作业 3: Aliens 游戏

王琛然（151220104、17721502736）

(南京大学 计算机科学与技术系, 南京 210093)

摘 要: 随着人工智能的迅速发展，如何训练机器、如何使机器学习成了主要的问题。机器学习的方法大体分为监督学习、无监督学习和半监督学习。本次实验仅涉及监督学习，讨论分析监督学习的几种常见算法：朴素贝叶斯算法、对数几率回归算法、决策树算法和随机森林四种算法。本次实验中，引入 Weka 机器学习包，对训练数据集分别调用不同的算法进行有监督学习。

关键词: 监督学习；朴素贝叶斯算法；对数几率回归；决策树；随机森林；Weka

中图法分类号: TP301 **文献标识码**: A

1 引言

监督学习是指：从已有的训练数据集中学习出一个模型参数，从而可以使用得到的模型参数对新的数据进行预测。监督学习的数据集包括输入和输出，即特征和目标。在本次实验中，通过运行 Train.java 文件进行游戏，获得训练数据集；然后通过运行 Weka 机器学习包中的不同算法得到不同的模型参数；再运行 Test.java 文件进行预测测试。

2 监督学习算法

2.1 朴素贝叶斯算法

2.1.1 算法介绍

贝叶斯分类算法以贝叶斯定理为基础，朴素贝叶斯是其中运用最为广泛的算法之一，它基于特征属性之间相互条件独立的假设，对于给出的待分类特征求出该特征在各个类别下出现的概率并取最大概率的类别作为自己的分类结果。

2.1.2 算法过程

从概率论与统计学角度来说，我们已知贝叶斯公式为：

$$P(Y_k|X) = \frac{P(X|Y_k)P(Y_k)}{\sum_{k=1} P(X|Y=Y_k)P(Y_k)}$$

再从数据角度来分析，朴素贝叶斯分类算法是以贝叶斯定理为基础：

（1） 分类模型样本 X 有 m 个特征属性集合

（2） 特征输出分为 k 个类别，分别为 $Y_1, Y_2, \ldots, Y_k$,

（3） 假设 X 中的所有属性相互条件独立

（4） 朴素贝叶斯的先验概率：$P(Y_k) = P(Y = Y_k)$

条件概率：

$$P(Y_k|X) = \frac{P(X|Y_k)P(Y_k)}{\sum_{k=1} P(X|Y=Y_k)P(Y_k)}$$

（5） 对于每一个特征属性，比较（5）中的所有类别 Y_k 条件概率的大小，有最大概率的类别 Y_k 即为其分类结果。

2.1.3 算法源代码分析

（1） 代码中，m_Distributions 表示条件概率，m_ClassDistribution 表示先验概率：

```
protected Estimator[][] m_Distributions;
protected Estimator m_ClassDistribution;
```

（2） **buildClassifier** 函数为每一个类创建一个 Estimator 分类器：

```
for(int j = 0; j < this.m_Instances.numClasses(); ++j) {
    switch(attribute.type()) {
    case 0:
        if (this.m_UseKernelEstimator) {
            this.m_Distributions[attIndex][j] = new KernelEstimator(numPrecision);
        } else {
            this.m_Distributions[attIndex][j] = new NormalEstimator(numPrecision);
        }
        break;
    case 1:
        this.m_Distributions[attIndex][j] = new DiscreteEstimator(attribute.numValues(),  laplace: true);
        break;
    default:
        throw new Exception("Attribute type unknown to NaiveBayes");
    }
}
```

（3） 样本分类函数 **distributionForInstance** 对输入样本进行分类：扫描每一个样本，temp 存储计算得到的类条件概率，防止结果过小，将得到的结果放大

```
for(int attIndex = 0; enumAtts.hasMoreElements(); ++attIndex) {
    Attribute attribute = (Attribute)enumAtts.nextElement();
    if (!instance.isMissing(attribute)) {
        double max = 0.0D;

        int j;
        for(j = 0; j < this.m_NumClasses; ++j) {
            double temp = Math.max(1.0E-75D, Math.pow(this.m_Distributions[attIndex][j].getProbability(instance.value(attribute)), this.m_Instances.attribute
            probs[j] *= temp;
            if (probs[j] > max) {
                max = probs[j];
            }

            if (Double.isNaN(probs[j])) {
                throw new Exception("NaN returned from estimator for attribute " + attribute.name() + ":\n" + this.m_Distributions[attIndex][j].toString());
            }
        }

        if (max > 0.0D && max < 1.0E-75D) {
            for(j = 0; j < this.m_NumClasses; ++j) {
                probs[j] *= 1.0E75D;
            }
        }
    }
}

Utils.normalize(probs);
return probs;
```

2.2 逻辑回归算法

2.2.1 算法介绍

逻辑回归算法直接对分类的可能行建模，适合做分类任务，多处理二分类问题。它将数据拟合到一个 logit 函数，确立代价函数，通过优化方法求解出最优的模型参数，即一组权值，对于输入的测试集，这组权值与测试数据线性加和，并根据结果得到分类结果。

2.2.2 算法过程

（1） 构造预测函数 logit：

$$y = logit(z) = \frac{1}{1+e^{-z}}$$

（2） 构造损失函数 J，通过极大似然估计推导得：（二分类情况下）

$$J(w) = \frac{1}{m}\sum_{i=1}^{m}[-y_i log(logit(x_i)) - (1 - y_i)log(1 - logit(x_i))]$$

（3） 使用优化方法（如梯度下降法或牛顿迭代法）使损失函数 J 最小，得到模型参数。

2.2.3 算法源代码分析

（1） nC：训练集的样本个数；

xMean[i]：前 i 个特征的平均值；

Xsd[i]：前 i 个特征的标准差；

sY：分类的类别；

weights[i]：第 i 个特征的权重；

m_Data：属性值

```
int nC = train.numInstances();
this.m_Data = new double[nC][nR + 1];
int[] Y = new int[nC];
double[] xMean = new double[nR + 1];
double[] xSD = new double[nR + 1];
double[] sY = new double[nK + 1];
double[] weights = new double[nC];
double totWeights = 0.0D;
```

