基于神经网络模型的 5v5 moba 游戏胜率预测

AAHoldingAces

摘要

本文通过使用给定的数据集，对比逻辑回归模型和全连接型神经网络模型的准确度，用基于全连接神经网络的模型进行监督学习，并在调节了最优参数后，预测了 5v5 moba 游戏获胜的概率值并绘制 ROC 曲线。

关键词

全连接神经网络 机器学习 监督学习 Python

一、准备工作

本文使用的库有 numpy、pandas、matplotlib.pyplot 和 seaborn，并读取 train 和 test 数据，具体代码如下：

In [1]:

```
#导入相关库
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
#读取数据
train_df = pd.read_csv('train.csv')
```

```
test_df = pd.read_csv('test.csv')
```

In [3]:

```
#train 数据集的维度
train_df.shape
```

Out[3]:

```
(82355, 117)
```

In [4]:

```
#提取 train 数据集中的因变量 y
train_df_copy = train_df.copy()
label = train_df.pop('y')
#把 train_df 和 test_df 拼接，可以同时进行数据清洗
df = pd.concat([train_df, test_df], keys=['train', 'test'])
df.shape
```

Out[4]:

```
(101319, 116)
```

二、数据预处理

(一) 数据清洗

观察原数据集，可以发现存在以下问题：

1．少部分数据存在缺失（为空值或非数值信息）；

2．部分数据存在异常值（从第 3-117 列的英雄选择数据中，只应包含 1、-1 和 0 三种，而不应该存在其他数据，如 100）；

3．不能直观确定每行数据符合游戏的 5v5 标准（即每行代表一局比赛，则我方应选 5 名英雄，对方应选 5 名英雄，则应满足每行数据包括 5 个 1 和 5 个-1，其余数据均为 0）；

4．此外，有些英雄从未被选择过，故无法提供有效信息，需要进行处理。

对此，进行以下数据清洗工作，具体代码如下：

In [5]:

```
#检查空值存在的列
columns_isull = df.columns[df.isnull().sum() != 0]
```

columns_isull

Out[5]:

Index(['Hero_19', 'Hero_33', 'Hero_34', 'Hero_37', 'Hero_38', 'Hero_39'], dtype='object')

在检查空值过后可以发现有几列存在数据为空的现象，但是不能确定填补空值的数据应该为 0，1 还是-1，故应该对数据进行进一步检查以确保英雄的数量正确。

In [6]:

```
#检查每局比赛的数据，两队的英雄数量是否正确（都应该等于 5）
df['my_heros'] = 0
df['enemy_heros'] = 0
for index, row in df.iterrows():
    row_list = row.to_list()[3:-2]
    df.loc[index, 'my_heros'] = row_list.count(1)   #检查我方英雄数量
df.loc[index, 'enemy_heros'] = row_list.count(-1) #检查敌方的英雄数量
```

In [7]:

```
#检查后发现没有不合规的数据
df[(df['my_heros']!= 5) | (df['enemy_heros']!= 5)]
```

Out [7]:

0 rows × 118 columns

故此时可以确定应该将缺失值填补为 0，并将异常值 100 更改为 0

In [8]:

```
#填补空缺值为 0
df.fillna(0, inplace=True)
```

In [9]:

```
#修改异常数据为 0
rows, cols = np.where(df.values==100)
for index, value in enumerate(rows):
df.iloc[value, cols[index]] = 0
```

之后可以查看从未被选择过的英雄，这些英雄对于胜率预测的参考价值很小，故可以被去除。

```
In [10]:
#检查是否存在英雄从来没有被选择过
delete_col = []
for col in df.columns:
    col_0 = df[col].to_list().count(0)
    if col_0 == df.shape[0]:
        delete_col.append(col)
delete_col   # 24 号和 108 号英雄从来没有被选择过
Out [10]:
['Hero_24', 'Hero_108']
```

```
In [10]:
#检查是否存在英雄从来没有被选择过
delete_col = []
for col in df.columns:
    col_0 = df[col].to_list().count(0)
    if col_0 == df.shape[0]:
        delete_col.append(col)
delete_col   # 24 号和 108 号英雄从来没有被选择过
Out[10]:
['Hero_24', 'Hero_108']
```

```
In [11]:
#删去这两个英雄所在的列
df.drop(delete_col, axis=1, inplace=True)
```

