
Aliens 游戏实验报告

周韧哲 (181220076、zhourz@smail.nju.edu.cn)

(南京大学 计算机科学与技术系, 南京 210093)

摘要: 完成了 Aliens 游戏的任务

关键词: 监督学习, NavieBayes, Random Forest, logistic, AdaBoost

1 引言

在这次的实验中, 介绍了四种学习方法: NavieBayes, Random Forest, logistic, AdaBoost, 并将其运用在 Aliens 游戏中, 比较了这四者的学习效果, 然后改进了特征提取方法, 进行了学习效果的横向对比和纵向对比。

2 学习方法介绍

2.1 Navie Bayes

朴素贝叶斯分类基于贝叶斯定理与特征条件独立假设:

$$\text{贝叶斯定理: } P(B_i|A) = \frac{P(B_i)P(A|B_i)}{(\sum_{j=1}^n P(B_j)P(A|B_j))}。$$

属性条件独立假设: 在分类标记给定的条件下, 特征之间是独立的。

已知给出一个训练数据集 $\{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$, 称 x_i 为示例, y_i 为分类标记, $y_i \in \{c_1, c_2, \dots, c_k\}$, 每个示例 x 有 n 个属性。朴素贝叶斯分类要做的是将一个新的实例 x 分类。朴素贝叶斯分类先计算每个分类标记的先验概率, 然后计算 x 在每个分类标记下的条件概率, 最后再根据贝叶斯定理计算每个分类标记在 x 下的后验概率, 则实例 x 就属于后验概率最大的那个类别。其算法具体如下:

1. 分类标记的 k 个先验概率 $P(Y = c_i) = \sum_{j=1}^m (y_j == c_i) / m$ 。
2. 计算 x 在 c_i 下的条件概率 $P(X = x|Y = c_i) = P(X^1 = x^1, \dots, X^n = x^n|Y = c_i)$, 由属性条件独立假设, x_j 之间是独立的, 所以我们有

$$P(X^1 = x^1, \dots, X^n = x^n|Y = c_i) = P(X = x|Y = c_i) = \prod_{j=1}^n P(X^j = x^j|Y = c_i)$$

3. 计算 c_i 在 x 下的后验概率:

$$P(Y = c_i|X = x) = \frac{P(Y = c_i, X = x)}{P(X = x)}$$

$$\begin{aligned}
&= \frac{P(X = x|Y = c_i)P(Y = c_i)}{\sum_i P(X = x, Y = c_i)} \\
&= \frac{P(X = x|Y = c_i)P(Y = c_i)}{\sum_i P(X = x|Y = c_i)P(Y = c_i)}
\end{aligned}$$

代入 2 中的 $P(X = x|Y = c_i)$, 我们可以得到

$$\frac{P(Y = c_i) \prod_{j=1}^n P(X^j = x^j|Y = c_i)}{\sum_i P(Y = c_i) \prod_{j=1}^n P(X^j = x^j|Y = c_i)}$$

此即为 c_i 在 x 下的后验概率, 我们只需取概率最大的那个类别即可, 因为分母是所有 c_i 的求和, 它对所有的 c_i 都相等, 所以我们可以忽略分母, 最终实例 x 的类别可以表示为:

$$y = f(x) = \operatorname{argmax}_{c_i} P(Y = c_i) \prod_{j=1}^n P(X^j = x^j|Y = c_i)$$

我们可以看出, Navie Bayes 做出了样例的每个属性是独立的假设从而避免了当 n 很大时造成的组合数计算灾难。当然在现实生活中此假设往往是不成立的, 但 Navie Bayes 有其优点, 对小规模的数据表现很好, 对缺失数据不敏感, 能处理多分类任务, 在文本分类譬如垃圾邮件过滤和新闻分类等问题上表现良好。

2.2 Random Forest

随机森林实质是对决策树算法的一种改进, 将多个决策树合并在一起。因此随机森林的结果是依赖于多棵决策树的结果, 这是一种集成学习的思想。它用随机的方式建立一个森林, 随机森林的每一棵决策树之间是没有关联的。在得到森林之后, 当有一个新的输入样本进入时, 就让森林中的每一棵决策树分别进行判断, 判断这个样本应该属于哪一类, 采用少数服从多数, 哪一类别被选择最多, 就预测这个样本为那一类。

其算法流程为:

1. 用 bootstrap 法在原始数据集中有放回随机抽样 k 组子数据集
2. 从样本的 N 个特征随机抽样 m 个特征
3. 对每个子数据集构建树分类器
4. 对于新的输入数据, 根据 K 个树分类器的分类结果, 少数服从多数得到最终结果

在训练过程中, 每棵树最大限度地生长, 不做任何修剪。随机森林简单, 容易实现, 计算开销小, 其随机性体现在随机采样(bootstrap): 在计算每棵树时, 从全部训练样本 (样本数为 n) 中选取一个可能有重复的、大小同样为 n 的数据集进行训练, 而特征选取的随机性体现在每个节点随机选取所有特征的一个子集, 用来计算最佳分割方式。随机森林的表现性能很好, 不容易过拟合, 泛化能力强。

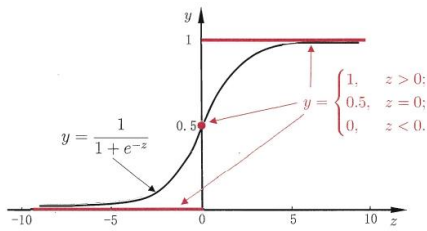
2.3 Logistic Regression

逻辑斯蒂回归其实是一种分类算法, 也称为对数几率回归。

考虑二分类任务 ($y \in \{0, 1\}$), 我们用 Sigmoid 函数(对数几率函数)将线性回归 $z = w^T x + b$ 的预测值映射到 $[0, 1]$ 之间:

$$y = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(w^T x + b)}}$$

若 y 大于 0.5，则归类 y 为 1；反之将 y 归类为 0。



(对数几率函数)

上式可推导出 $\ln \frac{y}{1-y} = w^T x + b$, y 可以看做是样本 x 分类为 1 的概率, $1-y$ 为其分类为 0 的概率, 其比值

称为几率。故重写为 $\ln \frac{P(y=1|x)}{P(y=0|x)} = w^T x + b$, 我们可以得到 $P(y=1|x) = \frac{e^{w^T x + b}}{1 + e^{w^T x + b}}$, $P(y=0|x) = \frac{1}{1 + e^{w^T x + b}}$ 。

我们用极大似然法来估计参数 w 与 b 。最大化训练样本的对数似然(log-likelihood)损失函数:

$$l(w, b) = \sum_{i=1}^m \ln P(y_i | x_i; w, b)$$

令 $\beta = (w; b)$, $\hat{x} = (x; 1)$, 则 $w^T x + b = \beta^T \hat{x}$ 。令 $P_1(\hat{x}; \beta) = P(y=1|\hat{x}; \beta)$, 则

$P_0(\hat{x}; \beta) = P(y=0|\hat{x}; \beta) = 1 - P_1(\hat{x}; \beta)$, 将其代入到似然函数中

$P(y_i | x_i; w, b) = y_i P_1(\hat{x}; \beta) + (1 - y_i) P_0(\hat{x}; \beta)$ 。这样一来, 一个式子就可以表达出 y 取 0 和 1 两种情况。

所以代入对数似然函数后就相当于最小化

$$l(\beta) = \sum_{i=1}^m (-y_i \beta^T \hat{x}_i + \ln(1 + e^{\beta^T \hat{x}_i}))$$

它是关于 β 的高阶连续可导函数, 可以使用梯度下降法、牛顿法求解。

而对于多分类问题, 我们可以逐个分类(one v.s. rest)、成对分类(one v.s. another)、分组分类(many v.s. many)的方法组合进行分类。

