華東理工大學

模式识别大作业

题　　目　NBA 球员薪水预测

学　　院　信息科学与工程

专　　业　控制科学与工程

组　　员　姜唯亮、顾泽权

指导教师　赵海涛

完成日期：　2019 年 12 月 3 日

NBA 球员薪水预测

组员：姜唯亮、顾泽权

在赵老师的教导下，通过对模式识别的学习，我们已经对模式识别和机器学习的理论方面有了一定的理解，但是“纸上得来终觉浅，绝知此事要躬行”，因此这次的大作业正好是锻炼我们理论应用于实践的好机会。

我们选择了一个应用多元线性回归进行预测的题目，下面将详细说明我们的解决过程。

一、NBA 球员薪水预测问题简介

NBA 球员工薪一般指 NBA 工资帽（Salary cap），NBA 工资帽是 NBA 最著名工资限制条款。球队花在球员身上的工资总额不得超过这个数字。工资帽的概念 1984 年被引入，NBA 联盟和球员工会之间首先进行谈判，只有双方同意，工资帽才被正式公布。每年 NBA 的工资帽都在增加，1997-98 赛季是 2690 万美元，2010 年是 5800 万美元。2013-2014 赛季又递增到了 5867.9 万美元。

与足球联赛不同，NBA 是一个没有升降级的封闭联赛，所以他们必须有自己的规定来保持整个体系的“生态平衡”，让没钱的球队也有自己的生存空间，避免发生一个或者数个豪门包揽冠军的情况发生。总之一句话就是，在 NBA，花钱也要受到各种限制。为了限制花钱，NBA 引入了工资帽和奢侈税两个概念。如果没有工资帽，财力更好的球队就会比其他队更容易地获得更好的自由球员。基本思想是如果球队的总工资单没有超出工资帽，球队只能签一位自由球员。如果哪支超出工资帽就要向联盟交纳 NBA 奢侈税。因此所有队都在同一起跑线上。

基于以上所述，每支球队对球员的工薪预测就至关重要。所要解决的问题是使用一个包含有关 2017-2018 赛季 NBA 职业球员的若干相关信息的数据集，利用多元线性回归来预测他们的薪水。

二、数据描述统计

2.1 数据结构分析

数据从 Kaggle 上下载，包括训练数据 train.csv 和测试数据 test.csv。数据的结构如下所示：

表 2.1 原始数据结构

Age	MP	PER	…	BLK	PTS	Salary

28	385	5.9	…	5	88	4666500
36	128	12.8	…	2	33	2328652
22	2129	12.8	…	30	1040	2422560
27	2809	25.1	…	32	1816	19301070
32	1700	10.9	…	22	538	2328652
21	1190	8.5	…	10	445	1386600

其中训练数据一共有 318 组，测试数据一共有 70 组，表中各列表示的意思如下：

(1) Age-相关球员的年龄

(2) MP-播放分钟数：整个赛季播放的分钟数

(3) PER-球员效率等级：球员的加权贡献

(4) TS%-真实投篮命中率：总计 3 分，2 分和罚球命中率

(5) FG-投篮得分：成功投篮次数

(6) FGA-尝试投篮得分：尝试射投篮次数

(7) FG% -投篮得分百分比：成功投篮的百分比

(8) 3P-3 分：成功获得 3 分的投篮数量

(9) 3PA-尝试 3 分：尝试投 3 分的投篮数量

(10) 3P%-3 分百分比：投 3 分成功的百分比

(11) eFG%-有效投篮命中率：基于 3P 和 2P 的成功投篮命中率

(12) FT%-罚球率：成功罚球的百分比（1 分）

(13) ORB-进攻篮板：进攻篮板数

(14) DRB-防守篮板：防守篮板数

(15) AST-助攻：本赛季计算的助攻数

(16) STL-抢断：本赛季统计的抢断次数

(17) BLK-方块：本赛季计算的方块数

(18) PTS-点数：本赛季计算的点数

(19) Salary-球员的薪水

2.2 数据预处理

在对数据进行分析前，首先要将数据进行清理。通过观察数据可以发现，训练数据集中最严重的问题就是有些球员的个别指标为空值，因此首先将指标存在空值的球员选择出来，然后再对缺失的数据进行填充。

```
import pandas as pd
data_train = pd.read_csv(r'train.csv')
empty_index = data_train.index[data_train.isnull().sum(axis=1) > 0]
data = data_train.iloc[empty_index, :]
```

```
print(data)
```

