Lab Report: Using a Decision Tree to Classify Images of Printed Lower-Case Letters

Univ. of Utah: Srivishnu Kaushik Satyavolu
Univ. of Utah: 04 October 2012

1 Introduction

This lab explores the use of a Decision Tree (DT) to classify images of lower-case printed characters. Questions of interest include:

1. Can a DT perform this task?
2. What are a good set of features?
3. What are the effects of using entropy on the DT?
4. Is entropy-based selection a better way than random selection for choosing the features at each level to build the DT?
5. What is a good way to select training and testing data?
6. Which performs better for this task? DT or RBF?

2 Method

A DT is characterized by the number of nodes in it, number of levels and the attributes/features used at each level inorder to classify the input. The final attributes that are used in the DT are a proper subset of the attributes/features that are chosen during the training of the DT. We show in this document how we can obtain a bunch of low-level, mid-level and high-level features to obtain a fairly good classifier using DT.

2.1 Features

2.1.1 Euler Number

This attribute is computed using the `bweuler` function in Matlab as follows:

\[
\text{Euler Number} = \text{number of objects in the image} - \text{total number of holes in the objects}
\]

2.1.2 Number of Full Rows

This attribute is computed as follows:

\[
\text{feature} = \text{number of fully filled rows}
\]

2.1.3 Number of Empty Rows

This attribute is computed as follows:

\[
\text{feature} = \text{number of empty rows}
\]

2.1.4 Number of Full Columns

This attribute is computed as follows:

\[
\text{feature} = \text{number of fully filled columns}
\]
2.1.5 Number of Empty Columns
This attribute is computed as follows:

\[ \text{feature} = \text{number of empty columns} \]

2.1.6 Top or Bottom?
This attribute is a boolean that decides where more number of 1s occur (in a binary image) among the top and bottom halves of the pixels. And is computed as:

\[ \text{feature} = \text{count of 1s in the upper half} \neq \text{count of 1s in the lower half} \]

2.1.7 Left or Right?
This attribute is a boolean that decides where more number of 1s occur (in a binary image) among the left and right halves of the pixels. And is computed as:

\[ \text{feature} = \text{count of 1s in the left half} \neq \text{count of 1s in the right half} \]

2.1.8 Count of protruding terminal pixels
This feature essentially decides how many pixels in the image are terminal. That is, it gives the count of pixels, for which, the number of 1s in its $3 \times 3$ neighborhood is 2 (including itself). Thus, it is computed as follows:

\[ \text{feature} = \text{pixels with } 3 \times 3 \text{ neighborhood counts 2} \]

2.1.9 Count of island pixels
This feature essentially decides how many pixels in the image are isolated. That is, it gives the count of pixels, for which, the number of 1s in its $3 \times 3$ neighborhood is 1 (including itself). Thus, it is computed as follows:

\[ \text{feature} = \text{pixels with } 3 \times 3 \text{ neighborhood counts 1} \]

2.2 Code (with annotations)
This section gives the code that is used for computing the above features and conducting a series of experiments using DT.

```
function [X,percent_correct,nodes,run] = DT_Main
% DT_Main - Trains, runs and tests a DT based on the available data set
% On output:
% X: m*n by p matrix with data items for its rows and features
% as its columns
% percent_correct: array of K=9 values showing the percentage of correct
% values for each of the 9 disjoint test sets
% nodes: nodes making up the DT
% run: a collection of statistics for each run such as, average gain and
% maximum tree depth
% Call:
% [X,percent_correct,nodes,run] = DT_Main;
```
% Author:
% Srivishnu Kaushik Satyavolu
% Fall 2012
% University of Utah

% Original Author:
% Prof. Tom Henderson
% Fall 2012
% University of Utah

%GLOBAL VARIABLES: BAD IDEA!!! So clear these two every time you use them
global nodes num_nodes

[X,T,attributes] = CS6350_DT_image_features_mod;

m = 26;
n = 9;
p = size(X,2);

% Data is divided into 9 disjoint sets. At each run, 8 of these sets will form
% the training set while 1 will be the test set.
% n_total_sets = n;
% n_training_sets = n-1;
% n_testing_sets = 1;

% These comprise of the entire 8 sets of the training data
train_X = zeros(n_training_sets*m,p+2);

% This is the final testing set
test_X = zeros(n_testing_sets*m,p+2);

percent_correct = zeros(n_total_sets,1);

% Perform every experiment for 9 different test sets
for current_test_set_num = 1:n_total_sets

num_nodes = 0;
nodes = [];
cur_level = 0;

% Initialize the indices
index = 0;
train_index = 1;
test_index = 1;

% Split into K=9 disjoint sets, 8 into training and 1 into testing
for class_num = 1:m
for sample_num = 1:n
index = index + 1;
if( sample_num == current_test_set_num )
test_X(test_index,1:p) = X(index,:);
test_X(test_index,p+1) = 1;
test_X(test_index,p+2) = class_num;
test_index = test_index + 1;
end
end
end
else
    train_X(train_index,1:p) = X(index,:);
    train_X(train_index,p+1) = 1;
    train_X(train_index,p+2) = class_num;
    train_index = train_index + 1;
end
end

