# Exp4 实验报告

**王湘峰 PB19030861**

# Part1

## 实验概述

实验四任务基于实验三，是实验三的拓展与延伸。Part1 中，手动实现一种分类算法（例如，决策树、 KNN 或者朴素贝叶斯）。并参考实验三特征工程，测试算法在 LoL 数据集上的预测性能，撰写实验报告。

## 实验要求

- 代码实现只允许使用 numpy、pandas 库和 python 内置库，不允许使用现有的机器学习库。
- 预测任务与实验三一致，以准确率作为评价指标。

自行在 LoL 数据集上划分训练集和验证集（4:1 比例、交叉验证），汇报算法在验证集上的性能。

- 实验报告需介绍实现算法的主要流程。

## 算法原理与实现

### • 算法原理

本次手动实现的分类算法是 K-NearestNeighbor 算法(KNN).它的主要原理是:

如果一个样本在特征空间中的 K 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。[1]

[1] 摘自百度百科

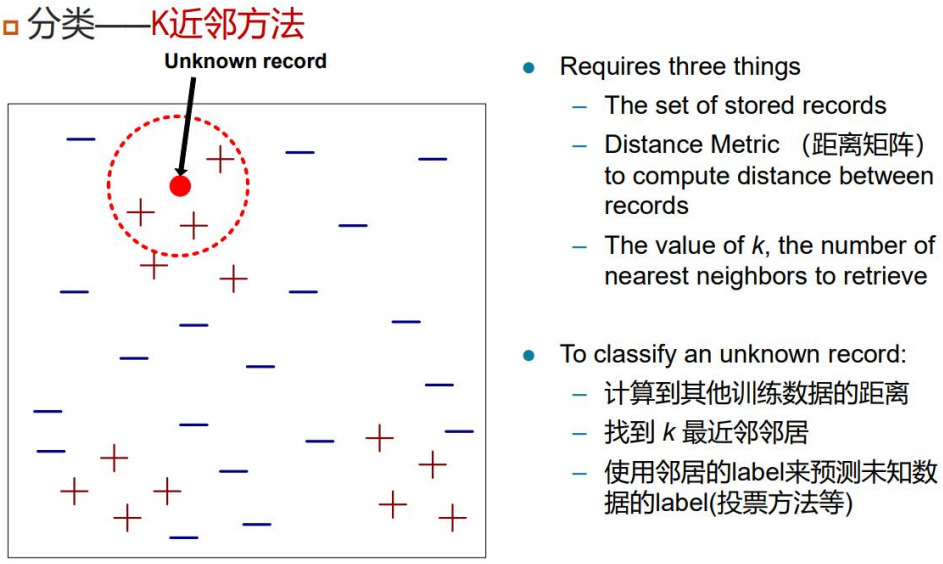


图 1：KNN 算法图解

了解了算法的主要思想之后，自然地提出以下三点疑问：

1. K 值如何选取？
2. 样本的距离如何计算？
3. 样本的各项特征是否应该加权？如何加权？

首先考虑距离的计算方式，一般为了简单起见，我们认为样本空间是平坦的，即曲率为 0，此时考虑闵可夫斯基距离，即

$$\boldsymbol{D(x,y)=\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{\frac{1}{p}}\quad p>0}$$

一般来说有 $p = 1$(曼哈顿距离)或者 $p = 2$(欧几里得距离)，本次实验中同时使用了欧几里得距离和曼哈顿距离作为度量标准，并比较他们的性能。

其次考虑样本的特征，在 exp3 中，我们发现首先拿下第一滴血、第一条大龙、峡谷先锋、水晶，经济领先等对胜率的提升都是有帮助的，因此本次我们考虑 11 个特征：

1. firstBlood
2. firstTower
3. firstInhibitor
4. firstBaron
5. firstDragon
6. firstRiftHerald
7. gold-Difference
8. kills-Difference

