
作业 3: Aliens 游戏 实验报告

丁云翔 (191250026, 191250026@smail.nju.edu.cn)

摘要: 本次作业目的是使用监督学习来模仿人玩游戏的动作。在本次作业中, 需要收集数据, 并训练学习模型。对于现有的特征提取方法, 收集训练数据, 尝试三种以上的学习方法, 撰写学习方法的介绍, 报告性能对比; 尝试修改特征提取方法, 得到更好的学习性能

关键词: RandomForest 算法、NaiveBayes 算法、AdaBoost 算法、KNN 算法、SimpleLogistic 算法、监督学习、特征提取方法。

1 任务 1

在任务一中, 我使用了两种游戏策略, 一种是时间最短型策略, 即开局后左移到地图中间偏左侧的空窗区, 按照外星人依次出现的时间间隔进行射击, 将所有外星人一个不漏地消灭在最高层, 这种策略在一开始移动至指定位置后就不再移动, 仅有射击动作, 因此也不牵涉到追击和躲避炸弹的操作; 另一种是混合型策略, 开局时与第一种类似, 同样是左移到地图中间偏左侧的空窗区定点射击, 但漏掉的外星人较多, 当外星人下降高度后, 采用移动攻击的策略, 期间有躲避炸弹的操作, 当只剩最后一个外星人时, 采用追击策略将其击落, 整个过程尽可能使时间最短。接下来将分别对这两个策略得到的训练数据进行学习。

关于学习方法, 我选择了 RandomForest 算法、NaiveBayes 算法、AdaBoost 算法、KNN 算法、SimpleLogistic 算法, 接下来将分别介绍。

学习方法介绍:

RandomForest 算法又称随机森林算法, 是指通过集成学习的思想将多棵决策树集成的一种算法。对于一个输入, 森林中的每一棵决策树都会有一个分类结果。而随机森林算法就是将森林中所有决策树的分类结果进行汇总, 并将其中个数最多的结果输出。这就要求每棵决策树的输入必须尽可能的不相关, 因为不同决策树的输入相关性越大, 其输出的相似性就越大。举一个极端的例子, 如果森林中每棵树的输入都相同, 那么输出也应该都是一样的, 采用森林就没有必要了。所以这也正是随机森林算法“随机”的重要性, 森林中每棵树的输入是随机的, 采用有放回的随机抽样从样本集中抽取一部分数据作为输入进行分类, 得到输出结果。最后森林输出所有树的输出结果汇总中票数最多的结果。所以对于随机森林算法, 影响其分类能力的最重要的参数就是森林中每棵树随机抽取的数据的数量, 数量大了则每棵树之间的相关性增强, 从而使森林的分类能力减弱; 数量小了则每棵树自身的分类能力减弱, 也使森林的分类能力减弱。所以关键是找到二者之间的平衡。因为有每棵树的输入的随机性和基于统计结果进行决策这两种特性, 随机森林算法有良好的抗噪声能力和防止过拟合的能力。

AdaBoost 算法全称为 Adaptive Boosting 算法, 即自适应增强算法。它的核心是通过多轮迭代训练多个弱分类器, 并将它们组合成一个强分类器。算法的自适应体现在对不同样本权重的调整和对不同弱分类器权重的调整。算法步骤如下: (1) 初始化样本集中各个样本的权值。如果样本集中的样本数为 n , 则样本集中每个样本点的初始权重都为 $1/n$ 。(2) 训练各个弱分类器。在训练过程中, 如果某个样本被正确地分类, 那么它的权重就会降低; 反之, 它的权重就会提高, 更新权值后的样本集被用于训练下一个分类器。同时, 对于每一个弱分类器, 都将统计出其分类的正确率, 正确率越高的分类器权重越大, 反之权重越低。对所有分类器训练完一轮以后, 就进入下一轮迭代, 再次从第一个分类器开始训练, 直到达到预定的轮数或者准确率达到预定值。(3) 将迭代训练得到的所有弱分类器按照权值组合成强分类器。误差率低的弱分类器在最终分类器中的话语权较

大，反之较小。强分类器最后的输出结果是将各个弱分类器的输出结果加权求和后得到的票数最多的结果。AdaBoost 算法可以看作是随机森林算法的进化版，加入了迭代的环节，并且各个弱分类器的类型可以不同，各个样本的权重和各个弱分类器的权重也各不相同，并且每个弱分类器都可以使用所有样本进行训练。AdaBoost 算法也可以防止过度拟合，并且准确度很高。

NaiveBayes 算法也称朴素贝叶斯算法，其核心是贝叶斯定理。由贝叶斯定理的公式形式我们可以得出：对于给定样本的特征，样本属于某个类别的概率 $P(\text{某个类别}|\text{特征}) = (P(\text{特征}|\text{某个类别}) * P(\text{某个类别})) / P(\text{特征})$ 。而朴素贝叶斯算法就是选择在给定样本特征时，选出 $P(\text{某个类别}|\text{特征})$ 最大或所有特征的 $P(\text{某个类别}|\text{特征})$ 乘积最大的那个类别作为分类结果。这种方法的优点在于计算简便，速度较快，并且分类效果也还不错。

KNN 算法又称 K-近邻算法，其中的 K 是该算法最重要的参数。该算法的思想是通过找到需要分类的样本在特征空间中最临近的 K 个训练集中的样本，如果这些样本属于某个类别的个数最多，则需要分类的样本大概率也属于这个类别。因为 K 值的大小会影响 K 个样本中不同类别的样本的数量，所以 K 值的选取对结果的影响很大。因为采用 K 个样本进行决策而不是单一样本（K 等于 1 时除外），该算法的准确度较高。但是也有缺点，分类所需的训练集对结果影响较大，如果训练集的样本个数太少或太稀疏，可能会影响分类的准确性；而样本个数如果太多，计算所需的时间开销又会很大，因为找出最临近的 K 个样本必须计算出所有样本与待分类样本的距离。

SimpleLogistic 算法又称简单逻辑回归算法，使用 LogitBoost 来拟合逻辑回归模型。在每次迭代中，对于每个类值都将一个简单的线性回归模型添加到现有的线性模型中。一旦添加其他简单线性回归模型后，估计得出的逻辑回归模型的分类误差未能降低，迭代就会停止，从而得到一个最优的拟合函数。再利用得到的拟合函数对待分类样本进行分类。

