Extending Classifiers for Incomplete Data

A dissertation submitted in partial fulfillment of
the requirements for the degree of
BACHELOR OF ENGINEERING in Computer Science
in
The Queen's University of Belfast
by

Christopher McKee

April 7, 2017

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Student Number:

Project Title:

Supervisor:

Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.

Student's signature *Date of submission*

Abstract

Data used in the real world has a tendency to be incomplete, with values missing due to problems ranging from clerical error to data corruption. This thesis is concerned with investigating the effects of repeatedly filling in missing values in training data, and especially by doing so using probabilistic methods. The accuracy of using models trained this way is then compared with models trained against the original data set, including the missing values. The accuracy of the implemented method is generally worse than using models which were trained against data sets with missing data, and therefore does not appear useful.

Contents

1 Introduction **4**

2 Design and Implementation **6**

2.1 Design 6

2.2 Implementation 7

2.2.1 Weka Architecture 7

2.2.2 weka.core 7

2.2.3 AbstractClassifier 8

2.2.4 SingleClassifierEnhancer 9

2.2.5 IterativeClassifier 9

2.2.6 Weka Interfaces 10

2.2.7 StateAnalyser 10

2.2.8 ProjectClassifier 11

3 Experiments **13**

3.1 Results 14

3.1.1 Naive Bayes 14

3.1.2 Bayesian Network 20

3.1.3 J48 Decision Tree 25

4 Conclusion **30**

A Experimental Data Sets **32**

B Hidden Variables **33**

C User Guide **34**

D JavaDoc **35**

List of Tables

3.1 Naive Bayes Percent Incorrect 16

3.2 Naive Bayes Mean Absolute Error 17

3.3 Naive Bayes Root Mean Squared Error 18

3.4 Naive Bayes Mean Entropy Gain 19

3.5 Bayesian Network Percent Incorrect 21

3.6 Bayesian Network Mean Absolute Error 22

3.7 Bayesian Network Root Mean Squared Error 23

3.8 Bayesian Network mean Entropy Gain 24

3.9 J48 Decision Tree Percent Incorrect 26

3.10 J48 Decision Tree Mean Absolute Error 27

3.11 J48 Root Mean Squared Error 28

3.12 J48 Decision Tree Mean Entropy Gain 29

Chapter 1

Introduction

The reliability of a machine learning model is heavily dependent on the data which was used to train it. Training data sets with missing values are therefore an interesting problem since there is no way of knowing what any given missing value could have been, and so the only two useful approaches involve either making statistical approximations of missing values or ignoring any record which contains missing data. Both mentioned approaches have positives as well as negatives. Ignoring a record prevents you from using unreliable data, but also may lead to valuable data in the other, non-missing attributes of the same record being discounted. Attempts to predict a missing value have no guarantee of being correct, but may be worth the risk to gain value from the other complete attributes. Using either approach may produce false outcomes. It is therefore extremely important that any imputation is carried out with care, and it must be very likely to be correct.

This project aims to investigate the usage of machine learning techniques on incomplete sets of discrete data. It will attempt to impute any missing values in the training data, by iteratively building and applying models until no further change is observed. It is expected that predicted missing values will trend towards the mean, leading to more reliable classification when this trained model is then applied to target data. Given sufficient time, it is also intended that this approach can be used to infer 'hidden variables', similarly to how neural networks produce hidden layers to create relationships between data. This will be achieved by adding a new attribute to a data set, and initially filling it full of random valid data. We will then repeatedly build, apply and rebuild models to classify this attribute until it stops changing, in an identical manner to how missing data is imputed in the training set. New data will be added to the training data set by this method, and this project aims to determine whether or not these hidden variables lead to a greater classification accuracy. All implemented code for this project, as well as some additional utility material, can be found at [1].

The goal of this project is to study the effects of iteratively building common classifiers and comparing the performance of classifiers built this way against the performance of the same classifier, but without attempting to impute missing training data. The main aims of the software to be developed will therefore be:

- A Weka plugin will be developed. This should integrate smoothly into any version of Weka >3.7.1, and will be made freely available.
- The classifier developed within the plugin should be usable from both the CLI and GUI.

- Running experiments using the plugin should be easy, and have well explained documentation.
- Allow any Weka classifier which is valid to use for a given dataset to be used within the newly implemented classifier, and for this to be easily configurable.
- Run a number of experiments to compare the performance of the iteratively trained classifier against some control cases on the same data sets.

Chapter 2

Design and Implementation

2.1 Design

This body of work is mainly interested in popular classification algorithms, and in trying to use them in a slightly unusual way. It therefore makes sense to extend a package in which these are already implemented, and in which they are easily accessible. The Weka project [2] suits this purpose since it has well maintained, efficient implementations of most machine learning algorithms, handles file I/O, and has both command line and graphical user interfaces. Since Weka is written in Java, it follows that the plugin should also be written in Java.

It also follows that any files which are used for testing should be in either .csv, or .arff format. The former is commonly used in data processing while the latter is a proprietary Weka format which is very similar, but differs in that it contains additional data at the top of the file pertaining to the dataset. Both are supported by Weka, and therefore either is usable.

Since the project relies upon the idea that predictions for missing training data values should converge, it is assumed that the developed classifier will initially only work on nominal data sets. This is because nominal data sets should eventually settle on concrete values (at least when imputed using probabilistic techniques), whilst numeric values are not guaranteed to every settle on a particular value at all. It is more likely that numeric predicitions would just change less and less between iterations, until the changes were miniscule. While it would be possible to implement a solution which stops iterating once a numeric data set only changes to a certain degree, as a proof of concept it is sensible to stick to nominal data.

It is intended that the implemented algorithm will work as follows:

1. Take a given training set, and make a copy of it
2. Fill the copy set with random, viable values
3. Use the produced data set to train a new model
4. Use the newly trained model to impute values into a new copy of the training set
5. Repeat 3 and 4 until identical data sets are produced by two consecutive iterations
6. Use the model built by the final iteration for classification of a test set

2.2 Implementation

As an existing piece of software is being extended, much of the work to implement the various classifiers to be tested is already done. The main aim of this work is to integrate successfully into Weka as a plugin, and to interface with it as necessary. This plugin will be provided as a zip file containing the implemented algorithm which can be loaded by Weka using its package manager.

2.2.1 Weka Architecture

Weka is a Java package, with a source code layout which looks somewhat like this if unused packages are omitted:

```
weka
├── associations
├── attributeSelection
├── classifiers
│   ├── AbstractClassifier
│   ├── IterativeClassifier
│   └── SingleClassifierEnhancer
├── clusterers
├── core
│   ├── Attribute
│   ├── Capabilities
│   ├── Instance
│   ├── Instances
│   ├── Option
│   └── Utils
├── datagenerators
├── estimators
├── experiment
├── filters
├── gui
└── knowledgeflow
```

Each of these subpackages is reasonably self explanatory. This project is mostly concerned with classifications, and thus code implemented for it lives in the *weka.classifiers* package, although there are some dependcies on *weka.core* since it contains much of the shared code for Weka. Provided that the rest of the project works as intended and the documentation is followed, there is no need to modify or discuss the other packages.

2.2.2 weka.core

From *weka.core*, the following classes are used:

- **Attribute** - An Attribute in Weka refers to a single column in a data set. This object is

mostly responsible for tracking the column's data type (e.g. nominal, numeric, etc), and other related information common to the entire column.

- **Instance** - An Instance is the representation of a single row of data, roughly analogous to a row in a CSV file. Each Instance belongs to a particular Instances object. Each attribute value is stored as a Java double, which is then used in connection with an Attribute object to determine the actual value of an attribute.
- **Instances** - An Instances object in Weka is used to represent a complete data set. It is roughly equivalent to the object representation of an ARFF file. It is comprised of a list of Instance objects, a list of Attribute objects and some other metadata such as the class attribute for the data set and its name.
- **Option** - An Option object represents a particular command line parameter, and the way in which it should be handled. This also contains a description to be printed in the help dialog if incorrect parameters are passed to the CLI.

2.2.3 AbstractClassifier

AbstractClassifier is the class from which all classifiers which make nominal or numeric predictions in Weka are derived from. It provides a number of helpful implementations to prevent code duplication in classifiers, and to ease implementation. A pseudocode description of some notable methods can be seen below:

```
abstract class AbstractClassifier {
    forName(classifierName, options) {
        return an instance of classifier <classifierName> with options <option>
    }

    runClassifier(classifier, options) {
        run any classifier by passing options to it
    }

    classifyInstance(instance) {
        returns the class which the instance is most
            likely to belong to as a double in Weka's internal representation
    }

    distributionForInstance(instance) {
        return a list of probabilities for each
            possibile class value for a given instance
    }

    listOptions() {
        return all of the command line options which
            a classifier accepts to configure how it will run
    }

    getOptions() {
        return the list of options which a classifier
            is currently set up to run with
    }

    setOptions(options) {
```

```
        sets the command line options for a
            classifier to options
    }

    getCapabilities() {
        returns a list of the attribute and class attribute
            types which are supported by this classifier
    }
}
```

There are a number of other utility and stub methods within this abstract class which may prove useful for other people in other cases, but have been omitted from the description above.

2.2.4 SingleClassifierEnhancer

SingleClassifierEnhancer is a concrete class which extends from *AbstractClassifier*, and from which the implemented ProjectClassifier extends. Extending from this class gives us a couple of benefits:

- *SingleClassifierEnhancer* is designed for 'meta' classifiers, or those which wrap another classifier. It already contains an internal classifier and handles passing options to it. This can easily be extended to copy this classifier multiple times with the same options.
- The internal classifier is already displayed nicely in the Weka GUI. This allows it to be configured via a drop down menu, as shown in figure [[put screenshot here?]].
- It already handles some edge cases, such as calling the *preExecution()* and *postExecution()* methods of the internal classifier, which handles any specific setup and cleanup needed. These are small edge cases that would be easy to forget about, so extending in this way improves reliability by removing some of the mental overhead of the implementation.

2.2.5 IterativeClassifier

IterativeClassifier is an Interface which is provided by Weka for 'classifiers which build models of increasing complexity' [[Javadoc citation?]]. It provides three method signatures:

```
public initializeClassifier() {
    do setup for the classifier
}

public next() {
    increase in complexity by one step by retraining the model
    return false if finished, or true if another step can be taken
}

public done() {
    stop increasing complexity and do any necessary cleanup
}
```

These methods generally come together in the following pattern in a classifier which implements the interface:

```
public buildClassifier(instances) {
    // some code here
    initializeClassifier(instances)
    while(next()) {}
    done()
    // some other code here
}
```

This means that building the classifier continues until it no longer makes sense to keep doing so, or no gain is achieved. This suits the project's use case, and therefore the interface makes sense to use.

2.2.6 Weka Interfaces

Calling a Weka classifier from the command line (assuming that the Weka jar file is already on the classpath) is reasonably straightforward. The target classifier is firstly called by its full path e.g. *java weka.classifiers.trees.J48*. Anything which comes after this point is passed to the classifier as an Option object. These are used as parameters to configure the behaviour of a particular classifier, and each class which extends from *AbstractClassifier* generally implements its own specific set of Options through a combination of private variables and specifically named methods. The GUI appears works in much the same way, except that it provides windows with dropdown menus and text fields to allow classifiers and their Options to be configured.

For example, imagine a given classifier had a private variable named *m_Example*. In order to expose this to the GUI and CLI it would require getter and setter methods, as well as an *exampleTipText()* method to display help text. Code specifying how to handle this Option would also need to be added to the *getOptions()*, *setOptions()* and *listOptions()* methods.