对数几率回归无需事先假设数据分布, 计算代价不高, 对数据中小噪声的鲁棒性较好, 在流行病学和医学中有较好的应用如胃癌的预测。

2.4 AdaBoost

AdaBoost(Adaptive Boosting)是典型的 Boosting 算法, 属于 Boosting 家族的一员。Boosting 算法是一族将

“弱学习算法”提升为“强学习算法”的算法：先从初始训练集中训练出一个弱学习器，据其表现调整样本分布，然后再训练下一个弱学习器，重复进行直至基学习器数目达到了指定值，组合这些弱学习器得到一个强学习器。AdaBoost 算法就是 Boosting 族算法中以指数作为损失函数的算法。算法有两个部分：加法模型和前向分布。

加法模型： $F_M(x; P) = \sum_{m=1}^n \beta_m h(x; a_m)$ ， h 是一系列弱学习器， a_m 为其学习到的最优参数， β_m 为其在强学习器中所占比重。

前向分布： $F_m(x) = F_{m-1}(x) + \beta_m h(x; a_m)$ ，训练过程中，下一轮迭代产生的学习器是在上一轮的基础上训练得来的。

AdaBoost 在迭代过程中会改变训练样本的概率分布，减小上一轮被正确分类的样本权值，提高那些被错误分类的样本权值。并且，AdaBoost 采用加权多数表决的方法，加大分类误差率小的弱学习器的权重，减小分类误差率大的弱学习器的权重。

AdaBoost 算法流程如下：

输入训练集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ ，迭代次数 M 。

1. 初始化训练样本的权值： $D_1 = (w_{1,1}, w_{1,2}, \dots, w_{1,n})$ ， $w_{1,i} = 1/n$ 。

2. For $m=1$ to M

2.1 用权值为 D_m 的训练集进行学习得到学习器 $G_m(x)$ 。

2.2 计算 $G_m(x)$ 在训练集上的训练误差 (I) 为指示函数， $I(\text{true})$ 返回 1， $I(\text{false})$ 返回 0)

$$e_m = \sum_{i=1}^n w_{m,i} I(G_m(x_i) \neq y_i)$$

2.3 计算 $G_m(x)$ 在强学习器中的权重

$$\alpha_m = \frac{1}{2} \ln \frac{(1 - e_m)}{e_m}$$

2.4 更新训练集的权值分布

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

3. 得到强学习器

$$F(x) = \text{sign}\left(\sum_{i=1}^n \alpha_m G_m(x)\right)$$

可以看出，AdaBoost 算法可以使用不同的学习算法来构建弱学习器，不容易发生过拟合，且充分考虑了每个学习器的权重。在训练过程中，可以通过交叉验证来确定弱学习器的数目。但是它对异常样本敏感，异常样本在迭代中可能会获得较高的权重，影响最终的强学习器的预测准确性。

3 原有特征提取方法下的结果

3.1 Navie Bayes

分类结果如下，4 个类别的 F-Measure 都在 0.5 左右：

Weka Explorer

Preprocess | **Classify** | Cluster | Associate | Select attributes | Visualize

Classifier: Choose **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.DecisionStump

Test options:

- ☒ Use training set
- ☐ Supplied test set (Set...)
- ☐ Cross-validation (Folds: 10)
- ☐ Percentage split (%: 66)
- More options...

(Nom) class: **Start** **Stop**

Result list (right-click for options):

- 11:52:23 - bayes.NaiveBayes
- 12:06:17 - trees.RandomForest
- 12:06:53 - functions.Logistic
- 12:07:35 - meta.AdaBoostM1

Classifier output

Correctly Classified Instances: 399 (45.4442 %)
 Incorrectly Classified Instances: 479 (54.5558 %)
 Kappa statistic: 0.2148
 Mean absolute error: 0.2744
 Root mean squared error: 0.4983
 Relative absolute error: 103.7874 %
 Root relative squared error: 137.186 %
 Total Number of Instances: 878

=== Detailed Accuracy By Class ===

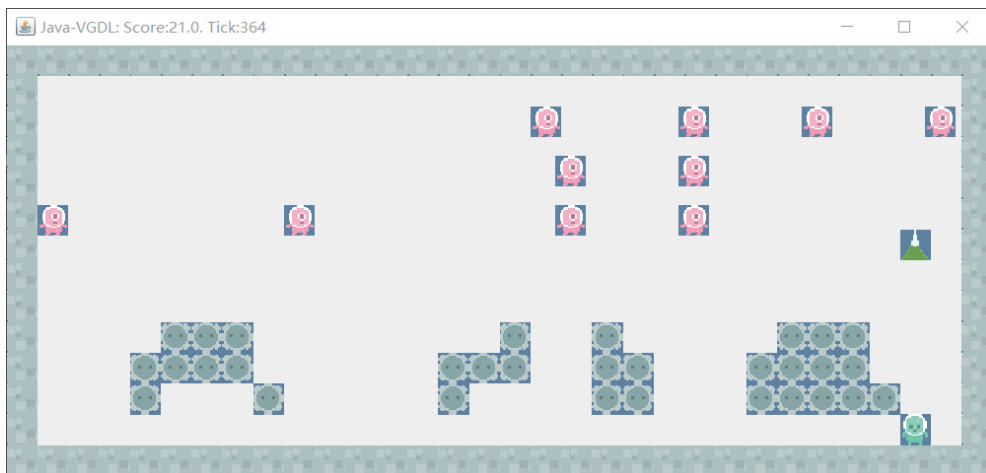
	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.352	0.19	0.763	0.352	0.482	0.635	0
	0.602	0.336	0.376	0.602	0.463	0.695	1
	0.532	0.019	0.61	0.532	0.568	0.925	2
	0.849	0.219	0.199	0.849	0.323	0.885	3
Weighted Avg.	0.454	0.219	0.623	0.454	0.472	0.681	

=== Confusion Matrix ===

	a	b	c	d	<-- classified as
196	210	16	135		a = 0
50	133	0	38		b = 1
10	4	25	8		c = 2
1	7	0	45		d = 3

Status: OK Log x 0

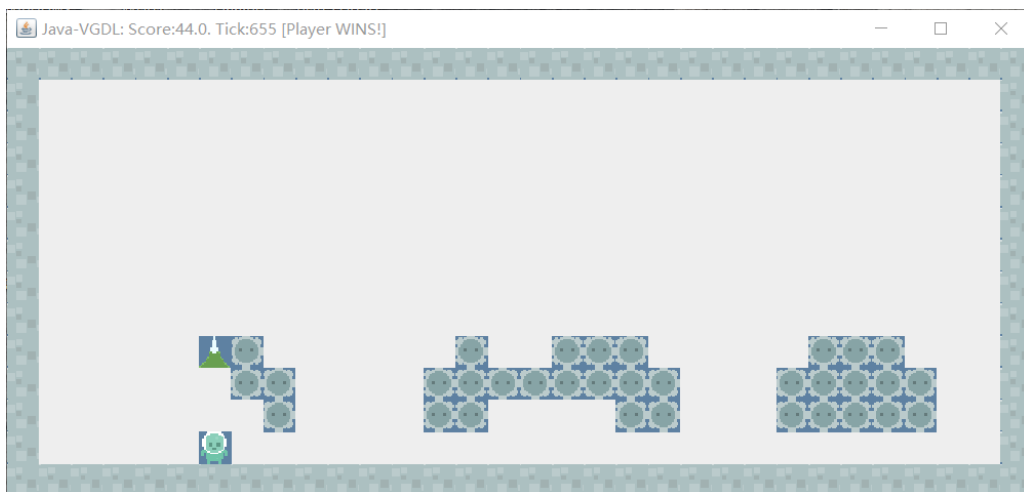
游戏过程中一直待在一个地方发射子弹。



最后阶段它不发射子弹了。



游戏成功截图



在游戏前期，学习到了待在一个位置持续发射来消灭敌人，在游戏中期，它学习到了移动来消灭最下层的敌人，在游戏末期，它学习到了等待剩余的几个敌人过来，但当敌人过来时它并未学会发射。当头顶落下子弹的时候它没有学会躲。

