

数据预测Dota2游戏结果



That was skill...Not luck!

—Miarana

何凌云 刘兴阳

2017年6月1日

目录

摘要	3
背景	4
数据分析	5
数据预处理	8
分类器	14
模型评估	18
总结	20

摘要

在这篇文章中，我们基于Dota2 Game Results数据集，对于数据进行预处理并分类，试图基于双方的英雄选择预测游戏结果，并对模型和预测结果进行了系统的评估。

我们充分利用了数据集中的数据，先对数据进行了直观地处理，建立了几个模型来评估英雄以及队伍的强度，将原本有116个属性的数据缩减到6个属性，甚至缩到2个属性，从而使数据适合被分类器分类。

在分类器的选择上我们综合比较了各个分类器之间的优缺点，最终决定用Logistics Regression（LR）进行分类。我们根据LR的原理自己实现了分类器，并且经过测试发现分类效果已经达到了sklearn库中的逻辑回归分类器的水平。

我们对我们的模型进行了系统的评估，除了计算准确率等基础评价标准之外，我们还绘制了ROC曲线并用Cross Validation对模型进行了进一步的评估。

Abstract

In this essay, we made preprocess and classification on the dataset of Dota2 Game Results. We tried to predict the results according to how the teams choose their heroes and gave a systematic evaluation of the model and prediction results.

We made full use of the given data. We built several models to evaluate the strength of heroes and reduced the number of properties to 2 so that the data could be more suitable to the classifier.

We compare the pros and cons of different classifiers and eventually choose Logistics Regression to make the classification. We built the classifier ourselves according to its principle and after we tested our program, we found that the results of classification were as good as that of the LR function in the sklearn library.

Then we made a systematic evaluation of the model through the calculating accuracy. We also drew the ROC curve and utilized the cross validation method to make a further evaluation.

背景

Background

Dota2是一款由Valve开发免费的多人在线战斗竞技类游戏。2016年，在时代周刊列举的49款史上最好的电子游戏中，称DOTA2为多人在线竞技游戏类的巅峰之作。同年，PC GAMER列举的第十二届最好的100款电子游戏中，称DOTA2为最有深度和竞技体验最佳的游戏。因此Dota2游戏具有极强的竞技性，每年也会举行许多重量级的电竞比赛。尽管Valve在不断更新维护、调整英雄的实力，但游戏的平衡性是玩家最关心的一点，再平衡的游戏也无法做到所有的英雄都实力平均，因此只能通过版本更新来达到动态平衡的效果。正是由于这一点，英雄的选择对于游戏的胜负有着至关重要的影响。一支专业的Dota2队伍肯定会对比赛数据进行收集和处理，以帮助他们选择正确的英雄阵容并获得更好的成绩。

我们这次的题目主要聚焦在了英雄选择对比赛胜负的影响，并通过建立模型来预测比赛的胜负。虽然说选择英雄不是游戏的全部，但根据双方的英雄选择，往往就能预测出胜负的概率。

我们用的数据集是UCI Machine Learning Repository的Dota2 Games Results Data Set。根据数据集的介绍，这个数据集的数据采集于2016年8月3号，所对应的是Dota2的TI版本，Dota2历史上最平衡的版本之一，用这个版本作为比赛胜负的预测是再合适不过了。

数据集的内容包括比赛的胜负（1或-1）、服务器编号、游戏模式（比如队长模式）、比赛类型（比如排位赛）以及双方所选择的英雄。具体数据格式见下表：

比赛胜负	服务器编号	游戏模式	比赛类型	英雄1	英雄2	...	英雄113

数据分析

Data Analysis

在上一章节中我们介绍了数据集中有哪些数据，经过分析我们发现所有的数据中，游戏模式、比赛类型等数据对于双方的队伍都是一样的，因此肯定不能作为推断游戏胜负的依据，只有双方选择的英雄是我们能够获得的对于胜负具有参考依据的数据。因此我们主要根据双方的英雄选择来推断比赛结果，在正式开始处理数据之前我们先提出如下合理假设：

1. 游戏模式和比赛类型只会影响英雄选择的决策，游戏开始后不会影响游戏的进行和结果；
2. 玩家水平从宏观上来看没有明显波动，在大量的游戏统计下，训练集和测试集的胜率统计基本保持一致；
3. 服务器的不同可能会对玩家的英雄偏好有影响，但不存在某个服务器的玩家玩某个英雄胜率特别高的情况；
4. 由于版本的缘故，不同的英雄存在一定的实力差距（或者说，对于平均水准的玩家有实力差距），并会对比赛结果造成影响。