2.2.7 StateAnalyser

The *StateAnalyser* class has been designed to track the progress of the *ProjectClassifier* as it is being trained. It contains a list of the training sets which have been produced by the *next()* method, and records the number of differences between them. All of the methods described below are just utility methods for working with or comparing the training sets which have been produced.

```
private convertToMatrix(instances) {
    create a matrix
    for each row in the data set {
        turn it into an array of floats
        add that array to the matrix
    }
    return matrix
}

public getNumberDifferences() {
    if at least two training sets recorded {
        convert the two most recent training sets to matrixes
        for each row in both matrices {
            if the two rows differ, increment the difference counter
```

```
        }

        return the number of different rows;
    }

    else
        return a negative number to indicate an invalid test;
}


public getNumberIterations() {
    return number of recorded sets
}
```

2.2.8 ProjectClassifier

The *ProjetClassifier* class is the implementation of the algorithm described in section 2.1. This extends from *SingleClassifierEnhancer* and implements the *IterativeClassifier* interface. It maintains an internal array of classifiers, one for each of the attributes in a data set, and uses these to impute values into the training set as well as to classify test data. It also maintains three copies of training set while building:

- *original* - a copy of the training set
- *last* - the last data set produced by imputing missing values, which can be used for retraining during the next iteration
- *current* - the data set which is currently having its missing values filled in

```
public buildClassifier(instances) {
    record the index of the original class attribute
    take a reference copy of the training set (original)
    find the location of missing data in the training set
    set up the classifier for the first run
    repeat until no change is observed between current and last
    perform any cleanup needed
}

public findMissingAttributes(instances) {
    for each instance in a training set {
        record any values which are missing for later use
    }
}

public initializeClassifier(instances) {
    create new tracker object
    create one classifier object per attribute in the data set
    take two modifiable copies of the training data set (last/current)
    replace any missing data in last using random values
}

public replaceMissingValues(instances) {
    for any each piece of missing data {
        input a random possible value
    }
```

```
}

public next() {
    replace last with current
    replace current with a new copy of the training data
    retrain classifiers against last
    for each recorded missing value {
        use trained classifiers to impute missing values in current
    }
    add current to the tracker object
    if (there are no differences between current and last)
        stop iterating
    else
        iterate again
}

public done() {
    do option specific cleanup
}

public retrainClassifiers(instances) {
    for each attribute in the training set {
        retrain a classifier against that attribute
    }
}
```

ProjetClassifier also accepts some options, which can be used to configure how it runs:

- **maxIterations** (-M number) - Some classifiers may not converge in the way in which we expect, or they make take a very long time to reach this point. This flag allows for a maximum number of training iterations to be set in order to ensure experiments finish in a timely fashion.
- **supervised** (-S) - It may be interesting to observe the difference between classifiers which are trained against a training set where the class attribute is trained iteratively along with the rest of the data set against a training set where missing class attribute values are not imputed until the classifier rest of the classifier has converged. This flag allows for the latter case to be tested.
- **randomData** (-R) - As a control set during experiments, it make sense to fill a training set with random data and determine whether or not the implemented algorithm outperforms classifiers which are trained against this. This allows this experiment to be performed.
- **hiddenVariables** (-N) - See appendix B.

Chapter 3

Experiments

The algorithm described in the previous section will be used in experiments with a number of internal classifiers in order to assess their performance when trained iteratively. These results will be compared with the results of using the internal classifier in a number of control cases, to determine if the implemented algorithm described has proven beneficial. Each case will be tested against 51 different sets of nominal data taken from the UCI Machine Learning Repository [3], each of which has had approximately 10% of its values removed at random in order to simulate the effects of having missing data. For each data set, 5 folds will be used for cross validation and this will be repeated 10 times. Additional information about the data sets can be found in appendix A.

For each classifier which we are testing we will use the following test cases, where *TestClassifier* represents the current classifier under test:

1. *ProjectClassifier -S with TestClassifier* - Iteratively trains the target classifier by imputing missing values, retraining against the newly completed data set and then imputing again until convergence or max iterations reached. The class attribute is not guessed.
2. *ProjectClassifier with TestClassifier* - Same as previous case, except that the initial class attribute is also iteratively imputed.
3. *TestClassifier* - 'Vanilla' classifier, run without attempting to remove missing values before classification.
4. *TestClassifier (missing data filled in with mode)* - Same as previous case, except every missing value will be replaced using the mode before it is run.
5. *ProjectClassifier -R with TestClassifier* - Similar to previous case, except missing values will be replaced using random possible values before classification. Only one iteration will be performed.

Each of these tests will henceforth be referred to as Case 1, 2, 3, 4 and 5 respectively. For each of the tables discussed below, the index of each column refers to the number of each case.

Some adaptations may need to be made for certain classifiers, but this is the general structure of each experiment. For each of these classifiers the percentage of incorrect classifications, mean absolute error (MAE), root mean squared error (RMSE), and mean entropy gain (MEG)

will be compared using a paired T test with a statistical significance threshold of 0.05. It is generally expected that Case 1, the iteratively trained classifier, will perform better than all other cases with the possible exception of Case 3.

3.1 Results

Tables referenced below can be found at the end of each subsection, since they are too large to be included in-line.

3.1.1 Naive Bayes

No adaptations needed to be made for the Naive Bayes classification method since it converged to a stable model as expected, and did so within a reasonable period of time.

Naive Bayes Percent Incorrect results are shown in Table 3.1. As expected, Case 1 generally performed better than Cases 2, 4 and 5. It seems logical that Cases 4 and 5 would perform the worst, since these methods generally just involved filling in missing data with the most common and random data respectively. Case 1 generally performed worse than Case 3, producing statistically even results in 30 tests and statistically worse results in 19 tests.

Naive Bayes MAE results are shown in Table 3.2. Cases 1, 2 and 3 proved approximately equal according to this test, with no real significant difference between results. Case 1 performed much better than Cases 4 or 5, showing better results in 37 and 45 tests respectively. This is an expected result, since Cases 4 and 5 are effectively control cases.

Naive Bayes RMSE results are shown in Table 3.3. Using this metric for measurement, Case 1 performed significantly better than both Cases 2 and 4, with better results shown in 41 and 28 data sets respectively. Interestingly, it performed about as well as Case 5, performing significantly better in 17 tests, approximately about as well in 16 tests, and significantly worse in 18 tests. This could be due to the fact that using random data is unlikely to lead to strong predictions being made, while iteratively imputing missing values will tend to result in either increasingly bad or increasingly good predicitons being made. Again, Case 3 performed significantly better than Case 1.

Naive Bayes MEG results are shown in Table 3.4. Case 1 performed significanty worse than Case 2 according to this metric, performing worse in 42/51 data sets. It is assumed that this is due to the iterative imputation of the class attribute in Case 2, which probably leads to lower entropy gains since it will produce a smaller range of values. For a very similar reason, Case 1 performs significantly better than Case 3. Since it's iteratively imputing all but one attribute in the training set, a smaller range of values are probably produced during classification. Cases 1, 4 and 5 perform about as well as each other, since Cases 4 and 5 are essentially just making random guesses.

It would seem that the implemented algorithm does not provide any significant gains when using the Naive Bayes classifier according to the measurements used. While signifcant decreases in entropy gain were observed, it was both less accurate as a classifier and more strongly wrong.

Table 3.1: Naive Bayes Percent Incorrect

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	36.33	38.44 ○	37.28 ○	50.09 ○	43.50 ○
autos	10.33	9.78 ●	10.22	9.62 ●	9.61 ●
balance-scale	26.54	25.96 ●	26.26 ●	28.49 ○	25.98 ●
breast-cancer	27.67	27.63	27.14	27.41	27.33
bridges-version1	35.44	33.67 ●	35.45	36.00	38.04 ○
car	19.26	19.67 ○	19.46	23.23 ○	21.39 ○
cmc	51.80	52.27 ○	50.69 ●	50.24 ●	51.24 ●
colic	21.45	21.94	20.92 ●	19.94 ●	21.72
cylinder-bands	32.26	33.12 ○	31.79 ●	32.69	32.92
dermatology	3.19	3.04	2.95	3.58	3.31
diabetes	28.12	28.90 ○	27.79	27.52	28.41
ecoli	33.61	33.52	32.67 ●	34.53 ○	32.44 ●
flags	37.83	37.60	38.46	40.51 ○	39.77 ○
glass	45.98	44.97	46.42	46.58	46.02
haberman	28.30	28.12	27.71	27.27 ●	27.31 ●
hayes-roth-train	51.28	52.26	47.16 ●	49.23	50.26
heart-h	17.98	18.05	18.39	19.23 ○	18.24
heart-statlog	27.15	27.15	27.10	27.51	27.03
hepatitis	18.98	19.05	19.06	19.20	19.35
hypothyroid	9.14	12.42 ○	8.74 ●	8.38 ●	7.76 ●
ionosphere	27.11	30.89 ○	24.93 ●	23.94 ●	25.97 ●
iris	21.29	21.59	21.51	23.54 ○	22.26
kr-vs-kp	14.69	16.55 ○	14.01 ●	16.69 ○	16.31 ○
labor	5.40	5.00	4.80	5.20	7.00 ○
letter	60.44	61.98 ○	59.84 ●	62.59 ○	60.33 ●
liver-disorders	38.00	39.84 ○	37.31	39.17	37.99
lung-cancer	49.89	47.90	53.02	45.55	53.19
lymph	16.60	16.30	15.65 ●	16.49	16.65
molecular-biology-promoters	9.73	10.18	8.56	14.00 ○	10.39
mushroom	5.06	6.86 ○	5.06	7.89 ○	8.06 ○
nursery	16.01	16.08	15.87	16.51 ○	15.63 ●
optdigits	13.60	14.06 ○	13.10 ●	16.32 ○	14.55 ○
page-blocks	16.82	19.30 ○	16.82	14.00 ●	9.17 ●
pendigits	21.97	22.49 ○	21.21 ●	26.14 ○	23.08 ○
postoperative-patient-data	30.69	31.41	29.98	30.36	33.26 ○
primary-tumor	54.72	55.26	54.13	54.94	56.04 ○
segment	28.72	30.80 ○	28.12 ●	33.04 ○	31.49 ○
shuttle-landing-control	44.00	42.67	46.67	36.67 ●	45.33
sick	9.64	10.26 ○	9.81 ○	8.84 ●	7.98 ●
solar-flare-2	28.23	28.22	28.16	28.39	28.21
sonar	24.86	26.31 ○	24.14 ●	25.33	24.60
soybean	9.62	10.51 ○	9.77	11.28 ○	11.91 ○
spambase	24.61	24.85 ○	24.51 ●	24.14 ●	24.57
tae	49.72	50.00	48.92	51.64 ○	51.37 ○
tic-tac-toe	30.03	30.21	29.49 ●	29.23 ●	29.33 ●
trains	33.00	33.00	31.00	39.00	33.00
vehicle	55.02	56.60 ○	53.71 ●	55.79 ○	54.53
vote	9.47	9.57 ○	9.47	9.34	9.64
vowel	70.35	71.30 ○	69.27 ●	72.30 ○	69.72
waveform-5000	22.34	22.65 ○	22.44	23.63 ○	23.04 ○
zoo	8.99	9.86 ○	8.78	11.33 ○	11.16 ○
Average	27.71	28.24	27.37	28.52	28.18

