作业3:Aliens游戏

张祎扬（181840326　181840326@smail.nju.edu.cn)

(南京大学　匡亚明学院，南京 210046)

摘要：使用监督学习来模仿人玩游戏的动作。在这里我选择了四种算法，分别是NaiveBayes，Random Forest，AdaBoostM1,Logistic.利用非常优秀的机器学习包weka来构建模型，在训练时记录游戏状态作为训练集，并将训练出的模型应用于测试。比较不同算法的性能并尝试改进特征提取方法。

关键词：机器学习、监督学习、朴素贝叶斯、随机森林、AdaBoost、Logistic、weka

1.学习方法介绍

1.1 朴素贝叶斯NaiveBayes

1.1.1 算法思想

朴素贝叶斯算法的**关键**在于认为模型中的各个特征都是独立的，它并没有考虑某些特征之间的相关性，这就使事情变得十分简单。所以在模型的各个属性的相关性较小的时候，朴素贝叶斯的性能可以更好地体现。

贝叶斯定理的本质是一个求概率的过程。就是在已知$P(A|B)$的情况下，如何求得$P(B|A)$.

首先解释条件概率的定义：

$P(A|B)$表示事件B已经发生的前提下，事件A发生的概率。它叫做**事件B发生下事件A的条件概率**。基本计算公式为

$$P(A|B)=\frac{P(AB)}{P(B)}$$

贝叶斯定理的有用之处，是因为我们可能会碰到这样的情况：$P(A|B)$很容易得到，但是却很难得出$P(B|A)$.但根据贝叶斯定理，就可以有如下推导：

$$P(A|B)=\frac{P(AB)}{P(B)}$$
$$P(B|A)=\frac{P(AB)}{P(A)}$$

联立以上两个式子，消去$P(AB)$,可以得到

$$P(B|A)=\frac{P(A|B)*P(B)}{P(A)}$$

朴素贝叶斯被广泛地应用于分类问题中，假设结果有几种类别，可以利用朴素贝叶斯定理分别求出结果属于每一类的概率，取概率最高的类别作为最后的预测类别。在本次作业中，就是对于现有的特征提取方法，收集训练数据(当前游戏状态和作出的动作所属的类别)。根据我对代码的理解，在recoder.java中将移动的类型(move type)定义为最后的类别。而应用的过程就是根据当前游戏局面的状态，根据分类

器的学习结果，计算出各个类别的概率，**选择概率最高的类别并做出相应的动作**。

整个朴素贝叶斯分类过程分为三个阶段：

1. 准备阶段。确定特征属性，形成训练样本集合。在本次作业任务中，就是通过玩游戏，来记录各种数据。根据我的理解，训练集的好坏其实对分类器的性能有很大的影响，而在本次任务中，数据的收集又很特殊，它需要靠我们手动去操作游戏。这就导致了每一个人的训练集都是完全不一样的，所以不同同学之间如果要比较分类器的性能，也是没有一个绝对的方法。在这一情况下，我觉得我更需要考虑的是怎么优化我的训练集。一个方法是优化特征提取的函数，让它更有利于赢得游戏，当然这是作业内容2中的部分，我将稍后再讨论这一点。另一个方法就是我在手动操作的时候要尽量玩出**"漂亮"**的游戏，但这其实是一个很难的过程，~~首先就是我的游戏水平不怎么过关，老是玩死emm~~,然后就是，在玩游戏的过程中，不仅要注意多玩几次，提高样本容量，更要注意考虑到多种情况，比如躲避怪物的子弹等，不能老是在原地不动，只是发射子弹，这样分类器习得的性能显然不是很优秀的。
2. 分类器训练阶段。计算每个类别在训练样本中的出现频率，以及每个特征属性对每个类别的条件概率。

   设$x=\{a_1,a_2,\dots,a_m\}$为一个样本，其中$a_1,\dots,a_m$是x的属性，而$C=\{y_1,y_2,\dots,y_n\}$是一个类别集合。根据贝叶斯定理，假设各特征属性相互独立，各类别下各个特征属性的条件概率为$P(a_1|y_1),P(a_2|y_1),\dots,P(a_m|y_1);P(a_2|y_2),\dots,P(a_m|y_2);\dots,P(a_m|y_m)$,根据贝叶斯定理推导：

$$P(y_i|x)=\frac{P(x|y_i)*P(y_i)}{P(x)}$$

   因为分母对于各个类别其实是常数，所以只要比较分子即可。

$$P(x|y_i)P(y_i)=P(y_i)\prod_{j=1}^{m}P(a_j|y_i)$$

3. 应用阶段。使用分类器对待分类项进行分类。计算所有的$P(y_i|x)$,如果$P(y_k|x)=max\{P(y_i|x),i=1,2,\dots,n\}$,则$x$的类别是$y_k$.

1.1.2 一些优化和修正

当特征属性是离散的，只要计算各类别出现的频率即可用来估计概率。但特征属性也可能是连续值，这时候通常假定其值服从高斯分布。

$$g(x,\alpha,\sigma)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\alpha)^2}{2\sigma^2}}$$

而$P(a_k|y_i)=g(a_k,\alpha_{y_i},\sigma_{y_i})$,因此只需要计算样本各个类别中此特征值的均值和标准差，带入上述公式即可。

另一个需要讨论的情况是$P(a|y)=0$，因为0乘任何数都为0，这实际上大大降低分类器的性能。解决这个问题的方法是引入Laplace校准，即对每一个类别下所有计数加一，这样避免了出现0的情况，同时当训练集数量充分大时，不会对结果产生什么影响。

注：以上简介参考了一些博客（主要是[1])和介绍机器学习算法的网站[2]。

1.2 随机森林Random Forest

随机森林的基本单元是决策树，它是一种将多棵决策树集成的算法。每棵决策树都是一个分类器，输入样本，每棵树会有一个分类结果。而随机森林集成了所有的分类投票结果，把投票次数最多的类别指定为最终的输出。

