Aliens 实验报告

宋磊（181220049、974534426@qq.com）

(南京大学 人工智能学院)

1 收集数据的方法

在经历了收集数据的过程后，认为这也是一个值得思考也是最重要的地方，数据的好坏直接影响着训练出来的模型的好坏，所以在这里记录一下。

开始只是觉得，收集数据只是简单地玩一玩，通关胜利就行，但后来发现，需要在一关游戏中经历各种状态，这样训练出来的模型才会对各种状态做出正确地回应。就像课程主页上说：“监督学习只能模拟你的行为，如果你的行为有错误，不能纠正你的错误”。

开始只是在同一个点按一定的间隔发射，会将绝大部分外星人消灭在这个点，之后只需要移动到下一个位置，不断发射，就可以全部消灭，但这样训练出来的模型也是只会在相同的点发射，甚至会把极少的移动当做噪声，完全没有学到正确的移动，而且因为很少会遇到外星人发射的情况，也没有学到躲避炮弹。

之后的收集过程，我会控制着追着第一个外星人打，但也不会移动过快，保证一颗子弹可以消灭一个外星人，并且会在有炮弹时刻意躲避，训练的效果好了很多，会在遇到炮弹时躲避，但发现在外星人移动到后面的位置时如果还没有被消灭，agent 只会躲在右下角静止，我认为是因为收集数据时，将所有的外星人在靠上的几层消灭，agent 没有学到在后面几层该怎么办。又进行了改进，不仅移动躲避炮弹，追杀外星人，也控制击杀外星人速度不会太快，留几个在最后几层击杀。

所以这个问题中一个好的收集数据的过程，我认为要满足几点：1、准确的击杀，每一次发射都可以击杀一个外星人；2、明显的躲避子弹的行为，在看到子弹时向相反方向明显移动；3、追赶击杀第一个外星人，并且控制在不同位置击杀外星人。这样使收集到的数据更加完整，训练出的模型对大部分情况都能做出正确地处理。

2 学习方法介绍和性能对比

2.1 随机森林

2.1.1 方法介绍

随机森林在以决策树为基学习器构建 Bagging（有放回随机采集样本，同时用剩下的样本作为验证集来对泛化性能进行“包外估计”）集成的基础上，进一步在决策树的训练过程中引入了随机属性选择。对基决策树的每个节点，先随机选择包含 k 个属性的子集（在 weka 中也提供了这个参数），再从这个子集中选择一个最优属性用于划分，通过 k 控制随机性的引入程度，一般推荐$k = log_2 d$ [1]。所以，在训练中，我也将参数 k 设置成这样。

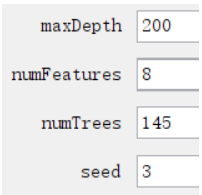
泛化误差取决于每一棵树分类的准确性以及每棵树之间的关联程度。当森林中树的数量增长时，森林的泛化误差逐渐收敛到一个确定的值。另外，虽然对每一个节点属性进行随机选择相比较 Adaboost 会增加一定的错误率，但也增加了模型对噪声的鲁棒性[2]。

在我的训练过程中，将 k 直接选为了$log_2 d$，参数主要是调整树的数量，树的数量太大或太小都会降低模型的准确性。

2.1.2 训练结果

	是否通关	发射	正确的移动(即追着第一个外星人打)	躲避子弹
随机森林	基本全部通关	1	0.9	0.5

在下图的参数下，获得了很好的效果，虽然实验数据都是从第 0 关收集到的，但是最终却可以以一个比较大的概率通过每一关，最终学到了发射和跟着第一个外星人移动，并且能够对半数的子弹进行躲避。



结果如图：

```
Time taken to build model: 1.19 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances         632               98.2893 %
Incorrectly Classified Instances        11                1.7107 %
Kappa statistic                          0.965
Mean absolute error                      0.0635
Root mean squared error                  0.1187
Relative absolute error                 26.1398 %
Root relative squared error             34.1084 %
Total Number of Instances              643

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.982    0.015      0.993     0.982     0.988      0.999    0
                 0.982    0.013      0.939     0.982     0.96       0.999    1
                 0.976    0          1         0.976     0.988      1        2
                 1        0.002      0.98      1         0.99       1        3
Weighted Avg.    0.983    0.013      0.983     0.983     0.983      0.999

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 436   7   0   1 |   a = 0
   2 108   0   0 |   b = 1
   1   0  40   0 |   c = 2
   0   0   0  48 |   d = 3
```