(二)数据可视化

为了能更直观地发现数据的特征，方便进行模型选择，本文对常用的英雄选择、数据中的队伍分布、训练集和测试集的队伍分布分别进行了数据可视化，以确定需不需要进一步的数据处理。

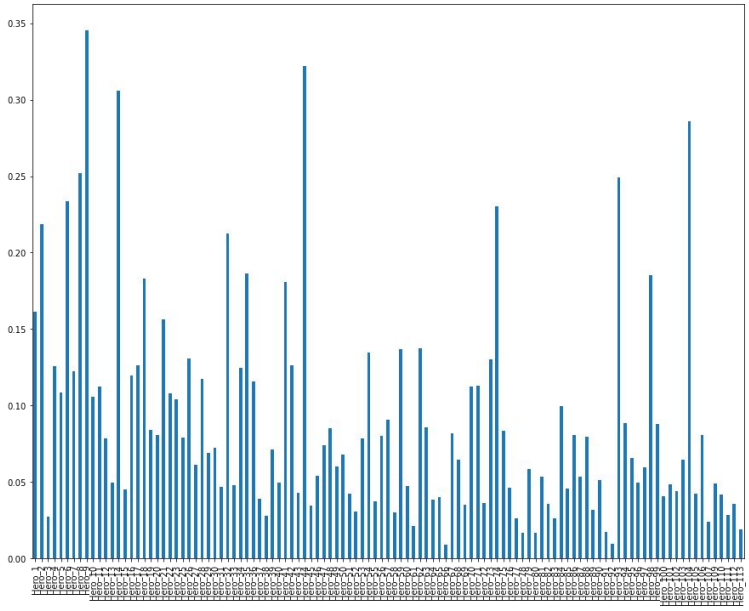

图 1 常用英雄选择

1. 查看常用的英雄选择

根据可视化结果可以发现英雄的选择概率最大不超过 0.35 且需要将数据分组寻找规律。

2. 查看数据中队伍的分布

