

# 宝可梦数据分析和预测

张思雨

10214804409

## 目录

1 引言	3
1.1 项目背景	3
1.1.1 选题背景	3
1.1.2 研究背景	3
1.2 研究方法	3
1.2.1 方法	3
2 数据采集与介绍	3
2.1 数据获取	3
2.1.1 数据来源	3
2.1.2 数据集介绍	3
3 数据预处理与可视化	4
3.1 数量分析	4
3.1.1 每一代精灵宝可梦的数量	4
3.1.2 每一代传奇宝可梦的数量	5
3.1.3 每一代传奇宝可梦的占比	6
3.2 捕获率和属性分析	6
3.2.1 传奇宝可梦和普通宝可梦捕获率对比	6
3.2.2 双系宝可梦数量	7
3.3 战斗分析	8
3.3.1 总体战斗值	8
3.3.2 龙系宝可梦战斗分析	10
3.4 抗性分析	10
3.4.1 各精灵抗性总值	10
3.4.2 各系精灵 against 属性值关系	12
4 数据预测	12
4.1 数据预处理	12
4.1.1 数据清洗	12
4.1.2 数据选取	13
4.1.3 标准化	13
4.2 模型预测	13
4.2.1 模型选取	13
4.2.2 预测正确率	14
4.3 KNN 调参	14
4.3.1 研究不同的 test_size 对预测结果的影响	14
4.4 探究预测过程中，各个特征的重要性	15
4.4.1 随机森林模型评估特征重要性	15
5. 总结	16
5.1 数据可视化部分	16
5.2 预测部分	16

# 1 引言

## 1.1 项目背景

### 1.1.1 选题背景

传奇宝可梦（Legendary Pokémon）作为宝可梦系列游戏中的稀有物种，具有极高的价值和吸引力。这些宝可梦通常具有特殊的能力和技能，而且在游戏中只能捕捉到一次。因此，预测一个宝可梦是否为传奇宝可梦成为了玩家们关注的重要问题。

### 1.1.2 研究背景

收集 1-8 代 1013 只精灵宝可梦的数据集，对宝可梦的各项数据特征，例如种族值、技能、进化条件进行深入研究，并构建一个预测模型。预测一个宝可梦是否为传奇宝可梦。

## 1.2 研究方法

### 1.2.1 方法

利用 excel 和 python 对数据进行预处理  
利用基于 Python 平台的 Numpy、Pandas 以及机器学习算法库 scikit-learn 提供的 KNeighborsClassifier 分析模型进行数据预测

# 2 数据采集与介绍

## 2.1 数据获取

### 2.1.1 数据来源

数据来源于阿里天池 <https://tianchi.aliyun.com/dataset/147523>

### 2.1.2 数据集介绍

名称	中文解释
pokedex_number	宝可梦图鉴 ID

name	名称
generation	第几代
classification	精灵类型
abilities	特殊能力
height_m	身高(米)
weight_kg	体重(千克)
type1	主属性
type2	副属性
base_total	基础总值
hp	声明值
attack	基础攻击属性
defense	基础防御属性
sp_attack	特殊攻击属性
sp_defense	特殊防御属性
speed	基础速度属性
capture_rate	捕捉几率
base_egg_steps	孵化阶段
base_happiness	活跃指数
against_?	18 项定向攻击的伤害指数
is_legendary	是否为传奇宝可梦(1 为是)

## 3 数据预处理与可视化

### 3.1 数量分析

#### 3.1.1 每一代精灵宝可梦的数量

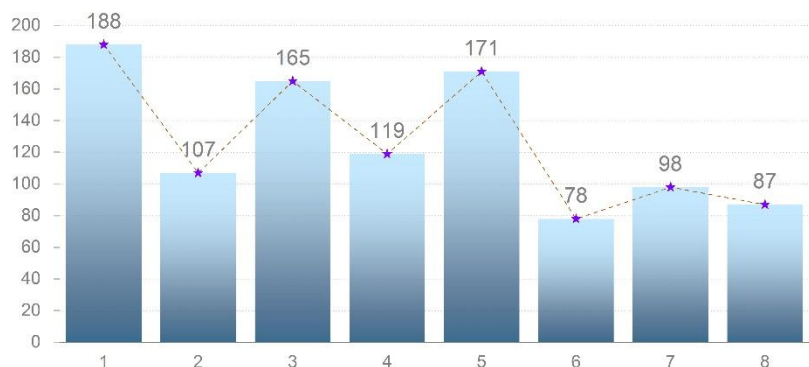
将 generation 相同的数量进行求和。

```

3 def get_count(excel_path, column_name):
4     # 读取工作簿
5     df = pd.read_excel(excel_path)
6     # 统计相同文本的数量
7     counts = df[column_name].value_counts()
8     return counts
9
10 # 测试代码
11 excel_path = 'pokemon.xlsx'
12 column_name = 'generation'
13 print(get_count(excel_path, column_name))