% Build the Decision Tree
MA = AI_decision_tree_learning(train_X, attributes, 1, ...
    'AI_choose_gain', 1);

% Collect the run statistics viz., average gains at each
% level (starting at 1) and the maximum level of the DT
run(current_test_set_num).gains = zeros(num_nodes-1,2);
run(current_test_set_num).max_level = 1;

for k = 1:num_nodes
    run(current_test_set_num).gains(nodes(k).level,1) = ...
        run(current_test_set_num).gains(nodes(k).level,1) + ...
        nodes(k).gain;
    run(current_test_set_num).gains(nodes(k).level,2) = ...
        run(current_test_set_num).gains(nodes(k).level,2) + ...
        1;
    if(run(current_test_set_num).max_level < nodes(k).level)...
        run(current_test_set_num).max_level = nodes(k).level;
    end
end

for cur_level = 1:run(current_test_set_num).max_level
    run(current_test_set_num).gains(cur_level,1) = ...
    run(current_test_set_num).gains(cur_level,1)/...
    run(current_test_set_num).gains(cur_level,2);
end

% Testing
    total_count = 0;

% Call Recall on the test data using the trained DT
for s = 1:size(test_X,1)
    y = CS5350_tree Decide(nodes,test_X(s,:));
    if y(1)==test_X(s,p+2)
        percent_correct(current_test_set_num) = ...
        percent_correct(current_test_set_num) + 1;
    end
    total_count = total_count + 1;
end

% Compute percentage correct
percent_correct(current_test_set_num) = ...
    (percent_correct(current_test_set_num)/total_count) * 100.0;
end
% File Name: CS6350_DT_image_features_mod.m
% Date: October 4th, 2012

function [X,T,attributes] = CS6350_DT_image_features_mod
% CS6350_DT_image_features_mod - Builds the feature vectors for the entire
% data set and also the attributes
% On output:
% X: m*n by p matrix with data items for its rows and features as its
% columns
% T: m*n by 1 array containing the desired target classification value
% for each data item
% attributes: set of attribute names, values and flags for inclusion

% Call:
% [X,percent_correct,nodes,run] = DT_Main;
% Author:
% Srivishnu Kaushik Satyavolu
% Fall 2012
% University of Utah

% Original Author:
% Prof. Tom Henderson
% Fall 2012
% University of Utah

letters = ['a','b','c','d','e','f','g','h','i','j','k','l','m',
'n','o','p','q','r','s','t','u','v','w','x','y','z'];
digits = ['1','2','3','4','5','6','7','8','9'];

num_im = 26*9;
res = eye(num_im,num_im);
X = zeros(num_im,9);
T = zeros(num_im,1);

count = 0;
for l = 1:26
    let = letters(l);
    for d = 1:9
        count = count + 1;
        dig = digits(d);
        filename = strcat('..\A1/A1\',let,dig,'.jpg');
        im = imread(filename);
        im = im>100;

        % Euler feature
        v1 = bweuler(im) + 1;

        [im_h,im_w] = size(im);
        im_half_h = floor(im_h/2);
        im_half_w = floor(im_w/2);

        % Full Row Count

        % ...
v2 = 0;
% Empty Row Count
v3 = 0;

for k = 1:im_h
    temp = sum(im(k,:));

    % Check if the number of 1s in a row is equal
    % to the column size of the image
    if temp == im_w
        v2 = v2 + 1;

    % Check if the row is empty
    elseif temp == 0
        v3 = v3 + 1;
    end
end

% Full Column Count
v4 = 0;
% Empty Column Count
v5 = 0;

for k = 1:im_w
    temp = sum(im(:,k));

    % Check if the number of 1s in a column is equal
    % to the row size of the image
    if temp == im_h
        v4 = v4 + 1;

    % Check if the row is empty
    elseif temp == 0
        v5 = v5 + 1;
    end
end

% Top or Bottom?
v6 = sum(sum(im(1:im_half_h,:))) > sum(sum(im(im_half_h+1:im_h,:)));
% Left or Right?
v7 = sum(sum(im(:,1:im_half_w))) > sum(sum(im(:,im_half_w+1:im_w)));

% Terminal Pixel Count
v8 = 0;

% Island Pixel Count
v9 = 0;

for i = 2:im_h-1
    for j = 2:im_w-1
        temp_sum = sum(sum(im(i-1:i+1,j-1:j+1)));  % Only change here
        if temp_sum == 1
            v8 = v8 + 1;
        elseif temp_sum == 2

        end
    end
end
v9 = v9 + 1;
end
end

% Attribute/feature vector
v = [v1, v2, v3, v4, v5, v6, v7, v8, v9];

% Total Data set
X(count,:) = v;

% Total Target set
T(count,:) = 1;
end
end

% Initialized attributes and their values to have unique values
% from each attribute column in the data set
for k = 1:size(X,2)
    attributes(k).name = strcat('Att',k);
    attributes(k).values = unique(X(:,k))';
    attributes(k).include = 1;
end

% Re-assign feature vectors as the indices of the above decided
% unique values for each attribute
for i = 1:size(X,1)
    for j = 1:size(X,2)
        X(i,j) = find(attributes(j).values == X(i,j))-1;
    end
end

% Compute the final attribute values as indices of the
% original attribute values
for k = 1:size(X,2)
    attributes(k).values = unique(X(:,k))';
end
end

The rest of the DT code used in matlab (which is not listed above) is given by Prof. Tom Henderson for CS5350/6350 Machine Learning course, Fall 2012, with some minor changes.