（2） 输入属性值、权重，计算属性均值、属性标准差、类别数，Y[i]记录每个样本的类别值

```
for(i = 0; i < nC; ++i) {
    Instance current = train.instance(i);
    Y[i] = (int)current.classValue();
    weights[i] = current.weight();
    totWeights += weights[i];
    this.m_Data[i][0] = 1.0D;
    p = 1;

    for(i = 0; i <= nR; ++i) {
        if (i != this.m_ClassIndex) {
            double x = current.value(i);
            this.m_Data[i][p] = x;
            xMean[p] += weights[i] * x;
            xSD[p] += weights[i] * x * x;
            ++p;
        }
    }

    ++sY[Y[i]];
}
```

（3） 计算 xMean、xSD：

```
xMean[0] = 0.0D;
xSD[0] = 1.0D;

for(i = 1; i <= nR; ++i) {
    xMean[i] /= totWeights;
    if (totWeights > 1.0D) {
        xSD[i] = Math.sqrt(Math.abs(xSD[i] - totWeights * xMean[i] * xMean[i]) / (totWeights - 1.0D));
    } else {
        xSD[i] = 0.0D;
    }
}
```

（4） 优化：m_MaxIts 是优化迭代的次数，优化方法为 findArgmin

```java
if (this.m_MaxIts == -1) {
    for(x = opt.findArgmin(x, b); x == null; x = opt.findArgmin(x, b)) {
        x = opt.getVarbValues();
        if (this.m_Debug) {
            System.out.println("200 iterations finished, not enough!");
        }
    }

    if (this.m_Debug) {
        System.out.println(" --------------<Converged>--------------");
    }
} else {
    opt.setMaxIteration(this.m_MaxIts);
    x = opt.findArgmin(x, b);
    if (x == null) {
        x = opt.getVarbValues();
    }
}
```

（5）　样本分类函数 **distributionForInstance** 对输入样本进行分类：

```java
public double[] distributionForInstance(Instance instance) throws Exception {
    this.m_ReplaceMissingValues.input(instance);
    instance = this.m_ReplaceMissingValues.output();
    this.m_AttFilter.input(instance);
    instance = this.m_AttFilter.output();
    this.m_NominalToBinary.input(instance);
    instance = this.m_NominalToBinary.output();
    double[] instDat = new double[this.m_NumPredictors + 1];
    int j = 1;
    instDat[0] = 1.0D;

    for(int k = 0; k <= this.m_NumPredictors; ++k) {
        if (k != this.m_ClassIndex) {
            instDat[j++] = instance.value(k);
        }
    }

    double[] distribution = this.evaluateProbability(instDat);
    return distribution;
}
```

（6）　计算概率：

```java
private double[] evaluateProbability(double[] data) {
    double[] prob = new double[this.m_NumClasses];
    double[] v = new double[this.m_NumClasses];

    int m;
    for(m = 0; m < this.m_NumClasses - 1; ++m) {
        for(int k = 0; k <= this.m_NumPredictors; ++k) {
            v[m] += this.m_Par[k][m] * data[k];
        }
    }

    v[this.m_NumClasses - 1] = 0.0D;

    for(m = 0; m < this.m_NumClasses; ++m) {
        double sum = 0.0D;

        for(int n = 0; n < this.m_NumClasses - 1; ++n) {
            sum += Math.exp(v[n] - v[m]);
        }

        prob[m] = 1.0D / (sum + Math.exp(-v[m]));
    }

    return prob;
}
```

2.3 决策树C4.5算法

2.3.1 算法介绍

决策树是一个树结构，每个非叶节点表示一个特征属性上的测试，每个分支表示这个特征属性在某个值域上的输出，每个叶节点存放一个类别。决策树的决策过程大致是：从根结点开始，测试待分类项中相应的特征属性，按照其值选择输出分支直到到达叶节点，将叶节点中存放的类别作为决策结果。

决策树中最为关键的是属性选择度量，即一种选择分裂准则，将给定的类标记和训练集合数据最好的划分为个体类。本实验介绍 C4.5 算法。

2.3.2 算法过程

（1） D 的信息熵为：

$$info(D) = -\sum_{i=1}^{m} p_i log_2(p_i)$$

其中，D 是一个用类别对训练集进行的划分结果，pi 是第 i 类在整个训练集中出现的概率

训练集 D 按属性 A 进行划分得到的信息熵为：

$$info_A(D) = -\sum_{i=1}^{m} \frac{|D_j|}{|D|} info(D_j)$$

（2） 信息增益：

$$gain(A) = info(D) - info_A(D)$$

（3） 定义“分裂信息”，训练集按属性 A 进行划分的分裂信息为：

$$split\_info_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} log_2(\frac{|D_j|}{|D|})$$

（4） 定义增益率：

$$gain\_ratio(A) = \frac{gain(A)}{split\_info(A)}$$

（5） 选择增益率最大的属性作为分裂属性

2.3.3 算法源代码分析

（1） buildClassifer 函数创建一个树结构：

首先判断是否为二叉树，再根据 m_reducedErrorPruning 选择构建树的方法，modSelection 用于选择分裂模型，m_root 是树的根结点

```java
public void buildClassifier(Instances instances) throws Exception {
    Object modSelection;
    if (this.m_binarySplits) {
        modSelection = new BinC45ModelSelection(this.m_minNumObj, instances);
    } else {
        modSelection = new C45ModelSelection(this.m_minNumObj, instances);
    }

    if (!this.m_reducedErrorPruning) {
        this.m_root = new C45PruneableClassifierTree((ModelSelection)modSelection, !this.m_unpruned, this.m_CF, this.m_subtreeRaising, !this.m_noCleanup);
    } else {
        this.m_root = new PruneableClassifierTree((ModelSelection)modSelection, !this.m_unpruned, this.m_numFolds, !this.m_noCleanup, this.m_Seed);
    }

    this.m_root.buildClassifier(instances);
    if (this.m_binarySplits) {
        ((BinC45ModelSelection)modSelection).cleanup();
    } else {
        ((C45ModelSelection)modSelection).cleanup();
    }

}
```

