**CS 634 - Data Mining**

**Final Term Project**

**Option 1- Supervised Data Mining (Classification)**

Decision Trees (Category 3)

Naïve Bayes (Category 5)

Name: Karthik Ganeshula

UCID: kg496

Email: [kg496@njit.edu](mailto:kg496@njit.edu)

**Contents**

**Introduction……………………………………………………………………3**

**Dataset used and Source……………………………………………………….6**

**Weka……………………………………………………………………………7**

**Running Naïve Bayes algorithm on the dataset…………………………….14**

**Running J48 algorithm on the dataset………………………………………18**

**Conclusion…………………………………………………………………….23**

**Sources for algorithms………………………………………………………..24**

**Source code of Naïve Bayes…………………………………………………..24**

**Source code of J48…………………………………………………………….45**

1. **Introduction**

**J48(C4.5) Algorithm**: This algorithm is a continuation for ID3 algorithm, it is used to generate the decision trees and developed by Ross Quinlan. It is mainly used for classification and a popular algorithm in the field of data mining.

The decision tree is described as follows:

→ Leaf nodes represent the labels or class values (Values that are needed to be found).

→ Branches in the tree are the result of the test done on an attribute.

→ Internal nodes represent a test on attribute.

The for J48 decision tree algorithm works as follows:

→ Check whether all instances belong to the same class. If true, tree is a leaf node labelled with that class.

→ For each attribute, Entropy, value I and information gain is calculated.

→ Find the best splitting attribute (which has highest information gain).

→ Repeat the above steps recursively and construct the tree.

The formula for the calculation of I is given below:

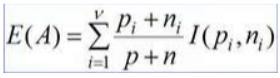


P and n denote two classes:

→ p = Number of instances in P class

→ n = Number of instances in n class

The formula for Entropy is given below:



A will give a result that can be partiontied to subsets ranging in the format of {S1,S2,….Sv}, pi is the total number of p instances in si and ni is the number of ‘n’ instances in si and the entropy is calculated as above Information Gain can be calculated using the entropy which will help to decide which attribute will become the part of tree branch



The gain for every attribute is calculated and we pick the attribute that has the highest information gain and branch that part to the tree at that attribute. Same step is repeated until the final tree is built.

When the tree is constructed the accuracy of the tree can be calculated by computing what percentage of the class the tree has accurately classified. If the accuracy is below 65% then it is assumed that the tree is not properly constructed and few changes are needed to be done mainly on the dataset.

Software like WEKA can be used to determine it.

**Naïve Bayes:**

Naïve Bayes is a [classification technique](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2/?utm_source=blog&utm_medium=6stepsnaivebayesarticle) based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

Text, letter

Description automatically generated

*→ P*(*c|x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).

*→ P*(*c*) is the prior probability of *class*.

*→ P*(*x|c*) is the likelihood which is the probability of *predictor* given *class*.

→ *P*(*x*) is the prior probability of *predictor*.

**Dataset used and Source:**

Source: https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews

**Dataset description:** This is a Women’s Clothing E-Commerce dataset revolving around the reviews written by customers. Its nine supportive features offer a great environment to parse out the text through its multiple dimensions. Because this is real commercial data, it has been anonymized, and references to the company in the review text and body have been replaced with “retailer”.

**Contents of the dataset:**

This dataset includes 23486 rows and 10 feature variables. Each row corresponds to a customer review, and includes the variables:

1. Id: This contains the index of the items in the dataset.
2. Clothing ID: Integer Categorical variable that refers to the specific piece being reviewed.
3. Age: Positive Integer variable of the reviewers age.
4. Rating: Positive Ordinal Integer variable for the product score granted by the customer from 1 Worst, to 5 Best.
5. Recommended IND: Binary variable stating where the customer recommends the product where 1 is recommended, 0 is not recommended.
6. Positive Feedback Count: Positive Integer documenting the number of other customers who found this review positive.
7. Division Name: Categorical name of the product high level division.
8. **Department Name:** Categorical name of the product department name.
9. Class Name: Categorical name of the product class name.

The above dataset has anonymous but real data. The data has been published in the following site previously: <https://www.researchgate.net/publication/323545316_Statistical_Analysis_on_E-Commerce_Reviews_with_Sentiment_Classification_using_Bidirectional_Recurrent_Neural_Network>

**Weka (Waikato Environment for Knowledge Analysis** (**Weka**):

Weka was developed at the University of Waikato, New Zeland and is a free software licensed under GNU General Public License, and the companion software to the book “Data Mining: Practical Machine learning Tools and Techniques”. Weka contains a collection of visualization tools and algorithms for [data analysis](https://en.wikipedia.org/wiki/Data_analysis) and [predictive modeling](https://en.wikipedia.org/wiki/Predictive_modeling), together with graphical user interfaces for easy access to these functions.[[1]](https://en.wikipedia.org/wiki/Weka_(machine_learning)#cite_note-:0-1) The original non-Java version of Weka was a [Tcl](https://en.wikipedia.org/wiki/Tcl)/Tk front-end to (mostly third-party) modeling algorithms implemented in other programming languages, plus [data preprocessing](https://en.wikipedia.org/wiki/Data_preprocessing) utilities in [C](https://en.wikipedia.org/wiki/C_(programming_language)), and a [Makefile](https://en.wikipedia.org/wiki/Makefile)-based system for running machine learning experiments. This original version was primarily designed as a tool for analyzing data from agricultural domains, but the more recent fully [Java](https://en.wikipedia.org/wiki/Java_(programming_language))-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research. Advantages of Weka include:

→ Free availability under the GNU General Public License.

→ Portability, since it is fully implemented in Java and thus runs on almost any modern computing platform.

→ A comprehensive collection of data pre-processing and modeling techniques.

→ Ease of use due to its graphical user interfaces.

Weka is/can be operated in 4 modes:

→ Simple CLI – Allowing users to access the WEKA through OS shell. It also allows incremental processing of the data

→Experimenter – Weka has GUI that allows large scale of predictive performance of learning algorithms

→ Explorer – It has a very popular interface for batch data processing, tab based interface for the algorithms

→ Work Bench – Includes methods for the main data mining problems, clustering, regression classification, rule mining and attribute selection.

**Weka download and Installation:**

Weka works on various operating systems like Windows, Linux, Mac etc.,

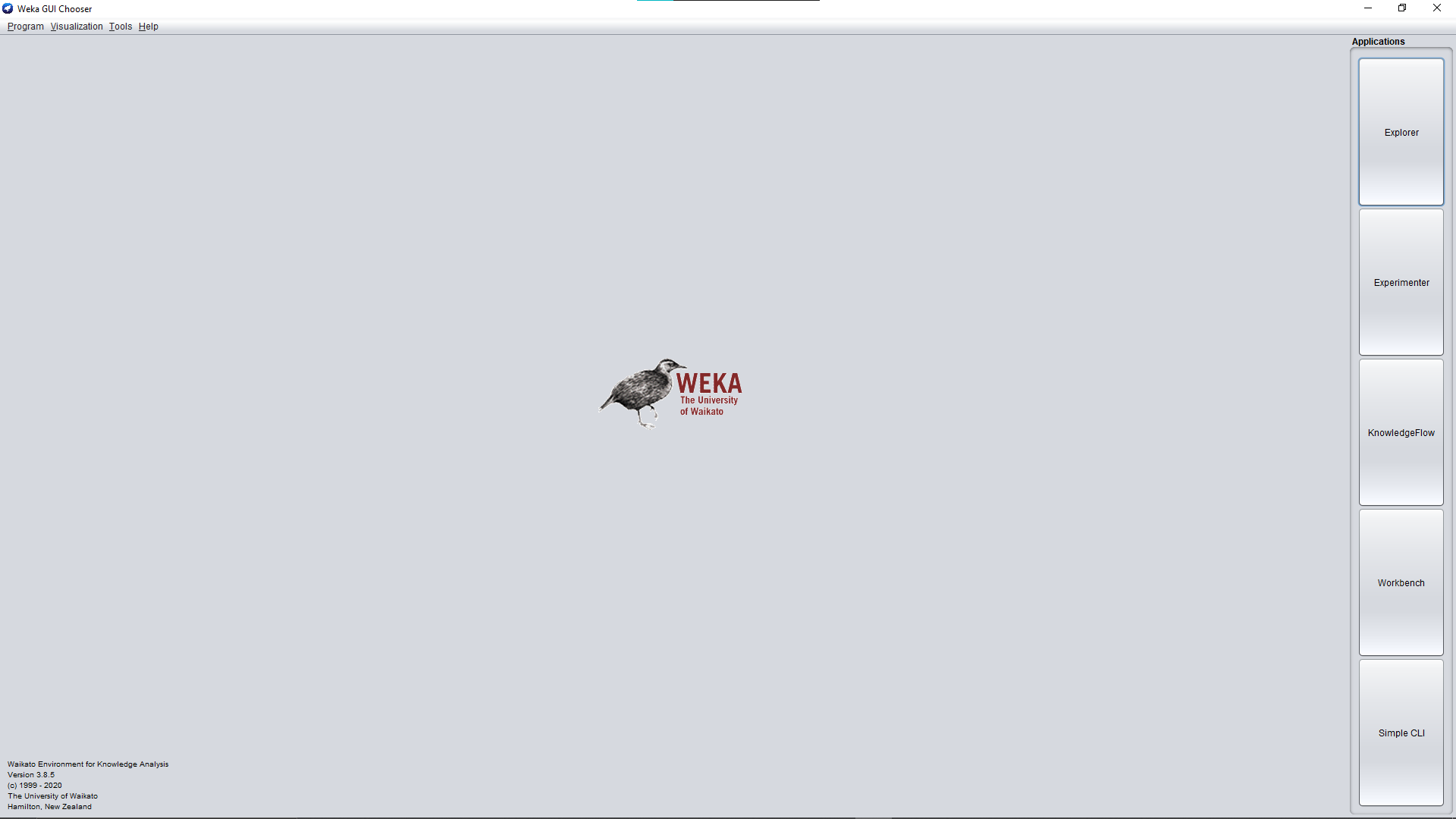
Weka can be downloaded from the following link: <http://www.cs.waikato.ac.nz/ml/weka/downloading.html>

Steps for installation on Windows:

1. Search for the stable version which is 3.8 currently and download the weka.exe file.
2. Once the file is downloaded, click the on the file and click “install” in the UI.
3. Once the installation is done, it is ready to use.