2.2 朴素贝叶斯

2.2.1 方法介绍

朴素贝叶斯是贝叶斯家族中一种特定形式的分类器。基于两个重要而简单的假设。一是假设所有预测属性条件独立地对分类结果产生影响，即属性条件独立性假设，二是假定没有隐藏属性影响预测的过程。虽然在现实中，假设很难完全满足，但朴素贝叶斯分类器在实践中展示出了很好的性能[3]。

基于朴素贝叶斯的假设，我们可以写出下图的公式[1]，其中 d 为属性的数量， c 是一个特定的类别标签，x 是一个观测到的特定属性值，给定一个测试数据 x，使用下面的公式可以计算出在当前观测值的条件下，属于每一个类别的可能性。

$$P(c \mid \boldsymbol{x}) = \frac{P(c)\, P(\boldsymbol{x} \mid c)}{P(\boldsymbol{x})} = \frac{P(c)}{P(\boldsymbol{x})} \prod_{i=1}^{d} P(x_i \mid c)$$

$P(x)$在所有类别中的值都相同，如果需要得到概率的值，计算中可以直接将其直接变为一个比例因子，之后进行正则化(normalize)，使概率之和为 1，就可以直接得到对应的概率。这里只是找出其中最大的一项，所以直接忽略这个常数，得到下面的公式[1]，就是朴素贝叶斯分类器的表达式。

$$h_{nb}(\boldsymbol{x}) = \arg\max_{c \in \mathcal{Y}} P(c) \prod_{i=1}^{d} P(x_i \mid c)$$

根据上面的公式可以看出，朴素贝叶斯分类器的训练过程，就是基于训练集来估计类别标签的先验概率$P(c)$和属性估计的条件概率$P(x_i|c)$的过程。

但上面的公式在处理某个属性值在训练集中没有与某个类同时出现过的测试数据时，计算出的概率会恒等于 0，显然是不合理的，需要进行平滑，常用拉普拉斯修正，将类别标签的先验概率和属性估计的条件概率修正为下面的形式[1]，从而避免了因训练集样本不充分而导致的概率估计为零的问题。

$$\hat{P}(c) = \frac{|D_c| + 1}{|D| + N},$$

$$\hat{P}(x_i \mid c) = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i}$$

2.2.2 训练结果

	是否通关	发射	正确的移动（即追着第一个外星人打）	躲避子弹
朴素贝叶斯	大概率不能通关	1	0	0

朴素贝叶斯分类器虽然训练出模型的速度十分快，但结果并不理想，很多时候一关也未能成功通过，最终只会在开局的时候移动到左下角，然后发射，偶尔左右移动一下，但很明显可以看出是随机的移动，不是为了躲避子弹的移动，也没有学到跟着第一个外星人移动以及躲避子弹。结果如图。

```
=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances         350               54.4323 %
Incorrectly Classified Instances       293               45.5677 %
Kappa statistic                          0.2878
Mean absolute error                      0.2271
Root mean squared error                  0.4552
Relative absolute error                 93.4617 %
Root relative squared error            130.8188 %
Total Number of Instances              643

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.561     0.176      0.877     0.561      0.684      0.764    0
                 0.409     0.124      0.405     0.409      0.407      0.773    1
                 0.805     0.251      0.179     0.805      0.293      0.871    2
                 0.479     0.069      0.359     0.479      0.411      0.874    3
Weighted Avg.    0.544     0.164      0.713     0.544      0.591      0.781

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 249  57 110  28 |   a = 0
  25  45  28  12 |   b = 1
   4   3  33   1 |   c = 2
   6   6  13  23 |   d = 3
```

2.3 KNN

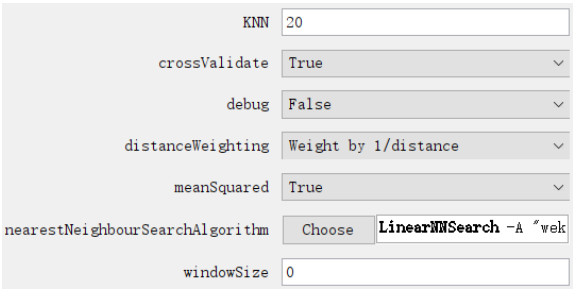
2.3.1 方法介绍

KNN 是一种常用的监督学习的方法，是“懒惰学习”的著名代表。运行过程是，给定测试样本，基于某种距离度量（通常是欧式距离）找出与其最靠近的 k 个训练样本，如果是分类任务，之后使用投票法，得到属性标签，如果是回归任务，则使用平均法，得到目标的预测值。还可以基于距离的远近进行加权，例如权重取距离的倒数，使得靠近的点权值更大，而离得远的点权值较小。