（2） C45PruneableClassifierTree 方法：

首先检测 data 是否能够分类，调用 buildTree 构建分类树，调用 collapse 进行树的塌缩，如果需要剪枝，则调用 prune 剪枝

```java
public void buildClassifier(Instances data) throws Exception {
    this.getCapabilities().testWithFail(data);
    data = new Instances(data);
    data.deleteWithMissingClass();
    this.buildTree(data, keepData: this.m_subtreeRaising || !this.m_cleanup);
    this.collapse();
    if (this.m_pruneTheTree) {
        this.prune();
    }

    if (this.m_cleanup) {
        this.cleanup(new Instances(data, capacity: 0));
    }

}
```

[1] buildTree 函数：

根据 m_toSelectModel 来选择一个模型把传入的数据集按相应的规则分成不同的子集，查看分裂子集的数量，若只有一个，则直接返回；否则根据 localModel 将传入的数据集分成不同的子特征，接着为每一个子特征建立新的 ClassiferTree 节点，并将其作为自己的子节点，再给子节点创建新树

```java
public void buildTree(Instances data, boolean keepData) throws Exception {
    if (keepData) {
        this.m_train = data;
    }

    this.m_test = null;
    this.m_isLeaf = false;
    this.m_isEmpty = false;
    this.m_sons = null;
    this.m_localModel = this.m_toSelectModel.selectModel(data);
    if (this.m_localModel.numSubsets() > 1) {
        Instances[] localInstances = this.m_localModel.split(data);
        data = null;
        this.m_sons = new ClassifierTree[this.m_localModel.numSubsets()];

        for(int i = 0; i < this.m_sons.length; ++i) {
            this.m_sons[i] = this.getNewTree(localInstances[i]);
            localInstances[i] = null;
        }
    } else {
        this.m_isLeaf = true;
        if (Utils.eq(data.sumOfWeights(), b: 0.0D)) {
            this.m_isEmpty = true;
        }

        data = null;
    }

}
```

[2] collapse 函数：

若子节点的出错率较高，则将这些子节点删除

```java
public final void collapse() {
    if (!this.m_isLeaf) {
        double errorsOfSubtree = this.getTrainingErrors();
        double errorsOfTree = this.localModel().distribution().numIncorrect();
        if (errorsOfSubtree >= errorsOfTree - 0.001D) {
            this.m_sons = null;
            this.m_isLeaf = true;
            this.m_localModel = new NoSplit(this.localModel().distribution());
        } else {
            for(int i = 0; i < this.m_sons.length; ++i) {
                this.son(i).collapse();
            }
        }
    }

}
```

（3） PruneableClassifierTree 方法：

与 C45PruneableClassifierTree 不同的是，该方法在构建树的时候，还传入了测试集并且去除了 collaspe 步骤

```
public void buildClassifier(Instances data) throws Exception {
    this.getCapabilities().testWithFail(data);
    data = new Instances(data);
    data.deleteWithMissingClass();
    Random random = new Random((long)this.m_seed);
    data.stratify(this.numSets);
    this.buildTree(data.trainCV(this.numSets, numFold: this.numSets - 1, random), data.testCV(this.numSets, numFold: this.numSets - 1), !this.m_cleanup);
    if (this.pruneTheTree) {
        this.prune();
    }

    if (this.m_cleanup) {
        this.cleanup(new Instances(data, capacity: 0));
    }

}
```

（4） C45 选择分裂模型：

[1] 在 public final ClassifierSplitModel selectModel(Instances data) 函数中，currentModel 存储在每个属性上构建出的分裂模型

```
C45Split[] currentModel = new C45Split[data.numAttributes()];
```

对于每一个特征，构建模型：

```
for(i = 0; i < data.numAttributes(); ++i) {
    if (i != data.classIndex()) {
        currentModel[i] = new C45Split(i, this.m_minNoObj, sumOfWeights);
        currentModel[i].buildClassifier(data);
```

对所有的属性构建完分裂模型后，选择信息增益率最大的模型作为最优模型：

```
for(i = 0; i < data.numAttributes(); ++i) {
    if (i != data.classIndex() && currentModel[i].checkModel() && currentModel[i].infoGain() >= averageInfoGain - 0.001D && Utils.gr(currentModel[i].gainR
        bestModel = currentModel[i];
        minResult = currentModel[i].gainRatio();
    }
}
```

[2] 模型构建函数 buildClassifier:

handleEnumeratedAttribute 对枚举型进行分裂，handlerNumericAttribute 对数值型进行分裂

```
public void buildClassifier(Instances trainInstances) throws Exception {
    this.m_numSubsets = 0;
    this.m_splitPoint = 1.7976931348623157E308D;
    this.m_infoGain = 0.0D;
    this.m_gainRatio = 0.0D;
    if (trainInstances.attribute(this.m_attIndex).isNominal()) {
        this.m_complexityIndex = trainInstances.attribute(this.m_attIndex).numValues();
        this.m_index = this.m_complexityIndex;
        this.handleEnumeratedAttribute(trainInstances);
    } else {
        this.m_complexityIndex = 2;
        this.m_index = 0;
        trainInstances.sort(trainInstances.attribute(this.m_attIndex));
        this.handleNumericAttribute(trainInstances);
    }

}
```

[3] 枚举型：handleEnumeratedAttribute 函数

```
private void handleEnumeratedAttribute(Instances trainInstances) throws Exception {
    this.m_distribution = new Distribution(this.m_complexityIndex, trainInstances.numClasses());
    Enumeration enu = trainInstances.enumerateInstances();

    while(enu.hasMoreElements()) {
        Instance instance = (Instance)enu.nextElement();
        if (!instance.isMissing(this.m_attIndex)) {
            this.m_distribution.add((int)instance.value(this.m_attIndex), instance);
        }
    }

    if (this.m_distribution.check((double)this.m_minNoObj)) {
        this.m_numSubsets = this.m_complexityIndex;
        this.m_infoGain = infoGainCrit.splitCritValue(this.m_distribution, this.m_sumOfWeights);
        this.m_gainRatio = gainRatioCrit.splitCritValue(this.m_distribution, this.m_sumOfWeights, this.m_infoGain);
    }

}
```