**Importing the data into weka:**

Once the installation is done, the file containing the data can be imported into the application.



The image above shows how the application starts up when opened. There are 4 modes that weka can be operated in as explained in the previous pages. To import the dataset into the application, click on the first tab i.e, “Explore”.

Graphical user interface, application

Description automatically generated

The image above shows the application interface once we click on the “Explore” button. Now the dataset file can be imported into the application. To do that, we need to click on the “Open File” button on the top left corner of the application.

Graphical user interface, application

Description automatically generated

The image above shows the interface asking the directory of the file from the device. To import the file, go to the directory in which the file is stored and click on “open”. Weka supports only a certain kinds of files to be processed on.

Graphical user interface, text, application

Description automatically generated

The image above shows the selection of the type of file that the dataset is in. In this project, the dataset used is in the form of .csv. So, the option 5 in the list is selected. Once, the selection is done, the file is imported into the application. If the file has some unusual information, the application asks the user to choose a different file or fix the original file. It is advisable to check the dataset file before importing it into the application. Once the dataset file is loaded into the application, it looks as follows.

Graphical user interface, application

Description automatically generated

The image above shows the application asking the user to select the class by which the visualisation should be done. In this project, the classification is done using the class “Class Name”. The visualization in the image shows the distributions of data based on the class name. The image in the following page shows the classification algorithm to be selected from the options given in the top left corner saying “classify”.

Graphical user interface, application

Description automatically generated

Now, to choose the classification algorithm, click on “choose” under the “classify” tab on the top left corner of the screen.

Graphical user interface, application

Description automatically generated

The image above shows the selected algorithm from the options given. In this project, the classification algorithm used are Naïve Bayes and Decision Trees(J48). So, Naïve bayes is selected from the options. For choosing the Naïve Bayes, the data must contain the classification properties that support the algorithm.

Graphical user interface, application

Description automatically generated

Once the algorithm is selected, click on “start” button under the “Test options” interface. After clicking the button, the algorithm selected runs on the dataset that has been imported into the application by the user.

**Running the Naïve Bayes algorithm on the dataset:** The dataset chosen for the project is in .csv format and is split into columns containing column labels that helps for the classification.

Graphical user interface, application, table, Excel

Description automatically generated

The above image shows the dataset split into columns and having labels for each. Now, once all the pre-processing of the data is done, the classification of the data using the algorithm is done. The class selected for the classification of the data is “Class Name” which can be seen in the above image.Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generatedA picture containing graphical user interface

Description automatically generated

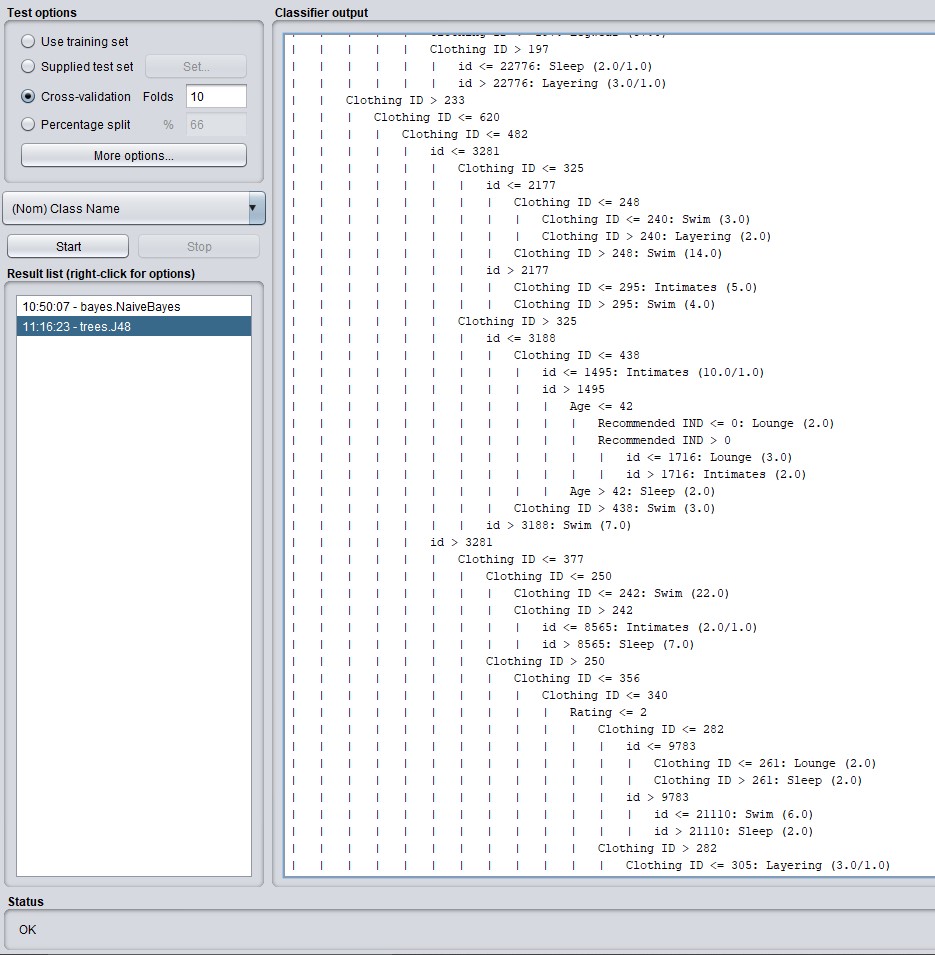
The images above show the working of the algorithm on the dataset. The first image shows the accuracy of the algorithm on the dataset. The time taken for the algorithm to run is 0.07 seconds. The accuracy of the algorithm is 83.382%. The percentage of the data that has not been classified incorrectly is 16.7178%. The second image shows the classifications of the dataset based on the attribute “class name” and shows the mean, std.deviation, weight sum, precision of each of the attributes.

The third image shows the detailed accuracy of each of the classes. The last row of the data in the image shows the weighted average of the classes. The fourth image shows the confusion matrix of each of the classes.

**Running the j48 algorithm on the dataset:**

**Graphical user interface, application

Description automatically generated**

**Table

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, table

Description automatically generatedGraphical user interface, application

Description automatically generated**

The images above show the working of the j48 algorithm on the dataset. The first three images show the j48 pruned tree of the dataset. The fourth image shows the performance of the algorithm on the dataset. The time taken for the execution of the algorithm is 0.27 seconds. The accuracy if the algorithm is 98.1808% and the percentage of the incorrectly classified instances is 1.8192%. The image also shows the detailed accuracy by each class of the dataset. The next image shows the confusion matrix of the algorithm. The last image shows the graph of the classification errors of the algorithm.

**Conclusion:** As far as accuracies are considered, the j48 algorithm has been more accurate than the Naïve Bayes algorithm with 98.1808% against 83.382%. The Naïve Bayes algorithm runs faster than the j48 algorithm with 0.07 seconds against 0.27 seconds. Based on the performance of both the algorithms, if the dataset is large j48 performs better than the Naïve Bayes algorithm. So, it can be concluded that the Decision Tree(j48) is a better algorithm than the Naïve Bayes algorithm.

**Source codes of the algorithms used:**

Source for the Naïve Bayes: <https://github.com/bnjmn/weka/blob/master/weka/src/main/java/weka/classifiers/bayes/NaiveBayes.java>

Source for J48: [https://github.com/bnjmn/weka/blob/master/weka/src/main/java/weka/classifi ers/trees/J48.java](https://github.com/bnjmn/weka/blob/master/weka/src/main/java/weka/classifi%20ers/trees/J48.java)

Naïve Bayes souce code:

*/\**

*\* This program is free software: you can redistribute it and/or modify*

*\* it under the terms of the GNU General Public License as published by*

*\* the Free Software Foundation, either version 3 of the License, or*

*\* (at your option) any later version.*

*\**

*\* This program is distributed in the hope that it will be useful,*

*\* but WITHOUT ANY WARRANTY; without even the implied warranty of*

*\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the*

*\* GNU General Public License for more details.*

*\**

*\* You should have received a copy of the GNU General Public License*

*\* along with this program. If not, see <http://www.gnu.org/licenses/>.*

*\*/*

*/\**

*\* NaiveBayes.java*

*\* Copyright (C) 1999-2012 University of Waikato, Hamilton, New Zealand*

*\**

*\*/*

**package** weka.classifiers.bayes;

**import** **java.util.Collections**;

**import** **java.util.Enumeration**;

**import** **java.util.Vector**;

**import** **weka.classifiers.AbstractClassifier**;

**import** **weka.core.\***;

**import** **weka.core.Capabilities.Capability**;

**import** **weka.core.TechnicalInformation.Field**;

**import** **weka.core.TechnicalInformation.Type**;

**import** **weka.estimators.DiscreteEstimator**;

**import** **weka.estimators.Estimator**;

**import** **weka.estimators.KernelEstimator**;

**import** **weka.estimators.NormalEstimator**;

*/\*\**

*\* <!-- globalinfo-start --> Class for a Naive Bayes classifier using estimator*

*\* classes. Numeric estimator precision values are chosen based on analysis of*

*\* the training data. For this reason, the classifier is not an*

*\* UpdateableClassifier (which in typical usage are initialized with zero*

*\* training instances) -- if you need the UpdateableClassifier functionality,*

*\* use the NaiveBayesUpdateable classifier. The NaiveBayesUpdateable classifier*

*\* will use a default precision of 0.1 for numeric attributes when*

*\* buildClassifier is called with zero training instances.<br/>*

*\* <br/>*

*\* For more information on Naive Bayes classifiers, see<br/>*

*\* <br/>*

*\* George H. John, Pat Langley: Estimating Continuous Distributions in Bayesian*

*\* Classifiers. In: Eleventh Conference on Uncertainty in Artificial*

*\* Intelligence, San Mateo, 338-345, 1995.*

*\* <p/>*

*\* <!-- globalinfo-end -->*

*\**

*\* <!-- technical-bibtex-start --> BibTeX:*

*\**

*\* <pre>*

*\* &#64;inproceedings{John1995,*

*\* address = {San Mateo},*

*\* author = {George H. John and Pat Langley},*

*\* booktitle = {Eleventh Conference on Uncertainty in Artificial Intelligence},*