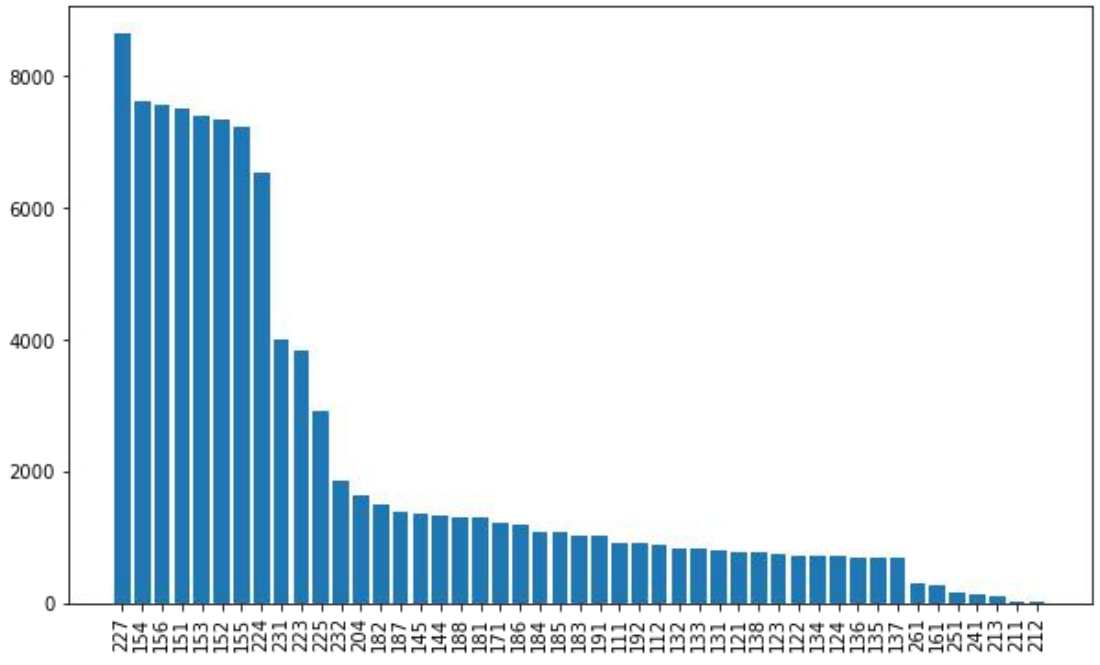

图 2 队伍分布

将训练集和测试集中的队伍分布进行对比，可以发现两个数据集中的队伍分布基本一致，没有明显的差别。

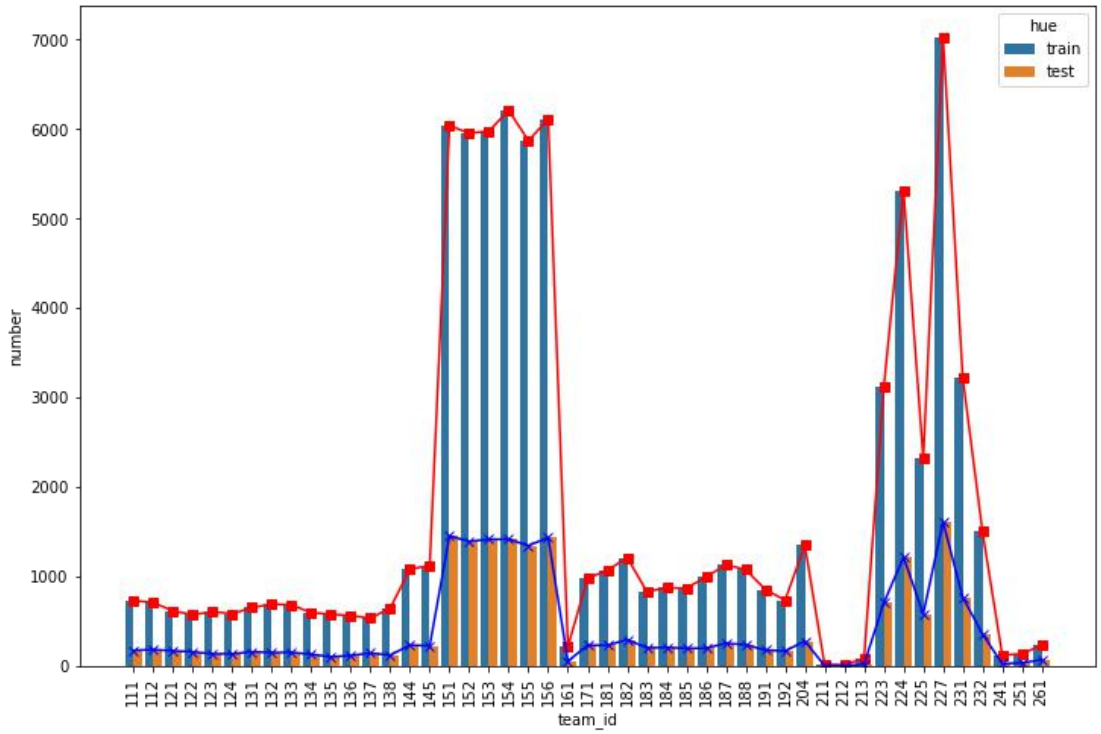

图 3 测试集与训练集队伍分布

3. 胜负场可视化

(1) 不同队伍的胜负场次

可以发现胜负场次基本表现出胜场多余负场的形态，总的来说并 0 和 1 之间的分布比例不存在巨大的差别。