通过如上 Python 代码，得到缺失数据的球员信息：

表 2.2 缺失数据的球员信息

Age	…	3P%	eFG%	FT%	…	Salary
29	…	0	0.21	0.556	…	
27	…		0.62	0.699	…	6000000
28	…		0.25	1	…	1865547
34	…		0.67	0.734	…	13000000
19	…		0.23	1	…	1312611
27	…		0.56	0.628	…	23775506
23	…		0.44	0	…	50000
20	…		0.51	0.556	…	1312611
22	…		0.64	0.531	…	2334520
30	…		0.51	0.84	…	4328000
20	…		0.71	1	…	1312611
21	…		0.58	0.544	…	2825640
25	…		0.44	0.724	…	172552
28	…		0.55	0.81	…	7000000
30	…		0.48	0.59	…	10595505
27	…		0.53	0.617	…	6352531
24	…		0.53	0.534	…	17000000
21	…	0.333	0.44		…	2451225
30	…	0.444	0.5		…	510921
24	…	0.25	0.61		…	25000

从上表可以看出有 1 个缺失 Salary 数据的球员，3 个缺失 FT%数据的球员以及 16 个缺失 3P%数据的球员。对于缺失值的处理主要有以下几种方法：

(1) 删除元组。

将存在遗漏信息属性值的对象（记录）删除，从而得到一个完备的信息表。这种方法在对象有多个属性缺失值、被删除的含缺失值的对象与信息表中的数据量相比非常小的情况下是非常有效的。然而这种方法丢弃了大量隐藏在这些对象中的信息。在信息表中对象很少的情况下会影响到结果的正确性，可能导致数据发生偏离，从而引出错误的结论。

(2) 人工填写

这个方法产生数据偏离最小，是填充效果最好的一种。当数据规模很大、空值很多的时候，该方法是不可行的。

(3) 特殊值补充

将空值作为一种特殊的属性值来处理，它不同于其他的任何属性值。如所有的空值都用“unknown”填充。这样将形成另一个概念，可能导致严重的数据偏离，一般不使用。

(4) 平均值填充

因为空值是数值属性，就可以使用该属性在其他所有对象的取值的平均值来填充该缺失的属性值。

(5) 热卡填充（就近补齐）

对于一个包含空值的对象，热卡填充法在完整数据中找到一个与它最相似的对象，然后用这个相似对象的值来进行填充。

(6) K 最近邻法

先根据欧式距离或相关分析来确定距离具有缺失数据样本最近的 K 个样本，将这 K 个值加权平均来估计该样本的缺失数。

考虑到本题的实际背景，选取平均值填充方法，利用 Python 进行编码填充：

```
import pandas as pd
data_train = pd.read_csv(r'train.csv')
empty_index = data_train.index[data_train.isnull().sum(axis=1) > 0]
for col in list(data_train.columns[data_train.isnull().sum() > 0]):
    mean_val = data_train[col].mean()
    data_train[col] = data_train[col].fillna(mean_val)
data = data_train.iloc[empty_index, :]
print(data)
```

填充后的数据如下表所示：

表 2.3 数据补全后的球员信息

Age	…	3P%	eFG%	FT%	…	Salary
29	…	0	0.21	0.556	…	7726388
27	…	0.312937	0.62	0.699	…	6000000
28	…	0.312937	0.25	1	…	1865547
34	…	0.312937	0.67	0.734	…	13000000
19	…	0.312937	0.23	1	…	1312611
27	…	0.312937	0.56	0.628	…	23775506
23	…	0.312937	0.44	0	…	50000
20	…	0.312937	0.51	0.556	…	1312611
22	…	0.312937	0.64	0.531	…	2334520
30	…	0.312937	0.51	0.84	…	4328000
20	…	0.312937	0.71	1	…	1312611

21	...	0.312937	0.58	0.544	...	2825640
25	...	0.312937	0.44	0.724	...	172552
28	...	0.312937	0.55	0.81	...	7000000
30	...	0.312937	0.48	0.59	...	10595505
27	...	0.312937	0.53	0.617	...	6352531
24	...	0.312937	0.53	0.534	...	17000000
21	...	0.333	0.44	0.751784	...	2451225
30	...	0.444	0.5	0.751784	...	510921
24	...	0.25	0.61	0.751784	...	25000