根据 exp3 中的分析，我们对不同特征赋予不同的权重，具体情况在 jupyter notebook 中的“par”列表里。

最后关于 K 的选取，经过数次测试，我们认为 30-70 都是效果不错的，本次实验以 30 为例。

## • 算法实现

1. 读入原始 csv 文件

2. 利用 exp3 实现的函数得到以上 8 个特征

3. 对特征进行归一化：$\bar{x} = \frac{x - \min(x)}{\max(x) - \min(x)}$

4. 对于每一行测试数据，依照距离公式计算它与训练集的距离

5. 在训练集中对距离降序排列并截断前 K 个

6. 统计训练集的胜负标签并投票

7. 统计正确率，并进行交差检验

## • 实验试错日志

（这里记录了实验过程中遇到的主要困难与尝试的解决方法）

**困难 1：**

没有对数据进行归一化，导致在设定参数时无从下手。

**解决方案：**

重新学习课件 PPT，发现了预处理的漏洞

**困难 2：**

在计算 KDA 时，发现有一些非常规对局（初步估计可能是投降对局）的 deaths 为 0，导致 KDA 为 $\infty$ ，在 pandas 中表示为 inf，无法进行后续的计算

**解决方案：**

初次尝试使用 for 循环进行搜索并把 inf 修改为 NaN，然后再 drop 掉，后来发现时间复杂度太高，最后的解决方法是使用 pandas 中的 options 选项进行设置，即可自动把 inf 转化为 NaN。

**困难 3：**

初始的代码是用索引+for 循环的形式实现的，但是数据集过大，导致性能低下，即使是在 1/10 的测试集上也需要十几分钟才能出结果，根本不能优化参数。

**解决方案：**

重新学习 pandas 的教程，尝试使用列操作从而减少了对内存数据 I/O，先前对于 16000 个 test 数据需要大约 1000 秒的时间才能出结果，优化后在无后台进程的情况下最快 240 秒(i7-8565U 1.8GHz)可以得到结果。

- **性能测试**

K 折检验结果如下：(欧几里得距离)

```
In [15]: test=geten(df.loc[0:16000])
         #train=geten(pd.concat([df.loc[32001:],df.loc[0:16000]],axis=0))
         train=geten(df.loc[16001:])
         KNN(train,test,k)
         100%|██████████| 15839/15839 [04:16<00:00, 61.68it/s]
         预测准确率是0.981312 KNN用时： 257.0684072971344
Out[15]: 0.9813119515120904

In [16]: test=geten(df.loc[16001:32000])
         train=geten(pd.concat([df.loc[32001:],df.loc[0:16000]],axis=0))
         KNN(train,test,k)
         100%|██████████| 15841/15841 [04:22<00:00, 60.27it/s]
         预测准确率是0.978852 KNN用时： 263.06513476371765
Out[16]: 0.9788523451802286

In [17]: test=geten(df.loc[32001:48000])
         train=geten(pd.concat([df.loc[48001:],df.loc[0:32000]],axis=0))
         KNN(train,test,k)
         100%|██████████| 15835/15835 [04:11<00:00, 62.87it/s]
         预测准确率是0.980044 KNN用时： 252.14181351661682
Out[17]: 0.980044205873066

In [18]: test=geten(df.loc[48001:64000])
         train=geten(pd.concat([df.loc[64001:],df.loc[0:48000]],axis=0))
         KNN(train,test,k)
         100%|██████████| 15848/15848 [04:14<00:00, 62.20it/s]
         预测准确率是0.980313 KNN用时： 255.012300491333
Out[18]: 0.9803129732458354

In [19]: test=geten(df.loc[64001:])
         train=geten(df.loc[0:64000])
         KNN(train,test,k)
         100%|██████████| 15824/15824 [04:12<00:00, 62.57it/s]
         预测准确率是0.979778 KNN用时： 253.15547800064087
Out[19]: 0.9797775530839231

In [ ]: The average is 0.98001139919079806
```

图 2：K 折检验结果(欧几里得距离)

