

Classifying Web Pages by Function

Henry Beecher
UCSD Linguistics Dept.
hbeecher@ucsd.edu

Abstract

This paper investigates using supervised learning techniques such as Naïve Bayes or SVM to classify web pages into five functional categories: personal homepage, corporate homepage, retail page, media page, and theme page. On 2-way decisions using Naïve Bayes, an average accuracy of 87.5% was achieved. Average accuracy drops to 70% when combining all five categories. Further research is needed to determine if combining supervised learning with decision rule approaches can improve multi-decision accuracy.

1. Introduction

Popular methods for categorizing pages on the World-Wide Web such as Yahoo's hierarchical directory of web sites or Google's authority-based ranking focus on content or subject matter as the most relevant criteria for classifying web-based sources. The continued dramatic growth of the Internet, however, makes narrowing the possible results to a query more difficult without simultaneously increasing burden on the user by requiring the use of more sophisticated search strings and/or additional feedback. The aim of this paper is to investigate the feasibility of supplementing traditional content-based web-page classification with automated assignment of categorization by functional type such as personal homepage, corporate web site, on-

line catalogue, archive, electronic journal, fact sheet, etc.

2. Considerations

Unlike subject-based categorization that can be directly correlated with well established guidelines for domain assignment such as found in both the library and information sciences, there are no commonly accepted, pre-defined 'functional types' for web pages (of which I am aware). In particular, such purportedly regulated domain extensions like .edu, .gov, .org, or .com are much too broadly and inconsistently applied to be sufficiently reliable. Thus, a necessary first step is identifying plausible types and determining minimum criteria for inclusion. For the purposes of this project, only a small number of prominent types are considered.

Another consideration is the selection of features used to determine type assignment. In this regard, no aspect of a web page is *a priori* excluded. However, it is anticipated that elements such as a page's URL, choice and content of HTML tags, document structure including hyperlinks, and the existence of advertisements or other such 'non-content' components will be more decisive in differentiating web pages by functional type.

3. Data set

The five functional categories selected are: personal, corporate, retail, media, and theme.

Although an effort was made to choose categories which are as maximally dis-similar as possible, this proved to be quite challenging as web pages often serve more than one major purpose. For each category, approximately 100 random URLs were manually collected. In deciding whether a web page belonged in a particular category, the following rudimentary guidelines were used.

Personal: typically an individual's homepage or professional page.

Corporate: a company homepage (exclusive of product merchandizing).

Retail: a page primarily devoted to online commerce, products or services.

Media: news, entertainment, sports, *etc.* pages (commercialized or not).

Theme: specialized, single-topic, info-page (preferably not selling anything).

Not every URL was individually retrieved in a browser or otherwise separately verified. A variety of collection techniques were availed to expedite the process. For instance, if a web page in a given category was found to contain URL links to other pages of the same type, those links might be added directly to the list for that category. Similarly, some URLs were obtained by using the "similar pages" feature in Google. Consequently, some of the URLs collected later turned out to be dead links; and, not all undesired commercialized pages (*e.g.* in the theme category) were avoided.

Custom Python routines¹ were used to extract any navigable text, image names, links, and HTML meta tags from the contents of the initial page for each URL. These elements were also retrieved in the contents of links one-level deep in the same domain. Any files smaller than 1k which often contain just flash or a redirector script as well as most Java script were eliminated.

¹ Coded by H. Rohde, see Appendix I

² Coded by H. Beecher, see Appendix II

A separate 'scrubbed' data file was produced for each URL processed. The number of URLs yielding useful data in each category is as follows: 102 personal; 85 theme; 79 media; 77 corporate; and 74 retail. The lower number of useful corporate and retail URLs is likely indicative of an increased utilization of flash in these web pages.

