数据库概论综合实习 3 报告

罗昊 1700010686　常辰 1500012710　易士程 1500012789

任务一 篮球运动员数据分析

导入数据

在数据分析之前，需要将 csv 文件导入数据库，实习中采用的是建表+插入操作，以 player_career.csv 为例，代码如下：

```
create table player_career(ilkid nvarchar(20),firstname nvarchar(20), lastname nvarchar(20),leag nvarchar(20),gp int,minutes int,pts int,dreb int,oreb int,reb int,arbs int,stl int,tlk int,turnover int,pf int,fga int,fgm int,fta int,ptm int,tpa int,tpm int)
bulk insert players_playoffs_career
from '/usr/local/spark/basketball_dataset/player_playoffs_career.csv'
WITH(FIELDTERMINATOR='',ROWTERMINATOR='\n', FORMAT='CSV')
```

数据分析

在实验中使用 python 中的 sklearn 的一些方法对数据进行了分析。在数据读入的部分，可以选择使用 python 的 MySQLdb 包来获取数据库数据。在实验中，对 player _career.csv 做了线性回归分析，PCA+LDA 以及对应的聚类分析，具体如下。

1、线性回归分析

我们选择队员的得分（pts）作为线性回归的预测值，使用除比赛场数、上场时间以及得分之外的其他信息作为自变量，具体的代码如下：

```
from sklearn.linear_model import  LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import linear_model,datasets,metrics
y = num[:,2]
x = num[:,3:]

X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.1,random_state=1)

LR = LinearRegression()
# 对训练数据进行拟合训练
LR.fit(X_train, y_train)
```

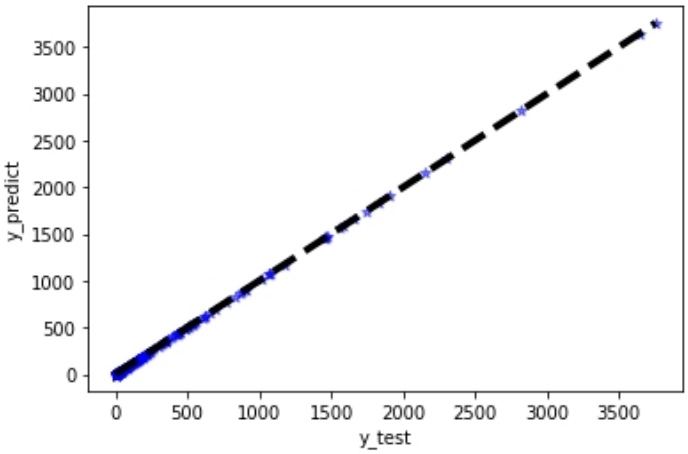
实验中使用 10%的数据作为训练集，以 MSE 和 RMSE 作为数据的评估指标。结果对应的线性回归的参数以及评估指标如下：

```
LR.intercept:
 -2.8421709430404007e-13
LR.coef:
 [-3.75348104e-16  8.62187190e-16 -7.94579101e-16  2.21880540e-15
 -3.34085914e-15 -3.72850165e-15 -1.57400434e-15 -6.69989701e-16
  1.13521021e-16  2.00000000e+00 -1.05946954e-16  1.00000000e+00
  5.05278723e-16  1.00000000e+00]
MSE: 2.529005583001078e-25
RMSE: 5.028921935167693e-13
```

其中，intercept 表示截距，coef 表示权重参数。可以看到，在线性回归中大部分参数都约等于 0，只有三个参数是明显不接近 0 的，线性回归拟合所表达的逻辑含义如下：

$$\text{得分} = \text{投篮命中数} \times 2 + \text{罚篮命中数} + \text{三分球命中数}$$

显然，这个公式是符合实际情况的。MSE 以及 RMSE 都约等于 0，表示该线性回归模型拟合测试集，以测试集预测值为 x 轴，以实际的得分为 y 轴画出图像如下：