*\* pages = {338-345},*

*\* publisher = {Morgan Kaufmann},*

*\* title = {Estimating Continuous Distributions in Bayesian Classifiers},*

*\* year = {1995}*

*\* }*

*\* </pre>*

*\* <p/>*

*\* <!-- technical-bibtex-end -->*

*\**

*\* <!-- options-start --> Valid options are:*

*\* <p/>*

*\**

*\* <pre>*

*\* -K*

*\* Use kernel density estimator rather than normal*

*\* distribution for numeric attributes*

*\* </pre>*

*\**

*\* <pre>*

*\* -D*

*\* Use supervised discretization to process numeric attributes*

*\* </pre>*

*\**

*\* <pre>*

*\* -O*

*\* Display model in old format (good when there are many classes)*

*\* </pre>*

*\**

*\* <!-- options-end -->*

*\**

*\* @author Len Trigg (trigg@cs.waikato.ac.nz)*

*\* @author Eibe Frank (eibe@cs.waikato.ac.nz)*

*\* @version $Revision$*

*\*/*

**public** **class** **NaiveBayes** **extends** AbstractClassifier **implements** OptionHandler,

WeightedInstancesHandler, WeightedAttributesHandler, TechnicalInformationHandler,

Aggregateable<NaiveBayes> {

*/\*\* for serialization \*/*

**static** **final** long serialVersionUID = 5995231201785697655L;

*/\*\* The attribute estimators. \*/*

**protected** Estimator[][] m\_Distributions;

*/\*\* The class estimator. \*/*

**protected** Estimator m\_ClassDistribution;

*/\*\**

*\* Whether to use kernel density estimator rather than normal distribution for*

*\* numeric attributes*

*\*/*

**protected** boolean m\_UseKernelEstimator = **false**;

*/\*\**

*\* Whether to use discretization than normal distribution for numeric*

*\* attributes*

*\*/*

**protected** boolean m\_UseDiscretization = **false**;

*/\*\* The number of classes (or 1 for numeric class) \*/*

**protected** int m\_NumClasses;

*/\*\**

*\* The dataset header for the purposes of printing out a semi-intelligible*

*\* model*

*\*/*

**protected** Instances m\_Instances;

*/\*\*\* The precision parameter used for numeric attributes \*/*

**protected** **static** **final** double DEFAULT\_NUM\_PRECISION = 0.01;

*/\*\**

*\* The discretization filter.*

*\*/*

**protected** weka.filters.supervised.attribute.Discretize m\_Disc = **null**;

**protected** boolean m\_displayModelInOldFormat = **false**;

*/\*\**

*\* Returns a string describing this classifier*

*\**

*\* @return a description of the classifier suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String globalInfo() {

**return** "Class for a Naive Bayes classifier using estimator classes. Numeric"

+ " estimator precision values are chosen based on analysis of the "

+ " training data. For this reason, the classifier is not an"

+ " UpdateableClassifier (which in typical usage are initialized with zero"

+ " training instances) -- if you need the UpdateableClassifier functionality,"

+ " use the NaiveBayesUpdateable classifier. The NaiveBayesUpdateable"

+ " classifier will use a default precision of 0.1 for numeric attributes"

+ " when buildClassifier is called with zero training instances.\n\n"

+ "For more information on Naive Bayes classifiers, see\n\n"

+ getTechnicalInformation().toString();

}

*/\*\**

*\* Returns an instance of a TechnicalInformation object, containing detailed*

*\* information about the technical background of this class, e.g., paper*

*\* reference or book this class is based on.*

*\**

*\* @return the technical information about this class*

*\*/*

@Override

**public** TechnicalInformation getTechnicalInformation() {

TechnicalInformation result;

result = **new** TechnicalInformation(Type.INPROCEEDINGS);

result.setValue(Field.AUTHOR, "George H. John and Pat Langley");

result.setValue(Field.TITLE,

"Estimating Continuous Distributions in Bayesian Classifiers");

result.setValue(Field.BOOKTITLE,

"Eleventh Conference on Uncertainty in Artificial Intelligence");

result.setValue(Field.YEAR, "1995");

result.setValue(Field.PAGES, "338-345");

result.setValue(Field.PUBLISHER, "Morgan Kaufmann");

result.setValue(Field.ADDRESS, "San Mateo");

**return** result;

}

*/\*\**

*\* Returns default capabilities of the classifier.*

*\**

*\* @return the capabilities of this classifier*

*\*/*

@Override

**public** Capabilities getCapabilities() {

Capabilities result = **super**.getCapabilities();

result.disableAll();

*// attributes*

result.enable(Capability.NOMINAL\_ATTRIBUTES);

result.enable(Capability.NUMERIC\_ATTRIBUTES);

result.enable( Capability.MISSING\_VALUES );

*// class*

result.enable(Capability.NOMINAL\_CLASS);

result.enable(Capability.MISSING\_CLASS\_VALUES);

*// instances*

result.setMinimumNumberInstances(0);

**return** result;

}

*/\*\**

*\* Generates the classifier.*

*\**

*\* @param instances set of instances serving as training data*

*\* @exception Exception if the classifier has not been generated successfully*

*\*/*

@Override

**public** void buildClassifier(Instances instances) **throws** Exception {

*// can classifier handle the data?*

getCapabilities().testWithFail(instances);

*// remove instances with missing class*

instances = **new** Instances(instances);

instances.deleteWithMissingClass();

m\_NumClasses = instances.numClasses();

*// Copy the instances*

m\_Instances = **new** Instances(instances);

*// Discretize instances if required*

**if** (m\_UseDiscretization) {

m\_Disc = **new** weka.filters.supervised.attribute.Discretize();

m\_Disc.setInputFormat(m\_Instances);

m\_Instances = weka.filters.Filter.useFilter(m\_Instances, m\_Disc);

} **else** {

m\_Disc = **null**;

}

*// Reserve space for the distributions*

m\_Distributions = **new** Estimator[m\_Instances.numAttributes() - 1][m\_Instances

.numClasses()];

m\_ClassDistribution = **new** DiscreteEstimator(m\_Instances.numClasses(), **true**);

int attIndex = 0;

Enumeration<Attribute> enu = m\_Instances.enumerateAttributes();

**while** (enu.hasMoreElements()) {

Attribute attribute = enu.nextElement();

*// If the attribute is numeric, determine the estimator*

*// numeric precision from differences between adjacent values*

double numPrecision = DEFAULT\_NUM\_PRECISION;

**if** (attribute.type() == Attribute.NUMERIC) {

m\_Instances.sort(attribute);

**if** ((m\_Instances.numInstances() > 0)

&& !m\_Instances.instance(0).isMissing(attribute)) {

double lastVal = m\_Instances.instance(0).value(attribute);

double currentVal, deltaSum = 0;

int distinct = 0;

**for** (int i = 1; i < m\_Instances.numInstances(); i++) {

Instance currentInst = m\_Instances.instance(i);

**if** (currentInst.isMissing(attribute)) {

**break**;

}

currentVal = currentInst.value(attribute);

**if** (currentVal != lastVal) {

deltaSum += currentVal - lastVal;

lastVal = currentVal;

distinct++;

}

}

**if** (distinct > 0) {

numPrecision = deltaSum / distinct;

}

}

}

**for** (int j = 0; j < m\_Instances.numClasses(); j++) {

**switch** (attribute.type()) {

**case** Attribute.NUMERIC:

**if** (m\_UseKernelEstimator) {

m\_Distributions[attIndex][j] = **new** KernelEstimator(numPrecision);

} **else** {

m\_Distributions[attIndex][j] = **new** NormalEstimator(numPrecision);

}

**break**;

**case** Attribute.NOMINAL:

m\_Distributions[attIndex][j] = **new** DiscreteEstimator(

attribute.numValues(), **true**);

**break**;

**default**:

**throw** **new** Exception("Attribute type unknown to NaiveBayes");

}

}

attIndex++;

}

*// Compute counts*

Enumeration<Instance> enumInsts = m\_Instances.enumerateInstances();

**while** (enumInsts.hasMoreElements()) {

Instance instance = enumInsts.nextElement();

updateClassifier(instance);

}

*// Save space*

m\_Instances = **new** Instances(m\_Instances, 0);

}

*/\*\**

*\* Updates the classifier with the given instance.*

*\**

*\* @param instance the new training instance to include in the model*

*\* @exception Exception if the instance could not be incorporated in the*

*\* model.*

*\*/*

**public** void updateClassifier(Instance instance) **throws** Exception {

**if** (!instance.classIsMissing()) {

Enumeration<Attribute> enumAtts = m\_Instances.enumerateAttributes();

int attIndex = 0;

**while** (enumAtts.hasMoreElements()) {

Attribute attribute = enumAtts.nextElement();

**if** (!instance.isMissing(attribute)) {

m\_Distributions[attIndex][(int) instance.classValue()].addValue(

instance.value(attribute), instance.weight());

}

attIndex++;

}

m\_ClassDistribution.addValue(instance.classValue(), instance.weight());

}

}

*/\*\**

*\* Calculates the class membership probabilities for the given test instance.*

*\**

*\* @param instance the instance to be classified*

*\* @return predicted class probability distribution*

*\* @exception Exception if there is a problem generating the prediction*

*\*/*

@Override

**public** double[] distributionForInstance(Instance instance) **throws** Exception {

**if** (m\_UseDiscretization) {

m\_Disc.input(instance);

instance = m\_Disc.output();

}

double[] probs = **new** double[m\_NumClasses];

**for** (int j = 0; j < m\_NumClasses; j++) {

probs[j] = m\_ClassDistribution.getProbability(j);

}

Enumeration<Attribute> enumAtts = instance.enumerateAttributes();

int attIndex = 0;

**while** (enumAtts.hasMoreElements()) {

Attribute attribute = enumAtts.nextElement();

**if** (!instance.isMissing(attribute)) {

double temp, max = 0;

**for** (int j = 0; j < m\_NumClasses; j++) {

temp = Math.max(1e-75, Math.pow(m\_Distributions[attIndex][j]

.getProbability(instance.value(attribute)),

m\_Instances.attribute(attIndex).weight()));

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"

+ m\_Distributions[attIndex][j].toString());

}

}

**if** ((max > 0) && (max < 1e-75)) { *// Danger of probability underflow*

**for** (int j = 0; j < m\_NumClasses; j++) {

probs[j] \*= 1e75;

}

}

}

attIndex++;

}

*// Display probabilities*

Utils.normalize(probs);

