, we will get the prediction life expectancy
predict_Y = regressor.predict(test_X)
# Calculate the absolute errors
errors = abs(predict_Y - test_Y)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.08 degrees.
```

```
#Determine Performance Metrics
# Calculate mean absolute percentage error
percentage_error = 100 * (errors / predict_Y)
# Calculate and display accuracy
accuracy = 100 - np.mean(percentage_error)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 60.97 %.
```

figure 7

From the above figure, we get the 0.08 degree, which is much better than the baseline error. Also, we get 60.97% accuracy. We will try to improve this accuracy later.
(First time we use another data, the accuracy is over 95%, but after rearrange the date, we get 60% accuracy, we need to do research on it in the future work)

4.4 variable importance

In order to quantify the usefulness of all the variables from the testing data, we want to find the relative importances of variables. The code we use the zip, osring and argument unpacking.

```
#figuring out the usefulness of all the variable in the entire random forest
# Get numerical feature importances
testing_data_without_year = testing_data.drop(testing_data.columns[[0]],axis = 1)
# print(testing_data_without_year)
importances = list(regressor.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(testing_data_without_year, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable:                   11 Importance: 0.47
Variable:                   16 Importance: 0.1
Variable:                    2 Importance: 0.04
Variable:                    9 Importance: 0.04
Variable:                   15 Importance: 0.04
Variable:                    6 Importance: 0.03
Variable:                   12 Importance: 0.03
Variable:                   13 Importance: 0.03
Variable:                   14 Importance: 0.03
Variable:                   17 Importance: 0.03
Variable:                   18 Importance: 0.03
```

figure 8

We can get the index 17, 12 , 1… are most relative variables in our testing data. Then we calculate importance twice, we pick the most important column from testing data. We calculate the importance again.

```
#picking up the most important variables to recalculate the importance
updated_testing = testing_data.iloc[:, [2,6,9,11,12,13,14,15,13,16,17]]
test_important = testing_data.iloc[:, [2,6,9,11,12,13,14,15,13,16,17]].values
train_important = training_data.iloc[:, [2,6,9,11,12,13,14,15,13,16,17]].values

# Get numerical feature importances
updated_importances = list(regressor.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(updated_testing, upd
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable:                   17 Importance: 0.47
Variable:                    6 Importance: 0.04
Variable:                   13 Importance: 0.04
Variable:                   13 Importance: 0.03
Variable:                    9 Importance: 0.02
Variable:                   11 Importance: 0.02
Variable:                   12 Importance: 0.02
Variable:                   14 Importance: 0.02
Variable:                   16 Importance: 0.02
```

figure 9

After we get the updated importance, we can figure which index will be more important.
We also test the mean absolute and accuracy again. We hope after training the most relative variable, the error and accuracy can further improve.

```
# New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 300, bootstrap = True)
# Train the random forest
rf_most_important.fit(train_important, train_Y)
# Make predictions and determine the error
predictions = rf_most_important.predict(test_important)
errors = abs(predictions - test_Y)
# Display the performance metrics
print('Mean Absolute Error:', np.mean(errors), 'degrees.')
updated_error_percentage = np.mean(100 * (errors / test_Y))
accuracy = 100 - updated_error_percentage
print('Accuracy:', accuracy, '%.')
```

```
Mean Absolute Error: 0.0776993698826252 degrees.
Accuracy: -inf %.
```

figure 10

From the above data we get, the error is improved. The accuracy is infinitely closed to the 100%

5. Math Background

We need to measure the performance of the model, In order to calculate the loss, using below formula:

$$\sum_{n=1}^{n} \frac{abs(predict_n - truth_n)}{truth_n}$$

We get the predicted value from regress.predict(), then calculate the percentage to the loss. Adding up all dimension test data through the formula above will help me to get a loss percentage later. Another method we used to test for errors is to calculate the mean square error of the random forest regressor model. For calculate the MSE, we use following formula:

$$\mathrm{NRMSE} = \frac{\sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\hat{c}_n - c_n\right)^2}}{\sigma\left(c\right)},$$

6 Additional works

In future work, we can select discrete feature variables and continuous feature variables to design a composite regression decision tree to deal with different types of feature variables.

Second, Because we have re-selected the data, but the accuracy of different data varies greatly, we need to study what factors affect the error.

https://github.com/xu842251462/cs542FinalProject