遍历所有的样本，若分裂属性不为空，则放入不同的 bag，并检查分裂是否满足要求。若满足要

求，则设置子集的数量，计算信息增益和信息增益率；否则子集数量为 0，在 buildClassifer 函数中认定无效

[4] 数值型：handlerNumericAttribute 函数

新建分布，数值型默认为二维分布，有效的样本均放在 bag1 中

```
this.m_distribution = new Distribution( numBags: 2, trainInstances.numClasses());
Enumeration enu = trainInstances.enumerateInstances();
```

```
int i;
for(i = 0; enu.hasMoreElements(); ++i) {
    Instance instance = (Instance)enu.nextElement();
    if (instance.isMissing(this.m_attIndex)) {
        break;
    }

    this.m_distribution.add( bagIndex: 1, instance);
}
```

寻找合适的分裂点：

```
for(double defaultEnt = infoGainCrit.oldEnt(this.m_distribution); next < firstMiss; ++next) {
    if (trainInstances.instance( index: next - 1).value(this.m_attIndex) + 1.0E-5D < trainInstances.instance(next).value(this.m_attIndex)) {
        this.m_distribution.shiftRange( from: 1,  to: 0, trainInstances, last, next);
        if (Utils.grOrEq(this.m_distribution.perBag( bagIndex: 0), minSplit) && Utils.grOrEq(this.m_distribution.perBag( bagIndex: 1), minSplit)) {
            double currentInfoGain = infoGainCrit.splitCritValue(this.m_distribution, this.m_sumOfWeights, defaultEnt);
            if (Utils.gr(currentInfoGain, this.m_infoGain)) {
                this.m_infoGain = currentInfoGain;
                splitIndex = next - 1;
            }

            ++this.m_index;
        }

        last = next;
    }
}
```

计算最大信息增益和信息增益率：

```
if (this.m_index != 0) {
    this.m_infoGain -= Utils.log2((double)this.m_index) / this.m_sumOfWeights;
    if (!Utils.smOrEq(this.m_infoGain,  b: 0.0D)) {
        this.m_numSubsets = 2;
        this.m_splitPoint = (trainInstances.instance( index: splitIndex + 1).value(this.m_attIndex) + trainInstances.instance(splitIndex).value(this.m_att
        if (this.m_splitPoint == trainInstances.instance( index: splitIndex + 1).value(this.m_attIndex)) {
            this.m_splitPoint = trainInstances.instance(splitIndex).value(this.m_attIndex);
        }

        this.m_distribution = new Distribution( numBags: 2, trainInstances.numClasses());
        this.m_distribution.addRange( bagIndex: 0, trainInstances,  startIndex: 0,  lastPlusOne: splitIndex + 1);
        this.m_distribution.addRange( bagIndex: 1, trainInstances,  startIndex: splitIndex + 1, firstMiss);
        this.m_gainRatio = gainRatioCrit.splitCritValue(this.m_distribution, this.m_sumOfWeights, this.m_infoGain);
    }
}
```

2.4 随机森林算法

2.4.1 算法介绍

随机森林是一种简单且有效的算法，基于 Bagging 的集成学习方法，常用来做分类、回归问题。核心思想为通过训练和组合不同的决策树，形成森林，最终的分类结果是个别树输出类别的众数。准确率高，训练速度快，抗噪能力好。

2.4.2 算法过程

（1） 对训练集进行有放回的抽样 N 次，得到的子集作为新的训练集

（2） 在新的训练集中随机抽出训练集的 K 个属性，训练一个决策树模型，不做剪枝操作

（3） 重复上述过程 M 次，得到 M 个决策树模型，即 M 个分类器

（4） 对于测试用例，使用 M 个分类器进行分类，最终的分类结果由这 M 个分类器投票决定。

2.4.3 算法源代码分析

（1） buildClassifier 函数创建分类器：

首先去除无效数据，构建一个随机树，设置属性值，设置最大深度，并将该随机树传给 Bag，调用 bagging 训练方法进行训练

```java
public void buildClassifier(Instances data) throws Exception {
    this.getCapabilities().testWithFail(data);
    data = new Instances(data);
    data.deleteWithMissingClass();
    this.m_bagger = new Bagging();
    RandomTree rTree = new RandomTree();
    this.m_KValue = this.m_numFeatures;
    if (this.m_KValue < 1) {
        this.m_KValue = (int)Utils.log2((double)(data.numAttributes() - 1)) + 1;
    }

    rTree.setKValue(this.m_KValue);
    rTree.setMaxDepth(this.getMaxDepth());
    this.m_bagger.setClassifier(rTree);
    this.m_bagger.setSeed(this.m_randomSeed);
    this.m_bagger.setNumIterations(this.m_numTrees);
    this.m_bagger.setCalcOutOfBag(true);
    this.m_bagger.buildClassifier(data);
}
```

（2） 分类过程使用 bagging 的 distributionInstance：
计算出概率的最大值并作为返回结果

```java
public double[] distributionForInstance(Instance instance) throws Exception {
    double[] sums = new double[instance.numClasses()];

    for(int i = 0; i < this.m_NumIterations; ++i) {
        if (instance.classAttribute().isNumeric()) {
            sums[0] += this.m_Classifiers[i].classifyInstance(instance);
        } else {
            double[] newProbs = this.m_Classifiers[i].distributionForInstance(instance);

            for(int j = 0; j < newProbs.length; ++j) {
                sums[j] += newProbs[j];
            }
        }
    }

    if (instance.classAttribute().isNumeric()) {
        sums[0] /= (double)this.m_NumIterations;
        return sums;
    } else if (Utils.eq(Utils.sum(sums),  b: 0.0D)) {
        return sums;
    } else {
        Utils.normalize(sums);
        return sums;
    }
}
```

3 修改特征提取方法

目前所用的特征提取方法比较简单，只记录了某一时刻游戏画面的特征和按下的按键。但是在实际的运行过程中，某一时刻游戏画面的特征效果并不好，从游戏获胜的角度来讲，应该最先追击离精灵最近的怪兽，否则一旦第一个怪兽碰到精灵，游戏结束。所以应该记录下精灵当前的位置和第一个怪兽的位置。

```java
feature[452] = obs.getAvatarPosition().x/25;
feature[453] = obs.getAvatarPosition().y/25;
```

```java
feature[455] = allobj.get(0).position.x/25;
feature[456] = allobj.get(0).position.y/25;
```