当然，我们手上拿到的是原始数据，根据以上假设我们在正式进行分类之前需要对数据进行预处理。首先我们列出要用到术语列表。

符号	含义
WR	胜率，获胜场次 ($\text{win}(A)$) / 出战场次 ($\text{selected}(A)$)
AR	协同系数，用于衡量英雄间的配合程度
OR	克制系数，用于衡量英雄间的克制程度
TWR、TAR、TOR	队伍的胜率、协同系数、克制系数
$x^{(i)} = (x_1, x_2, \dots, x_n)$	数据集中的第 <i>i</i> 个点，有 <i>n</i> 个属性

首先我们想到的当然是看看这个版本哪个英雄比较强势，即有他的队伍胜率比较高。因此我们对训练集的数据进行了简单的统计分析，分别

分析了每个英雄的出场率和胜率。我们称这是英雄的基础数据，决定了英雄在平均水准下的能力发挥，是我们在数据分析中对一个英雄能力评估的基准值。这是非常重要的，根据我们的假设，游戏中的英雄存在强弱差别，胜率是衡量游戏中英雄强弱的重要指标。胜率（WinRate）的定义非常简单：

$$WR(A) = \frac{win(A)}{selected(A)}$$

根据我们的假设，我们在宏观上不考虑除了英雄选择之外的因素对比赛胜负的影响，因此出场率对于我们的比赛结果预测不是最重要的。胜率才是衡量一个英雄强弱的标准。

除了基础数据，还有其他因素会影响一支队伍的強大程度：

1. 己方英雄对方英雄的克制；
2. 己方英雄之间的协同效应(比如说能打出一些精彩的COMBO)；
3. 团队资源的分配（也就是不同英雄的位置分配）

第1点我们主要通过两个英雄之间的对阵情况分析，如果英雄A对阵英雄B的实际胜率比A的期望胜率高出许多，我们就认为英雄A克制英雄B。第2点和第3点我们认为都是体现在两个英雄的搭档胜率上的，如果两个英雄之间的协同效应强，他们就能对他们的团队贡献更多的胜率；如果两个英雄的位置冲突，他们的搭档就会降低团队的胜率。因此，我们就不对这两种情况进行区分了。

那我们如何判断两个英雄之间的克制关系呢？首先，设A、B两个英雄的基础胜率为 p_A 和 p_B ，通过条件概率我们可以算出理论上不考虑克制关系时英雄A战胜英雄B的概率 p_{AB} 。这个概率是一方对另一方的获胜期望概率，即根据平日成绩而不考虑对手的特殊情况下计算出来的。当两个英雄独立地进行两场比赛，他们的胜负关系可以用下表的公式算出来；然而，当两个英雄在同一场比赛中同时出现时，他们之间的胜负不再是独立事件，因此我们要用条件概率来计算。

A\B	Win	Lose
Win	$p_A \times p_B$	$p_A \times (1 - p_B)$
Lose	$(1 - p_A) \times p_B$	$(1 - p_A) \times (1 - p_B)$

当两个英雄是对手时，只可能出现Win/Lose和Lose/Win两种情况。因此我们可以推导出当两个英雄交手时一方对另一方的胜率 $p_{A,B}$ ，如下面的公式所示。

$$p_{A,\bar{B}} = \frac{p_A - p_A \times p_B}{p_A + p_B - 2 \times p_A \times p_B}$$

如果两个英雄交手多次，而各自胜负都与全局胜率相当，那就说明双方都是正常发挥，谁也没有压制谁。而如果两个英雄交手时，英雄B表现出来的胜率显著低于 $p_{A,B}$ ，这说明英雄A对英雄B有明显的克制，从而让英雄B不能发挥正常的胜率水平。比如，一个在该版本非常强势的英雄B有65%的胜率而英雄A只有35%的胜率，则正常情况下英雄A对英雄B交手的胜率应该在19%左右，但是英雄A对英雄B却打出了40%的胜率，这说明英雄A别的不行，克制英雄B还是有一手的。因此我们引入克制系数（Overcome Rate）， $\hat{p}_{A,B}$ 是英雄A对英雄B的实际胜率：

$$OR(A, B) = \frac{\hat{p}_{A,B} - p_{A,B}}{p_{A,B}}$$

同理，对于英雄A对英雄B的协同性我们也可以用Associate Rate来衡量。当两个英雄处于同一支队伍时，不考虑互相之间的协同效应，他们队的期望胜率：

$$p_{A,B} = \frac{p_A \times p_B}{1 - p_A - p_B + 2 \times p_A \times p_B}$$

由此协同系数为（ $\hat{p}_{A,B}$ 是英雄A与英雄B互为队友时的实际胜率）：

$$AR(A, B) = \frac{\hat{p}_{A,B} - p_{A,B}}{p_{A,B}}$$

有了以上的三个主要参数，我们就可以定量分析个体英雄在一局游戏中的胜利贡献值的期望了。接下来我们将会讨论如何对数据预处理从而放入分类器中进行计算。

数据预处理


Data PreProcess

在上一章节中我们对数据进行了分析，得到了很多直观的结论，在这一章中我们将利用我们在上一章中建立的模型来对数据进行预处理。

首先我们观察源数据集可以发现一个特点，数据属性非常多而且稀疏，如果我们直接把这些属性当做特征扔进分类器进行计算，效率会非常低而且准确率也不够高。因此，我们要预先对数据进行抽象，精简属性。