3.2 Random Forest

分类结果如下，各个类别的 F-Measure 均超出了 0.9，类别 3 接近 1：

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier: Choose **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.DecisionStump

Test options:

- ☒ Use training set
- ☐ Supplied test set
- ☐ Cross-validation Folds
- ☐ Percentage split %

(Nom) class

Result list (right-click for options):

- 11:52:23 - bayes.NaiveBayes
- 12:06:17 - trees.RandomForest**
- 12:06:53 - functions.Logistic
- 12:07:35 - meta.AdaBoostM1

Classifier output:

Correctly Classified Instances 840 95.672 %
 Incorrectly Classified Instances 38 4.328 %
 Kappa statistic 0.9176
 Mean absolute error 0.0822
 Root mean squared error 0.1529
 Relative absolute error 31.0993 %
 Root relative squared error 42.0899 %
 Total Number of Instances 878

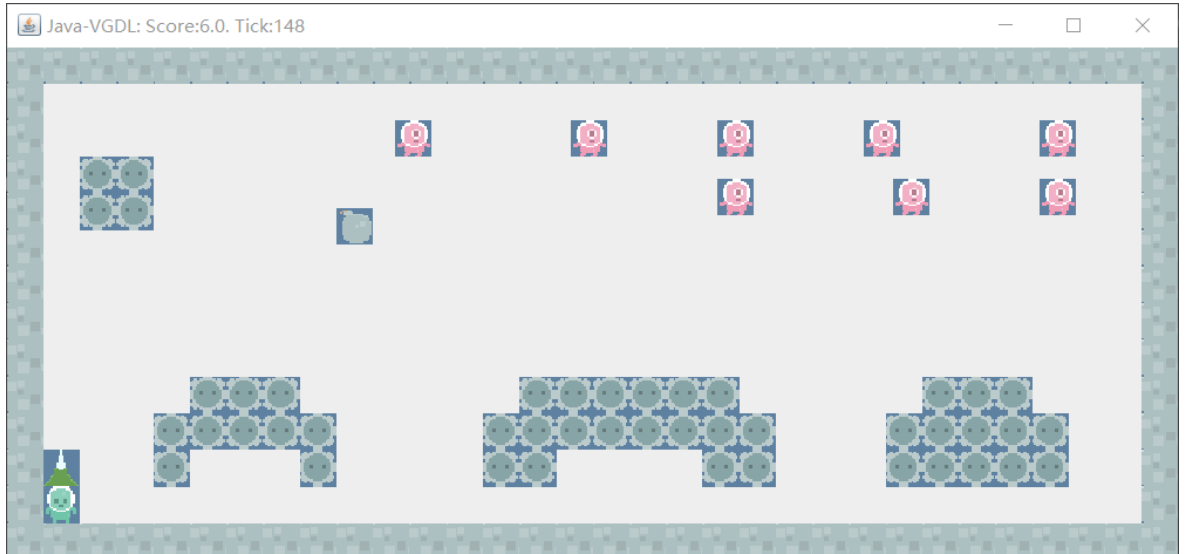
=== Detailed Accuracy By Class ===

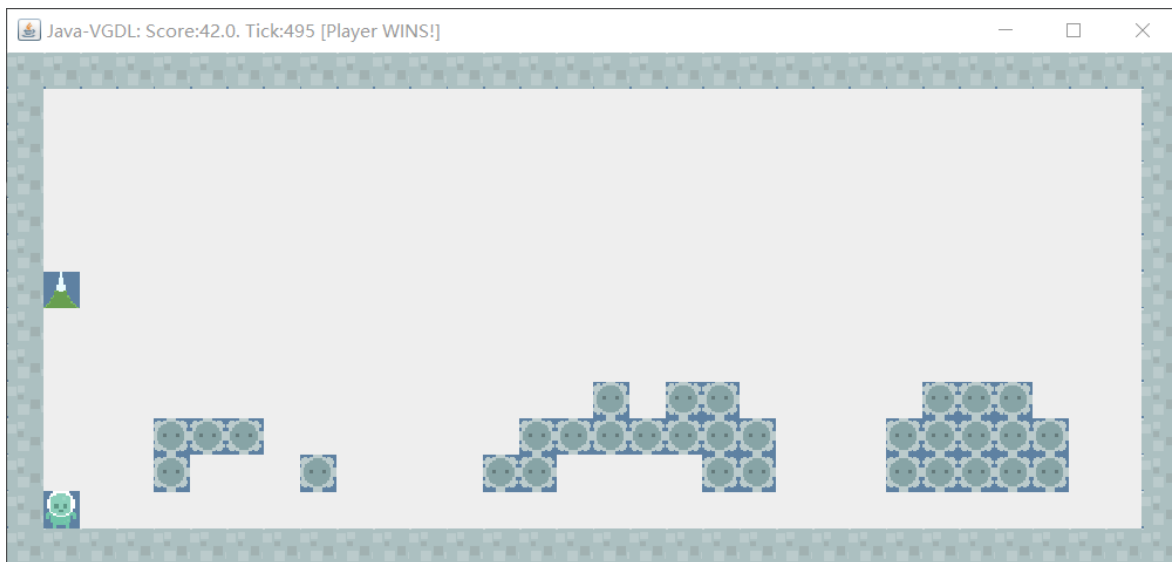
	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.971	0.069	0.961	0.971	0.966	0.993	0
	0.919	0.017	0.949	0.919	0.933	0.996	1
	0.915	0.005	0.915	0.915	0.915	0.999	2
	1	0.001	0.981	1	0.991	1	3
Weighted Avg.	0.957	0.048	0.957	0.957	0.957	0.995	

=== Confusion Matrix ===

a	b	c	d	<-- classified as
541	11	4	1	a = 0
18	203	0	0	b = 1
4	0	43	0	c = 2
0	0	0	53	d = 3

Status: OK x 0





它仅仅学习到了待到最左端持续发射子弹,从游戏结果来说这一做法大概率能导致游戏赢, RandomForest 学习到了最聪明最省力的方法。

3.3 Logistic Regression

分类结果如下, F-Measure 在 0.6~1 之间, 其中类别 3 的接近 1:

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier: Choose **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.DecisionStump

Test options:

- ☒ Use training set
- ☐ Supplied test set
- ☐ Cross-validation Folds
- ☐ Percentage split %

(Nom) class

Result list (right-click for options):

- 11:52:23 - bayes.NaiveBayes
- 12:06:17 - trees.RandomForest
- 12:06:53 - functions.Logistic**
- 12:07:35 - meta.AdaBoostM1

Classifier output:

Correctly Classified Instances 733 83.4852 %
 Incorrectly Classified Instances 145 16.5148 %
 Kappa statistic 0.6742
 Mean absolute error 0.1102
 Root mean squared error 0.235
 Relative absolute error 41.7046 %
 Root relative squared error 64.6888 %
 Total Number of Instances 878

