Modeling assignment: Classification using decision trees

Hector Lopez CAP6673 FAU SPRING 2018

Introduction

A decision tree classifier for prediction can be used for a wide range of applications. This assignment uses the same fit and test input data as before but adds a class attribute for the fault-prone and non-fault-prone records. The decision tree will be built using the J48 (C4.5) classification algorithm. The Weka tool will be used to create the models and run the experiments. This report will describe the necessary background for model creation and evaluate the results by comparatively analyzing the models' performance.

Background

Decision Tree Classification

A decision tree is a method of representing the relationship between each attribute in a dataset by defining the relationships as rules to achieve a result for a classification attribute. Another way to describe it is as a tree starting with and root node and branching out using rules that are decisions that need to be met to branch to other leaf nodes. The class is thus defined at each leaf node.

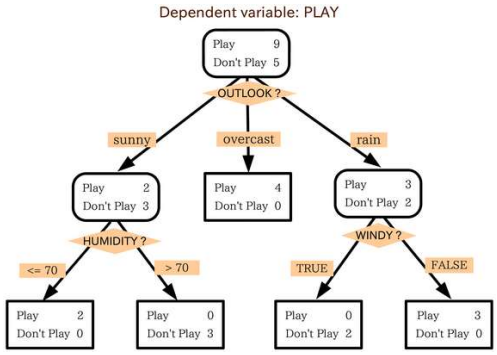


Figure 1: An example decision tree

J48(C4.5) Classification Algorithm

We will use Weka to create a decision tree with the C4.5 classification algorithm. The algorithm was created by Ross Quinlan in 1993. It is a recursive divide-and-conquer process aiming for the smallest possible tree. These are the general steps the algorithm performs:

1. Select the root node attribute
2. Select a value for the attribute for each branch
3. If the branch ends in a "pure" node, i.e. all the same class , then create a leaf node, else select a new node attribute and repeat until some threshold or all leaves are pure (I.e. all instances are used)

The selection of the root node is very important in the creation of the decision tree. With a different root node, the selections could result in a much larger tree. The root node gets selected by considering the entropy and information gain. Entropy is the measure of impurity or uncertainty of attributes. Information Gain is a means to find the most informative attribute. The goal is to minimize entropy and maximize the information gain.

Information gain is a probability of independent events. The probability of tossing two fair coins and getting heads is 50% on one coin and 50% on the other coin. We can also see that a single coin getting heads twice in a row would then be 25%. Both cases assume variable independence. Let's consider an unfair coin toss resulting in the 80% probability of getting heads for each coin, similarly one of these coins tossed twice would result in a 64% probability of getting heads twice in a row. The example shows how the increase in the probability of an outcome allow us to have more information about what will happen and what is currently happening. The fair coins have a variability that results in no "actionable" information so we can consider as "random" chance. If that randomness is skewed, we now have an indicator that there is a measurable bias in the event. When selecting the attributes, a similar processed is used to determine the amount of information the attribute can provide. Let's define the information as the log of the probability, p, of an event, x, and the expected information as the average of the information over all the events. The entropy is then the level of uncertainty in this information. We can see the relationship between entropy and information in the formula below:

$$entropy(x) = \sum_x p_x info(x) = -\sum_x p_x log_2(p_x)$$

To choose the root node attribute the information gain needs to be calculated. The information gain is the delta between the expected information of all attributes before the split and the expected information after the split.

Model Evaluation

For classification problems there are two types of errors: false positives and false negatives. The false positive rate (FPR), or Type I error, is the count of those negative class values misclassified in the positive class; whereas the false negative rate (FNR), or Type II error, is the count of those positive class values misclassified in the negative class. A balanced model is preferred where Type I and Type II errors are approximately equal with the FNR being as low as possible. Of course, if the FNR is 20% and the FPR is 20% in model A and the FNR is 20% and the FPR is 5% in model B, even though model A is "balanced," model B has much lower overall error and lower FNR thus model B should be chosen so some discretion is advised. The confusion matrix displays the number of correct and incorrect predictions made by the classification model compared to the actual data, with the positive diagonal being the misclassifications.