预测率的平均值为 98.0%，KNN 算法取得了不错的效果！下面是使用曼哈顿距离测试的结果：

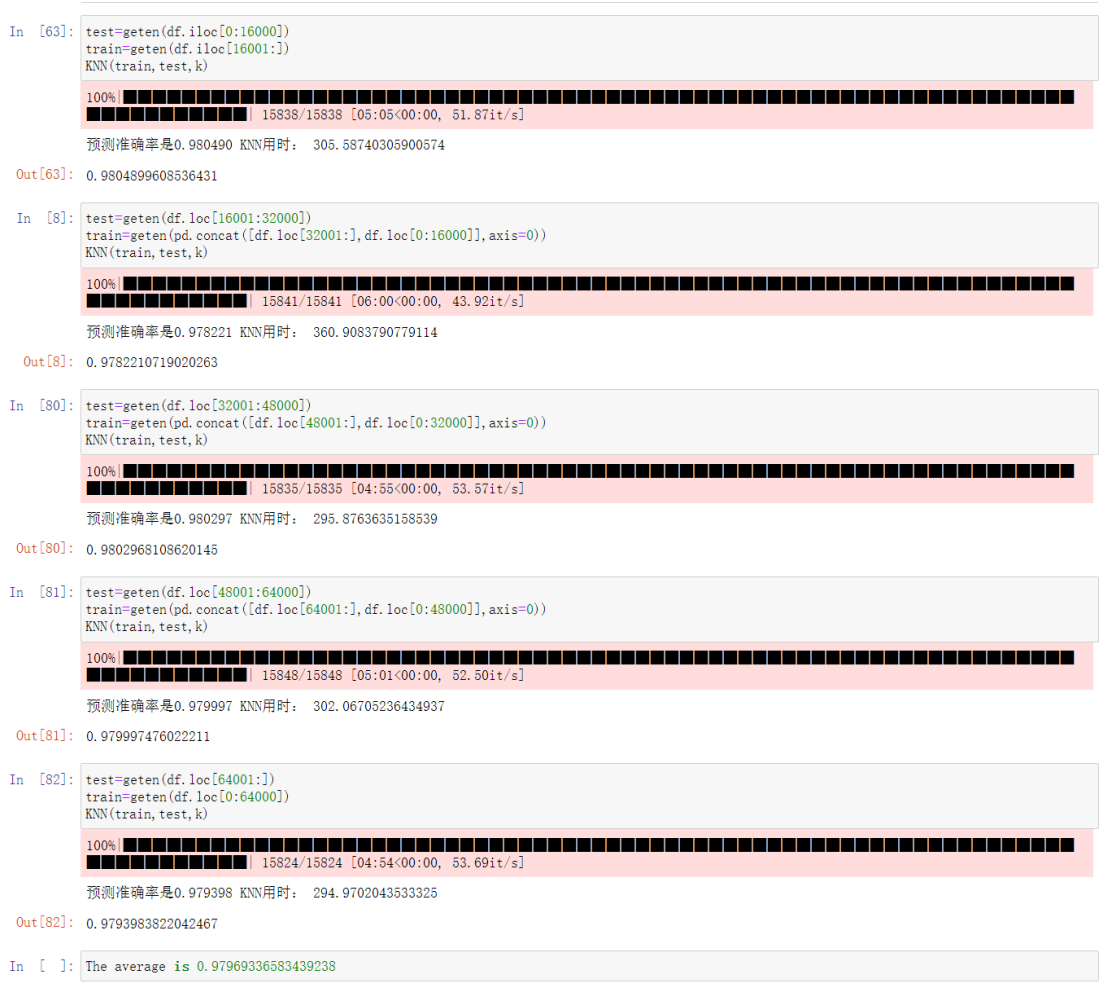


图 3：K 折检验结果(曼哈顿距离)