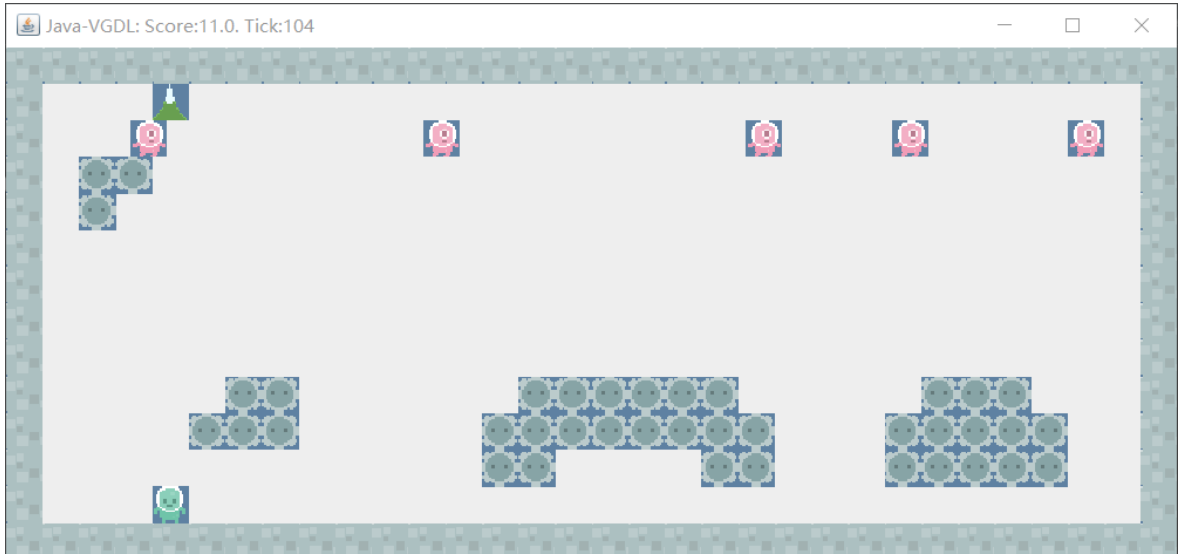
=== Detailed Accuracy By Class ===

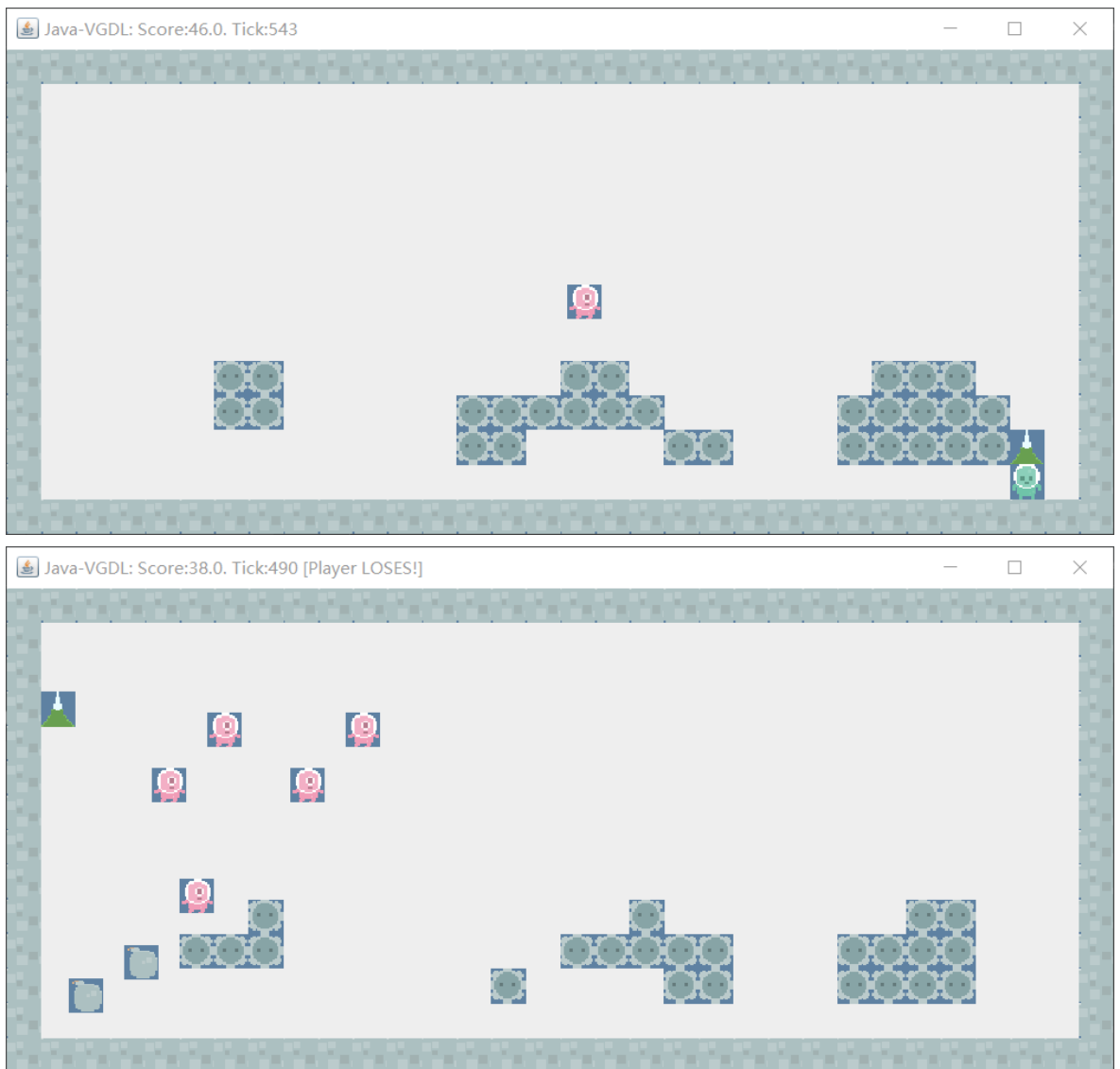
	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.912	0.299	0.841	0.912	0.875	0.918	0
	0.588	0.068	0.743	0.588	0.657	0.912	1
	0.894	0.004	0.933	0.894	0.913	0.999	2
	1	0.001	0.981	1	0.991	1	3
Weighted Avg.	0.835	0.207	0.83	0.835	0.829	0.926	

=== Confusion Matrix ===

	a	b	c	d	<-- classified as
508	45	3	1		a = 0
91	130	0	0		b = 1
5	0	42	0		c = 2
0	0	0	53		d = 3

Status: OK x 0





Logistic 学习到了待在左端持续发射子弹，且学会了左右移动来追逐目标，但它的有些移动是无目的的，体现在它会突然左右晃动然后回到原位置。且它没有学会躲避子弹。

3.4 AdaBoost

分类结果如下，所有类别都被分类为类别 0，类别 1、2、3 的 F-Measure 均为 0:

Weka Explorer

Preprocess | **Classify** | Cluster | Associate | Select attributes | Visualize

Classifier: Choose **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.DecisionStump

Test options:

- ☒ Use training set
- ☐ Supplied test set Set...
- ☐ Cross-validation Folds
- ☐ Percentage split %

More options...

(Nom) class ▼

Start Stop

Result list (right-click for options):

- 11:52:23 - bayes.NaiveBayes
- 12:06:17 - trees.RandomForest
- 12:06:53 - functions.Logistic
- 12:07:35 - meta.AdaBoostM1**

Classifier output:


Correctly Classified Instances 557 63.4396 %
 Incorrectly Classified Instances 321 36.5604 %
 Kappa statistic 0
 Mean absolute error 0.2565
 Root mean squared error 0.3581
 Relative absolute error 97.0379 %
 Root relative squared error 98.6014 %
 Total Number of Instances 878