从图像也能看出，线性回归模型具有较高的准确性。

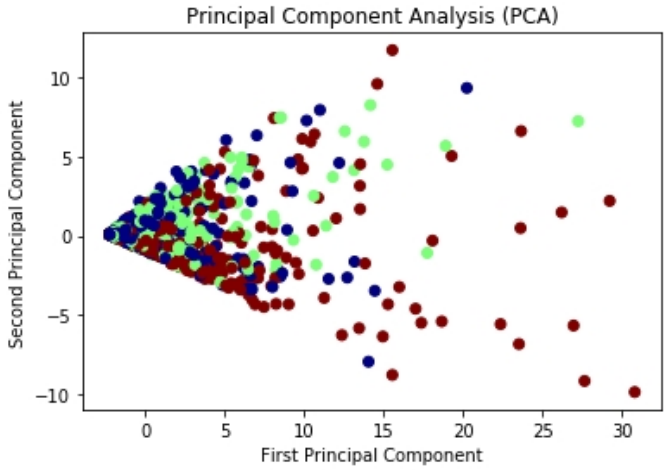
2、 PCA+聚类

PCA(Principal Component Analysis)，即主成分分析方法，是一种使用最广泛的数据降维算法。PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。

在实验中我们做了一个预处理，以$\frac{\text{队员得分}}{\text{上场时间}+1}$作为得分能力标签，得分能力在前 25%标记为 2，在 25-50%之间标记为 1，在后 50%标记为 0。同时在实验过程中我们去掉了出厂时间小于 5 分钟的数据（有一些数据项有问题），PCA 的具体实现代码如下：

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
X = StandardScaler().fit_transform(num)

Target_2d = tmp
delete_list = []
for i in range(len(num)):
    if num[i][1] <= 5:
        delete_list.append(i)

Target_2d = np.delete(Target_2d, delete_list, axis = 0)
X = np.delete(X, delete_list, axis = 0)

n_components = 2
pca = PCA(n_components=2)
pca.fit(X)
X_2d = pca.transform(X)
```

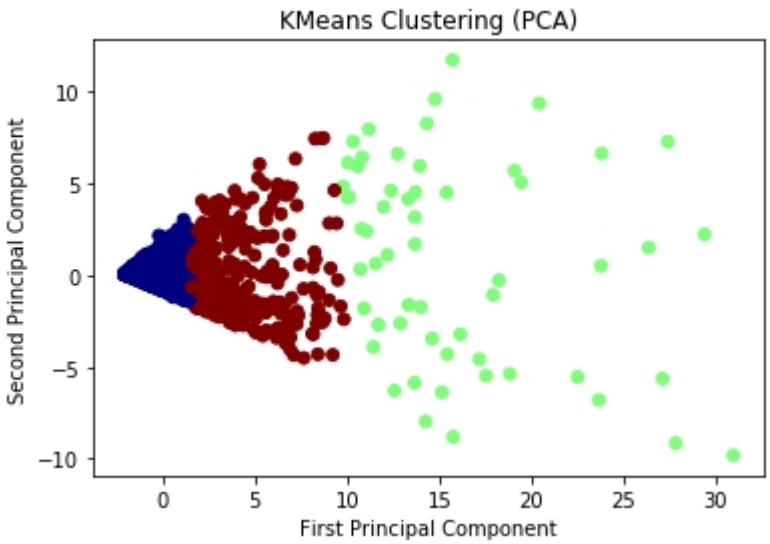
即将原数据降到 2 维，在画图过程中我们对应画出的图像如下：



图像中的蓝、绿、红三种颜色的点分别对应了 0,1,2 三种标签。PCA 的结果表明，降维的结果并没有对应到得分能力上，我们在实验中也使用了得分、上场时间等不同标签，均未在 PCA 中看到明显类别分割，故我们得到的结论是：PCA 将维到 2 维后，横纵坐标表达的是原来一些项的组合含义。

由于该图像仅仅只有一个密度较大的簇和很多散点，故只简单使用了 K-means 对其进行聚类（3 类）的代码以及结果图如下：

```
#PCA+K-means
from sklearn.cluster import KMeans # KMeans clustering

kmeans = KMeans(n_clusters=3)