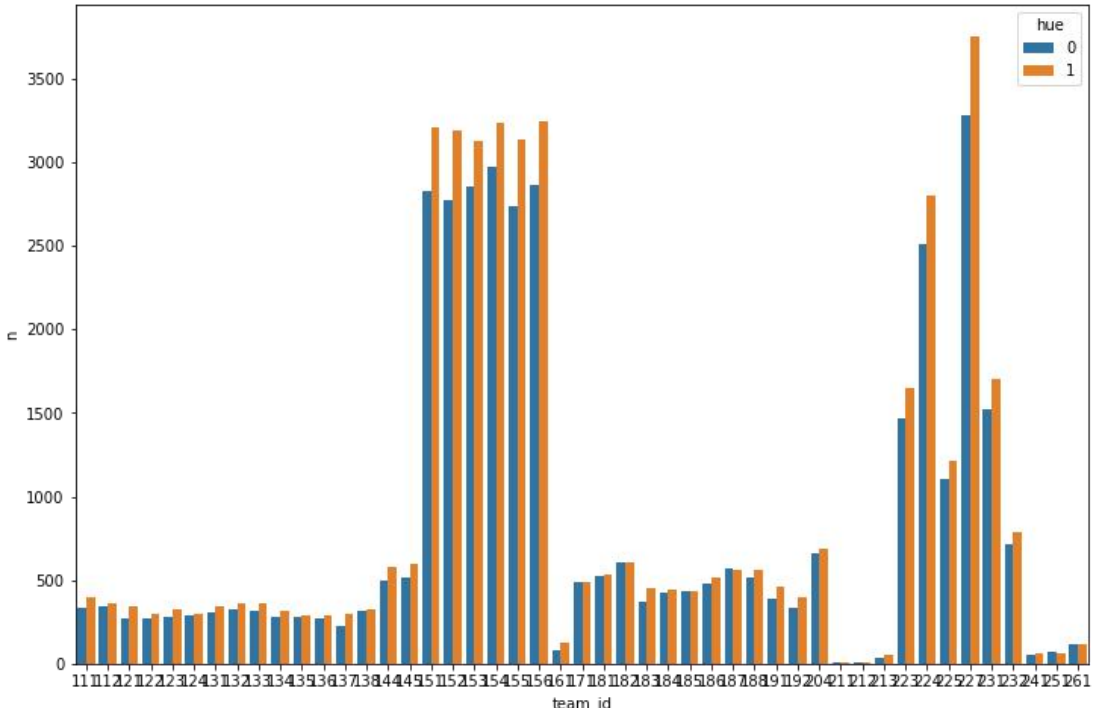

图 4 队伍的胜负场次

考察不同比赛类型和模式可以得到类似的胜负场次分布情况。

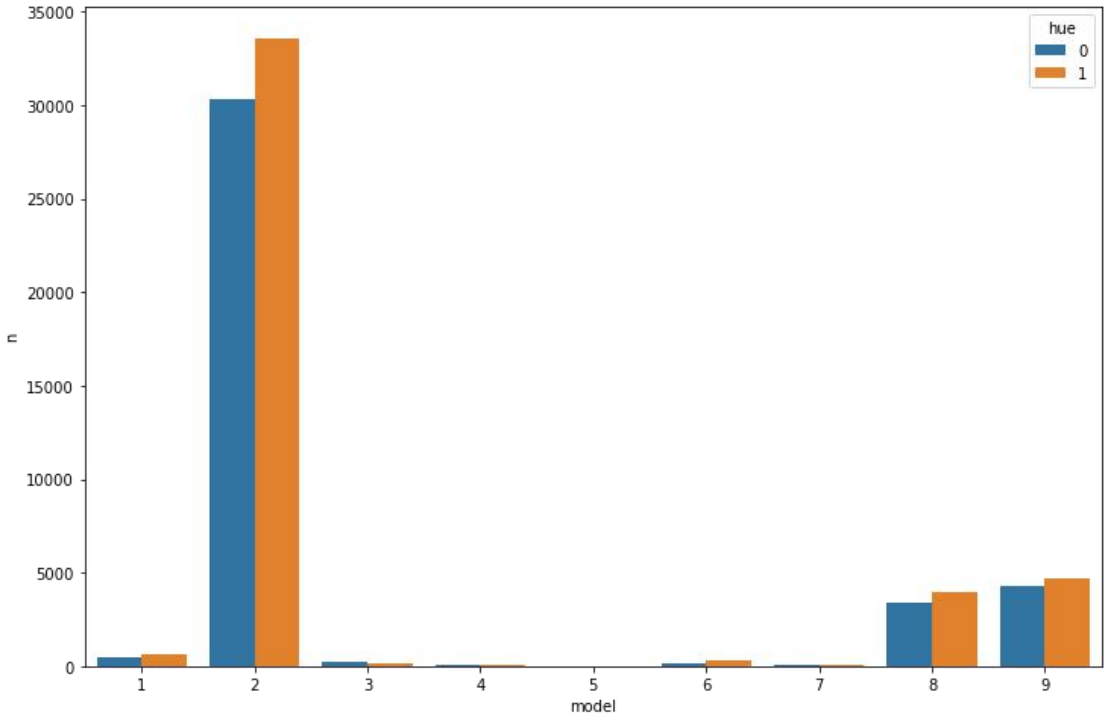

图 5 比赛模式的胜负场次

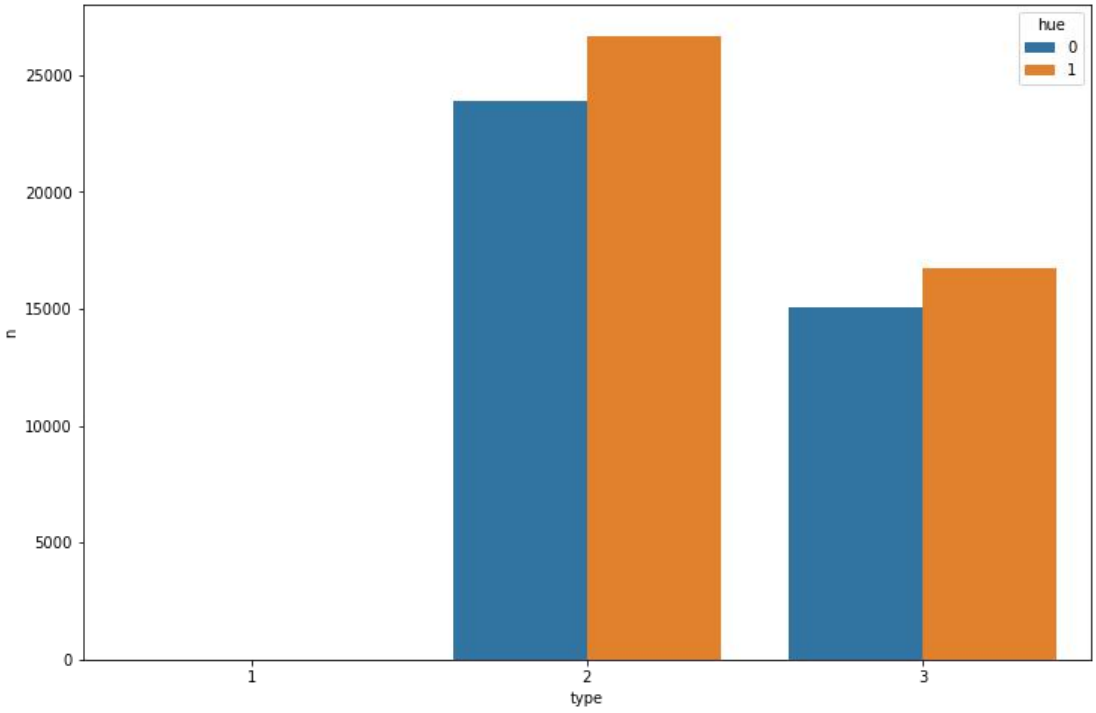

图 6 比赛类型的胜负场次

三、建立模型

本文根据预测值的特征和数据的分布情况，选择了两种模型对数据进行预测，并选取了其中准确度较高的模型进行回归预测。

(一) 逻辑回归

选择使用逻辑回归是因为输出的预测值为二值变量，符合逻辑回归的原理。

1. 逻辑回归的原理

Logistic Regression 虽然被称为回归，但其实际上是分类模型，并常用于二分类。Logistic 回归的本质是：假设数据服逻辑分布，然后使用极大似然估计做参数的估计。

