Laboratory Module 3
Classification with Decision Trees

Purpose:

- Understand how to build simple baseline models for classification;
- Understand how to build decision trees for classification;
- Understand how different parameters for decision tree algorithms affect their output;
- Assess the accuracy of several models using cross-validation;
- Communicate the information captured in the decision tree model in well written English.

1. Theoretical Aspects

Classification is one of the major data mining tasks. Although this task is accomplished by generating a predictive model of data, interpreting the model frequently provides information for discriminating labeled classes in data. Decision trees provide a predictive model that is easy to interpret to provide a description of data.

In order to believe any predictive model, the accuracy of the model must be estimated. Several methods for evaluating the accuracy of models will be discussed during class lectures. For this assignment, 10-fold cross validation will be used for model assessment.

The data mining task you are to perform is to provide descriptions of acceptable and unacceptable labor contracts contained in *labor.arff*. You are to back up you description by the evidence you collect by building decision tree models of the data.

1.1 Entropy

Putting together a decision tree is all a matter of choosing which attribute to test at each node in the tree. We shall define a measure called information gain which will be used to decide which attribute to test at each node. Information gain is itself calculated using a measure called entropy, which we first define for the case of a binary decision problem and then define for the general case.

Given a binary categorization, **C**, and a set of examples, **S**, for which the proportion of examples categorized as positive by **C** is $p+$ and the proportion of examples categorized as negative by **C** is *p-*, then the **entropy** of **S** is:

$$Entropy(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

The reason we defined entropy first for a binary decision problem is because it is easier to get an impression of what it is trying to calculate.

Given an arbitrary categorization, **C** into categories **c1, ..., cn**, and a set of examples, **S**, for which the proportion of examples in $\mathbf{c_i}$ is $\mathbf{p_i}$, then the entropy of **S** is:

$$Entropy(S) = \sum_{i=1}^{n} -p_i \log_2(p_i)$$

This measure satisfies our criteria, because of the -p*log2(p) construction: when p gets close to zero (i.e., the category has only a few examples in it), then the log(p) becomes a big negative number, but the p part dominates the calculation, so the entropy works out to be nearly zero. Remembering that entropy calculates the disorder in the data, this low score is good, as it reflects our desire to reward categories with few examples in. Similarly, if p gets close to 1 (i.e., the category has most of the examples in), then the log(p) part gets very close to zero, and it is this which dominates the calculation, so the overall value gets close to zero. Hence we see that both when the category is nearly - or completely - empty, or when the category nearly contains - or completely contains - all the examples, the score for the category gets close to zero, which models what we wanted it to. Note that 0*ln(0) is taken to be zero by convention.

1.2 Information gain

We now return to the problem of trying to determine the best attribute to choose for a particular node in a tree. The following measure calculates a numerical value for a given attribute, **A**, with respect to a set of examples, **S**. Note that the values of attribute **A** will range over a set of possibilities which we call **Values(A)**, and that, for a particular value from that set, ***v***, we write ***Sv*** for the set of examples which have value ***v*** for attribute $\boldsymbol{A}$.

The information gain of attribute $\boldsymbol{A}$, relative to a collection of examples, $\boldsymbol{S}$, is calculated as:

$$Gain\ (S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

The information gain of an attribute can be seen as the expected reduction in entropy caused by knowing the value of attribute $\boldsymbol{A}$.

1.3 Sample calculation on Entropy and Information gain

Instances:

Weekend	Weather	Parents	Money	Decision (Category)
W1	Sunny	Yes	Rich	Cinema
W2	Sunny	No	Rich	Tennis
W3	Windy	Yes	Rich	Cinema
W4	Rainy	Yes	Poor	Cinema
W5	Rainy	No	Rich	Stay in
W6	Rainy	Yes	Poor	Cinema
W7	Windy	No	Poor	Cinema
W8	Windy	No	Rich	Shopping
W9	Windy	Yes	Rich	Cinema
W10	Sunny	No	Rich	Tennis

The first thing we need to do is work out which attribute will be put into the node at the top of our tree: **weather, parents or money**. To do this, we need to calculate:

Entropy(S) = $-p_{cinema} \log_2(p_{cinema}) - p_{tennis} \log_2(p_{tennis}) - p_{shopping} \log_2(p_{shopping}) - p_{stay\_in} \log_2(p_{stay\_in})$
= $-(6/10) * \log_2(6/10) - (2/10) * \log_2(2/10) - (1/10) * \log_2(1/10) - (1/10) * \log_2(1/10)$
= $-(6/10) * -0.737 - (2/10) * -2.322 - (1/10) * -3.322 - (1/10) * -3.322$
= $0.4422 + 0.4644 + 0.3322 + 0.3322 =$ **1.571**

and we need to determine the best of:

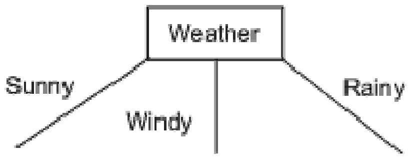
Gain(S, weather) = $1.571 - (|S_{sun}|/10)*\text{Entropy}(S_{sun}) - (|S_{wind}|/10)*\text{Entropy}(S_{wind}) - (|S_{rain}|/10)*\text{Entropy}(S_{rain})$
= $1.571 - (0.3)*\text{Entropy}(S_{sun}) - (0.4)*\text{Entropy}(S_{wind}) - (0.3)*\text{Entropy}(S_{rain})$
= $1.571 - (0.3)*(0.918) - (0.4)*(0.81125) - (0.3)*(0.918) = 0.70$

Gain(S, parents) = $1.571 - (|S_{yes}|/10)*\text{Entropy}(S_{yes}) - (|S_{no}|/10)*\text{Entropy}(S_{no})$
= $1.571 - (0.5) * 0 - (0.5) * 1.922 = 1.571 - 0.961 = 0.61$

Gain(S, money) = $1.571 - (|S_{rich}|/10)*\text{Entropy}(S_{rich}) - (|S_{poor}|/10)*\text{Entropy}(S_{poor})$
= $1.571 - (0.7) * (1.842) - (0.3) * 0 = 1.571 - 1.2894 = 0.2816$

This means that the first node in the decision tree will be the weather attribute. As an exercise, convince yourself why this scored (slightly) higher than the parents attribute - remember what entropy means and look at the way information gain is calculated.

From the weather node, we draw a branch for the values that weather can take: sunny, windy and rainy:



Finishing this tree off is left as a exercise.

2. Decision Tree Induction Algorithms

2.1 ID3 (Iterative Dichotomiser 3) Algorithm

The ID3 algorithm can be summarized as follows:

1. Take all unused attributes and count their entropy concerning test samples
2. Choose attribute for which entropy is maximum
3. Make node containing that attribute

The actual algorithm is as follows:

```
ID3 (Examples, Target_Attribute, Attributes)
    * Create a root node for the tree
    * If all examples are positive, Return the single-node tree Root, with label = +.
    * If all examples are negative, Return the single-node tree Root, with label = -.
    * If number of predicting attributes is empty, then Return the single node tree Root, with
      label = most common value of the target attribute in the examples.
    * Otherwise Begin
        o A = The Attribute that best classifies examples.
        o Decision Tree attribute for Root = A.
        o For each possible value, vi, of A,
            + Add a new tree branch below Root, corresponding to the test A = vi.
            + Let Examples(vi), be the subset of examples that have the value vi for A
            + If Examples(vi) is empty
                # Then below this new branch add a leaf node with label = most common target value in the examples
            + Else below this new branch add the subtree ID3 (Examples(vi), Target_Attribute, Attributes – {A})
    * End
    * Return Root
```

2.2 C4.5 Algorithm

C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set S = s1,s2,... of already classified samples. Each sample si = x1,x2,... is a vector where x1,x2,... represent attributes or features of the sample. The training data is augmented with a vector C = c1,c2,... where c1,c2,... represent the class to which each sample belongs.

At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the smaller sublists.

This algorithm has a few base cases.

* All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.

* None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.

* Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

In pseudocode the algorithm is:

```
1. Check for base cases
2. For each attribute a
    1. Find the normalized information gain from splitting on a
3. Let a_best be the attribute with the highest normalized information gain
4. Create a decision node that splits on a_best
5. Recurse on the sublists obtained by splitting on a_best, and add those nodes as children of node
```

3. Build Baseline Classification Models

A baseline model is one that can be used to evaluate the success of your target model, in this case a decision tree model. Baseline models are typically simple, an inaccurate, but occasionally data is so sim ple to describe, attempting to use a complex model results in worse behavior than a simple model.