○, ● statistically significant improvement or degradation

Table 3.2: Naive Bayes Mean Absolute Error

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	0.03	0.03 ○	0.03 ○	0.04 ○	0.04 ○
autos	0.10	0.10 ●	0.10 ●	0.10	0.10
balance-scale	0.25	0.25 ●	0.27 ○	0.28 ○	0.29 ○
breast-cancer	0.33	0.32 ●	0.32	0.32	0.34 ○
bridges-version1	0.15	0.14 ●	0.15 ○	0.15 ○	0.16 ○
car	0.13	0.13 ●	0.13 ○	0.14 ○	0.17 ○
cmc	0.38	0.38 ●	0.38 ○	0.39 ○	0.39 ○
colic	0.23	0.24 ○	0.23 ●	0.22 ●	0.25 ○
cylinder-bands	0.35	0.35 ○	0.34	0.35	0.37 ○
dermatology	0.01	0.01 ●	0.01 ●	0.02 ○	0.02 ○
diabetes	0.33	0.33 ○	0.33 ○	0.33 ○	0.35 ○
ecoli	0.10	0.10 ●	0.10 ○	0.11 ○	0.11 ○
flags	0.10	0.10	0.10 ○	0.11 ○	0.11 ○
glass	0.16	0.16 ●	0.16 ○	0.16 ○	0.17 ○
haberman	0.37	0.36 ●	0.37	0.37 ●	0.39 ○
hayes-roth-train	0.31	0.31	0.31	0.31 ○	0.32 ○
heart-h	0.09	0.09 ●	0.09 ○	0.10 ○	0.11 ○
heart-statlog	0.30	0.29	0.30 ○	0.31 ○	0.31 ○
hepatitis	0.21	0.21	0.21 ○	0.21 ○	0.23 ○
hypothyroid	0.07	0.08 ○	0.07 ●	0.07 ●	0.10 ○
ionosphere	0.28	0.31 ○	0.26 ●	0.26 ●	0.28
iris	0.18	0.17 ●	0.18	0.21 ○	0.20 ○
kr-vs-kp	0.23	0.24 ○	0.23 ●	0.25 ○	0.29 ○
labor	0.08	0.08	0.08	0.09 ○	0.11 ○
letter	0.05	0.05 ●	0.05 ●	0.06 ○	0.06 ○
liver-disorders	0.46	0.46	0.46	0.46	0.47 ○
lung-cancer	0.34	0.33	0.36 ○	0.31 ●	0.35
lymph	0.10	0.10 ○	0.10	0.11 ○	0.11 ○
molecular-biology-promoters	0.11	0.12 ○	0.11	0.15 ○	0.13 ○
mushroom	0.05	0.07 ○	0.05 ●	0.08 ○	0.08 ○
nursery	0.09	0.09 ●	0.10 ○	0.13 ○	0.14 ○
optdigits	0.03	0.03 ○	0.03 ●	0.04 ○	0.04 ○
page-blocks	0.08	0.09 ○	0.08 ●	0.07 ●	0.10 ○
pendigits	0.05	0.05 ●	0.05 ●	0.07 ○	0.06 ○
postoperative-patient-data	0.28	0.28 ○	0.28 ○	0.27 ●	0.30 ○
primary-tumor	0.06	0.06 ●	0.06	0.06 ○	0.06 ○
segment	0.09	0.10 ○	0.09 ●	0.11 ○	0.11 ○
shuttle-landing-control	0.46	0.45	0.46	0.44 ●	0.47 ○
sick	0.12	0.12 ○	0.12 ○	0.11 ●	0.16 ○
solar-flare-2	0.11	0.11 ●	0.11 ○	0.12 ○	0.13 ○
sonar	0.25	0.26 ○	0.25 ●	0.26 ○	0.25 ●
soybean	0.01	0.01 ○	0.01	0.01 ○	0.02 ○
spambase	0.27	0.27 ○	0.27 ○	0.28 ○	0.29 ○
tae	0.40	0.40	0.40 ○	0.41 ○	0.41 ○
tic-tac-toe	0.36	0.36 ●	0.37 ○	0.37 ○	0.39 ○
trains	0.32	0.35	0.33	0.35	0.33
vehicle	0.29	0.29 ○	0.28 ●	0.29 ○	0.29
vote	0.10	0.10 ○	0.10 ●	0.10 ○	0.10 ○
vowel	0.14	0.14 ○	0.14 ●	0.15 ○	0.14 ○
waveform-5000	0.16	0.16 ○	0.16 ●	0.17 ○	0.17 ○
zoo	0.03	0.03	0.03	0.04 ○	0.04 ○
Average	0.19	0.19	0.19	0.20	0.20

○, ● statistically significant improvement or degradation

Table 3.3: Naive Bayes Root Mean Squared Error

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	0.16	0.16 ○	0.16 ○	0.17 ○	0.16 ○
autos	0.29	0.28 ●	0.28 ●	0.27 ●	0.28 ●
balance-scale	0.34	0.34 ○	0.34 ○	0.36 ○	0.35 ○
breast-cancer	0.45	0.45 ○	0.45 ●	0.44 ●	0.44 ●
bridges-version1	0.28	0.29	0.28	0.29 ○	0.29 ○
car	0.24	0.24 ○	0.25 ○	0.26 ○	0.27 ○
cmc	0.46	0.47 ○	0.46 ●	0.45 ●	0.45 ●
colic	0.42	0.43 ○	0.41 ●	0.41 ●	0.41 ●
cylinder-bands	0.47	0.49 ○	0.47 ●	0.47 ●	0.46 ●
dermatology	0.08	0.08	0.08 ●	0.09 ○	0.09 ○
diabetes	0.43	0.44 ○	0.43 ●	0.42 ●	0.42 ●
ecoli	0.23	0.23 ○	0.23 ●	0.23 ○	0.23
flags	0.26	0.27 ○	0.26	0.27 ○	0.27
glass	0.29	0.29 ○	0.29 ●	0.29	0.29
haberman	0.44	0.44	0.44 ●	0.44 ○	0.44
hayes-roth-train	0.39	0.39 ○	0.38 ●	0.39 ○	0.39 ○
heart-h	0.23	0.23 ○	0.23 ●	0.23 ○	0.23
heart-statlog	0.43	0.44 ○	0.43 ●	0.42 ●	0.42 ●
hepatitis	0.39	0.39 ○	0.39	0.39	0.38 ●
hypothyroid	0.19	0.22 ○	0.18 ●	0.18 ●	0.19 ○
ionosphere	0.45	0.50 ○	0.44 ●	0.43 ●	0.44 ●
iris	0.32	0.32	0.32	0.34 ○	0.32 ○
kr-vs-kp	0.32	0.34 ○	0.32 ●	0.34 ○	0.35 ○
labor	0.18	0.18	0.17 ●	0.19	0.20 ○
letter	0.17	0.17 ○	0.17 ●	0.17 ○	0.17 ●
liver-disorders	0.48	0.49 ○	0.48 ●	0.48 ○	0.48
lung-cancer	0.51	0.50	0.52	0.48 ●	0.52
lymph	0.24	0.24 ○	0.23 ●	0.24	0.24
molecular-biology-promoters	0.24	0.26 ○	0.24	0.31 ○	0.26 ○
mushroom	0.20	0.25 ○	0.20 ●	0.26 ○	0.26 ○
nursery	0.21	0.21 ○	0.21 ○	0.23 ○	0.23 ○
optdigits	0.15	0.15 ○	0.14 ●	0.16 ○	0.15
page-blocks	0.22	0.24 ○	0.21 ●	0.20 ●	0.19 ●
pendigits	0.18	0.18 ○	0.18 ●	0.20 ○	0.18 ○
postoperative-patient-data	0.39	0.40 ○	0.39	0.39	0.40
primary-tumor	0.18	0.18 ○	0.18 ●	0.18 ○	0.18 ○
segment	0.25	0.25 ○	0.24 ●	0.26 ○	0.25
shuttle-landing-control	0.49	0.49	0.49	0.48 ●	0.50
sick	0.26	0.27 ○	0.26 ●	0.25 ●	0.25 ●
solar-flare-2	0.24	0.25 ○	0.24 ●	0.25 ○	0.25
sonar	0.45	0.46 ○	0.44 ●	0.45 ○	0.43 ●
soybean	0.09	0.10 ○	0.09	0.10 ○	0.10 ○
spambase	0.43	0.43 ○	0.42 ●	0.42 ●	0.41 ●
tae	0.46	0.46 ○	0.46 ●	0.46	0.46
tic-tac-toe	0.44	0.44 ○	0.43 ●	0.44 ○	0.44
trains	0.40	0.42	0.41	0.43	0.42
vehicle	0.45	0.46 ○	0.44 ●	0.44 ●	0.43 ●
vote	0.29	0.30 ○	0.29 ●	0.29 ●	0.29 ●
vowel	0.27	0.28 ○	0.27 ●	0.28 ○	0.27 ●
waveform-5000	0.35	0.35 ○	0.35 ●	0.35 ○	0.35 ●
zoo	0.13	0.13	0.12	0.14 ○	0.14 ○
Average	0.31	0.32	0.31	0.32	0.31

○, ● statistically significant improvement or degradation

Table 3.4: Naive Bayes Mean Entropy Gain

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	-1.07	-1.31 •	-1.30 •	-2.14 •	-0.49 ◦
autos	-0.10	-0.12 •	-0.04 ◦	0.11 ◦	0.17 ◦
balance-scale	0.42	0.41 •	0.39 •	0.32 •	0.35 •
breast-cancer	-0.04	-0.07 •	-0.02 ◦	-0.00 ◦	0.03 ◦
bridges-version1	0.73	0.68 •	0.75 ◦	0.67 •	0.72
car	0.65	0.65 •	0.63 •	0.57 •	0.45 •
cmc	0.01	-0.08 •	0.04 ◦	0.04 ◦	0.08 ◦
colic	-0.32	-0.43 •	-0.22 ◦	-0.17 ◦	0.01 ◦
cylinder-bands	-0.02	-0.16 •	0.02 ◦	0.04 ◦	0.07 ◦
dermatology	2.32	2.32	2.33 ◦	2.29 •	2.30 •
diabetes	0.14	0.12 •	0.16 ◦	0.16 ◦	0.17 ◦
ecoli	0.98	0.96 •	1.00 ◦	0.96	0.95 •
flags	0.27	0.12 •	0.31 ◦	0.27	0.45 ◦
glass	0.53	0.48 •	0.55 ◦	0.53	0.50
haberman	-0.00	-0.01	0.01 ◦	-0.01 •	0.01 ◦
hayes-roth-train	0.13	0.08 •	0.14 ◦	0.10 •	0.07 •
heart-h	0.27	0.25 •	0.31 ◦	0.30 ◦	0.27
heart-statlog	0.17	0.14 •	0.20 ◦	0.22 ◦	0.23 ◦
hepatitis	-0.16	-0.21 •	-0.14 ◦	-0.09 ◦	0.03 ◦
hypothyroid	0.09	-0.12 •	0.10 ◦	0.09 ◦	0.01 •
ionosphere	-0.26	-0.65 •	-0.18 ◦	-0.08 ◦	-0.08 ◦
iris	0.89	0.89	0.89	0.76 •	0.87 •
kr-vs-kp	0.52	0.49 •	0.53 ◦	0.48 •	0.43 •
labor	0.68	0.71 ◦	0.71 ◦	0.70	0.70
letter	1.69	1.50 •	1.75 ◦	1.58 •	1.76 ◦
liver-disorders	0.05	0.01 •	0.06 ◦	0.03 •	0.05
lung-cancer	-1.45	-1.58	-1.40	-1.21 ◦	-1.12 ◦
lymph	0.66	0.63 •	0.70 ◦	0.72 ◦	0.67
molecular-biology-promoters	0.63	0.59	0.66 ◦	0.43 •	0.61
mushroom	0.71	0.51 •	0.76 ◦	0.57 •	0.61 •
nursery	1.20	1.20 •	1.19 •	1.00 •	0.98 •
optdigits	2.41	2.36 •	2.46 ◦	2.37 •	2.57 ◦
page-blocks	0.05	-0.11 •	0.07 ◦	0.11 ◦	0.09 ◦
pendigits	2.10	2.06 •	2.13 ◦	2.00 •	2.19 ◦
postoperative-patient-data	-0.16	-0.20 •	-0.14 ◦	-0.17	-0.12
primary-tumor	0.79	0.74 •	0.84 ◦	0.71 •	0.82 ◦
segment	1.27	1.16 •	1.32 ◦	1.25 •	1.41 ◦
shuttle-landing-control	0.00	0.01	0.00	0.04 ◦	-0.03
sick	0.02	-0.01 •	0.02 ◦	0.04 ◦	0.02 ◦
solar-flare-2	1.38	1.35 •	1.40 ◦	1.28 •	1.34 •
sonar	-0.43	-0.59 •	-0.37 ◦	-0.33 ◦	-0.16 ◦
soybean	3.23	3.14 •	3.25 ◦	3.23	3.21 •
spambase	0.11	0.06 •	0.14 ◦	0.17 ◦	0.22 ◦
tae	0.02	0.01 •	0.05 ◦	0.03	0.06 ◦
tic-tac-toe	0.13	0.13 •	0.14 ◦	0.12 •	0.13 •
trains	0.08	-0.14	0.07	0.01	-0.00
vehicle	-1.15	-1.54 •	-1.00 ◦	-0.68 ◦	-0.60 ◦
vote	0.14	0.08 •	0.18 ◦	0.28 ◦	0.34 ◦
vowel	0.80	0.58 •	0.85 ◦	0.63 •	0.80
waveform-5000	0.51	0.46 •	0.54 ◦	0.60 ◦	0.66 ◦
zoo	2.16	2.16	2.18 ◦	2.06 •	2.09 •
Average	0.47	0.39	0.49	0.45	0.53

