* Code _ 10조(박성원, 정은서, 김주현)

- **dataPreprocessing_.py**

```
import pandas as pd

import numpy as np


# Function to return a column which has max correlation.

def maxCorr(dataset, missing_col) :


    max_corr_col = dataset[missing_col].idxmax()


    return max_corr_col


#Function to sort the data by max_corr_col and fill missing values in missing_col by ffill

def sortCol(dataset, missing_col, max_corr_col) :


    dataset.sort_values(by = max_corr_col, inplace = True)


    dataset[missing_col].fillna(method = 'ffill',inplace = True)


    return dataset


#Function to preprocessing

def preprocessing(df) :

    # a dataframe for the absolute value of the correlation of each column.

    corr = df.corr().abs()
```

```
    # Replace the correlation of the same column with 0
    corr = corr.replace(1, 0)

    #dataset's columns.
    df_col = df.columns

    # check every column
    for i in df_col :

        # if it has missing data fill them.
        if (df[i].isnull().sum() > 0) :

            # find a column which has max correaltion value
            max_corr_col = maxCorr(corr, i)

            # fill missing values by sortCol function
            df = sortCol(df, i, max_corr_col)

    return df

# Load the dataset from csv file
df = pd.read_csv('heart_disease.csv')
df.drop(columns = ['education'], inplace = True)

#check
```

```
print(df.head(10))


#check the missing data
print(df.isnull().sum())


#fill NaN values by preprocessing function
df = preprocessing(df)


#sort the dataset by index
df.sort_index(inplace = True)


#check that all missing data is filled
print(df.isnull().sum())


#store preprocessed data by csv file
df.to_csv('PreprocessedData.csv', index = False)
```

- **KNN_algorithm.py**

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.model_selection import train_test_split

warnings.filterwarnings(action='ignore')


df=pd.read_csv('PreprocessedData.csv')


X=df.drop(columns=['TenYearCHD'])

y=df['TenYearCHD'].values


#Scaling data(MinMax Scaler)

scaler=MinMaxScaler()

X=scaler.fit_transform(X)


x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3)


#k value candidate

k_list = range(1,50)

accuracies = []

max=0    #the max accuracy

index=0     #the k value which has max accuracy


#Find the best K value

for k in k_list:

        knn= KNeighborsClassifier(n_neighbors = k)

        knn.fit(x_train,y_train)

        accuracies.append(knn.score(x_test,y_test))

        if(max<knn.score(x_test,y_test)):
```

```
            max=knn.score(x_test,y_test)
            index=k

#display best k value
print("The best k value :",index)
#display best accuracy
print("** The accuracy of prediction: " , max)

#Draw a plot showing the accuracy of each K value.
plt.plot(k_list, accuracies)
plt.xlabel("k")
plt.ylabel("Validation Accuracy")
plt.title("Classifier Accuracy")
plt.show()
```

- **Logistic_regression.py**

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

```
# load the preprocessed dataset
df = pd.read_csv('PreprocessedData.csv')

#divide the dataset into input feature and target attribute
x = df.drop('TenYearCHD', axis=1)
y = df['TenYearCHD']

# Scaling
standard_scaler = StandardScaler()
x = standard_scaler.fit_transform(x)

# Splitting the dataset into Training and Test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

# Parameter candidate
C = np.logspace(-3, 3, 7)

penalty = ["l2"]

param_grid = dict(C=C, penalty=penalty)

# Create Logistic Regression model
logistic_model = LogisticRegression()

# Find a model which has the best parameter by Grid Search
grid_search_model = GridSearchCV(logistic_model, param_grid, cv=5)
```

```
# fit the data to the best model(training)
best_model = grid_search_model.fit(x_train, y_train)

# Measure accuracy of this model
best_score = grid_search_model.best_score_

# print the result(best accuracy, the best parameter)
print('Best Score : ', best_score)
print('Best C :', best_model.best_estimator_.get_params()['C'])
print('Best Penalty : ', best_model.best_estimator_.get_params()['penalty'])

#####################################

# predict y_test values by x_test
y_pred = grid_search_model.predict(x_test)

# make confusion matrix by y_test and y_pred
cnf_metrix = metrics.confusion_matrix(y_test, y_pred)

# calculate accurancy by confusion matrix
total = np.sum(cnf_metrix)

TP = cnf_metrix[0][0]
TN = cnf_metrix[1][1]
FP = cnf_metrix[1][0]
```

```
FN = cnf_metrix[0][1]


confusion_accurancy = round((TP + TN) / total, 2)


#the model's score
best_score = round(grid_search_model.best_score_, 2)


#display the score of model and the accurancy which calculated by confusion matrix
print('Confusion_matrix_score :', confusion_accurancy)
print('Model_score : ', best_score)