2.3 数据可视化

缺失值补充后，需要对定量的属性进行分析。定量的属性（数值属性）通常蕴涵着可排序性，例如本题中的各个属性都属于定量属性。下面首先分析各个属性与薪水之间的相关系数，从中挑选出相关系数较高的属性，再通过绘图的方式展现出其与薪水之间的关系。

```
import pandas as pd
data_train = pd.read_csv(r'train.csv')
for col in list(data_train.columns[data_train.isnull().sum() > 0]):
    mean_val = data_train[col].mean()
    data_train[col] = data_train[col].fillna(mean_val)
data_corr = data_train.corr()
print(data_corr.iloc[:-1, -1])
```

利用上述代码可以得到各属性与薪水之间的相关系数：

表 2.4 各属性与薪水的相关系数

Age	MP	PER	TS%	FG	FGA	FG%	3P	3PA
0.230819	0.665308	0.578053	0.264458	0.736031	0.712471	0.201510	0.492910	0.484009
3P%	eFG%	FT%	ORB	DRB	AST	STL	BLK	PTS
0.054517	0.206771	0.115375	0.419331	0.626041	0.568807	0.579351	0.410430	0.737645

从上表可以看出，FG、FGA、PTS 与薪水的相关系数都在 0.7 以上，说明这几个指标与薪水有较强的相关性，因此选取这三个指标来描绘其与薪水之间的关系。

```
import pandas as pd
import matplotlib.pyplot as plt
data_train = pd.read_csv(r'train.csv')
```

```
for col in list(data_train.columns[data_train.isnull().sum() > 0]):
        mean_val = data_train[col].mean()
        data_train[col] = data_train[col].fillna(mean_val)

data_salary = data_train['Salary']
data_FG = data_train['FG']
data_FGA = data_train['FGA']
data_PTS = data_train['PTS']
plt.figure(1)
plt.scatter(data_FG, data_salary, c='r')
plt.xlabel('FG')
plt.ylabel('Salary')
plt.figure(2)
plt.scatter(data_FGA, data_salary, c='b')
plt.xlabel('FGA')
plt.ylabel('Salary')
plt.figure(3)
plt.scatter(data_PTS, data_salary, c='g')
plt.xlabel('PTS')
plt.ylabel('Salary')
plt.show()
```