除此之外，由于怪兽会发射子弹，精灵需要躲避子弹，所以也要记录和精灵横坐标一样子弹的坐标；与此同时，还需要记录按下的按键和当前画面特征。

```
for(Observation o: allobj){
    if(o.itype == Types.TYPE_FROMAVATAR) {
        feature[454] = o.position.x/25;
        break;
    }
}
```

4 实验结果

A. 现有特征提取结果

分别在 lv0-lv4 关卡游戏胜利，将结果存储到 AliensRecorder.arff 文件中，在 weka 中分别调用上述 4 个算法查看结果，每种方法的每一个关卡运行 5 次求平均结果

4.1 朴素贝叶斯算法

Lv0:

```
Time taken to build model: 0.06 seconds

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

Correctly Classified Instances         371               53.6903 %
Incorrectly Classified Instances       320               46.3097 %
Kappa statistic                          0.2798
Mean absolute error                      0.2348
Root mean squared error                  0.4563
Relative absolute error                 89.8628 %
Root relative squared error            126.3939 %
Total Number of Instances              691

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.506     0.208      0.787     0.506     0.616      0.695    0
                 0.53      0.246      0.521     0.53      0.526      0.687    1
                 1         0.109      0.223     1         0.365      0.979    2
                 0.762     0.115      0.172     0.762     0.281      0.944    3
Weighted Avg.    0.537     0.215      0.662     0.537     0.568      0.708

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 211 112  48  46 |   a = 0
  56 123  22  31 |   b = 1
   0   0  21   0 |   c = 2
   1   1   3  16 |   d = 3
```

	Result	Score	TimeStep
1	Lose	40	853
2	Win	42	593
3	Lose	43	853
4	Win	42	521
5	Win	42	533
Average	60%	42	671

Lv1:

```
Time taken to build model: 0.03 seconds

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

Correctly Classified Instances         347               48.736  %
Incorrectly Classified Instances       365               51.264  %
Kappa statistic                          0.2196
Mean absolute error                      0.2586
Root mean squared error                  0.4769
Relative absolute error                123.8641 %
Root relative squared error            147.9525 %
Total Number of Instances              712

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.439     0.188      0.868     0.439     0.583      0.688    0
                 0.481     0.244      0.316     0.481     0.381      0.68     1
                 1         0.128      0.2       1         0.333      0.969    2
                 1         0.148      0.223     1         0.365      0.949    3
Weighted Avg.    0.487     0.195      0.717     0.487     0.528      0.706

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 231 141  73  81 |   a = 0
  35  65  15  20 |   b = 1
   0   0  22   0 |   c = 2
   0   0   0  29 |   d = 3
```

	Result	Score	TimeStep
1	Win	42	301
2	Lose	44	893
3	Win	42	301
4	Win	46	539
5	Lose	23	214
Average	60%	42	671

Lv2:

```
Time taken to build model: 0.03 seconds

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

Correctly Classified Instances         334               48.4058 %
Incorrectly Classified Instances       356               51.5942 %
Kappa statistic                          0.2054
Mean absolute error                      0.2633
Root mean squared error                  0.4836
Relative absolute error                109.5129 %
Root relative squared error            139.7076 %
Total Number of Instances              690

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.433     0.213      0.802     0.433     0.562      0.627    0
                 0.606     0.303      0.429     0.606     0.502      0.695    1
                 0.25      0.032      0.25      0.25      0.25       0.865    2
                 1         0.198      0.095     1         0.173      0.968    3
Weighted Avg.    0.484     0.23       0.664     0.484     0.525      0.662

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 199 145  16 100 |   a = 0
  44 114   5  25 |   b = 1
   5   7   7   9 |   c = 2
   0   0   0  14 |   d = 3
```

	Result	Score	TimeStep
1	Win	44	511
2	Lose	40	457
3	Lose	18	244
4	Lose	42	937
5	Win	44	511
Average	40%	38	532

Lv3:

```
Time taken to build model: 0.02 seconds

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

Correctly Classified Instances         277               63.8249 %
Incorrectly Classified Instances       157               36.1751 %
Kappa statistic                          0.4146
Mean absolute error                      0.1983
Root mean squared error                  0.3881
Relative absolute error                 68.8449 %
Root relative squared error            102.4006 %
Total Number of Instances              434

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.655     0.292      0.72      0.655     0.686      0.727    0
                 0.532     0.156      0.661     0.532     0.589      0.759    1
                 0.917     0.113      0.423     0.917     0.579      0.979    2
                 1         0.023      0.444     1         0.615      0.994    3
Weighted Avg.    0.638     0.223      0.669     0.638     0.641      0.764

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 152  43  32   5 |   a = 0
  56  84  13   5 |   b = 1
   3   0  33   0 |   c = 2
   0   0   0   8 |   d = 3
```

	Result	Score	TimeStep
1	Win	43	646
2	Lose	13	232
3	Win	43	646
4	Win	43	519
5	Win	43	646
Average	80%	38	538

Lv4:

```
Time taken to build model: 0.06 seconds

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

Correctly Classified Instances         395               57.2464 %
Incorrectly Classified Instances       295               42.7536 %
Kappa statistic                          0.2681
Mean absolute error                      0.212
Root mean squared error                  0.4404
Relative absolute error                 86.9448 %
Root relative squared error            126.3029 %
Total Number of Instances              690

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.505     0.22       0.801     0.505     0.619      0.719    0
                 0.676     0.389      0.456     0.676     0.545      0.708    1
                 1         0.026      0.28      1         0.438      0.995    2
                 0.778     0.061      0.255     0.778     0.384      0.963    3
Weighted Avg.    0.572     0.269      0.669     0.572     0.587      0.724

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 222 177  12  29 |   a = 0
  55 152   6  12 |   b = 1
   0   0   7   0 |   c = 2
   0   4   0  14 |   d = 3
```

	Result	Score	TimeStep
1	Lose	8	163
2	Lose	42	777
3	Win	49	695
4	Lose	29	295
5	Win	44	493
Average	40%	34	485

