北 京 科 技 大 学

生产实习报告

学　　院	数理学院
专业班级	信计 182
姓　名	张宇洋
学　号	41821219
指导教师	吕国才

2021 年 7 月

说　　明

一、 生产实习是本科教学重要的实践教学环节，是提高学生应用能力的重要措施。所有参加生产实习的学生都必须撰写实习报告

二、 生产实习报告的质量反映了生产实习的教学质量，它是评定生产实习成绩的主要依据之一。不交实习报告者不得参加实习成绩评定。

三、 实习报告要求条理清晰，内容详尽，数据准确。

四、 生产实习活动结束后请将本报告上交学院存档保存。

报告一、机器学习

一、开发语言和环境

1、开发技术：numpy、pandas、sklearn

2、开发环境：python3、spyder、macOS

二、系统分析

1、需求分析：为矿石数据集做二分类的预测模型

2、系统结构：

数据结构：声纳数据集（Sonar Dataset）------- sonar.all-data.csv

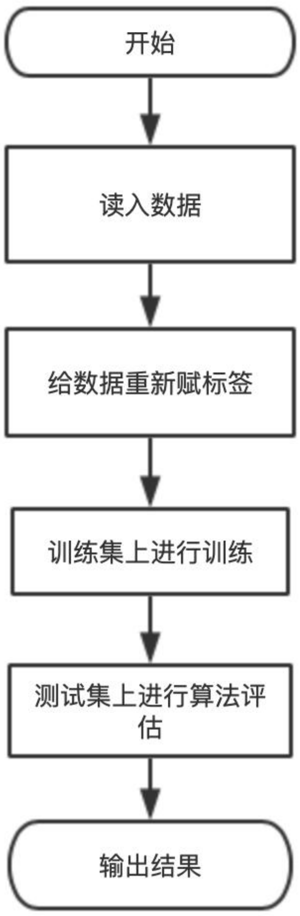
一共有 208 个观察值，60 个输入变量（不同的声呐探测数据）和 1 个输出变量（分类结果，M 为金属，R 为岩石）。各输入变量的数据类型均为 float64。

数据处理方法：利用 train_test_split 函数将数据分成训练集与测试集，在训练集上进行模型训练，并在测试集上进行模型评估。

所用算法：8 种机器学习常用的分类回归算法，SVM、EM、Kmeans、Kneighbors、Cart、C4.5、AdaBoost、NaiveBayes

三、主要代码：

1.算法流程图



2.程序代码

```
import pandas as   pd
import numpy as np
from sklearn.model_selection import   train_test_split
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score


fname = 'sonar.all-data.csv'
raw_data = pd.read_csv(fname)
data = raw_data.values
X = np.array(data[:, 0:60])
Y = np.array(data[:, 60])
Y_to_int = []
for i in range(0, len(Y)):
    if Y[i] == 'R':
        Y_to_int.append(0)
    else:
        Y_to_int.append(1)
        Y = np.array(Y_to_int)
test_size = 0.2
seed = 0
X_train,X_test,Y_train,Y_test = train_test_split(X, Y, test_size = test_size, random_state =
seed)


###########################    SVM    ###############################
from sklearn import svm
svm = svm.SVC(kernel = 'rbf', gamma = 1, C = 1).fit(X_train, Y_train)s
vm_pred = svm.predict(X_test)
#accuracy_svm = svm.score(X_test, Y_test)
#print("Accuracy of SVM Radial Basis Kernel:", accuracy_svm)
accuracy_svm = metrics.f1_score(Y_test, svm_pred, average = 'weighted')
print("F1 of SVM:", accuracy_svm)



###########################    EM    ###############################
from sklearn.mixture import GaussianMixturegm = GaussianMixture(n_components = 2,
random_state = 0).fit(X_train, Y_train)gm_pred = gm.predict(X_test)accuracy_gm =
gm.score(X_test, Y_test)
#accuracy_em = accuracy_score(Y_test, gm_pred)
```

```
#print("Result of EM:", accuracy_em)
accuracy_em = metrics.f1_score(Y_test, gm_pred, average = 'weighted')
print("F1 of EM:",accuracy_em)



############################     Kmeans    ###############################
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2, random_state = 0).fit(X_train, Y_train)
kmeans_pred = kmeans.predict(X_test)
accuracy_kmeans = metrics.f1_score(Y_test, kmeans_pred, average = 'weighted')
print("F1 of K-Means:", accuracy_kmeans)



############################     KNeighbors     ###########################
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3).fit(X_train, Y_train)
neigh_pred = neigh.predict(X_test)
accuracy_knn = metrics.f1_score(Y_test, neigh_pred, average = 'weighted')
print("F1 of KNN:", accuracy_knn)



############################      Cart       #############################
from sklearn import tree
cart = tree.DecisionTreeClassifier().fit(X_train, Y_train)
cart_pred = cart.predict(X_test)
accuracy_cart = metrics.f1_score(Y_test, cart_pred, average = 'weighted')
print("F1 of Cart:", accuracy_cart)



############################     C4.5       ###########################
from sklearn import treec45 = tree.DecisionTreeClassifier(criterion="entropy").fit(X_train,
Y_train)
c45_pred = c45.predict(X_test)
accuracy_c45 = metrics.f1_score(Y_test, c45_pred, average = 'weighted')
print("F1 of C4.5:", accuracy_c45)
```

```
############################     AdaBoost      ########################
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100, random_state=0).fit(X_train, Y_train)
ada_pred = ada.predict(X_test)
accuracy_ada = metrics.f1_score(Y_test, ada_pred, average = 'weighted')
print("F1 of AdaBoost:", accuracy_ada)
```

```
################### ######     naive_bayes     ########################
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB().fit(X_train, Y_train)
naive_bayes_pred = ada.predict(X_test)accuracy_
naive_bayes = metrics.f1_score(Y_test, naive_bayes_pred, average = 'weighted')   print("F1 of
Naive Bayes:", accuracy_naive_bayes)
```

四、**总结:**

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

利用 F1 值进行算法评估，得到各种算法的 F1 值如下：

F1 of SVM: 0.8330484330484331

F1 of EM: 0.4366471734892786

F1 of K-Means: 0.5468457468457469

F1 of KNN: 0.7838659328021029

F1 of Cart: 0.6788897377132672

F1 of C4.5: 0.7062184873949579

F1 of AdaBoost: 0.7282913165266107

F1 of Naive Bayes: 0.7282913165266107

可以看出在声纳数据上，带有 rbf 核函数的 SVM 效果比较好。

报告二、深度学习

一、开发语言和环境

1、开发技术：numpy、pandas、sklearn

2、开发环境：python3、spyder、macOS

二、系统分析

1、需求分析：解决手写数字集的识别问题，以及猫狗大战中对生活场景下猫和狗图片的识别。

2、系统结构：

基本流程：

神经网络：

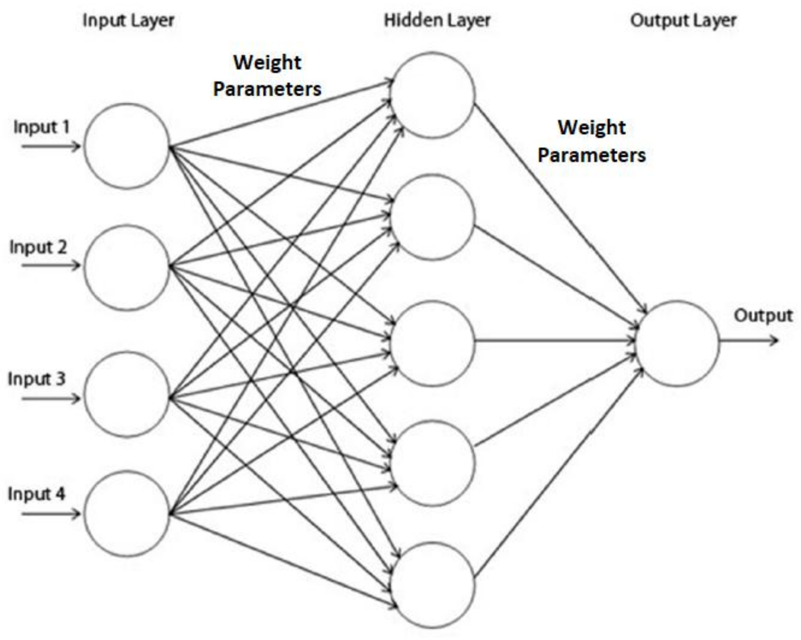
1. 获取数据，并做一定的预处理
2. 构建神经网络，初始化权重参数，准备开始训练
3. 将数据分成若干批次，代入模型跑一次前向运算并计算损失函数
4. 反向传播求偏导数，得到损失函数对权重的梯度
5. 使用优化算法修改权重，得到下一次前向传播的权重
6. 重复 3~5 步，每完成一个 epoch 输出一次预测结果
7. 完成设定次数的 epoch 后训练完成，得到最终模型，在此基础上对网络加以改进