通过以上代码可以获得 FG、FGA、PTS 与薪水的散布图：

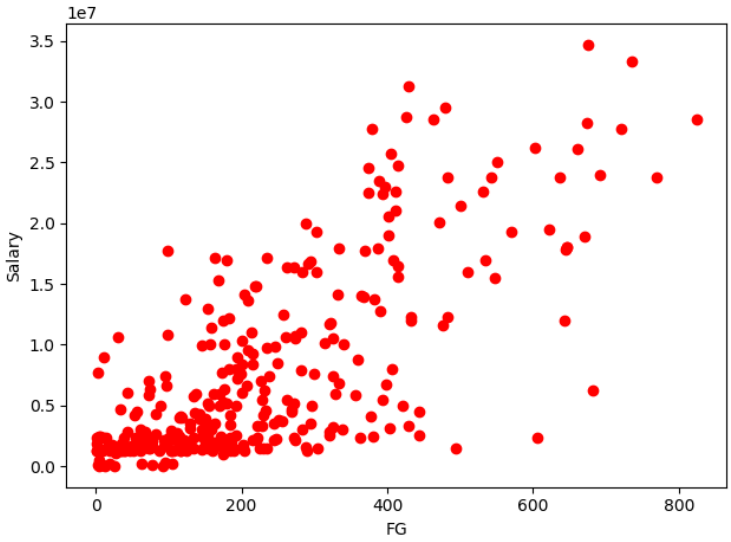

图 2.1 FG 与薪水散布图

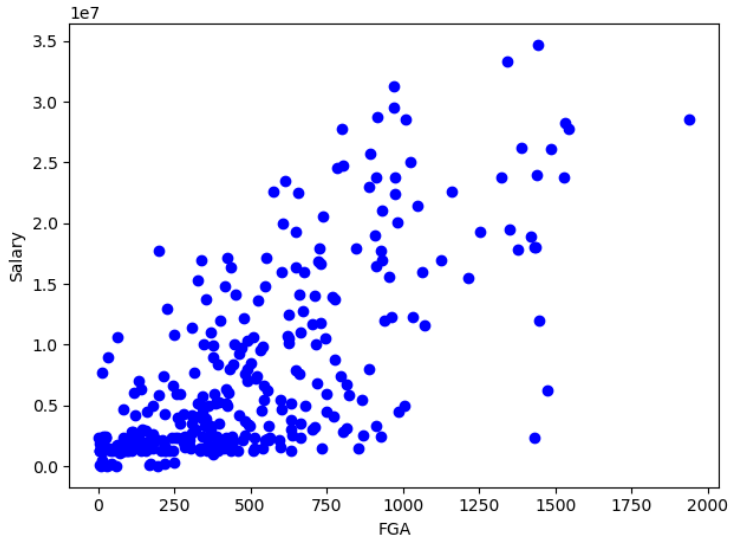

图 2.2 FGA 与薪水散布图

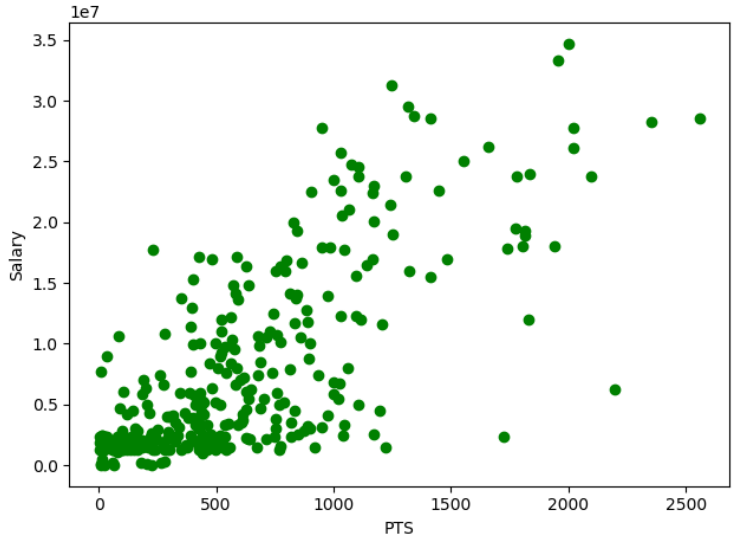

图 2.3 PTS 与薪水散布图

从图 2.1、图 2.2 和图 2.3 可以看出，除了个别几位球员外，大部分球员的 FG、FGA 和 PTS 指标与薪水成正比关系，即薪水随着其增加而增加，因此得到一个解决本题预测问题的一个思路：利用多元线性回归算法进行预测。

三、问题的求解与分析

3.1 多元线性回归

多元线性回归预测是一元线性回归预测的直接推广，其包含一个因变量和二个或二个以上的自变量。多元线性回归与一元线性回归类似，可以用最小二乘法估计模型参数。

回归分析实质上是一个数学过程:其通过变量之间的数学表达式来定量地描

述变量之间的相关性,同时通过此方程式用基于自变量的值来预测因变量的值。其数学模型为:

$$\begin{Bmatrix} y_1 = \beta_0 + \boldsymbol{x}_1\boldsymbol{\beta} + \varepsilon_1 \\ y_2 = \beta_0 + \boldsymbol{x}_2\boldsymbol{\beta} + \varepsilon_2 \\ \vdots \\ y_n = \beta_0 + \boldsymbol{x}_n\boldsymbol{\beta} + \varepsilon_n \end{Bmatrix}$$

令$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$, $X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \ddots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$, $E = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$, $B = \begin{bmatrix} \beta_0 \\ \boldsymbol{\beta} \end{bmatrix}$,则有

$$Y = XB + E$$

$$J(B) = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - (\sum x_{ij}\beta_j + \beta_0)\right]^2 = \frac{1}{n}(Y - XB)^T(Y - XB)$$