```

1~8 代宝可梦数量

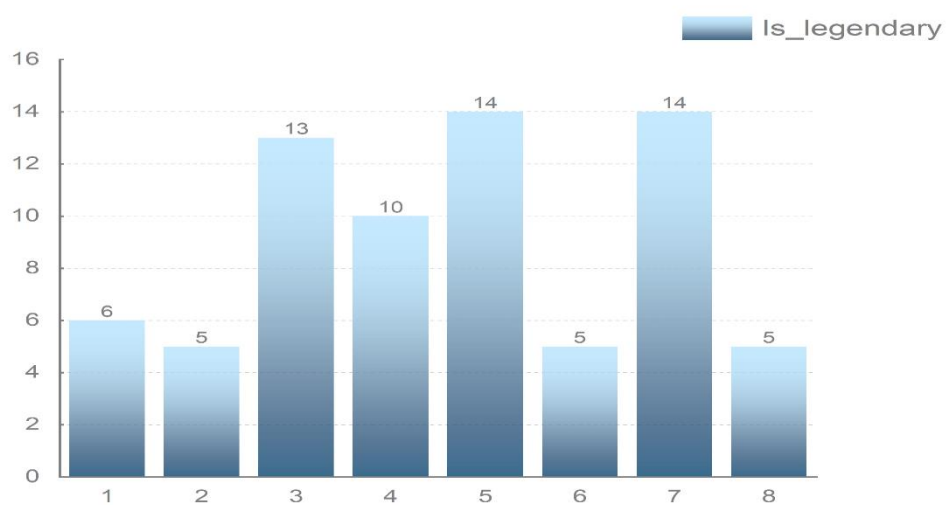


第一代精灵数量最多，6、7、8代相对较少。

### 3.1.2 每一代传奇宝可梦的数量

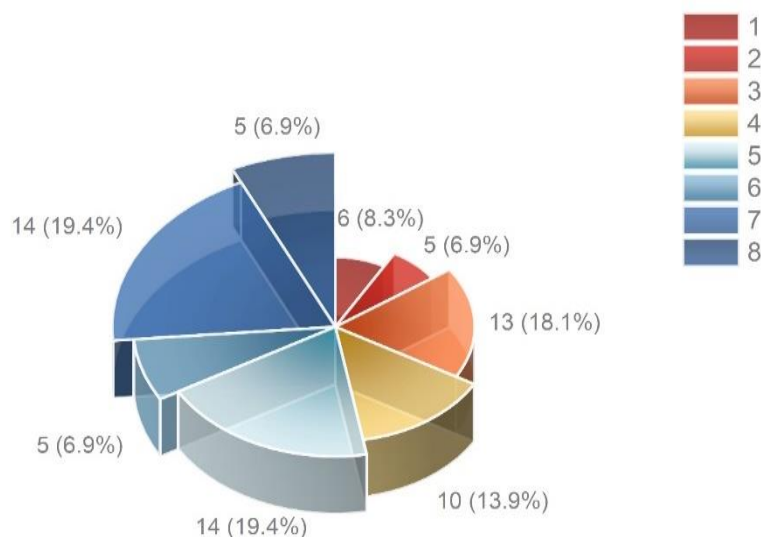
```
24  
25 group_generation = df.groupby('generation') # 按generation分组  
26 group_generation = group_generation[['is_Legendary']].sum()  
27
```

