Data mining with the ID3 algorithm.

Jeroen Laros

July 18, 2003

Contents

1 Introduction.

This document is about an algorithm that tries to find search engines in the access log files of a web server. This could be used for example to give non-search engines a higher priority in order to reduce the time needed for interactive traffic, which might be usefull if your webserver is very busy.

2 Problem.

We have a couple of access log files, and a list of known search engines. We want to find out if a classification can be made in order to determine if a host is a search engine or not.

3 Approach.

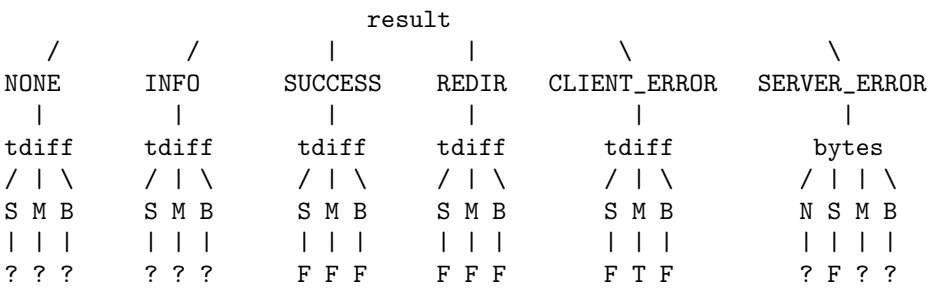
We use the ID3 algorithm to build a classification tree.

4 Implementation.

First we have to make sure all non-discrete values are converted to discrete ones in order to make the ID3 algorithm work. At the same time, we convert the access log file to a list of values, which can be parsed more easily by the ID3 algorithm. The program that does this conversions is included in the appendix under the name opg3.pl. Its output is dumped in a file called training.tbl. For more information about this file, see the appendix. The next step is to make a decision tree from these values, this is done by the ID3 algorithm.

5 Results.

The tree we got was:



The N stands for NONE, S for SMALL, M for MEDIUM and B for BIG. The T stands for TRUE and the F for false. If no further classification can be made because there is an equal amount of TRUE and FALSE attributes are marked with an '?'.

6 Conclusion.

Because we were the ones who gave the rules to convert the real-valued variables to discrete ones, this decision tree is not very good. We obviously made a couple of very wrong choices. A better result would be obtained if an other algorithm like C4.5 was used.

7 References.

For the assignment itself visit: http://www.liacs.nl/ kosters/AI/ For more information about the ID3 algorithm and related algorithms visit: http://yoda.cis.temple.edu:8080/UGAIWWW/lectures/C45/