2.3.2 训练结果

	是否通关	发射	正确的移动(即追着第一个外星人打)	躲避子弹
KNN	只能稳定通过第一关	1	0.5	0.3

参数及训练结果如图，由于打开了交叉验证选择最优的参数 k，所以选择了一个比较大的 k=20，最后训练的结果显示，使用 3 近邻会达到最好效果（如第二张图所示）。




```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 3 inverse-distance-weighted nearest neighbour(s) for classification


=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances         632               98.2893 %
Incorrectly Classified Instances        11                1.7107 %
Kappa statistic                          0.9642
Mean absolute error                      0.014
Root mean squared error                  0.0675
Relative absolute error                  5.7703 %
Root relative squared error             19.3913 %
Total Number of Instances              643

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.995    0.045      0.98      0.995     0.988      0.999    0
                 0.936    0.004      0.981     0.936     0.958      0.999    1
                 0.976    0          1         0.976     0.988      1        2
                 0.979    0          1         0.979     0.989      1        3
Weighted Avg.    0.983    0.032      0.983     0.983     0.983      0.999

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 442   2   0   0 |   a = 0
   7 103   0   0 |   b = 1
   1   0  40   0 |   c = 2
   1   0   0  47 |   d = 3
```

KNN 只能通过一关，最终学到了发射，也学到了移动，但是移动的范围只局限在左半边，很少会向右半边移动，在遇到子弹时，也会进行一定的躲避，但躲过的概率不太高

2.4 性能对比

	是否通关	发射	正确的移动（即追着第一个外星人打）	躲避子弹
随机森林	基本全部通关	1	0.9	0.5
朴素贝叶斯	大概率不能通关	1	0	0
KNN	只能稳定通过第一关	1	0.5	0.3

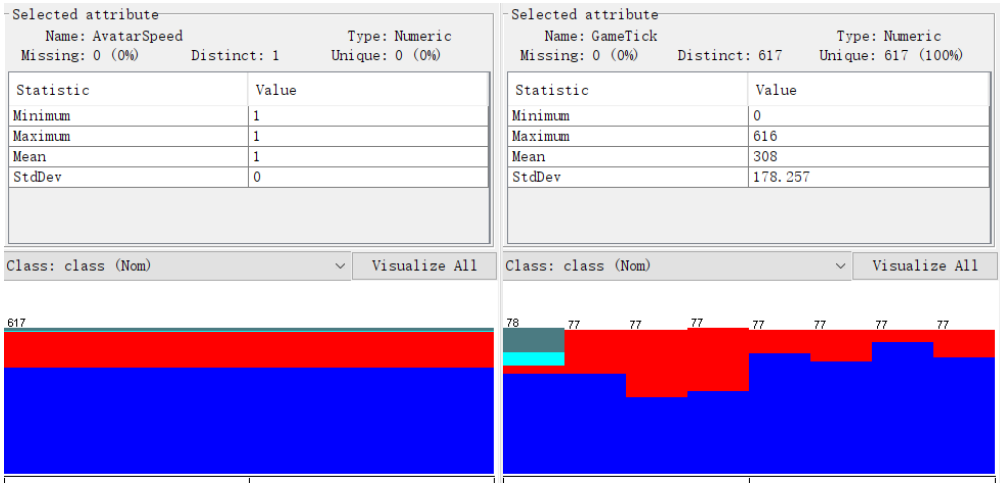
可以看出随机森林的结果比较好，只有躲避子弹存在一定的问题；而朴素贝叶斯表现较差，只学到了发射；KNN 的效果居中，学到了发射和简单移动，希望在下一问改良特征提取方法后，获得更好地性能。