第一种策略的学习结果：

RandomForest 算法：

Classifier output

Correctly Classified Instances	291	84.593 %
Incorrectly Classified Instances	53	15.407 %
Kappa statistic	0.6814	
Mean absolute error	0.1035	
Root mean squared error	0.2272	
Relative absolute error	41.5742 %	
Root relative squared error	64.57 %	
Total Number of Instances	344	

Detailed Accuracy By Class ==

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.917	0.259	0.839	0.917	0.876	0.932	0
	0.744	0.071	0.868	0.744	0.802	0.94	1
	0.667	0.006	0.667	0.667	0.667	0.995	2
	0	0	0	0	0	?	3
Weighted Avg.	0.846	0.182	0.848	0.846	0.844	0.936	

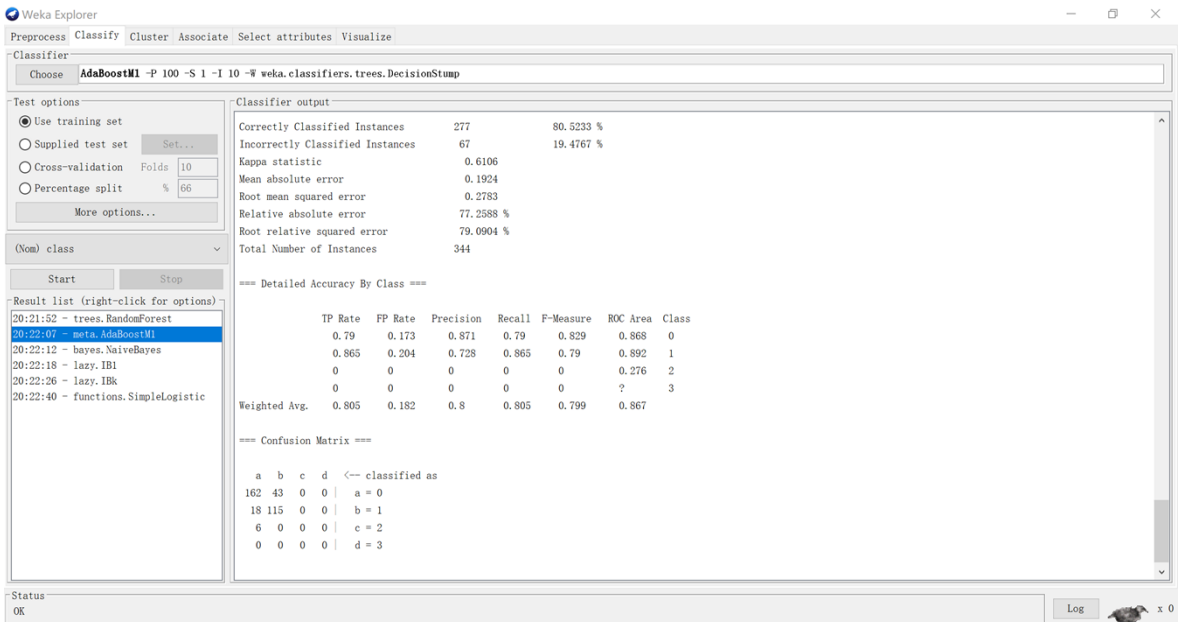
Confusion Matrix ==

a	b	c	d	← classified as
188	15	2	0	a = 0
34	99	0	0	b = 1
2	0	4	0	c = 2
0	0	0	0	d = 3

多次运行 Test.java，观察精灵的行为，我发现在一开局精灵就会移动到最左边的位置并保持静止，并且不断射击。这可能是由于我在进行游戏时先向左移动导致的，而且由于我游戏的时候向左移动后就没再移动过了，所以学习的结果就是向左移动后停留在原地一直射击。我游戏时控制的与外星人飞行间隔相同的射击间隔也没有被学习，精灵只是简单地采用连续射击的最短间隔。并且精灵也不会追击目标或者对炸弹做出闪避。（对

于第一种策略追击目标和对炸弹做出闪避的特性由于我的操作中没有，所以都不会习得，下面的四种算法也不再赘述）一般需要 500-600ticks 才能完成游戏。

AdaBoost 算法:



The screenshot shows the Weka Explorer interface with the AdaBoostM1 classifier selected. The 'Test options' section shows 'Use training set' selected. The 'Classifier output' section displays the following performance metrics:

Metric	Value
Correctly Classified Instances	277
Incorrectly Classified Instances	67
Kappa statistic	0.6106
Mean absolute error	0.1924
Root mean squared error	0.2783
Relative absolute error	77.2588 %
Root relative squared error	79.0904 %
Total Number of Instances	344

The 'Detailed Accuracy By Class' table is as follows:

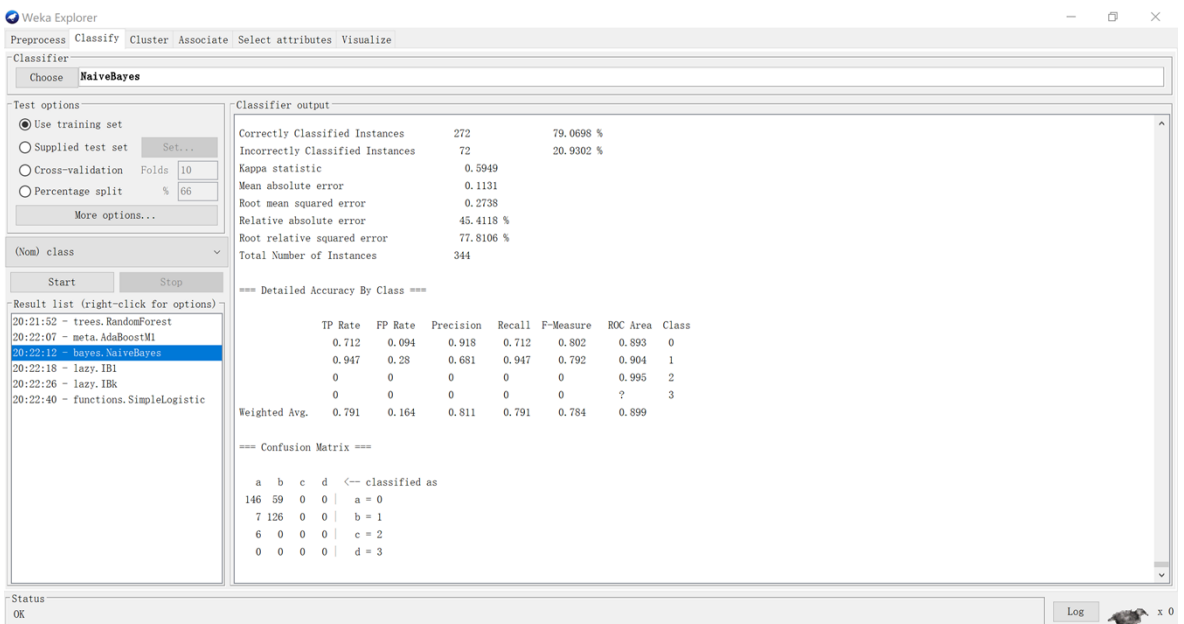
TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.79	0.173	0.871	0.79	0.829	0.868	0
0.865	0.204	0.728	0.865	0.79	0.892	1
0	0	0	0	0	0.276	2
0	0	0	0	0	?	3
Weighted Avg.	0.805	0.182	0.8	0.805	0.799	0.867