**return** probs;

}

*/\*\**

*\* Returns an enumeration describing the available options.*

*\**

*\* @return an enumeration of all the available options.*

*\*/*

@Override

**public** Enumeration<Option> listOptions() {

Vector<Option> newVector = **new** Vector<Option>(3);

newVector.addElement(**new** Option(

"\tUse kernel density estimator rather than normal\n"

+ "\tdistribution for numeric attributes", "K", 0, "-K"));

newVector.addElement(**new** Option(

"\tUse supervised discretization to process numeric attributes\n", "D",

0, "-D"));

newVector

.addElement(**new** Option(

"\tDisplay model in old format (good when there are "

+ "many classes)\n", "O", 0, "-O"));

newVector.addAll(Collections.list(**super**.listOptions()));

**return** newVector.elements();

}

*/\*\**

*\* Parses a given list of options.*

*\* <p/>*

*\**

*\* <!-- options-start --> Valid options are:*

*\* <p/>*

*\**

*\* <pre>*

*\* -K*

*\* Use kernel density estimator rather than normal*

*\* distribution for numeric attributes*

*\* </pre>*

*\**

*\* <pre>*

*\* -D*

*\* Use supervised discretization to process numeric attributes*

*\* </pre>*

*\**

*\* <pre>*

*\* -O*

*\* Display model in old format (good when there are many classes)*

*\* </pre>*

*\**

*\* <!-- options-end -->*

*\**

*\* @param options the list of options as an array of strings*

*\* @exception Exception if an option is not supported*

*\*/*

@Override

**public** void setOptions(String[] options) **throws** Exception {

**super**.setOptions(options);

boolean k = Utils.getFlag('K', options);

boolean d = Utils.getFlag('D', options);

**if** (k && d) {

**throw** **new** IllegalArgumentException("Can't use both kernel density "

+ "estimation and discretization!");

}

setUseSupervisedDiscretization(d);

setUseKernelEstimator(k);

setDisplayModelInOldFormat(Utils.getFlag('O', options));

Utils.checkForRemainingOptions(options);

}

*/\*\**

*\* Gets the current settings of the classifier.*

*\**

*\* @return an array of strings suitable for passing to setOptions*

*\*/*

@Override

**public** String[] getOptions() {

Vector<String> options = **new** Vector<String>();

Collections.addAll(options, **super**.getOptions());

**if** (m\_UseKernelEstimator) {

options.add("-K");

}

**if** (m\_UseDiscretization) {

options.add("-D");

}

**if** (m\_displayModelInOldFormat) {

options.add("-O");

}

**return** options.toArray(**new** String[0]);

}

*/\*\**

*\* Returns a description of the classifier.*

*\**

*\* @return a description of the classifier as a string.*

*\*/*

@Override

**public** String toString() {

**if** (m\_displayModelInOldFormat) {

**return** toStringOriginal();

}

StringBuffer temp = **new** StringBuffer();

temp.append("Naive Bayes Classifier");

**if** (m\_Instances == **null**) {

temp.append(": No model built yet.");

} **else** {

int maxWidth = 0;

int maxAttWidth = 0;

boolean containsKernel = **false**;

*// set up max widths*

*// class values*

**for** (int i = 0; i < m\_Instances.numClasses(); i++) {

**if** (m\_Instances.classAttribute().value(i).length() > maxWidth) {

maxWidth = m\_Instances.classAttribute().value(i).length();

}

}

*// attributes*

**for** (int i = 0; i < m\_Instances.numAttributes(); i++) {

**if** (i != m\_Instances.classIndex()) {

Attribute a = m\_Instances.attribute(i);

**if** (a.name().length() > maxAttWidth) {

maxAttWidth = m\_Instances.attribute(i).name().length();

}

**if** (a.isNominal()) {

*// check values*

**for** (int j = 0; j < a.numValues(); j++) {

String val = a.value(j) + " ";

**if** (val.length() > maxAttWidth) {

maxAttWidth = val.length();

}

}

}

}

}

**for** (Estimator[] m\_Distribution : m\_Distributions) {

**for** (int j = 0; j < m\_Instances.numClasses(); j++) {

**if** (m\_Distribution[0] **instanceof** NormalEstimator) {

*// check mean/precision dev against maxWidth*

NormalEstimator n = (NormalEstimator) m\_Distribution[j];

double mean = Math.log(Math.abs(n.getMean())) / Math.log(10.0);

double precision = Math.log(Math.abs(n.getPrecision()))

/ Math.log(10.0);

double width = (mean > precision) ? mean : precision;

**if** (width < 0) {

width = 1;

}

*// decimal + # decimal places + 1*

width += 6.0;

**if** ((int) width > maxWidth) {

maxWidth = (int) width;

}

} **else** **if** (m\_Distribution[0] **instanceof** KernelEstimator) {

containsKernel = **true**;

KernelEstimator ke = (KernelEstimator) m\_Distribution[j];

int numK = ke.getNumKernels();

String temps = "K" + numK + ": mean (weight)";

**if** (maxAttWidth < temps.length()) {

maxAttWidth = temps.length();

}

*// check means + weights against maxWidth*

**if** (ke.getNumKernels() > 0) {

double[] means = ke.getMeans();

double[] weights = ke.getWeights();

**for** (int k = 0; k < ke.getNumKernels(); k++) {

String m = Utils.doubleToString(means[k], maxWidth, 4).trim();

m += " ("

+ Utils.doubleToString(weights[k], maxWidth, 1).trim() + ")";

**if** (maxWidth < m.length()) {

maxWidth = m.length();

}

}

}

} **else** **if** (m\_Distribution[0] **instanceof** DiscreteEstimator) {

DiscreteEstimator d = (DiscreteEstimator) m\_Distribution[j];

**for** (int k = 0; k < d.getNumSymbols(); k++) {

String size = "" + d.getCount(k);

**if** (size.length() > maxWidth) {

maxWidth = size.length();

}

}

int sum = ("" + d.getSumOfCounts()).length();

**if** (sum > maxWidth) {

maxWidth = sum;

}

}

}

}

*// Check width of class labels*

**for** (int i = 0; i < m\_Instances.numClasses(); i++) {

String cSize = m\_Instances.classAttribute().value(i);

**if** (cSize.length() > maxWidth) {

maxWidth = cSize.length();

}

}

*// Check width of class priors*

**for** (int i = 0; i < m\_Instances.numClasses(); i++) {

String priorP = Utils.doubleToString(

((DiscreteEstimator) m\_ClassDistribution).getProbability(i),

maxWidth, 2).trim();

priorP = "(" + priorP + ")";

**if** (priorP.length() > maxWidth) {

maxWidth = priorP.length();

}

}

**if** (maxAttWidth < "Attribute".length()) {

maxAttWidth = "Attribute".length();

}

**if** (maxAttWidth < " weight sum".length()) {

maxAttWidth = " weight sum".length();

}

**if** (containsKernel) {

**if** (maxAttWidth < " [precision]".length()) {

maxAttWidth = " [precision]".length();

}

}

maxAttWidth += 2;

temp.append("\n\n");

temp.append(pad("Class", " ",

(maxAttWidth + maxWidth + 1) - "Class".length(), **true**));

temp.append("\n");

temp.append(pad("Attribute", " ", maxAttWidth - "Attribute".length(),

**false**));

*// class labels*

**for** (int i = 0; i < m\_Instances.numClasses(); i++) {

String classL = m\_Instances.classAttribute().value(i);

temp.append(pad(classL, " ", maxWidth + 1 - classL.length(), **true**));

}

temp.append("\n");

*// class priors*

temp.append(pad("", " ", maxAttWidth, **true**));

**for** (int i = 0; i < m\_Instances.numClasses(); i++) {

String priorP = Utils.doubleToString(

((DiscreteEstimator) m\_ClassDistribution).getProbability(i),

maxWidth, 2).trim();

priorP = "(" + priorP + ")";

temp.append(pad(priorP, " ", maxWidth + 1 - priorP.length(), **true**));

}

temp.append("\n");

temp.append(pad(

"",

"=",

maxAttWidth + (maxWidth \* m\_Instances.numClasses())

+ m\_Instances.numClasses() + 1, **true**));

temp.append("\n");

*// loop over the attributes*

int counter = 0;

**for** (int i = 0; i < m\_Instances.numAttributes(); i++) {

**if** (i == m\_Instances.classIndex()) {

**continue**;

}

String attName = m\_Instances.attribute(i).name();

temp.append(attName + "\n");

**if** (m\_Distributions[counter][0] **instanceof** NormalEstimator) {

String meanL = " mean";

temp.append(pad(meanL, " ", maxAttWidth + 1 - meanL.length(), **false**));

**for** (int j = 0; j < m\_Instances.numClasses(); j++) {

*// means*

NormalEstimator n = (NormalEstimator) m\_Distributions[counter][j];

String mean = Utils.doubleToString(n.getMean(), maxWidth, 4).trim();

temp.append(pad(mean, " ", maxWidth + 1 - mean.length(), **true**));

}

temp.append("\n");

*// now do std deviations*

String stdDevL = " std. dev.";

temp.append(pad(stdDevL, " ", maxAttWidth + 1 - stdDevL.length(),

**false**));

**for** (int j = 0; j < m\_Instances.numClasses(); j++) {

NormalEstimator n = (NormalEstimator) m\_Distributions[counter][j];

String stdDev = Utils.doubleToString(n.getStdDev(), maxWidth, 4)

.trim();

temp.append(pad(stdDev, " ", maxWidth + 1 - stdDev.length(), **true**));

}

temp.append("\n");

*// now the weight sums*

String weightL = " weight sum";

temp.append(pad(weightL, " ", maxAttWidth + 1 - weightL.length(),

**false**));

**for** (int j = 0; j < m\_Instances.numClasses(); j++) {

NormalEstimator n = (NormalEstimator) m\_Distributions[counter][j];

String weight = Utils.doubleToString(n.getSumOfWeights(), maxWidth,

4).trim();

temp.append(pad(weight, " ", maxWidth + 1 - weight.length(), **true**));

}

temp.append("\n");

*// now the precisions*

String precisionL = " precision";

temp.append(pad(precisionL, " ",

maxAttWidth + 1 - precisionL.length(), **false**));

**for** (int j = 0; j < m\_Instances.numClasses(); j++) {

NormalEstimator n = (NormalEstimator) m\_Distributions[counter][j];

String precision = Utils.doubleToString(n.getPrecision(), maxWidth,

4).trim();

temp.append(pad(precision, " ", maxWidth + 1 - precision.length(),

**true**));

}

temp.append("\n\n");