所用数据集：

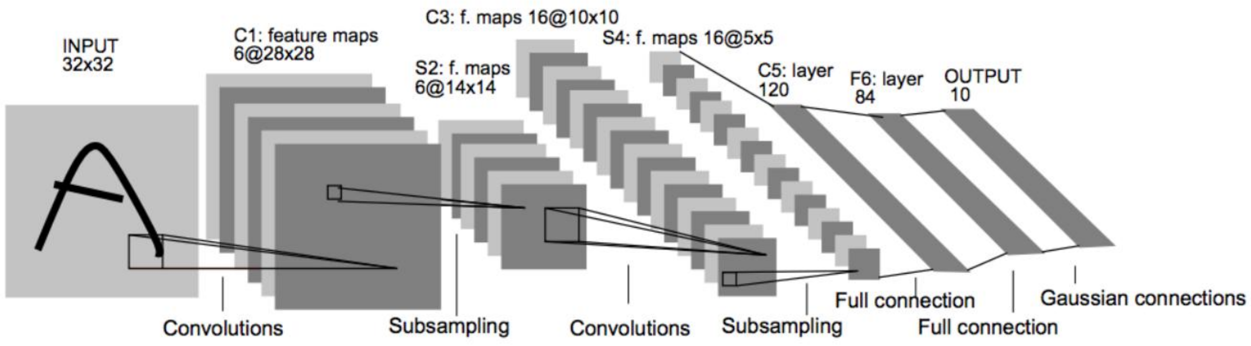
MNIST 手写数字识别数据集

三、主要代码：

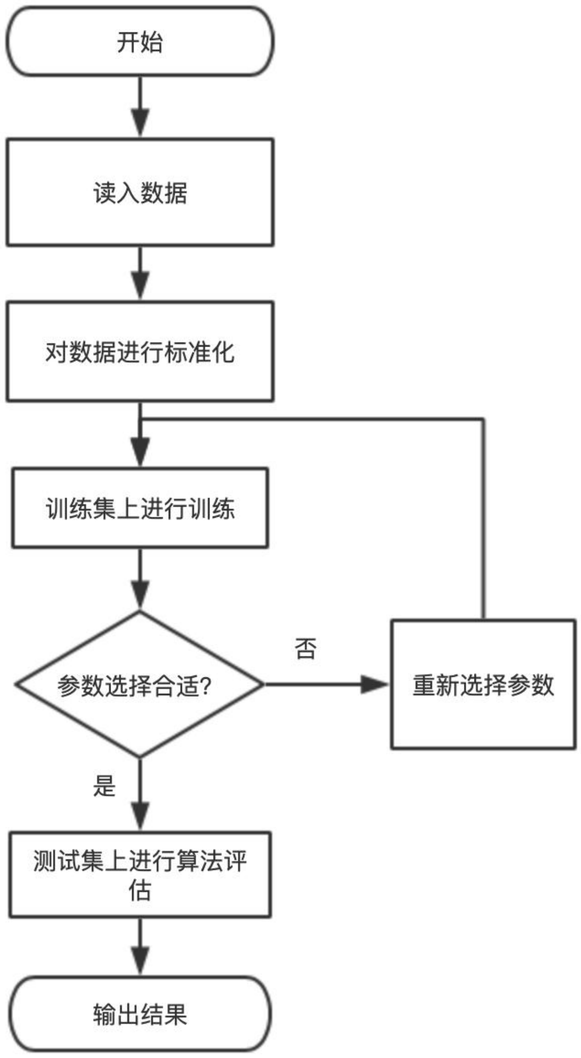
1.BP 神经网络基本构造：



2.卷积神经网络 LeNet-5：



3.算法流程



2.程序代码

1.BP 神经网络

```
#!/usr/bin/env python3
```

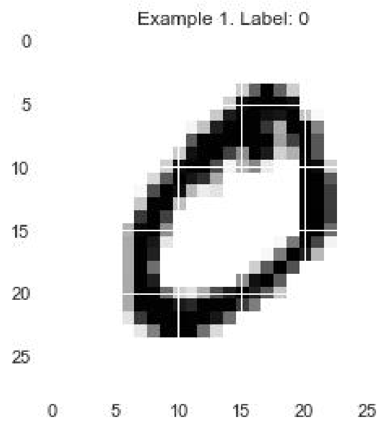
```
# -*- coding: utf-8 -*-
"""
Created on Tue Jul 13 17:17:25 2021
@author: zyy
"""
import pandas as  pd
import numpy as np
from sklearn.model_selection import  train_test_split
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
import struct,os
from array import array as pyarray
from numpy import append, array, int8, uint8, zeros
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score,classification_report
def load_mnist(image_file, label_file, path = "."):
    digits = np.arange(10)

    fname_image = os.path.join(path, image_file)
    fname_label = os.path.join(path, label_file)

    flbl = open(fname_label, 'rb')
    # >:  大端
    # I unsignedint
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    # b signedchar
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_image, 'rb')

    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    # B unsigned char
    img = pyarray("B", fimg.read())
    fimg.close()

    images = zeros((size, rows*cols), dtype = uint8)
    labels = zeros((size, 1), dtype = int8)
    for i in range(size):
        images[i] = array(img[ i *rows*cols : (i+1)*rows*cols ]).reshape((1, rows*cols))
        labels[i] = lbl[i]
        return images, labels

  if __name__=="__main__":
        train_image, train_label = load_mnist("train-images.idx3-ubyte",
"train-labels.idx1-ubyte")
        test_image, test_label = load_mnist("t10k-images.idx3-ubyte", "t10k-labels.idx1-ubyte")
        train_image = [im/255.0 for im in train_image]
        test_image = [im/255.0 for im in test_image]

        # alpha = 0.0001
        clf = MLPClassifier(solver = 'adam', max_iter = 100, alpha = 1e-3, hidden_layer_sizes =
(50,20), random_state = 1, verbose = True)
        clf.fit(train_image, train_label)
        # 模型效果获取
        r = clf.score(train_image, train_label)
        print("训练集准确率: %.4lf" % r)
        predict = clf.predict(test_image)
       print("预测集准确率: %.4lf" % accuracy_score(predict,test_label))
```

2.卷积神经网络 LeNet-5

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 18 12:41:41 2021
@author: zyy
"""


import gzip
import numpy as np
import pandas as pd
from time import time
from sklearn.model_selection import train_test_split
import tensorflow as tf
import keras
import keras.layers as layers
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from keras.callbacks import TensorBoard



def read_mnist(images_path: str, labels_path: str):
    with gzip.open(labels_path, 'rb') as labelsFile:
        labels = np.frombuffer(labelsFile.read(), dtype=np.uint8, offset=8)

    with gzip.open(images_path,'rb') as imagesFile:
        length = len(labels)
        # Load flat 28x28 px images (784 px), and convert them to 28x28 px
        features = np.frombuffer(imagesFile.read(), dtype=np.uint8, offset=16) \
                        .reshape(length, 784) \
                        .reshape(length, 28, 28, 1)

    return features, labels

import matplotlib.pyplot as pltimport seaborn as snssns.set()




train = {}
test = {}
from keras.datasets import mnist
#(train['features'], train['labels']),(test['features'], test['labels']) = mnist.load_data()
train['features'], train['labels'] = read_mnist('train-images-idx3-ubyte.gz',
'train-labels-idx1-ubyte.gz')
test['features'], test['labels'] = read_mnist('t10k-images-idx3-ubyte.gz',
't10k-labels-idx1-ubyte.gz')




print('# of training images:', train['features'].shape[0])print('# of test images:',
test['features'].shape[0])




def display_image(position):
    image = train['features'][position].squeeze()
    plt.title('Example %d. Label: %d' % (position, train['labels'][position]))
    plt.imshow(image, cmap=plt.cm.gray_r)
display_image(0)display_image(1)
train_labels_count = np.unique(train['labels'], return_counts=True)
dataframe_train_labels = pd.DataFrame({'Label':train_labels_count[0],
'Count':train_labels_count[1]})
validation = {}
train['features'], validation['features'], train['labels'], validation['labels'] =
train_test_split(train['features'], train['labels'], test_size=0.2, random_state=0)


print('# of training images:', train['features'].shape[0])
print('# of validation images:', validation['features'].shape[0])


# Pad images with 0s
train['features'] = np.pad(train['features'], ((0,0),(2,2),(2,2),(0,0)), 'constant')
validation['features'] = np.pad(validation['features'], ((0,0),(2,2),(2,2),(0,0)), 'constant')
test['features'] = np.pad(test['features'], ((0,0),(2,2),(2,2),(0,0)), 'constant')

print("Updated Image Shape: {}".format(train['features'][0].shape))

model = keras.Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu',
input_shape=(32,32,1)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(5, 5), activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='relu'))
model.add(layers.Dense(units=84, activation='relu'))
model.add(layers.Dense(units=10, activation = 'softmax'))
model.summary()
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(),
metrics=['accuracy'])
EPOCHS = 10
BATCH_SIZE = 128
X_train, y_train = train['features'], to_categorical(train['labels'])
X_validation, y_validation = validation['features'], to_categorical(validation['labels'])
train_generator = ImageDataGenerator().flow(X_train, y_train, batch_size=BATCH_SIZE)
validation_generator = ImageDataGenerator().flow(X_validation, y_validation,
batch_size=BATCH_SIZE)
```

```
print('# of training images:', train['features'].shape[0])
print('# of validation images:', validation['features'].shape[0])
steps_per_epoch = X_train.shape[0]//BATCH_SIZE
validation_steps = X_validation.shape[0]//BATCH_SIZE
tensorboard = TensorBoard(log_dir="logs/{}".format(time()))
model.fit_generator(train_generator, steps_per_epoch=steps_per_epoch, epochs=EPOCHS,
                              validation_data=validation_generator,validation_steps=validation_steps,
                                shuffle=True, callbacks=[tensorboard])
score = model.evaluate(test['features'], to_categorical(test['labels']))
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