首先，我们要算出所有英雄的基础胜率，统计整个训练集中所有英雄的胜率，并存在basic_rate.csv中。具体的计算过程详见Basic_Rate函数。







表格 1-1：英雄基础数据统计（胜率前十）

英雄名	出场数	胜场数	出场率	胜率
 Enchantress	8320	5063	8.9800%	60.8534%
 Legion Commander	5959	3490	6.4317%	58.5669%
 Warlock	10729	6056	11.5801%	56.4451%

Doom	7601	4210	8.2040%	55.3874%
Death Prophet	11650	6448	12.5742%	55.3476%
Mirana	32048	17729	34.5904%	55.3201%
Outworld Devourer	7780	4274	8.3972%	54.9357%
Magnus	4621	2531	4.9876%	54.7717%
Crystal Maiden	10050	5504	10.8473%	54.7662%
Elder Titan	4079	2230	4.4026%	54.6703%





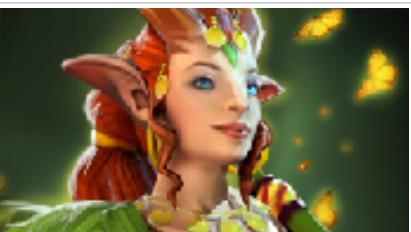

接着，我们在基础胜率的基础上进一步算出英雄两两之间的协同系数和克制系数。首先，我们先算出了英雄两两之间的交手胜率以及队友胜率，然后利用上一章节推导的公式得出了协同系数和克制系数。

表格 1-2：英雄克制排行-1

克制英雄	被克制英雄	克制率 (%)
 Chen	 Batrider	71.3791
 Winter Wyvern	 Wisp	69.8454
 Chen	 Legion Commander	62.0440
Chen	Dragon Knight	58.1132
Shadow Demon	Warlock	57.3783
Bristleback	Winter Wyvern	54.2626
Chen	Skywrath Mage	52.7308
Shadow Demon	Brewmaster	49.3139

Shadow Demon	Ember Spirit	48.2980
Night Stalker	Chen	48.1185

表格 1-2 英雄协作排行

协作英雄A	协作英雄B	协同率 (%)
 Kunkka	 Omniknight	110.89960085968700
 Abyssal Underlord	 Omniknight	110.89960085968700
 Enchantress	 Omniknight	104.38771285869900
Oracle	Omniknight	98.399944013931700
Chen	Centaur Warrunner	86.691591297663100
Queen of Pain	Omniknight	84.094651432121400
Kunkka	Elder Titan	82.705548805184300
Abyssal Underlord	Elder Titan	82.705548805184300
Visage	Omniknight	82.114325079496400
Wisp	Abaddon	79.22629604007130

为了更直观地表现所有英雄之间的克制与协同关系，此处均用 BetterRate 表示，我们做了更加清晰的三维散点图，其中 X-Y 轴分别代表 A、B 两个英雄，Z 轴表示两个英雄间的克制系数或协同系数。对于散点我们用三种颜色进行标识，黄色表示没有明显的克制/协同倾向；红色表示