◦, • statistically significant improvement or degradation

3.1.2 Bayesian Network

For testing, a Tree Augmented Naive Bayes (TAN) Bayesian Network using MDL for scoring was used as the internal classifier for the ProjectClassifier, although some small adaptations to the experiment were needed for testing performance. In some cases, most frequently with small data sets, an iteratively trained Bayesian Network did not converge as expected. This led to it running for longer than it would be practical to test for, with no guarantee that it would ever finish iterating. In order to produce results, training was limited to 5000 iterations.

Bayesian Network Percent Incorrect results are shown in Table 3.5. Case 1 did not perform particularly well according to this metric, with results than were approximately equivalent to the results from Case 5, and were slightly worse than the results seen in Case 4. Since Case 1 significantly outperformed Case 2 (with more accurate or similarly accurate results shown in 49/51 tests) and Case 3 performed as well or better than Case 1 in 38/51 tests, this may show that the more a Bayesian Network is trained iteratively the less accurate it becomes.

Bayesian Network Mean Absolute Error results are shown in Table 3.6. Case 1 produced much better results than either control case, doing as well or better than Case 4 in 45/51 tests, and as well or better than Case 5 in 50/51 tests. Case 2 provided very similar results to Case 1, with neither method proving significantly better or worse according to this metric. Case 1 performed better than Case 3 in 27/51 tests, as well as Case 3 in 5/51 tests, and worse than Case 3 in 19/51 tests. This does not allow any clear conclusion to be drawn, since while Case 1 did perform better in more cases, it was not enough to be significant.

Bayesian Network Root Mean Squared Error results are displayed in Table 3.7. Using this measure, it was determined that Cases 4 and 5 both outperformed Case 1. Case 4 produced better results in 32/51 tests, while Case 5 produced better results in 27/51 tests and similar results in 14/51 tests. Case 3 produced a similar result, with better results in 31/51 test cases. The only case which Case 1 performed better than was Case 2, which it did in 37/51 tests. This would appear to indicate that further training using this classifier leads to stronger predictions which are less accurate.

Bayesian Network Mean Entropy Gain results are shown in Table 3.8. Case 1 again showed smaller entropy gains than Cases 3, 4 and 5 in at least 60% of cases. Case 2 again showed even smaller entropy gains, probably due to the increased iteration removing variation.

It would seem that again, the implemented algorithm does not provide any significant gains when using a TAN Bayesian Network. Again, entropy gain significantly decreased due to the repeated training and imputation cycles, but the classifications became more inaccurate, and judging by the RMSE results they were more likely to be very inaccurate.

Table 3.5: Bayesian Network Percent Incorrect

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	37.73	38.68 ○	41.04 ○	50.09 ○	41.78 ○
autos	10.06	9.23 ●	10.83	9.62	9.29 ●
balance-scale	30.55	30.14	28.49 ●	28.49 ●	29.90
breast-cancer	27.48	27.71	29.46 ○	27.41	27.59
bridges-version1	35.45	34.22	38.89 ○	36.00	38.32 ○
car	20.09	20.56 ○	15.41 ●	23.23 ○	21.45 ○
cmc	52.33	52.66 ○	49.26 ●	50.24 ●	52.12
colic	20.00	21.48 ○	19.84	19.94	21.54 ○
cylinder-bands	34.00	33.67	28.38 ●	32.69 ●	34.31
dermatology	4.60	4.75	6.10 ○	3.58 ●	4.84
diabetes	28.72	29.12	27.28 ●	27.52 ●	28.88
ecoli	35.80	35.54	33.03 ●	34.53 ●	34.63 ●
flags	38.75	38.74	42.66 ○	40.51 ○	39.48
glass	47.75	47.65	42.56 ●	46.58	47.03
haberman	31.22	30.37	31.03	27.27 ●	29.26 ●
hayes-roth-train	50.69	53.61 ○	46.97 ●	49.23	51.37
heart-h	18.20	18.55	20.18 ○	19.23 ○	19.15
heart-statlog	26.42	26.42	26.62	27.51 ○	26.50
hepatitis	19.06	18.98	20.20	19.20	20.07 ○
hypothyroid	9.55	10.63 ○	8.15 ●	8.38 ●	7.75 ●
ionosphere	26.38	30.54 ○	19.62 ●	23.94 ●	24.96 ●
iris	26.89	26.75	19.66 ●	23.54 ●	23.31 ●
kr-vs-kp	16.01	17.52 ○	11.73 ●	16.69 ○	17.27 ○
labor	5.60	5.20	8.00 ○	5.20	7.60 ○
letter	61.73	63.06 ○	56.25 ●	62.59 ○	61.68
liver-disorders	38.26	39.94 ○	37.99	39.17	38.72
lung-cancer	50.64	49.99	57.43 ○	45.55	50.90
lymph	15.70	16.58	16.14	16.49	17.30 ○
molecular-biology-promoters	13.11	11.75	23.31 ○	14.00	13.98
mushroom	5.74	6.93 ○	0.57 ●	7.89 ○	7.92 ○
nursery	17.12	18.40 ○	15.16 ●	16.51 ●	16.69 ●
optdigits	16.98	17.33 ○	10.67 ●	16.32 ●	15.99 ●
page-blocks	14.87	16.45 ○	9.23 ●	14.00 ●	10.21 ●
pendigits	29.42	29.88 ○	18.20 ●	26.14 ●	26.59 ●
postoperative-patient-data	30.95	33.40 ○	33.61 ○	30.36	33.62 ○
primary-tumor	55.23	56.20 ○	57.97 ○	54.94	56.91 ○
segment	38.08	36.91 ●	26.16 ●	33.04 ●	32.92 ●
shuttle-landing-control	45.33	46.67	40.00 ●	36.67 ●	41.33
sick	8.81	9.25 ○	7.47 ●	8.84	7.64 ●
solar-flare-2	29.38	29.20	28.22 ●	28.39 ●	29.18
sonar	25.33	25.91	23.43	25.33	24.56
soybean	10.43	11.10 ○	11.23 ○	11.28 ○	12.88 ○
spambase	26.10	25.99	22.76 ●	24.14 ●	25.85
tae	52.85	52.20	53.69	51.64	51.36
tic-tac-toe	30.40	30.45	27.29 ●	29.23 ●	30.12
trains	29.00	32.00	20.00 ●	39.00 ○	32.00
vehicle	55.43	57.54 ○	42.62 ●	55.79	54.90
vote	9.08	9.44 ○	6.78 ●	9.34 ○	9.41
vowel	72.03	73.17 ○	63.98 ●	72.30	71.61
waveform-5000	23.28	23.63 ○	24.25 ○	23.63 ○	23.68 ○
zoo	12.71	13.16	13.50	11.33	12.51
Average	28.85	29.40	26.93	28.52	28.80

○, ● statistically significant improvement or degradation

Table 3.6: Bayesian Network Mean Absolute Error

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	0.03	0.03	0.04 ○	0.04 ○	0.04 ○
autos	0.10	0.10	0.11 ○	0.10	0.10
balance-scale	0.26	0.26 ●	0.26 ●	0.28 ○	0.29 ○
breast-cancer	0.32	0.32 ●	0.35 ○	0.32	0.33 ○
bridges-version1	0.14	0.14 ●	0.16 ○	0.15 ○	0.16 ○
car	0.12	0.12 ○	0.12	0.14 ○	0.17 ○
cmc	0.39	0.38 ●	0.38	0.39	0.40 ○
colic	0.22	0.23 ○	0.23 ○	0.22	0.25 ○
cylinder-bands	0.35	0.35 ○	0.32 ●	0.35	0.37 ○
dermatology	0.02	0.02	0.03 ○	0.02 ●	0.02 ○
diabetes	0.33	0.33 ○	0.34 ○	0.33 ○	0.35 ○
ecoli	0.10	0.10 ●	0.11 ○	0.11 ○	0.12 ○
flags	0.10	0.10	0.11 ○	0.11 ○	0.11 ○
glass	0.16	0.16 ●	0.16	0.16 ○	0.17 ○
haberman	0.38	0.37 ●	0.38	0.37 ●	0.40 ○
hayes-roth-train	0.31	0.31	0.30 ●	0.31 ○	0.32 ○
heart-h	0.09	0.09 ●	0.11 ○	0.10 ○	0.11 ○
heart-statlog	0.30	0.30	0.33 ○	0.31 ○	0.32 ○
hepatitis	0.20	0.20 ●	0.23 ○	0.21 ○	0.23 ○
hypothyroid	0.07	0.07	0.06 ●	0.07 ●	0.10 ○
ionosphere	0.27	0.30 ○	0.22 ●	0.26	0.27
iris	0.20	0.20	0.22 ○	0.21 ○	0.21 ○
kr-vs-kp	0.23	0.24 ○	0.18 ●	0.25 ○	0.30 ○
labor	0.08	0.08	0.11 ○	0.09 ○	0.12 ○
letter	0.06	0.06 ●	0.05 ●	0.06 ○	0.06 ○
liver-disorders	0.46	0.46	0.44 ●	0.46	0.47 ○
lung-cancer	0.34	0.33	0.39 ○	0.31 ●	0.34
lymph	0.10	0.10 ○	0.10 ○	0.11 ○	0.11 ○
molecular-biology-promoters	0.14	0.13	0.24 ○	0.15 ○	0.15 ○
mushroom	0.06	0.07 ○	0.01 ●	0.08 ○	0.08 ○
nursery	0.09	0.10 ○	0.10 ○	0.13 ○	0.14 ○
optdigits	0.04	0.04 ○	0.03 ●	0.04 ○	0.04 ○
page-blocks	0.08	0.08 ○	0.06 ●	0.07 ●	0.10 ○
pendigits	0.07	0.07	0.05 ●	0.07 ○	0.07 ○
postoperative-patient-data	0.27	0.28 ○	0.28 ○	0.27	0.30 ○
primary-tumor	0.06	0.06 ●	0.06 ○	0.06 ○	0.06 ○
segment	0.11	0.11 ●	0.10 ●	0.11 ●	0.11 ●
shuttle-landing-control	0.44	0.44	0.45 ○	0.44	0.44
sick	0.10	0.11 ○	0.10 ●	0.11 ○	0.15 ○
solar-flare-2	0.11	0.11 ●	0.12 ○	0.12 ○	0.13 ○
sonar	0.26	0.26	0.27	0.26	0.26
soybean	0.01	0.01 ○	0.01 ○	0.01 ○	0.02 ○
spambase	0.28	0.28	0.30 ○	0.28 ○	0.29 ○
tae	0.40	0.40	0.42 ○	0.41 ○	0.41 ○
tic-tac-toe	0.36	0.36 ●	0.36 ●	0.37 ○	0.39 ○
trains	0.30	0.34	0.23 ●	0.35 ○	0.34
vehicle	0.29	0.29 ○	0.25 ●	0.29 ○	0.29 ○
vote	0.10	0.10 ○	0.08 ●	0.10 ○	0.10 ○
vowel	0.14	0.14	0.13 ●	0.15 ○	0.14 ○
waveform-5000	0.16	0.17 ○	0.19 ○	0.17 ○	0.17 ○
zoo	0.04	0.04	0.05 ○	0.04	0.05 ○
Average	0.19	0.19	0.19	0.20	0.21