四、**总结:**



神经网络训练过程（70 次迭代之后）

```
Iteration 70, loss = 0.00365914
Iteration 71, loss = 0.00362521
Iteration 72, loss = 0.00357932
Iteration 73, loss = 0.01018362
Iteration 74, loss = 0.02372291
Iteration 75, loss = 0.00735141
Iteration 76, loss = 0.00497743
Iteration 77, loss = 0.00406544
Iteration 78, loss = 0.00374507
Iteration 79, loss = 0.00359304
Iteration 80, loss = 0.00409540
Iteration 81, loss = 0.00373427
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
训练集准确率: 1.0000
预测集准确率: 0.9754
```

```
LeNet-5:
# of training images: 60000
# of test images: 10000
# of training images: 48000
# of validation images: 12000
Updated Image Shape: (32, 32, 1)
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_2 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_3 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_4 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_5 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
# of training images: 48000
# of validation images: 12000

375/375 [==============================] - 10s 26ms/step - loss: 1.8283 - accuracy: 0.7683 - val_loss: 0.1093 - val_accuracy: 0.9666
Epoch 2/10
375/375 [==============================] - 10s 27ms/step - loss: 0.0943 - accuracy: 0.9708 - val_loss: 0.0789 - val_accuracy: 0.9766
Epoch 3/10
375/375 [==============================] - 11s 28ms/step - loss: 0.0578 - accuracy: 0.9829 - val_loss: 0.0583 - val_accuracy: 0.9819
Epoch 4/10
```

```
375/375 [==============================] - 11s 29ms/step - loss: 0.0456 - accuracy: 0.9857 -
val_loss: 0.0597 - val_accuracy: 0.9820
Epoch 5/10
375/375 [==============================] - 11s 29ms/step - loss: 0.0317 - accuracy: 0.9898 -
val_loss: 0.0605 - val_accuracy: 0.9819
Epoch 6/10
375/375 [==============================] - 11s 29ms/step - loss: 0.0283 - accuracy: 0.9907 -
val_loss: 0.0573 - val_accuracy: 0.9832
Epoch 7/10
375/375 [==============================] - 13s 35ms/step - loss: 0.0275 - accuracy: 0.9909 -
val_loss: 0.0513 - val_accuracy: 0.9859
Epoch 8/10
375/375 [==============================] - 11s 30ms/step - loss: 0.0225 - accuracy: 0.9925 -
val_loss: 0.0667 - val_accuracy: 0.9840
Epoch 9/10
375/375 [==============================] - 12s 31ms/step - loss: 0.0225 - accuracy: 0.9926 -
val_loss: 0.0564 - val_accuracy: 0.9851
Epoch 10/10
375/375 [==============================] - 12s 31ms/step - loss: 0.0185 - accuracy: 0.9941 -
val_loss: 0.0562 - val_accuracy: 0.9852
313/313 [==============================] - 1s 2ms/step - loss: 0.0403 - accuracy: 0.9880
Test loss: 0.040332358330488205
Test accuracy: 0.9879999756813049
```

可以看到在 0.001 的误差水平下，模型在 81 次迭代时就已经收敛了，并且在训练集上的准确率接近 1，但是模型预测效果依然比较好，所以并没有发生比较严重的过拟合现象，可能是由于神经网络的参数比较多，拟合模型所需数据量比较大，并没有那么容易发生过拟合。

可以看出 LeNet-5 的效果也不错，并且更加迅速，准确率略高，当然调节 BP 神经网络的结果也能达到类似的效果但是训练时间比较长

报告三、社交网络

一、开发语言和环境

1、开发技术：numpy、pandas、sklearn

2、开发环境：python3、spyder、Ubuntu

二、系统分析

1、需求分析：从《权力的游戏》的关系背景出发，通过图论的基础发现社区的存在，并度量各种中心性，划分出最短路径。

2、系统结构：

算法流程：

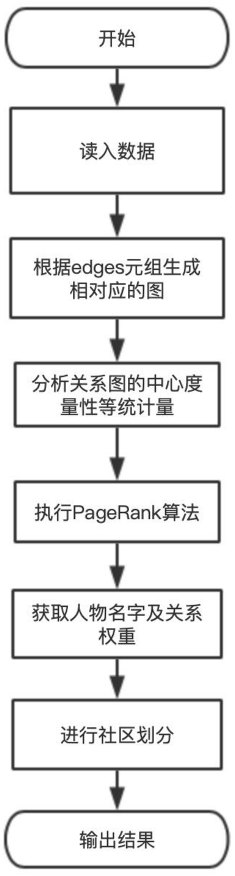
1. 读取数据，构建关系图
2. 计算各类指标并输出
3. 划分社区

所用数据：

权利的游戏人物关系表

三、主要代码：

主要流程：



```
#!/usr/bin/env python3
```

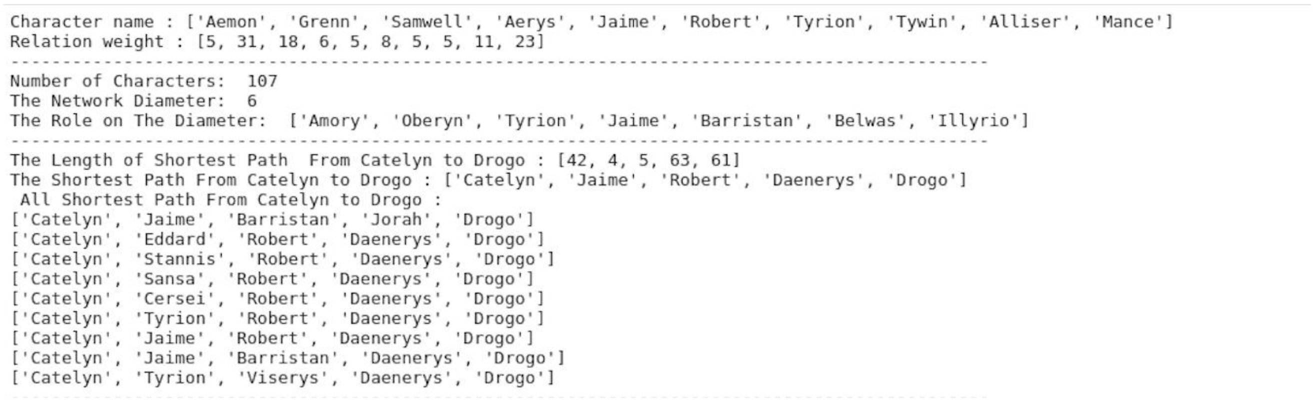
```
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 16 15:13:37 2021
@author: zyy
"""
import csv
from igraph import Graph as   IGraph
# read data and skip first line items# save data in edges
def read_data(filename):
        edges = []
        firstline = True
        with open(filename,'r',encoding = 'utf-8') as f:
                for row in csv.reader(f.read().splitlines()):
                        if firstline == True:
                                firstline = False
                                continue
                        u, v, weight = [i for i in row]
                        edges.append((u,v,int(weight)))
        return edges

# Using package IGraph TupleList function, generate graph depends on tuple edges
# undirected
def get_igraph(edges):
        g = IGraph.TupleList(edges = edges, directed = False, vertex_name_attr = "name", \
                                          edge_attrs = None, weights = True)
        return g

# Get Game of Thrones characters name and relation weight
# output 10 items
def get_graphData(graph):
        names = graph.vs["name"]
        weights = graph.es["weight"]
        print('Character name :', names[:10])
        print('Relation weight :', weights[:10])
        return names, weights

# Degree Centrality
def Degree_Centrality(g):
        p = list(zip(g.vs['name'],g.vs.degree()))
        p_sort = sorted(p,key = lambda p:p[1], reverse = True)
        return p_sort

# Weighted Degree Centrality
```

```
def Weighted_Degree_Centrality(g,edges):
        WDC = []
        for p in g.vs:
                neighbors = [x['name'] for x in p.neighbors()]
                weightedDegree = sum([w for a, b, w in edges for c in neighbors \
                                                if (a == p['name']) and    b == c or (b == p['name'] and a == c)])
                WDC.append([p['name'], weightedDegree])
        WDC_sort = sorted(WDC, key = lambda p:p[1], reverse = True)
        return WDC_sort


# Neighbor Average Degree
def Neighbor_Average_Degree(g):
        p = list(zip(g.vs['name'], g.knn()[0]))
        p_sort = sorted(p, key = lambda p:p[1], reverse = True)
        return p_sort

# Betweenness Centrality
def Betweenness_Centrality(g):
        p = list(zip(g.vs['name'], g.betweenness()))
        p_sort = sorted(p, key = lambda p:p[1], reverse = True)
        return p_sort

# Closeness Centrality
def Closeness_centrality(g):
        p = list(zip(g.vs['name'], g.closeness()))
        p_sort = sorted(p, key = lambda p:p[1], reverse = True)
        return p_sort

# PageRank
def PageRank(g):
        p = list(zip(g.vs['name'], g.pagerank()))
        p_sort = sorted(p, key = lambda p:p[1], reverse = True)
        return p_sort

# Community Detection
def Community_Detection(g):
        clusters = IGraph.community_walktrap(g, weights = "weight").as_clustering()
        n = len(clusters.sizes())
        for i in range(n):
                print('Community', i, ":", [g.vs['name'][j] for j in clusters[i]])
        return clusters
```

```
if __name__ == '__main__':
    # read data
    edges = read_data('stormofswords.csv')
    # generate graph
    g = get_igraph(edges)


    # characters name and relation weight
    names, weights = get_graphData(g)


    # Analyze
    print('--------------------------------------------------')
    # Number Of Characters
    print('Number of Characters: ', g.vcount())
    # The Role on The Diameter
    print('The Network Diameter: ', g.diameter())
    print('The Role on The Diameter: ', [names[x] for x in g.get_diameter()])



    print('--------------------------------------------------')
    # Get Shortest Path Using Function of IGraph , get_shortest_paths and
get_all_shortest_paths
    print('The Length of Shortest Path   From Catelyn to Drogo :', \
                                    g.get_shortest_paths('Catelyn','Drogo')[0])

    print('The Shortest Path From Catelyn to Drogo :', [names[x]     \
                                    for x in     g.get_shortest_paths('Catelyn', 'Drogo')[0]])
    print('\r All Shortest Path From Catelyn to Drogo :')
    paths = g.get_all_shortest_paths('Catelyn', 'Drogo')
    for p in paths:
        print([names[x] for x in p])

    print('--------------------------------------------------')


    # Degree Centrality
    print('Degree Centrality: ', Degree_Centrality(g)[:10])
    print('Weighted Degree Centrality: ', Weighted_Degree_Centrality(g, edges)[:10])
    print('Neighbor Average Degree: ', Neighbor_Average_Degree(g)[:10])
    print('Betweenness Centrality: ', Betweenness_Centrality(g)[:10])
    print('Closeness centrality: ', Closeness_centrality(g)[:10])
    print('PageRank Alg: ', PageRank(g)[:10])

    print('------------------------------------------------------------')
    print('Community Detection: ', Community_Detection(g))
```