英雄A明显克制/协同英雄B；蓝色表示英雄A明显被英雄B克制/英雄A与英雄B不兼容。

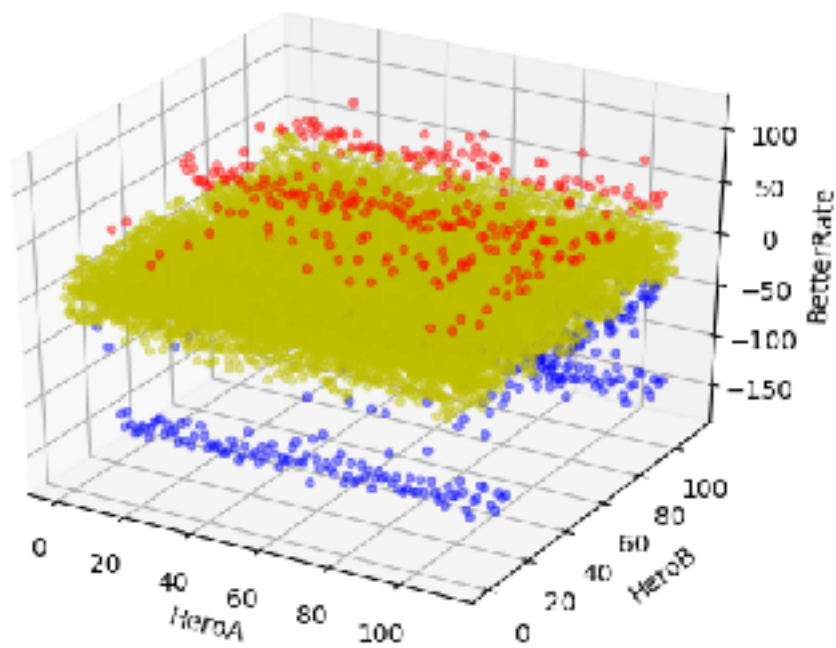


图1.1：英雄间的克制系数

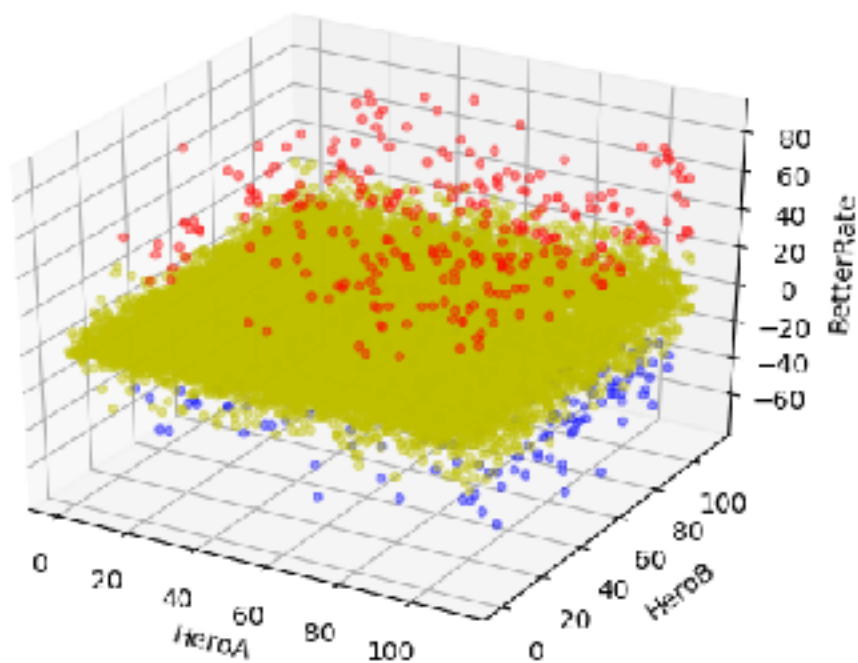


图1.2：英雄间的协同系数

从上图我们可以看出，大部分的英雄都是处于相对比较均衡的状态，但也有少部分的英雄存在比较明显的克制关系。

现在，我们得到了完整的英雄与英雄之间的强弱模型，但是DOTA2是一个团队游戏，我们要如何才能评估一个队伍的强弱程度呢？

第一个想法是我们直接把两队是个英雄的个人强弱属性放进分类器由分类器自己判断关系。这样的方法的属性有：

1. 两队英雄的基础胜率：10
2. 同队英雄之间的两两协同系数： $2 \times C_5^2 = 20$
3. 两队英雄之间的两两克制系数： $5 \times 5 = 25$

共计有55个属性，显然对于分类器而言还是太多了，所以不是一个好主意，我们最好继续化简属性。

首先，不考虑对手的情况，一个队伍的基础强度主要取决于英雄的基础胜率和队伍协同胜率。由于在同一队当中我们无法判断哪儿英雄具有特殊的地位，所以我们暂且认为英雄地位均等，于是我们采用平均值的方式估计一个队的基础胜率（TWR）和协同系数（TAR）。

$$TWR = \frac{1}{5} \sum_{i=1}^5 WR(i) \quad TAR = \frac{1}{25} \sum_{i=1}^5 \sum_{j=1}^5 AR(i, j)$$

其次，考虑对手之后，我们认为一个聪明的队伍会在游戏中最大化地发挥自己的优势，当发现自己有英雄克制对方的英雄时，会尽量为自己的英雄创造跟对方被克制英雄单挑的机会。所以我们选择用最大值来衡量一个英雄对对面队伍的克制情况，如下为一个队伍的克制系数（TOR）。

$$TOR = \frac{1}{5} \sum_{i=1}^5 \max_{j=1}^5 (OR(i, j))$$

于是一个队伍的强弱我们就可以用非常简洁的三个参数来描述了： $Team(TWR, TAR, TOR)$ ，两个队伍共计6个特征，已经是接受的数据量了，我们可以把这个数据放进分类器进行运算了。