Confusion Matrix		Predicted Class			
		Positive	Negative		
Actual Class	Positive	a	b	Recall	a/(a+b)
	Negative	c	d	Specificity	d/(c+d)
		Positive Predictive Value (Precision)	Negative Predictive Value		
		a/(a+c)	d/(b+d)		

Table 2 – Confusion Matrix

Definitions		
Precision	a/(a+c)	proportion of positive cases predicted correctly
Negative Predictive Value	d/(b+d)	proportion of negative cases predicted correctly
Recall	a/(a+b)	proportion of actual positives correct
Specificity	d/(c+d)	proportion of actual negative correct
False Positive Rate (FPR)	c/(c+d)	misclassify negative in positive class, e.g. nfp classified as fp
False Negative Rate (FNR)	b/(a+b)	misclassify positive in negative class, e.g. fp classified as nfp
Error Rate	(c+b)/(a+b+c+d)	proportion of total number of incorrect predictions
Accuracy	(a+d)/(a+b+c+d)	proportion of total number of correct predictions
or	(1-Error Rate)	

Models & Results

Part 1

Build a pruned decision tree classification model using J48 (C4.5) using the fit data set and 10-fold cross validation. Follow by validating model with test dataset.

Fit Data

Misclassification error rates (%)

- Type I (FPR): 20%
- Type II (FNR): 9%

Confusion Matrix

[Predicted Class]
a(nfp) b(fp)
---

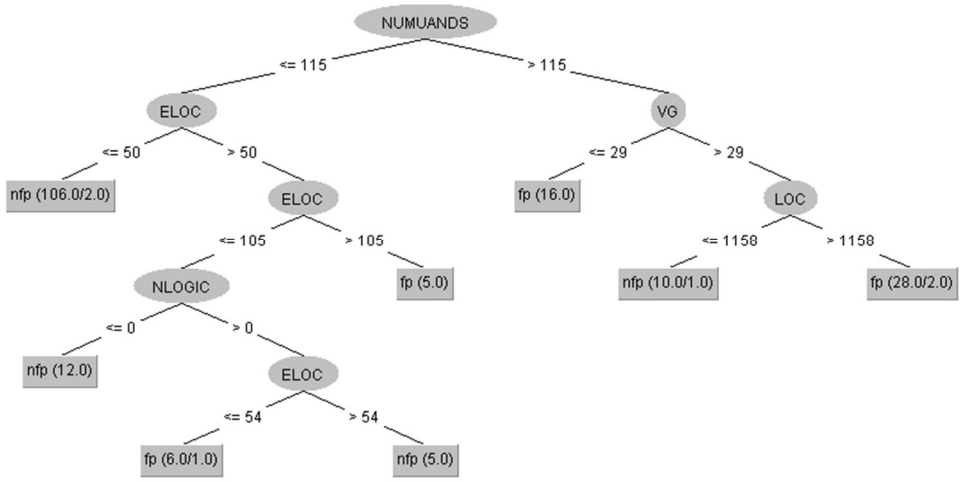
121 12 | a = nfp (Positive Class) [Actual Class]
11 44 | b = fp (Negative Class)

Performance Measurements

Specificity	.8
Recall	.91
Precision	.917
Negative Predicted Value	.786

False Negative Rate	.09
False Positive Rate	.200
Error Rate	.122
Accuracy	.878

Plot of Tree



Test Data

Misclassification error rates (%)

- Type I (FPR): 32.1%
- Type II (FNR): 7.6%

Confusion Matrix

[Predicted Class]
a(nfp) b(fp)
---
61 5 | a = nfp (Positive Class) [Actual Class]
9 19 | b = fp (Negative Class)

Performance Measurements

Specificity	.679
Recall	.924
Precision	.871
Negative Predicted Value	.792
False Negative Rate	.076
False Positive Rate	.321
Error Rate	.149
Accuracy	.851

Part1: Performance Analysis

The fitted, pruned decision tree model is accurate but unbalanced between the FPR and FNR since it has a high FNR. Validation with the test dataset continues this trend and shows worse results. Additional pruning and/or cost adjustments should be made to produce a smaller tree with less classification errors.