○, ● statistically significant improvement or degradation

Table 3.7: Bayesian Network Root Mean Squared Error

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	0.16	0.16 ○	0.16 ○	0.17 ○	0.16 ○
autos	0.28	0.28	0.30 ○	0.27 •	0.27 •
balance-scale	0.35	0.35 ○	0.35 ○	0.36 ○	0.36 ○
breast-cancer	0.45	0.46 ○	0.45	0.44 •	0.44 •
bridges-version1	0.29	0.29	0.30 ○	0.29	0.29 ○
car	0.27	0.27 ○	0.23 •	0.26 •	0.27
cmc	0.46	0.47 ○	0.45 •	0.45 •	0.46 •
colic	0.42	0.43 ○	0.40 •	0.41 •	0.41 •
cylinder-bands	0.48	0.50 ○	0.45 •	0.47 •	0.47 •
dermatology	0.10	0.10	0.12 ○	0.09 •	0.11
diabetes	0.43	0.44 ○	0.42 •	0.42 •	0.43 •
ecoli	0.24	0.25 ○	0.23 •	0.23 •	0.24 •
flags	0.27	0.27	0.28 ○	0.27	0.27
glass	0.30	0.30 ○	0.29 •	0.29 •	0.29 •
haberman	0.45	0.45	0.45	0.44 •	0.45 •
hayes-roth-train	0.39	0.40 ○	0.38 •	0.39 ○	0.39
heart-h	0.24	0.24	0.24 ○	0.23 •	0.23
heart-statlog	0.43	0.44 ○	0.42 •	0.42 •	0.43 •
hepatitis	0.39	0.39	0.39	0.39 •	0.38 •
hypothyroid	0.19	0.20 ○	0.18 •	0.18 •	0.19 ○
ionosphere	0.45	0.50 ○	0.39 •	0.43 •	0.43 •
iris	0.36	0.36 ○	0.33 •	0.34 •	0.34 •
kr-vs-kp	0.33	0.35 ○	0.29 •	0.34 ○	0.36 ○
labor	0.18	0.18	0.22 ○	0.19	0.22 ○
letter	0.17	0.18 ○	0.16 •	0.17 ○	0.17 •
liver-disorders	0.48	0.49 ○	0.48 •	0.48 ○	0.48
lung-cancer	0.52	0.52	0.56 ○	0.48 •	0.52
lymph	0.23	0.24 ○	0.24	0.24	0.24
molecular-biology-promoters	0.30	0.29	0.43 ○	0.31	0.31
mushroom	0.22	0.25 ○	0.07 •	0.26 ○	0.26 ○
nursery	0.23	0.25 ○	0.21 •	0.23 ○	0.24 ○
optdigits	0.16	0.17 ○	0.13 •	0.16 •	0.16 •
page-blocks	0.21	0.23 ○	0.17 •	0.20 •	0.19 •
pendigits	0.21	0.21 ○	0.16 •	0.20 •	0.20 •
postoperative-patient-data	0.39	0.40 ○	0.41 ○	0.39	0.40 ○
primary-tumor	0.18	0.19 ○	0.18 ○	0.18 •	0.18
segment	0.28	0.28	0.23 •	0.26 •	0.25 •
shuttle-landing-control	0.48	0.48	0.49 ○	0.48	0.49
sick	0.25	0.26 ○	0.23 •	0.25 •	0.24 •
solar-flare-2	0.26	0.26 ○	0.25 •	0.25 •	0.25 •
sonar	0.45	0.46 ○	0.42 •	0.45	0.44 •
soybean	0.10	0.10 ○	0.09	0.10 ○	0.10 ○
spambase	0.44	0.44 ○	0.39 •	0.42 •	0.42 •
tae	0.47	0.47 ○	0.47 ○	0.46 •	0.46 •
tic-tac-toe	0.44	0.44 ○	0.43 •	0.44 •	0.44 •
trains	0.39	0.43	0.29 •	0.43	0.43
vehicle	0.45	0.46 ○	0.37 •	0.44 •	0.44 •
vote	0.29	0.29 ○	0.23 •	0.29	0.29
vowel	0.28	0.29 ○	0.27 •	0.28 •	0.27 •
waveform-5000	0.36	0.36 ○	0.34 •	0.35 •	0.35 •
zoo	0.15	0.15	0.15	0.14 •	0.15
Average	0.32	0.33	0.31	0.32	0.32

○, • statistically significant improvement or degradation

Table 3.8: Bayesian Network mean Entropy Gain

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	-1.00	-1.07 •	-1.86 •	-2.14 •	-0.12 ◦
autos	-0.14	-0.14	0.20 ◦	0.11 ◦	0.17 ◦
balance-scale	0.37	0.37 •	0.37	0.32 •	0.33 •
breast-cancer	-0.07	-0.10 •	-0.02 ◦	-0.00 ◦	0.01 ◦
bridges-version1	0.55	0.47 •	0.61 ◦	0.67 ◦	0.65 ◦
car	0.26	0.24 •	0.67 ◦	0.57 ◦	0.44 ◦
cmc	-0.02	-0.12 •	0.09 ◦	0.04 ◦	0.05 ◦
colic	-0.34	-0.45 •	0.00 ◦	-0.17 ◦	0.00 ◦
cylinder-bands	-0.05	-0.19 •	0.08 ◦	0.04 ◦	0.05 ◦
dermatology	2.26	2.26	2.15 •	2.29 ◦	2.25
diabetes	0.13	0.10 •	0.17 ◦	0.16 ◦	0.15 ◦
ecoli	0.78	0.74 •	0.97 ◦	0.96 ◦	0.87 ◦
flags	0.04	-0.08 •	-0.09 •	0.27 ◦	0.25 ◦
glass	0.43	0.36 •	0.49	0.53 ◦	0.46
haberman	-0.05	-0.05	-0.03	-0.01 ◦	-0.02 ◦
hayes-roth-train	0.12	0.07 •	0.15 ◦	0.10 •	0.08 •
heart-h	0.23	0.22	0.26	0.30 ◦	0.25
heart-statlog	0.14	0.10 •	0.21 ◦	0.22 ◦	0.20 ◦
hepatitis	-0.20	-0.24 •	-0.14	-0.09 ◦	0.01 ◦
hypothyroid	0.07	0.02 •	0.10 ◦	0.09 ◦	0.01 •
ionosphere	-0.24	-0.65 •	0.01 ◦	-0.08 ◦	-0.06 ◦
iris	0.63	0.60 •	0.79 ◦	0.76 ◦	0.78 ◦
kr-vs-kp	0.51	0.47 •	0.60 ◦	0.48 •	0.42 •
labor	0.66	0.68 ◦	0.65	0.70 ◦	0.65
letter	1.55	1.36 •	2.03 ◦	1.58 ◦	1.67 ◦
liver-disorders	0.04	0.01 •	0.06	0.03 •	0.04
lung-cancer	-2.01	-2.02	-1.97	-1.21 ◦	-1.63 ◦
lymph	0.71	0.67 •	0.62 •	0.72	0.68
molecular-biology-promoters	0.39	0.43	-0.28 •	0.43	0.39
mushroom	0.64	0.46 •	0.97 ◦	0.57 •	0.59 •
nursery	0.81	0.48 •	1.14 ◦	1.00 ◦	0.89 ◦
optdigits	2.14	2.10 •	2.78 ◦	2.37 ◦	2.47 ◦
page-blocks	0.04	-0.12 •	0.24 ◦	0.11 ◦	0.09 ◦
pendigits	1.62	1.56 •	2.49 ◦	2.00 ◦	2.03 ◦
postoperative-patient-data	-0.20	-0.23 •	-0.24	-0.17 ◦	-0.18
primary-tumor	0.56	0.49 •	0.62 ◦	0.71 ◦	0.66 ◦
segment	0.76	0.76	1.81 ◦	1.25 ◦	1.33 ◦
shuttle-landing-control	0.04	0.03	-0.01 •	0.04	-0.00
sick	0.02	-0.01 •	0.08 ◦	0.04 ◦	0.03 ◦
solar-flare-2	1.19	1.15 •	1.32 ◦	1.28 ◦	1.26 ◦
sonar	-0.51	-0.67 •	0.09 ◦	-0.33 ◦	-0.22 ◦
soybean	3.17	3.08 •	3.31 ◦	3.23 ◦	3.14 •
spambase	0.06	0.01 •	0.28 ◦	0.17 ◦	0.18 ◦
tae	-0.02	-0.04 •	-0.02	0.03 ◦	0.05 ◦
tic-tac-toe	0.11	0.10 •	0.16 ◦	0.12	0.12
trains	-0.17	-0.58 •	0.32 ◦	0.01	-0.26
vehicle	-1.17	-1.51 •	0.51 ◦	-0.68 ◦	-0.60 ◦
vote	0.12	0.06 •	0.66 ◦	0.28 ◦	0.32 ◦
vowel	0.57	0.34 •	1.00 ◦	0.63 ◦	0.69 ◦
waveform-5000	0.40	0.34 •	0.78 ◦	0.60 ◦	0.61 ◦
zoo	1.95	1.91	1.94	2.06 ◦	1.96
Average	0.35	0.27	0.53	0.45	0.47

◦, • statistically significant improvement or degradation

3.1.3 J48 Decision Tree

When testing the J48 decision tree, adaptations were needed. The J48 failed to converge as expected in most cases, although this was most common and also most time consuming with very large data sets. Through experimentation it was determined that iteratively training the J48 classifier caused it to reach a certain state of stability around which it would fluctuate by changing a handful of rows between iterations, usually after around 15-20 rounds of training and reimputing. It therefore made sense to limit training to 30 iterations to allow experiments to be run in a reasonable period of time while also providing some 10-15 extra iterations to allow for even more volatile datasets to reach relative stability.

J48 Percent Incorrect results are shown in Table 3.9. Case 1 performed as well as or better than Cases 2, 4 and 5 in almost all tests, which was the expected result for the control cases. Case 1 was again outperformed by Case 3, performing worse in 16/51 tests and even in 26/51.