各代传奇宝可梦的数量



### 3.1.3 每一代传奇宝可梦的占比

各代传奇宝可梦的占比



第一代精灵宝可梦数量虽多，但传奇宝可梦占比不高。综合来看，3、5、7代中传奇宝可梦占比较大

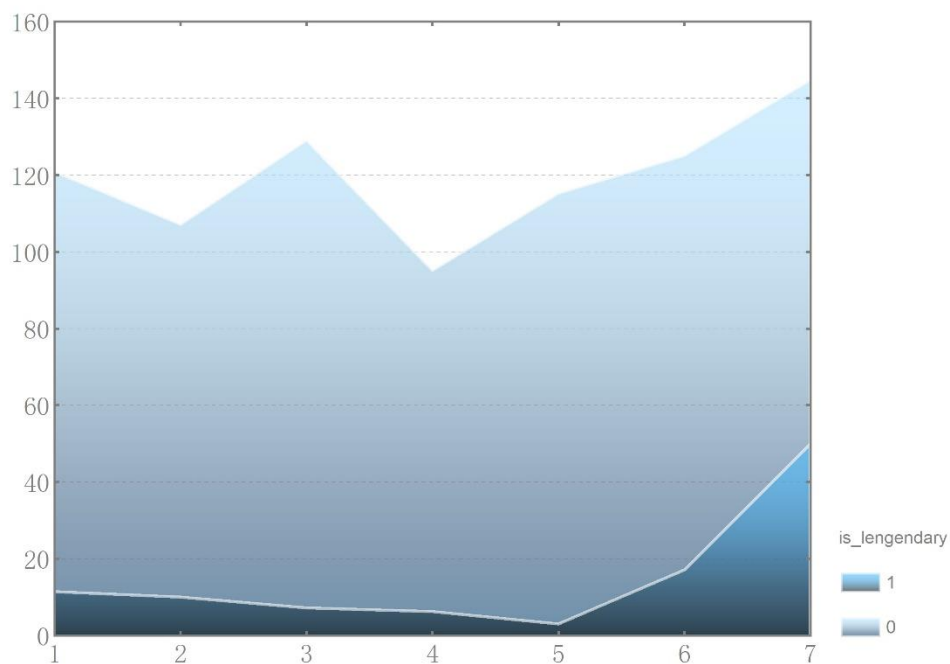
## 3.2 捕获率和属性分析

### 3.2.1 传奇宝可梦和普通宝可梦捕获率对比

将传奇宝可梦和普通宝可梦分开分别求捕获率平均值。

```
3 def get_average(excel_path, sheet_name, name_column, number_column):
4     # 读取工作簿
5     df = pd.read_excel(excel_path, sheet_name)
6     # 统计相同名称的数量并求平均值
7     average = df.groupby(name_column)[number_column].mean()
8     return average
9
10 excel_path = 'demo.xlsx'
11 sheet_name = '捕获率对比'
12 name_column = 'generation'
13 number_column = 'capture_rate'
14 print(get_average(excel_path, sheet_name, name_column, number_column))
```

### 传奇宝可梦与普通宝可梦捕获率对比



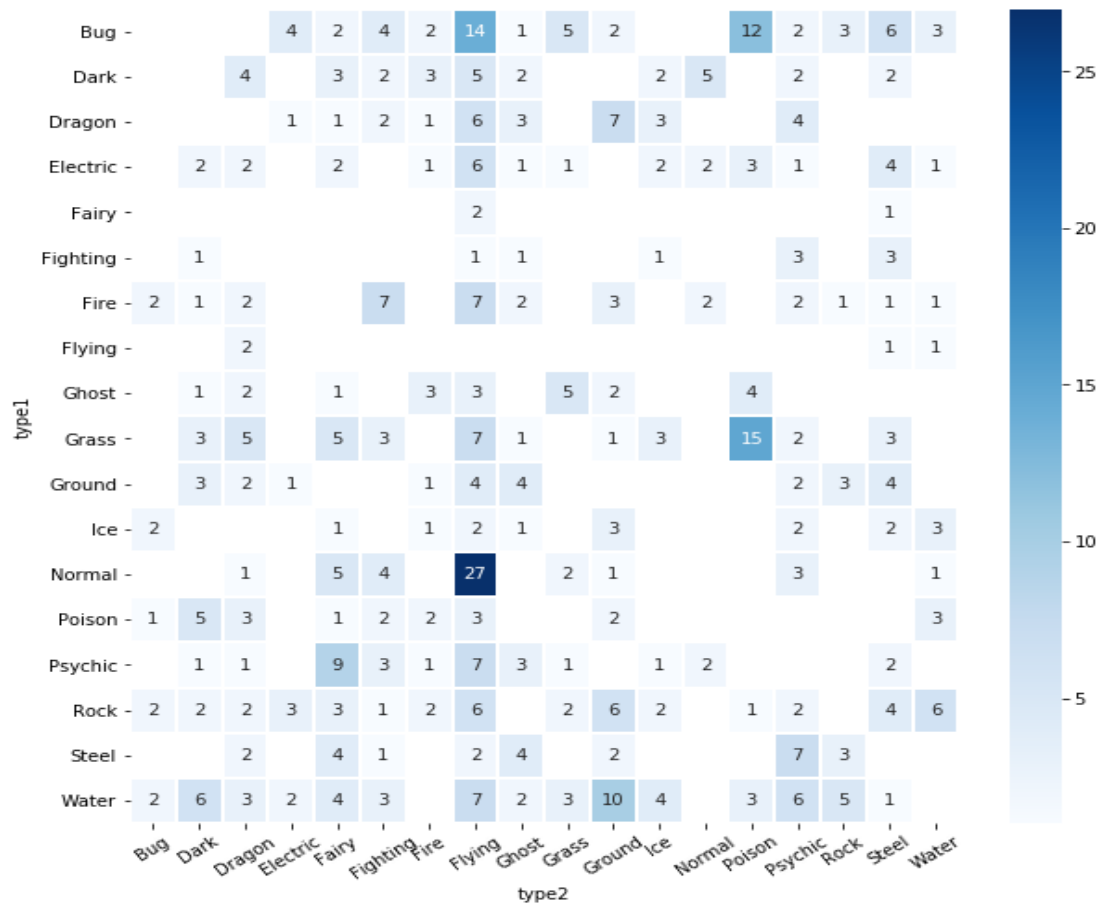
捕获率方面龙系捕获率最低，传奇系远低于普通系，直到第七代，传奇系捕获率才有一个明显上升。由于龙系及传奇系本身就很强大，而越强大的宝可梦就越稀少，捕获难度自然就高。