4. Classification

Supervised learning trials were conducted using both Naïve Bayes and SVM in the Rainbow statistical classifier software package. A few additional trials were run using the SVM^{light} software. The trials executed using Rainbow included all combinations of 2-way decisions, some 3-way and 4-way decisions as well as trials with all five categories together. An average of 87.5% on 2-way decisions represented the highest accuracy obtained using Naïve Bayes. However, other combinations with Naïve Bayes were progressively less accurate: 77.3% on average for 3-way decisions; 75.7% on average for 4-way decisions; and 70% on average for a 5-way decision.

The same input files created for the Naïve Bayes trials in Rainbow were used for the SVM trials in Rainbow. The SVM^{light} software, however, required all features to be represented numerically and preceded in the data file by either a 1 for a positive example or a -1 for a negative example. This was accomplished by feeding a list of features (*i.e.* words) through a custom Python routine² which assigned both the numerical representation and the 1/-1 identifier. Curiously, both SVM Rainbow and SVM^{light} yielded the lowest accuracy on 2-way decisions: 59.3% on average for SVM Rainbow; and 55% on average for SVM^{light}.³

1	Personal / Retail	93.3%	6	Media / Theme	88%
2	Corporate / Retail	91%	7	Corporate / Media	88%
3	Corporate / Theme	91%	8	Personal / Corporate	84.6%
4	Retail / Theme	90%	9	Personal / Media	84.6%
5	Media / Retail	88.8 %	10	Personal / Theme	76.6%

Table 1. Accuracy results for 2-way decisions

1	Personal / Corporate / Retail	87%	6	Personal / Retail / Theme	77.4%
2	Corporate / Media / Theme	82.6%	7	Corporate / Media / Retail	75.6%
3	Personal / Media / Theme	79.9%	8	Personal / Corporate / Media	75%
4	Personal / Media / Retail	79.9%	9	Corporate / Retail / Theme	69.2%
5	Personal / Corporate / Theme	78.7 %	10	Media / Retail / Theme	68.5%

Table 2. Accuracy results for 3-way decisions

1	Personal / Corporate / Media / Theme	81.7%	(- Retail)
2	Personal / Corporate / Media / Retail	79.1%	(- Theme)
3	Corporate / Retail / Theme / Personal	77%	(- Media)
4	Personal / Media / Retail / Theme	72.6%	(- Corporate)
5	Corporate / Media / Retail / Theme	68.3 %	(- Personal)

Table 3. Accuracy results for 4-way decisions

5. Results

Details of the accuracy results obtained for 2-way, 3-way, and 4-way decisions using Naïve Bayes in Rainbow are shown in Tables 1-3. All of the percentages identified represent the average of ten trials. Table 1 depicts the individual accuracy results for each of the ten possible pairs which may be differentiated using the five categories. Distinguishing between personal and retail pages had the highest accuracy at 93.3% and likely indicates that these two categories are the most dissimilar of the five. The lowest accuracy of 76.6% was associated with deciding between personal and theme pages, likely the two least dissimilar categories of the five.

Perhaps not surprisingly, the highest

accuracy shown in Table 2 also includes both the personal and retail categories (the two categories for which differentiating in a 2-way decision had the highest accuracy). It is also interesting to note that the personal category is in four of the five 3-way decision combinations having higher accuracies. This would predict that elimination of the personal category would likely result in noticeably poorer accuracy.

As expected the results for 4-way decisions in Table 3 show that not including the personal category yields the lowest accuracy. Conversely, eliminating either the retail or theme categories has relatively little negative impact on the overall accuracy results. This echoes the previously noted conclusion that the retail category (and

³ The discrepancy in these results with SVM raise concerns about the implementation of SVM in Rainbow on the one hand; and the generation of a proper input file for SVM^{light} on the other hand.

perhaps theme as well) is least distinguishable among the five categories.