J48 Mean Absolute Error results can be found in Table 3.10. Interestingly, using this metric Case 1 performed much better than Cases 3, 4 or 5, with better results in 38/51, 37/51 and 47/51 data sets respectively. This result was not seen while using Naive Bayes or Bayesian Networks as the classifier. This could be due to the fact that J48 was capped at 30 iterations to reduce experiment time and this may suggest that there is an optimal number of iterations for this method. Case 2 also was significantly more accurate than Case 1 in 21/51 tests, and approximately the same in 26/51.

J48 Root Mean Squared Error resuls are shown in Table 3.11. Case 1 again performed much better than either Case 4 or 5, which was expected since these were the control cases. It also performed as well as or better than Case 2 in all 51 data sets. Case 1 was heavily outperformed by Case 3 using RMSE as the metric for comparison, since the results for Case 3 were as good or better in 44/51 data sets.

J48 Mean Entropy Gain results are shown in Table 3.12. Case 1 signifcantly outperformed Case 3 using this measure, as it rewards a reduction in entropy. Since the iterative training will naturally lead to decreased variety in the results shown, this result is expected. Case 1 generally produced a smaller reduction in entropy than Case 2, since it was not also iteratively training the class attribute. Cases 4 and 5 can effectively be disregarded here, since they're just imputing pseudo-random data, and this will naturally lead to a lot of variance between results.

From the above results, there does not appear to be much benefit to using the implemented algorithm with the J48 classifier. The classifier became less accurate when trained iteratively, and also appeared to make larger errors according to the RMSE test. The MAE result is an interesting case though, and could prove interesting if further explored.

Table 3.9: J48 Decision Tree Percent Incorrect

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	29.13	31.13 ○	32.94 ○	35.33 ○	36.43 ○
autos	13.68	14.78	13.02	14.68	12.59
balance-scale	28.72	29.47 ○	29.02	29.56 ○	29.10
breast-cancer	25.70	26.91 ○	26.80 ○	26.68	28.09 ○
bridges-version1	43.54	45.20	41.77	45.89	44.09
car	16.03	15.95	16.73 ○	20.18 ○	17.85 ○
cmc	49.48	49.07	50.17 ○	50.40	51.20 ○
colic	18.03	18.18	17.32 ●	18.98 ○	18.52
cylinder-bands	34.06	34.01	34.84	35.73 ○	35.38
dermatology	10.97	12.26	10.04	13.33 ○	13.97 ○
diabetes	29.29	29.00	28.61	27.73 ●	28.77
ecoli	35.34	34.72	34.92	37.23 ○	34.46
flags	40.07	42.00 ○	41.94	44.77 ○	42.50
glass	41.32	40.66	38.89 ●	41.35	42.51
haberman	27.56	26.90 ●	26.57 ●	26.57 ●	26.57 ●
hayes-roth-train	37.26	37.78	37.95	42.43 ○	41.13 ○
heart-h	21.09	21.81	22.94 ○	23.77 ○	21.81
heart-statlog	32.01	30.36 ●	32.32	30.04 ●	32.17
hepatitis	22.84	22.02	22.10	23.28	22.39
hypothyroid	7.76	7.73	7.73	7.73	7.73
ionosphere	20.67	18.76 ●	19.37	21.55	22.19
iris	25.87	26.59	20.50 ●	22.24 ●	23.71
kr-vs-kp	5.97	6.05	6.09	5.88	7.79 ○
labor	23.60	26.20	27.80 ○	21.40	21.80
letter	49.11	49.10	47.39 ●	51.04 ○	49.21
liver-disorders	39.83	40.06	40.16	38.54	41.57 ○
lung-cancer	59.79	62.11	67.29 ○	55.14	65.87
lymph	20.02	20.34	21.75	23.36 ○	21.01
molecular-biology-promoters	21.19	21.59	20.22	26.85 ○	23.67
mushroom	2.22	3.47 ○	0.86 ●	0.57 ●	1.33 ●
nursery	12.00	12.22 ○	12.58 ○	14.34 ○	13.15 ○
optdigits	17.18	17.09	13.90 ●	23.70 ○	17.69 ○
page-blocks	7.27	7.30 ○	7.25	7.27	7.28
pendigits	15.72	15.79	11.80 ●	16.89 ○	13.67 ●
postoperative-patient-data	29.84	29.24	29.24	29.24	29.35
primary-tumor	61.77	61.54	60.68	61.71	62.96
segment	26.45	26.61	21.77 ●	25.39 ●	24.26 ●
shuttle-landing-control	41.33	42.00	40.00	41.33	43.33
sick	6.20	6.19	6.19	6.19	6.21
solar-flare-2	30.14	30.30	28.17 ●	29.45 ●	29.04 ●
sonar	32.41	33.22	30.93	32.63	32.86
soybean	18.86	18.09	16.24 ●	18.71	20.59 ○
spambase	22.43	22.36	21.79 ●	22.36	21.84 ●
tae	49.35	49.85	48.07 ●	51.07	52.30 ○
tic-tac-toe	22.47	22.08	21.55	21.98	23.79 ○
trains	1.00	10.00 ○	0.00	0.00	8.00
vehicle	40.13	40.98 ○	38.88 ●	42.76 ○	41.39
vote	8.67	8.44	6.34 ●	7.03 ●	7.49 ●
vowel	49.88	50.02	52.36 ○	54.88 ○	53.20 ○
waveform-5000	27.95	27.44 ●	25.72 ●	32.41 ○	29.50 ○
zoo	15.94	18.01 ○	15.39	18.76 ○	18.10
Average	26.85	27.31	26.41	27.85	27.87

○, ● statistically significant improvement or degradation

Table 3.10: J48 Decision Tree Mean Absolute Error

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	0.03	0.03	0.04 ○	0.04 ○	0.04 ○
autos	0.19	0.19	0.19	0.21 ○	0.21 ○
balance-scale	0.26	0.26	0.29 ○	0.28 ○	0.30 ○
breast-cancer	0.37	0.37	0.38 ○	0.37	0.40 ○
bridges-version1	0.16	0.17	0.17 ○	0.18 ○	0.19 ○
car	0.09	0.09	0.12 ○	0.12 ○	0.14 ○
cmc	0.39	0.38 ●	0.40 ○	0.39 ○	0.40 ○
colic	0.26	0.25 ●	0.28 ○	0.27 ○	0.30 ○
cylinder-bands	0.37	0.37	0.40 ○	0.39 ○	0.40 ○
dermatology	0.05	0.06 ○	0.07 ○	0.07 ○	0.09 ○
diabetes	0.36	0.35 ●	0.38 ○	0.36 ○	0.38 ○
ecoli	0.10	0.10	0.11 ○	0.11 ○	0.12 ○
flags	0.12	0.12	0.13 ○	0.13 ○	0.13 ○
glass	0.14	0.14	0.15 ○	0.15 ○	0.16 ○
haberman	0.39	0.38 ●	0.39	0.38 ●	0.40 ○
hayes-roth-train	0.24	0.23 ●	0.26 ○	0.27 ○	0.28 ○
heart-h	0.11	0.11	0.12 ○	0.13 ○	0.14 ○
heart-statlog	0.37	0.36	0.40 ○	0.38	0.39 ○
hepatitis	0.29	0.27 ●	0.29	0.28	0.30
hypothyroid	0.07	0.07 ●	0.07	0.07 ●	0.11 ○
ionosphere	0.25	0.23 ●	0.27 ○	0.26 ○	0.28 ○
iris	0.20	0.20	0.22 ○	0.23 ○	0.24 ○
kr-vs-kp	0.09	0.09 ●	0.14 ○	0.13 ○	0.19 ○
labor	0.25	0.26	0.32 ○	0.22	0.27
letter	0.05	0.04 ●	0.05 ○	0.05 ○	0.05 ○
liver-disorders	0.45	0.45	0.46 ○	0.46 ○	0.47 ○
lung-cancer	0.39	0.40	0.44 ○	0.37	0.44 ○
lymph	0.13	0.13	0.15 ○	0.14 ○	0.15 ○
molecular-biology-promoters	0.25	0.25	0.27	0.30 ○	0.29 ○
mushroom	0.05	0.05 ○	0.03 ●	0.05 ○	0.09 ○
nursery	0.06	0.06 ○	0.09 ○	0.10 ○	0.12 ○
optdigits	0.05	0.05 ●	0.05 ○	0.06 ○	0.06 ○
page-blocks	0.05	0.05 ●	0.05 ○	0.05 ○	0.08 ○
pendigits	0.05	0.05 ●	0.05 ○	0.06 ○	0.06 ○
postoperative-patient-data	0.28	0.28 ●	0.28	0.28 ●	0.30 ○
primary-tumor	0.06	0.06	0.07 ○	0.06	0.07 ○
segment	0.10	0.09 ●	0.10 ○	0.11 ○	0.12 ○
shuttle-landing-control	0.49	0.49	0.49	0.49	0.49
sick	0.12	0.11 ●	0.12	0.11 ●	0.16 ○
solar-flare-2	0.12	0.12 ●	0.13 ○	0.13 ○	0.14 ○
sonar	0.34	0.35	0.35	0.34	0.35
soybean	0.02	0.02 ●	0.03 ○	0.03 ○	0.04 ○
spambase	0.31	0.30 ●	0.32 ○	0.33 ○	0.33 ○
tae	0.40	0.39 ●	0.40 ○	0.41 ○	0.41 ○
tic-tac-toe	0.28	0.28	0.30 ○	0.28	0.32 ○
trains	0.01	0.09 ○	0.00	0.00	0.09 ○
vehicle	0.23	0.23	0.25 ○	0.25 ○	0.25 ○
vote	0.11	0.10	0.11	0.13 ○	0.15 ○
vowel	0.10	0.10	0.12 ○	0.11 ○	0.11 ○
waveform-5000	0.21	0.21 ●	0.22 ○	0.24 ○	0.24 ○
zoo	0.06	0.07	0.07	0.07 ○	0.08 ○
Average	0.19	0.19	0.21	0.20	0.22