### 3.2.2 双系宝可梦数量

统计双系宝可梦的数量,type2 缺失较多

```
1 plt.subplots(figsize=(10, 10))
2 sns.heatmap(df[df['type2']!= 'None'].groupby(['type1', 'type2']).size().unstack(), linewidths=1, annot=True, cmap="Bl")
3 plt.xticks(rotation=35)
4 plt.show()
```

双系宝可梦数量图



type1 为 Normal, type2 为 Flying 的宝可梦数量最多

## 3.3 战斗分析

### 3.3.1 总体战斗值

对血量、攻击力、防御力、特殊攻击、特殊防御、速度六个基础值做标准化处理，并单独取出传奇的龙系。

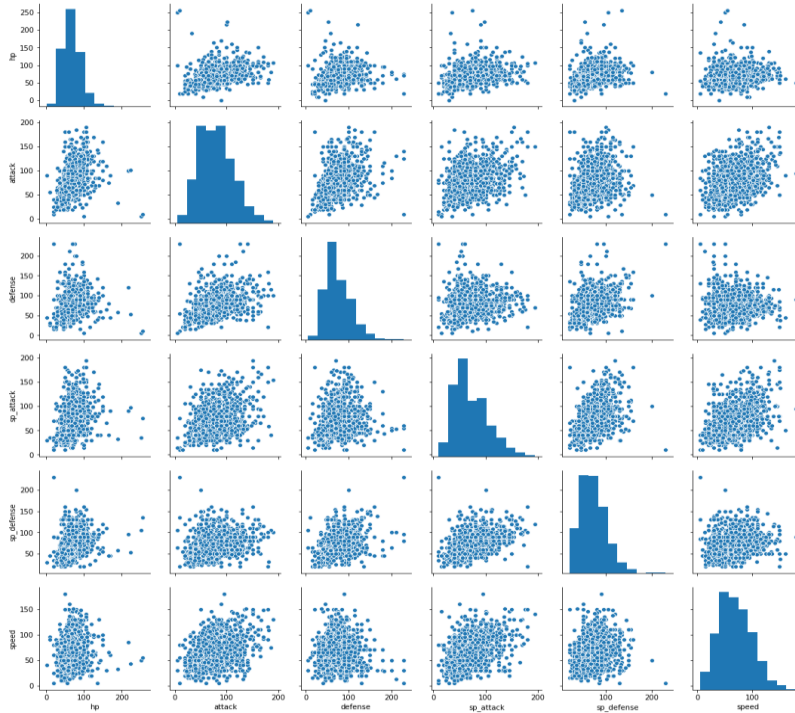
```
1 interested = ['hp', 'attack', 'defense', 'sp_attack', 'sp_defense', 'speed']
2 sns.pairplot(df[interested])
3 plt.show()
```



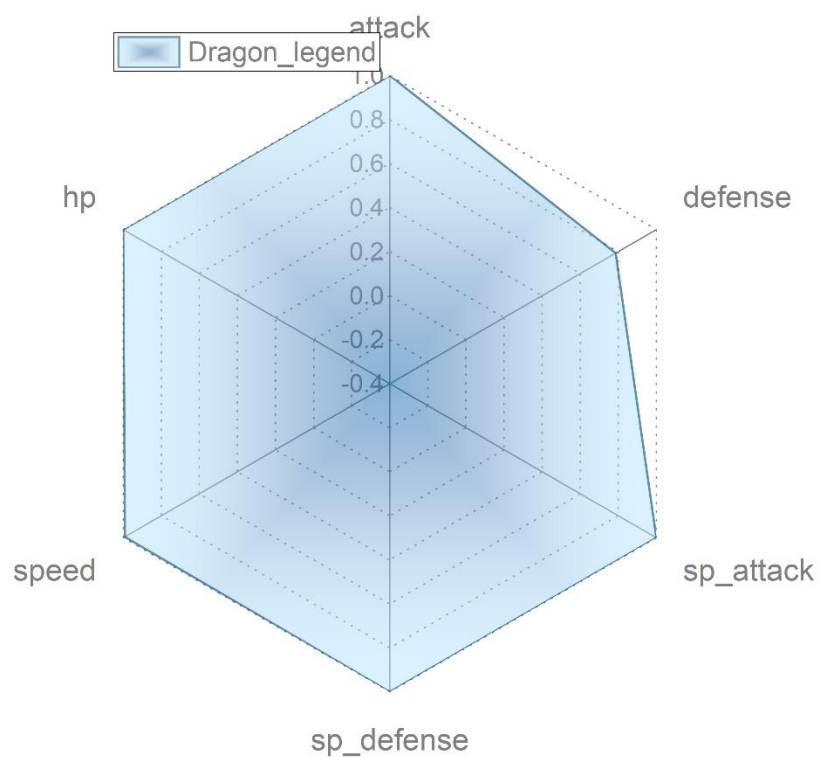
```

1 from sklearn.preprocessing import MinMaxScaler
2 col = ['attack', 'defense', 'sp_attack', 'sp_defense', 'speed', 'hp']
3 data_scaled=df[col]
4 data_scaled=pd.concat([data_scaled,df['type1'],df['is_Legendary']],axis=1,join='outer') # 外连接
5 # 分组统计
6 data_scaled1=data_scaled.groupby('type1').mean()
7 data_scaled1=data_scaled1.drop(['is_Legendary'],axis=1)
8 # 构建传奇龙系属性
9 data_scaled2=data_scaled.groupby(['is_Legendary','type1']).mean().reset_index()
10 dat=data_scaled2[(data_scaled2['is_Legendary']==1)&(data_scaled2['type1']=='dragon')] # 取出传奇的龙系
11 dat=dat.drop(['is_Legendary'],axis=1)
12 dat=dat.rename({'20':'dragon_Legend'}) # 重命名行索引
13 # 构建其它传奇属性
14 dat1=data_scaled.groupby(['is_Legendary']).mean().reset_index()
15 dat1=dat1[dat1['is_Legendary']==1]
16 dat1=dat1.drop(['is_Legendary'],axis=1)
17 dat1=dat1.rename({'1':'is_Legendary'})
18 # 数据合并
19 data_scaled1=pd.concat([data_scaled1,dat,dat1],axis=0,join='outer')
20 data_scaled1=data_scaled1.drop(['type1'],axis=1)
21 ind = list(data_scaled1.index) # 获取行索引
22 # 标准化
23 model_scaler = MinMaxScaler()
24 data_scaled1 = model_scaler.fit_transform(data_scaled1) # 标准化处理
25 data_scaled1= pd.DataFrame(data_scaled1,index=ind,columns=col)
26 data_scaled1