The most frequently occurring features for any given category can be generated as a list using the Rainbow software. Examining the highest occurring features can be edifying. For instance, some of the most frequent features in the personal category are *text*, *sound*, *visit* and *porn*. In the media category *editor*, *writer*, *read* and *graphics* are among the most frequent. The high occurrence of these features in these particular categories is to some extent predictable, but stands in stark contrast to the most frequently occurring features in other categories like corporate or retail. In the corporate category, features like *img*, *gif*, *jpg* or *layout* are among the most prevalent. Features such as *images*, *pixel*, *graphics* or *icon* are among the most frequent in the retail category. The nature of the highest occurring features in the corporate and retail categories appears to indicate a significant number of professionally produced web pages and/or pages created with high-end web development tools.

6. Related Work

Although a plentiful number of scholarly publications exist on content-based web page classifying, there is relatively little previous research which specifically addresses classifying pages by functional type in the manner investigated herein. In her dissertation, *An Empirical Foundation for Automated Web Interface Evaluation*, Melody Ivory-Ndiaye collapses a variety of functional categories separately proposed in work by Karlgren 2000, Pirolli *et al* 1996, and Stein 1997 into the following five page types.

Home: main entry pages to a site that typically provide a broad overview of site contents.

Link: pages that mainly provide one or

more list of links. Links may be annotated with text or grouped with headings (e.g., yahoo directory, redirect page, or sitemap). This functional type includes category pages (entry pages to sub-site or major content areas).

Content: pages that mainly provide text. This functional type includes reference (e.g., a glossary, FAQ, search and site tips, and acronyms) and legal (e.g., disclaimers, privacy statements, terms, policies, and copy-right notices) pages.

Form: pages that are primarily HTML forms.

Other: all remaining graphical (e.g., splash pages, image maps, and Flash) and non-graphical (e.g., blank, under construction, error, applets, text-based forms, and redirect) pages.

Using the Classification and Regression Tree (C&RT) method of Breiman *et al* 1984, Ivory-Ndiaye reports 75% accuracy on a sample of 1770 web pages. While these results are interesting, they are not directly comparable to the findings of the present project. The nature of the functional categories defined by Ivory-Ndiaye (and thus those of Karlgren, Pirolli and Stein from which they derive) relate more to the role a particular page serves within an overall site rather than evaluating a given page in order to categorize the primary purpose of an entire site.

7. Summary

The findings of this investigation show some promising results for categorizing web pages by function as indicated by the trials using Naïve Bayes in Rainbow. The trials using both versions of SVM were inconclusive due to separate and unrelated circumstances. While an average accuracy of 70% was achieved in trials combining all five categories, a more impressive 87.5% average accuracy was achieved in 2-way

decision trials. For production purposes, however, 2-way decisions are the least efficient. This leaves for future research investigating alternatives for differentiating web pages among multiple categories without such degradation in accuracy. Among the possibilities for achieving this could be combining supervised learning with decision rules which assign increased weight to specific features such as the presence *shopping cart* in retail pages or *guest book* in personal pages.