Part1: Data Analysis

Let's interpret the data from our tree and see what it is telling us. The "NUMUANDS" attribute was selected as the root node. This means this attribute as the root provided the decision tree with the most information gain. The number of unique operands, or the number of unique values to perform operations on would make sense as the attribute that would provide the most amount of information. This attribute would define the amount of operations and the likelihood that one of the unique values can lead to a fault. It then stands to reason that the complexity measurement "VG" and executable lines of code "ELOC" are branches stemming from the amount of unique values to computer. The greater the amount of unique values to compute the greater the complexity gets and then the "VG" would increase. The next insight the tree is providing is that when there are low amounts of unique values to compute the executable lines of code "ELOC" is an important attribute to classify the non-fault prone and the fault prone instances.

Part 2

Rebuild the model as an unpruned decision tree in the same way as Part 1 and repeat all the steps.

FIT data

Misclassification error rates (%)

- Type I (FPR): 20%
- Type II (FNR): 9%

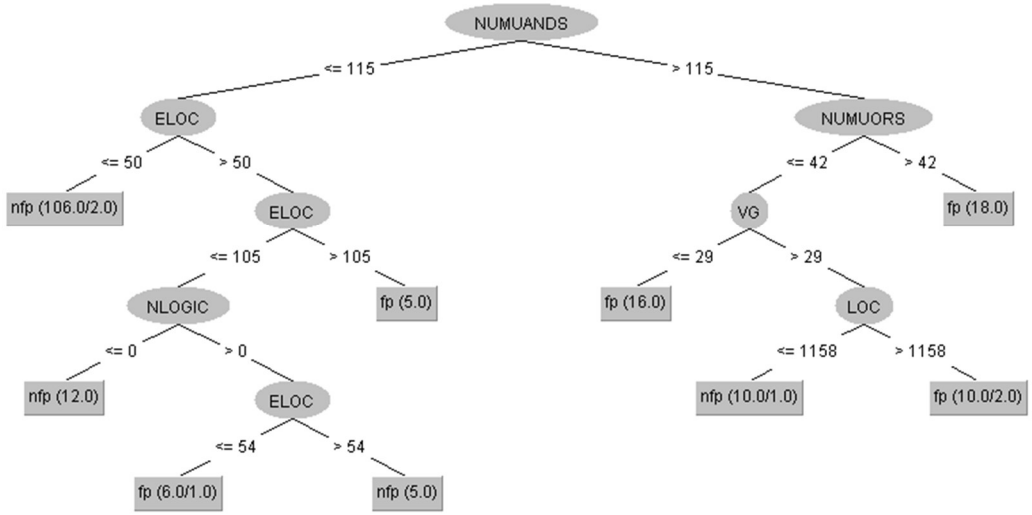
Confusion Matrix

```
[Predicted Class]
a(nfp)   b(fp)
---------------------
121              12 |   a = nfp (Positive Class)      [Actual Class]
11               44 |   b = fp (Negative Class)
```

Performance Measurements:

Specificity	.8
Recall	.91
Precision	.917
Negative Predicted Value	.786
False Negative Rate	.09
False Positive Rate	.200
Error Rate	.122
Accuracy	.878

Plot of Tree:



Test Data

Misclassification error rates (%)

- Type I (FPR): 28.6%
- Type II (FNR): 7.6%

Confusion Matrix

```
[Predicted Class]
a(nfp)   b(fp)
----------------------
61          5   |   a = nfp (Positive Class)      [Actual Class]
8          20   |   b = fp (Negative Class)
```

Performance Measurements

Specificity	.714
Recall	.924
Precision	.884
Negative Predicted Value	.800
False Negative Rate	.286
False Positive Rate	.076
Error Rate	.138
Accuracy	.862

Part2: Performance Analysis - Pruned and Unpruned Tree Comparison

The pruning process that the J48 algorithm uses was removed. By setting the “unpruned” value to TRUE in the Weka options. The unpruned tree is larger , so it is more prone to overfitting, and it is the same performance as the pruned tree. The pruned tree removed the NUMUORS nodes from under the

NUMUANDS node having all the values ending at the fp leaf under the LOC node. This shifted 18 correct classifications from the unpruned NUMUORS fp leaf node to the pruned fp LOC leaf node. The NUMUORS sub-tree was removed in the pruned tree. The below is a side-by-side comparison of the trees showing the pruned nodes in red for the unpruned tree. Where those values were assigned in the pruned tree is shown in green. Note that the overall delta between the two models is 0.