○, ● statistically significant improvement or degradation

Table 3.11: J48 Root Mean Squared Error

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	0.13	0.14 ○	0.14 ○	0.15 ○	0.15 ○
autos	0.33	0.35	0.33	0.35	0.32
balance-scale	0.37	0.38 ○	0.37	0.38 ○	0.38
breast-cancer	0.44	0.45	0.44	0.45	0.45 ○
bridges-version1	0.32	0.33 ○	0.31	0.33 ○	0.33 ○
car	0.23	0.23	0.23 ○	0.26 ○	0.25 ○
cmc	0.46	0.46	0.45 ●	0.46 ○	0.46
colic	0.38	0.39	0.37 ●	0.40 ○	0.39
cylinder-bands	0.49	0.50	0.48 ●	0.53 ○	0.50
dermatology	0.17	0.18 ○	0.16	0.19 ○	0.19 ○
diabetes	0.44	0.45 ○	0.44 ●	0.44	0.44
ecoli	0.24	0.24 ○	0.23	0.24 ○	0.24
flags	0.27	0.28	0.27	0.30 ○	0.28
glass	0.29	0.29	0.28 ●	0.30 ○	0.29
haberman	0.45	0.44	0.44 ●	0.44	0.44
hayes-roth-train	0.36	0.36	0.36	0.38 ○	0.37 ○
heart-h	0.25	0.25	0.26 ○	0.27 ○	0.25
heart-statlog	0.47	0.47	0.47	0.47	0.47
hepatitis	0.43	0.42	0.41 ●	0.44	0.42
hypothyroid	0.19	0.19	0.19	0.19	0.20 ○
ionosphere	0.40	0.38	0.38	0.43 ○	0.42 ○
iris	0.33	0.33	0.32 ●	0.34 ○	0.33
kr-vs-kp	0.21	0.21	0.22 ○	0.24 ○	0.26 ○
labor	0.40	0.41	0.45 ○	0.40	0.39
letter	0.15	0.16 ○	0.15 ●	0.16 ○	0.16 ○
liver-disorders	0.49	0.50 ○	0.49	0.50	0.50
lung-cancer	0.56	0.56	0.57	0.55	0.59 ○
lymph	0.28	0.29	0.29	0.31 ○	0.29
molecular-biology-promoters	0.41	0.41	0.40	0.47 ○	0.43
mushroom	0.14	0.15 ○	0.09 ●	0.10 ●	0.13 ●
nursery	0.18	0.18 ○	0.19 ○	0.22 ○	0.20 ○
optdigits	0.16	0.16	0.15 ●	0.20 ○	0.17 ○
page-blocks	0.16	0.16	0.16 ○	0.16 ○	0.16 ○
pendigits	0.15	0.15	0.14 ●	0.17 ○	0.15
postoperative-patient-data	0.38	0.38	0.38	0.38	0.38
primary-tumor	0.20	0.20	0.19 ●	0.20 ○	0.20
segment	0.22	0.23	0.21 ●	0.24 ○	0.23
shuttle-landing-control	0.50	0.50	0.50	0.50	0.51
sick	0.24	0.24	0.24	0.24	0.25 ○
solar-flare-2	0.25	0.25 ○	0.25 ●	0.26 ○	0.26 ○
sonar	0.51	0.52	0.48 ●	0.54 ○	0.51
soybean	0.12	0.12	0.11 ●	0.13 ○	0.13 ○
spambase	0.39	0.40 ○	0.39 ●	0.41 ○	0.39
tae	0.46	0.46	0.45 ●	0.47 ○	0.46
tic-tac-toe	0.41	0.41	0.39 ●	0.41	0.42
trains	0.01	0.09 ○	0.00	0.00	0.10 ○
vehicle	0.36	0.36 ○	0.35 ●	0.39 ○	0.37 ○
vote	0.24	0.24	0.23 ●	0.25	0.24
vowel	0.24	0.25	0.24	0.27 ○	0.25 ○
waveform-5000	0.38	0.37	0.35 ●	0.43 ○	0.38 ○
zoo	0.19	0.19	0.18 ●	0.20 ○	0.20
Average	0.31	0.31	0.31	0.32	0.32

○, ● statistically significant improvement or degradation

Table 3.12: J48 Decision Tree Mean Entropy Gain

Dataset	(1)	(2)	(3)	(4)	(5)
audiology	-111.99	-117.88	-87.04 ○	-158.79 ●	-132.51 ●
autos	-17.82	-18.41	-3.68 ○	-21.39	-10.74
balance-scale	-2.57	-2.22	0.25 ○	-2.59	-0.13 ○
breast-cancer	-13.39	-10.55	-1.20 ○	-16.63	-10.55
bridges-version1	-151.55	-162.36	-77.69 ○	-133.17	-133.59
car	-17.59	-17.81	-1.06 ○	-39.71 ●	-13.41 ○
cmc	-24.22	-20.66	-0.50 ○	-31.94 ●	-15.37 ○
colic	-19.82	-20.17	-0.69 ○	-36.02 ●	-15.21
cylinder-bands	-48.29	-54.02	-1.51 ○	-101.43 ●	-45.65
dermatology	-19.57	-28.30 ●	-0.63 ○	-55.02 ●	-25.22
diabetes	-2.86	-4.58	0.11 ○	-2.54	-1.14
ecoli	-38.97	-39.01	-21.92 ○	-51.93 ●	-21.36 ○
flags	-176.53	-191.86	-59.40 ○	-262.88 ●	-194.28
glass	-85.34	-95.00	-20.88 ○	-134.49 ●	-66.81 ○
haberman	-1.60	-0.01	0.00	-0.00	-0.00
hayes-roth-train	-26.51	-26.47	0.32 ○	-24.76	-6.18 ○
heart-h	-24.50	-22.09	-0.60 ○	-21.34	-11.60 ○
heart-statlog	-17.59	-24.89 ●	0.08 ○	-24.89	-13.21
hepatitis	-34.79	-43.30	-5.45 ○	-56.53	-27.84
hypothyroid	-0.25	-0.00	-0.00	-0.00	-0.06
ionosphere	-37.49	-31.33	-2.47 ○	-86.84 ●	-43.67
iris	-17.69	-18.53	0.06 ○	-14.53	0.79 ○
kr-vs-kp	-0.18	-0.03	0.73 ○	-4.54 ●	-1.74 ●
labor	-59.74	-46.92	-14.96 ○	-78.97	-38.26
letter	-49.47	-50.59 ●	-2.75 ○	-123.96 ●	-42.88 ○
liver-disorders	-7.41	-8.49	-0.34 ○	-7.05	-1.76 ○
lung-cancer	-363.37	-347.14	-285.77	-349.74	-427.12
lymph	-59.55	-57.39	-14.26 ○	-98.35 ●	-58.21
molecular-biology-promoters	-90.56	-102.45	-10.05 ○	-120.80 ●	-86.50
mushroom	0.91	0.90 ●	0.94 ○	0.12 ●	0.83 ●
nursery	-4.78	-5.08	1.23 ○	-8.25 ●	-1.33 ○
optdigits	-46.57	-47.29	-5.72 ○	-172.03 ●	-48.23
page-blocks	-0.44	-0.52	-0.10 ○	-1.16 ●	0.08 ○
pendigits	-10.10	-10.69 ●	1.74 ○	-52.19 ●	-5.57 ○
postoperative-patient-data	-1.28	-0.01	-0.01	-0.01	-1.28
primary-tumor	-297.02	-284.90	-120.47 ○	-366.49 ●	-315.85
segment	-11.43	-12.00	0.95 ○	-51.51 ●	-6.16 ○
shuttle-landing-control	-0.01	-0.03	-0.00	-0.01	-21.46
sick	0.00	-0.00	0.00	-0.00	-0.02 ●
solar-flare-2	-20.25	-23.66	-1.14 ○	-45.09 ●	-21.53
sonar	-132.17	-130.59	-26.02 ○	-213.09 ●	-120.45
soybean	-24.16	-22.74	-4.49 ○	-112.98 ●	-36.20 ●
spambase	-0.12	-0.41 ●	0.28 ○	0.15 ○	0.27 ○
tae	-16.71	-9.90	0.09 ○	-18.33	-5.29 ○
tic-tac-toe	-30.09	-29.23	0.12 ○	-34.08	-23.32 ○
trains	0.99	-42.09 ●	1.01	1.01	-42.08 ●
vehicle	-36.01	-35.91	-3.43 ○	-125.50 ●	-40.04
vote	-5.11	-6.47	-0.43 ○	-19.92 ●	-5.95
vowel	-103.51	-103.65	-15.15 ○	-292.68 ●	-121.50 ●
waveform-5000	-49.76	-46.96	-0.12 ○	-165.15 ●	-55.58 ●
zoo	-45.13	-40.46	-19.36 ○	-111.23 ●	-50.10
Average	-46.16	-47.34	-15.71	-75.48	-46.37

○, ● statistically significant improvement or degradation

Chapter 4

Conclusion

This body of work has implemented a novel method of training models for use in classifying nominal data sets, and has experimented with using this method together with a number of established classification techniques. Through these experiments it has been shown that the implemented method of iteratively imputing and retraining classifiers does not lead to greater classification accuracy, but does decrease the entropy of classified data sets. This may be useful for some specific domains in which data generally has a low statistical entropy anyway, but is probably not useful in the vast majority of cases.

As a project, all objectives have been fulfilled. A functional, distributable Weka plugin has been developed which integrates into the system through its package manager and passing all unit tests imposed by the system. It is usable from both the CLI and GUI, and allows for easy configuration of its parameters and the parameters of its internal classifiers. It has also been used for experiments as intended, and some minor insights have been gleaned from these.

Bibliography

[1] C. McKee and C. P. de Campos, "Gitlab repository." `https://gitlab.eeecs.qub.ac.uk/cassiopc/cdc04`, 2017.

[2] E. Frank, M. A. Hall, and I. H. Witten, *The WEKA Workbench*. Morgan Kaufmann, fourth ed., 2016.

[3] M. Lichman, "UCI machine learning repository." `http://archive.ics.uci.edu/ml`, 2013.

Appendix A

Experimental Data Sets

Make a table showing some size / info about each data set here

Appendix B

Hidden Variables

TODO: Gather hidden variables results and discuss.

Appendix C

User Guide

Explain:

- installing plugin from the command line / from the gui - using plugin from the CLI - using plugin from the GUI

Appendix D

JavaDoc

CDC04 TeXDoclet

Christopher McKee

March 19, 2017

Contents

Class Hierarchy **2**

1 Package cdc04 **3**
1.1 Class StateAnalyser . . . 3
1.1.1 Declaration . . . 3
1.1.2 Constructor summary . . . 3
1.1.3 Method summary . . . 3
1.1.4 Constructors . . . 3
1.1.5 Methods . . . 4

2 Package weka.classifiers.meta **5**
2.1 Class ProjectClassifier . . . 5
2.1.1 Declaration . . . 5
2.1.2 Constructor summary . . . 6
2.1.3 Method summary . . . 6
2.1.4 Constructors . . . 6
2.1.5 Methods . . . 7
2.1.6 Members inherited from class SingleClassifierEnhancer . . . 13
2.1.7 Members inherited from class AbstractClassifier . . . 13

Class Hierarchy

Classes

- java.lang.Object
    - cdc04.StateAnalyser (in 1.1, page 3)
    - weka.classifiers.AbstractClassifier
        - weka.classifiers.SingleClassifierEnhancer
            - weka.classifiers.meta.ProjectClassifier (in 2.1, page 5)

Chapter 1

Package cdc04

Package Contents *Page*

Classes

StateAnalyser 3
Class written to record a series of Weka instances, and to determine differences between them.

1.1 Class StateAnalyser

Class written to record a series of Weka instances, and to determine differences between them.

1.1.1 Declaration

```
public class StateAnalyser
 extends java.lang.Object implements java.io.Serializable
```

1.1.2 Constructor summary

StateAnalyser() Constructs a new instance with no recorded instances

1.1.3 Method summary

addInstances(Instances) Adds an Instances object to be tracked
getNumberDifferences() Returns an integer representation of the number of rows which have changed between the previous two iterations of the classifier A negative return values shows that the number of differences could not be calculated, since there are not at least two iterations present.
getNumberIterations() Returns the number of instances which are currently contained in the tracker for analysis

1.1.4 Constructors

- **StateAnalyser**

```
public StateAnalyser()
```

- **Description**
  Constructs a new instance with no recorded instances

1.1.5 Methods

- **addInstances**

```
public void addInstances(weka.core.Instances toAdd)
```

  - **Description**
    Adds an Instances object to be tracked
  - **Parameters**
    - `toAdd` – Instances object to be added

- **getNumberDifferences**

```
public int getNumberDifferences()
```

  - **Description**
    Returns an integer representation of the number of rows which have changed between the previous two iterations of the classifier A negative return values shows that the number of differences could not be calculated, since there are not at least two iterations present.

- **getNumberIterations**

```
public int getNumberIterations()
```

  - **Description**
    Returns the number of instances which are currently contained in the tracker for analysis
  - **Returns** – the current number of recorded instances

Chapter 2

Package weka.classifiers.meta

Package Contents *Page*

Classes

ProjectClassifier 5
A classifier which is iteratively trained, imputing missing values into copies of the training data until no further change is observed.