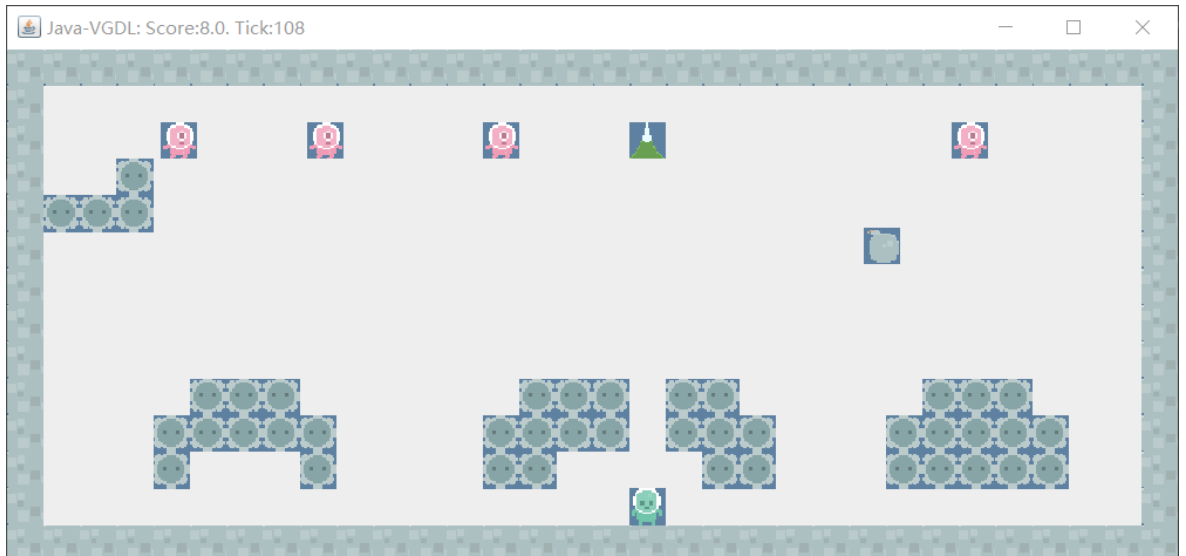
=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0	1	1	0.634	1	0.776	0.554	0
1	0	0	0	0	0	0.579	1
2	0	0	0	0	0	0.713	2
3	0	0	0	0	0	0.792	3
Weighted Avg.	0.634	0.634	0.402	0.634	0.492	0.583	

=== Confusion Matrix ===

	a	b	c	d	<-- classified as
557	0	0	0	0	a = 0
221	0	0	0	0	b = 1
47	0	0	0	0	c = 2
53	0	0	0	0	d = 3

Status: OK Log  x 0





AdaBoost 只学会了待在原地持续发射子弹，完全不会移动，虽然这样也可以赢，但若有子弹落头上的话就完蛋了。

3.5 四种学习方法的对比

在四种方法中，从达到游戏目标来说，AdaBoost 的效果最好，因为 Avatar 总是待在中间位置不停地射击，这样的话相比其他方法命中率会更高，而且被炸弹炸到的概率更低，所以它几乎总是赢得游戏，但是它没有学会躲避炸弹，也不会移动；而 Logistic Regression 的分类结果虽然误差小，但游戏效果并不好，过拟合较为严重；Random Forest 的分类误差也很小，它学习到的东西和 AdaBoost 差不多，都是待在同一个位置持续射击，但它是会移动的；Navie Bayes 分类结果的 F-Measure 比上述三种方法都低，但是它学习到了更多东西，会射击会移动，在游戏中表现比较灵活。唯一的共同点是，这四种方法都未学会躲避子弹。

综合来说，四种学习方法的效果比较是：Random Forest>AdaBoost>Navie Bayes>Logistic Regression。

4 改进特征提取方法

观察 Recoder.java, 函数 `featureExtract(StateObservation obs)` 是特征提取函数, 原始代码记录了屏幕上每个位置的信息(即每个位置是什么物体), 以及 4 个游戏状态信息: `GameTick`, `AvatarSpeed`, `AvatarHealthPoints`, `AvatarType`. 但是原始代码中的这四个游戏状态信息都存储在 `feature[448]` 中, 应该是不对的, 所以更改如下:

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

因为我们期待模型能学会躲避炸弹, 所以我们应该添加关于炸弹的 feature, 同时因为最下面一层的敌人是比较重要应该最先打死的, 所以我们也添加这一 feature, 躲避炸弹与追踪最下面一层的敌人与 Avatar 的横向移动有关, 所以我也添加了 Avatar 的横坐标这一 feature.

```

public static double[] featureExtract(StateObservation obs){

    double[] feature = new double[456]; // 448 + 4 + 3(add) + 1(class)
    Vector2d avatarPos=obs.getAvatarPosition();
    boolean Bomb=false;
    int LastyNum=0;
    double lastyLocation=10000;
    int AvatarPosX=(int) (avatarPos.x/25);
    // 448 locations
    int[][] map = new int[32][14];
    // 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()) 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(p.x-avatarPos.x)<=25){
            Bomb=true;
        }
        if(o.itype==6){
            if(p.y==lastyLocation){
                LastyNum+=1;
            }
            else if(p.y<lastyLocation){
                lastyLocation=p.y;
                LastyNum=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();
if(Bomb) feature[452]=100;
else feature[452]=0;
feature[453]=LastyNum;
feature[454]=AvatarPosX;
return feature;
}

```

在更改后的代码中,扩充 feature 为大小 456 的数组,取得 Avatar 的位置 avatarPos, 设置了一个布尔变量 Bomb 以指示是否遇到炸弹。LastyNum 为最下面一层的敌人数量, lastyLocation 为最下面一层的敌人的纵坐标, AvatarPosX 为 avatar 的横坐标。在 for 循环中,若发现 o 的 itype 为 5(炸弹)并且它的位置与 avatar 的位置在同一列,则判定有炸弹在 avatar 上方,将 Bomb 置为 true;若 o 的 itype 为 6(敌人 NPC),则判断它是不是当前最下面一层的敌人,并相应地更新 LastyNum。最后若有炸弹则将 feature[452]置为 100,否则为 0;feature[453]为 LastyNum,feature[454]为 AvatarPosX。同时在 datasetHeader()中添加相应的格式信息。

```

att = new Attribute("Bomb" ); attInfo.addElement(att);
att = new Attribute("lastyNum" ); attInfo.addElement(att);
att = new Attribute("AvatarPosX" ); attInfo.addElement(att);

```

5 改进特征提取方法后的结果

5.1 Navie Bayes

分类结果如下, F-Measure 较未改进特征提取方法方差要大些, 原有方法下各类别的 F-Measure 比较均匀, 现在类别 0 的 F-Measure 远超其余类别, 其余类别的 F-Measure 或有所降低或基本不变:

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier: Choose **NaiveBayes**

Test options:

- ☒ Use training set
- ☐ Supplied test set
- ☐ Cross-validation Folds
- ☐ Percentage split %

(Nom) class

Result list (right-click for options):

00:34:30 - bayes.NaiveBayes

Classifier output:


Correctly Classified Instances	378	58.4235 %
Incorrectly Classified Instances	269	41.5765 %
Kappa statistic	0.2952	
Mean absolute error	0.2101	
Root mean squared error	0.4251	
Relative absolute error	87.3292 %	
Root relative squared error	122.7869 %	
Total Number of Instances	647	