4.2 Logistic算法

Lv0:

```
Time taken to build model: 2.41 seconds

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

Correctly Classified Instances         574               83.068  %
Incorrectly Classified Instances       117               16.932  %
Kappa statistic                          0.6763
Mean absolute error                      0.1125
Root mean squared error                  0.2371
Relative absolute error                 43.0474 %
Root relative squared error             65.6868 %
Total Number of Instances              691

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.866     0.212      0.862     0.866     0.864      0.921    0
                 0.75      0.109      0.777     0.75      0.763      0.926    1
                 0.952     0.006      0.833     0.952     0.889      0.998    2
                 0.905     0.007      0.792     0.905     0.844      0.999    3
Weighted Avg.    0.831     0.165      0.83      0.831     0.83       0.928

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 361  48   4   4 |   a = 0
  57 174   0   1 |   b = 1
   0   1  20   0 |   c = 2
   1   1   0  19 |   d = 3
```

	Result	Score	TimeStep
1	Win	62	872
2	Win	56	713
3	Lose	57	812
4	Lose	56	926
5	Lose	53	671
Average	40%	57	799

Lv1:

```
Time taken to build model: 3.75 seconds

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

Correctly Classified Instances         633               88.9045 %
Incorrectly Classified Instances        79               11.0955 %
Kappa statistic                          0.7335
Mean absolute error                      0.0688
Root mean squared error                  0.1856
Relative absolute error                 32.9551 %
Root relative squared error             57.576  %
Total Number of Instances              712

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.926     0.204      0.928     0.926     0.927      0.965    0
                 0.807     0.047      0.801     0.807     0.804      0.98     1
                 0.636     0.006      0.778     0.636     0.7        0.993    2
                 0.793     0.015      0.697     0.793     0.742      0.992    3
Weighted Avg.    0.889     0.161      0.89      0.889     0.889      0.969

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 487  27   4   8 |   a = 0
  24 109   0   2 |   b = 1
   8   0  14   0 |   c = 2
   6   0   0  23 |   d = 3
```

	Result	Score	TimeStep
1	Win	52	798
2	Lose	30	535
3	Lose	10	163
4	Lose	20	226
5	Lose	43	989
Average	20%	31	542

Lv2:

```
Time taken to build model: 2.89 seconds

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

Correctly Classified Instances         559               81.0145 %
Incorrectly Classified Instances       131               18.9855 %
Kappa statistic                          0.5914
Mean absolute error                      0.1252
Root mean squared error                  0.25
Relative absolute error                 52.064  %
Root relative squared error             72.2286 %
Total Number of Instances              690

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.885     0.339      0.839     0.885     0.861      0.89     0
                 0.644     0.086      0.738     0.644     0.688      0.907    1
                 0.714     0.011      0.741     0.714     0.727      0.994    2
                 0.786     0.004      0.786     0.786     0.786      0.997    3
Weighted Avg.    0.81      0.25       0.806     0.81      0.807      0.901

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 407  43   7   3 |   a = 0
  67 121   0   0 |   b = 1
   8   0  20   0 |   c = 2
   3   0   0  11 |   d = 3
```

	Result	Score	TimeStep
1	Lose	49	523
2	Win	47	634
3	Win	48	592
4	Win	49	523
5	Win	46	543
Average	80%	48	563

Lv3:

```
Time taken to build model: 0.79 seconds

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

Correctly Classified Instances         372               85.7143 %
Incorrectly Classified Instances        62               14.2857 %
Kappa statistic                          0.751
Mean absolute error                      0.095
Root mean squared error                  0.217
Relative absolute error                 32.9846 %
Root relative squared error             57.251  %
Total Number of Instances              434

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.871     0.158     0.863     0.871     0.867      0.946    0
                 0.804     0.105     0.814     0.804     0.809      0.945    1
                 0.972     0.003     0.972     0.972     0.972      1        2
                 1         0         1         1         1          1        3
Weighted Avg.    0.857     0.123     0.857     0.857     0.857      0.951

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 202  29   1   0 |   a = 0
  31 127   0   0 |   b = 1
   1   0  35   0 |   c = 2
   0   0   0   8 |   d = 3
```

	Result	Score	TimeStep
1	Lose	13	238
2	Lose	36	889
3	Lose	40	761
4	Lose	24	352
5	Lose	9	163
Average	0	24	481

Lv4:

```
Time taken to build model: 1.94 seconds

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

Correctly Classified Instances         556               80.5797 %
Incorrectly Classified Instances       134               19.4203 %
Kappa statistic                          0.593
Mean absolute error                      0.1205
Root mean squared error                  0.2464
Relative absolute error                 49.4138 %
Root relative squared error             70.6812 %
Total Number of Instances              690

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.873     0.308     0.833     0.873     0.852      0.904    0
                 0.653     0.12      0.724     0.653     0.687      0.898    1
                 1         0         1         1         1          1        2
                 1         0.001     0.947     1         0.973      1        3
Weighted Avg.    0.806     0.236     0.802     0.806     0.803      0.906

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 384  56   0   0 |   a = 0
  77 147   0   1 |   b = 1
   0   0   7   0 |   c = 2
   0   0   0  18 |   d = 3
```

	Result	Score	TimeStep
1	Lose	31	277
2	Lose	29	352
3	Lose	41	686
4	Win	56	917
5	Lose	37	457
Average	20%	38.8	598

4.3 决策树C4.5算法

Lv0:

```
Time taken to build model: 0.59 seconds

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

Correctly Classified Instances         596               83.7079 %
Incorrectly Classified Instances       116               16.2921 %
Kappa statistic                          0.514
Mean absolute error                      0.1334
Root mean squared error                  0.2583
Relative absolute error                 63.9113 %
Root relative squared error             80.1229 %
Total Number of Instances              712

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.979     0.565     0.831     0.979     0.899      0.781    0
                 0.496     0.019     0.859     0.496     0.629      0.824    1
                 0.273     0         1         0.273     0.429      0.945    2
                 0.276     0         1         0.276     0.432      0.918    3
Weighted Avg.    0.837     0.421     0.848     0.837     0.814      0.8

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 515  11   0   0 |   a = 0
  68  67   0   0 |   b = 1
  16   0   6   0 |   c = 2
  21   0   0   8 |   d = 3
```

	Result	Score	TimeStep
1	Win	42	444
2	Win	52	573
3	Win	42	444
4	Win	42	444
5	Win	55	697
Average	100%	47	520