The 'Confusion Matrix' is as follows:

a	b	c	d	← classified as
162	43	0	0	a = 0
18	115	0	0	b = 1
6	0	0	0	c = 2
0	0	0	0	d = 3

多次运行 Test.java，观察精灵的行为，我发现精灵每次都是固定在起始位置不移动，并且一直在射击。对炸弹也不会闪避。如果精灵没被炸死，一般需要 444ticks 完成游戏。

NaiveBayes 算法:



The screenshot shows the Weka Explorer interface with the NaiveBayes classifier selected. The 'Test options' section shows 'Use training set' selected. The 'Classifier output' section displays the following performance metrics:

Metric	Value
Correctly Classified Instances	272
Incorrectly Classified Instances	72
Kappa statistic	0.5949
Mean absolute error	0.1131
Root mean squared error	0.2738
Relative absolute error	45.4118 %
Root relative squared error	77.8106 %
Total Number of Instances	344

The 'Detailed Accuracy By Class' table is as follows:

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.712	0.094	0.918	0.712	0.802	0.893	0
0.947	0.28	0.681	0.947	0.792	0.904	1
0	0	0	0	0	0.995	2
0	0	0	0	0	?	3
Weighted Avg.	0.791	0.164	0.811	0.791	0.784	0.899

The 'Confusion Matrix' is as follows:

a	b	c	d	← classified as
146	59	0	0	a = 0
7	126	0	0	b = 1
6	0	0	0	c = 2
0	0	0	0	d = 3

多次运行 Test.java，观察精灵的行为，我发现这种情况下必输无疑。精灵在一开局还是移动到最左边，并

且不断以最小射击间隔而不是我控制的射击间隔射击。当有外星人下降到第三高的层时，精灵会迅速移动到最右边并且停止射击，直到被炸死或被外星人抓住。

KNN 算法:

K=1:

The screenshot shows the Weka Explorer interface with the K=1 KNN classifier selected. The 'Classifier output' tab displays the following performance metrics:

Metric	Value
Correctly Classified Instances	277
Incorrectly Classified Instances	67
Kappa statistic	0.598
Mean absolute error	0.0974
Root mean squared error	0.3121
Relative absolute error	39.109 %
Root relative squared error	88.6992 %
Total Number of Instances	344

The 'Detailed Accuracy By Class' table is as follows:

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.888	0.317	0.805	0.888	0.845	0.786	0
0.684	0.09	0.827	0.684	0.749	0.797	1
0.667	0.012	0.5	0.667	0.571	0.827	2
0	0	0	0	0	?	3
Weighted Avg.						0.805 0.224 0.808 0.805 0.803 0.791

The 'Confusion Matrix' is shown below:

a	b	c	d	<- classified as
182	19	4	0	a = 0
42	91	0	0	b = 1
2	0	4	0	c = 2
0	0	0	0	d = 3

多次运行 Test.java，观察精灵的行为，一开始向左移动但没有移动到最左侧，而是停留在左边的空窗区一直射击，与我游戏时的操作相符，但只停留了一两秒就继续移动到最左侧并保持不动了。射击间隔依然是最小间隔而不是我控制的间隔。游戏时间在 550-600ticks 不等。

K=3:

The screenshot shows the Weka Explorer interface with the K=3 KNN classifier selected. The 'Classifier output' tab displays the following performance metrics:

Metric	Value
Correctly Classified Instances	289
Incorrectly Classified Instances	55
Kappa statistic	0.6702
Mean absolute error	0.1063
Root mean squared error	0.2307
Relative absolute error	42.7004 %
Root relative squared error	65.5674 %
Total Number of Instances	344

The 'Detailed Accuracy By Class' table is as follows:

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.907	0.259	0.838	0.907	0.871	0.927	0
0.744	0.081	0.853	0.744	0.795	0.937	1
0.667	0.006	0.667	0.667	0.667	0.995	2
0	0	0	0	0	?	3
Weighted Avg.						0.84 0.186 0.841 0.84 0.838 0.932

The 'Confusion Matrix' is shown below:

a	b	c	d	<- classified as
186	17	2	0	a = 0
34	99	0	0	b = 1
2	0	4	0	c = 2
0	0	0	0	d = 3

多次运行 Test.java，观察精灵的行为，发现 $K=3$ 时效果不如 $K=1$ 时，几乎必败。开局后会在偏左但不是我游戏时停留的位置停留一秒左右，然后继续移动至最左边，并且一开始还是连续射击，射击间隔依然是最小间隔而不是我控制的间隔，但之后的射击会出现不太容易察觉的更长的间断，导致不能击杀所有精灵，并最后被精灵抓住。

SimpleLogistic 算法:

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier: Choose SimpleLogistic -I 0 -M 500 -H 50 -W 0.0

Test options

- ☒ Use training set
- ☐ Supplied test set
- ☐ Cross-validation Folds 10
- ☐ Percentage split % 66

More options...

(Nom) class

Start Stop

Result list (right-click for options)

- 20:21:52 - trees.RandomForest
- 20:22:07 - meta.AdaBoostM1
- 20:22:12 - bayes.NaiveBayes
- 20:22:18 - lazy.IB1
- 20:22:26 - lazy.IBk
- 20:22:40 - functions.SimpleLogistic

Classifier output

Correctly Classified Instances 285 82.8488 %

Incorrectly Classified Instances 59 17.1512 %

Kappa statistic 0.6432

Mean absolute error 0.1258

Root mean squared error 0.2452

Relative absolute error 50.5237 %

Root relative squared error 69.6871 %

Total Number of Instances 344

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.888	0.259	0.835	0.888	0.861	0.901	0
	0.774	0.109	0.817	0.774	0.795	0.927	1
	0	0	0	0	0	0.875	2
	0	0	0	0	0	?	3
Weighted Avg.	0.828	0.196	0.814	0.828	0.82	0.911	

=== Confusion Matrix ===

a	b	c	d	<-- classified as
182	23	0	0	a = 0
30	103	0	0	b = 1
6	0	0	0	c = 2
0	0	0	0	d = 3

Status OK

多次运行 Test.java，观察精灵的行为，我发现精灵一开始也是移动到最左侧并开始连续射击，射击间隔依然是最小间隔而不是我控制的间隔。在 450-550ticks 时会出现间隔时长略微增加的情况，随后恢复正常。一般需要 718ticks 完成游戏。

第二种策略的学习结果:

RandomForest 算法:

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: 10
- ☐ Percentage split %: 66
- More options...