在本题中，自变量分别取训练数据集中的各列指标，因变量取球员薪水。

求解$J(B)$的最小值有几种方法，下面用求导数的方式来求解，即用梯度下降法来求解。

$$J(B) = \frac{1}{n}(Y - XB)^T(Y - XB)$$

$$= \frac{1}{n}(Y^TY - Y^TXB - B^TX^TY + B^TX^TXB)$$

$$= \frac{1}{n}(Y^TY - 2B^TX^TY + B^TX^TXB)$$

$$\nabla J(B) = \frac{2}{n}(-X^TY + X^TXB)$$

因此线性回归问题可用梯度下降法来解。下面给出梯度下降法进行求解线性回归问题的一般步骤：

①给定初值B_1，$i = 1$，学习率α，给定阈值ε；；

②求$B_{i+1} = B_i - \alpha\nabla J(B_i)$ ；

③若$\| J(B_{i+1}) - J(B_i) \|_2^2 \leq \varepsilon$，转向④，否则，$i = i + 1$，转向②

④输出B_{i+1}.

其中

$$\Delta B = -\nabla J(B) = X^TY - X^TXB$$

也可以使用牛顿法来求解，即$\Delta B = -\left(\nabla^2 J(B)\right)^{-1}\nabla J(B) = -(X^TX)^{-1}(-X^TY + X^TXB)$，则$B^* = B + \Delta B = B - B + (X^TX)^{-1}X^TY = (X^TX)^{-1}X^TY$。

多元性回归模型与一元线性回归模型一样，在得到参数的最小二乘法的估计值之后，也需要进行必要的检验与评价，以决定模型预测的好坏。根据题目的要求，利用均方根误差（RMSE）对预测结果进行评判。

均方根误差，即因变量 y 的实际值与回归方程求出的估计值之间的标准误差，均方根误差越小，回归方程拟合程度越好。

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(pred_i - y_i)^2}{n}}$$

3.2 实验与结果分析

3.2.1 梯度下降法

利用前文所述的梯度下降法对 NBA 球员的薪水进行预测。编写代码时发现，算法中不同的参数初值会导致不同的结果；步长的选取也至关重要，步长选取的不合理可能会导致算法发散；不同的阈值会影响算法的效率。因此在多次尝试后，选取参数的初值为$[1,1,\ldots,1]$，步长为10^{-9}，阈值为10^{5}，代码如下：

```
import numpy as np
import pandas as pd
def main(df_train, df_predict):
    beta = 1e-9
    x_train = df_train.iloc[:, :18]
    y_train = df_train.iloc[:, -1]
    b = np.ones([x_train.shape[1] + 1, 1])
    x = np.hstack((np.ones((x_train.shape[0], 1)), x_train))
    J_pre = np.dot((y_train - np.squeeze(np.dot(x, b))).T, (y_train - np.squeeze(np.dot(x, b)))) / len(y_train)
    diff = np.dot(x.T, y_train) - np.squeeze(np.dot(np.dot(x.T, x), b))
    b = np.squeeze(b) + beta * diff
    J_af = np.dot((y_train - np.squeeze(np.dot(x, b))).T, (y_train - np.squeeze(np.dot(x, b)))) / len(y_train)
    while np.sqrt(np.sum(np.square(J_af - J_pre))) >= 1e+5:
        J_pre = J_af
        diff = np.dot(x.T, y_train) - np.squeeze(np.dot(np.dot(x.T, x), b))
        b = np.squeeze(b) + beta * diff
        J_af = np.dot((y_train - np.squeeze(np.dot(x, b))).T, (y_train - np.squeeze(np.dot(x, b)))) / len(y_train)
    x_predict = df_predict.iloc[:, 1:]
    x_ = np.hstack((np.ones((x_predict.shape[0], 1)), x_predict))
```

```
    y_predict = np.dot(x_, b)
    rmse = np.sqrt(np.sum(np.square(np.dot(x, b) - y_train)) / len(y_train))
    return y_predict, rmse, b


if __name__ == '__main__':
    data_train = pd.read_csv(r'train.csv')
    for col in list(data_train.columns[data_train.isnull().sum() > 0]):
        mean_val = data_train[col].mean()
        data_train[col] = data_train[col].fillna(mean_val)
    data_predict = pd.read_csv(r'test.csv')
    y_predict, rmse, b = main(data_train, data_predict)
    dataframe = pd.DataFrame(y_predict, columns=["prediction"])
    dataframe.to_csv(r"gradient_descent_prediction.csv")
    print("RMSE：{}，系数：{}".format(rmse, b))
```