但是，6维的数据仍然不够直观，我们能不能把数据降到2维呢？因此我们又引入了一个更简洁的模型，用一个数T去描述一支队伍的情况，综合考虑TWR、TAR和TOR： $T = TWR \times (1 + TAR) \times (1 + TOR)$

将模型降到两维的好处有两个，首先进一步节省计算量，其次我们可以做出更加直观的图像来佐证我们的判断，如下图分别是我们画出的训练集和测试集的图像，可以看出存在一定的分界，能够做分类。

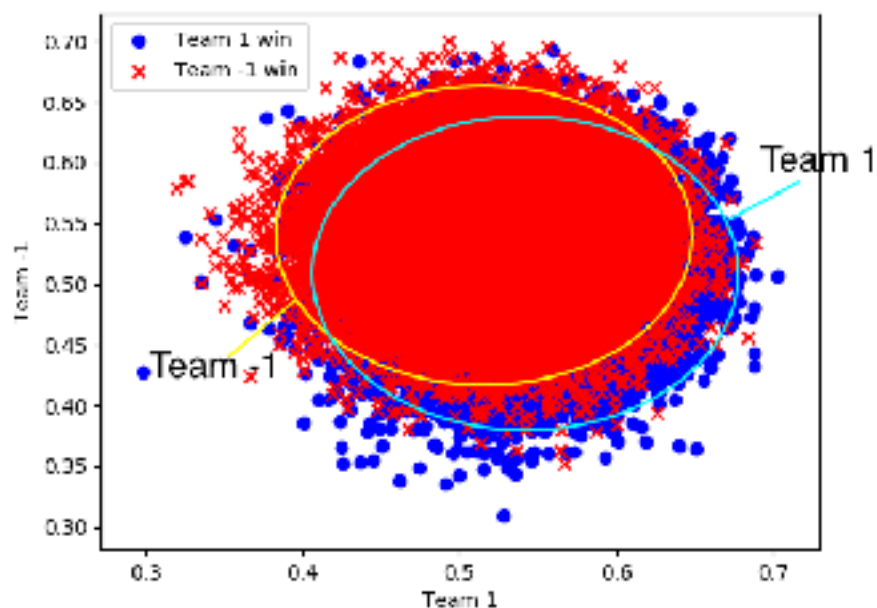


图4.1 训练集散点图示

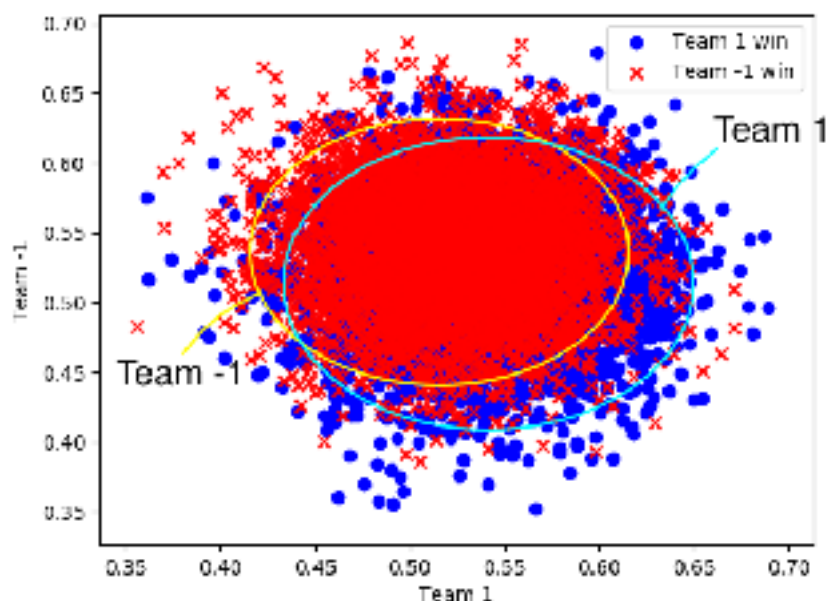


图4.2 测试集散点图示

分类器

Classifier

◆ 在选择分类器之前我们对几种常见分类算法的特点做了总结整理：

1. 朴素贝叶斯：属于生成式模型，即对于输入 x ，类别标签 y ，生成式模型估计它们的联合概率分布 $P(x, y)$ ；而根据我们的数据集特点，应在已知各个属性的情况下预测输赢概率，应用估计条件概率的模型更合适。除此之外，朴素贝叶斯需要各个属性之间相互独立才能表现出更快的收敛速度，而根据数据预处理的结果，我们所选取的属性之间并非相互独立的，所以朴素贝叶斯并不合适。
2. 决策树：该算法根据属性进行分支，选择一个合适的特征作为判断节点，可以快速分类，减少决策树的深度。引入信息熵的概念，用分类前后信息熵的差值衡量分类效果的好坏。但是该算法忽略了数据之间的相关性，而我们的数据之间具有相关性，使用该算法不能充分地利用数据所给的信息。