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

Result list (right-click for options):

- 17:48:25 - trees.RandomForest
- 17:48:39 - meta.AdaBoostM1
- 17:48:46 - bayes.NaiveBayes
- 17:48:54 - lazy.IB1
- 17:49:06 - lazy.IBk
- 17:49:36 - functions.SimpleLogistic

Classifier output:

```

Correctly Classified Instances   449      92.1971 %
Incorrectly Classified Instances  38      7.8029 %
Kappa statistic                 0.832
Mean absolute error             0.0984
Root mean squared error        0.1815
Relative absolute error         41.6572 %
Root relative squared error     52.9307 %
Total Number of Instances      487

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
      -----  -
      0.951    0.138    0.934    0.951    0.943    0.983    0
      0.85     0.045    0.876    0.85    0.863    0.983    1
      0.875    0      1      0.875    0.933    0.999    2
      1      0      1      1      1      1      3
Weighted Avg.   0.922    0.106    0.922    0.922    0.922    0.984

=== Confusion Matrix ===

  a  b  c  d  <-- classified as
312 16  0  0  | a = 0
 20 113 0  0  | b = 1
  2  0 14  0  | c = 2
  0  0  0 10  | d = 3

```

Status: OK

多次运行 Test.java，观察精灵的行为，我发现在一开局精灵就会移动到最左边的位置，并且不断射击。有几次会移动到最最后再向右移动几格，然后保持静止或再次移动到最左侧。这可能是由于我在进行游戏时先向左移动导致的，而有时出现的向右移动可能是因为右方目标较多，所以触发了追击策略。但是当只剩下最后一个目标时又不会触发追击策略，而是依旧停留在原地一直射击。并且精灵也不会对炸弹做出闪避。一般需要 500-700ticks 才能完成游戏。

AdaBoost 算法:

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: 10
- ☐ Percentage split %: 66
- More options...

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

Result list (right-click for options):

- 17:48:25 - trees.RandomForest
- 17:48:39 - meta.AdaBoostM1
- 17:48:46 - bayes.NaiveBayes
- 17:48:54 - lazy.IB1
- 17:49:06 - lazy.IBk
- 17:49:36 - functions.SimpleLogistic

Classifier output:

```

Correctly Classified Instances   328      67.3511 %
Incorrectly Classified Instances 159      32.6489 %
Kappa statistic                 0
Mean absolute error             0.2331
Root mean squared error        0.3414
Relative absolute error        98.6343 %
Root relative squared error     99.5531 %
Total Number of Instances      487

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
      -----  -
      1      1      0.674    1      0.805    0.511    0
      0      0      0      0      0      0.533    1
      0      0      0      0      0      0.614    2
      0      0      0      0      0      0.759    3
Weighted Avg.   0.674    0.674    0.454    0.674    0.542    0.525

=== Confusion Matrix ===

  a  b  c  d  <-- classified as
328  0  0  0  | a = 0
133  0  0  0  | b = 1
 16  0  0  0  | c = 2
 10  0  0  0  | d = 3

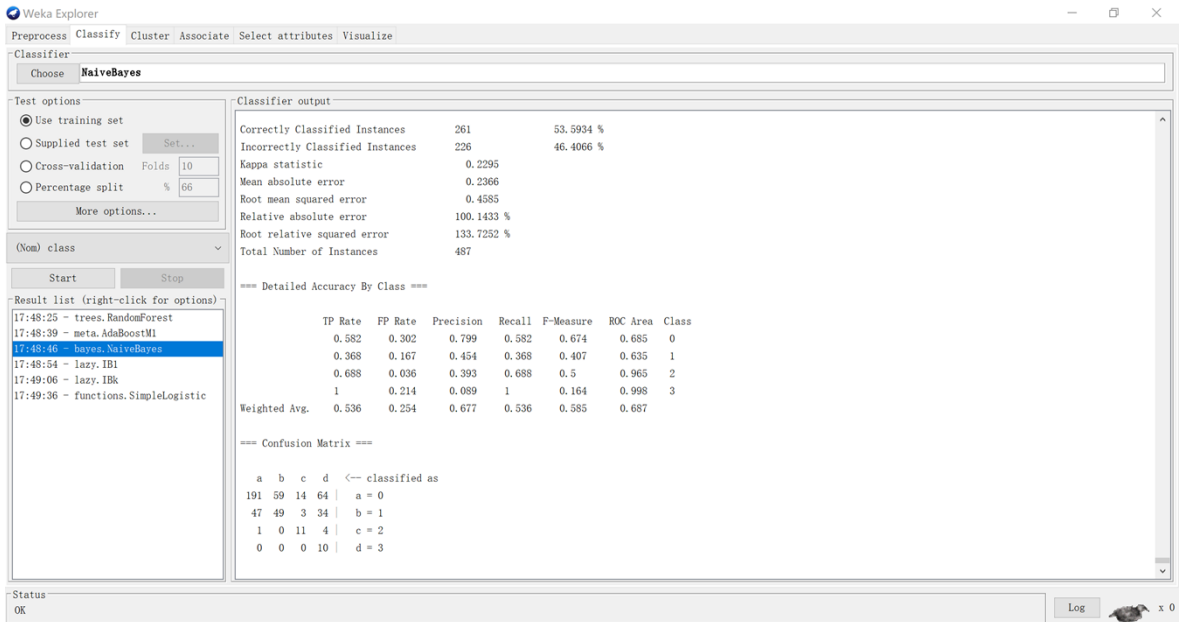
```

Status: OK

多次运行 Test.java，观察精灵的行为，我发现精灵的动作和第一种策略完全一样，每次都是固定在起始位置不移动，并且一直在射击。对炸弹也不会闪避。这种算法好像只学会了连续射击的动作。如果精灵没被炸

死，一般需要 444ticks 完成游戏。

NaiveBayes 算法:



The screenshot shows the Weka Explorer interface with the NaiveBayes classifier selected. The 'Test options' section on the left shows 'Use training set' selected. The 'Classifier output' section on the right displays the following results:

Correctly Classified Instances: 261 (53.5934 %)
 Incorrectly Classified Instances: 226 (46.4066 %)
 Kappa statistic: 0.2295
 Mean absolute error: 0.2366
 Root mean squared error: 0.4585
 Relative absolute error: 100.1433 %
 Root relative squared error: 133.7252 %
 Total Number of Instances: 487

Below the main output, there is a 'Detailed Accuracy By Class' table:

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0	0.582	0.302	0.799	0.582	0.674	0.685	0
1	0.368	0.167	0.454	0.368	0.407	0.635	1
2	0.688	0.036	0.393	0.688	0.5	0.965	2
3	1	0.214	0.089	1	0.164	0.998	3
Weighted Avg.	0.536	0.254	0.677	0.536	0.585	0.687	

At the bottom, there is a 'Confusion Matrix' section:

```

a  b  c  d  <- classified as
191 59 14 64 | a = 0
47 49  3 34 | b = 1
1  0 11  4 | c = 2
0  0  0 10 | d = 3
  
```

多次运行 Test.java，观察精灵的行为，我发现精灵的动作具有比较大的随机性，每次测试间的规律性并不强。除了一开局移动到最右方并开始连续射击以外（不知道为什么移到最右方，我游戏的时候是向左移开局的），后面的动作每次都不太相同。通过观察精灵的移动规律，发现其运动有一定的追击性质。在多次实验中我还发现，精灵在开始射击时，由于外星人在最右侧的格子需要下降高度，所以停留时间变长，导致按照连续射击的间隔正好可以将外星人依次消灭（也可能是其射击间隔与前两种算法不同，这种算法下精灵的射击间隔更接近外星人的飞行间隔），所以一开局精灵就能将大部分外星人击杀在最高层的最右侧。只剩最后一个目标时，精灵会开始四处移动，追击最后一个目标。美中不足的是，精灵依然没有学习到躲避炸弹的策略。游戏时间最快只需 414ticks，最慢需要 700+ticks。

KNN 算法:

K=1:

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier: Choose **IB1**

Test options:

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

(Nom) class: [dropdown]

Start Stop

Result list (right-click for options):

- 17:48:25 - trees.RandomForest
- 17:48:39 - meta.AdaBoostM1
- 17:48:46 - bayes.NaiveBayes
- 17:48:54 - lazy.IB1**
- 17:49:06 - lazy.IBk
- 17:49:36 - functions.SimpleLogistic

Classifier output:

Correctly Classified Instances 447 91.7864 %
 Incorrectly Classified Instances 40 8.2136 %
 Kappa statistic 0.8297
 Mean absolute error 0.0411
 Root mean squared error 0.2027
 Relative absolute error 17.3804 %
 Root relative squared error 59.0997 %
 Total Number of Instances 487

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.918	0.082	0.959	0.918	0.938	0.918	0
	0.91	0.071	0.829	0.91	0.867	0.92	1
	0.938	0.004	0.882	0.938	0.909	0.967	2
	1	0	1	1	1	1	3
Weighted Avg.	0.918	0.074	0.921	0.918	0.919	0.922	

=== Confusion Matrix ===

a	b	c	d	<-- classified as
301	25	2	0	a = 0
12	121	0	0	b = 1
1	0	15	0	c = 2
0	0	0	10	d = 3

Status: OK Log

多次运行 Test.java，观察精灵的行为，我发现这次模型的拟合效果更好，一开始向左移动但没有移动到最左侧，而是停留在左边的空窗区一直射击，与我游戏时的操作相符。当外星人高度下降时，开始左右移动并继续保持射击。但是移动只在外星人较多时有效，当外星人较少时，精灵反而固定不动了。精灵这次貌似学会了躲避炸弹，有时会做出闪避，但有时还是会无动于衷。游戏时间在 500-750ticks 不等。

K=3:

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier: Choose **IBk -K 3 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""**

Test options:

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

(Nom) class: [dropdown]

Start Stop

Result list (right-click for options):

- 17:48:25 - trees.RandomForest
- 17:48:39 - meta.AdaBoostM1
- 17:48:46 - bayes.NaiveBayes
- 17:48:54 - lazy.IB1
- 17:49:06 - lazy.IBk**
- 17:49:36 - functions.SimpleLogistic

Classifier output:

Correctly Classified Instances 390 80.0821 %
 Incorrectly Classified Instances 97 19.9179 %
 Kappa statistic 0.5212
 Mean absolute error 0.1473
 Root mean squared error 0.2611
 Relative absolute error 62.3435 %
 Root relative squared error 76.1579 %
 Total Number of Instances 487

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.945	0.497	0.797	0.945	0.865	0.875	0
	0.481	0.045	0.8	0.481	0.601	0.893	1
	0.813	0.004	0.867	0.813	0.839	0.997	2
	0.3	0	1	0.3	0.462	0.993	3
Weighted Avg.	0.801	0.347	0.804	0.801	0.784	0.886	