8. References

- Breiman, Leo, J. H. Friedman, R. A. Olsen, and C.J. Stone. 1984. *Classification and Regression Trees*. Belmont, CA: Wadsworth Publishing Company.
- Cohen, William. Improving a page classifier with anchor extraction and link analysis. (ms.)
- Ivory-Ndiaye, Melody, 2001. *An Empirical Foundation for Automated Web Interface Evaluation*. PhD dissertation, UC Berkeley.
- Joachims, T. 1999. Making large-Scale SVM Learning Practical. *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burges and A. Smola (eds.), MIT-Press.
- . 2001. A statistical learning model of text classification for support vector machines. In *Proceedings of SIGIR'01*, New Orleans.
- Karlgren, Jussi, 2000. *Stylistic Experiments for Information Retrieval*. PhD dissertation, Stockholm University.
- Kwong, On-Wong and Jong-Hyeok Lee. 2000. Web page classification based on k-nearest neighbor approach. In *Proceedings of the 5th International Workshop on Information Retrieval with Asian Languages*.
- Lu, Cheng and Mark Drew. 2001. Construction of a hierarchical classifier schema using a combination of text-based and image-based approaches. In *Proceedings of SIGIR '01*, New Orleans.
- Mladenovic, Dunja. 1998. Turning Yahoo into an automatic web-page classifier. In *Proceedings of the 13th European Conference on Artificial Intelligence*.
- Pirolli, Peter, James Pitkow, and Ramana Rao. 1996. Silk from a sow's ear: Extracting usable structure from the web. In *Proceedings of the Conference on Human Factors in Computing Systems: Common Ground*, 118-125.
- Shen, Dou and Zheng Chen, Qiang Yang, Hua-Jun Zeng, Benyu Zhang, Yuchang Lu, Wei-Ying Ma. 2004. Web-page classification through summarization. In *Proceedings of SIGIR'04*, South Yorkshire.
- Stein, Lincoln. 1997. The rating game. Available at <http://stein.cshl.org/~lstein/rater/>.

Appendix I

```
#!/usr/bin/python

import BeautifulSoup
import urllib
import sys
import re
import string
import tidy
import urlparse
import os

def wrap(text,width):
    return reduce(lambda line, word, width=width: '%s%s%s' %
                  (line,
                   ' \n'[(len(line[line.rfind('\n')+1:])
                        + len(word.split('\n',1)[0]) >= width)],
                   word),
                  text.split(' '))

def sanitize(t):
    t = re.sub(r'<!--[^\>]*-->', '', t)
    t = re.sub(r'\n +', '\n', t)
    t = re.sub(r' +', ' ', t)
    return t.strip()

def getDomainURLs(root,soup,urlsToTraverse):
    for link in soup('a'):
        try:
            target = link['href']
            newName = ""
            if re.search(r'mailto:',target): continue
            elif re.search(root,target):
                newName = target
            elif re.search(r'^https?://',target): continue
            elif re.search(r'^/',target): continue
            else:
                root =
re.sub(r'[/^\^]*(\.\asp|\.\jsp|\.\html)?/?', '/', root)
                newName = root+target

            urlsToTraverse.append(newName)

        except KeyError:
            pass
    return urlsToTraverse

def traverse(item,name,text):
    if isinstance(item,BeautifulSoup.NavigableText):
```

```

    t = item.string.rstrip()
    semicolon = t.split(";")
    braces = t.split("{")
    if(t != '' and t != '&nbsp;' and not re.search(r'<!',t) and
(len(semicolon)<2 and len(braces)<2)):
        text.append(t)
    if isinstance(item,BeautifulSoup.Tag):
        if item.name == 'br':
            text.append('\n')
        elif item.name == 'img':
            text.append(item.get('src', 'SOURCE'))
    if(not isinstance(item,BeautifulSoup.NavigableText)):
        for subitem in item.contents:
            traverse(subitem,name,text)

return text

def robTraverse(item,text):
    if isinstance(item,BeautifulSoup.Tag):
        if item.name == 'br':
            text.append('\n')
        elif item.name != 'table':
            for i in item.contents:
                text = robTraverse(i,text)
    else:
        text.append(re.sub(r'\n',' ',item.string))
    return text

def loadSoup(soup,root):
    for i in xrange(3):
        try:
            html = urllib.urlopen(root).read()
            success = True
        except:
            html = ''
            success = False
        if success:
            break
    oldhtml = html
    html = tidy.parseString(html)
    if re.match(r'^\w*$',str(html)):
        # tidy stripped everything -- return to oldhtml
        html = oldhtml
    try:
        soup.feed(str(html))
    except:
        success = False
    return soup

def getName(groupName,root):
    name = re.sub(r'http://\w*\.(.*)',r'\1',root)
    name = re.sub('~',' ',name)
    name = re.sub('\./','.',name)
    print >>sys.stderr, root