四、总结:

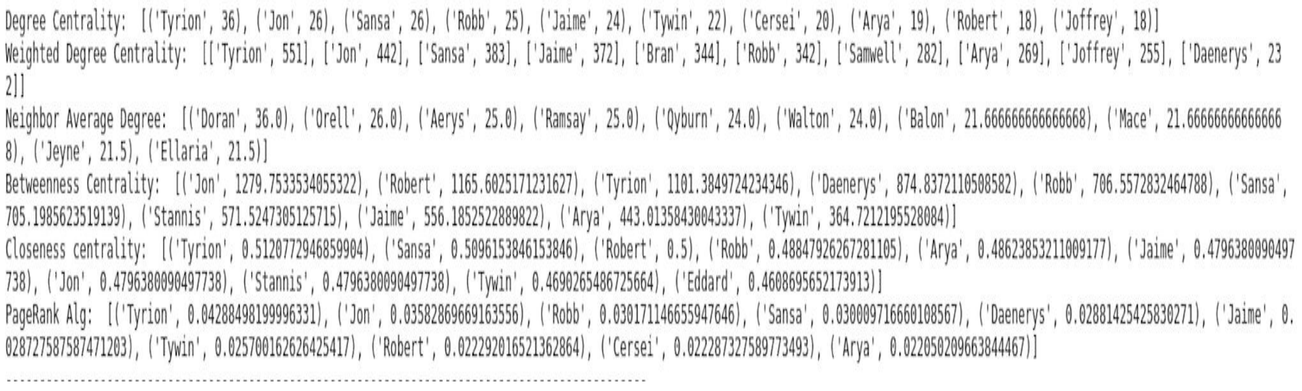
人物网络:



```
Character name : ['Aemon', 'Grenn', 'Samwell', 'Aerys', 'Jaime', 'Robert', 'Tyrion', 'Tywin', 'Alliser', 'Mance']
Relation weight : [5, 31, 18, 6, 5, 8, 5, 5, 11, 23]
------------------------------------------------------------------------------------------
Number of Characters:  107
The Network Diameter:  6
The Role on The Diameter:  ['Amory', 'Oberyn', 'Tyrion', 'Jaime', 'Barristan', 'Belwas', 'Illyrio']
------------------------------------------------------------------------------------------
The Length of Shortest Path  From Catelyn to Drogo : [42, 4, 5, 63, 61]
The Shortest Path From Catelyn to Drogo : ['Catelyn', 'Jaime', 'Robert', 'Daenerys', 'Drogo']
 All Shortest Path From Catelyn to Drogo :
['Catelyn', 'Jaime', 'Barristan', 'Jorah', 'Drogo']
['Catelyn', 'Eddard', 'Robert', 'Daenerys', 'Drogo']
['Catelyn', 'Stannis', 'Robert', 'Daenerys', 'Drogo']
['Catelyn', 'Sansa', 'Robert', 'Daenerys', 'Drogo']
['Catelyn', 'Cersei', 'Robert', 'Daenerys', 'Drogo']
['Catelyn', 'Tyrion', 'Robert', 'Daenerys', 'Drogo']
['Catelyn', 'Jaime', 'Robert', 'Daenerys', 'Drogo']
['Catelyn', 'Jaime', 'Barristan', 'Daenerys', 'Drogo']
['Catelyn', 'Tyrion', 'Viserys', 'Daenerys', 'Drogo']
------------------------------------------------------------------------------------------
```

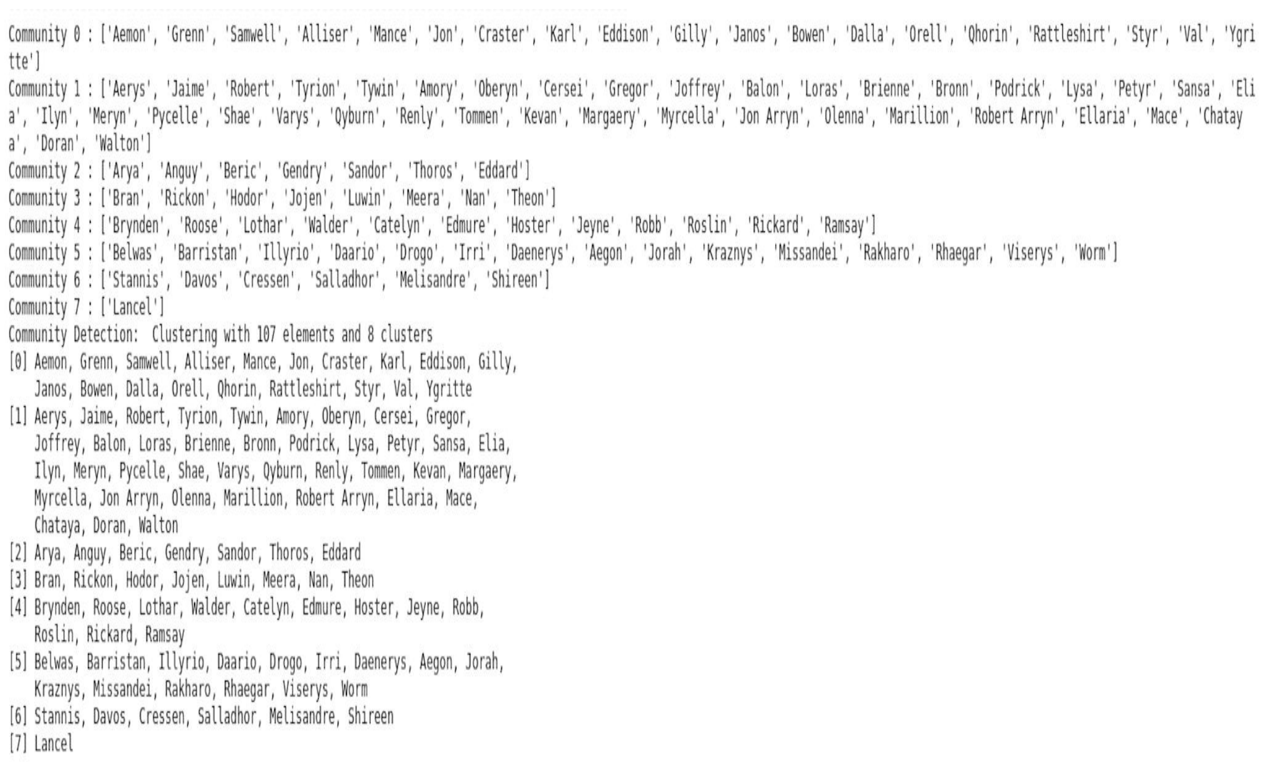
度中间性:

$$C(x) = \Big[\sum_{y\neq x}^{N} \frac{d(x,y)}{(N-1)}\Big]^{-1}$$



```
Degree Centrality:  [('Tyrion', 36), ('Jon', 26), ('Sansa', 26), ('Robb', 25), ('Jaime', 24), ('Tywin', 22), ('Cersei', 20), ('Arya', 19), ('Robert', 18), ('Joffrey', 18)]
Weighted Degree Centrality:  [['Tyrion', 551], ['Jon', 442], ['Sansa', 383], ['Jaime', 372], ['Bran', 344], ['Robb', 342], ['Samwell', 282], ['Arya', 269], ['Joffrey', 255], ['Daenerys', 23
2]]
Neighbor Average Degree:  [('Doran', 36.0), ('Orell', 26.0), ('Aerys', 25.0), ('Ramsay', 25.0), ('Qyburn', 24.0), ('Walton', 24.0), ('Balon', 21.666666666666668), ('Mace', 21.6666666666666
8), ('Jeyne', 21.5), ('Ellaria', 21.5)]
Betweenness Centrality:  [('Jon', 1279.7533534055322), ('Robert', 1165.6025171231627), ('Tyrion', 1101.3849724234346), ('Daenerys', 874.8372110508582), ('Robb', 706.5572832464788), ('Sansa',
705.1985623519139), ('Stannis', 571.5247305125715), ('Jaime', 556.1852522889822), ('Arya', 443.01358430043337), ('Tywin', 364.7212195528084)]
Closeness centrality:  [('Tyrion', 0.5120772946859904), ('Sansa', 0.5096153846153846), ('Robert', 0.5), ('Robb', 0.4884792626728105), ('Arya', 0.4862385321100917), ('Jaime', 0.4796380090497
738), ('Jon', 0.4796380090497738), ('Stannis', 0.4796380090497738), ('Tywin', 0.4690265486725664), ('Eddard', 0.4608695652173913)]
PageRank Alg:  [('Tyrion', 0.04288498199996331), ('Jon', 0.03582869669163556), ('Robb', 0.030171146655947646), ('Sansa', 0.030009716660108567), ('Daenerys', 0.02881425425830271), ('Jaime', 0.
028727587587471203), ('Tywin', 0.025700162626425417), ('Robert', 0.022292016521362864), ('Cersei', 0.022287327589773493), ('Arya', 0.022050209663844467)]
------------------------------------------------------------------------------------------
```