每棵树生成过程如下：

1. 如果训练集的个数是N，在N个训练集中随机取样，但是不能取完全相同的样本做训练集。（我的理解是这些小的集合中的元素可以有交叉，但是不能完全一样）。这个小集合就将是这一棵树的训练集。
2. 如果有M个变量，确定一个数m<<M,用来确定每棵树结点选择多少个变量。
3. 每棵树都尽可能扩张，没有剪枝。

从以上过程可以看出，树的生成过程是很随机的，这也减少了树与树之间的相关性。

而随机森林的错误率依赖于两个因素：

- 森林中任意两棵树的相关性。增加相关性会提高错误率。
- 森林中每棵树的分类能力。个体树的分类能力的增加可以增加整个森林的分类能力。

袋外错误率out-of-bag(oob) error

每棵树都是用不完全相同的训练集生长出来的，对于第k棵树的生长，原数据集中大约有$\frac{1}{3}$没有用到，它们称为第k棵树的oob样本。把这些没有用到的数据集用第k棵树去分类。用这种方式，每个case都经过了大约$\frac{1}{3}$的树的分类，把这些分类结果进行投票，并选择票数最多的作为最终的分类结果。最后用误分类个数占样本总数的比率作为随机森林的oob误分率。

根据我的理解，随机树的随机性体现在两个方面：一个是每棵树的样本选择的随机性，另一个是m值选择的随机性。由于这种随机性，使得随机森林的过拟合概率减少。同时它是一种集成的分类方式，所以有更强的分类性能。

以上参考了随机森林算法的描述文件[3].

1.3 Adaboost

在搜索网页获取资料的过程中，我了解到集成学习常见的两种方案为bagging和boosting.其中，bagging是一种基于投票的方案，boosting是一种基于改变原始数据集分布的迭代串行方案。鉴于以上我选择了bagging的代表算法Random Forest, 所以我打算再选择一种boosting算法AdaboostM1.

1.3.1 Adaboost

Adaboost是一种用于二分类问题的集成方法。而M1则在它的基础上把问题扩展到多分类。

定义损失函数为指数损失函数$L(y,f(x))=exp(-yf(x))$,根据加型模型，第m轮的分类函数$f_m(x)=f_{m-1}(x)+\alpha_m G_m(x)$,其中，$\alpha_m$为基分类器$G_m(x)$的组合系数。AdaBoost采用前向分布(forward stagewise)这种贪心算法最小化损失函数，求解子模型的α_m.

$$\alpha_m = \frac{1}{2} log \frac{1-e_m}{e_m}$$

其中，e_m为$G_m(x)$的分类误差率。第m+1轮第训练数据集权值分布
$D_{m+1} = (w_{m+1,1}, \ldots, w_{m+1,i}, \ldots, w_{m+1,N})$

$$w_{m+1,i} = \frac{w_{m,i}}{Z_m} exp(-\alpha_m y_i G_m(x_i))$$

其中，Z_m为规范化因子

$$Z_m = \sum_{i=1}^{N} w_{m,i} * exp(-\alpha_m y_i G_m(x_i))$$

则得到最终分类器

$$sign(f(x)) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$$

α_m是e_m的单调递减函数，特别地，当$e_m \leq \frac{1}{2}$时，$\alpha_m \geq 0$;当$e_m > \frac{1}{2}$时，即基分类器不满足弱可学习的条件，则应该停止迭代。具体算法的流程如下：

> 1 $D_1(i) = 1/N$ %Initialize the weight distribution
>
> 2 for $m = 1, \ldots, M$:
>
> 3 learn base classifier $G_m(x)$;
>
> 4 if $e_m > 0.5$ then back;
>
> 5 update α_m and D_{m+1};
>
> 6 end for

从算法步骤中我们可以看出，权重的更新方式为：

$$D_{t+1}(i) = \begin{cases} D_t(i) * e^{-\alpha_t} & if h_t(x_i) = y_i, \\ D_t(i) * e^{\alpha_t} & if not. \end{cases}$$

一个样本是否被正确分类，它的权重将乘以不同的值。

可以看出，AdaBoost的核心步骤就是计算基学习器权重和样本权重分布。

1.3.2 AdaBoostM1

AdaBoostM1算法在原算法的基础上，把二分类问题扩展为多分类问题。但它和原算法有一些不同点。

- 分类函数的形式发生了变化。没有使用$sign()$映射转化。

$$H(x) = argmax \sum_{t=1}^{T} ln(\frac{1}{\beta_t})[h_t(x) = y]$$

- 权重更新函数作了一定的调整。如果一个样本被上一个分类器错误分类，那么它的权重不变，如果这个样本被上一个分类器正确分类，那么它的权重将乘以$\frac{\epsilon_t}{1-\epsilon_t}$,也就是说错误分类的样本权重值相对于正确分类的样本权重值扩大了$\frac{1-\epsilon_t}{\epsilon_t}$倍。