上述代码会得到一个薪水预测结果文档，并且输出相应的均方根误差和系数。得到的均方根误差为 4846990.013671898，其值偏大，是因为薪水本身数值就很大，加上一共有 18 个指标，并不是每个指标都与薪水存在类线性关系。薪水与各个指标的关系由下面的式子给出：

$$\begin{aligned} y = &-18505.62 + 100838.12x_1 - 2030.93x_2 - 158474.96x_3 - 13635.68x_4 \\ &+ 7479.40x_5 - 8029.75x_6 - 10889.46x_7 + 153653.11x_8 \\ &- 57451.92x_9 - 16471.03x_{10} - 13004.65x_{11} - 17867.54x_{12} \\ &+ 8179.92x_{13} + 11166.19x_{14} + 8169.33x_{15} + 18342.90x_{16} \\ &+ 10664.73x_{17} + 13008.70x_{18} \end{aligned}$$

下表列出了部分预测结果：

表 2.5 梯度下降法预测结果

Id	Age	MP	PER	…	BLK	PTS	Salary
1	30	2295	13.7	…	53	849	10252319.69
2	28	1885	21.1	…	22	1142	15333611.88
3	25	1186	15.8	…	40	476	6644218.346
4	23	1426	11.5	…	20	419	5723899.029
5	24	2803	16.3	…	15	1518	13744213.1
6	29	1667	10.8	…	15	662	6916781.215

3.2.2 牛顿法

利用牛顿法对球员进行预测，其运行效率远远大于梯度下降法。相应的 Python 代码如下：

```
import numpy as np
import pandas as pd
def main(df_train, df_predict):
    x_train = df_train.iloc[:, :18]
    y_train = df_train.iloc[:, -1]
    x = np.hstack((np.ones((x_train.shape[0], 1)), x_train))
    b = np.dot(np.linalg.inv(np.dot(x.T, x)), np.dot(x.T, y_train))
    x_predict = df_predict.iloc[:, 1:]
    x_ = np.hstack((np.ones((x_predict.shape[0], 1)), x_predict))
    y_predict = np.dot(x_, b)
    rmse = np.sqrt(np.sum(np.square(np.dot(x, b) - y_train)) / len(y_train))
    return rmse, y_predict, b

if __name__ == '__main__':
    data_train = pd.read_csv(r'train.csv')
    for col in list(data_train.columns[data_train.isnull().sum() > 0]):
        mean_val = data_train[col].mean()
        data_train[col] = data_train[col].fillna(mean_val)
    data_predict = pd.read_csv(r'test.csv')
    rmse, y_predict, b = main(data_train, data_predict)
    dataframe = pd.DataFrame(y_predict, columns=["prediction"])
    dataframe.to_csv(r"newton_prediction.csv")
    print("RMSE：{}，系数：{}".format(rmse, b))
```

得到的均方根误差为 4740932.19885361。球员薪水与各个指标的关系由下面的式子给出：

$$y = 2.45 \times 10^6 + 2.30 \times 10^5 x_1 + 89.08 x_2 + 1.14 \times 10^5 x_3 - 2.84 \times 10^6 x_4 + 8.46 \times 10^3 x_5 - 1.04 \times 10^4 x_6 - 1.78 \times 10^6 x_7 + 1.53 \times 10^5 x_8 - 5.46 \times 10^4 x_9 - 5.20 \times 10^6 x_{10} - 7.22 \times 10^6 x_{11} - 2.78 \times 10^6 x_{12} - 1.13 \times 10^3 x_{13} + 9.47 \times 10^3 x_{14} + 4.60 \times 10^3 x_{15} + 1.04 \times 10^4 x_{16} + 1.04 \times 10^4 x_{17} + 1.17 \times 10^4 x_{18}$$

下表列出了部分预测结果：

表 2.6 牛顿法预测结果

Id	Age	MP	PER	…	BLK	PTS	Salary
1	30	2295	13.7	…	53	849	11162253.37
2	28	1885	21.1	…	22	1142	15640808.83

3	25	1186	15.8	…	40	476	6955180.348
4	23	1426	11.5	…	20	419	4849383.926
5	24	2803	16.3	…	15	1518	13511004.54
6	29	1667	10.8	…	15	662	7133527.325