社区划分:



```
Community 0 : ['Aemon', 'Grenn', 'Samwell', 'Alliser', 'Mance', 'Jon', 'Craster', 'Karl', 'Eddison', 'Gilly', 'Janos', 'Bowen', 'Dalla', 'Orell', 'Qhorin', 'Rattleshirt', 'Styr', 'Val', 'Ygri
tte']
Community 1 : ['Aerys', 'Jaime', 'Robert', 'Tyrion', 'Tywin', 'Amory', 'Oberyn', 'Cersei', 'Gregor', 'Joffrey', 'Balon', 'Loras', 'Brienne', 'Bronn', 'Podrick', 'Lysa', 'Petyr', 'Sansa', 'Eli
a', 'Ilyn', 'Meryn', 'Pycelle', 'Shae', 'Varys', 'Qyburn', 'Renly', 'Tommen', 'Kevan', 'Margaery', 'Myrcella', 'Jon Arryn', 'Olenna', 'Marillion', 'Robert Arryn', 'Ellaria', 'Mace', 'Chatay
a', 'Doran', 'Walton']
Community 2 : ['Arya', 'Anguy', 'Beric', 'Gendry', 'Sandor', 'Thoros', 'Eddard']
Community 3 : ['Bran', 'Rickon', 'Hodor', 'Jojen', 'Luwin', 'Meera', 'Nan', 'Theon']
Community 4 : ['Brynden', 'Roose', 'Lothar', 'Walder', 'Catelyn', 'Edmure', 'Hoster', 'Jeyne', 'Robb', 'Roslin', 'Rickard', 'Ramsay']
Community 5 : ['Belwas', 'Barristan', 'Illyrio', 'Daario', 'Drogo', 'Irri', 'Daenerys', 'Aegon', 'Jorah', 'Kraznys', 'Missandei', 'Rakharo', 'Rhaegar', 'Viserys', 'Worm']
Community 6 : ['Stannis', 'Davos', 'Cressen', 'Salladhor', 'Melisandre', 'Shireen']
Community 7 : ['Lancel']
Community Detection:  Clustering with 107 elements and 8 clusters
[0] Aemon, Grenn, Samwell, Alliser, Mance, Jon, Craster, Karl, Eddison, Gilly,
    Janos, Bowen, Dalla, Orell, Qhorin, Rattleshirt, Styr, Val, Ygritte
[1] Aerys, Jaime, Robert, Tyrion, Tywin, Amory, Oberyn, Cersei, Gregor,
    Joffrey, Balon, Loras, Brienne, Bronn, Podrick, Lysa, Petyr, Sansa, Elia,
    Ilyn, Meryn, Pycelle, Shae, Varys, Qyburn, Renly, Tommen, Kevan, Margaery,
    Myrcella, Jon Arryn, Olenna, Marillion, Robert Arryn, Ellaria, Mace,
    Chataya, Doran, Walton
[2] Arya, Anguy, Beric, Gendry, Sandor, Thoros, Eddard
[3] Bran, Rickon, Hodor, Jojen, Luwin, Meera, Nan, Theon
[4] Brynden, Roose, Lothar, Walder, Catelyn, Edmure, Hoster, Jeyne, Robb,
    Roslin, Rickard, Ramsay
[5] Belwas, Barristan, Illyrio, Daario, Drogo, Irri, Daenerys, Aegon, Jorah,
    Kraznys, Missandei, Rakharo, Rhaegar, Viserys, Worm
[6] Stannis, Davos, Cressen, Salladhor, Melisandre, Shireen
[7] Lancel
```

在实验过程中安装 igraph 的包可能会遇到问题，首先 igraph 与 python-igraph 不能共存，需要卸载 python-igraph 包，如果使用 Linux 系统不会遇到其他问题，如果使用 windows 系统，由于此包已经不再被官方支持，属于旧包我们需要自己手动安装，首先运行 pip debug --verbose 命令查看 pip 支持 whl 版本，然后再登陆 https://www.fld.uci.edu/~gohlke/pythonlibs 下载支持的 python-igraph 与 pycairo 再利用 pip 手动安装即可以解决问题。

报告四、数据分析

一、开发语言和环境

1、开发技术：numpy、pandas、sklearn

2、开发环境：python3、spyder、macOS

二、系统分析

1、需求分析：为波士顿房价做回归预测模型

2、系统结构：

数据结构：

波士顿房价数据集（Boston House Proce）------- housing.csv

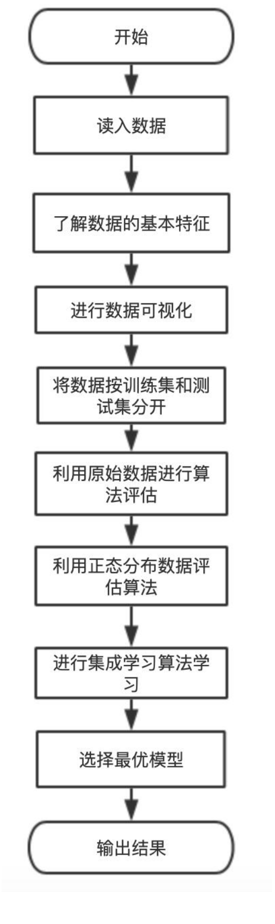
该数据集中每一行数据都是对波士顿周边或城镇房价的描述，是 1978 年统计收集的，数据中包含 14 个特征和 506 条数据。其中前 13 个特征为输入变量，第 14 个特征为输出变量。

所用算法：

线性算法：LA、LASSO、EN，非线性算法 CART、SVM、KNN，并选择集成学习种 Bagging 的中的 RF 和 ET 算法，与 Boosting 中的 Ada、GBM 算法。

三、主要代码：

1.主要流程



2.程序代码

#!/usr/bin/env python3

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 16 01:58:22 2021
@author: zyy
"""

import numpy as np
from numpy import arange
from matplotlib import pyplot
from pandas import read_csv
from pandas import set_option
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error

# import   data
filename = '02-boston_house_prices.csv'
names = ['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PRTATTO','B',
'LSTAT','MEDV']
dataset = read_csv(filename, names = names,encoding = 'utf-8', header = 1)
# out
# print data dim
print(dataset.shape)
# check data type
print(dataset.dtypes)
#dataset['CRIM'] = dataset['CRIM'].astype(np.float)
# check 10 head
print(dataset.head(10))
print('DATA decribe:')
print(dataset.describe())

print('Pearson Corr:')
print(dataset.corr(method='pearson'))

##############################################################################
#                                   Data Visualization
##############################################################################

# Histogram
dataset.hist(sharex = False, sharey = False, xlabelsize = 1, ylabelsize = 1)
pyplot.show()

# Density
dataset.plot(kind ='density',subplots = True, layout=(4,4), sharex = False, fontsize = 1)
pyplot.show()

# Boxplot
dataset.plot(kind ='box', subplots = True, layout = (4,4), sharex = False, sharey = False, fontsize
= 8)
pyplot.show()

# Scatter Matrix
scatter_matrix(dataset)
pyplot.show()

# Correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
ca = ax.matshow(dataset.corr(), vmin = -1, vmax = 1, interpolation = 'none')
fig.colorbar(cax)
ticks = np.arange(0, 14, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
labels(names)
pyplot.show()


# Split
array = dataset.values
```

```
X = array[:, 0:13]
Y = array[:, 13]
validation_size = 0.2
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y,test_size = validation_size,
random_state = seed, shuffle = True)