和随机森林不同的是，AdaBoost算法加入了迭代的过程，通过改变样本的权重来改变分类器模型函数。而随机森林则是一个少数服从多数的投票过程。

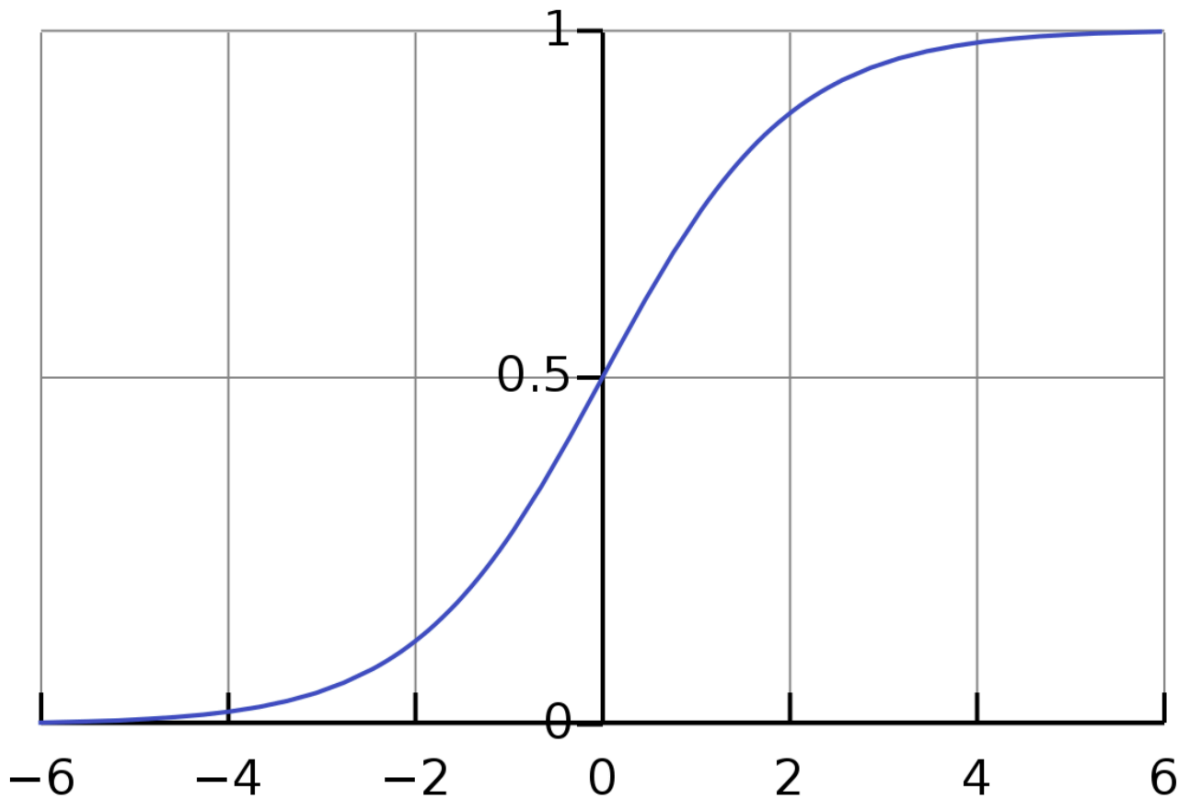
以上资料参考了github上的机器学习笔记[4]和一些博客[5-6]。

1.4 Logistic

Logistic回归输出一个0-1之间的离散结果。它通过使用其固有的logistic函数估计概率，来衡量因变量与一个或多个自变量之间的关系。自变量的取值范围可以从负无穷到正无穷，但是我们需要把输出控制在0和1.因此需要sigmoid函数。

$$z = \theta_0 = \theta_1 * x_1 + \theta_2 * x_2 + \dots$$
$$g(x) = \frac{1}{1+e^{-x}}$$
$$h = g(z) = \frac{1}{1+e^{-z}}$$

其图像如下



Sigmoid Function graph

需要一个损失函数来计算误分类的代价：

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if y} = 1 \\ -log(1-h_\theta(x)) & \text{if y} = 0 \end{cases}$$

为了方便求梯度，可以改写成下面的式子

$$-\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} log h_\theta(x^{(i)}) + (1-y^{(i)}) log(1-h_\theta(x^{(i)}))]$$

通过梯度的帮助，可以更新θ的值

$$J = \frac{-1}{m}[\sum_{i=1}^{m} y_i log h_i + (1-y_i) log(1-h_i)]$$

$$\frac{\delta J}{\delta \theta_n} = \frac{-1}{m}[\sum_{i=1}^{m} \frac{y_i}{h_i} h_i^2 x_n \frac{1-h_i}{h_i} + \frac{1-y_i}{1-h_i}(-h_i^2) x_n \frac{1-h_i}{h_i}]$$

$$\frac{\delta J}{\delta \theta_n} = \frac{-1}{m}[\sum_{i=1}^{m} x_n(1-h_i)y_i - x_n h_i(1-y_i)]$$

$$\frac{\delta J}{\delta \theta_n} = \frac{1}{m} x_i [\sum_{i=1}^{m} h_i - y_i]$$

以上参考了来源[7-9].

2.学习性能对比

2.1第0关

在第0关中，我在一开始就往右移动到没有障碍物的地方，并且保持一定的节奏发射子弹，做到每一发子弹都消灭了一个怪物，把所有怪物都消灭在了第一排。

2.1.1 朴素贝叶斯

学习结果如下

```
Correctly Classified Instances         299               77.4611 %
Incorrectly Classified Instances        87               22.5389 %
Kappa statistic                          0.4651
Mean absolute error                      0.1083
Root mean squared error                  0.314
Relative absolute error                 67.0479 %
Root relative squared error            111.33   %
Total Number of Instances              386

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.772     0.189      0.945     0.772     0.85       0.871    0
                 0.785     0.15       0.515     0.785     0.622      0.895    1
                 0         0          0         0         0          0.999    2
                 1         0.066      0.219     1         0.359      0.991    3
Weighted Avg.    0.775     0.179      0.855     0.775     0.798      0.878

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 241  46   0  25 |   a = 0
  14  51   0   0 |   b = 1
   0   2   0   0 |   c = 2
   0   0   0   7 |   d = 3
```

可以看到分类的正确率为77.4611%，并不算很高。仔细观察细节可以发现，所有的右移动作都被分类正确，这大概是因为我在玩游戏的过程中几乎没有左移的动作，基本保持静止在原地。什么都不做的动作的分类准确率最高，发射子弹的动作的准确率只有50%左右。同时从混淆矩阵可以看出，模型更容易把动作分类为发射子弹。