} **else** **if** (m\_Distributions[counter][0] **instanceof** DiscreteEstimator) {

Attribute a = m\_Instances.attribute(i);

**for** (int j = 0; j < a.numValues(); j++) {

String val = " " + a.value(j);

temp.append(pad(val, " ", maxAttWidth + 1 - val.length(), **false**));

**for** (int k = 0; k < m\_Instances.numClasses(); k++) {

DiscreteEstimator d = (DiscreteEstimator) m\_Distributions[counter][k];

String count = "" + d.getCount(j);

temp.append(pad(count, " ", maxWidth + 1 - count.length(), **true**));

}

temp.append("\n");

}

*// do the totals*

String total = " [total]";

temp.append(pad(total, " ", maxAttWidth + 1 - total.length(), **false**));

**for** (int k = 0; k < m\_Instances.numClasses(); k++) {

DiscreteEstimator d = (DiscreteEstimator) m\_Distributions[counter][k];

String count = "" + d.getSumOfCounts();

temp.append(pad(count, " ", maxWidth + 1 - count.length(), **true**));

}

temp.append("\n\n");

} **else** **if** (m\_Distributions[counter][0] **instanceof** KernelEstimator) {

String kL = " [# kernels]";

temp.append(pad(kL, " ", maxAttWidth + 1 - kL.length(), **false**));

**for** (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator) m\_Distributions[counter][k];

String nk = "" + ke.getNumKernels();

temp.append(pad(nk, " ", maxWidth + 1 - nk.length(), **true**));

}

temp.append("\n");

*// do num kernels, std. devs and precisions*

String stdDevL = " [std. dev]";

temp.append(pad(stdDevL, " ", maxAttWidth + 1 - stdDevL.length(),

**false**));

**for** (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator) m\_Distributions[counter][k];

String stdD = Utils.doubleToString(ke.getStdDev(), maxWidth, 4)

.trim();

temp.append(pad(stdD, " ", maxWidth + 1 - stdD.length(), **true**));

}

temp.append("\n");

String precL = " [precision]";

temp.append(pad(precL, " ", maxAttWidth + 1 - precL.length(), **false**));

**for** (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator) m\_Distributions[counter][k];

String prec = Utils.doubleToString(ke.getPrecision(), maxWidth, 4)

.trim();

temp.append(pad(prec, " ", maxWidth + 1 - prec.length(), **true**));

}

temp.append("\n");

*// first determine max number of kernels accross the classes*

int maxK = 0;

**for** (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator) m\_Distributions[counter][k];

**if** (ke.getNumKernels() > maxK) {

maxK = ke.getNumKernels();

}

}

**for** (int j = 0; j < maxK; j++) {

*// means first*

String meanL = " K" + (j + 1) + ": mean (weight)";

temp

.append(pad(meanL, " ", maxAttWidth + 1 - meanL.length(), **false**));

**for** (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator) m\_Distributions[counter][k];

double[] means = ke.getMeans();

double[] weights = ke.getWeights();

String m = "--";

**if** (ke.getNumKernels() == 0) {

m = "" + 0;

} **else** **if** (j < ke.getNumKernels()) {

m = Utils.doubleToString(means[j], maxWidth, 4).trim();

m += " ("

+ Utils.doubleToString(weights[j], maxWidth, 1).trim() + ")";

}

temp.append(pad(m, " ", maxWidth + 1 - m.length(), **true**));

}

temp.append("\n");

}

temp.append("\n");

}

counter++;

}

}

**return** temp.toString();

}

*/\*\**

*\* Returns a description of the classifier in the old format.*

*\**

*\* @return a description of the classifier as a string.*

*\*/*

**protected** String toStringOriginal() {

StringBuffer text = **new** StringBuffer();

text.append("Naive Bayes Classifier");

**if** (m\_Instances == **null**) {

text.append(": No model built yet.");

} **else** {

**try** {

**for** (int i = 0; i < m\_Distributions[0].length; i++) {

text.append("\n\nClass " + m\_Instances.classAttribute().value(i)

+ ": Prior probability = "

+ Utils.doubleToString(m\_ClassDistribution.getProbability(i), 4, 2)

+ "\n\n");

Enumeration<Attribute> enumAtts = m\_Instances.enumerateAttributes();

int attIndex = 0;

**while** (enumAtts.hasMoreElements()) {

Attribute attribute = enumAtts.nextElement();

**if** (attribute.weight() > 0) {

text.append(attribute.name() + ": "

+ m\_Distributions[attIndex][i]);

}

attIndex++;

}

}

} **catch** (Exception ex) {

text.append(ex.getMessage());

}

}

**return** text.toString();

}

**private** String pad(String source, String padChar, int length, boolean leftPad) {

StringBuffer temp = **new** StringBuffer();

**if** (leftPad) {

**for** (int i = 0; i < length; i++) {

temp.append(padChar);

}

temp.append(source);

} **else** {

temp.append(source);

**for** (int i = 0; i < length; i++) {

temp.append(padChar);

}

}

**return** temp.toString();

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String useKernelEstimatorTipText() {

**return** "Use a kernel estimator for numeric attributes rather than a "

+ "normal distribution.";

}

*/\*\**

*\* Gets if kernel estimator is being used.*

*\**

*\* @return Value of m\_UseKernelEstimatory.*

*\*/*

**public** boolean getUseKernelEstimator() {

**return** m\_UseKernelEstimator;

}

*/\*\**

*\* Sets if kernel estimator is to be used.*

*\**

*\* @param v Value to assign to m\_UseKernelEstimatory.*

*\*/*

**public** void setUseKernelEstimator(boolean v) {

m\_UseKernelEstimator = v;

**if** (v) {

setUseSupervisedDiscretization(**false**);

}

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String useSupervisedDiscretizationTipText() {

**return** "Use supervised discretization to convert numeric attributes to nominal "

+ "ones.";

}

*/\*\**

*\* Get whether supervised discretization is to be used.*

*\**

*\* @return true if supervised discretization is to be used.*

*\*/*

**public** boolean getUseSupervisedDiscretization() {

**return** m\_UseDiscretization;

}

*/\*\**

*\* Set whether supervised discretization is to be used.*

*\**

*\* @param newblah true if supervised discretization is to be used.*

*\*/*

**public** void setUseSupervisedDiscretization(boolean newblah) {

m\_UseDiscretization = newblah;

**if** (newblah) {

setUseKernelEstimator(**false**);

}

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String displayModelInOldFormatTipText() {

**return** "Use old format for model output. The old format is "

+ "better when there are many class values. The new format "

+ "is better when there are fewer classes and many attributes.";

}

*/\*\**

*\* Set whether to display model output in the old, original format.*

*\**

*\* @param d true if model ouput is to be shown in the old format*

*\*/*

**public** void setDisplayModelInOldFormat(boolean d) {

m\_displayModelInOldFormat = d;

}

*/\*\**

*\* Get whether to display model output in the old, original format.*

*\**

*\* @return true if model ouput is to be shown in the old format*

*\*/*

**public** boolean getDisplayModelInOldFormat() {

**return** m\_displayModelInOldFormat;

}

*/\*\**

*\* Return the header that this classifier was trained with*

*\**

*\* @return the header that this classifier was trained with*

*\*/*

**public** Instances getHeader() {

**return** m\_Instances;

}

*/\*\**

*\* Get all the conditional estimators.*

*\**

*\* @return all the conditional estimators.*

*\*/*

**public** Estimator[][] getConditionalEstimators() {

**return** m\_Distributions;

}

*/\*\**

*\* Get the class estimator.*

*\**

*\* @return the class estimator*

*\*/*

**public** Estimator getClassEstimator() {

**return** m\_ClassDistribution;

}

*/\*\**

*\* Returns the revision string.*

*\**

*\* @return the revision*

*\*/*

@Override

**public** String getRevision() {

**return** RevisionUtils.extract("$Revision$");

}

@SuppressWarnings({ "rawtypes", "unchecked" })

@Override

**public** NaiveBayes aggregate(NaiveBayes toAggregate) **throws** Exception {

*// Highly unlikely that discretization intervals will match between the*

*// two classifiers*

**if** (m\_UseDiscretization || toAggregate.getUseSupervisedDiscretization()) {

**throw** **new** Exception("Unable to aggregate when supervised discretization "

+ "has been turned on");

}

**if** (!m\_Instances.equalHeaders(toAggregate.m\_Instances)) {

**throw** **new** Exception("Can't aggregate - data headers don't match: "

+ m\_Instances.equalHeadersMsg(toAggregate.m\_Instances));

}

((Aggregateable) m\_ClassDistribution)

.aggregate(toAggregate.m\_ClassDistribution);

*// aggregate all conditional estimators*

**for** (int i = 0; i < m\_Distributions.length; i++) {

**for** (int j = 0; j < m\_Distributions[i].length; j++) {

((Aggregateable) m\_Distributions[i][j])

.aggregate(toAggregate.m\_Distributions[i][j]);

}

}

**return** **this**;

}

@Override

**public** void finalizeAggregation() **throws** Exception {

*// nothing to do*

}

*/\*\**

*\* Main method for testing this class.*

*\**

*\* @param argv the options*

*\*/*

**public** **static** void main(String[] argv) {

runClassifier(**new** NaiveBayes(), argv);

}

}

J48 Source code:

*/\**

*\* This program is free software: you can redistribute it and/or modify*

*\* it under the terms of the GNU General Public License as published by*

*\* the Free Software Foundation, either version 3 of the License, or*

*\* (at your option) any later version.*

*\**

*\* This program is distributed in the hope that it will be useful,*

*\* but WITHOUT ANY WARRANTY; without even the implied warranty of*

*\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the*

*\* GNU General Public License for more details.*

*\**

*\* You should have received a copy of the GNU General Public License*

*\* along with this program. If not, see <http://www.gnu.org/licenses/>.*

*\*/*

*/\**

*\* J48.java*

*\* Copyright (C) 1999-2012 University of Waikato, Hamilton, New Zealand*

*\**

*\*/*

**package** weka.classifiers.trees;

**import** **java.util.Collections**;

**import** **java.util.Enumeration**;

**import** **java.util.Vector**;

**import** **weka.classifiers.AbstractClassifier**;

**import** **weka.classifiers.Sourcable**;

**import** **weka.classifiers.trees.j48.BinC45ModelSelection**;

**import** **weka.classifiers.trees.j48.C45ModelSelection**;

**import** **weka.classifiers.trees.j48.C45PruneableClassifierTree**;

**import** **weka.classifiers.trees.j48.ClassifierTree**;

**import** **weka.classifiers.trees.j48.ModelSelection**;

**import** **weka.classifiers.trees.j48.PruneableClassifierTree**;

**import** **weka.core.AdditionalMeasureProducer**;

**import** **weka.core.Capabilities**;

**import** **weka.core.Drawable**;

**import** **weka.core.Instance**;

**import** **weka.core.Instances**;

**import** **weka.core.Matchable**;

**import** **weka.core.Option**;

**import** **weka.core.OptionHandler**;

**import** **weka.core.PartitionGenerator**;

**import** **weka.core.RevisionUtils**;

**import** **weka.core.Summarizable**;

**import** **weka.core.TechnicalInformation**;

**import** **weka.core.TechnicalInformation.Field**;

**import** **weka.core.TechnicalInformation.Type**;

**import** **weka.core.TechnicalInformationHandler**;

**import** **weka.core.Utils**;

**import** **weka.core.WeightedInstancesHandler**;

**import** **weka.core.Capabilities**;

**import** **weka.core.Capabilities.Capability**;

*/\*\**

*\* <!-- globalinfo-start --> Class for generating a pruned or unpruned C4.5*

*\* decision tree. For more information, see<br/>*

*\* <br/>*

*\* Ross Quinlan (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann*

*\* Publishers, San Mateo, CA.*

*\* <p/>*

*\* <!-- globalinfo-end -->*

*\**

*\* <!-- technical-bibtex-start --> BibTeX:*

*\**

*\* <pre>*

*\* &#64;book{Quinlan1993,*

*\* address = {San Mateo, CA},*

*\* author = {Ross Quinlan},*

*\* publisher = {Morgan Kaufmann Publishers},*

*\* title = {C4.5: Programs for Machine Learning},*

*\* year = {1993}*

*\* }*

*\* </pre>*

*\* <p/>*

*\* <!-- technical-bibtex-end -->*

*\**

*\* <!-- options-start --> Valid options are:*

*\* <p/>*

*\**

*\* <pre>*

*\* -U*

*\* Use unpruned tree.*

*\* </pre>*

*\**

*\* <pre>*

*\* -O*

*\* Do not collapse tree.*

*\* </pre>*

*\**

*\* <pre>*

*\* -C &lt;pruning confidence&gt;*

*\* Set confidence threshold for pruning.*

*\* (default 0.25)*

*\* </pre>*

*\**

*\* <pre>*

*\* -M &lt;minimum number of instances&gt;*

*\* Set minimum number of instances per leaf.*

*\* (default 2)*

*\* </pre>*

*\**

*\* <pre>*

*\* -R*

*\* Use reduced error pruning.*

*\* </pre>*

*\**

*\* <pre>*

*\* -N &lt;number of folds&gt;*

*\* Set number of folds for reduced error*

*\* pruning. One fold is used as pruning set.*

*\* (default 3)*

*\* </pre>*

*\**

*\* <pre>*

*\* -B*

*\* Use binary splits only.*

*\* </pre>*

*\**

*\* <pre>*

*\* -S*

*\* Don't perform subtree raising.*

*\* </pre>*

*\**

*\* <pre>*

*\* -L*

*\* Do not clean up after the tree has been built.*

*\* </pre>*

*\**

*\* <pre>*

*\* -A*

*\* Laplace smoothing for predicted probabilities.*

*\* </pre>*

*\**

*\* <pre>*

*\* -J*

*\* Do not use MDL correction for info gain on numeric attributes.*

*\* </pre>*

*\**

*\* <pre>*

*\* -Q &lt;seed&gt;*

*\* Seed for random data shuffling (default 1).*

*\* </pre>*

*\**

*\* <pre>*

*\* -doNotMakeSplitPointActualValue*

*\* Do not make split point actual value.*

*\* </pre>*

*\**

*\* <!-- options-end -->*

*\**

*\* @author Eibe Frank (eibe@cs.waikato.ac.nz)*

*\* @version $Revision$*

*\*/*

**public** **class** **J48** **extends** AbstractClassifier **implements** OptionHandler, Drawable,

Matchable, Sourcable, WeightedInstancesHandler, Summarizable,

AdditionalMeasureProducer, TechnicalInformationHandler, PartitionGenerator {

*/\*\* for serialization \*/*

**static** **final** long serialVersionUID = -217733168393644444L;

*/\*\* The decision tree \*/*

**protected** ClassifierTree m\_root;

*/\*\* Unpruned tree? \*/*

**protected** boolean m\_unpruned = **false**;

*/\*\* Collapse tree? \*/*

**protected** boolean m\_collapseTree = **true**;

*/\*\* Confidence level \*/*

**protected** float m\_CF = 0.25f;

*/\*\* Minimum number of instances \*/*

**protected** int m\_minNumObj = 2;

*/\*\* Use MDL correction? \*/*

**protected** boolean m\_useMDLcorrection = **true**;

*/\*\**

*\* Determines whether probabilities are smoothed using Laplace correction when*

*\* predictions are generated*

*\*/*

**protected** boolean m\_useLaplace = **false**;

*/\*\* Use reduced error pruning? \*/*

**protected** boolean m\_reducedErrorPruning = **false**;

*/\*\* Number of folds for reduced error pruning. \*/*

**protected** int m\_numFolds = 3;

*/\*\* Binary splits on nominal attributes? \*/*

**protected** boolean m\_binarySplits = **false**;

*/\*\* Subtree raising to be performed? \*/*

**protected** boolean m\_subtreeRaising = **true**;

*/\*\* Cleanup after the tree has been built. \*/*

**protected** boolean m\_noCleanup = **false**;

*/\*\* Random number seed for reduced-error pruning. \*/*

**protected** int m\_Seed = 1;

*/\*\* Do not relocate split point to actual data value \*/*

**protected** boolean m\_doNotMakeSplitPointActualValue;

*/\*\**

*\* Returns a string describing classifier*

*\**

*\* @return a description suitable for displaying in the explorer/experimenter*

*\* gui*

*\*/*

**public** String globalInfo() {

**return** "Class for generating a pruned or unpruned C4.5 decision tree. For more "

+ "information, see\n\n" + getTechnicalInformation().toString();

}

*/\*\**

*\* Returns an instance of a TechnicalInformation object, containing detailed*

*\* information about the technical background of this class, e.g., paper*

*\* reference or book this class is based on.*

*\**

*\* @return the technical information about this class*

*\*/*

@Override

**public** TechnicalInformation getTechnicalInformation() {

TechnicalInformation result;

result = **new** TechnicalInformation(Type.BOOK);

result.setValue(Field.AUTHOR, "Ross Quinlan");

result.setValue(Field.YEAR, "1993");

result.setValue(Field.TITLE, "C4.5: Programs for Machine Learning");

result.setValue(Field.PUBLISHER, "Morgan Kaufmann Publishers");

result.setValue(Field.ADDRESS, "San Mateo, CA");

**return** result;

}

*/\*\**

*\* Returns default capabilities of the classifier.*

*\**

*\* @return the capabilities of this classifier*

*\*/*

@Override

**public** Capabilities getCapabilities() {

Capabilities result;

result = **new** Capabilities(**this**);

result.disableAll();

*// attributes*

result.enable(Capability.NOMINAL\_ATTRIBUTES);

result.enable(Capability.NUMERIC\_ATTRIBUTES);

result.enable(Capability.DATE\_ATTRIBUTES);

result.enable(Capability.MISSING\_VALUES);

*// class*

result.enable(Capability.NOMINAL\_CLASS);

result.enable(Capability.MISSING\_CLASS\_VALUES);

*// instances*

result.setMinimumNumberInstances(0);

**return** result;

}

*/\*\**

*\* Generates the classifier.*

*\**

*\* @param instances the data to train the classifier with*

*\* @throws Exception if classifier can't be built successfully*

*\*/*

@Override

**public** void buildClassifier(Instances instances) **throws** Exception {

getCapabilities().testWithFail(instances);

ModelSelection modSelection;

**if** (m\_binarySplits) {

modSelection = **new** BinC45ModelSelection(m\_minNumObj, instances,

m\_useMDLcorrection, m\_doNotMakeSplitPointActualValue);

} **else** {

modSelection = **new** C45ModelSelection(m\_minNumObj, instances,

m\_useMDLcorrection, m\_doNotMakeSplitPointActualValue);

}

**if** (!m\_reducedErrorPruning) {

m\_root = **new** C45PruneableClassifierTree(modSelection, !m\_unpruned, m\_CF,

m\_subtreeRaising, !m\_noCleanup, m\_collapseTree);

} **else** {

m\_root = **new** PruneableClassifierTree(modSelection, !m\_unpruned,

m\_numFolds, !m\_noCleanup, m\_Seed);

}

m\_root.buildClassifier(instances);

**if** (m\_binarySplits) {

((BinC45ModelSelection) modSelection).cleanup();

} **else** {

((C45ModelSelection) modSelection).cleanup();

}

}

*/\*\**

*\* Classifies an instance.*

*\**

*\* @param instance the instance to classify*

*\* @return the classification for the instance*

*\* @throws Exception if instance can't be classified successfully*

*\*/*

@Override

**public** double classifyInstance(Instance instance) **throws** Exception {

**return** m\_root.classifyInstance(instance);

}

*/\*\**

*\* Returns class probabilities for an instance.*

*\**

*\* @param instance the instance to calculate the class probabilities for*

*\* @return the class probabilities*

*\* @throws Exception if distribution can't be computed successfully*

*\*/*

@Override

**public** **final** double[] distributionForInstance(Instance instance)

**throws** Exception {

**return** m\_root.distributionForInstance(instance, m\_useLaplace);

}

*/\*\**

*\* Returns the type of graph this classifier represents.*

*\**

*\* @return Drawable.TREE*

*\*/*

@Override

**public** int graphType() {

**return** Drawable.TREE;

}

*/\*\**

*\* Returns graph describing the tree.*

*\**

*\* @return the graph describing the tree*

*\* @throws Exception if graph can't be computed*

*\*/*

@Override

**public** String graph() **throws** Exception {

**return** m\_root.graph();

}

*/\*\**

*\* Returns tree in prefix order.*

*\**

*\* @return the tree in prefix order*

*\* @throws Exception if something goes wrong*

*\*/*

@Override

**public** String prefix() **throws** Exception {

**return** m\_root.prefix();

}

*/\*\**

*\* Returns tree as an if-then statement.*

*\**

*\* @param className the name of the Java class*

*\* @return the tree as a Java if-then type statement*

*\* @throws Exception if something goes wrong*

*\*/*

@Override

**public** String toSource(String className) **throws** Exception {

StringBuffer[] source = m\_root.toSource(className);

