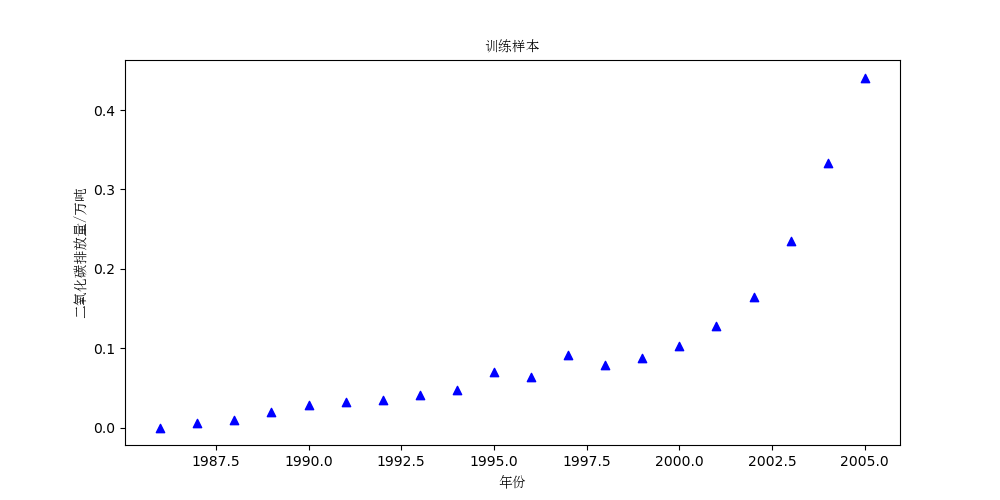
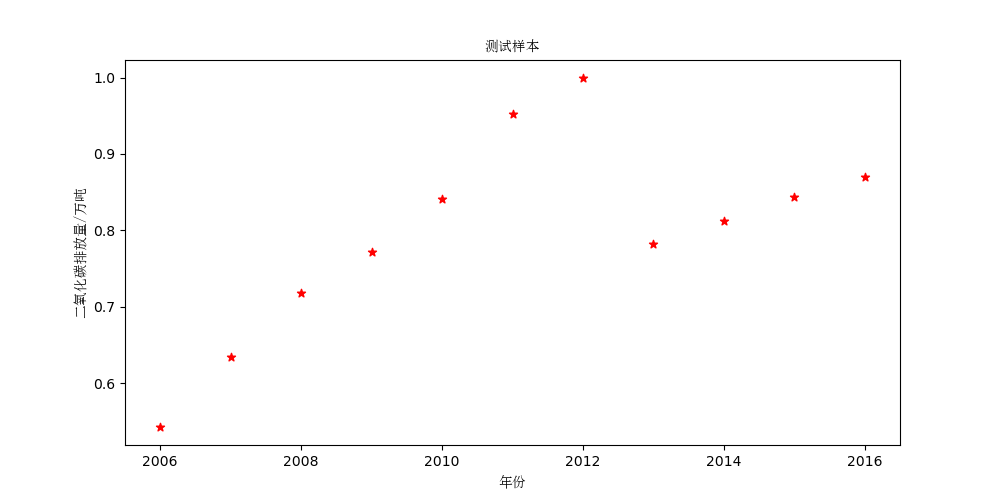
# 第二章

## 2.1

20个训练



11个测试

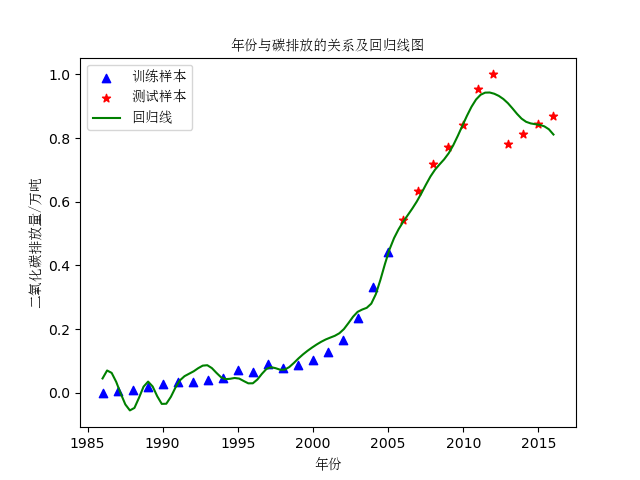


参量：

各个参数的权重：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **经济水平（人均GDP）/万元** | **产业结构（第二产业占比）/%** | **人口规模/万人** | **城镇化率/%** | **能源结构(煤炭占比)/%** | **能源强度（吨标准煤/万元）** |
| 0.24998571 | 0.68491248 | 0.34941828 | 0.09439233 | -0.01114089 | 0.30197353 |