Lv1:

```
Time taken to build model: 0.39 seconds

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

Correctly Classified Instances         558               80.8696 %
Incorrectly Classified Instances       132               19.1304 %
Kappa statistic                          0.5411
Mean absolute error                      0.1441
Root mean squared error                  0.2685
Relative absolute error                 59.9534 %
Root relative squared error             77.5548 %
Total Number of Instances              690

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.963     0.496      0.795     0.963     0.871      0.838    0
                 0.516     0.028      0.874     0.516     0.649      0.869    1
                 0.464     0.006      0.765     0.464     0.578      0.942    2
                 0.357     0          1         0.357     0.526      0.909    3
Weighted Avg.    0.809     0.338      0.82      0.809     0.792      0.852

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 443  13   4   0 |   a = 0
  91  97   0   0 |   b = 1
  14   1  13   0 |   c = 2
   9   0   0   5 |   d = 3
```

	Result	Score	TimeStep
1	Lose	39	977
2	Win	42	301
3	Win	42	301
4	Win	42	401
5	Lose	17	690
Average	60%	36	534

Lv2:

```
Time taken to build model: 0.16 seconds

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

Correctly Classified Instances         386               88.9401 %
Incorrectly Classified Instances        48               11.0599 %
Kappa statistic                          0.8046
Mean absolute error                      0.0864
Root mean squared error                  0.2079
Relative absolute error                 30.0114 %
Root relative squared error             54.8584 %
Total Number of Instances              434

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.935     0.163      0.868     0.935     0.9        0.947    0
                 0.804     0.054      0.894     0.804     0.847      0.944    1
                 0.944     0          1         0.944     0.971      0.997    2
                 1         0          1         1         1          1        3
Weighted Avg.    0.889     0.107      0.891     0.889     0.889      0.951

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 217  15   0   0 |   a = 0
  31 127   0   0 |   b = 1
   2   0  34   0 |   c = 2
   0   0   0   8 |   d = 3
```

	Result	Score	TimeStep
1	Lose	15	346
2	Win	45	567
3	Win	44	583
4	Lose	40	841
5	Lose	34	729
Average	40%	34	613

Lv3:

```
Time taken to build model: 0.38 seconds

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

Correctly Classified Instances         597               86.5217 %
Incorrectly Classified Instances        93               13.4783 %
Kappa statistic                          0.7152
Mean absolute error                      0.1014
Root mean squared error                  0.2252
Relative absolute error                 41.5867 %
Root relative squared error             64.5879 %
Total Number of Instances              690

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.925     0.228      0.877     0.925     0.9        0.929    0
                 0.764     0.073      0.835     0.764     0.798      0.932    1
                 0.857     0          1         0.857     0.923      0.997    2
                 0.667     0.003      0.857     0.667     0.75       0.989    3
Weighted Avg.    0.865     0.169      0.864     0.865     0.863      0.933

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 407  32   0   1 |   a = 0
  52 172   0   1 |   b = 1
   0   1   6   0 |   c = 2
   5   1   0  12 |   d = 3
```

	Result	Score	TimeStep
1	Win	44	524
2	Win	44	524
3	Win	44	524
4	Lose	25	283
5	Lose	27	325
Average	60%	37	436

Lv4:

```
Time taken to build model: 0.38 seconds

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

Correctly Classified Instances         597               86.5217 %
Incorrectly Classified Instances        93               13.4783 %
Kappa statistic                          0.7152
Mean absolute error                      0.1014
Root mean squared error                  0.2252
Relative absolute error                 41.5867 %
Root relative squared error             64.5879 %
Total Number of Instances              690

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.925     0.228      0.877     0.925     0.9        0.929    0
                 0.764     0.073      0.835     0.764     0.798      0.932    1
                 0.857     0          1         0.857     0.923      0.997    2
                 0.667     0.003      0.857     0.667     0.75       0.989    3
Weighted Avg.    0.865     0.169      0.864     0.865     0.863      0.933

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 407  32   0   1 |   a = 0
  52 172   0   1 |   b = 1
   0   1   6   0 |   c = 2
   5   1   0  12 |   d = 3
```

	Result	Score	TimeStep
1	Win	48	567
2	Lose	45	997
3	Lose	43	901
4	Lose	15	193
5	Lose	45	853
Average	20%	40	702

4.4 随机森林算法

Lv0:

```
Time taken to build model: 2.08 seconds

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

Correctly Classified Instances         691              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.074
Root mean squared error                  0.1207
Relative absolute error                 28.3141 %
Root relative squared error             33.4261 %
Total Number of Instances              691

=== 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
 417   0   0   0 |   a = 0
   0 232   0   0 |   b = 1
   0   0  21   0 |   c = 2
   0   0   0  21 |   d = 3
```

	Result	Score	TimeStep
1	Win	49	687
2	Lose	42	496
3	Lose	32	346
4	Win	51	514
5	Win	51	755
Average	60%	45	560

Lv1:

```
Time taken to build model: 2.37 seconds

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

Correctly Classified Instances         712              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.0438
Root mean squared error                  0.0997
Relative absolute error                 20.9666 %
Root relative squared error             30.9301 %
Total Number of Instances              712

=== 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
 526   0   0   0 |   a = 0
   0 135   0   0 |   b = 1
   0   0  22   0 |   c = 2
   0   0   0  29 |   d = 3
```

	Result	Score	TimeStep
1	Win	42	301
2	Win	42	301
3	Win	42	301
4	Win	42	301
5	Lose	19	175
Average	80%	37	276

Lv2:

```
Time taken to build model: 2.21 seconds

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

Correctly Classified Instances         690              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.0585
Root mean squared error                  0.1112
Relative absolute error                 24.3454 %
Root relative squared error             32.1171 %
Total Number of Instances              690

=== 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
 460   0   0   0 |   a = 0
   0 188   0   0 |   b = 1
   0   0  28   0 |   c = 2
   0   0   0  14 |   d = 3
```

	Result	Score	TimeStep
1	Win	47	589
2	Lose	42	719
3	Win	44	464
4	Win	45	593
5	Win	47	528
Average	80%	45	579