3. 最近邻算法KNN：主要思想为计算训练样本和测试样本点的距离，并求该距离升序排序后选取前 k 个距离最小的样本，根据这 k 个样本的标签进行投票后得到最后的分类类别。虽然该算法具有较强的一致性结果，但是根据数据预处理的结果来看，我们的样本属性呈现部分不平衡的问题（比如部分英雄之间有很大的克制率或协同率，而其余英雄之间则没有这么强的效应），对于该模型这个问题可能会导致分类结果较差，所以我们也放弃了这个算法。
4. SVM支持向量机：一个可以解决高维问题的算法，并且具有很高的准确率，只要选定合适的核函数就能运行得很好；但是该算法的最大缺点是内存消耗大，当观测样本高时效率低下，考虑到硬件设备的配置以及所消耗的时间我们毫不犹豫的放弃了这个算法。

5. 逻辑回归：属于判别式模型，即估计输入属性和类别标签之间的条件概率，并且可以使用在线梯度下降算法利用新数据来训练模型。该算法实现简单，分类时计算量小，存储资源低，是我们最终选择的算法，

◆ 下面对该算法的原理以及在我们处理数据过程中的应用做简要介绍：

逻辑回归（Logistics Regression）的原理其实与线性回归是类似的，通过一个含有大量数据点的n维点集 $X = \{(\mathbf{x}, y) \in X | \mathbf{x} = (x_1, x_2, \dots, x_n)\}$ ，找出一个线性函数 $f(x_1, x_2, \dots, x_n)$ 最好地满足标定的值。但是，两者的不同之处在于，逻辑回归需要将输出的很大范围的数压缩到0和1之间，表达为更加直观易懂的概率。于是，我们引入了一个sigmoid函数将无穷大的定义域压缩到[0,1]之间。如图2-1非常直观地解释了逻辑回归与线性回归的区别。