回归线：



## 2.2

参量：

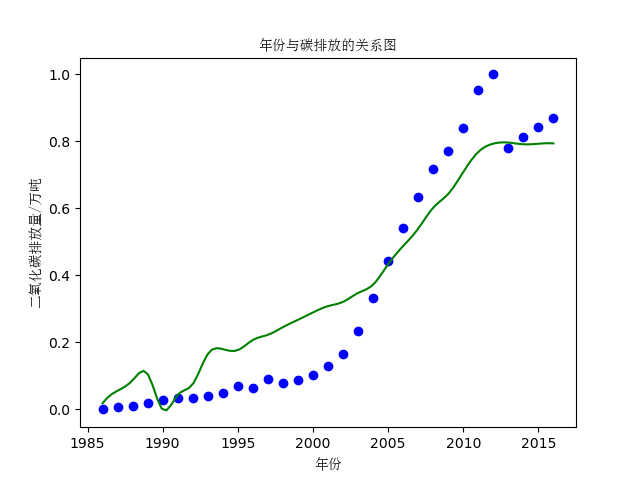
model = SGDRegressor(max\_iter=1000, learning\_rate='constant', eta0=0.01)

获得的参量：

各个参数的权重：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **经济水平（人均GDP）/万元** | **产业结构（第二产业占比）/%** | **人口规模/万人** | **城镇化率/%** | **能源结构(煤炭占比)/%** | **能源强度（吨标准煤/万元）** |
| 0.22142576 | 0.23144907 | 0.15705287 | 0.18906126 | 0.05254334 | -0.1036842 |

截距： [0.02098137]



## 2.3

 model = KernelRidge(alpha=1.0, kernel='linear')

图表

描述已自动生成

# 主要代码

## 数据预处理代码

1. **import** pandas as pd
3. #初始化参数，数据标准化
4. filename = '内蒙古碳排放指标统计数据.xlsx'
5. filename1 = '离差标准化后的数据.xlsx'
6. data = pd.read\_excel(filename,index\_col ="年份（内蒙古）")
7. data = (data-data.min())/(data.max()-data.min())
8. data = data.reset\_index()
9. data.to\_excel(filename1,index = False)

## 基于正则矩阵的线性回归方法

1. **def** Linear():
2. **global** trained\_model
3. X,Y,Z = read\_xlrd()
4. model = Ridge(alpha = 0.000001)#设置线性回归
5. model.fit(X, Y)     # 训练模型
6. trained\_model = model  # 保存训练好的模型
7. a = model.coef\_     #各个参数的权重
8. b = model.intercept\_#截距b的值 Y=a1\*x1+a2\*x2+a3\*x3+b
9. # 输出错误差值平方均值
10. **print**('各个参数的权重：',a)
11. # print('截距：',b)
12. x\_test = X[20:]#测试数据的X数组的值
13. y\_test = Y[20:]#测试数据的Y的值
14. **print**(y\_test)
15. predictions = model.predict(x\_test)#选择后11个数据作为测试数据
16. **for** i, prediction **in** enumerate(predictions):
17. **print**('预测: %s, 结果: %s' % (prediction, y\_test[i]))
18. c = np.mean((model.predict(x\_test) - y\_test) \*\* 2)
19. **print**('错误差值平方均值',c)
20. a = model.score(x\_test,y\_test)
21. **print**('得分：%.2f' %a)#模型得分

## 基于梯度下降的多元线下回归方法

1. **def** gradient\_descent(X, Y, learning\_rate=0.01, iterations=1000):
2. m, n = X.shape
3. theta = np.zeros(n)
4. X = np.concatenate((np.ones((m, 1)), X), axis=1)  # 添加截距项
6. **for** i **in** range(iterations):
7. prediction = np.dot(X, theta)
8. error = prediction - Y
9. gradient = (1/m) \* np.dot(X.T, error)
10. theta -= learning\_rate \* gradient
12. **return** theta
14. **def** SGDLinear():
15. **global** trained\_model
16. X, Y, Z = read\_xlrd()
17. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)
19. # 设置SGDRegressor模型，并给出迭代次数和步长
20. model = SGDRegressor(max\_iter=1000, learning\_rate='constant', eta0=0.01)
21. model.fit(X\_train, Y\_train)  # 训练模型
23. # 使用测试集进行预测
24. Y\_pred = model.predict(X\_test)
26. # 计算MSE和R² 分数
27. mse = mean\_squared\_error(Y\_test, Y\_pred)
28. r2 = r2\_score(Y\_test, Y\_pred)
30. **print**('均方误差（MSE）:', mse)
31. **print**('R² 分数:', r2)
33. trained\_model = model
34. # 获取训练后的参量
35. a = model.coef\_  # 各个特征的权重
36. b = model.intercept\_  # 截距b的值 Y = a1\*x1 + a2\*x2 + ... + an\*xn + b
37. **print**('各个参数的权重：', a)
38. **print**('截距：', b)

## 基于kernel线性回归

1. **def** KernelRidgeRegression():
2. **global** trained\_model
3. X, Y, Z = read\_xlrd()
5. # 使用KernelRidge模型
6. model = KernelRidge(alpha=1.0, kernel='linear')  # 也可以使用其他核，比如'rbf'、'poly'等
7. model.fit(X, Y)
8. trained\_model = model
10. x\_test = X[20:]  # 测试数据的X数组的值
11. y\_test = Y[20:]  # 测试数据的Y的值
12. predictions = model.predict(x\_test)  # 选择后部分数据作为测试数据
14. # 打印预测结果与实际结果的对比
15. **for** i, prediction **in** enumerate(predictions):
16. **print**(f'预测: {prediction}, 结果: {y\_test[i]}')
18. # 计算模型得分
19. score = model.score(x\_test, y\_test)
20. **print**(f'得分：{score:.2f}')  # 模型得分