使用曼哈顿距离得到的平均预测准确率为 97.97%，虽略低于欧氏距离但效果也很不错。

总之，不论是用欧式距离还是曼哈顿距离，KNN 都有着不错的表现，个人觉得 part1 总体上比较成功。

# Part2

## 实验要求

• 大家利用所学的知识，进行一场数据科学实战。数据集仍然使用实验三的数据集，但预测任务有变。可以利用开源工具包，也可以参考实验三的数据分析与特征工程。

• 实验报告需要记录最终的方案流程，也鼓励大家记录每一次失败的尝试。

## 算法实现

- **算法选取**

目前针对本次实验的算法主要有分类和回归两种，在 Part2 中我们将同时采用两种算法并进行比较。其中由于 Part2 的预测目标是连续的标签(游戏时长)，因此分类算法在使用前应当对数据进行预处理。我的做法是：把训练集的时间分割成一个个相等的区间作为数据的标签，然后再做成分类问题，这样预测得到的结果偏差一般不会特别大。

综合考虑，本次实验将使用 **KNN 分类、KNN 回归、决策树回归、神经网络-多层感知器回归**分别进行预测。

## • 特征选取

与 Part1 不同，游戏时长一般来说更多的取决于双方的经济总量、击杀死亡和助攻总数，即我们认为游戏结束时双方经济等指标越高则游戏时间越长。因此本次实验选取了以下 4 个特征：

1. **双方经济总和**
2. **双方击杀总和**
3. **双方死亡总和**
4. **双方助攻总和**

## • 数据预处理

在 KNN 分类算法中，首先通过 pandas 提取以上四个特征并进行归一化。然后对于训练集，我们对 gameDuration 进行分割区间，以 50 一个区间为例，先将 gameDuration 除以 50，对其取上整后再乘以 50，就得到了它的标签。之后再带入 KNN 函数中即可。

对于回归算法，则仅做到归一化那一步即可。

## • 性能测试

**KNN 分类**：经过多次调参，得到最佳区间长度为 35。

下图是网格搜索最佳 K 值以及最小均方误差以及合格率情况：

In [7]:
```
#KNN分类算法（分隔区间法）
min=float('inf')
for k in tqdm(range(5,70)):  #网格搜索最佳K值
    knn = neighbors.KNeighborsClassifier(k)
    knn.fit(x_train,y_train)
    pre=knn.predict(x_test)
    err=metrics.mean_squared_error(pre,y_test)
    if err<min:
        min=err
        best=k
print('best k is:',best,'min error is:',min)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 65/65 [00:40<00:00,  1.62it/s]
best k is: 63 min error is: 15444.495707975664
```

In [8]:
```
knn = neighbors.KNeighborsClassifier(best)
knn.fit(x_train,y_train)
```

Out[8]:
```
KNeighborsClassifier(n_neighbors=63)
```

In [9]:
```
#如果预测值与真实值误差小于100那么认为预测是“合格”的，计算合格率
n=0
for i in range(len(pre)):
    if abs(pre[i]-y_test[i])<100:
        n=n+1
print('合格率为：',n/len(pre))
```

```
合格率为： 0.8434036169680806
```

图 4：KNN 分类的均方误差与合格率

可以看出， KNN 分类算法均方误差约为 15000，效果中规中矩，属于勉强及格的类型。

**KNN 回归：**下图是 KNN 回归算法的性能测试，同样使用网格搜索最佳 K 值：

```
#网格搜索最佳K值
min=float('inf')
for k in tqdm(range(5,70)):
    knn = neighbors.KNeighborsRegressor(k)
    knn.fit(x_train,y_train)
    pre=knn.predict(x_test)
    err=metrics.mean_squared_error(pre,y_test)
    if err<min:
        min=err
        best=k
print('best k is:',best,'min error is:',min)
```

```
100%|████████████████████████████████████████████████████████████| 65/65 [00:20<00:00,  3.18it/s]
```

```
best k is: 25 min error is: 12326.612138744895
```

```
knn = neighbors.KNeighborsRegressor(best)
knn.fit(x_train,y_train)
```

```
KNeighborsRegressor(n_neighbors=25)
```

```
#如果预测值与真实值误差小于100那么认为预测是“合格”的，计算合格率
n=0
for i in range(len(pre)):
    if abs(pre[i]-y_test[i])<100:
        n=n+1
print(n/len(pre))
```

```
0.8784898741561796
```