运行test模式，发现模型学习到了我右移和发射子弹的动作，所以它在一开始就移动到了最右边并且一直发射子弹。但它并没有学习到我发射子弹的节奏，所以它一直不停地发射子弹，但是还是有一些怪物没有被打中，最后输掉了。

2.1.2 随机森林

```
Correctly Classified Instances         340               88.0829 %
Incorrectly Classified Instances        46               11.9171 %
Kappa statistic                          0.5893
Mean absolute error                      0.0815
Root mean squared error                  0.2091
Relative absolute error                 50.4779 %
Root relative squared error             74.1478 %
Total Number of Instances              386

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.952     0.419      0.905     0.952     0.928      0.924    0
                 0.585     0.044      0.731     0.585     0.65       0.935    1
                 0         0          0         0         0          0.995    2
                 0.714     0.003      0.833     0.714     0.769      0.997    3
Weighted Avg.    0.881     0.346      0.87      0.881     0.874      0.928

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 297  14   0   1 |   a = 0
  27  38   0   0 |   b = 1
   2   0   0   0 |   c = 2
   2   0   0   5 |   d = 3
```

可以看出，正确被分类的比例是88%，和朴素贝叶斯相比提高了不少，它的分类精度也有所提高。它更倾向于把动作分类为什么也不做。

在运行test时，agent同样移到了最右边，但是它有一些左右摇摆的动作。它也保持一直发射子弹，但是因为无法规避子弹被击中。

2.1.3 AdaBoostM1

```
Correctly Classified Instances         336               87.0466 %
Incorrectly Classified Instances        50               12.9534 %
Kappa statistic                          0.5551
Mean absolute error                      0.1571
Root mean squared error                  0.2445
Relative absolute error                 97.3033 %
Root relative squared error             86.6834 %
Total Number of Instances              386

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.942     0.432      0.902     0.942     0.922      0.744    0
                 0.646     0.056      0.7       0.646     0.672      0.721    1
                 0         0          0         0         0          0.778    2
                 0         0          0         0         0          0.942    3
Weighted Avg.    0.87      0.359      0.847     0.87      0.858      0.744

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 294  18   0   0 |   a = 0
  23  42   0   0 |   b = 1
   2   0   0   0 |   c = 2
   7   0   0   0 |   d = 3
```

它分类正确的比例和随机森林差不多，精度也差不多，但是ROC Area有所下降。同时游戏表现也是移到最右边然后不停射击。

2.1.4 逻辑斯蒂回归

```
Correctly Classified Instances         322               83.4197 %
Incorrectly Classified Instances        64               16.5803 %
Kappa statistic                          0.5152
Mean absolute error                      0.0893
Root mean squared error                  0.2692
Relative absolute error                 55.3259 %
Root relative squared error             95.4452 %
Total Number of Instances              386

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.878     0.338      0.916     0.878     0.897      0.822    0
                 0.646     0.093      0.583     0.646     0.613      0.865    1
                 0.5       0.013      0.167     0.5       0.25       0.686    2
                 0.714     0.011      0.556     0.714     0.625      0.959    3
Weighted Avg.    0.834     0.289      0.85      0.834     0.841      0.831

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 274  30   4   4 |   a = 0
  22  42   1   0 |   b = 1
   1   0   1   0 |   c = 2
   2   0   0   5 |   d = 3
```

正确率相比前两个有略微下降，但是比朴素贝叶斯还是高很多。精度表现和之前相差不大，同时ROC Area也不是很理想。在实际游戏过程中，agent除了移到画面最右边以外，有一些左右摇摆的行为。同时发射子弹的频率不够高，导致agent很容易被敌人的子弹击中。

2.2 第1关

在这一关中，我没有待在原地不动，而是尽量有一些移动去追击最低的怪物的移动，同时注意躲避怪物的攻击。

2.2.1朴素贝叶斯

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.446     0.275      0.772     0.446     0.566      0.628    0
                 0.484     0.241      0.364     0.484     0.415      0.686    1
                 1         0.131      0.242     1         0.389      0.967    2
                 0.647     0.116      0.268     0.647     0.379      0.819    3
Weighted Avg.    0.489     0.252      0.629     0.489     0.514      0.666

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 166  95  60  51 |   a = 0
  45  59   9   9 |   b = 1
   0   0  22   0 |   c = 2
   4   8   0  22 |   d = 3
```

模型分类的准确率并不高，大约只有一半的被分类正确了。各项指标的表现都~~(很)~~不怎么样。在运行test的过程中，观察到agent有左右移动和追击怪物的行为，没有观察到明显的躲避怪物的子弹的行为，同时agent的行为有一些杂乱，难以理解。同时它学习到了一直发射子弹的动作。

2.2.2 随机森林

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.892     0.191      0.907     0.892     0.9        0.897    0
                 0.795     0.075      0.752     0.795     0.773      0.921    1
                 0.818     0.006      0.857     0.818     0.837      0.992    2
                 0.853     0.01       0.853     0.853     0.853      0.971    3
Weighted Avg.    0.865     0.147      0.867     0.865     0.866      0.911

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 332  32   3   5 |   a = 0
  25  97   0   0 |   b = 1
   4   0  18   0 |   c = 2
   5   0   0  29 |   d = 3
```

随机森林的正确率比朴素贝叶斯好很多。它更倾向于把动作分类为什么都不做。在实际应用过程中，前期agent有明显的追踪怪物的行为和持续发射子弹的行为。但是后期的表现不好，它在最右端静止。

2.2.3 AdaBoostM1

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         1          0.676     1         0.807      0.523    0
                 0         0          0         0         0          0.544    1
                 0         0          0         0         0          0.666    2
                 0         0          0         0         0          0.476    3
Weighted Avg.    0.676     0.676      0.457     0.676     0.546      0.531