Logistic 分布是一种连续型的概率分布，其**分布函数**和**密度函数**分别为

$$F(x) = P(X \le x) = \frac{1}{1+e^{\frac{-(x-\mu)}{\gamma}}}$$

$$f(x) = F'(x) = \frac{e^{\frac{-(x-\mu)}{\gamma}}}{\gamma(1+e^{\frac{-(x-\mu)}{\gamma}})^2}$$

其中，μ表示**位置参数**，$\gamma > 0$ 为**形状参数**。我们可以看下其图像特征：

图 7 逻辑分布的密度函数和分布函数

Logistic 分布是由其位置和尺度参数定义的连续分布。Logistic 分布的形状与正态分布的形状相似，但是 Logistic 分布的尾部更长，所以我们可以使用 Logistic 分布来建模比正态分布具有更长尾部和更高波峰的数据分布。在深度学习中常用到的 Sigmoid 函数就是 Logistic 的分布函数在$\mu = 0, \gamma = 1$ 的特殊形式。

回归结果如下所示：

In [12]:

```
# 调整前模型
l_model = LogisticRegression()
l_model.fit(train_x, train_y)
predict = l_model.predict(train_x)
fpr, tpr, thresholds = roc_curve(train_y, predict, pos_label=1)
```

```
auc(fpr, tpr)
```

```
Out [12]:
```

```
0.5962886571103336
```

```
In [13]:
```

```
# 调整后的逻辑回归模型
adjusted_l_model = LogisticRegression(solver='sag', C=0.1)
adjusted_l_model.fit(train_x, train_y)
predict = adjusted_l_model.predict(train_x)
fpr, tpr, thresholds = roc_curve(train_y, predict, pos_label=1)
auc(fpr, tpr)#逻辑回归的效果一般
```

```
Out [13]:
```

```
0.5962642993105929
```

(二) 全连接神经网络

在调整参数后逻辑回归的 auc 值仍然没有较大变化，且拟合效果较为一般，故更换模型进行预测。本文在尝试选择多种模型后选择全连接神经网络模型进行预测。

3.2.1 全连接神经网络的原理

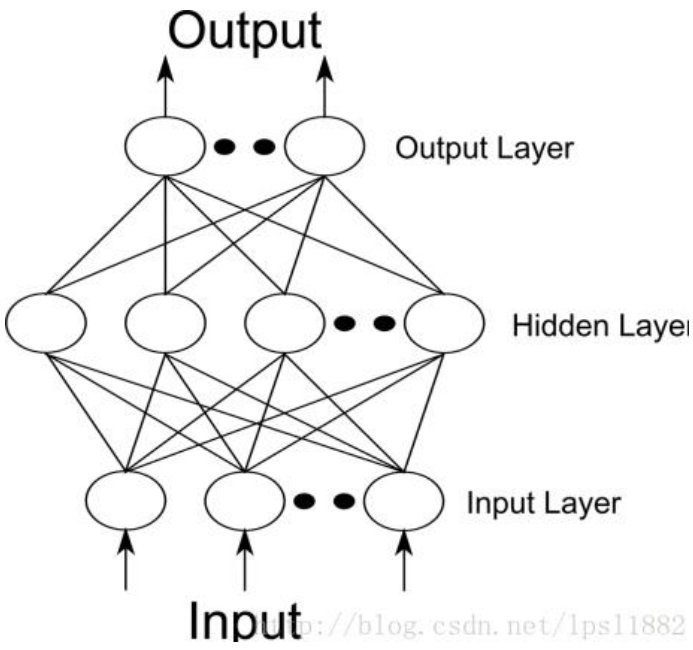

图 8 全连接神经网络的特点

如上图，是一个简单的分类全连接神经网络，每一根线表示权重相乘。如果没看透这个计算关系，那么我们很容易这样设计程序：每一个节点都制作一个

class 类，每个类节点要与其他类节点有连接关系，要传递数据，要计算梯度误差。

假设只研究两层数据之间的关系，每根线代表一个权重乘法。设权重为w_{ij}^1，前一层数据是x_i^1，后一层数据为x_j^2，那么后一层每一个数据就是$x_i^2 = \sum i\, x_i^1 w_{ij}^1 + b_i^1$。为了形象表示数据从左向右传递的过程，写成