```



### 3.3.2 龙系宝可梦战斗分析



## 3.4 抗性分析

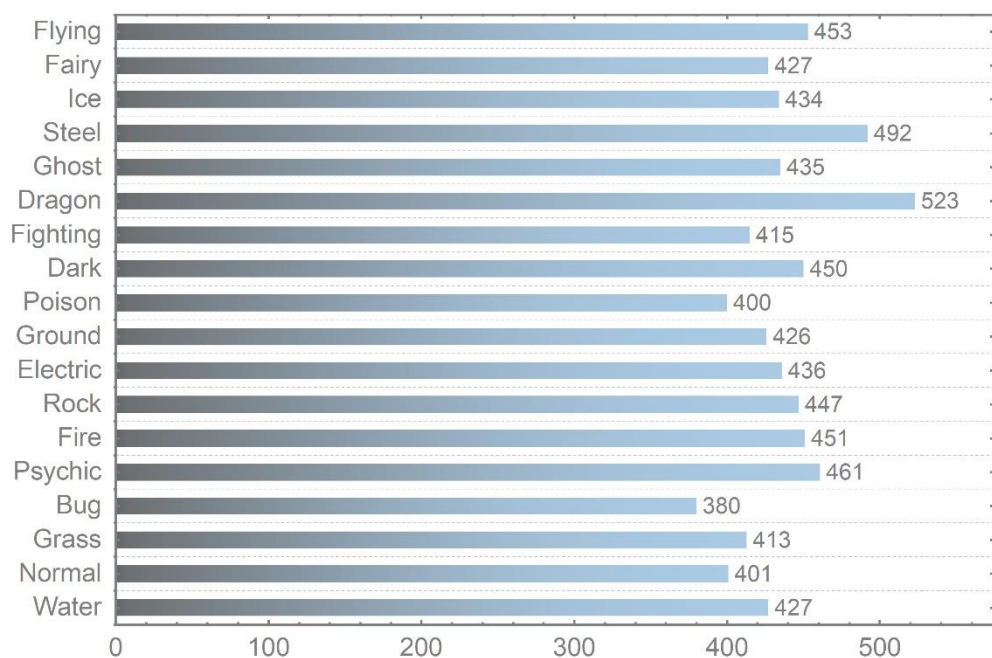
### 3.4.1 各精灵抗性总值

针对 against\_bug, against\_dark, against\_dragon, against\_electric, against\_fairy, against\_fighting, against\_fire, against\_flying, against\_ghost, against\_grass, against\_ground, against\_ice, against\_normal, against\_poison, against\_psychic, against\_rock, against\_steel, against\_water,

即含有 “against”的数据进行加总。首先读文件，并将结果存储在 df 变量中。然后使用 groupby 函数对指定的属性列和数值列进行分组，并使用 sum 函数对每个分组的数值列进行求和。最后打印出每个属性的总和。