X_clustered = kmeans.fit_predict(X_2d)
```



其聚类结果代表了某种能力的强弱，但其在逻辑上可解释性较低。

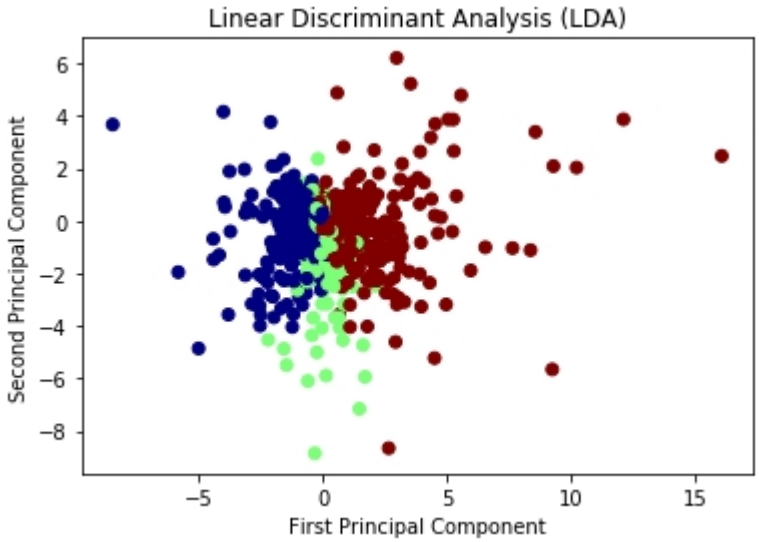
3、 LDA+聚类

LDA 是一种监督学习的降维技术，也就是说它的数据集的每个样本是有类别输出的。这点和 PCA 不同。PCA 是不考虑样本类别输出的无监督降维技术。LDA 的思想可以用一句话概括，就是“投影后类内方差最小，类间方差最大”。

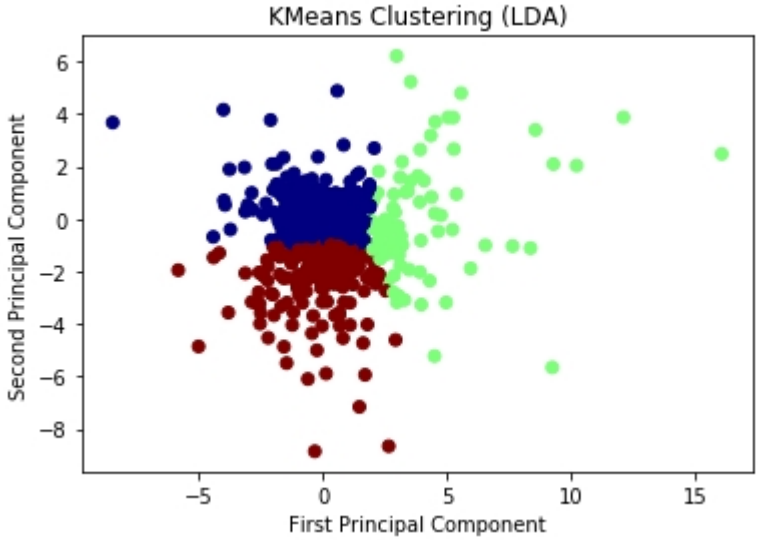
同样我们做了一个与 PCA 相同的预处理，LDA 的具体代码实现如下：

```
#使用LDA有监督降维到2维，绘制散点图。使用kmeans进行聚类。
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
X = StandardScaler().fit_transform(num)
lda = LDA(n_components=2)

Target_LDA = tmp
delete_list = []
for i in range(len(num)):
    if num[i][1] <= 5:
        delete_list.append(i)

Target_LDA = np.delete(Target_LDA,delete_list,axis = 0)
XD = np.delete(X,delete_list,axis = 0)