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

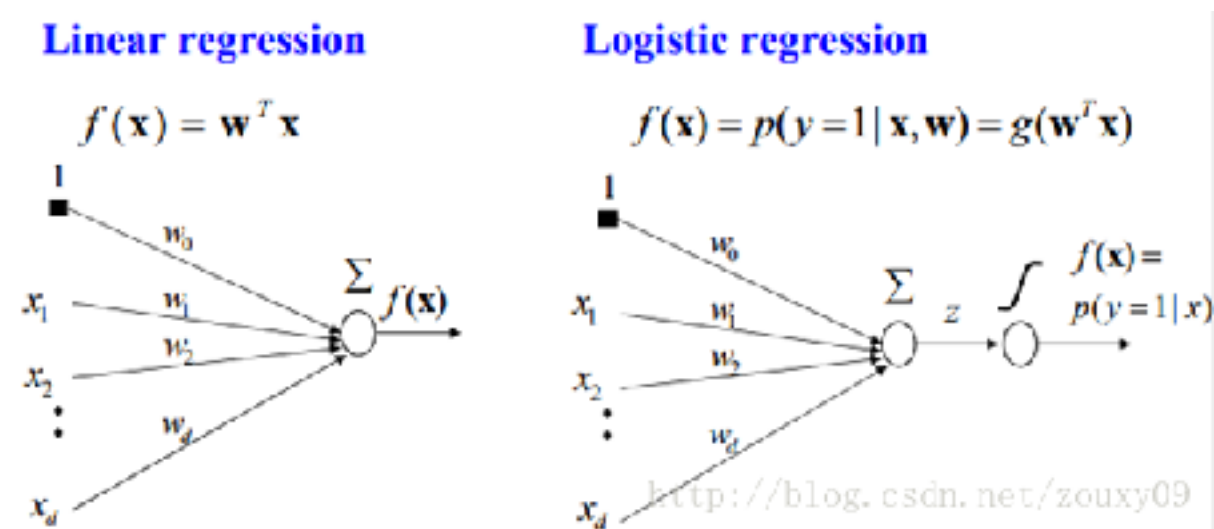


图2-1 逻辑回归与线性回归

逻辑回归的基本过程可以分为以下几步：

1. 找到一个合适的预测函数 $h(\mathbf{x})$ ，对于二分类器，该函数的输出的是两个分类的概率，所以我们用sigmoid函数。在这里我们使用的 h 函数是一个线性函数 $h(\mathbf{x}) = \text{sigmoid}(\theta_0 + \theta_1 x + \dots + \theta_n x_n)$

```
# sigmoid函数: sigmoid(x)=\frac{1}{1+ e^{-x}}
def Sigmoid(z):
    G_of_Z = float(1.0 / float((1.0 + math.exp(-1.0*z))))
    return G_of_Z

# h函数是所有属性的线性组合，并通过sigmoid函数将值域压缩到[0,1]
def Hypothesis(theta, x):
    z = 0
    for i in xrange(len(theta)):
        z += x[i]*theta[i]
    return Sigmoid(z)
```

2. 构造cost函数以及 $J(\theta)$ 模型来衡量 h 函数预测的结果是否合理。我们采用的cost函数来评判 h 函数的预测结果，用 $J(\theta)$ 对所有点的cost函数进行评估。cost函数和 $J(\theta)$ 如下所示：

$$\text{cost}(h(\mathbf{x}), y) = \begin{cases} \log(h(\mathbf{x})) & \text{if } y = 1 \\ \log(1 - h(\mathbf{x})) & \text{if } y = 0 \end{cases}$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{cost}(h(x^{(i)}), y)$$

```
# cost函数用于衡量h函数预测的结果是否合理
# J模型是cost函数对于所有点的平均值
def Cost_Function(X, Y, theta, m):
    sumOfErrors = 0
    for i in xrange(m):
        x1 = X[i]
        hi = Hypothesis(theta, x1)
        if Y[i] == 1:
            error = Y[i] * math.log(hi)
        elif Y[i] == 0:
            error = (1-Y[i]) * math.log(1-hi)
        sumOfErrors += error
    const = -1/m
    J = const * sumOfErrors
    # print 'cost is ', J
    return J
```

3. 用梯度下降法求出 $J(\theta)$ 的最小值，并输出此时的 θ 序列。更新过程如下：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta} J(\theta) \quad (j \text{ 代表第 } j \text{ 个属性})$$

α 代表学习率，反映了一次学习的步长，如果设得太小，则需要很多次迭代，大大增加了计算量；如果设得太大，则会导致调整不够精细，无法达到最佳点。化简之后我们得到了公式：

$$\frac{\partial}{\partial \theta_j} = -\frac{\alpha}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})x_j^{(i)}$$

```
# 对cost函数求偏导，为梯度下降法提供正确的方向
# a为学习率，决定了学习的步长
def Cost_Function_Derivative(X,Y,theta,j,m,alpha):
    sumErrors = 0
    for i in xrange(m):
        xi = X[i]
        xij = xi[j]
        hi = Hypothesis(theta,X[i])
        error = (hi - Y[i])*xij
        sumErrors += error
    m = len(Y)
    constant = float(alpha)/float(m)
    J = constant * sumErrors
    return J
```

最后，我们通过迭代的方式一步一步接近最佳分类值，返回最佳的 θ 作为结果。然后我们利用LR_test函数对分类结果进行测试，测试结果将在下一章详细说明。

最后说一下在构建分类器时遇到的问题。最开始刚刚写好分类器的时候，跑训练数据发现cost函数返回的值不收敛，即越迭代分类效果越差。因此，我在网上搜索出现这个情况的原因，发现是因为标定值的原因。在原始数据集中，分类的两个值是1和-1，然而在我们用到的分类器公式中，最佳的分类二值为1和0，不然后要重新推导cost函数的偏导。衡量了一下代价，我们决定把所有的-1改为0，经过这个方法，分类器就可以收敛了。

之后又发现分类器学习速度过慢的问题，由于Python本来的执行效率就相当低下，而迭代次数又多，在跑6个属性的数据时进行5000次迭代就要花3到4个小时。后来我们适当增加了学习率，经过多次实验找出了最佳学习率为 $\alpha = 1$ ，此时的最佳训练次数大约在2000次左右。

模型评估

Model Evaluation

首先我们从几个基本的参数来对模型的分类效果进行评估。当我们使用2个属性的模型时，训练结果如表6.1所示。

表格 6.1: 2个属性模型训练结果

训练次数	1000	2000	3000	4000	5000
Accuracy	59.53%	59.63%	59.45%	59.52%	59.50%
Recall	75.22%	69.37%	67.36%	66.47%	66.01%
Precision	59.62%	60.70%	61.16%	61.24%	61.24%