In this section, you will build and record the accuracy to two baseline modes: ZeroR and OneR. The ZeroR model simply classifies every data item in the same class. For example, a ZeroR model may classify all loan applications as high risk without even considering the attributes of each data instance. The OneR model seeks to generate classification rules using a single attribute only.

1. Start Weka. On studsys, this can be accomplished with the command:
java -jar …/weka-3-2-3/weka.jar
2. Open Weka's Explorer interface.
3. Open *labor.arff* from the Explorer interface.
You may also want to open this file in a text editor.
4. Typically, at this point, you would record a summary of your data.
5. Click on the Classify tab in the Weka window.
6. The ZeroR classifier should be selected, but if it is not, click on the rectangle underneath the word Classifier and select the ZeroR classifier from the menu that pops up.
7. Select the Cross-validation radio button from the Test options. Use 10 folds.
8. Make sure that the class attribute is selected as the classification label.
9. Click the Start button to build and evaluate the model. Record the results in the output window.
10. Repeat the process, but select the OneR classifier from the Classifier menu using the default minimum bucket size of 6.

4. Generating Decision Trees (example 1)

Once you generate your baseline models and estimate their accuracy, you can create the target model of interest. Even though you will be creating a decision tree model, the algorithm you will use has several parameters. You need to modify the parameters in order to generate a reasonable model. Each time you modify a parameter, you might end up with a different model.

1. Before generating a decision tree, set accuracy goals for you classifier based on the accuracy of your baseline models.
2. Select the J48 algorithm for creating decision trees from the Classifier menu. Don't use the PART version.
3. Generate decision trees for the following combinations of attributes: all false, binarySplits only true, reducedErrorPruning only true, subtreeRaising only true, unpruned only true, useLaplace only true. Record the resulting decision trees and their associated accuracy and error information.
4. Generate decision trees with some combinations of the boolean parameters. Note that if you use reducedErrorPruning, the value of subtreeRaising is ignored. Again, record the results of each trial.
5. Now, for your best model so far, attempt to find a combination of confidenceFactor, minNumObj, numFolds that will improve your results. Use your understanding of the theory and documentation to make good selections. Record the results of each attempt.

5. Generating Decision Trees (example 2)

1. Start Weka. On studsys, this can be accomplished with the command:

java -jar …/weka-3-2-3/weka.jar

2. Open Weka's Explorer interface.
3. Open *credit.arff* from the Explorer interface.

You may also want to open this file in a text editor.

4. Run a decision tree classifier over the data by selecting classifiers > trees > J48 under the Classify tab.
5. Set a confidenceFactor of 0.2 in the options dialog.
6. Use a test percentage split of 90%.

Observe the output of the classifier. The full decision tree is output for your perusal; you may need to scroll up for this. The tree may also be viewed in graphical form by right-clicking the run in the Result list at the bottom-left and selecting Visualize tree, although it may be very cluttered for large trees.

- How would you assess the performance of the classifier? Hint: check the number of good and bad cases in the test sample (e.g. using the confusion matrix)
- Looking at the decision tree itself, are the rules it applies sensible? Are there any branches which appear absurd?
- What is the effect of the *confidenceFactor* option? Try increasing or decreasing the value of this option and observe the results.

For comparison, we also learn a decision stump. This is a decision tree with only a single node:

- Select the *DecisionStump* classifier.
- Select *Cross-validation* with 10 folds for the test option.

Now build the classifier, and observe the results:

- What single attribute does the algorithm use to make its decision? Do you expect this to be useful in its own right?
  Hint: visualisation could assist here.
- How do the results compare to that of the J48 tree? Is this what we would expect?
- Is the stump actually discriminating between anything?
  Hint: return to the Preprocess tab and observe the distribution of the Approve attribute. If 70% of applications are approved and 30% are not, how do we make a classifier that is 70% accurate?

6. Assignments

1. What are acceptable and unacceptable labor contracts according to your results?
2. Compare and contrast the pruned and unpruned trees you generated.
3. Do you feel your best tree is overfitting the data? Why or why not?
4. Which class is generally better recognized by the decision trees?
5. Decision trees are limited in the kinds classification problems they can solve

Can you find evidence that would lead you to believe other classification techniques would perform worse, better, or equally as well on the labor data?