**return** "class " + className + " {\n\n"

+ " public static double classify(Object[] i)\n"

+ " throws Exception {\n\n" + " double p = Double.NaN;\n"

+ source[0] *// Assignment code*

+ " return p;\n" + " }\n" + source[1] *// Support code*

+ "}\n";

}

*/\*\**

*\* Returns an enumeration describing the available options.*

*\**

*\* Valid options are:*

*\* <p>*

*\**

*\* -U <br>*

*\* Use unpruned tree.*

*\* <p>*

*\**

*\* -C confidence <br>*

*\* Set confidence threshold for pruning. (Default: 0.25)*

*\* <p>*

*\**

*\* -M number <br>*

*\* Set minimum number of instances per leaf. (Default: 2)*

*\* <p>*

*\**

*\* -R <br>*

*\* Use reduced error pruning. No subtree raising is performed.*

*\* <p>*

*\**

*\* -N number <br>*

*\* Set number of folds for reduced error pruning. One fold is used as the*

*\* pruning set. (Default: 3)*

*\* <p>*

*\**

*\* -B <br>*

*\* Use binary splits for nominal attributes.*

*\* <p>*

*\**

*\* -S <br>*

*\* Don't perform subtree raising.*

*\* <p>*

*\**

*\* -L <br>*

*\* Do not clean up after the tree has been built.*

*\**

*\* -A <br>*

*\* If set, Laplace smoothing is used for predicted probabilites.*

*\* <p>*

*\**

*\* -Q <br>*

*\* The seed for reduced-error pruning.*

*\* <p>*

*\**

*\* @return an enumeration of all the available options.*

*\*/*

@Override

**public** Enumeration<Option> listOptions() {

Vector<Option> newVector = **new** Vector<Option>(13);

newVector.addElement(**new** Option("\tUse unpruned tree.", "U", 0, "-U"));

newVector.addElement(**new** Option("\tDo not collapse tree.", "O", 0, "-O"));

newVector.addElement(**new** Option("\tSet confidence threshold for pruning.\n"

+ "\t(default 0.25)", "C", 1, "-C <pruning confidence>"));

newVector.addElement(**new** Option(

"\tSet minimum number of instances per leaf.\n" + "\t(default 2)", "M",

1, "-M <minimum number of instances>"));

newVector.addElement(**new** Option("\tUse reduced error pruning.", "R", 0,

"-R"));

newVector.addElement(**new** Option("\tSet number of folds for reduced error\n"

+ "\tpruning. One fold is used as pruning set.\n" + "\t(default 3)", "N",

1, "-N <number of folds>"));

newVector.addElement(**new** Option("\tUse binary splits only.", "B", 0, "-B"));

newVector.addElement(**new** Option("\tDo not perform subtree raising.", "S", 0,

"-S"));

newVector.addElement(**new** Option(

"\tDo not clean up after the tree has been built.", "L", 0, "-L"));

newVector.addElement(**new** Option(

"\tLaplace smoothing for predicted probabilities.", "A", 0, "-A"));

newVector.addElement(**new** Option(

"\tDo not use MDL correction for info gain on numeric attributes.", "J",

0, "-J"));

newVector.addElement(**new** Option(

"\tSeed for random data shuffling (default 1).", "Q", 1, "-Q <seed>"));

newVector.addElement(**new** Option("\tDo not make split point actual value.",

"-doNotMakeSplitPointActualValue", 0, "-doNotMakeSplitPointActualValue"));

newVector.addAll(Collections.list(**super**.listOptions()));

**return** newVector.elements();

}

*/\*\**

*\* Parses a given list of options.*

*\**

*\* <!-- options-start --> Valid options are:*

*\* <p/>*

*\**

*\* <pre>*

*\* -U*

*\* Use unpruned tree.*

*\* </pre>*

*\**

*\* <pre>*

*\* -O*

*\* Do not collapse tree.*

*\* </pre>*

*\**

*\* <pre>*

*\* -C &lt;pruning confidence&gt;*

*\* Set confidence threshold for pruning.*

*\* (default 0.25)*

*\* </pre>*

*\**

*\* <pre>*

*\* -M &lt;minimum number of instances&gt;*

*\* Set minimum number of instances per leaf.*

*\* (default 2)*

*\* </pre>*

*\**

*\* <pre>*

*\* -R*

*\* Use reduced error pruning.*

*\* </pre>*

*\**

*\* <pre>*

*\* -N &lt;number of folds&gt;*

*\* Set number of folds for reduced error*

*\* pruning. One fold is used as pruning set.*

*\* (default 3)*

*\* </pre>*

*\**

*\* <pre>*

*\* -B*

*\* Use binary splits only.*

*\* </pre>*

*\**

*\* <pre>*

*\* -S*

*\* Don't perform subtree raising.*

*\* </pre>*

*\**

*\* <pre>*

*\* -L*

*\* Do not clean up after the tree has been built.*

*\* </pre>*

*\**

*\* <pre>*

*\* -A*

*\* Laplace smoothing for predicted probabilities.*

*\* </pre>*

*\**

*\* <pre>*

*\* -J*

*\* Do not use MDL correction for info gain on numeric attributes.*

*\* </pre>*

*\**

*\* <pre>*

*\* -Q &lt;seed&gt;*

*\* Seed for random data shuffling (default 1).*

*\* </pre>*

*\**

*\* <pre>*

*\* -doNotMakeSplitPointActualValue*

*\* Do not make split point actual value.*

*\* </pre>*

*\**

*\* <!-- options-end -->*

*\**

*\* @param options the list of options as an array of strings*

*\* @throws Exception if an option is not supported*

*\*/*

@Override

**public** void setOptions(String[] options) **throws** Exception {

*// Other options*

String minNumString = Utils.getOption('M', options);

**if** (minNumString.length() != 0) {

m\_minNumObj = Integer.parseInt(minNumString);

} **else** {

m\_minNumObj = 2;

}

m\_binarySplits = Utils.getFlag('B', options);

m\_useLaplace = Utils.getFlag('A', options);

m\_useMDLcorrection = !Utils.getFlag('J', options);

*// Pruning options*

m\_unpruned = Utils.getFlag('U', options);

m\_collapseTree = !Utils.getFlag('O', options);

m\_subtreeRaising = !Utils.getFlag('S', options);

m\_noCleanup = Utils.getFlag('L', options);

m\_doNotMakeSplitPointActualValue = Utils.getFlag(

"doNotMakeSplitPointActualValue", options);

**if** ((m\_unpruned) && (!m\_subtreeRaising)) {

**throw** **new** Exception(

"Subtree raising doesn't need to be unset for unpruned tree!");

}

m\_reducedErrorPruning = Utils.getFlag('R', options);

**if** ((m\_unpruned) && (m\_reducedErrorPruning)) {

**throw** **new** Exception(

"Unpruned tree and reduced error pruning can't be selected "

+ "simultaneously!");

}

String confidenceString = Utils.getOption('C', options);

**if** (confidenceString.length() != 0) {

**if** (m\_reducedErrorPruning) {

**throw** **new** Exception("Setting the confidence doesn't make sense "

+ "for reduced error pruning.");

} **else** **if** (m\_unpruned) {

**throw** **new** Exception(

"Doesn't make sense to change confidence for unpruned " + "tree!");

} **else** {

m\_CF = (**new** Float(confidenceString)).floatValue();

**if** ((m\_CF <= 0) || (m\_CF >= 1)) {

**throw** **new** Exception(

"Confidence has to be greater than zero and smaller " + "than one!");

}

}

} **else** {

m\_CF = 0.25f;

}

String numFoldsString = Utils.getOption('N', options);

**if** (numFoldsString.length() != 0) {

**if** (!m\_reducedErrorPruning) {

**throw** **new** Exception("Setting the number of folds"

+ " doesn't make sense if"

+ " reduced error pruning is not selected.");

} **else** {

m\_numFolds = Integer.parseInt(numFoldsString);

}

} **else** {

m\_numFolds = 3;

}

String seedString = Utils.getOption('Q', options);

**if** (seedString.length() != 0) {

m\_Seed = Integer.parseInt(seedString);

} **else** {

m\_Seed = 1;

}

**super**.setOptions(options);

Utils.checkForRemainingOptions(options);

}

*/\*\**

*\* Gets the current settings of the Classifier.*

*\**

*\* @return an array of strings suitable for passing to setOptions*

*\*/*

@Override

**public** String[] getOptions() {

Vector<String> options = **new** Vector<String>();

**if** (m\_noCleanup) {

options.add("-L");

}

**if** (!m\_collapseTree) {

options.add("-O");

}

**if** (m\_unpruned) {

options.add("-U");

} **else** {

**if** (!m\_subtreeRaising) {

options.add("-S");

}

**if** (m\_reducedErrorPruning) {

options.add("-R");

options.add("-N");

options.add("" + m\_numFolds);

options.add("-Q");

options.add("" + m\_Seed);

} **else** {

options.add("-C");

options.add("" + m\_CF);

}

}

**if** (m\_binarySplits) {

options.add("-B");

}

options.add("-M");

options.add("" + m\_minNumObj);

**if** (m\_useLaplace) {

options.add("-A");

}

**if** (!m\_useMDLcorrection) {

options.add("-J");

}

**if** (m\_doNotMakeSplitPointActualValue) {

options.add("-doNotMakeSplitPointActualValue");

}

Collections.addAll(options, **super**.getOptions());

**return** options.toArray(**new** String[0]);

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String seedTipText() {

**return** "The seed used for randomizing the data "

+ "when reduced-error pruning is used.";

}

*/\*\**

*\* Get the value of Seed.*

*\**

*\* @return Value of Seed.*

*\*/*

**public** int getSeed() {

**return** m\_Seed;

}

*/\*\**

*\* Set the value of Seed.*

*\**

*\* @param newSeed Value to assign to Seed.*

*\*/*

**public** void setSeed(int newSeed) {

m\_Seed = newSeed;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String useLaplaceTipText() {