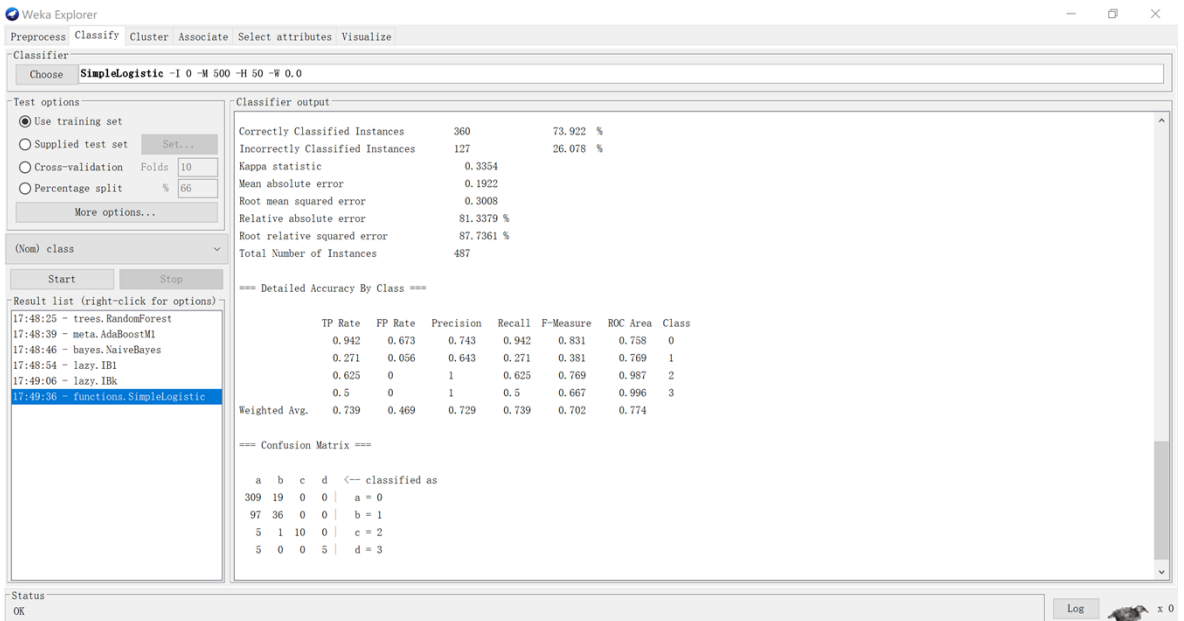
=== Confusion Matrix ===

a	b	c	d	<-- classified as
310	16	2	0	a = 0
69	64	0	0	b = 1
3	0	13	0	c = 2
7	0	0	3	d = 3

Status: OK Log

多次运行 Test.java，观察精灵的行为，发现 K=3 时效果不如 K=1 时，精灵的动作出现了倒退，比如开局依旧会移动至最左边，并且一开始还是连续射击，但之后的射击会出现间断，导致不能击杀所有精灵。并且对炸弹依旧并不会闪避。一般需要 700+ticks 才能完成游戏。

SimpleLogistic 算法:



多次运行 `Test.java`，观察精灵的行为，我发现精灵一开始也是移动到最左侧并开始连续射击，然后开始无规则运动，并且射击偶尔会出现较长间隔。不能确定这种无规则运动是不是为了追击。关于躲避炸弹，由于精灵的运动太过无规则，也不能确定是否与躲避炸弹有关，但是能观察到精灵确实有因运动而躲避过炸弹，但是由于精灵在中间阶段几乎一直在左右横跳并且偶尔也有被炸死的情况出现，所以不能确定二者间的关系。一般需要 550-750ticks 才能完成游戏。

性能对比：对于第一种策略，选取的五种算法都没有学习到核心操作，即利用控制射击间隔从而在最短的时间内击杀所有外星人，故不针对这种策略进行对比。我主要针对第二种策略的学习进行对比。对比五种算法的运行结果，将学习效果并从好到坏排序，我认为 $K=1$ 的 KNN 算法效果最好，与我游戏时实际的操作最接近，向左移、连续射击、左右移动追击、躲避炸弹（疑似）都学到了；其次是 NaiveBayes 算法，虽然没有学到向左移，但是它会向右移，并且连续射击和追击都学到了，但是不会躲避炸弹；再次是 SimpleLogistic 算法，学到了向左移、连续射击，但不能确定无规则运动是不是学会了追击和躲避炸弹；接下来是 RandomForest 算法，学到了向左移和连续射击，并且有疑似追击的行为；效果最差的是 AdaBoost 算法，除了连续射击其他的都没学会。关于五种算法普遍对躲避炸弹的学习能力较弱，我认为也有可能是训练集的问题，因为炸弹落在头顶的概率本就很小，再加上闪避的反应时间也很短，在一次游戏中想要闪避两次以上的炸弹攻击是很难做到的，所以我在游戏时也只有一次躲避炸弹的操作，可能因此导致样本量很小而难以被学习。

2 任务 2

首先原来 Recorder 类中的 `featureExtract` 函数存在一处错误，四个值都存在同一个位置了，如下图所示：

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

改正后如下:

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

我尝试修改特征提取方法,得到更好的学习性能。由于任务一中对五种学习方法进行性能对比时,发现对躲避炸弹和追击的学习效果普遍较差,而对左移和连续射击的学习效果普遍较好,所以我着重从有助于学习躲避炸弹和追击的角度对特征提取方法进行修改。躲避炸弹主要是靠观察精灵所在列有没有炸弹出现,如果有炸弹就移动躲避;追击主要是靠观察精灵左右两侧外星人数量的差值,向外星人多的方向移动追击。因此需要增加两个记录的特征,分别为所在的列是否有炸弹和精灵左右两侧外星人数量的差值。

具体操作:一是将 feature 数组的长度改为 455,然后在 featureExtract 函数中加入对上述两特征的提取和存储,二是修改 datasetHeader 函数。但是从学习结果来看,性能提升并不明显。我推测是只保存左右两侧外星人数量的差值并不准确,因为当只剩下最后一个外星人时,左右两侧的差值应该是 1 或-1,然而在还有很多外星人时,差值为 1 或-1 的情况也很常见,而这两种情况一种需要追击,另一种则不需要追击,所以容易混淆。因此我再加入了左侧外星人数量和右侧外星人数量两个特征,将上述两种情况更好地区分开。具体操作是将 feature 数组的长度改为 457,然后在 featureExtract 函数中加入增加的四个特征的提取和存储,并修改修改 datasetHeader 函数。修改后的代码如下:

```
// Extract features
boolean bomb = false;
int left = 0;
int right = 0;
double avatar_X = obs.getAvatarPosition().x;
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:l){
            double NPC_X = o.position.x;
            if(NPC_X < avatar_X){
                left++;
            }
            else if(NPC_X > avatar_X){
                right++;
            }
        }
    }
}
```

```

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 == Types.TYPE_FROMAVATAR && Math.abs(p.x - avatar_X) <= 25){
        bomb = true;
    }
}

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] = bomb ? 1000.0 : -1000.0;
feature[453] = left;
feature[454] = right;
feature[455] = left - right;

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( attributeName: "object_at_position_x=" + x + "_y=" + y);
            attInfo.addElement(att);
        }
    }
    Attribute att = new Attribute( attributeName: "GameTick" ); attInfo.addElement(att);
    att = new Attribute( attributeName: "AvatarSpeed" ); attInfo.addElement(att);
    att = new Attribute( attributeName: "AvatarHealthPoints" ); attInfo.addElement(att);
    att = new Attribute( attributeName: "AvatarType" ); attInfo.addElement(att);
    att = new Attribute( attributeName: "bomb"); attInfo.addElement(att);
    att = new Attribute( attributeName: "left");attInfo.addElement(att);
    att = new Attribute( attributeName: "right");attInfo.addElement(att);
    att = new Attribute( attributeName: "left - right");attInfo.addElement(att);
}

```

通过 Train.java，选用任务一中介绍的策略二完成一场游戏并收集到特征，同样使用五种学习方法进行学习，以下是学习结果。

RandomForest 算法：

The screenshot shows the Weka Explorer interface with the RandomForest classifier selected. The classifier output is displayed in the main window.