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 372   0   0   0 |   a = 0
 122   0   0   0 |   b = 1
  22   0   0   0 |   c = 2
  34   0   0   0 |   d = 3
```

它的准确率也不太高。它倾向于把动作分类为什么都不做的动作。

在实际应用中，agent一直保持静止，不停发射子弹。竟然奇迹般地赢了该模型对追踪怪物和躲避子弹的学习都不是很好。

2.2.4 Logistic

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.855     0.264      0.871     0.855     0.863      0.851    0
                 0.721     0.093      0.688     0.721     0.704      0.889    1
                 0.773     0.009      0.773     0.773     0.773      0.951    2
                 0.706     0.021      0.686     0.706     0.696      0.874    3
Weighted Avg.    0.813     0.201      0.815     0.813     0.814      0.865

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 318  38   5  11 |   a = 0
  34  88   0   0 |   b = 1
   5   0  17   0 |   c = 2
   8   2   0  24 |   d = 3
```

它的分类准确率比朴素贝叶斯好很多。但是在test的时候，它更多的是无意义地左右移动，并且它对发射子弹这一个动作的学习效果不好，并没有怎么发射子弹。

2.3 第2关

在这一关中，我尽量避免设计到遮挡物。同时我尽量追踪怪物，但是我在玩游戏的过程中本身没有什么躲避子弹的行为。

2.3.1 贝叶斯

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.515     0.275      0.858     0.515     0.643      0.673    0
                 0.562     0.175      0.315     0.562     0.404      0.77     1
                 1         0.149      0.274     1         0.431      0.947    2
                 0.471     0.104      0.219     0.471     0.299      0.76     3
Weighted Avg.    0.544     0.246      0.722     0.544     0.582      0.705

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 229  83  80  53 |   a = 0
  26  41   2   4 |   b = 1
   0   0  31   0 |   c = 2
  12   6   0  16 |   d = 3
```

分类的精确率等各项指标不是很好。它倾向于把什么都不做的动作分类到其他类别。

在实际应用中，agent并没有保持一直发射子弹。同时，它有表现出追踪怪物的行为，但是也有一些无意义的左右移动的表现。我无法判判断它是无规律地左右移动还是躲避子弹。

2.3.2 随机森林

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.926     0.42       0.877     0.926     0.901      0.906    0
                 0.521     0.057      0.567     0.521     0.543      0.917    1
                 0.774     0.004      0.923     0.774     0.842      0.996    2
                 0.529     0.004      0.9       0.529     0.667      0.991    3
Weighted Avg.    0.844     0.328      0.842     0.844     0.839      0.917

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 412  29   2   2 |   a = 0
  35  38   0   0 |   b = 1
   7   0  24   0 |   c = 2
  16   0   0  18 |   d = 3
```

随机森林的准确率很高。在实际应用中，agent有发射子弹的行为，但是几乎没有追踪怪物和躲避子弹的行为。

2.3.3 AdaBoostM1

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         1          0.763     1         0.866      0.526    0
                 0         0          0         0         0          0.58     1
                 0         0          0         0         0          0.824    2
                 0         0          0         0         0          0.501    3
Weighted Avg.    0.763     0.763      0.583     0.763     0.661      0.547

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 445   0   0   0 |   a = 0
  73   0   0   0 |   b = 1
  31   0   0   0 |   c = 2
  34   0   0   0 |   d = 3
```

可以看到它倾向于把所有类别预测为a类，准确率表现一般。在实际表现中，它先在原地不动，一直发射子弹，到还剩一个怪物的时候却停止发射子弹并且停在最左边。可以看出这个模型对追踪怪物的学习很差。

2.3.4 逻辑斯蒂

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.867     0.428      0.867     0.867     0.867      0.799    0
                 0.274     0.076      0.339     0.274     0.303      0.773    1
                 0.71      0.033      0.55      0.71      0.62       0.901    2
                 0.824     0.02       0.718     0.824     0.767      0.957    3
Weighted Avg.    0.782     0.339      0.776     0.782     0.778      0.81

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 386  34  15  10 |   a = 0
  49  20   3   1 |   b = 1
   4   5  22   0 |   c = 2
   6   0   0  28 |   d = 3
```

准确率表现一般，倾向于把b类分类为a类。在实际应用中，agent一直发射子弹，并且表现出了一定的追踪怪物的行为。但是这种追踪行为并不是非常完善的。

2.4 第3关

在这一关中，由于缺少了障碍物的限制，我可以更自如地追赶怪物和躲避子弹。

2.4.1 贝叶斯

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.649     0.536      0.731     0.649     0.687      0.597    0
                 0.169     0.092      0.2       0.169     0.183      0.619    1
                 0.286     0.156      0.174     0.286     0.216      0.643    2
                 0.319     0.097      0.238     0.319     0.273      0.694    3
Weighted Avg.    0.526     0.406      0.567     0.526     0.543      0.612

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 244  41  59  32 |   a = 0
  37  11   7  10 |   b = 1
  33   1  16   6 |   c = 2
  20   2  10  15 |   d = 3
```

分类准确率依然不高，只有50左右。更倾向于把动作分类为左移。在实际应用中，agent对子弹发射的学习和怪物追踪的学习都不错。也表现出了躲避子弹的行为。

2.4.2 随机森林

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.923     0.524      0.798     0.923     0.856      0.856    0
                 0.354     0.04       0.548     0.354     0.43       0.859    1
                 0.393     0.033      0.579     0.393     0.468      0.899    2
                 0.426     0.018      0.69      0.426     0.526      0.885    3
Weighted Avg.    0.757     0.372      0.736     0.757     0.736      0.863

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 347  13  10   6 |   a = 0
  35  23   5   2 |   b = 1
  28   5  22   1 |   c = 2
  25   1   1  20 |   d = 3
```