```

```

        return groupName+".data/"+name

# main program
for i in range(1,len(sys.argv)):
    groupName = sys.argv[i]
    group = open(groupName,'r')
    urls = group.readlines()
    group.close()
    try:
        os.mkdir(groupName+".data/")
    except OSError:
        continue
    for root in urls:
        root = re.sub(r"[\r\n]+$", "", root)
        name = getName(groupName,root)
        stream = open(name,'w')
        urlsToTraverse = [root]
        text = [ ]
        # get all URLs to traverse
        soup = loadSoup(BeautifulSoup.BeautifulSoup(),root)
        urlsToTraverse = getDomainURLs(root,soup,urlsToTraverse)
        numURLsToTraverse = 25 #need to randomly take url??
        print >>sys.stderr, len(urlsToTraverse),"links to follow"
        while len(urlsToTraverse)>0 and numURLsToTraverse>0:
            currentURL = urlsToTraverse.pop(0)
            soup = loadSoup(BeautifulSoup.BeautifulSoup(),currentURL)
            print >>sys.stderr,"\t", currentURL
            text = traverse(soup,name,text)
            t = sanitize(string.join(text,' '))
            if t!='':
                print >>stream, wrap(t,75)
            numURLsToTraverse = numURLsToTraverse - 1

```

Appendix II

```
#!/usr/local/bin/python2.3

# a program to extract class features

import os          # this module contains directory manipulation
functions

# input files
file_a = 'ct-list.txt'
file_b = 'rt-list.txt'

# output files
trainfile = 't-train.dat'
testfile = 't-test.dat'

# main program

itemcount = 1
poscount = 0
negcount = 0
tempdict = {}          # initialize
dictionaries
posdict = {}
negdict = {}

rf = open(file_a, 'r')
for line in rf:        # iterate over lines
    line = line.replace('-', '')
    wordcount = 0
    for word in line.split():
        if wordcount == 1:
            if word in tempdict:
                tempdict[word] += 1
            else:
                tempdict[word] = 1
        wordcount += 1
rf.close()

rf = open(file_b, 'r')
for line in rf:
    line = line.replace('-', '')
    wordcount = 0
    for word in line.split():
        if wordcount == 0:
            tempword = str(float(word) + itemcount)
        else:
            if word in tempdict:
                if tempword in posdict:
                    posdict[tempword] += 1
            else:
```

```

        posdict[tempword] = 1
    else:
        if tempword in negdict:
            negdict[tempword] += 1
        else:
            negdict[tempword] = 1
    wordcount += 1
    itemcount += 1
rf.close()

# now we've collected each list -- time to generate output
poslist = [] # initialize final list
for item in posdict.items(): # translate dict to a list of
    tuples
    poslist.append([item[0]]) # add to list

neglist = [] # initialize final list
for item in negdict.items(): # translate dict to a list of
    tuples
    neglist.append([item[0]]) # add to list

outcount = 1

# write output
wf = open(trainfile, 'w') # open output file
for line in poslist:
    poscount += 1
    wf.write("1 "+str(outcount)+":"+line[0])
    wf.write('\n')
    outcount += 1
for line in neglist:
    negcount += 1
    wf.write("-1 "+str(outcount)+":"+line[0])
    wf.write('\n')
    outcount += 1
wf.close() # close output file

testcount = 1
outcount = 1

wf = open(testfile, 'w') # open output file
for line in poslist:
    if testcount < 201:
        wf.write("1 "+str(outcount)+":"+line[0])
        wf.write('\n')
        outcount += 1
    testcount += 1
testcount = 1
for line in neglist:
    if testcount < 201:
        wf.write("-1 "+str(outcount)+":"+line[0])
        wf.write('\n')
        outcount += 1
    testcount += 1

```

```
wf.close()                                # close output file

#####

print poscount, "positive features found"  # print summary
statistics
print negcount, "negative features found"
```