J48 pruned tree (Part 1)	J48 unpruned tree (Part 2)	
----------	----------	
(188, 6)	(188, 6)	delta: (0, 0)
NUMUANDS <= 115	NUMUANDS <= 115	
| ELOC <= 50: nfp (106.0/2.0)	| ELOC <= 50: nfp (106.0/2.0)	
| ELOC > 50	| ELOC > 50	
| | ELOC <= 105	| | ELOC <= 105	
| | | NLOGIC <= 0: nfp (12.0)	| | | NLOGIC <= 0: nfp (12.0)	
| | | NLOGIC > 0	| | | NLOGIC > 0	
| | | | ELOC <= 54: fp (6.0/1.0)	| | | | ELOC <= 54: fp (6.0/1.0)	
| | | | ELOC > 54: nfp (5.0)	| | | | ELOC > 54: nfp (5.0)	
| | ELOC > 105: fp (5.0)	| | ELOC > 105: fp (5.0)	
NUMUANDS > 115	NUMUANDS > 115	
	| NUMUORS <= 42	
| VG <= 29: fp (16.0)	| | VG <= 29: fp (16.0)	
| VG > 29	| | VG > 29	
| | LOC <= 1158: nfp (10.0/1.0)	| | | LOC <= 1158: nfp (10.0/1.0)	
| | LOC > 1158: fp (28.0/2.0)	| | | LOC > 1158: fp (10.0/2.0)	
	| NUMUORS > 42: fp (18.0)	
Number of Leaves: 8	Number of Leaves: 9	
Size of the tree: 15	Size of the tree: 17	

Part2: Data Analysis

The pruned and unpruned trees have one distinction the NUMUORS node. This node is the number of unique operators attribute. The tree is indicating the number of unique operators may not be significant if all in determining the fault prone or non-fault prone class. The use of different operations could be ubiquitous so that its well distributed among many programs. This would remove its ability to provide any new information or any actionable bias.

Part 3

Set the confidence factor (C) from 0.25 to 0.01 and rebuild the model in the same way as the initial tree in Part 1 and repeat all the steps.

FIT data

Misclassification error rates (%)

a. Type I (FPR): 27.3%
b. Type II (FNR): 9.8%

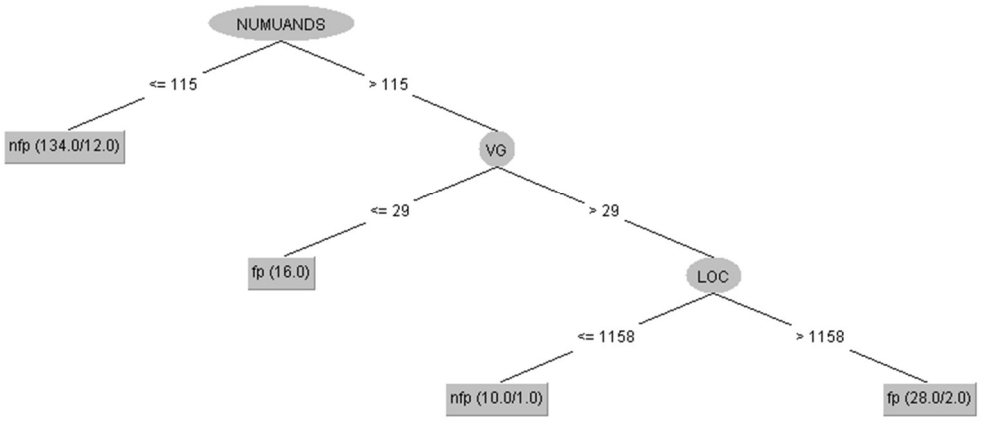
1. Confusion Matrix

[Predicted Class]
a(nfp) b(fp)
---
120 13 | a = nfp (Positive Class) [Actual Class]
15 40 | b = fp (Negative Class)