图 5：KNN 回归的均方误差和合格率

相较于 KNN 分类，KNN 回归的性能具有较大的提升，不论是均方误差还是合格率都有不小的进步。事实上 KNN 回归算法是本次 Part2 实验中效果最佳的算法，最终提交的 csv 文件也是用 KNN 回归预测的。

**决策树回归：**一般对于连续型预测目标来说，回归算法比分类算法好些，因此后续的实验都直接采用回归算法进行测试。

```
In [21]: clf = tree.DecisionTreeRegressor()
         clf = clf.fit(x_train,y_train)
         pre=clf.predict(x_test)
         err=metrics.mean_squared_error(pre,y_test)
         print(err)

         24668.776481373447

In [22]: #检测合格率
         n=0
         for i in range(len(pre)):
             if abs(pre[i]-y_test[i])<100:
                 n=n+1
         print('预测合格率为：',n/len(pre))

         预测合格率为： 0.7678973247770647
```

图 6：决策树回归的均方误差与合格率

从结果上看，决策树效果与 KNN 分类算法效果相近（甚至还差一点）

**神经网络-多层感知机回归：**

```
In [38]: clf = MLPRegressor(solver='adam', alpha=1e-5,max_iter=1500,learning_rate_init=0.005,
                            hidden_layer_sizes=(8,),random_state=1)

         clf.fit(x_train,y_train)

Out[38]: MLPRegressor(alpha=1e-05, hidden_layer_sizes=(8,), learning_rate_init=0.005,
                      max_iter=1500, random_state=1)

In [39]: pre=clf.predict(x_test)
         err=metrics.mean_squared_error(pre,y_test)
         print(pre,err)

         [1184.91415253 1303.64160819 1073.08453765 ... 1225.1635191  1741.45968532
          2350.14716721] 18261.572422111127

In [40]: #检测合格率
         n=0
         for i in range(len(pre)):
             if abs(pre[i]-y_test[i])<100:
                 n=n+1
         print(n/len(pre))

         0.8611550962580214
```

图 7：MLP 回归的均方误差与合格率

在 MLP 算法的测试中，对于 solvers 选项我先后使用了 lbfgs，sgd 和 adam 方法来调参，最终发现 adam 方法最佳；同时找到了最佳的神经网络隐含层数量和神经元的数量（单层，8 个）.最终 MLP 给出的答案合格率较高但均方误差略大（相较于最佳的 KNN 回归而言），这说明 MLP 对大多数的数据预测偏差较小而对于少数数据预测误差过大。

# 实验总结与分析

通过实验 Part1，我自己动手实现了数据挖掘中的 KNN 分类算法，加深了对算法本身的认识，同时熟悉了 pandas、numpy 等常用数据分析库的使用方法。这次实验是我目前接触到的数据量最大的一次实验，在漫长的等待中我才真切感受到时间复杂度的概念以及优化算法的重要性。同时对于数据的预处理、调参等工作流程也有了初步的认识，可谓是收获满满。

实验的 Part2 部分主要通过开源工具包实现的，当我使用了 sklearn 的 KNN 函数的时候，我才意识到和专业开发者的差距：同样的数据量使用 sklearn 的函数可以在 15 秒之内得到结果。后来仔细阅读了 sklearn 的中文文档之后才发现前人们为了解决 KNN 算法时间复杂度过高的问题，发明了 KD 树、球树等数据结构来优化计算，不得不佩服数据科学家前辈们的智慧。

总而言之，通过这两部分实验，我认为要想算法性能好，特征的选取是最关键的。我曾尝试过直接将 Part1 的特征带入 Part2 中，但是发现效果较差，经过思考之后重新选择了新的特征，取得了较为不错的效果。

此外，除了 Part2 的报告中提到的 4 种算法，课下我还尝试了朴素贝叶斯、支持向量机、广义线性回归、岭回归、lasso 回归等算法，不过由于测试性能不佳（均方误差在 20000-50000）以及不能调参等原因，没有写入报告里。对于这次实验，我个人认为 Part1 和 Part2 循序渐进，过渡较为自然，实验难度相对来说也比较友好，在这次实验中我收获了很多。同时也感谢老师与助教的精心设计！