# draw heatmap by   confusion matrix
sns.heatmap(pd.DataFrame(cnf_metrix), annot=True, cmap='YlGnBu', fmt='g')
plt.title("Confusion matrix")
plt.ylabel("Actual label")
plt.xlabel("Predict label")
plt.show()
```

- **KNN_kfold.py**

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedShuffleSplit

warnings.filterwarnings(action='ignore')

df=pd.read_csv('PreprocessedData.csv')

X=df.drop(columns=['TenYearCHD'])
y=df['TenYearCHD'].values

# MinMaxScaler
mmscaler=MinMaxScaler()
mmX=mmscaler.fit_transform(X)

# StandardScaler
stdscaler=StandardScaler()
stdX=stdscaler.fit_transform(X)

# RobustScaler
robscaler=RobustScaler()
robX=robscaler.fit_transform(X)

#train/test
stratified_shuffle_split = StratifiedShuffleSplit(train_size=0.7, test_size=0.3, n_splits=10)
```

```
knn_cv = KNeighborsClassifier (n_neighbors = 31)
 #10-fold cross validation
mm_scores = cross_val_score (knn_cv, mmX, y, cv = stratified_shuffle_split,scoring='accuracy')
mm_avg_score = mm_scores.mean()

std_scores=cross_val_score (knn_cv, stdX, y, cv = stratified_shuffle_split,scoring='accuracy')
std_avg_score = std_scores.mean()

rob_scores=cross_val_score (knn_cv, robX, y, cv = stratified_shuffle_split,scoring='accuracy')
rob_avg_score = rob_scores.mean()

print('MinMax Average score :', round(mm_avg_score, 2))
print('Scores :', mm_scores)
print()

print('Standard Average score :', round(std_avg_score, 2))
print('Scores :', std_scores)
print()

print('Robust Average score :', round(rob_avg_score, 2))
print('Scores :', rob_scores)
```

- **Logistic_Kfold.py**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import RobustScaler

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import cross_val_score

from sklearn.model_selection import StratifiedShuffleSplit


df = pd.read_csv('PreprocessedData.csv')


#divide data  by input feature and  target attribute

x = df.drop('TenYearCHD', axis=1)

y = df['TenYearCHD']


#Standard Scaler

standard_scaler = StandardScaler()

std_x = standard_scaler.fit_transform(x)


#Robust Scaler

robust_scaler = RobustScaler()

rob_x = robust_scaler.fit_transform(x)


# Create Logistic Regression model

logistic_model = LogisticRegression(C = 1.0, penalty = "l2")


#use StratifiedShuffleSplit for K-Fold(K = 10)

stratified_shuffle_split = StratifiedShuffleSplit(

      train_size=0.7, test_size=0.3, n_splits=10)
```

```
#10-fold cross validataion
std_scores = cross_val_score(logistic_model, std_x, y, cv=stratified_shuffle_split)
std_avg_score = std_scores.mean()

rob_scores = cross_val_score(logistic_model, rob_x, y, cv=stratified_shuffle_split)
rob_avg_score = rob_scores.mean()

print('Standard Average score :', round(std_avg_score, 2))
print('Scores :', std_scores)
print('₩n')
print('Robust Average score :', round(rob_avg_score, 2))
print('Scores :', rob_scores)
```

- **Random_forest.py**

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

#Read preprocessed data from csv file
```

```
df = pd.read_csv('PreprocessedData.csv')


# divide the dataset into input feature and targe attribute
x = df.drop('TenYearCHD', axis=1)
y = df['TenYearCHD']


# Scaling
standard_scaler = StandardScaler()
x = standard_scaler.fit_transform(x)


# Split the dataset into train and test dataset
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)


# Create random forest model
random_forest_model = RandomForestClassifier()



# fit the data to random forest model
ensemble_model = random_forest_model.fit(x_train, y_train)


# predict target value with test input data
y_pred = ensemble_model.predict(x_test)


# Make confusion matrix by predict result
cnf_metrix = metrics.confusion_matrix(y_test, y_pred)
```

```
# calcualte accuracy by confusion matrix
total = np.sum(cnf_metrix)

TP = cnf_metrix[0][0]
TN = cnf_metrix[1][1]
FP = cnf_metrix[1][0]
FN = cnf_metrix[0][1]

confusion_accurancy = round((TP + TN) / total, 2)

# display the result(the model's accuracy)
print('Confusion_matrix_score :', confusion_accurancy)

# Draw heatmap with confusion matrix
sns.heatmap(pd.DataFrame(cnf_metrix), annot=True, cmap='YlGnBu', fmt='g')
plt.title("Confusion matrix", y=1.1)
plt.ylabel("Actual label")
plt.xlabel("Predict label")
plt.show()
```

- **RandomForest_Kfold.py**

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedShuffleSplit

df = pd.read_csv('PreprocessedData.csv')

#divide data  by input feature and  target attribute
x = df.drop('TenYearCHD', axis=1)
y = df['TenYearCHD']

#Standard Scaler
standard_scaler = StandardScaler()
std_x = standard_scaler.fit_transform(x)

#Robust Scaler
robust_scaler = RobustScaler()
rob_x = robust_scaler.fit_transform(x)

# Create Random Forest model
random_forest_model = RandomForestClassifier()

#use StratifiedShuffleSplit for K-Fold(K = 10)
stratified_shuffle_split = StratifiedShuffleSplit(
      train_size=0.7, test_size=0.3, n_splits=10)
```

```
#10-fold cross validataion
std_scores = cross_val_score(random_forest_model, std_x, y, cv=stratified_shuffle_split)
std_avg_score = std_scores.mean()

rob_scores = cross_val_score(random_forest_model, rob_x, y, cv=stratified_shuffle_split)
rob_avg_score = rob_scores.mean()

#print result
print('Standard Average score :', round(std_avg_score, 2))
print('Scores :', std_scores)
print('₩n')
print('Robust Average score :', round(rob_avg_score, 2))
print('Scores :', rob_scores)
```