Classifier output

Correctly Classified Instances	862	100 %
Incorrectly Classified Instances	0	0 %
Kappa statistic	1	
Mean absolute error	0.0727	
Root mean squared error	0.1158	
Relative absolute error	26.9879 %	
Root relative squared error	31.5716 %	
Total Number of Instances	862	

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

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

==== Confusion Matrix ====

a	b	c	d	<- classified as
497	0	0	0	a = 0
0	308	0	0	b = 1
0	0	29	0	c = 2
0	0	0	28	d = 3

多次运行 Test.java，观察精灵的行为，我发现在一开局精灵会移动到偏左的位置，并且不断射击。然后会左右小范围移动，然后继续保持静止。这可能是由于我在进行游戏时先向左移动导致的，而出现的左右移动可能是触发了追击策略。但是当只剩下最后几个目标时又不会触发追击策略，而是依旧停留在原地一直射击。并且精灵依旧不会对炸弹做出闪避。一般需要 450-500ticks 完成游戏。

AdaBoost 算法:

The screenshot shows the Weka Explorer interface with the AdaBoostM1 classifier selected. The classifier output is displayed in the main window.

Classifier output

Correctly Classified Instances	497	57.6566 %
Incorrectly Classified Instances	365	42.3434 %
Kappa statistic	0	
Mean absolute error	0.2657	
Root mean squared error	0.3645	
Relative absolute error	98.6494 %	
Root relative squared error	99.4125 %	
Total Number of Instances	862	

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

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
1	1	0.577	1	0.731	0.51	0	
0	0	0	0	0	0.565	1	
0	0	0	0	0	0.763	2	
0	0	0	0	0	0.629	3	
Weighted Avg.	0.577	0.577	0.332	0.577	0.422	0.542	

==== Confusion Matrix ====

a	b	c	d	<- classified as
497	0	0	0	a = 0
308	0	0	0	b = 1
29	0	0	0	c = 2
28	0	0	0	d = 3

多次运行 Test.java，观察精灵的行为，我发现精灵的动作和未修改特征提取方法时完全一样，还是固定在起始位置不移动，并且一直在射击。对炸弹也不会闪避。多提取的特征对这种算法没有帮助。但是由于精灵不移动，所以游戏获胜的概率还是很高的。如果精灵没被炸死，一般需要 444ticks 完成游戏。

NaiveBayes 算法:

The screenshot shows the Weka Explorer interface with the NaiveBayes classifier selected. The 'Test options' section shows 'Use training set' is selected. The 'Classifier output' section displays the following data:

Correctly Classified Instances: 464 (53.8283 %)
 Incorrectly Classified Instances: 398 (46.1717 %)
 Kappa statistic: 0.2923
 Mean absolute error: 0.2292
 Root mean squared error: 0.4556
 Relative absolute error: 85.0794 %
 Root relative squared error: 124.256 %
 Total Number of Instances: 862

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.419	0.164	0.776	0.419	0.544	0.692	0
	0.669	0.327	0.532	0.669	0.593	0.707	1
	0.897	0.102	0.234	0.897	0.371	0.96	2
	0.857	0.086	0.25	0.857	0.387	0.956	3
Weighted Avg.	0.538	0.218	0.654	0.538	0.55	0.715	

=== Confusion Matrix ===

a	b	c	d	<- classified as
208	179	66	44	a = 0
56	206	19	27	b = 1
1	1	26	1	c = 2
3	1	0	24	d = 3

多次运行 Test.java, 观察精灵的行为, 我发现精灵在一开始会移动到最左侧, 并开始连续射击, 然后会在左半部分小范围左右移动。当剩余一两个外星人时, 精灵则会迅速地不规则地大范围地左右移动, 应该是追击策略发挥作用了, 但是运动实在是太无规律了所以击杀效率很低, 往往因此而失败。游戏时间需要 700-800ticks。

KNN 算法:

K=1:

The screenshot shows the Weka Explorer interface with the KNN classifier selected and K=1. The 'Test options' section shows 'Use training set' is selected. The 'Classifier output' section displays the following data:

Correctly Classified Instances: 862 (100 %)
 Incorrectly Classified Instances: 0 (0 %)
 Kappa statistic: 1
 Mean absolute error: 0
 Root mean squared error: 0
 Relative absolute error: 0 %
 Root relative squared error: 0 %
 Total Number of Instances: 862

=== Detailed Accuracy By Class ===

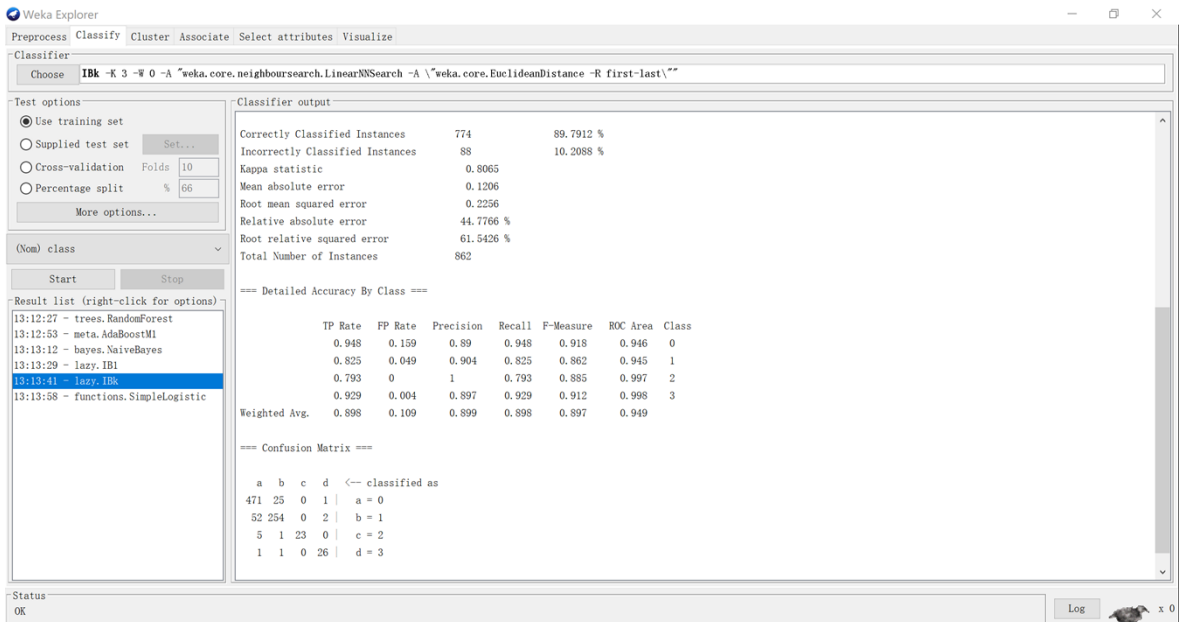
	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	0	1	1	1	1	0
	1	0	1	1	1	1	1
	1	0	1	1	1	1	2
	1	0	1	1	1	1	3
Weighted Avg.	1	0	1	1	1	1	