8 Appendix.

```
The attributes of the training.tbl file:
Categorical:
isengine -: {TRUE, FALSE}

Non-categorical:
tdiff ----: {SMALL, MEDIUM, BIG}
action ---: {GET, HEAD}
url ------: {PLAIN, BIN}
result ---: {NONE, INFO, SUCCESS, REDIR, CLIENT_ERR, SERVER_ERR}
bytes ----: {NONE, SMALL, MEDIUM, BIG}

opg3.pl:
#!/usr/bin/perl -w

# Config variables.
$spiderfiles_location = "../spiders";
$logs_location = "../www";
@textlike_files = ("TXT", "HTM", "TEX", "PS", "DVI", "XML", "CGI", "DOC",
                   "PATCH", "PHP", "PDF");

# Global variables.
@spiders = ();
%results = ();

# Functions.

# This function reads all files in $spiderfiles_location and fills the array
# @spiders with the class C networks in these files.
#
# Arguments: none
```

```
# Returns: nothing
sub loadspiders {
  my @spiderfiles = `ls $spiderfiles_location`;
  my @temparr = ();
  my $t = 0;

  foreach (@spiderfiles) {
    open (SPIDER, "$spiderfiles_location/$_") or die "Can not open $_: $!";
    while (<SPIDER>) {
      if (/^(\d{1,3}\.){2}\d{1,3}/) {    # An ip address. (a.b.c)
        $_ =~ s/\s.*\s//;
        if (/^(\d{1,3}\.){3}\d{1,3}/) {  # A full ip address. (a.b.c.d)
          $_ =~ s/\.\d{1,3}\s//;         # Strip the class C network.
        }#if
        $temparr[$t] = $_;               # Put it in the temp array.
        $t += 1;
      }#if
    }#while
    close (SPIDER);
  }#foreach
  @temparr = sort @temparr;              # Sort the array.

  $t = 0;                                # Remove duplicates.
  $spiders[0] = $temparr[0];
  for (my $i = 0; $i < $#temparr; $i++) {
    if ($temparr[$i] ne $temparr[$i + 1]) {
      $t++;
      $spiders[$t] = $temparr[$i + 1];   # And put the result in @spiders.
    }#if
  }#for
}#loadspiders

# This function converts a date to seconds counting from the beginning of
# the month. Note that this might cause problems when this script is runned
# at the beginning of a month.
#
# Arguments: a string which contains a date  e.g: "1/Jan/1970:00:00:00 +0100"
# Returns: integer
sub date2sec {
  my $l = shift;
  my ($vt, $v) = (0, 0);

  $vt = $l; #days
  $vt =~ s/\/.*//;
  $v = $vt * 86400;
  $l =~ s/^[^:]*://;
```

```
  $vt = $l; #hours
  $vt =~ s/:.*//;
  $v += $vt * 3600;
  $l =~ s/^[^:]*://;

  $vt = $l; #minutes
  $vt =~ s/:.*//;
  $v += $vt * 60;
  $l =~ s/^[^:]*://;

  $l =~ s/\s.*//; #seconds
  $v += $l;

  $date2sec = $v;
}#date2sec

# This function calculates the difference between two dates.
#
# Arguments: two strings containing a date e.g: "1/Jan/1970:00:00:00 +0100"
# Returns: integer
sub diff {
  my ($l1, $l2) = (shift, shift);

  $v1 = date2sec("$l1");
  $v2 = date2sec("$l2");

  $diff = $v1 - $v2;
}#diff

# This function looks if a given network is in the @spiders list.
#
# Arguments: a sting containing a class C network e.g: "10.20.30"
# Returns: 1 if the network is found.
#          0 otherwise.
sub checkip {
  my $l = shift;

  for (my $i = 0; $i < $#spiders; $i++) {
    if ($l eq $spiders[$i]) {
      return 1;
    }#if
  }#for
  return 0;
}#checkip
```

```
# This is the I() function from the ID3 algorithm.
# It calculates: -(p1 * log(p1) + p2 * log(p2) + .. + pn * log(pn))
#
# Arguments: an arbitrary amount of floats as long as the sum is 1.
# Returns: float
sub I {
  my $l = shift;

  $I = 0;
  while ($l) {
    $I += $l * (log($l) / log(2));
    $l = shift;
  }#while
  $I = -$I;
}#I

# This is an alternative of the info() function from the ID3 alhorithm.
# It calculates: Sum for i from 1 to n of |Ti| / |T| * I(Ti)
#
# Arguments: |T|, |T1|, I(T1), |T2|, I(T2), ...
# Returns: float.
sub info {
  my ($T, $l, $Il) = (shift, shift, shift);

  $info = 0;
  while ($l) {
    $info += ($l / $T) * $Il;
    $l = shift;
    $Il = shift;
  }#while
}#info

# This function looks for three successive hits in all files in the
# $logs_location directory. It also measures the time between next hits if any,
# Note that the @spiders array must be initialized. Leave out the marked if-else
# clauses when you want to use the C4.5 or other algorithms.
#
# Arguments: none
# Returns: nothing
sub crawl {
  my @logfiles = `ls $logs_location`;