3 Verification of Program

3.1 Verification of Slope Change Characteristic Feature Calculation

In this section, the verification of the main part of the function CS6350_DT_image_features_mod.m that is responsible for feature calculation is provided.

For testing purposes, the following image is taken as input:

```
0100000010010100101000000
```

The following were the values obtained for each feature:
Euler Number = 2
Full Row Count = 0
Empty Row Count = 1
Full Column Count = 0
Empty Column Count = 3
Top or Bottom = 1
Left or Right = 0
Terminal Pixel Count = 4
Island Pixel Count = 1

The program produced exactly the same values as above as outputs. Hence, it is also verified.

4 Data

![DT Performance](image)

Figure 1: Figure showing general testing performance of DT for K=9 test runs

This section gives graphs that illustrate the general performance of the resultant DT. All the experiments in this section are carried out for K=9 test runs. And all the confidence intervals reported later are at t=1.860 with 90% confidence.

5 Analysis

Figure 1 shows the general performance of DT across multiple runs. One can clearly see that DT performs very well and stays pretty much across 100 except for an occasional dip to 92 (roughly) in the last run. This suggests that DT is suitable for the current classification task by a choice of good features, which in this case are a combination of low, medium and higher-level features.

Figure 2 shows how the average gains vary across multiple levels of the DT. A decreasing trend, as observed in the graph, is encouraging and is expected, because, the idea of using entropy is to choose features at each level that maximize the gain of the parent node, which is exactly what is shown in the graph. Hence, entropy is doing its job.
Figure 2: Figure showing the variation of Gain across multiple levels in the DT by averaging them across k=9 test runs.

Figure 3: Figure showing the difference in testing performance accuracy of entropy and random selection on the DT across k=9 test runs.

Figure 3 shows the comparison of entropy-based selection against a series of random runs (k=1 to 9). One can see that, both perform equally good in general and a general statement of which performs better can not be made.

Figure 4 shows the comparison of the tree depth for entropy based selection vs random selection. One can clearly notice that, the entropy based selection always chooses the optimal number of levels (5 in this case), while random selection, by choosing a random order of features, ends up having to use more levels.
Figure 4: Figure showing the effect of random selection vs entropy-based selection on a DT for k=9 test runs

Figure 5: Figure showing Difference in testing performance accuracy of DT and RBF for k=9 test runs

than necessary. Thus, entropy-based selection is always preferrable when tree depth is a concern. We can say that it outperforms random selection by a range of 3.7133 to 4.9533 (tree depth) with 90% confidence.

Figure 5 shows the performance of DT vs RBF for this specific feature set. One can see that RBF significantly underperforms than DT in this case. So, for this specific case, one can clearly conclude that the DT is better than RBF by a range of 25.4609 to 32.6587 with 90% confidence. However, without testing with more general data, no general statement should be made on which is better. Because, it might be that DT may not generalize well to perform on noisy data, while RBF might!
6 Interpretaion

The answers to the questions posed are as follows:

1. Can a DT perform this task?
   Yes. Very well.

2. What are a good set of features?
   A combination of a set of low-level, medium and high-level features served good for this purpose.

3. What are the effects of using entropy on the DT?
   Entropy-based selection, as expected, results in trees with smaller heights than other selections using the optimal number of features (usually just a subset of the total number of features).

4. Is entropy-based selection a better way than random selection for choosing the features at each level to build the DT?
   Performance-wise, the answer is No. But, if tree depth is a criterion, then, Yes.

5. What is a good way to select training and testing data?
   Selecting 8 parts of data set for training and 1 set for testing and doing permutations on them seemed like a good choice for the current purpose.

6. Which performs better for this task? DT or RBF?
   For this specific feature set, one can conclude that DT performs better than RBF. However, without more testing data (which is significantly different from the training data), no general statement can be made in this regard.

7 Critique

From the above results and analysis, one can see that a DT, in general can be tuned to perform very well by choosing a good set of features. Entropy based selection can be important when dealing with trees with a huge number of nodes. Because, in that case, the performance of tree is proportional to the depth of the tree. If more time was there, I would have loved to explore the DT vs RBF performance, trying to test the above results on limited training data, but significantly different testing data. Because, intuition tells that, in such a case, a RBF will generalize better than an equivalent DT.

8 Log

Time spent on coding the feature extraction, upgrading and finalizing the feature set:
   - October 1st, 2012 from 9 PM to 12 PM

Time spent on writing the report, generating the results, analyzing the various relationships, coding and conducting experiments:
   - October 3rd, 2012 from 10 PM to 1 AM
   - October 4th, 2012 from 7 AM to 10 AM

References