2. Performance Measurements:

a.	Specificity	.727
b.	Recall	.902
c.	Precision	.889
d.	Negative Predicted Value	.755
e.	False Negative Rate	.098
f.	False Positive Rate	.273
g.	Error Rate	.149
h.	Accuracy	.851

3. Plot of Tree:
    a. Nodes = 3
    b. Leaves = 4



TEST data

4. Misclassification error rates (%)
    a. Type I (FPR): 39.3%
    b. Type II (FNR): 6.1%

5. Confusion Matrix

[Predicted Class]
a(nfp) b(fp)
---
62 4 | a = nfp (Positive Class) [Actual Class]
11 17 | b = fp (Negative Class)

6. Performance Measurements:

a.	Specificity	.607
b.	Recall	.939
c.	Precision	.849
d.	Negative Predicted Value	.810
e.	False Negative Rate	.061
f.	False Positive Rate	.393
g.	Error Rate	.150
h.	Accuracy	.840

Part 3: Performance Analysis – Confidence Factor Adjustments

By decreasing the confidence factor the tree can lower its performance but also trim any nodes that only give marginal improvement. A smaller tree allows for a more robust model since it is not overfitted to rare or unique constraints that would decrease its performance in the face of a more generalized datasets. We do not want to build a model that conforms to the fit data provided but a general model that can handle any new data well. The size has dropped by trimming the entire NUMUANDS <= 115 decision path. The changes to the confidence factor from .25 to .01 has seem to have dropped the model accuracy in training from 86% to 84% respectively. The validation of each model with the test data shows a ~1% delta in accuracy between the two trees. With such a small difference in accuracy and the improved model the confidence value of .01 is a much better setting for the decision tree classification. In review, when the confidence is low in the training data set a lower confidence factor can increase the error estimates per node, and increase the uncertainty in the nodes classification error and then finally resulting in that node being pruned.

Part 3: Data Analysis

Looking at the more robust model we can see the weight of the executable lines of code in our assessment of the classification. Notice that the fault prone instances are insignificant on the branch that was trimmed. This makes overall sense since the complexity of these programs is not large and the "VG" attribute is the most intuitive way to show that with more complexity comes more chances for faults to occur. The number of executable lines of code may show faults due to fatigue or human error which would make sense is less likely to occur than faults during comprehension or doing mental acrobatics.

Part 4

Use the cost sensitive classifier combined with J48 (C4.5) to determine the optimal cost ratio by setting the cost of a Type I error to one and varying the cost of the Type II error.

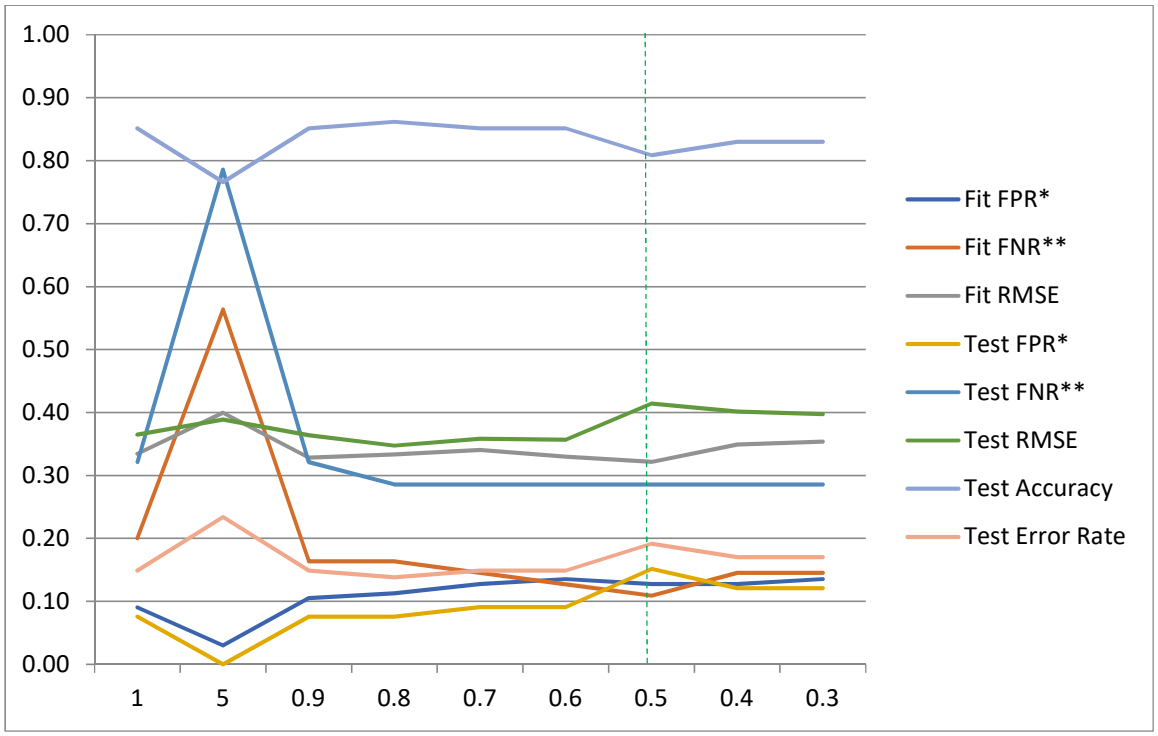
1) Observe the trends in the misclassification rates. What happens when the cost of a Type II error decreases/increases? Evaluate all the models on the fit and test datasets (with the same assumptions as in Part 1).