$$\begin{bmatrix} x_1^1 & x_2^1 & \dots \end{bmatrix} \begin{bmatrix} w_{1j}^1 \\ w_{2j}^1 \\ \dots \end{bmatrix} + b_1^1$$

如果我们扩展计算整个全连接层，就可以得到简单的矩阵计算：

$$\begin{bmatrix} x_1^1 & x_2^1 & \dots \end{bmatrix} \begin{bmatrix} w_{11}^1 & w_{12}^1 & \dots \\ w_{21}^1 & w_{22}^1 & \dots \\ \dots & & \end{bmatrix} + \begin{bmatrix} b_1^1 & b_2^1 & \dots \end{bmatrix} = \begin{bmatrix} x_1^2 & x_2^2 & \dots \end{bmatrix}$$

$$\rightarrow X^1 W^1 + B^1 = X^2$$

所以，一个全连接层，实际上就是一个矩阵乘法和一个矩阵加法，数据都是向量，权重参数可以用矩阵表示，计算过程能够编码为一个 layer 类，即一个神经网络层。相应的，激活层其实就是对数据向量，逐个元素进行计算，即

$$Activation(X^2) = \begin{bmatrix} f(x_1^2) & f(x_2^2) & \dots \end{bmatrix}$$

这两种操作都可以向量化，下面关键就是如何计算梯度来修正权重，梯度下降法的定义是 $\theta_{new}=\theta_{old} - \frac{\partial F}{\partial \theta}$。矩阵求导的公式(注意上标 2 并不是平方，是序号)是：

$$\frac{\partial X^2}{\partial W^1} = (X^1)^T$$

$$\frac{\partial E}{\partial B^1} = I$$

其中，设损失函数为 E。

集体的建立模型过程如下：

In [14]:

```
#导入建模所需库
from sklearn.linear_model import LogisticRegression
import tensorflow as tf
```

```
from tensorflow.keras import Sequential, layers
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, auc, roc_curve
```

In [15]:

```
#所有数据+1 之后，所有数据的-1 都变为 0，1 变为 2，方便建立模型
df = df + 1
train_x, train_y = df.loc['train'].iloc[:, :-2], label #  模型拆分
test_x = df.loc['test'].iloc[:, :-2]

#  对数据标准化
std = StandardScaler()
std.fit(train_x)
train_x = std.transform(train_x)
test_x = std.transform(test_x)

train_x.shape, train_y.shape, test_x.shape
```

Out [15]:

```
((82355, 114), (82355,), (18964, 114))
```

In [16]:

```
#调整参数前
model = Sequential(
    [
        layers.Dense(16, activation='relu'),
        layers.Dense(2, activation='sigmoid')
    ]
)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']) #  配置损失函数和优化器
model.fit(train_x, tf.one_hot(train_y,depth=2), batch_size=16, epochs=10, validation_split=0.2, verbose=0) #  训练模型

#计算 auc 值
y1_lr1 = model.predict(train_x)[:, 1]
```

```
fpr_lr1, tpr_lr1, thresholds_lr1 = roc_curve(train_y, y1_lr1)
roc_auc_lr1 = auc(fpr_lr1, tpr_lr1)
roc_auc_lr1
```

```
Out [16]:
0.6573112245705339
```

```
In [17]:
# 调整参数后的模型
# 主要调整了网络结构，batch size 和 epochs
adjusted_model = Sequential(
    [
        layers.Dense(32, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(2, activation='sigmoid')
    ]
)

adjusted_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']) # 配置损失函数 和 优化器
adjusted_model.fit(train_x, tf.one_hot(train_y,depth=2), batch_size=64, epochs=40, validation_split=0.2, verbose=0) # 训练模型

#计算 auc 值
y1_lr2 = adjusted_model.predict(train_x)[:, 1]
fpr_lr2, tpr_lr2, thresholds_lr2 = roc_curve(train_y, y1_lr2)
roc_auc_lr2 = auc(fpr_lr2, tpr_lr2)
roc_auc_lr2
```

```
Out [17]:
0.7368710237884647
```