当我们使用6个属性的模型时，训练记过如表6.2所示。从表中数据可以看出，2个属性的模型虽然更加简洁，但分类效果并没有因此下降，所以综合考虑，我们认为两个属性的模型更加符合我们的需求，所以之后的模型分析我们都是建立在2个属性的模型的基础上进行讨论。

表格 6.2: 6个属性模型训练结果-1

训练次数	1000	2000	3000	4000	5000
Accuracy	59.66%	59.41%	59.21%	59.26%	59.16%
Recall	79.04%	71.37%	68.65%	67.50%	66.67%
Precision	59.18%	60.14%	60.42%	60.69%	60.75%

接着我们通过调整thresh值来控制False Positive Rate，比较发现True Positive Rate和False Positive Rate的关系，并作出ROC曲线如图6.1所示。

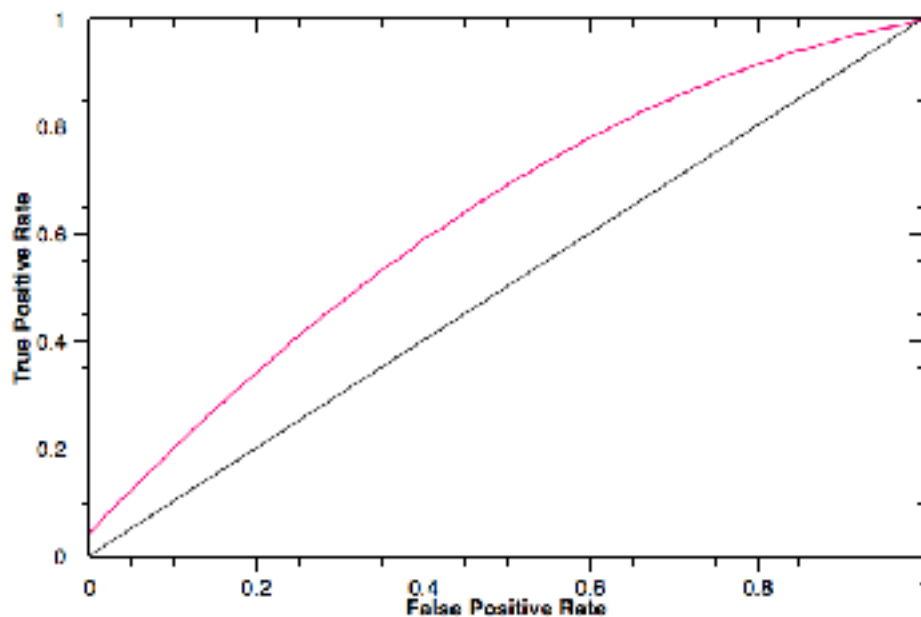


图6.1: ROC曲线

最后我们对我们的数据集进行了Cross Validation，将测试集和训练集混合之后随机分成两个新的数据集分别作为训练集和测试集，重新进行模型的评估，出乎我们意料的是，我们得到了更好的结果：

Accuracy	61.53%
Recall	71.14%
Precision	62.27%

经过我们的分析，这是由于原始数据集中训练集的胜率与测试集的胜率有比较大的偏差，跟我们的假设不符，所以导致分类结果有所下降。经过重新混合数据集后，分类效果有所提高。

总结

Sum Up

得到分类结果后，虽然准确率不算很高，但是对于这类竞技游戏来说得到过高的准确率，预测过程过于顺利反而是不正常的事，所以总体上我们对这个结果还是比较满意的。由于时间仓促，且整体工作量比较大，没有再编写支持向量机等可能得到较好结果的分类器，接下来如果有时间可以继续研究下去。

经过这次大作业，我们感到收获很多。在此之前小组成员都没有接触过类似的课程和作业，对机器学习算法知之甚少，对于Dota游戏也不了解，全凭兴趣选择了这个题目（虽然这个题目庞大的数据量给我们之后的数据处理带来种种困难）。我们与熟悉该游戏的同学一起讨论，查阅了官方文档和数据，对数据集有了足够多的了解后进行对数据集的分析，并根据其特点进行数据预处理。我们对预处理后得到的属性进一步精简、综合，最后得到了两个能包括主要特征的属性放入分类器进行分类，并用cross validation对数据集做了处理以提高分类的准确率。做完整个实验我们最大的体会是数据挖掘关键在于对于属性的选择，数据预处理后仍然得到众多属性，如何选择、计算出最合适的属性来区分数据的特征是分类是否成功的关键。总体来说，整个过程工作量还是比较大的，也使我们得到了充分的锻炼，对机器学习算法尤其是逻辑回归有了更深的理解，同时提高了我们对问题的解决能力，对Python语言也有了更好的掌握。最后十分感谢老师在课上的指导以及布置了这次大作业。