# 10 flod check, using NMSE Function Evaluate
num_folds = 10
seed = 7
scoring = 'neg_mean_squared_error'

# Linear Model:
# Linear Regression  (LR)
# least absolute shrinkage and selection  (LASSO)
# ElasticNet  (EN)
models = {}
models['LR'] = LinearRegression()
models['LASSO'] = Lasso()
models['EN'] = ElasticNet()
# non-Linear Model:
# Classification And Regression Tree  (CART)
# Support vector machine  (SVM)
# k-nearest neighbors   (KNN)
models['KNN'] = KNeighborsRegressor()
models['CART'] = DecisionTreeRegressor()
models['SVM'] = SVR()


results = []
for key in models:
    kfold = KFold(n_splits = num_folds, random_state = seed, shuffle = True)
    cv_result = cross_val_score(models[key], X_train, Y_train, cv = kfold, scoring = scoring)
    results.append(cv_result)
    print('%13s: mean: %.4f       std: %.4f' % (key, cv_result.mean(), cv_result.std()))
# Boxplot
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

```
# Evaluate
# Use normally distributed data to evaluate the model
pipelines = {}
pipelines['ScalerLR'] = Pipeline([('Scaler', StandardScaler()), ('LR',LinearRegression())])
pipelines['ScalerLASSO'] = Pipeline([('Scaler', StandardScaler()), ('LASSO', Lasso())])
pipelines['ScalerEN'] = Pipeline([('Scaler',StandardScaler()), ('EN', ElasticNet())])
pipelines['ScalerKNN'] = Pipeline([('Scaler',StandardScaler()), ('KNN',KNeighborsRegressor())])
pipelines['ScalerCART'] = Pipeline([('Scaler',StandardScaler()),
('CART',DecisionTreeRegressor())])
pipelines['ScalerSVM'] = Pipeline([('Scaler',StandardScaler()), ('SVM', SVR())])
results = []for key in pipelines:
    kfold = KFold(n_splits=num_folds, random_state = seed, shuffle = True)
    cv_result = cross_val_score(pipelines[key], X_train, Y_train, cv = kfold, scoring = scoring)
    results.append(cv_result)
    print('%13s: mean: %.4f      std: %.4f' % (key, cv_result.mean(), cv_result.std()))

# Boxplot
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(pipelines.keys())
pyplot.show()



print('\n*************************    KNN    *************************')
# Choice Best Neighbor Number
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_neighbors': [1, 2, 3, 4, 5, 7, 9, 11, 15, 17, 20]}
model = KNeighborsRegressor()
kfold = KFold(n_splits = num_folds, random_state = seed, shuffle = True)
grid = GridSearchCV(estimator = model, param_grid = param_grid, scoring = scoring, cv =
kfold)
grid_result = grid.fit(X = rescaledX, y = Y_train)
print('Best KNN Result parameter: %s    Use: %s' % (grid_result.best_score_,
grid_result.best_params_))
cv_results =
zip(grid_result.cv_results_['mean_test_score'],grid_result.cv_results_['std_test_score'],grid_resu
lt.cv_results_['params']
)
```

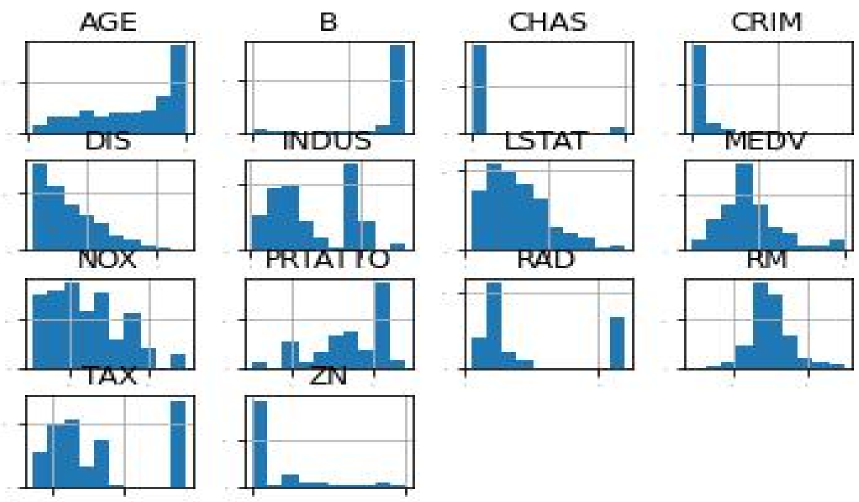
```
for mean, std, param in cv_results:
        print('mean: %.4f std: %f with %r' % (mean, std, param))

################################   Enseemble   ###################################
#   Bagging : randomForest  (RF)  Extremely randomized trees(ET)
#   Boosting:  AdaBoost  (Ada)  Gradient Boosting Machin  (GBT)
################################################################################
print('\n************************   Bagging   **************************')
ensembles = {}
# CartReg
ensembles['ScaledAB'] = Pipeline([('Scaler',StandardScaler()), ('AB', AdaBoostRegressor())])
ensembles['ScaledAB-KNN'] = Pipeline([('Scaler',StandardScaler()), ('ABKNN',
AdaBoostRegressor(base_estimator = KNeighborsRegressor(n_neighbors = 3)))])
ensembles['ScaledAB-LR'] = Pipeline([('Scaler',StandardScaler()),
('ABLR',AdaBoostRegressor(LinearRegression()))])
ensembles['ScaledGBR'] = Pipeline([('Scaler',StandardScaler()), ('RBR',
GradientBoostingRegressor())])
# *********************   Add   ***************************
ensembles['ScaledAB-DR'] = Pipeline([('Scaler',StandardScaler()),
('RBR',AdaBoostRegressor(DecisionTreeRegressor()))])


ensembles['ScaledRFR'] = Pipeline([('Scaler',StandardScaler()), ('RFR',
RandomForestRegressor())])
ensembles['ScaledETR'] = Pipeline([('Scaler',StandardScaler()), ('ETR', ExtraTreesRegressor())])
results = []
for key in ensembles:
        kfold = KFold(n_splits = num_folds, random_state = seed, shuffle = True)
        cv_result = cross_val_score(ensembles[key], X_train, Y_train, cv = kfold, scoring =
scoring)
        results.append(cv_result)
        print('%13s: mean: %.4f       std: %.4f' % (key, cv_result.mean(), cv_result.std()))
# Boxplot
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xtick
labels(ensembles.keys())
pyplot.show()
```

```
# Adjustment parameters    ---- GBM
print('\n*************************    GBM    *************************')
caler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_estimators': [10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900]}
model = GradientBoostingRegressor()
kfold = KFold(n_splits = num_folds, random_state = seed, shuffle = True)
grid = GridSearchCV(estimator = model, param_grid=param_grid, scoring=scoring, cv =
kfold)grid_result = grid.fit(X = rescaledX, y = Y_train)
print('Best GBM: %s Use :%s' % (grid_result.best_score_, grid_result.best_params_))


# Adjustment parameters    ---- ET
print('\n*************************    ET    *************************')
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_estimators': [5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}
model = ExtraTreesRegressor()
kfold = KFold(n_splits = num_folds, random_state = seed, shuffle = True)
grid = GridSearchCV(estimator = model, param_grid = param_grid, scoring = scoring, cv =
kfold)grid_result = grid.fit(X = rescaledX, y = Y_train)
print(' Best ET: %s USe:%s' % (grid_result.best_score_, grid_result.best_params_))

# Adjustment parameters    ---- AdaDT
print('\n***************    Ada_DecisionTreeRegressor    ********************')
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_estimators': [100, 120, 130, 140 ,150]}
model = AdaBoostRegressor(DecisionTreeRegressor(max_depth = 100), random_state = seed,
loss='linear')
kfold = KFold(n_splits = num_folds, random_state = seed, shuffle = True)
grid = GridSearchCV(estimator = model, param_grid = param_grid, scoring = scoring, cv =
kfold)grid_result = grid.fit(X = rescaledX, y = Y_train)
print(' Best Ada: %s USe:%s' % (grid_result.best_score_, grid_result.best_params_))



###########################    Final Training    #####################
```

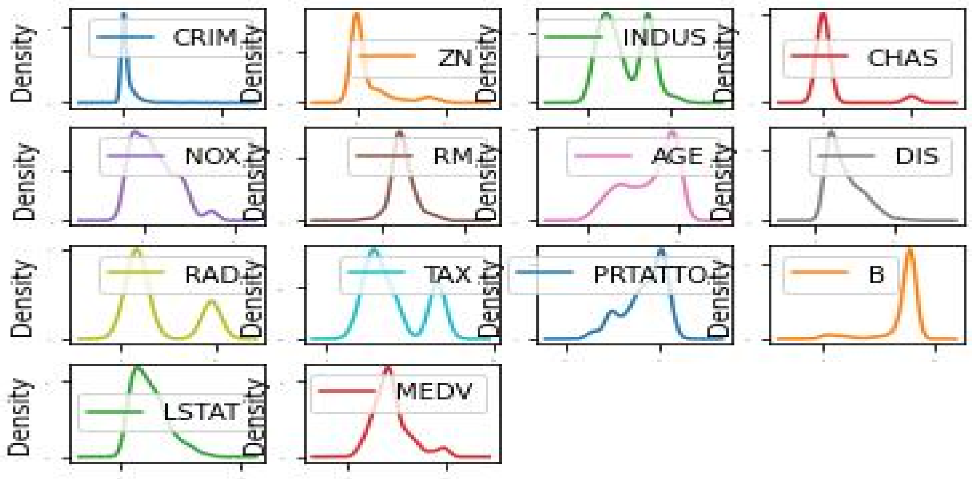
```
print('\n*******************    Final Training    *************************')#
Trainingcaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
gbr = ExtraTreesRegressor(n_estimators = 80)
gbr.fit(X = rescaledX, y = Y_train)
# Test
rescaledX_validation = scaler.transform(X_validation)
predictions = gbr.predict(rescaledX_validation)
print('Prediction set ET', mean_squared_error(Y_validation, predictions))

# Training
caler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
gbr = GradientBoostingRegressor(n_estimators = 600)
gbr.fit(X = rescaledX, y = Y_train)
# Test
rescaledX_validation = scaler.transform(X_validation)
predictions = gbr.predict(rescaledX_validation)
print('Prediction set GBR', mean_squared_error(Y_validation, predictions))

# Training
caler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
gbr = AdaBoostRegressor(DecisionTreeRegressor(max_depth = 100), random_state=seed,
loss='linear')
gbr.fit(X = rescaledX, y = Y_train)
# Test
rescaledX_validation = scaler.transform(X_validation)
predictions = gbr.predict(rescaledX_validation)
print('Prediction set AdaDR', mean_squared_error(Y_validation, predictions))
```