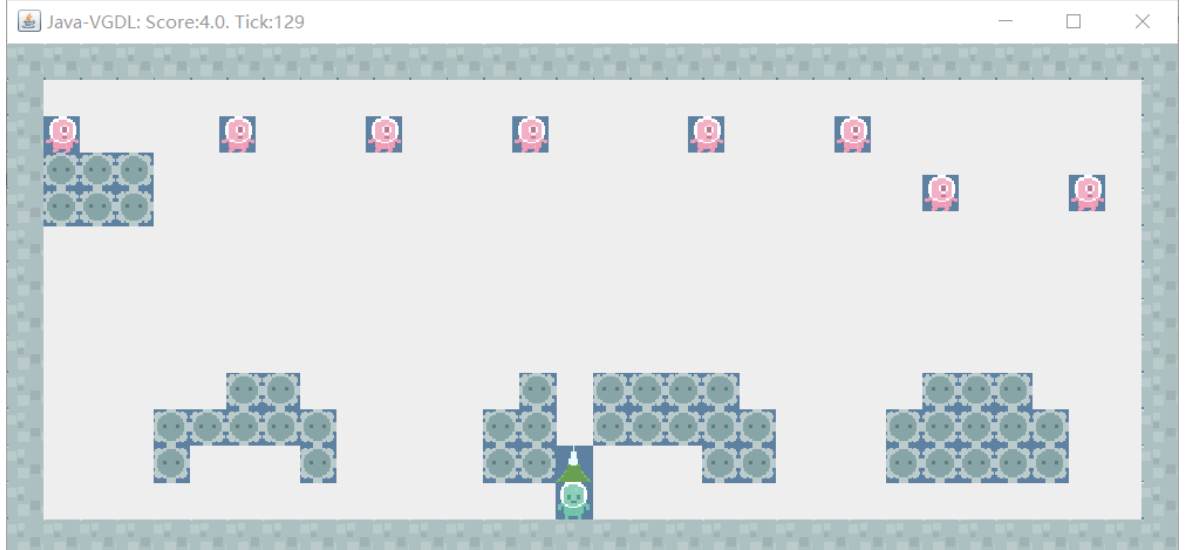
=== Detailed Accuracy By Class ===

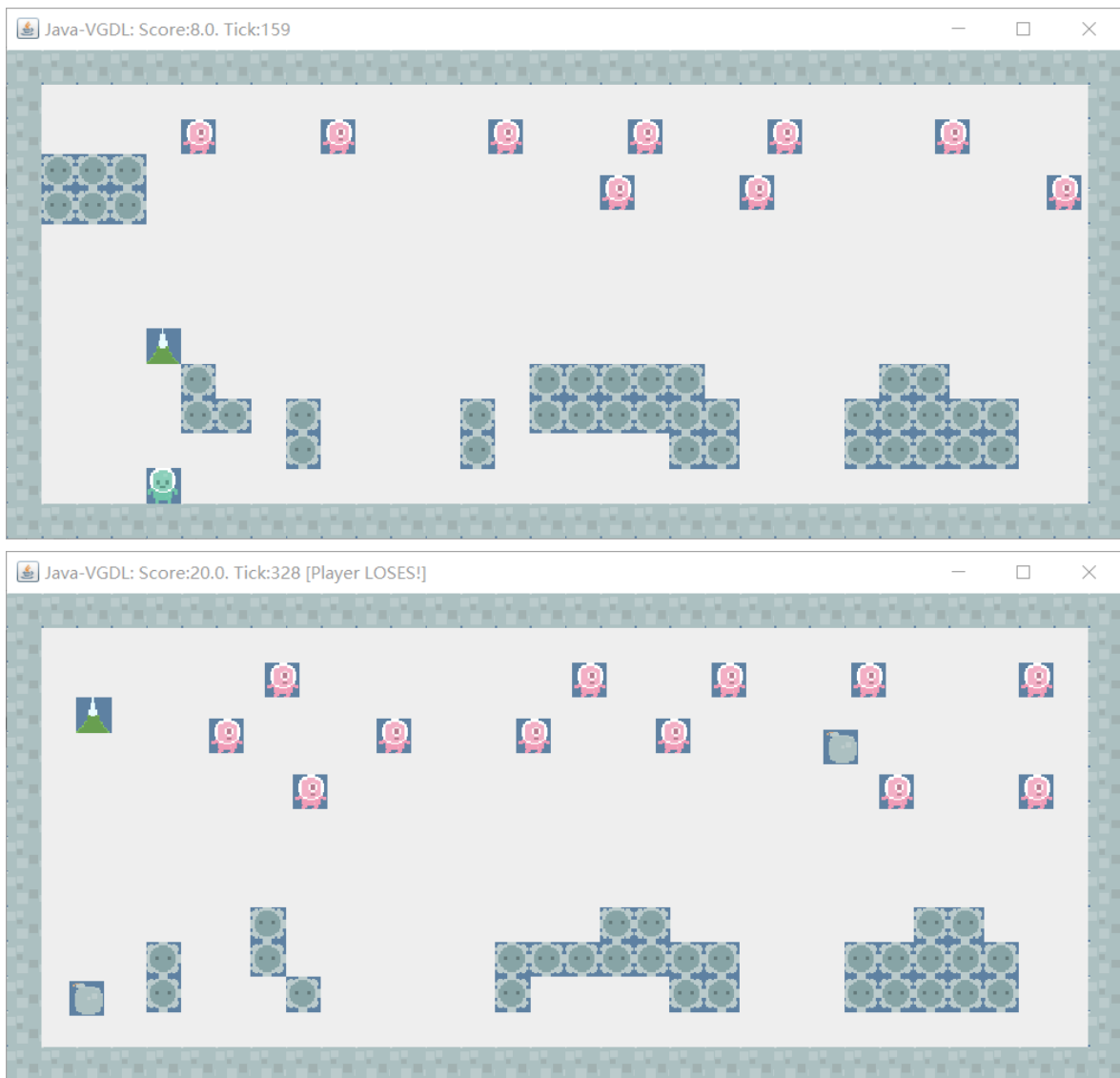
	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.606	0.254	0.83	0.606	0.7	0.723	0
	0.479	0.211	0.445	0.479	0.462	0.713	1
	0.632	0.08	0.194	0.632	0.296	0.937	2
	0.88	0.103	0.256	0.88	0.396	0.942	3
Weighted Avg.	0.584	0.232	0.688	0.584	0.614	0.735	

=== Confusion Matrix ===

	a	b	c	d	<-- classified as
263	96	34	41		a = 0
52	81	16	20		b = 1
1	3	12	3		c = 2
1	2	0	22		d = 3

Status: OK  x 0





改进特征提取方法后，明显发现 Avatar 的左右移动多了起来，而且学会了跟踪最下面一层的敌人去攻击，但是，它并没有学会躲避炸弹，而且当敌人到了倒数第二层时，它束手无策，只是待在那里不动。

5.2 Random Forest

分类结果如下，由混淆矩阵知每个类别都被精确分类，四个类别的 F-Measure 均为 1，把未改进提取方法之前的结果中的所有分类错误的样例都正确分类了：

Weka Explorer

Preprocess | **Classify** | Cluster | Associate | Select attributes | Visualize

Classifier: Choose **RandomForest** -I 100 -K 0 -S 1

Test options:

- ☒ Use training set
- ☐ Supplied test set
- ☐ Cross-validation Folds
- ☐ Percentage split %

(Nom) class

Result list (right-click for options):

- 00:34:30 - bayes.NaiveBayes
- 00:36:58 - functions.Logistic
- 00:37:50 - trees.RandomForest**

Classifier output:

Correctly Classified Instances 647 100 %
 Incorrectly Classified Instances 0 0 %
 Kappa statistic 1
 Mean absolute error 0.0598
 Root mean squared error 0.1068
 Relative absolute error 24.8775 %
 Root relative squared error 30.8573 %
 Total Number of Instances 647