综上，以调整后的神经网络模型作为预测模型，并输出预测概率值。[①]

[①] 输出的预测概率值可参照源码文件。

四、绘制 ROC 曲线

In [18]:

```
#绘制 ROC 曲线
plt.plot(fpr_lr1, tpr_lr1, fpr_lr2, tpr_lr2, lw=2, alpha=.6)
plt.plot([0, 1], [0, 1], lw=2, linestyle="--")
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.legend(["ANN_1    (AUC {:.4f})".format(roc_auc_lr1),
            "ANN_2    (AUC {:.4f})".format(roc_auc_lr2)], fontsize=8, loc=2)
```

Out [18]:

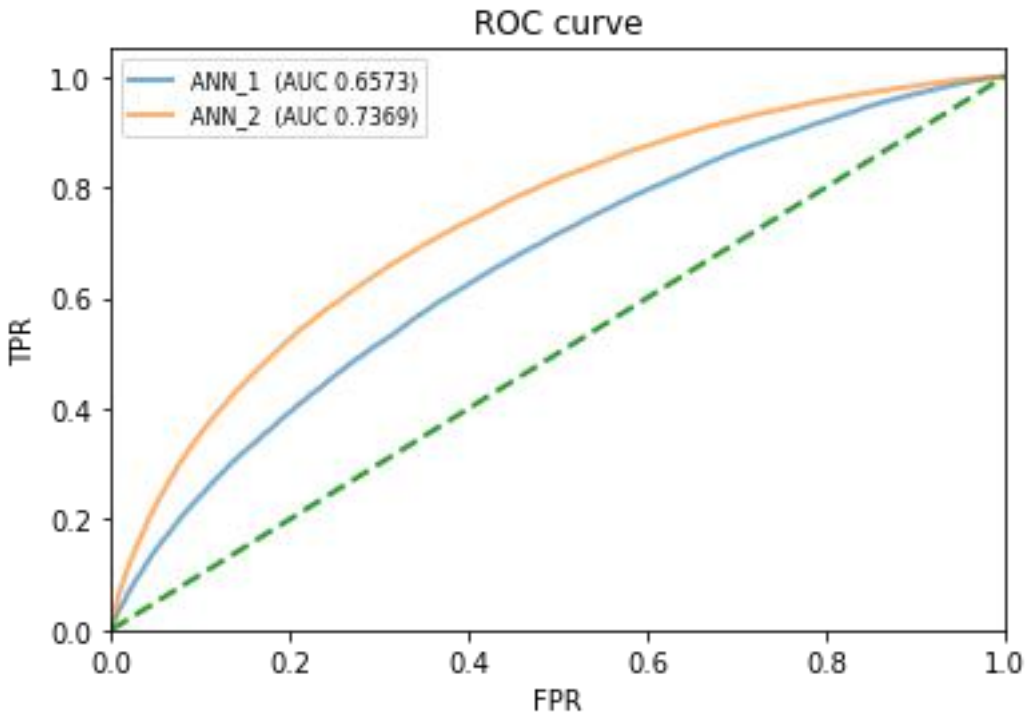

图 9 ROC 曲线

五、参考资料

[1] 全连接神经网络：https://itpcb.com/a/579155

[2] 全连接神经网络：

https://blog.csdn.net/baidu_36602427/article/details/96429838

[3] Tensorflow 多层全连接神经网络：
https://www.cnblogs.com/expedition/p/11627079.html

[4] 【机器学习】动手写一个全连接神经网络：
https://blog.csdn.net/lpsl1882/article/details/53790716/