=== Confusion Matrix ===

a	b	c	d	<- classified as
497	0	0	0	a = 0
0	308	0	0	b = 1
0	0	29	0	c = 2
0	0	0	28	d = 3

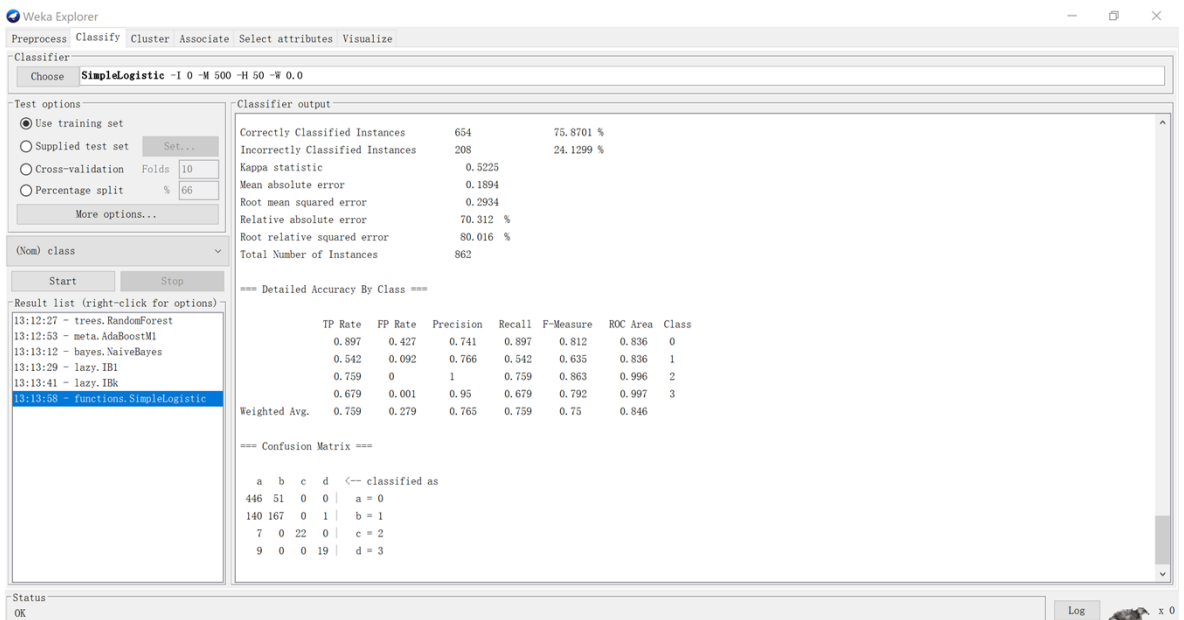
多次运行 Test.java，观察精灵的行为，我发现这次模型的拟合效果最好。开局移动到左侧空窗区并开始连续射击，然后开始移动追击，整体表现与我原先的游戏操作很相似。对于炸弹大部分情况下都能闪避，但也有少数情况下没有反应。一般能在 411ticks 完成游戏。

K=3:



多次运行 Test.java，观察精灵的行为，发现 K=3 时效果不如 K=1 时，但是学习效果也较为不错。精灵开局会移动至最左边，并且连续射击，之后会在左半侧左右移动进行追击。对闪避炸弹的学习效果一般，有时可以躲开，有时则不能。一般需要 500+ticks 完成游戏。

SimpleLogistic 算法:



多次运行 Test.java，观察精灵的行为，我发现一开始精灵的行为与 $K=1$ 时的 KNN 算法类似，也是移动到左侧空窗处并开始连续射击，然后开始左右移动追击。但是追击效果更差一些，并且当只剩下一两个外星人并且它们都在较低层时，射击会出现较长间隔，导致有可能因不能击杀最后一个目标而失败。对闪避炸弹的学习也一般，有时可以躲开，有时则不能。一般需要 500-850ticks 才能完成游戏。

与修改前的对比：修改后的五种算法学习效果并不是都有提升，但是最优的算法的学习效果确实比修改前更好了。效果最好的依旧是 $K=1$ 的 KNN 算法，开局向左移动到空窗区、连续射击、移动追击、躲避炸弹的学习效果都很不错，花费的时间也最少，相比修改特征提取方法之前效果提升明显。其次是 RandomForest 算法、NaiveBayes 算法和 SimpleLogistic 算法，三者在开局阶段表现都很好，都会向左移并连续射击，之后也会进行追击。但是当只剩下一两个外星人时，三种学习方法下精灵的表现各不相同，但效果都很差，常常导致游戏失败，可能是由于样本中没有涉及到只剩下一两个外星人并且都在较低层的情况的处理，所以无法准确地应对。另外对闪避炸弹的学习也一般。效果最差的依旧是 AdaBoost 算法，还是只学会了连续射击。总体效果上，修改特征提取方法后的学习性能相比修改前有所提升，尤其是在移动追击的动作上，但是闪避炸弹的学习效果提升幅度并不大，我推测应该还是躲避炸弹的样本数太少所致，在这次游戏中我闪避炸弹的次数达到了两次，虽然还是很少但基本已经很难做到比两次更多了。所以，bomb 这一特征为正数（即精灵所在列有炸弹）的样本实在太少了，容易被当作噪音而被忽略，所以学习效果会比较差。如果在一次游戏中成功躲避炸弹的次数能再多一些（虽然在当前游戏中很难做到了），学习效果应该会有很大改善。

修改特征提取方法后， $K=1$ 的 KNN 算法学习性能提升很大，基本达到了预期，明显优于未修改特征提取方法时的学习效果，任务完成。