   The Type II error was adjusted while keeping the Type I error fixed at one. The use of one for both the Type I and Type II errors gives the same results as the training and validation runs in Part 1. The adjustment of the Type II error cost pushes the model farther away from or closer to balancing the FPR and FNR percentages.

The table and line graph show the results of adjusting the Type II error cost for both model training (fit data) and validation(test data). Going above a cost of 1 increases the Type II error rate while decreasing the Type I error; going below a cost of 1 decreases the Type II error while increasing the Type I error. This makes sense giving the symmetric relationship between FNR and FPR . The most balanced value was found to be where the Type II error cost equaled 0.5 creating the best balance of FPR and FNR while keeping the FNR low. Given the choice of the 0.5 Type II error cost, running the same error cost values on the validation training datasets showed that this 0.5 value is the most balanced between FPR and FNR. Model accuracy and error rate at c = 0.5 was higher than most of the other cost values for the validation which may be chance due to the test dataset.

	MODEL FIT DATA			**MODEL TEST DATA**				
TYPE II ERROR COST ADJ[3]	FPR[1]	FNR[2]	RMSE	FPR[1]	FNR[2]	RMSE	Accuracy	Error Rate
1	0.09023	0.20000	0.33436	0.07576	0.32143	0.36477	0.85110	0.14890
5	0.03008	0.56364	0.39968	0.00000	0.78571	0.38902	0.76600	0.23400
0.9	0.10526	0.16364	0.32848	0.07576	0.32143	0.36399	0.85110	0.14890
0.8	0.11278	0.16364	0.33362	0.07576	0.28571	0.34739	0.86170	0.13830
0.7	0.12782	0.14545	0.34038	0.09091	0.28571	0.35826	0.85110	0.14890
0.6	0.13534	0.12727	0.33004	0.09091	0.28571	0.35695	0.85110	0.14890
0.5	0.12782	0.10909	0.32160	0.15152	0.28571	0.41413	0.80850	0.19150
0.4	0.12782	0.14545	0.34911	0.12121	0.28571	0.40139	0.82980	0.17020
0.3	0.13534	0.14545	0.35389	0.12121	0.28571	0.39730	0.82980	0.17020

[1]False Positive Rate (FPR) - Type I error, nfp classified as fp
[2]False Negative Rate (FNR) - Type II error, fp classified as nfp
[3]Adjusting the Type II cost only with Type I being one



Appendix

Data Conditioning:

```
import pandas as pd


#classifyDataset('test');

def classifyDataSet (filename):
    df=pd.read_csv(filename+"_data.csv",header=None)

    def converter(x):
        if int(x) >= 2 :

            return "fp"
        else :
            print(x)
            return "nfp"

    df1 = df[8].apply(converter)
    df[8] = df1[1];
    df.to_csv(filename+'_data_classified.arff', sep=',',header=None,index=False)
    print(df);


classifyDataSet('fit');
```