2.1 Class ProjectClassifier

A classifier which is iteratively trained, imputing missing values into copies of the training data until no further change is observed. Builds one learner per attribute, and therefore can take quite a while to run. Valid options are:

```
-W classifier
Full path to the target classifier to use, e.g. weka.classifiers.trees.J48

-S
Defines whether or not the classifier will impute a value for the class
attribute as it trains.

-R
If set, the classifiers will be trained with any missing arguments filled in
by random data. The classifier will then only iterate once.

-M integer
Sets the maximum number of times that a particular classifier will iterate
before determining that it is trained.
```

Options after – are passed to the currently selected classifier.

2.1.1 Declaration

public class ProjectClassifier

extends weka.classifiers.SingleClassifierEnhancer **implements** weka.classifiers.IterativeClassifier

2.1.2 Constructor summary

ProjectClassifier() Constructor

2.1.3 Method summary

buildClassifier(Instances) Builds a set of classifiers based on the training data.
classifierOptionsTipText() Tip text to be displayed in the GUI for this property
classifyInstance(Instance) Classifies an instance.
defaultClassifierString() String describing default classifier.
distributionForInstance(Instance) Returns class probabilities for an instance.
done() Method called when iteration has terminated.
getCapabilities() Returns default capabilities of the classifier.
getClassifierOptions() Gets classifier options
getMaxIterations() Get the value of m_MaxIterations
getNumHiddenVariables()
getOptions() Gets the current settings of the Classifier.
getRandomData() Get the value of m_RandomData
getSupervised() Get the value of m_Supervised
globalInfo() Global information about the class
initializeClassifier(Instances) Makes copies of the training data which can be mutated, and initialise the array of Classifier objects
listOptions() Returns an enumeration describing the available options.
main(String[]) Main method for testing this class.
maxIterationsTipText() Tip text to be displayed in the GUI for this property
next() Retrains each of the classifiers, then attempts to impute missing data in a copy of the training data.
numHiddenVariablesTipText()
randomDataTipText() Tip text to be displayed in the GUI for this property
setClassifierOptions(String[]) Sets classifier options
setMaxIterations(int) Set the value of m_MaxIterations.
setNumHiddenVariables(int)
setOptions(String[]) Parses a given list of options.
setRandomData(boolean) Set the value of m_RandomData
setSupervised(boolean) Set the value of m_Supervised
supervisedTipText() Tip text to be displayed in the GUI for this property

2.1.4 Constructors

- **ProjectClassifier**

  public ProjectClassifier()

  - **Description**
    Constructor

2.1.5 Methods

- **buildClassifier**

```
public void buildClassifier(weka.core.Instances instances)
  throws java.lang.Exception
```

  - **Description**
    Builds a set of classifiers based on the training data. These are iteratively trained on copies of the data.
  - **Parameters**
    * `instances` – the Instances object which comprises the training data
  - **Throws**
    * `java.lang.Exception` – exception thrown is raised to a Weka error handler

- **classifierOptionsTipText**

```
public java.lang.String classifierOptionsTipText()
```

  - **Description**
    Tip text to be displayed in the GUI for this property
  - **Returns** – tip text to be displayed in the GUI

- **classifyInstance**

```
public double classifyInstance(weka.core.Instance instance)
  throws java.lang.Exception
```

  - **Description**
    Classifies an instance.
  - **Parameters**
    * `instance` – the instance to classify
  - **Returns** – the classification for the instance
  - **Throws**
    * `java.lang.Exception` – if instance can't be classified successfully

- **defaultClassifierString**

```
protected java.lang.String defaultClassifierString()
```

  - **Description**
    String describing default classifier.

- **distributionForInstance**

```
public double[] distributionForInstance(weka.core.Instance
    instance) throws java.lang.Exception
```

  - **Description**
    Returns class probabilities for an instance.
  - **Parameters**
    - `instance` – the instance to calculate the class probabilities for
  - **Returns** – the class probabilities
  - **Throws**
    - `java.lang.Exception` – if distribution can't be computed successfully

- **done**

```
public void done() throws java.lang.Exception
```

  - **Description**
    Method called when iteration has terminated. Imputes class values if m_Supervised is set.

- **getCapabilities**

```
public weka.core.Capabilities getCapabilities()
```

  - **Description**
    Returns default capabilities of the classifier.
  - **Returns** – the capabilities of this classifier

- **getClassifierOptions**

```
public java.lang.String[] getClassifierOptions()
```

  - **Description**
    Gets classifier options
  - **Returns** – array of String objects to be passed to each classifier

- **getMaxIterations**

```
public int getMaxIterations()
```

  - **Description**
    Get the value of m_MaxIterations
  - **Returns** – value of m_MaxIterations

- **getNumHiddenVariables**

```
public int getNumHiddenVariables()
```

- **getOptions**

```
public java.lang.String[] getOptions()
```

  - **Description**
    Gets the current settings of the Classifier.
  - **Returns** – an array of strings suitable for passing to setOptions

- **getRandomData**

```
public boolean getRandomData()
```

  - **Description**
    Get the value of m_RandomData
  - **Returns** – value of m_RandomData

- **getSupervised**

```
public boolean getSupervised()
```

  - **Description**
    Get the value of m_Supervised
  - **Returns** – value of m_Supervised

- **globalInfo**

```
public java.lang.String globalInfo()
```

  - **Description**
    Global information about the class
  - **Returns** – information about the classifier which is displayed in the CLI/GUI

- **initializeClassifier**

```
public void initializeClassifier(weka.core.Instances instances)
  throws java.lang.Exception
```

- **Description**
  Makes copies of the training data which can be mutated, and initialise the array of Classifier objects
- **Parameters**
  - `instances` – the training data

- **listOptions**

```
public java.util.Enumeration listOptions()
```

- **Description**
  Returns an enumeration describing the available options.
- **Returns** – an enumeration of all the available options.

- **main**

```
public static void main(java.lang.String[] args)
```

- **Description**
  Main method for testing this class.
- **Parameters**
  - `args` – the options

- **maxIterationsTipText**

```
public java.lang.String maxIterationsTipText()
```

- **Description**
  Tip text to be displayed in the GUI for this property
- **Returns** – tip text to be displayed in the GUI

- **next**

```
public boolean next() throws java.lang.Exception
```

- **Description**
  Retrains each of the classifiers, then attempts to impute missing data in a copy of the training data. Does not iterate again if the results of current iteration match the results of the previous iteration, or the max number of iterations has been reached.

- **Returns** – true if another iteration should be performed, otherwise false.

- **numHiddenVariablesTipText**

```
public java.lang.String numHiddenVariablesTipText()
```

- **randomDataTipText**

```
public java.lang.String randomDataTipText()
```

  - **Description**
    Tip text to be displayed in the GUI for this property
  - **Returns** – tip text to be displayed in the GUI

- **setClassifierOptions**

```
public void setClassifierOptions(java.lang.String[]
    classifierOptions)
```

  - **Description**
    Sets classifier options
  - **Parameters**
    * `classifierOptions` – array of String objects to be passed to each classifier

- **setMaxIterations**

```
public void setMaxIterations(int maxIterations)
```

  - **Description**
    Set the value of m_MaxIterations. Defaults to Integer.MAX_VALUE if value less than 0 is supplied.
  - **Parameters**
    * `maxIterations` – new value of m_MaxIterations

- **setNumHiddenVariables**

```
public void setNumHiddenVariables(int numHiddenVariables)
```

- **setOptions**

```
public void setOptions(java.lang.String[] options) throws java.
    lang.Exception
```

- **Description**
  Parses a given list of options. Valid options are:
  -W classifier
  Full path to the target classifier to use, e.g. weka.classifiers.trees.J48
  -S
  Defines whether or not the classifier will impute a value for the class attribute as it trains.
  -R
  If set, the classifiers will be trained with any missing arguments filled in by random data. The classifier will then only iterate once.
  -M integer
  Sets the maximum number of times that a particular classifier will iterate before determining that it is trained.
  Options after – are passed to the currently selected classifier.
- **Parameters**
  - `options` – The list of options as an array of Strings
- **Throws**
  - `java.lang.Exception` – if an option is not supported

- **setRandomData**

```
public void setRandomData(boolean randomData)
```

  - **Description**
    Set the value of m_RandomData
  - **Parameters**
    - `randomData` – new value of m_RandomData

- **setSupervised**

```
public void setSupervised(boolean supervised)
```

  - **Description**
    Set the value of m_Supervised
  - **Parameters**
    - `supervised` – the new value of m_Supervised

- **supervisedTipText**

```
public java.lang.String supervisedTipText()
```

  - **Description**
    Tip text to be displayed in the GUI for this property
  - **Returns** – tip text to be displayed in the GUI

2.1.6 Members inherited from class SingleClassifierEnhancer

weka.classifiers.SingleClassifierEnhancer

- public String **classifierTipText**()
- protected String **defaultClassifierOptions**()
- protected String **defaultClassifierString**()
- public Capabilities **getCapabilities**()
- public Classifier **getClassifier**()
- protected String **getClassifierSpec**()
- public String **getOptions**()
- public Enumeration **listOptions**()
- protected **m_Classifier**
- public void **postExecution**() throws java.lang.Exception
- public void **preExecution**() throws java.lang.Exception
- public void **setClassifier**(Classifier **arg0**)
- public void **setOptions**(java.lang.String[] **arg0**) throws java.lang.Exception

2.1.7 Members inherited from class AbstractClassifier

weka.classifiers.AbstractClassifier

- public static **BATCH_SIZE_DEFAULT**
- public String **batchSizeTipText**()
- public double **classifyInstance**(weka.core.Instance **arg0**) throws java.lang.Exception
- public String **debugTipText**()
- public double **distributionForInstance**(weka.core.Instance **arg0**) throws java.lang.Exception
- public double **distributionsForInstances**(weka.core.Instances **arg0**) throws java.lang.Exception
- public String **doNotCheckCapabilitiesTipText**()
- public static Classifier **forName**(java.lang.String **arg0**, java.lang.String[] **arg1**) throws java.lang.Exception
- public String **getBatchSize**()
- public Capabilities **getCapabilities**()
- public boolean **getDebug**()
- public boolean **getDoNotCheckCapabilities**()
- public int **getNumDecimalPlaces**()
- public String **getOptions**()
- public String **getRevision**()
- public boolean **implementsMoreEfficientBatchPrediction**()
- public Enumeration **listOptions**()
- protected **m_BatchSize**
- protected **m_Debug**
- protected **m_DoNotCheckCapabilities**
- protected **m_numDecimalPlaces**
- public static Classifier **makeCopies**(Classifier **arg0**, int **arg1**) throws java.lang.Exception
- public static Classifier **makeCopy**(Classifier **arg0**) throws java.lang.Exception
- public static **NUM_DECIMAL_PLACES_DEFAULT**
- public String **numDecimalPlacesTipText**()
- public void **postExecution**() throws java.lang.Exception
- public void **preExecution**() throws java.lang.Exception
- public void **run**(java.lang.Object **arg0**, java.lang.String[] **arg1**) throws java.lang.Exception
- public static void **runClassifier**(Classifier **arg0**, java.lang.String[] **arg1**)
- public void **setBatchSize**(java.lang.String **arg0**)
- public void **setDebug**(boolean **arg0**)
- public void **setDoNotCheckCapabilities**(boolean **arg0**)
- public void **setNumDecimalPlaces**(int **arg0**)
- public void **setOptions**(java.lang.String[] **arg0**) throws java.lang.Exception