准确率在75左右。agent有比较明显的躲避子弹的行为，但是对怪物的追踪行为不是很明显。

2.4.3AdaBoostM1

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         1          0.691     1         0.817      0.489    0
                 0         0          0         0         0          0.472    1
                 0         0          0         0         0          0.454    2
                 0         0          0         0         0          0.509    3
Weighted Avg.    0.691     0.691      0.478     0.691     0.565      0.485

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 376   0   0   0 |   a = 0
  65   0   0   0 |   b = 1
  56   0   0   0 |   c = 2
  47   0   0   0 |   d = 3
```

准确率在70左右，仍然倾向于把所有类别分类为a。

在实际应用中，agent只是在原地不动不停的发射子弹，并没有学习到左右移动追击怪物和躲避子弹的行为。

2.4.4 逻辑斯蒂

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.79      0.435      0.803     0.79      0.796      0.742    0
                 0.308     0.077      0.351     0.308     0.328      0.704    1
                 0.518     0.059      0.5       0.518     0.509      0.793    2
                 0.596     0.062      0.475     0.596     0.528      0.861    3
Weighted Avg.    0.688     0.321      0.689     0.688     0.688      0.753

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 297  31  24  24 |   a = 0
  37  20   3   5 |   b = 1
  24   1  29   2 |   c = 2
  12   5   2  28 |   d = 3
```

准确率不到70，TP率总体较低，从混淆矩阵也可以看出数据的总体分布是比较杂乱的。

在实际应用中，agent有左右移动的行为，但看起来不像是追击怪物。它表现出了一定的规避子弹的行为，但它并没有很好的保持发射子弹的行为。

2.5第4关

2.5.1贝叶斯

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.593     0.309      0.84      0.593     0.696      0.707    0
                 0.431     0.147      0.268     0.431     0.331      0.73     1
                 0.95      0.061      0.413     0.95      0.576      0.98     2
                 0.481     0.169      0.266     0.481     0.342      0.744    3
Weighted Avg.    0.578     0.264      0.693     0.578     0.61       0.725

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 200  51  27  59 |   a = 0
  19  22   0  10 |   b = 1
   0   1  19   0 |   c = 2
  19   8   0  25 |   d = 3
```

分类的准确率是60不到。倾向于把动作分类为右移。在实际应用中，agent没有体现出很好的追踪怪物左右移动的性能。

2.5.2 随机森林

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.955     0.504      0.839     0.955     0.893      0.877    0
                 0.431     0.037      0.595     0.431     0.5        0.911    1
                 0.4       0.002      0.889     0.4       0.552      0.974    2
                 0.481     0.012      0.833     0.481     0.61       0.95     3
Weighted Avg.    0.82      0.375      0.813     0.82      0.803      0.893

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 322  10   1   4 |   a = 0
  28  22   0   1 |   b = 1
  11   1   8   0 |   c = 2
  23   4   0  25 |   d = 3
```

随机森林的准确率超过了80，算是比较高的了，同时它的FP率也比较低，总的来说表现不错。在实际应用中，agent对发射子弹的学习效果比较好，也有一定程度表现出追踪怪物的左右移动，但是它的规避子弹的能力没有体现出来，最后被击中了。

2.5.3 AdaBoostM1

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.979     0.976      0.733     0.979     0.839      0.542    0
                 0         0          0         0         0          0.488    1
                 0         0          0         0         0          0.654    2
                 0.058     0.017      0.3       0.058     0.097      0.626    3
Weighted Avg.    0.724     0.717      0.571     0.724     0.625      0.551

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 330   0   0   7 |   a = 0
  51   0   0   0 |   b = 1
  20   0   0   0 |   c = 2
  49   0   0   3 |   d = 3
```

准确率超过70%,仍旧是倾向于把所有类别预测为a类。在实际应用中，agent并没有追踪击打怪物的能力，而是静止在原地不动，发射子弹。同时也没有规避子弹的行为。

2.5.4 逻辑斯蒂

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.792     0.293      0.881     0.792     0.834      0.768    0
                 0.588     0.1        0.423     0.588     0.492      0.767    1
                 0.6       0.02       0.571     0.6       0.585      0.955    2
                 0.731     0.066      0.585     0.731     0.65       0.863    3
Weighted Avg.    0.754     0.234      0.783     0.754     0.765      0.787

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 267  36   9  25 |   a = 0
  20  30   0   1 |   b = 1
   7   0  12   1 |   c = 2
   9   5   0  38 |   d = 3
```

分类的准确率在75左右，FP相对之前的比较低。在实际应用中有一定的追踪怪物的行为和规避子弹的行为。

2.6 总结

朴素贝叶斯虽然在数据上不是特别好看，准确率一直不太高，但是在实际应用中的表现还不错。它基本上学习了发射子弹的动作。随机森林的准确率基本上一直是最高的，在实际表现中也表现出了一些对怪物的追踪和对子弹的规避。AdaBoostM1的分类一直比较暴力，它基本上把所有的都分为a类，在实际应用中，agent基本上一直都是在原地不动，保持一直发射子弹，但是基本没有对怪物的追击和规避子弹的行为。逻辑斯蒂对发射子弹的行为的学习不是特别完善，表现较其他稍有逊色，同时它也没有很好地学习追踪怪物的行为，它的左右移动显得有些没有章法，有很大的随机性。

于我自己而言，这整个过程本身是非常主观的。这些学习首先依赖于我自己的操作，它们只能模仿我的动作。在实际打游戏的过程中，我并不是特别擅长打这种游戏。我花了大量的时间尝试，想要打出一场完美的游戏，但实际上很难兼顾到各个方面。又要不停发射子弹，还要追踪怪物，还要躲避怪物发射的子弹。在一局游戏中，这样的机会是很少的。甚至我可以在原地不动，只依靠控制发射子弹的节奏就直接把怪物消灭在第一行。但模型并不能学习到这个节奏，而它们又不能学习到我的左右移动和规避子弹，所以结果会很糟糕。所以在玩游戏的过程中，我不仅要获得游戏的胜利，更要在一局游戏中体现出更多更复杂的情况，这样才能有利于模型的学习。~~但是我的游戏水平真的特别菜~~,从模型的分析结果也可以看出，各种准确率之类的指标其实挺低的，但是我真的尽力了.....