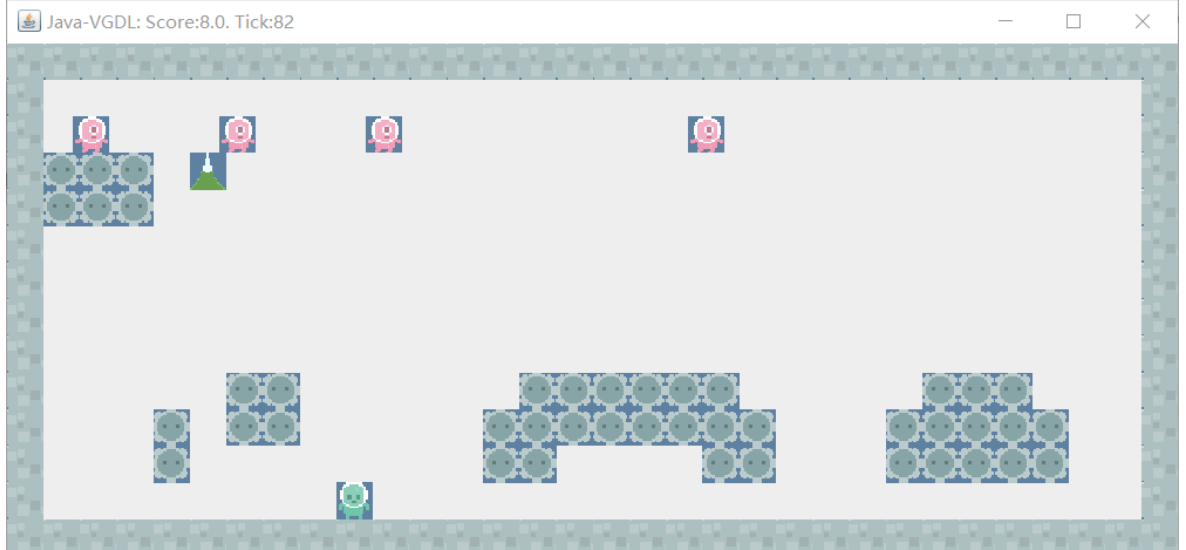
=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	0	1	1	1	1	0
	1	0	1	1	1	1	1
	1	0	1	1	1	1	2
	1	0	1	1	1	1	3
Weighted Avg.	1	0	1	1	1	1	

=== Confusion Matrix ===

a	b	c	d	<-- classified as
434	0	0	0	a = 0
0	169	0	0	b = 1
0	0	19	0	c = 2
0	0	0	25	d = 3

Status: OK x 0







Avatar 学会了移动，而且全程持续发射子弹而非像以前那样到了游戏后期敌人变少了之后就停止了发射子弹，即使有敌人到了最后一层，它也仍在发射子弹。刚开始时，它会先跑到左边一些来打敌人，然后跑到右边追踪敌人，接着经常性地左右移动来调整姿态攻击敌人。但是它还是没有学会躲避炸弹。

5.3 Logistic Regression

分类结果如下，相较之前的结果，各类别的 F-Measure 基本不变，其中类别 2 的 F-Measure 达到了 1:

Weka Explorer

Preprocess | **Classify** | Cluster | Associate | Select attributes | Visualize

Classifier: Choose **Logistic** -R 1.0E-8 -M -1

Test options:

- ☒ Use training set
- ☐ Supplied test set
- ☐ Cross-validation Folds
- ☐ Percentage split %

(Nom) class

Result list (right-click for options):

- 00:34:30 - bayes.NaiveBayes
- 00:36:58 - functions.Logistic**

Classifier output:


Correctly Classified Instances 540 83.4621 %
 Incorrectly Classified Instances 107 16.5379 %
 Kappa statistic 0.643
 Mean absolute error 0.1117
 Root mean squared error 0.236
 Relative absolute error 46.4412 %
 Root relative squared error 68.1539 %
 Total Number of Instances 647

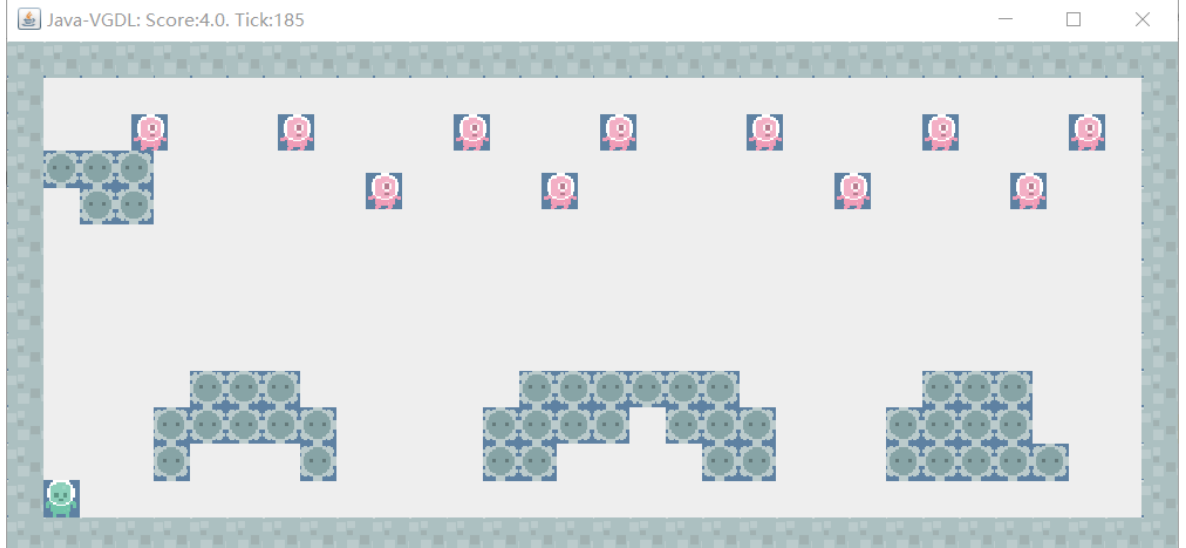
=== Detailed Accuracy By Class ===

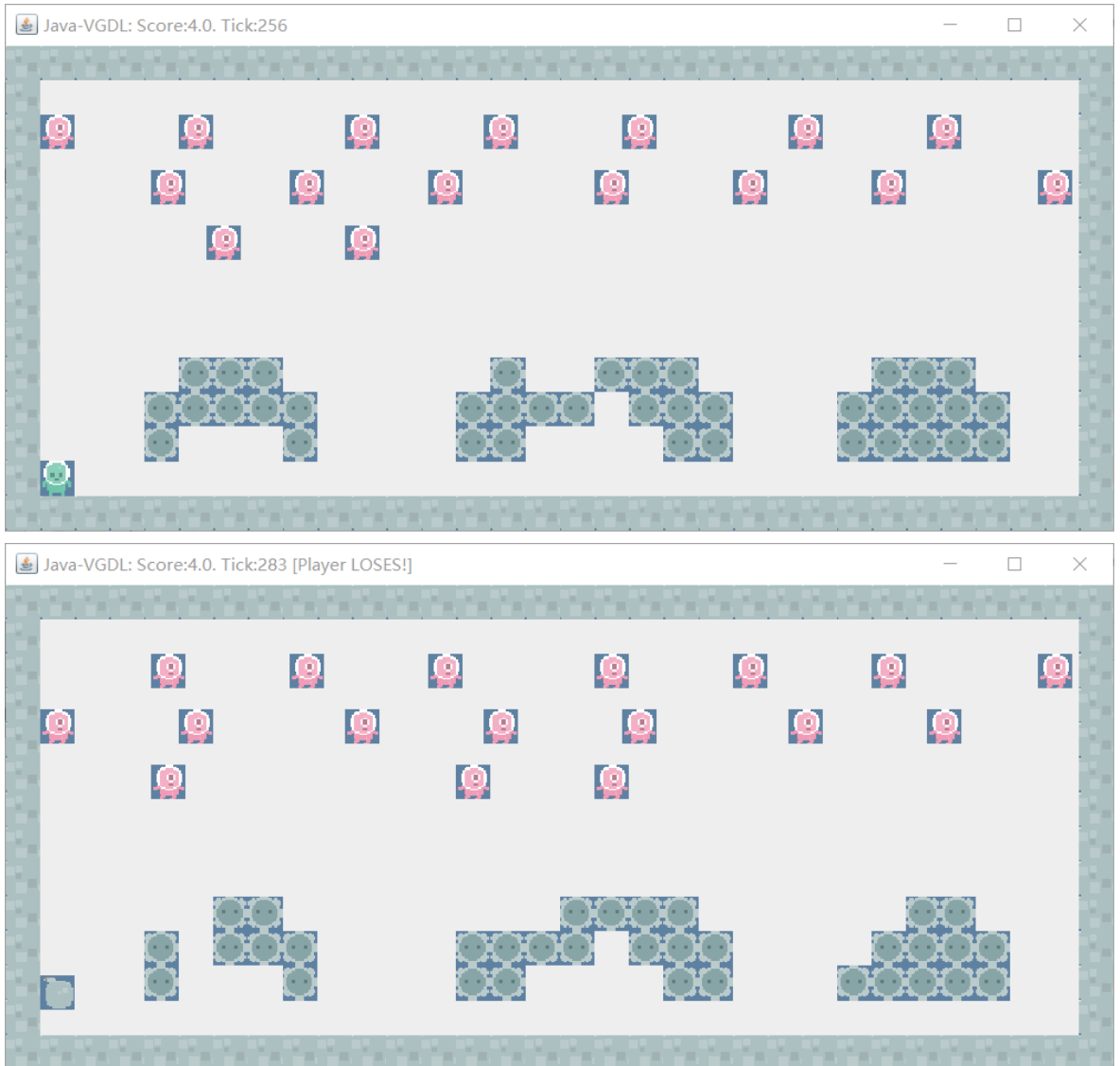
	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.908	0.305	0.858	0.908	0.882	0.911	0
	0.615	0.086	0.717	0.615	0.662	0.898	1
	1	0	1	1	1	1	2
	0.92	0.002	0.958	0.92	0.939	1	3
Weighted Avg.	0.835	0.227	0.83	0.835	0.831	0.914	