  my ($ip, $date, $action, $url, $result, $bytes, $tdiff, $isengine);

  foreach (@logfiles) {
    open (TRAINING, "$logs_location/$_") or die "Can not open $_: $!";
```

```
while (<TRAINING>) {
  $ip = $_;
  $ip =~ s/\s.*//;         # Strip the ip address.
  $ip =~ s/\.\d{1,3}\s//;  # Truncate after the class C network.
  $_ =~ s/.*\s\[//;
  $date = $_;
  $date =~ s/\]\s.*//;
  $_ =~ s/.*\s\"//;
  $action = $_;
  $action =~ s/\s.*\s//;
  $_ =~ s/\S*\s//;
  $url = $_;
  $url =~ s/\s.*\s//;
  $url =~ s/^.*\///;
  if ($url =~ /\./) { # Mark.
    $url =~ s/^.*\.//;
    $url =~ s/.*/\U$url/;
    foreach (@textlike_files) {
      $url = "PLAIN" if ($url = $_);
    }#foreach
    $url = "BIN" if ($url ne "PLAIN");
  }#if
  else {
    $url = "PLAIN";
  }#else              # End mark.

  #$_ =~ s/(\S*\s){2}//;
  $_ =~ s/.*\"\s//;
  $result = $_;
  $result =~ s/\s.*\s//;
  if (($result eq "-") || ($result eq "")) { # Mark.
    $result = "NONE";
  }#if
  elsif ($result < 200) {
    $result = "INFO";
  }#if
  elsif ($result < 300) {
    $result = "SUCCESS";
  }#if
  elsif ($result < 400) {
    $result = "REDIR";
  }#if
  elsif ($result < 500) {
    $result = "CLIENT_ERR";
  }#if
  else {
```

```
      $result = "SERVER_ERR";
    }#else                                      # End mark.

    $_ =~ s/\S*\s//;
    $bytes = $_;
    $bytes =~ s/\s//;
    if (($bytes eq "-") || ($bytes eq "")) { # Mark.
      $bytes = "NONE";
    }#if
    elsif ($bytes < 1000) {
      $bytes = "SMALL";
    }#if
    elsif ($bytes < 10000) {
      $bytes = "MEDIUM";
    }#if
    else {
      $bytes = "BIG";
    }#else                                     # End mark.

    if (defined $results{"$ip"}) {
      $tdiff = diff($date, $results{"$ip"});
    }#if
    else {
      $tdiff = 10000000;
    }#else
    $results{"$ip"} = $date;
    if ($tdiff <= 60) { # Mark.
      $tdiff = "SMALL";
    }#if
    elsif ($tdiff <= 1000) {
      $tdiff = "MEDIUM";
    }#if
    else {
      $tdiff = "BIG";
    }#else              # End mark.

    if (checkip($ip)) {
      $isengine = "TRUE";
    }#if
    else {
      $isengine = "FALSE";
    }#else
    print "0$tdiff 1$action 2$url 3$result 4$bytes $isengine\n";
  }#while
  close (TRAINING);
}#foreach
```

```
}#crawl

# Main.
loadspiders;
crawl;

id3.pl:
#!/usr/bin/perl -w

# Global variables.
%ptrs = (
  0 => {0 => {0, "0SMALL", 1, 0, 2, 0},
        1 => {0, "0MEDIUM", 1, 0, 2, 0},
        2 => {0, "0BIG", 1, 0, 2, 0}},
  1 => {0 => {0, "1GET", 1, 0, 2, 0},
        1 => {0, "1HEAD", 1, 0, 2, 0}},
  2 => {0 => {0, "2PLAIN", 1, 0, 2, 0},
        1 => {0, "2BIN", 1, 0, 2, 0}},
  3 => {0 => {0, "3NONE", 1, 0, 2, 0},
        1 => {0, "3INFO", 1, 0, 2, 0},
        2 => {0, "3SUCCESS", 1, 0, 2, 0},
        3 => {0, "3REDIR", 1, 0, 2, 0},
        4 => {0, "3CLIENT_ERR", 1, 0, 2, 0},
        5 => {0, "3SERVER_ERR", 1, 0, 2, 0}},
  4 => {0 => {0, "4NONE", 1, 0, 2, 0},
        1 => {0, "4SMALL", 1, 0, 2, 0},
        2 => {0, "4MEDIUM", 1, 0, 2, 0},
        3 => {0, "4BIG", 1, 0, 2, 0}}
);

# Functions.

# Protected division.
# It devides the first argument by the second one.
#
# Arguments: two floats.
# Returns: float.
sub div {
  my ($l1, $l2) = (shift, shift);

  if (!$l2) {
    return 0;
  }#if
  return $l1 / $l2;
}#div
```

```
# This is the I() function from the ID3 algorithm.
# It calculates: -(p1 log(p1) + p2 log(p2) + .. + pn log(pn))
#
# Arguments: an arbitrary amount of floats as long as the sum is 1.
# Returns: float
sub I {
  my $l = shift;
  my $temp = 0;

  while ($l) {
    $temp += $l * (log($l) / log(2));
    $l = shift;
  }#while
  return -$temp;
}#I

# This is an alternative for the info() function from the ID3 alhorithm.
# It calculates: Sum(i) from 1 to n of |Ti| / |T| * I(Ti)
#
# Arguments: |T|, |T1|, I(T1), |T2|, I(T2), ...
# Returns: float.
sub info {
  my ($T, $l, $Il) = (shift, shift, shift);
  my $temp = 0;

  while ($l) {
    $temp += div($l, $T) * $Il;
    $l = shift;
    $Il = shift;
  }#while
  return $temp;
}#info

# This function returns the n'th argument in a line.
#
# Arguments: A list of words seperated with whitespace characters.
#            an integer n.
# Returns: list[n].
sub cut {