3 修改特征提取方法

3.1 代码本身漏洞的思考

在 datasetHeader()中，在记录完 448 个位置信息之后，又记录了 GameTick，AvatarSpeed，AvatarHealthPoints，AvatarType。但在 featureExtract()函数中，却把这四个属性都存储在 feature[448]同一个位置，显然是不对的，需要对代码进行改正。本来认为需要将这四个属性存储在正确的位置 448-451，但在后来发现，后三个属性其实是冗余特征，在游戏的全过程没有发生变化。我选择了直接删除。

首先对记录的冗余属性进行删除，很多属性例如 AvatarSpeed（图一），最大最小值相等，方差为 0，即在游戏全过程没有发生过变化，但是却被记录这样肯定会对训练结果造成影响。发生变化的属性会有完全不同的图，例如 GameTick（图二）。在 featureExtract()，函数中添加了 System.out.println(obs.getAvatarSpeed()) 等打印语句，对中间结果进行打印，也可以看出输出 AvatarSpeed 的值全部为 1，没有发生变化，而 GameTick 则在不断增加，所以可以删除 AvatarSpeed 这类冗余属性。



发现的冗余属性还有 AvatarHealthPoints，getAvatarType。在 object_at_position 属性中。以及本来我希望记录移动方向 AvatarOrientation，但尝试后发现这个属性在全程也不会发生变化，所以没有进行添加。

所以具体的改动如下，对原来的代码进行了注释，之后新添加特征从 449 开始：

```
Attribute att = new Attribute( attributeName: "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);
```

```
// 4 states
feature[448] = obs.getGameTick();
//feature[449] = obs.getAvatarSpeed();
//feature[450] = obs.getAvatarHealthPoints();
//feature[451] = obs.getAvatarType();
```

3.2 添加属性

依次添加了游戏当前的分数，Avatar 的位置信息，以及周围是否存在炮弹（用 dangerous 变量表示）。

```
att = new Attribute( attributeName: "GameScore" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarPosX" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarPosY" ); attInfo.addElement(att);
att = new Attribute( attributeName: "Dangerous" ); attInfo.addElement(att);
```

以下为在 featureExtract()函数中的修改：

3.2.1 初始化变量：

```
int ava_x = 0;
int ava_y = 0;
boolean dangerous = false;
```

3.2.2 记录游戏分数

```
//GameScore
feature[449] = obs.getGameScore();
```

3.2.3 记录位置信息

```
if(obs.getAvatarPosition()!=null){
    Vector2d pos = obs.getAvatarPosition();
    ava_x = (int)(pos.x/25);
    ava_y = (int)(pos.y/25);
}
```

```
//Position
feature[450] = ava_x;
feature[451] = ava_y;
```

3.2.4 判断是否危险

判断周围是否有炸弹，由于对原本的位置信息进行离散化后，整张地图的大小为 32*14，所以对危险区域判断为 x 方向小于 5，y 方向小于 6,的范围内存在炸弹，dangerous=true。

```
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) {
        //System.out.println((int)(p.x/25)  + " + " + (int)(p.y/25));
        int bomb_x = (int)(p.x/25);
        int bomb_y = (int)(p.y/25);
        if(Math.abs(ava_x-bomb_x)<5 && Math.abs(ava_y-bomb_y)<6)
            dangerous = true;
    }
}
```

```
//Dangerous if there is bomb around the Avatar
feature[452] = dangerous ? 1 : 0;
```

3.3 性能对比

	是否通关	发射	正确的移动（即追着第一个外星人打）	躲避子弹
随机森林	基本全部通关	1	0.8	0.8
朴素贝叶斯	能够通过第一关	1	0.5	0
KNN	可以通过前两关	1	0.8	0.3

改良之后，随机森林的移动能力略微下降偶尔会出现一些震荡，但躲避子弹能力上升，可以躲避大多数子弹；朴素贝叶斯也学到了移动的能力，但躲避子弹能力没有上升；KNN 移动方面得到了很大提升，能够跟着第一个外星人进行移动，但在躲避子弹方面还是没有很好地学到。

可以看到，改良之后性能有一定提升，但后两个模型在躲避子弹方面做得还是不够好，所以在后面的缺少障碍的关中，很容易被杀死。

附中文参考文献（后两篇文献来自 weka 分类方法对应的文档）：

[1] 周志华.机器学习[M].清华大学出版社.

[2] Leo Breiman (2001). Random Forests. Machine Learning. 45(1):5-32.

[3] George H. John, Pat Langley: Estimating Continuous Distributions in Bayesian Classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 338-345, 1995.