X_LDA_2D = lda.fit_transform(X, Target_LDA)
```

即将原数据降到 2 维，在画图过程中我们对应画出的图像如下：



图像中的蓝、绿、红三种颜色的点分别对应了 0,1,2 三种标签。可以很明显地看到，球员的得分能力体现在了 LDA 结果的横坐标上。其对应的 K-means 聚类结果如下：



这里的颜色不对应标签，可以看到，在 LDA 模型中，由于其为有监督的降维，其降维的结果有比较好的可解释性，用 K-means 聚类也能够得到较好的结果。

任务二 MovieLen 推荐

导入数据

在数据分析之前，需要将 csv 文件导入数据库，同样采用的是建表+插入操作。数据分为三个部分，分别是将 movies.csv、ratings.csv、tags.csv 导入数据库并进行处理。

以 tags.csv 为例，建表操作如下：

```
create table tags(userId int,movieId int,tag nvarchar(300),timestamp_ bigint)
```

然后是导入数据：

```
bulk insert tags
FROM '/usr/local/spark/wky/tags_unicode.csv'
WITH(FIELDTERMINATOR=',',ROWTERMINATOR='\n',FORMAT='CSV',DATAFILETYPE = 'widechar')
```

对 movies 表进行拆分

实现的代码如下：

```
create table genre_movie(movieID INT,genres VARCHAR(50))

insert into genre_movie(movieID,genres)
SELECT movieId,value
FROM movies
CROSS APPLY STRING_SPLIT(genres,'|')

create table movieTitlePubDate(movieId int,title NVARCHAR(500),pub_date int)

insert into movieTitlePubDate(movieId,title,pub_date)
Select movieId  , Substring(title, 1 ,LEN(title)-6) as title,
cast(Substring(title, LEN(title)-4, 4) as int) as  pub_date
from movies
```

结果：

	movieID	genres
1	467	Comedy
2	468	Comedy
3	468	Romance
4	469	Drama
5	469	Romance
6	470	Comedy
7	471	Comedy
8	472	Comedy
9	472	Drama
10	473	Comedy

	movieId	title	pub_date
1	1	Toy Story	1995
2	2	Jumanji	1995
3	3	Grumpier Old M..	1995
4	4	Waiting to Exha...	1995
5	5	Father of the Br...	1995
6	6	Heat	1995
7	7	Sabrina	1995
8	8	Tom and Huck	1995
9	9	Sudden Death	1995
10	10	GoldenEye	1995

ratings 表和 tags 表的时间转换

具体的代码如下：

```
create table ratingsTime(userId int, movieId int, rating float, time_ date)

insert into ratingsTime(userId,movieId,rating,time_)
select userId, movieId, rating, dateadd(second,timestamp_,{d'1970-01-01'}) as time_
from ratings
```

```
create table tagsTime(userId int, movieId int, tag nvarchar(200), time_ date)

insert into tagsTime(userId,movieId,tag,time_)
select userId, movieId, tag, dateadd(second,timestamp_,{d'1970-01-01'}) as time_
from tags
```

结果：

	userId	movieId	rating	time_
1	187723	6936	4	2005-01-27
2	187723	6947	4.5	2006-02-11
3	187723	7090	5	2004-09-07
4	187723	7143	4	2007-03-09
5	187723	7153	4.5	2004-06-25
6	187723	7235	3	2004-06-26
7	187723	7254	4	2006-11-01
8	187723	7325	3.5	2004-03-08
9	187723	7360	5	2004-06-25
10	187723	7361	5	2007-03-17

基于用户兴趣度进行推荐

基于用户兴趣度的推荐逻辑如下：统计一下用户对哪种类型的电影评分比较高，然后向他推荐他还没看过的这种类型的高分电影。故首先建立基于每个用户对每个类别的电影的平均评分，建表操作如下：

```
create table recommand_tab(userId int, genre nvarchar(200), avg_rating float)

insert into recommand_tab(userId,genre,avg_rating)
select ratingsTime.userId as userId,genre_movie.genres as genre,avg(rating) as avg_rating
from genre_movie inner join ratingsTime on genre_movie.movieID=ratingsTime.movieId
group by ratingsTime.userId,genre_movie.genres
order by userId,avg_rating DESC
```

表的大致结构如下：

	userId	genre	avg_rating
1	1	Romance	3.5
2	1	Horror	3
3	1	Western	3
4	1	Musical	3
5	1	Thriller	2.83333333333...
6	1	Action	2.8
7	1	Drama	2.57142857142...
8	1	Children	2.5
9	1	Crime	2.5
10	1	Sci-Fi	2.4
11	1	Adventure	2.16666666666...

同时再基于每一个类别构建一个电影评分递减的表，其建表操作如下：