  my ($l1, $l2) = (shift, shift);

  $l1 =~ s/^(\S*\s){$l2}//;
  $l1 =~ s/\s.*\s//;
  return $l1;
}#cut
```

```
# This function updates the $ptrs datastructure. It counts the TRUE and
# FALSE cases for all non-categorical attributes.
#
# Arguments: A list of non-categorical attribute values.
#            a categorical attribute value.
# Returns: nothing.
sub updatestats {
  my ($l1, $l2) = (shift, shift);

  for (my $i = 0; $i < 5; $i++) {
    if (defined $ptrs{$i}) {
      my $temp = cut($l1, $i);
      my $j = 0;
      while (defined $ptrs{$i}{$j}) {
        if ($ptrs{$i}{$j}{0} eq $temp) {
          if ($l2 eq "TRUE") {
            $ptrs{$i}{$j}{1}++;
          }#if
          else {
            $ptrs{$i}{$j}{2}++;
          }#else
        }#if
        $j++;
      }#while
    }#if
  }#for
}#updatestats

# This function re-initializes the $ptrs datastructure.
#
# Arguments: none.
# Returns: nothing.
sub reinit {
  my $j = 0;

  for (my $i = 0; $i < 5; $i++) {
    $j = 0;
    while (defined $ptrs{$i}{$j}) {
      $ptrs{$i}{$j}{1} = 0;
      $ptrs{$i}{$j}{2} = 0;
      $j++;
    }#while
  }#for
}#reinit

# This function prints the 'value' of a leaf. If the TRUE count is higher
```

```
# than the FALSE count, it will print TRUE, and vice versa. If both counts
# are the same it will print a question mark.
#
# Arguments: A non-catagorical attribute value.
# Returns: nothing.
sub printptrs {
  my $leaf = shift;
  my $j = 0;

  for (my $i = 0; $i < 5; $i++) {
    $j = 0;
    while (defined $ptrs{$i}{$j}) {
      if ($ptrs{$i}{$j}{0} =~ /$leaf/) {
        print "$ptrs{$i}{$j}{0} ";
        if ($ptrs{$i}{$j}{1} > $ptrs{$i}{$j}{2}) {
          print "TRUE\n";
        }
        else {
          if ($ptrs{$i}{$j}{1} < $ptrs{$i}{$j}{2}) {
            print "FALSE\n";
          }
          else {
            print "?\n";
          }
        }
      }
      $j++;
    }#while
  }#for
}#printptrs

# Main.
$root = -1;
$imax = 0;
my ($truecount, $iter) = (0, 0);

# The first level.
# First collect all information needed.
open(TRAINING, "training.tbl") or die "Can not open training.tbl: $!";
while (<TRAINING>) {
  if (/TRUE/) {
    updatestats($_, "TRUE");
    $truecount++;
  }#if
  else {
    updatestats($_, "FALSE");
```

```
  }#else
  $iter++;
}#while
close(TRAINING);

# And now calculate the information each non-categorical attribute has.
my $temp = I($truecount / $iter, ($iter - $truecount) / $iter);
for (my $i = 0; $i < 5; $i++) {
  my @args = ();
  my $j = 0;
  while (defined $ptrs{$i}{$j}) {
    my $total = $ptrs{$i}{$j}{1} + $ptrs{$i}{$j}{2};
    $args[$j * 2] = $total;
    $args[($j * 2) + 1] = I(div($ptrs{$i}{$j}{1}, $total),
        div($ptrs{$i}{$j}{2}, $total));
    $j++;
  }#while
  my $max = $temp - info($iter, @args);
  if ($max > $imax) {
    $imax = $max;
    $root = $i;
  }#if
}#for

# Now do the same trick for all non-categorical attribute values of the root.
print "level 1:\n$root\n---------------------------------------\nlevel 2:\n";
my $newroot = 0;
while (defined $ptrs{$root}{$newroot}) {
  print "$ptrs{$root}{$newroot}{0}\n";
  $truecount = 0;
  $iter = 0;
  reinit;
  open(TRAINING, "training.tbl") or die "Can not open training.tbl: $!";
  while (<TRAINING>) {
    if (/$ptrs{$root}{$newroot}{0}/) {
      if (/TRUE/) {
        updatestats($_, "TRUE");
        $truecount++;
      }#if
      else {
        updatestats($_, "FALSE");
      }#else
      $iter++;
    }#if
  }#while
  close(TRAINING);
```

```
  my $temp = I(div($truecount, $iter), div(($iter - $truecount), $iter));
  $imax = 0;
  my $leaf = 0;
  for (my $i = 0; $i < 5; $i++) {
    if ($i == $root) {
      $i++;
    }
    my @args = ();
    my $j = 0;
    while (defined $ptrs{$i}{$j}) {
      my $total = $ptrs{$i}{$j}{1} + $ptrs{$i}{$j}{2};
      $args[$j * 2] = $total;
      $args[($j * 2) + 1] = I(div($ptrs{$i}{$j}{1}, $total),
          div($ptrs{$i}{$j}{2}, $total));
      $j++;
    }#while
    my $max = $temp - info($iter, @args);
    if ($max > $imax) {
      $imax = $max;
      $leaf = $i;
    }#if
  }#for
  printptrs $leaf;
  print "---------------------------------------\n";
  $newroot++;
}#while
print "\n";
```