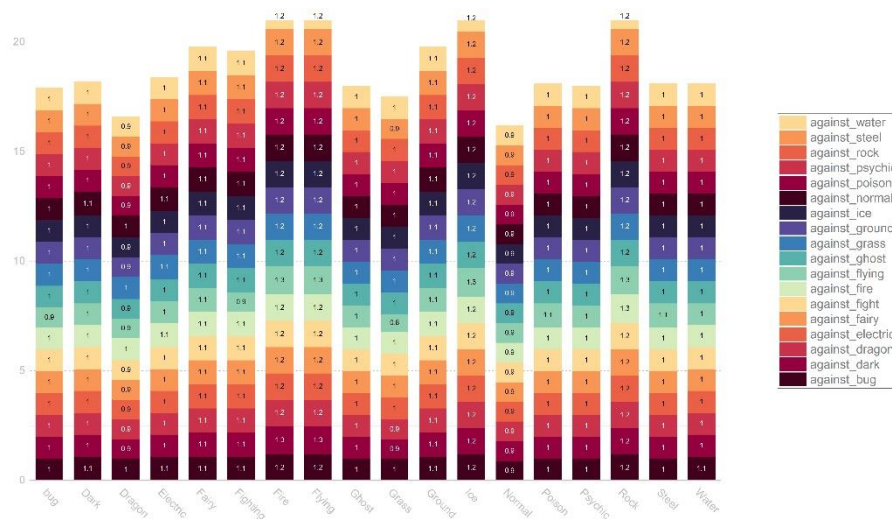
```
3 def sum_by_attribute(excel_path, attribute_column, value_column):
4     # 读取 Excel 文件
5     df = pd.read_excel(excel_path)
6
7     # 对指定属性列和数值列进行加总求和
8     total_sum = df.groupby(attribute_column)[value_column].sum()
9
10    # 打印结果
11    print(total_sum)
12
13 excel_path = 'pokemon.xlsx'
14 attribute_column = 'type'
15 value_column = 'against-'
16 sum_by_attribute(excel_path, attribute_column, value_column)
```

种族属性总值



龙系的属性总值最高，超过了 500，虫系最低，低于 400，与上面雷达图展示一致。虫系历来就是被玩家认为是属性，抗性也较弱，很多属性对虫系都有抵抗效果，能力较高的虫系宝可梦，数量也相对较少。但虫系的宝可梦培育的难度比较小，可以很快的进化到最终形态，能够在游戏的前期发挥重要的作用。

## 3.4.2 各系精灵 against 属性值关系



抗性方面:钢系抗性总值是最好的,除了地面、格斗、火属性伤害值相对较高,其余都较低。岩石系及冰系抗性是最弱的,岩石系对格斗、草系、地面、钢系、水系抗性较低,冰系对火系、格斗、岩石、钢系抗性较低,龙系处于中间阶段。

```
3 def get_average(excel_path, sheet_name, name_column, content_column):
4     # 读取工作簿
5     df = pd.read_excel(excel_path, sheet_name)
6     # 统计相同名称的数量并求平均值
7     average = df.groupby(name_column)[content_column].apply(lambda x: x.str.split(',').len()).mean()
8     return average
9
10 excel_path = 'pokemon.xlsx'
11 sheet_name = 'ability'
12 name_column = 'type'
13 content_column = 'abilities'
14 print(get_average(excel_path, sheet_name, name_column, content_column))
```

## 4 数据预测

### 4.1.数据预处理

#### 4.1.1 数据清洗

观察发现 type1 数据为字符串, 分别替换为数值 1~19

```
# 数据清洗
df['type1'].replace(['Bug', 'Dark', 'Dragon', 'Electric', 'Fairy', 'Fighting', 'Fire',
                    'Flying', 'Ghost', 'Grass', 'Ground', 'Ice', 'Normal', 'Poison',
                    'Psychic', 'Rock', 'Steel', 'Water'],
                    list(range(1, 19)), inplace=True)
```

### 4.1.2 数据选取

由于 type2 数据缺失过多，classification 与 abilities 为多重复合特征，在预测时去除这三个特征，并去除预测目标“is\_legendary”列

```
25 # 划分数据集
26 target = 'is_legendary'
27 X = df.iloc[:, 2:].drop(columns=['type2', 'classification', 'abilities', target])
```

### 4.1.3 标准化

由于不同特征的数据数量级相差较大，因此进行了标准化处理

```
15 # 标准化处理
16 X = (X - np.mean(X)) / np.std(X)
```