```
create table best_movies_genres(genres nvarchar(200), title nvarchar(20),
avg_score float)
insert into best_movies_genres(genres, title, avg_score)
select g1.genres as genres, top.title as title, top.avg_score as avg_score
from (select distinct genres from genre_movie) as g1 cross apply(
    select g1.genres, m.title, avg(r.rating) as avg_score
    from movies as m, ratings as r, genre_movie as g2
    where g1.genres = g2.genres and m.movieId = r.movieId and m.movieId =
g2.movieID and m.movieId in(
        select movieID
```

```
        from ratings
        group by movieID
    )
    group by g2.genres, m.title
    order by g2.genres, avg_score
) as top
order by g1.genres
```

对于推荐部分，实验选用 python 的 MySQLdb 包来获取数据。python 代码部分的总体思路为：

（1）获取 best_movies_genres，recommand_tab 两个数据库的数据

（2）获取要查询的用户 ID

（3）从数据库中获取信息：这个人看过那些电影（需要除去已经看过的电影）

（4）获取最多 10 部最喜欢类型的电影

具体实现使用 python 代码，见/任务二 code/ part2/recommend.py

基于 Python 进行协同过滤的推荐

首先需要建立用户电影关联表，用户电影关联表首先需要对电影建立索引，因为电影的编号是非连续的：

```
create table movieIdIndex(indentity int,movieId int)

insert into movieIdIndex(indentity,movieId)
select ROW_NUMBER() OVER(ORDER BY movieId ASC) AS indentity,movieId
from movies
```

然后则是进行用户电影关联表的建立：

```
create table user_movie(userID int,indentity int,rating float)

insert into user_movie(userID,indentity,rating)
select userId,movieIdIndex.indentity,rating
from ratings,movieIdIndex
where ratings.movieId=movieIdIndex.movieId
```

结果：

	userID	indentity	rating
1	1	31	2.5
2	1	834	3
3	1	860	3
4	1	907	2
5	1	932	4
6	1	1018	2
7	1	1042	2
8	1	1048	2
9	1	1084	3.5
10	1	1088	2

接着我们使用 python 的 MySQLdb 包来获取数据，记录到一个 pandas 的 dataframe 中。

推荐算法为基于用户的协同过滤推荐算法以及基于商品的协同过滤推荐算法

对于“UBCF”（基于用户的协同过滤推荐算法），是指找到与待推荐商品的用户 u 兴趣爱好最为相似的 K 个用户，根据他们的兴趣爱好将他们喜欢的商品视为用户 u 可能会感兴趣的商品对用户 u 进行推荐。基于用户的协同过滤推荐算法主要分为两步，第一步是求出用户之间的相似度，第二步是根据用户之间的相似度找出与待推荐的用户最为相似的几个用户并根据他们的兴趣爱好向待推荐用户推荐其可能会感兴趣的商品。用户之间的相似度的计算主要可以通过 Jaccard 公式和余弦相似度公式得到。

Jaccard 公式如下：

$$w_{uv} = \frac{\left|N(u) \cap N(v)\right|}{\left|N(u) \cup N(v)\right|}$$

余弦相似度公式为：

$$w_{uv} = \frac{\left|N(u) \cap N(v)\right|}{\sqrt{\left|N(u)\right|\left|N(v)\right|}}$$

其中，N(u)为用户 u 感兴趣的商品，N(v)为用户 v 感兴趣的商品。

而计算用户 u 对商品 i 的感兴趣程度打分公式如下：

$$p(u,i) = \sum_{v \in S(u,K) \cap N(i)} w_{uv} r_{vi}$$

S(u,K)包含了和用户 u 兴趣最接近的 K 个用户，N(i)表示对商品 i 有过打分行为的用户集合，w_{uv} 表示计算出的用户 u 和用户 v 的兴趣相似度，r_{vi}表示用户 v 对商品 i 打的分数。得到了用户 u 对所有未打分商品的感兴趣程度分数后，将分数最高的几个商品作为用户 u 最有可能感兴趣的商品推荐给用户 u。

另一种推荐算法是基于商品的协同过滤推荐算法（IBCF），相较于用户之间的相似度，商品之间的相似度相对是静态的，当新用户注册并有了一些自己感兴趣的商品信息时，无需再进行计算，直接根据之前存储的商品之间的相似度，将用户可能感兴趣的商品推荐给用户。它也是分为两步，第一步是根据数据库中已有的信息求出商品的相似度，第二步是利用求出的商品之间的相似度计算用户对某种商品可能的兴趣程度。在我们的任务中，即要求出电影之间的相似度。

同样在这一部分我们也是计算了电影间的余弦相似度。

具体的代码流程如下：

首先我们根据获取的结果生成 User-Movies 二维 rating 矩阵。