从以上模型可以看出，从整体上来说，对于追击怪物和躲避子弹的学习是比较困难的，一个是因为这些情况本身比较复杂，另一个就是我以上提到的，想要在玩游戏的过程中很好的体现出这些过程是很有难度的。

3.修改特征提取方法

原特征提取函数只记录屏幕上每个位置的信息，我觉得这是不够的。当前模型缺少的是规避炸弹的能力和左右移动追踪怪物的能力，所以改进的方法应该服务于这两个目的。解决如何规避炸弹的问题，我的想法是记录agent和炸弹的距离。解决追踪怪物的能力，我的解决方案是记录位于agent两侧的怪物的数量。一般情况下agent倾向于往怪物数量多的一侧移动。

改进后的部分代码如下：

```
public static double[] featureExtract(StateObservation obs){

        double[] feature = new double[456];  // 448 + 4 + 1(class)

        // 448 locations
        int[][] map = new int[32][14];
        int left=0;
        int right=0;
        int bomb=0;
        Vector2d avatar_position = obs.getAvatarPosition();
        // Extract features
        LinkedList<Observation> allobj = new LinkedList<>();
        if( obs.getImmovablePositions()!=null )
            for(ArrayList<Observation> l : obs.getImmovablePositions())
allobj.addAll(l);
        if( obs.getMovablePositions()!=null )
            for(ArrayList<Observation> l : obs.getMovablePositions())
allobj.addAll(l);
        if( obs.getNPCPositions()!=null )
            for(ArrayList<Observation> l : obs.getNPCPositions()){
                for(Observation npc_ob : l){
                    //int x=(int)(npc_ob.position.x/25);
                    if(npc_ob.position.x<avatar_position.x)left++;
                    if(npc_ob.position.x>avatar_position.x)right++;
                }
                allobj.addAll(l);
            }

        for(Observation o : allobj){
            Vector2d p = o.position;
            int x = (int)(p.x/25);
            int y= (int)(p.y/25);
            map[x][y] = o.itype;
            if(o.itype==5 && Math.abs(o.position.x-avatar_position.x)
<25)bomb=1;
        }
```

```
        for(int y=0; y<14; y++)
            for(int x=0; x<32; x++)
                feature[y*32+x] = map[x][y];

        // 4 states
        feature[448] = obs.getGameTick();
        feature[449] = obs.getAvatarSpeed();
        feature[450] = obs.getAvatarHealthPoints();
        feature[451] = obs.getAvatarType();
        feature[452] = left;
        feature[453] = right;
        feature[454] = bomb;

        return feature;
    }

public static Instances datasetHeader(){
        FastVector attInfo = new FastVector();
        // 448 locations
        for(int y=0; y<14; y++){
            for(int x=0; x<32; x++){
                Attribute att = new Attribute("object_at_position_x=" + x
+ "_y=" + y);
                attInfo.addElement(att);
            }
        }
        Attribute att = new Attribute("GameTick" );
attInfo.addElement(att);
        att = new Attribute("AvatarSpeed" ); attInfo.addElement(att);
        att = new Attribute("AvatarHealthPoints" );
attInfo.addElement(att);
        att = new Attribute("AvatarType" ); attInfo.addElement(att);
        att = new Attribute("left number");attInfo.addElement(att);
        att = new Attribute("right number");attInfo.addElement(att);
        att = new Attribute("bomb is detected");attInfo.addElement(att);
        //class
        FastVector classes = new FastVector();
        classes.addElement("0");
        classes.addElement("1");
        classes.addElement("2");
        classes.addElement("3");
        att = new Attribute("class", classes);
        attInfo.addElement(att);

        Instances instances = new Instances("AliensData", attInfo, 0);
        instances.setClassIndex( instances.numAttributes() - 1);

        return instances;
    }
```

4.学习性能对比

在这里因为时间原因我就不把每一关的对比放上来了。由于第三关没有障碍物的限制，我可以更自如地移动，所以我把第三关的结果放上来。

4.1 朴素贝叶斯

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.489     0.349      0.704     0.489     0.577      0.597    0
                 0.4       0.166      0.325     0.4       0.359      0.644    1
                 0.488     0.214      0.216     0.488     0.299      0.709    2
                 0.307     0.094      0.258     0.307     0.28       0.719    3
Weighted Avg.    0.456     0.28       0.545     0.456     0.482      0.629

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 240  81 121  49 |   a = 0
  49  52  16  13 |   b = 1
  26  13  41   4 |   c = 2
  26  14  12  23 |   d = 3
```

emm可以看到朴素贝叶斯的正确率依然只有一半左右，而且各类别的误分类率都不低。但是在实际应用中，agent表现出了比较明显的追踪怪物的性能，可以看到它对追踪怪物和发射子弹的行为的学习还是比较成功的。

4.2 随机森林

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.9       0.491      0.757     0.9       0.822      0.839    0
                 0.423     0.046      0.647     0.423     0.512      0.875    1
                 0.536     0.03       0.682     0.536     0.6        0.928    2
                 0.467     0.014      0.778     0.467     0.583      0.959    3
Weighted Avg.    0.74      0.322      0.732     0.74      0.724      0.866

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 442  24  15  10 |   a = 0
  71  55   4   0 |   b = 1
  36   3  45   0 |   c = 2
  35   3   2  35 |   d = 3
```