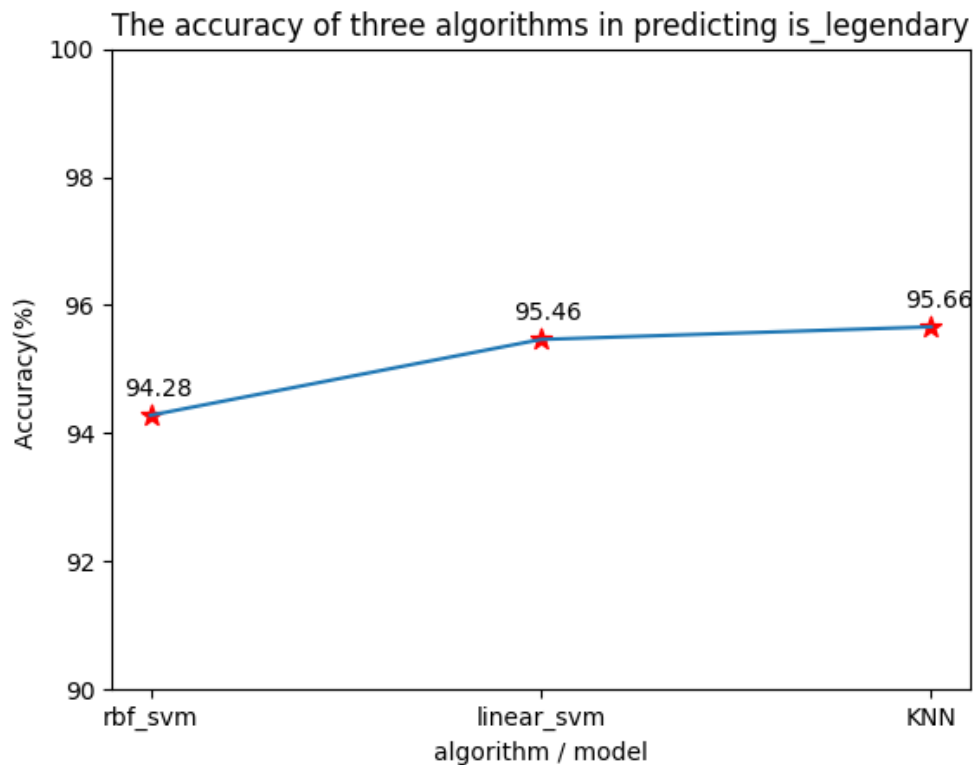
## 4.2 模型预测

### 4.2.1 模型选取

使用 rbf\_svm、linear\_svm、KNN 三种不同的预测模型精灵宝可梦是否为传奇精灵，并查看预测的正确率

```
31 # 使用不同的预测模型精灵宝可梦是否为传奇精灵，并查看预测的正确率
32 machine_name = np.array(['rbf_svm', 'linear_svm', 'KNN'])
33 machine_x = np.array([1, 2, 3])
34 machine_score = np.array([])
35
```

## 4.2.2 预测正确率

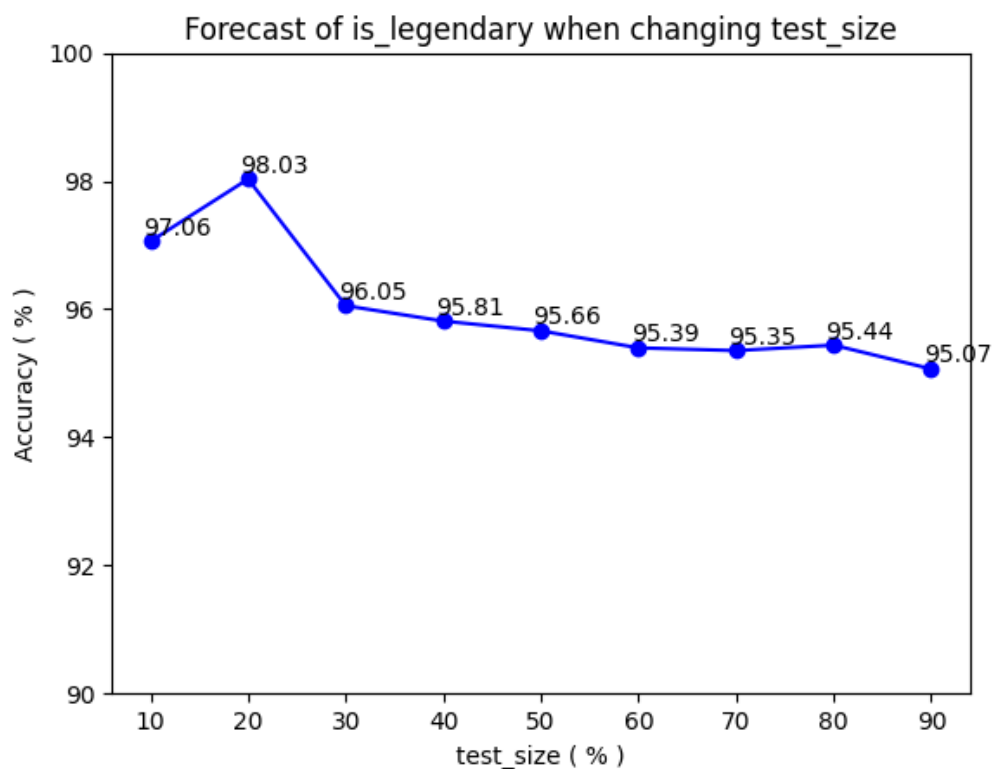


观察发现 KNN 算法的正确率最高，于是对 KNN 算法进行调参，观察正确率的变化

## 4.3 KNN 调参

### 4.3.1 研究不同的 test\_size 对预测结果的影响

```
71 #改变test_size的取值，观察对正确率的影响
72 knn = KNeighborsClassifier()
73 y_list = np.array([]) # 存储正确率
74 test_list = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]) # 不同的test_size
75 for i in range(9):
76     X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_list[i], random_state=18) # 划分数据集
77     knn.fit(X_train, y_train.values.ravel()) # 训练
78     y_KNCpred = knn.predict(X_test) # 计算正确率
79     y_list = np.append(y_list, [metrics.accuracy_score(y_KNCpred, y_test) * 100]) # 添加点值
80 for i, v in enumerate(y_list):
81     plt.text(test_list[i]*100-0.8, v+0.1, '%.2f' % v)
82 plt.plot(test_list * 100, y_list, 'b-o') # 绘制折线图
83 plt.title('Forecast of ' + target + ' when changing test_size')
84 plt.xlabel('test_size ( % )')
85 plt.ylabel('Accuracy ( % )')
86 plt.ylim(90, 100)
87 plt.show()
```