**return** "Whether counts at leaves are smoothed based on Laplace.";

}

*/\*\**

*\* Get the value of useLaplace.*

*\**

*\* @return Value of useLaplace.*

*\*/*

**public** boolean getUseLaplace() {

**return** m\_useLaplace;

}

*/\*\**

*\* Set the value of useLaplace.*

*\**

*\* @param newuseLaplace Value to assign to useLaplace.*

*\*/*

**public** void setUseLaplace(boolean newuseLaplace) {

m\_useLaplace = newuseLaplace;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String useMDLcorrectionTipText() {

**return** "Whether MDL correction is used when finding splits on numeric attributes.";

}

*/\*\**

*\* Get the value of useMDLcorrection.*

*\**

*\* @return Value of useMDLcorrection.*

*\*/*

**public** boolean getUseMDLcorrection() {

**return** m\_useMDLcorrection;

}

*/\*\**

*\* Set the value of useMDLcorrection.*

*\**

*\* @param newuseMDLcorrection Value to assign to useMDLcorrection.*

*\*/*

**public** void setUseMDLcorrection(boolean newuseMDLcorrection) {

m\_useMDLcorrection = newuseMDLcorrection;

}

*/\*\**

*\* Returns a description of the classifier.*

*\**

*\* @return a description of the classifier*

*\*/*

@Override

**public** String toString() {

**if** (m\_root == **null**) {

**return** "No classifier built";

}

**if** (m\_unpruned) {

**return** "J48 unpruned tree\n------------------\n" + m\_root.toString();

} **else** {

**return** "J48 pruned tree\n------------------\n" + m\_root.toString();

}

}

*/\*\**

*\* Returns a superconcise version of the model*

*\**

*\* @return a summary of the model*

*\*/*

@Override

**public** String toSummaryString() {

**return** "Number of leaves: " + m\_root.numLeaves() + "\n"

+ "Size of the tree: " + m\_root.numNodes() + "\n";

}

*/\*\**

*\* Returns the size of the tree*

*\**

*\* @return the size of the tree*

*\*/*

**public** double measureTreeSize() {

**return** m\_root.numNodes();

}

*/\*\**

*\* Returns the number of leaves*

*\**

*\* @return the number of leaves*

*\*/*

**public** double measureNumLeaves() {

**return** m\_root.numLeaves();

}

*/\*\**

*\* Returns the number of rules (same as number of leaves)*

*\**

*\* @return the number of rules*

*\*/*

**public** double measureNumRules() {

**return** m\_root.numLeaves();

}

*/\*\**

*\* Returns an enumeration of the additional measure names*

*\**

*\* @return an enumeration of the measure names*

*\*/*

@Override

**public** Enumeration<String> enumerateMeasures() {

Vector<String> newVector = **new** Vector<String>(3);

newVector.addElement("measureTreeSize");

newVector.addElement("measureNumLeaves");

newVector.addElement("measureNumRules");

**return** newVector.elements();

}

*/\*\**

*\* Returns the value of the named measure*

*\**

*\* @param additionalMeasureName the name of the measure to query for its value*

*\* @return the value of the named measure*

*\* @throws IllegalArgumentException if the named measure is not supported*

*\*/*

@Override

**public** double getMeasure(String additionalMeasureName) {

**if** (additionalMeasureName.compareToIgnoreCase("measureNumRules") == 0) {

**return** measureNumRules();

} **else** **if** (additionalMeasureName.compareToIgnoreCase("measureTreeSize") == 0) {

**return** measureTreeSize();

} **else** **if** (additionalMeasureName.compareToIgnoreCase("measureNumLeaves") == 0) {

**return** measureNumLeaves();

} **else** {

**throw** **new** IllegalArgumentException(additionalMeasureName

+ " not supported (j48)");

}

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String unprunedTipText() {

**return** "Whether pruning is performed.";

}

*/\*\**

*\* Get the value of unpruned.*

*\**

*\* @return Value of unpruned.*

*\*/*

**public** boolean getUnpruned() {

**return** m\_unpruned;

}

*/\*\**

*\* Set the value of unpruned. Turns reduced-error pruning off if set.*

*\**

*\* @param v Value to assign to unpruned.*

*\*/*

**public** void setUnpruned(boolean v) {

**if** (v) {

m\_reducedErrorPruning = **false**;

}

m\_unpruned = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String collapseTreeTipText() {

**return** "Whether parts are removed that do not reduce training error.";

}

*/\*\**

*\* Get the value of collapseTree.*

*\**

*\* @return Value of collapseTree.*

*\*/*

**public** boolean getCollapseTree() {

**return** m\_collapseTree;

}

*/\*\**

*\* Set the value of collapseTree.*

*\**

*\* @param v Value to assign to collapseTree.*

*\*/*

**public** void setCollapseTree(boolean v) {

m\_collapseTree = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String confidenceFactorTipText() {

**return** "The confidence factor used for pruning (smaller values incur "

+ "more pruning).";

}

*/\*\**

*\* Get the value of CF.*

*\**

*\* @return Value of CF.*

*\*/*

**public** float getConfidenceFactor() {

**return** m\_CF;

}

*/\*\**

*\* Set the value of CF.*

*\**

*\* @param v Value to assign to CF.*

*\*/*

**public** void setConfidenceFactor(float v) {

m\_CF = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String minNumObjTipText() {

**return** "The minimum number of instances per leaf.";

}

*/\*\**

*\* Get the value of minNumObj.*

*\**

*\* @return Value of minNumObj.*

*\*/*

**public** int getMinNumObj() {

**return** m\_minNumObj;

}

*/\*\**

*\* Set the value of minNumObj.*

*\**

*\* @param v Value to assign to minNumObj.*

*\*/*

**public** void setMinNumObj(int v) {

m\_minNumObj = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String reducedErrorPruningTipText() {

**return** "Whether reduced-error pruning is used instead of C.4.5 pruning.";

}

*/\*\**

*\* Get the value of reducedErrorPruning.*

*\**

*\* @return Value of reducedErrorPruning.*

*\*/*

**public** boolean getReducedErrorPruning() {

**return** m\_reducedErrorPruning;

}

*/\*\**

*\* Set the value of reducedErrorPruning. Turns unpruned trees off if set.*

*\**

*\* @param v Value to assign to reducedErrorPruning.*

*\*/*

**public** void setReducedErrorPruning(boolean v) {

**if** (v) {

m\_unpruned = **false**;

}

m\_reducedErrorPruning = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String numFoldsTipText() {

**return** "Determines the amount of data used for reduced-error pruning. "

+ " One fold is used for pruning, the rest for growing the tree.";

}

*/\*\**

*\* Get the value of numFolds.*

*\**

*\* @return Value of numFolds.*

*\*/*

**public** int getNumFolds() {

**return** m\_numFolds;

}

*/\*\**

*\* Set the value of numFolds.*

*\**

*\* @param v Value to assign to numFolds.*

*\*/*

**public** void setNumFolds(int v) {

m\_numFolds = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String binarySplitsTipText() {

**return** "Whether to use binary splits on nominal attributes when "

+ "building the trees.";

}

*/\*\**

*\* Get the value of binarySplits.*

*\**

*\* @return Value of binarySplits.*

*\*/*

**public** boolean getBinarySplits() {

**return** m\_binarySplits;

}

*/\*\**

*\* Set the value of binarySplits.*

*\**

*\* @param v Value to assign to binarySplits.*

*\*/*

**public** void setBinarySplits(boolean v) {

m\_binarySplits = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String subtreeRaisingTipText() {

**return** "Whether to consider the subtree raising operation when pruning.";

}

*/\*\**

*\* Get the value of subtreeRaising.*

*\**

*\* @return Value of subtreeRaising.*

*\*/*

**public** boolean getSubtreeRaising() {

**return** m\_subtreeRaising;

}

*/\*\**

*\* Set the value of subtreeRaising.*

*\**

*\* @param v Value to assign to subtreeRaising.*

*\*/*

**public** void setSubtreeRaising(boolean v) {

m\_subtreeRaising = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String saveInstanceDataTipText() {

**return** "Whether to save the training data for visualization.";

}

*/\*\**

*\* Check whether instance data is to be saved.*

*\**

*\* @return true if instance data is saved*

*\*/*

**public** boolean getSaveInstanceData() {

**return** m\_noCleanup;

}

*/\*\**

*\* Set whether instance data is to be saved.*

*\**

*\* @param v true if instance data is to be saved*

*\*/*

**public** void setSaveInstanceData(boolean v) {

m\_noCleanup = v;

}

*/\*\**

*\* Returns the tip text for this property*

*\**

*\* @return tip text for this property suitable for displaying in the*

*\* explorer/experimenter gui*

*\*/*

**public** String doNotMakeSplitPointActualValueTipText() {

**return** "If true, the split point is not relocated to an actual data value."

+ " This can yield substantial speed-ups for large datasets with numeric attributes.";

}

*/\*\**

*\* Gets the value of doNotMakeSplitPointActualValue.*

*\**

*\* @return the value*

*\*/*

**public** boolean getDoNotMakeSplitPointActualValue() {

**return** m\_doNotMakeSplitPointActualValue;

}

*/\*\**

*\* Sets the value of doNotMakeSplitPointActualValue.*

*\**

*\* @param m\_doNotMakeSplitPointActualValue the value to set*

*\*/*

**public** void setDoNotMakeSplitPointActualValue(

boolean m\_doNotMakeSplitPointActualValue) {

**this**.m\_doNotMakeSplitPointActualValue = m\_doNotMakeSplitPointActualValue;

}

*/\*\**

*\* Returns the revision string.*

*\**

*\* @return the revision*

*\*/*

@Override

**public** String getRevision() {

**return** RevisionUtils.extract("$Revision$");

}

*/\*\**

*\* Builds the classifier to generate a partition.*

*\*/*

@Override

**public** void generatePartition(Instances data) **throws** Exception {

buildClassifier(data);

}

*/\*\**

*\* Computes an array that indicates node membership.*

*\*/*

@Override

**public** double[] getMembershipValues(Instance inst) **throws** Exception {

**return** m\_root.getMembershipValues(inst);

}

*/\*\**

*\* Returns the number of elements in the partition.*

*\*/*

@Override

**public** int numElements() **throws** Exception {

**return** m\_root.numNodes();

}

*/\*\**

*\* Main method for testing this class*

*\**

*\* @param argv the commandline options*

*\*/*

**public** **static** void main(String[] argv) {

runClassifier(**new** J48(), argv);

}

}