3.2.3 Sklearn 库

Python 中的 Sklearn 库自带有可以进行多元线性回归分析的函数，利用该库也可以对球员的薪水进行预测。Sklearn 库中的 sklearn.linear_model.LinearRegression 类是一个估计器，该估计器依据观测值来预测结果。库中所有的估计器都带有 fit()和 predict()方法，fit()用来分析模型参数，predict()是通过 fit()算出的模型参数构成的模型，对解释变量进行预测获得的值。相应的 Python 代码如下：

```
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
def main(df_train, df_predict):
    x_train = df_train.iloc[:, :18]
    y_train = df_train.iloc[:, -1]
    linear_re = LinearRegression()
    linear_re.fit(x_train, y_train)
    x_predict = df_predict.iloc[:, 1:]
    y_predict = linear_re.predict(x_predict)
    rmse = np.sqrt(np.sum(np.square(linear_re.predict(x_train) - y_train)) / len(y_train))
    return y_predict, rmse, linear_re.coef_
if __name__ == '__main__':
    data_train = pd.read_csv(r'train.csv')
    for col in list(data_train.columns[data_train.isnull().sum() > 0]):
        mean_val = data_train[col].mean()
        data_train[col] = data_train[col].fillna(mean_val)
    data_predict = pd.read_csv(r'test.csv')
    y_predict, rmse, intercept = main(data_train, data_predict)
    dataframe = pd.DataFrame(y_predict, columns=["prediction"])
    dataframe.to_csv(r"sklearn_prediction.csv")
    print("RMSE：{}，系数：{}".format(rmse, intercept))
```

得到的均方根误差为 4740932.19885361。球员薪水与各个指标的关系由下面

的式子给出：

$$y = 2.45 \times 10^6 + 2.30 \times 10^5 x_1 + 89.08 x_2 + 1.14 \times 10^5 x_3 - 2.84 \times 10^6 x_4 + 8.46 \times 10^3 x_5 - 1.04 \times 10^4 x_6 - 1.78 \times 10^6 x_7 + 1.53 \times 10^5 x_8 - 5.46 \times 10^4 x_9 - 5.20 \times 10^6 x_{10} - 7.22 \times 10^6 x_{11} - 2.78 \times 10^6 x_{12} - 1.13 \times 10^3 x_{13} + 9.47 \times 10^3 x_{14} + 4.60 \times 10^3 x_{15} + 1.04 \times 10^4 x_{16} + 1.04 \times 10^4 x_{17} + 1.17 \times 10^4 x_{18}$$

下表列出了部分预测结果：

表 2.7 Sklearn 预测结果

Id	Age	MP	PER	…	BLK	PTS	Salary
1	30	2295	13.7	…	53	849	11162253.37
2	28	1885	21.1	…	22	1142	15640808.83
3	25	1186	15.8	…	40	476	6955180.348
4	23	1426	11.5	…	20	419	4849383.926
5	24	2803	16.3	…	15	1518	13511004.54
6	29	1667	10.8	…	15	662	7133527.325

3.2.4 结果分析

对比上面的三种求解方法，发现利用牛顿法和 Sklearn 库得到的结果精确度更高。实际上，如果梯度下降法中的步长取最优步长，并且阈值取更小的值，那么最终梯度下降法的均方根误差将会收敛到牛顿法和 Sklearn 库得到均方根误差。虽然牛顿法的效率很高，但是其运算过程中用到了矩阵的求逆运算，在本题中并没有太大问题，但是当数据量很庞大的时候，求逆就会变得非常复杂，牛顿法不一定能得到很好的结果。因此，当数据量较大时，梯度下降法和 Sklearn 库的使用更能确保得到精确的结果。

四、总结与展望

对于多元线性回归中的梯度下降法，在选取步长时，可以进一步采用一维搜索算法选取最佳步长，以使得算法收敛得更快。多元线性回归归根结底还是线性回归，其是一种较为简单的回归算法，因此得到的效果比较有限。

事实上，还有很多算法可以对数据进行预测，例如循环神经网络等。未来可以针对更庞大的数据，选取更有效、更准确的算法进行预测。

五、小组分工

资料查找及初步报告：顾泽权

程序设计、编写及调试，最终报告编写、修改：姜唯亮