四、 **总结:**

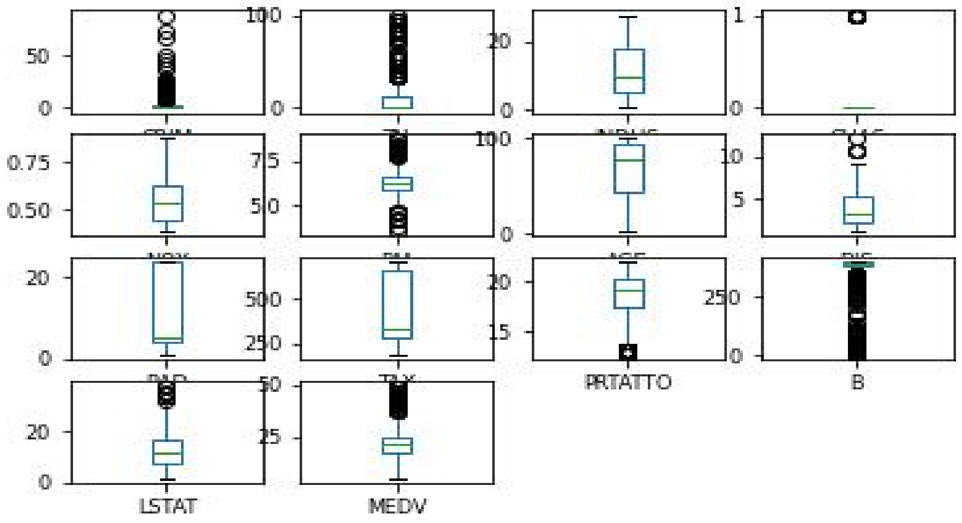
数据直方图:


AGE
B
CHAS
CRIM
DIS
INDUS
LSTAT
MEDV
NOX
PRTATTO
RAD
RM
TAX
ZN

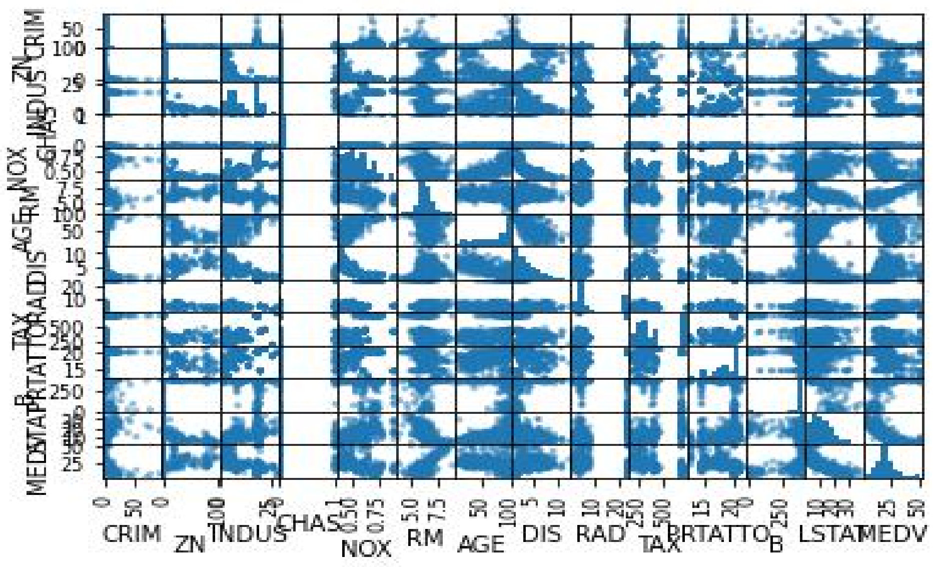
密度图：


Density
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PRTATTO
B
LSTAT
MEDV