```
train_data_matrix = np.zeros((n_users, n_items))
for i in range(len(train_data)):
    row = int(train_data.iloc[[i]]['user_id'])
    col = int(train_data.iloc[[i]]['movie_id'])
    train_data_matrix[row][col] = int(train_data.iloc[[i]]['rating'])

test_data_matrix = np.zeros((n_users, n_items))
for i in range(len(test_data)):
    row = int(test_data.iloc[[i]]['user_id'])
    col = int(test_data.iloc[[i]]['movie_id'])
    test_data_matrix[row][col] = int(test_data.iloc[[i]]['rating'])
```

接着我们计算 user-user 余弦相似度矩阵和 item-item 余弦相似度矩阵

```
import math
from sklearn.metrics.pairwise import cosine_similarity
#计算余弦相似度

#user_similarity = np.zeros((n_users, n_users))
user_similarity = cosine_similarity(train_data_matrix)


#item_similarity = np.zeros((n_items, n_items))
item_similarity = cosine_similarity(train_data_matrix.transpose())
```

通过相似度矩阵进行预测

```
#实现的是KNN思想，取k=3,即打分取3个的平均值
def predict(ratings, similarity, type='user'):
    prediction = np.zeros((n_users, n_items))
    if type == 'user':
        for i in range(1,n_users):
            ind = np.argsort(user_similarity[i])
            #print(ind)
            index1 = ind[-1]
            index2 = ind[-2]
            index3 = ind[-3]
            for j in range(1,n_items):
                if train_data_matrix[i][j] == 0:
                    prediction[i][j] = (train_data_matrix[index1][j] + train_data_matrix[index2][j] + train_data_matrix[index3][j])/3

    elif type == 'item':
        for i in range(1,n_items):
            ind = np.argsort(item_similarity[i])
            index1 = ind[-1]
            index2 = ind[-2]
            index3 = ind[-3]
            for j in range(1,n_users):
                if train_data_matrix[j][i] == 0:
                    prediction[j][i] = (train_data_matrix[j][index1] + train_data_matrix[j][index2] + train_data_matrix[j][index3])/3

    return prediction

item_prediction = predict(train_data_matrix, item_similarity, type='item')
user_prediction = predict(train_data_matrix, user_similarity, type='user')
```

最后通过 rmse 作为指标进行评估

```
from sklearn.metrics import mean_squared_error
from math import sqrt
def rmse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, ground_truth))

print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
```

最后得到的 RMSE 如下：

```
User-based CF RMSE: 2.680136812428301
Item-based CF RMSE: 2.5374160084621518
```

对于一个用户我们查询了两种不同推荐方式下推荐出来的三个电影如下：

```
——User-based——
Pulp Fiction (1994)
Dark Knight, The (2008)
Fight Club (1999)
——Item-based——
Boys of St. Vincent, The (1992)
Burnt by the Sun (Utomlyonnye solntsem) (1994)
Farinelli: il castrato (1994)
```