准确率超过了百分之七十，误分类率相比于朴素贝叶斯减少了许多，FPrate都基本低于0.1了。ROC Area的值也增加了不少。在实际应用中，agent表现出了非常明显的追踪怪物的行为，也有一定的躲避怪物的行为。基本上消灭了所有怪物。

4.3 AdaBoostM1

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         1          0.629     1         0.773      0.459    0
                 0         0          0         0         0          0.486    1
                 0         0          0         0         0          0.576    2
                 0         0          0         0         0          0.549    3
Weighted Avg.    0.629     0.629      0.396     0.629     0.486      0.485

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 491   0   0   0 |   a = 0
 130   0   0   0 |   b = 1
  84   0   0   0 |   c = 2
  75   0   0   0 |   d = 3
```

准确率在63左右，和之前的情况很类似，AdaBoostM1算法还是倾向于把所有的动作分类为第一类。在实际应用中，agent仍然表现为在原地不动保持发射子弹的姿势。

4.4 Logistic

```
Correctly Classified Instances         502               64.359  %
Incorrectly Classified Instances       278               35.641  %
Kappa statistic                          0.3874
Mean absolute error                      0.1809
Root mean squared error                  0.4186
Relative absolute error                 65.0294 %
Root relative squared error            112.3699 %
Total Number of Instances              780

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.721     0.329      0.788     0.721     0.753      0.709    0
                 0.5       0.132      0.43      0.5       0.463      0.731    1
                 0.5       0.079      0.433     0.5       0.464      0.795    2
                 0.547     0.06       0.494     0.547     0.519      0.804    3
Weighted Avg.    0.644     0.243      0.662     0.644     0.651      0.731

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 354  66  39  32 |   a = 0
  50  65  10   5 |   b = 1
  24  13  42   5 |   c = 2
  21   7   6  41 |   d = 3
```

准确率64左右，误分类为b类的概率比较高。在实际应用中，agent表现出了比较优秀的持续射击能力和追踪怪物左右移动的能力，最终赢了。它有一定的左右移动规避炸弹的行为。

4.5 总结

由于样本容量太小，同时我自己在玩游戏的过程中有比较强的不确定性，所以这些学习的性能只能用作一个参考。同时，在修改特征提取方法前后进行对比的过程中，我并不能做到两次玩游戏都玩的一模一样，所以是否可以看到性能提升很大程度上也和训练集有关。即使性能其实没有提升，假如你玩的更好了，结果也会有一些明显的提升。退一步说，就算模型真的得到了改进，假如训练集不够优秀，也可以造成表面上的性能下降。所以从这个角度上来说，我觉得这些数据只是具有参考价值。

从数据上来看，朴素贝叶斯在数据上是最不好看的，它的准确率最低，各项指标也没有其他三种好，明显差一截。其他三个算法在数据上相差不大。但是在实际应用中，朴素贝叶斯的表现还是不错的，因为它的误分类率并不低，所以它的行为其实有一些随机性，反而会容易杀到怪物。随机森林算法和AdaBoostM1的准确率一直是最高的，和贝叶斯相比，这也体现出了集成学习的明显优势。其中，随机森林的表现比较优秀，不仅准确率高，在实际应用中agent的表现也比较好，追踪怪物的行为比较明显，也保持一直发射子弹，也可以在一定程度上看出规避子弹的行为。但是AdaBoostM1在所有关卡中都倾向于把动作分类为a类，它的分类结果是十分简单粗暴的。在实际应用中，agent永远保持在原地不动，一直发射子弹，几乎没有规避子弹和左右移动追踪怪物的行为，是非常不灵活的。逻辑斯蒂算法的准确率略微逊色于两种集成算法，但也比贝叶斯算法优秀许多。在实际应用中，逻辑斯蒂的表现也是比较优秀的，表现出了左右移动追击怪物，也有规避子弹的行为。

总的来说，在改进了特征提取方法以后，虽然在指标上没有明显的提升，但是在实际应用时，我明显地感觉到agent左右移动追击怪物的能力增加了不少。但对于规避子弹的能力的提升我并没有明显的感觉。我觉得首先是因为这个行为本身比较难学习，其次我的特征提取方法的改进对于学习规避子弹这个行为还是远远不够的，再就是我在玩游戏的过程中也并没有展现一个比较完美的训练集（这实在是个人水平有限，我已经不知道玩了多少轮了qwq）。顺便一点感想，玩游戏的能力有时候挺重要的:)

5.结束语

这次实验对我更多的是一个探索和比较的过程，鉴于它本身的影响因素很多，导致结果并不是绝对的客观。但我觉得更重要的是，这一过程中，通过自己的探索，去比较不同的算法之间的优劣，去思考怎么提升模型的学习性能。以及在无数次的玩游戏过程中思考，怎么表现才能让训练集更为完美，才能让模型尽可能地学习更多的技能。

致谢：也许是我的JAVA版本的问题，在打开weka.jar包的时候一直出错。最后我暂时的解决方案是在虚拟机里重新安装了一个IDE。非常感谢刘驭壬助教对我的帮助。

References:

[1]https://www.cnblogs.com/leoo2sk/archive/2010/09/17/naive-bayesian-classifier.html

[2]https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/

[3]https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#inter

[4]https://github.com/Vay-keen/Machine-learning-learning-notes/blob/master/周志华《Machine%20Learning》学习笔记(10)--集成学习.md

[5]https://www.jianshu.com/p/121980ea415d

[6]https://www.cnblogs.com/en-heng/p/5974371.html

[7]https://hackernoon.com/introduction-to-machine-learning-algorithms-logistic-regression-cbdd82d81a36

[8]https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html

[9]http://www.sohu.com/a/231549006_129720