=== Confusion Matrix ===

a	b	c	d	<-- classified as
394	40	0	0	a = 0
64	104	0	1	b = 1
0	0	19	0	c = 2
1	1	0	23	d = 3

Status: OK  x 0





Avatar 先是向左移动一些然后攻击，接着移动到最左边，停止了攻击，直到(如果有)敌人扔下了炸弹被炸死。真是愚蠢，不知道为什么添加了特征后反而降低了模型的学习效果，使 Avatar 没有完全学习到发射子弹(发射子弹的重要性不言而喻)；或者是做一些无意义的移动(体现在迅速晃动然后回到了原位置)，然后隔一段时间放一发子弹，直到(如果有)敌人扔下了炸弹被炸死，或者敌人移动到了最后一层。

5.4 AdaBoost

分类结果如下，分类器依然将所有样例都分给了类别 0，类别 0 的 F-Measure 有所提高：

Weka Explorer

Preprocess | **Classify** | Cluster | Associate | Select attributes | Visualize

Classifier: Choose **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.DecisionStump

Test options:

- ☒ Use training set
- ☐ Supplied test set
- ☐ Cross-validation Folds
- ☐ Percentage split %

(Nom) class

Result list (right-click for options):

- 00:34:30 - bayes.NaiveBayes
- 00:36:58 - functions.Logistic
- 00:37:50 - trees.RandomForest
- 00:38:18 - meta.AdaBoostM1**

Classifier output:

Correctly Classified Instances 434 67.0788 %
 Incorrectly Classified Instances 213 32.9212 %
 Kappa statistic 0
 Mean absolute error 0.3184
 Root mean squared error 0.3772
 Relative absolute error 132.374 %
 Root relative squared error 108.9344 %
 Total Number of Instances 647

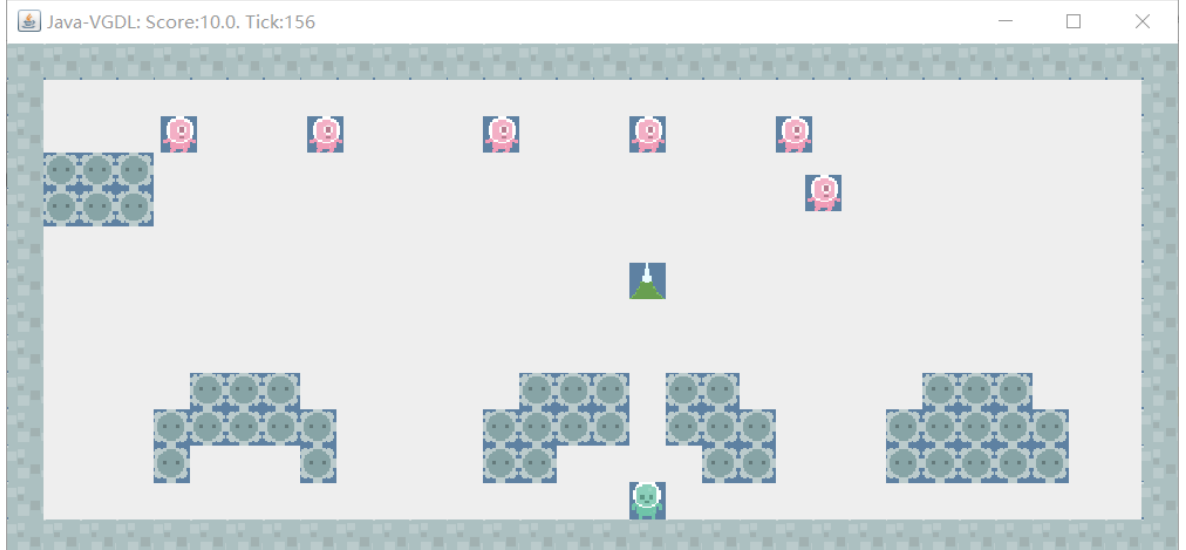
=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	1	0.671	1	0.803	0.63	0
	0	0	0	0	0	0.642	1
	0	0	0	0	0	0.66	2
	0	0	0	0	0	0.412	3
Weighted Avg.	0.671	0.671	0.45	0.671	0.539	0.626	

=== Confusion Matrix ===

	a	b	c	d	<-- classified as
434	0	0	0	0	a = 0
169	0	0	0	0	b = 1
19	0	0	0	0	c = 2
25	0	0	0	0	d = 3

Status: OK x 0





Avatar 只学会了持续发射子弹，完全不会移动，从混淆矩阵来看，分类器将所有样例都分为了一个类别。从而可以看出，新增的特征对于学习效果几乎没有提升，虽然这样常常导致赢得游戏，但是还是比较愚蠢。

5.5 四种学习方法的对比

在四种方法中，AdaBoost 的效果未得到明显的提升，依然将所有样例都分给了同一个类别，Avatar 的表现也和原来一样总是待在中间位置不停地射击，它也总是赢得游戏，没有学会躲避炸弹，也不会移动；Logistic Regression 的效果不升反降，丢掉了持续发射子弹的习惯，移动更频繁；Random Forest 的分类误差降到了极小，相比之前效果有较大提升，它在游戏进行中一直持续发射子弹，而且会移动，会追踪敌人，游戏表现更为灵活；Navie Bayes 移动变多，也学习到了跟踪最下层的敌人，表现灵活。而这四种方法都未学会躲避子弹，我认为原因是训练集中关于躲避炸弹的操作太少了，因为本来炸弹掉到正上方的概率就小，并且若刻意地想要跑到炸弹下面然后躲避它，只会被炸弹炸死(炸弹下落的速度很快，手速不行)。我相信若训练集中躲避炸弹的操作变多的话，应该能学会躲避炸弹。

综合来说，四种学习方法的效果比较仍是：Random Forest>AdaBoost>Navie Bayes>Logistic Regression

6 结束语

在这次的实验中，我介绍了四种学习方法：NavieBayes，Random Forest，logistic，AdaBoost，并将其运用在 Aliens 游戏中，比较了这四者的学习效果，然后改进了特征提取方法，进行了学习效果的横向对比和纵向对比。

附中文参考文献：

- [1] 周志华. 机器学习，清华大学出版社，2016
- [2] 黄书剑. 2019 年春<人工智能程序设计>课程讲义
- [3] 博客 <https://www.cnblogs.com/ScorpioLu/p/8295990.html>