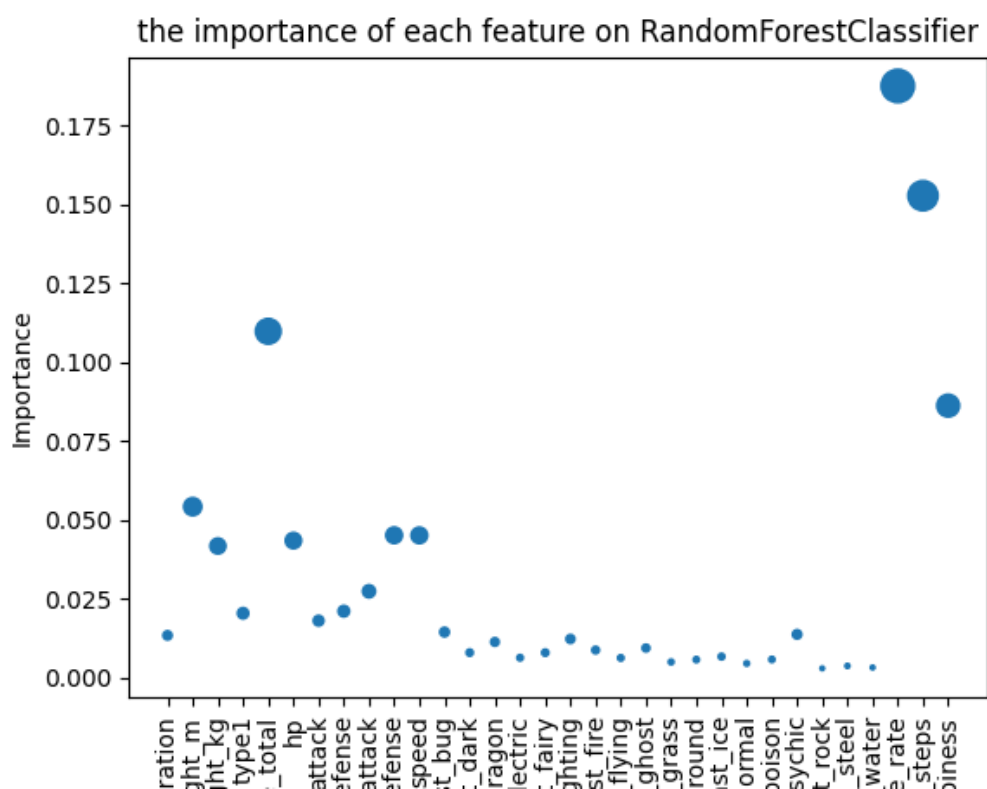
发现当 test\_size 取 0.2 时，预测的准确率最高

## 4.4 探究预测过程中，各个特征的重要性

### 4.4.1 随机森林模型评估特征重要性

```
91 # 训练随机森林模型
92 model = RandomForestClassifier(n_estimators=100, random_state=42)
93 model.fit(X_train, y_train)
94
95 # 输出特征重要性
96 importance = model.feature_importances_
97 indices = np.argsort(importance)[::-1]
98
99 # 绘制散点图显示每个特征的重要性
100 importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': model.feature_importances_})
101 print(importance_df)
102 plt.scatter(X.columns, importance, s= list(map(lambda x: x * 1000, importance)) )
103 plt.xlabel('Feature')
104 plt.ylabel('Importance')
105 plt.xticks(rotation=90, fontsize=10)
106 plt.title('the importance of each feature on RandomForestClassifier')
107 plt.show()
108
```

各特征在预测算法中重要性如图



由图可以得出在预测宝可梦是否为传奇宝可梦的过程中,较为重要的几个特征及其对应的重要性分别为

capture_rate	0.187650
base_egg_steps	0.152833
base_total	0.109798
base_happiness	0.086215

## 5.总结

### 5.1 数据可视化部分

本次项目对各种族宝可梦的血量、攻击力、防御力、特殊攻击、特殊防御、速度六个基础值, 每代宝可梦的数量, 每代传奇宝可梦的数量, 以及宝可梦的战斗能力进行了简要分析。

### 5.2 预测部分

采用了三种预测模型, 通过宝可梦的各项特征对是否为传奇宝可梦进行预测。默认 test\_size 为 0.5 时, KNN 模型的准确率最高。对 KNN 进行参数调整, 当 test\_size 取 0.2 时准确率最高。并且对预测过程中各特征的重要性进行分析, 其中较为重要的几个特征依次为 capture\_rate、base\_egg\_steps、base\_total、base\_happiness。