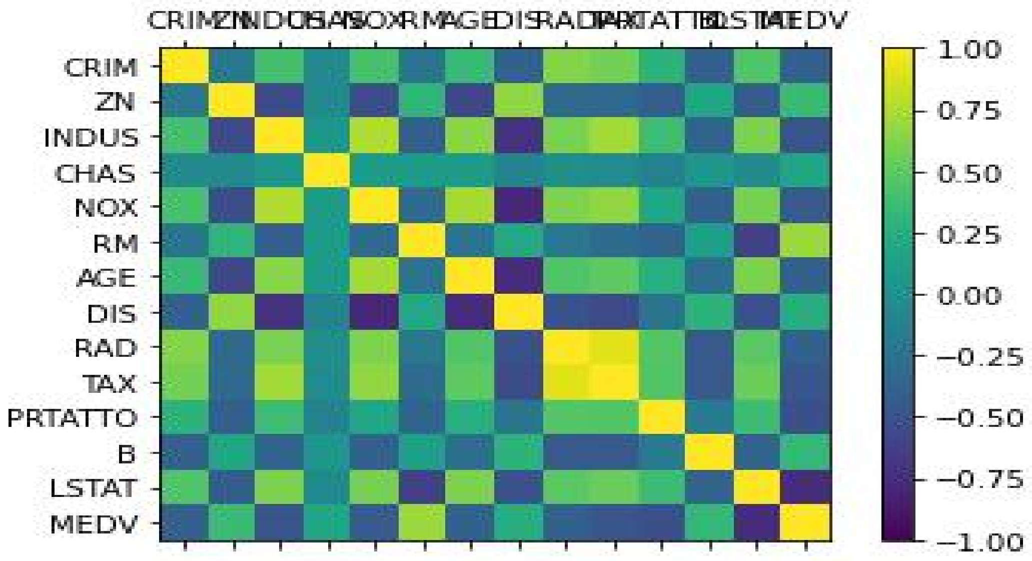
箱线图：


CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PRTATTO
B
LSTAT
MEDV

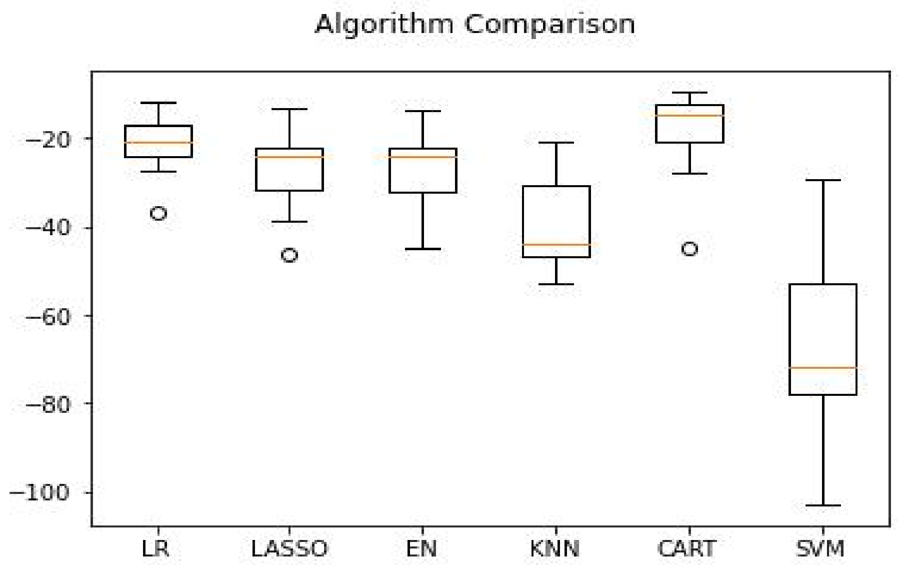
散点矩阵：

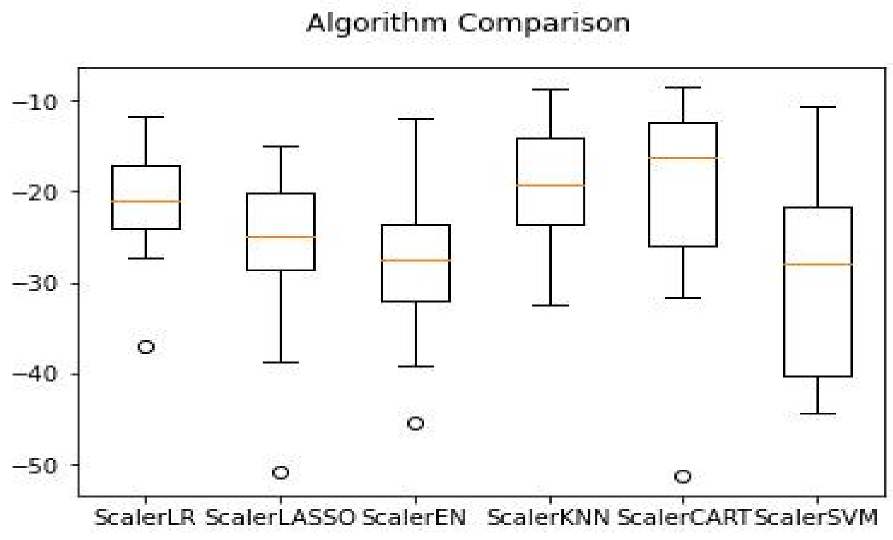

CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PRTATTO
B
LSTAT
MEDV

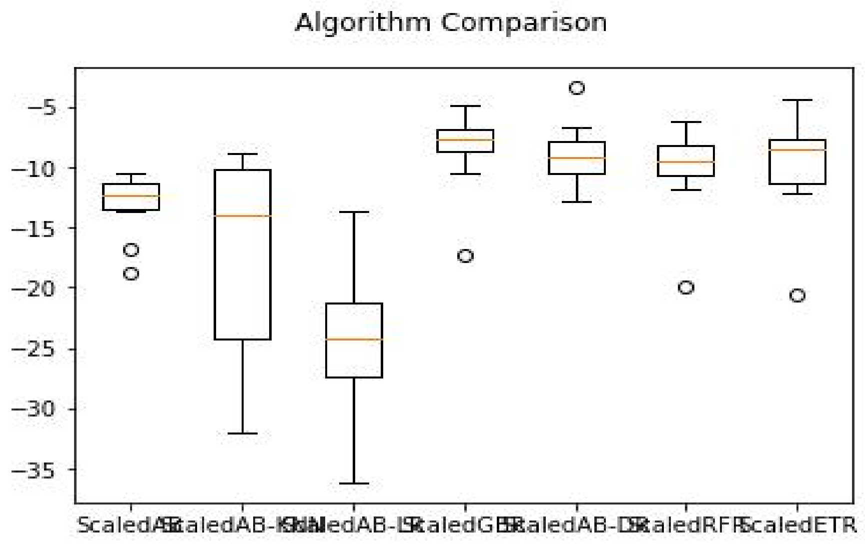
相关系数


CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PRTATTO
B
LSTAT
MEDV
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

算法箱线图:


Algorithm Comparison
−20
−40
−60
−80
−100
LR
LASSO
EN
KNN
CART
SVM


Algorithm Comparison
−10
−20
−30
−40
−50
ScalerLR
ScalerLASSO
ScalerEN
ScalerKNN
ScalerCART
ScalerSVM


Algorithm Comparison
−5
−10
−15
−20
−25
−30
−35
ScaledAB
ScaledAB-KNN
ScaledAB-LR
ScaledGBR
ScaledAB-DR
ScaledRFR
ScaledETR

```
数据类型
(505, 14)
CRIM          float64
ZN           float64
INDUS         float64
CHAS            int64
NOX           float64
RM            float64
AGE           float64
DIS          float64
RAD             int64
TAX             int64
PRTATTO       float64
B            float64
LSTAT         float64
MEDV          float64
dtype: object
       CRIM     ZN   INDUS   CHAS     NOX  ...  TAX  PRTATTO        B  LSTAT
MEDV
0   0.02731    0.0    7.07      0  0.469  ...  242      17.8  396.90   9.14  21.6
1   0.02729    0.0    7.07      0  0.469  ...  242      17.8  392.83   4.03  34.7
2   0.03237    0.0    2.18      0  0.458  ...  222      18.7  394.63   2.94  33.4
3   0.06905    0.0    2.18      0  0.458  ...  222      18.7  396.90   5.33  36.2
4   0.02985    0.0    2.18      0  0.458  ...  222      18.7  394.12   5.21  28.7
5   0.08829   12.5    7.87      0  0.524  ...  311      15.2  395.60  12.43  22.9
6   0.14455   12.5    7.87      0  0.524  ...  311      15.2  396.90  19.15  27.1
7   0.21124   12.5    7.87      0  0.524  ...  311      15.2  386.63  29.93  16.5
8   0.17004   12.5    7.87      0  0.524  ...  311      15.2  386.71  17.10  18.9
9   0.22489   12.5    7.87      0  0.524  ...  311      15.2  392.52  20.45  15.0

数据基本特征:
[10 rows x 14 columns]
DATA decribe:
              CRIM          ZN        INDUS  ...           B        LSTAT
MEDV
count   505.000000  505.000000  505.000000  ...  505.000000  505.000000  505.000000
mean      3.620667   11.350495   11.154257  ...  356.594376   12.668257   22.529901
std       8.608572   23.343704    6.855868  ...   91.367787    7.139950    9.205991
min       0.009060    0.000000    0.460000  ...    0.320000    1.730000    5.000000
25%       0.082210    0.000000    5.190000  ...  375.330000    7.010000   17.000000
50%       0.259150    0.000000    9.690000  ...  391.430000   11.380000   21.200000
75%       3.678220   12.500000   18.100000  ...  396.210000   16.960000   25.000000
```

```
max          88.976200    100.000000      27.740000    ...    396.900000      37.970000      50.000000
```

Pearson 相关性:

```
[8 rows x 14 columns]
Pearson Corr:
                    CRIM             ZN        INDUS    ...             B        LSTAT           MEDV
CRIM          1.000000 -0.200283   0.406251   ... -0.384839   0.455329 -0.388249
ZN          -0.200283   1.000000 -0.534022    ...    0.175319 -0.412894    0.360393
INDUS         0.406251 -0.534022   1.000000   ... -0.356506   0.602737 -0.484126
CHAS        -0.056132 -0.042550    0.062350   ...    0.049040 -0.054576    0.175364
NOX           0.420934 -0.516574   0.764556   ... -0.380006   0.591262 -0.427295
RM           -0.218978    0.311835 -0.391330   ...    0.127754 -0.613734    0.695365
AGE           0.352701 -0.569524   0.645543   ... -0.273486   0.602782 -0.376932
DIS         -0.379627    0.664395 -0.708848   ...    0.291451 -0.497277    0.249896
RAD           0.625396 -0.311717   0.594167   ... -0.444065   0.487608 -0.381690
TAX           0.582568 -0.314351   0.720561   ... -0.441505   0.543435 -0.468543
PRTATTO    0.289394 -0.391713   0.380955   ... -0.176515   0.372148 -0.508411
B           -0.384839    0.175319 -0.356506   ...    1.000000 -0.365637    0.333394
LSTAT        0.455329 -0.412894   0.602737   ... -0.365637   1.000000 -0.738187
MEDV        -0.388249    0.360393 -0.484126   ...    0.333394 -0.738187    1.000000
```

基本算法模型训练结果

```
[14 rows x 14 columns]
               LR: mean: -21.2606        std: 7.1240
            LASSO: mean: -26.8961        std: 9.7794
               EN: mean: -26.6873        std: 9.2284
              KNN: mean: -40.0204        std: 11.0280
             CART: mean: -18.6414        std: 10.2051
              SVM: mean: -66.2907        std: 20.0101
         ScalerLR: mean: -21.2606        std: 7.1240
      ScalerLASSO: mean: -26.7722        std: 10.3717
         ScalerEN: mean: -27.8833        std: 9.4433
        ScalerKNN: mean: -19.1121        std: 7.0713
       ScalerCART: mean: -21.2088        std: 12.3301
        ScalerSVM: mean: -28.8106        std: 11.4886
```

KNN 参数调整:

```
*************************    KNN    *************************
Best KNN Result parameter: -17.956190396341462    Use: {'n_neighbors': 2}
mean: -18.3584 std: 9.351399 with {'n_neighbors': 1}
mean: -17.9562 std: 9.269155 with {'n_neighbors': 2}
mean: -19.0841 std: 8.741352 with {'n_neighbors': 3}
```

```
mean: -19.7569 std: 8.300340 with {'n_neighbors': 4}
mean: -19.3263 std: 7.001439 with {'n_neighbors': 5}
mean: -19.7839 std: 7.156892 with {'n_neighbors': 7}
mean: -19.5728 std: 6.822209 with {'n_neighbors': 9}
mean: -20.0701 std: 6.939799 with {'n_neighbors': 11}
mean: -21.5848 std: 6.844447 with {'n_neighbors': 15}
mean: -22.6353 std: 7.121345 with {'n_neighbors': 17}
mean: -23.6770 std: 7.657120 with {'n_neighbors': 20}
```

Bagging 算法结果输出：

```
*************************    Bagging    *************************
        ScaledAB: mean: -13.1277       std: 2.5656
  ScaledAB-KNN: mean: -17.0797       std: 8.3124
    ScaledAB-LR: mean: -25.0912       std: 6.6084
       ScaledGBR: mean: -8.5292       std: 3.2320
    ScaledAB-DR: mean: -8.9072       std: 2.5000
       ScaledRFR: mean: -10.0995       std: 3.7289
       ScaledETR: mean: -9.9940       std: 4.1852
```

GBM 算法参数调整：

```
*************************    GBM    *************************
Best GBM: -7.892540381567537 Use :{'n_estimators': 300}
```

ET 算法参数调整：

```
*************************    ET    *************************
  Best ET: -9.573458598132616 USe:{'n_estimators': 80}
```

AdaBoost 算法参数调整：

```
*************************    Ada_DecisionTreeRegressor    *************************
  Best Ada: -8.6645 USe:{'n_estimators': 150}
```

```
*************************    Final Training    *************************
Prediction set ET 11.785750247524762
Prediction set GBR 16.170731597788848
Prediction set AdaDR 19.61732673267327
```

可以看出大部分算法的效果都比较符合预期，但是比较奇怪的是 Bagging 的效果比 Boosting 好，推测可能的原因是 boosting 模型比较简单效果不是很好，同时没用做 stacking，另一个比较大的原因可能是数据量比较小, 不能体现 boosting 算法的优势, 虽然经过参数调整 boosting 也能达到和 bagging 查不多的结果，可能是由于 boosting 对模型的参数与离群点比 bagging 更加敏感，所以在数据集上 boosting 并未达到 bagging 的效果。