Lv3:

```
Time taken to build model: 1.07 seconds

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

Correctly Classified Instances         434              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.0727
Root mean squared error                  0.1149
Relative absolute error                 25.2457 %
Root relative squared error             30.3203 %
Total Number of Instances              434

=== 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
 232   0   0   0 |   a = 0
   0 158   0   0 |   b = 1
   0   0  36   0 |   c = 2
   0   0   0   8 |   d = 3
```

	Result	Score	TimeStep
1	Lose	16	247
2	Win	45	787
3	Win	46	500
4	Win	45	787
5	Lose	16	250
Average	60%	43	514

Lv4:

```
Time taken to build model: 1.73 seconds

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

Correctly Classified Instances         690              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.0689
Root mean squared error                  0.1157
Relative absolute error                 28.2358 %
Root relative squared error             33.1846 %
Total Number of Instances              690

=== 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
 440   0   0   0 |   a = 0
   0 225   0   0 |   b = 1
   0   0   7   0 |   c = 2
   0   0   0  18 |   d = 3
```

	Result	Score	TimeStep
1	Win	46	735
2	Win	46	735
3	Win	46	735
4	Win	46	735
5	Win	46	735
Average	100%	46	735

可以看出，朴素贝叶斯算法耗时短，但互相独立的条件要求高，所以准确率受到限制，大概只有 50%左右；logistic 算法耗时长，训练效果在数据集上显示较好，有 80%左右的准确率，但是在具体的测试上，胜率并不高，可能是受到训练集和学习方法的限制；决策树和随机森林都有着较好的准确率，特别是随机森林，在分

类的时候可以实现无错分类，准确率达到 100%，算法运行时间略长。

B. 修改特征提取方法结果

由于原理一样，所以在此不再过多的测试，只用上述四种算法对 lv0 进行训练与测试，结果如下：

（1） 朴素贝叶斯

```
Time taken to build model: 0.03 seconds

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

Correctly Classified Instances         414               66.0287 %
Incorrectly Classified Instances       213               33.9713 %
Kappa statistic                          0.4411
Mean absolute error                      0.1743
Root mean squared error                  0.3947
Relative absolute error                 69.6674 %
Root relative squared error            111.7943 %
Total Number of Instances              627

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.638     0.162      0.891     0.638     0.744      0.803    0
                 0.636     0.16       0.487     0.636     0.552      0.823    1
                 0.833     0.088      0.366     0.833     0.508      0.936    2
                 0.787     0.081      0.44      0.787     0.565      0.916    3
Weighted Avg.    0.66      0.151      0.749     0.66      0.68       0.823

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 270  71  42  40 |   a = 0
  27  77  10   7 |   b = 1
   0   6  30   0 |   c = 2
   6   4   0  37 |   d = 3
```

（2） Logistics 算法

```
Time taken to build model: 27.79 seconds

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

Correctly Classified Instances         587               93.6204 %
Incorrectly Classified Instances        40                6.3796 %
Kappa statistic                          0.8696
Mean absolute error                      0.0472
Root mean squared error                  0.1523
Relative absolute error                 18.8877 %
Root relative squared error             43.143  %
Total Number of Instances              627

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.967     0.127      0.94      0.967     0.953      0.986    0
                 0.826     0.02       0.909     0.826     0.866      0.99     1
                 1         0          1         1         1          1        2
                 0.894     0.007      0.913     0.894     0.903      0.996    3
Weighted Avg.    0.936     0.09       0.936     0.936     0.935      0.988

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 409  10   0   4 |   a = 0
  21 100   0   0 |   b = 1
   0   0  36   0 |   c = 2
   5   0   0  42 |   d = 3
```

（3） 决策树 C45 算法

```
Time taken to build model: 0.21 seconds

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

Correctly Classified Instances         545               86.9219 %
Incorrectly Classified Instances        82               13.0781 %
Kappa statistic                          0.7123
Mean absolute error                      0.103 
Root mean squared error                  0.2269
Relative absolute error                 41.1734 %
Root relative squared error             64.2683 %
Total Number of Instances              627     

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.967     0.314      0.865     0.967     0.913      0.911    0
                 0.719     0.028      0.861     0.719     0.784      0.921    1
                 0.75      0.007      0.871     0.75      0.806      0.98     2
                 0.468     0          1         0.468     0.638      0.957    3
Weighted Avg.    0.869     0.217      0.875     0.869     0.861      0.921

=== Confusion Matrix ===

   a   b   c   d   <-- classified as
 409  11   3   0 |   a = 0
  33  87   1   0 |   b = 1
   6   3  27   0 |   c = 2
  25   0   0  22 |   d = 3
```

（4） 随机森林

```
Time taken to build model: 1.14 seconds

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

Correctly Classified Instances         627              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1     
Mean absolute error                      0.0566
Root mean squared error                  0.1049
Relative absolute error                 22.6438 %
Root relative squared error             29.7146 %
Total Number of Instances              627     

=== 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
 423   0   0   0 |   a = 0
   0 121   0   0 |   b = 1
   0   0  36   0 |   c = 2
   0   0   0  47 |   d = 3
```

可以看出，经过修改特征，准确率有所提升，但 Logistics 算法耗时过长，推测原因应该是类别过多，学习方法选取的问题。

致谢 在此,感谢人工智能授课老师和辛勤工作的助教表示感谢.

References:

[1] https://blog.csdn.net/roger__wong/article/details/39119701 Weka 算法 Classifier-tree-J48 源码分析（二）ClassifierTree

[2] https://blog.csdn.net/liangliang8086/article/details/8654847 **weka 分类器-C4.5 决策树**

[3] https://blog.csdn.net/qq547276542/article/details/78304454 随机森林算法学习.

[4] https://blog.csdn.net/liangliang8086/article/details/8651480 weka 分类器-NaiveBayes.

[5] https://blog.csdn.net/roger__wong/article/details/39119701 NaiveBayes 朴素贝叶斯分类器 weka 实现.

[6] https://blog.csdn.net/qiao1245/article/details/50897518 Andrew Ng 机器学习笔记+Weka 相关算法实现（二）生成学习/朴素贝叶斯.

[7] https://www.cnblogs.com/pinard/p/6069267.html 朴素贝叶斯算法原理小结.

[8] https://blog.csdn.net/han_xiaoyang/article/details/49123419 机器学习系列(1)_逻辑回归初步.