Automatically Detecting Banner Ads in Web Pages

Douglas Greiman

Stanford University, Stanford, CA 94305

duggelz@gmail.com

1. Introduction

This paper describes AdZap, a program for detecting and blocking advertisements on web pages. AdZap uses a set of labeled training data collected from the user as input to a supervised learning algorithm. The trained algorithm then examines images embedded in HTML documents shown to the user and hides images classified as advertisements.

2. Background

There are a number of mechanisms for delivering advertisements on the web. For simplicity, AdZap only considers IMG tags, which are the most common mechanism. An HTML document contains embedded images where each image is represented by an IMG element. An IMG element has a set of HTML attributes, and a set of CSS attributes. One of the HTML attributes is a URL pointing to the image file to be displayed in the document. In the rest of this paper, references to "HTML attributes" will mean all attributes except the URL.

Advertisement URLs are usually generated by ad-serving software based on the location and content of the document in which they appear, along with other factors such as current advertisement inventory. The same page viewed at different times will display different advertisements, although usually in the same format and the same location on the page. There are a large number of different ad-serving software packages available. Each package constructs URLs in a fairly arbitrary and distinctive way.

There are also number of ad-blocking software packages available to detect and block advertisements on web pages. Currently, these ad-blockers use hand-constructed lists of regular expressions. Every URL fetched by the user's browser is compared against the regular expressions. Images that match are hidden from display or replaced with a blank image. Other HTML and CSS attributes of the images are ignored.

Lists of regular expressions are reasonably effective for advertisements already seen. However, they require regular maintenance. Furthermore, the great majority of end users are not capable of or willing to craft regular expressions to identify advertisements. Thus, ad-blocking programs rely on lists maintained by small groups of experts and distributed to end users on a regular basis.

3. Classifying Images

AdZap constructs labeled feature vectors based on clicks from the user, as described below. There are two possible labels: "ad" and "not-ad". These vectors are used to train a Naïve Bayes Multivariate Bernoulli event model. The model is retrained each time the user creates a new feature vector. When the model changes, all currently displayed images are reclassified and redisplayed as appropriate.

An image has two types of information associated with it: image content, and image context. Image content is the actual pixel data of the image. Image context is all the ancillary information required by the browser to locate and display the image correctly. AdZap, like all other ad-blockers, only uses context information. The actual content of the image is not examined.

There is a lot of context information available for use. The IMG element contains HTML and CSS attributes, which are key-value pairs with semantics based on key names. The most important is the "src" attribute, whose value is the URL of the image file to display. This attribute is treated separately from all others.

3.1 Feature Construction

AdZap uses a very simple method of creating features from the URL. The URL is broken into tokens at non-alphanumeric boundaries. Series of tokens that are separated only by dots or hyphens are joined together to create additional larger tokens. For example, the URL `"http://speed.pointroll.com/Point-Roll/…"` is translated to the feature set: `{"url:speed", "url:pointroll", "url:com", "url:speed.pointroll.com", "url:Point", "url:Roll", "url:Point-Roll", … }`. The hostname is not distinguished from the other parts of the URL. It is common for ad URLs to include redirection steps, where the actual address of the image file is embedded as a query parameter inside a URL to a redirector site, for user tracking and other purposes.

AdZap uses another simple method of creating features from the HTML and CSS attributes. Each key/value pair is concatenated into a single token. For example, `<IMG height="20" border="1">` is translated to the feature set `{"attribute:height:20", "attribute:border:1"}`.

Note that attributes with the same key and different values are treated as unrelated by the model. That is, `"height:20"` and `"height:30"` are separate dimensions in the feature vector, with possible values 0 and 1, rather than a single dimension `"height"` with values from 0 to 1000. Also, integer valued attributes are not grouped into a smaller number of buckets (e.g. values from `"height:25"` to `"height:35"` are not normalized to `"height:30"`). This is motivated by the fact that the ad industry has a set of standard ad sizes, specified to the pixel. An image with one of the standard sizes is very likely an ad, and conversely, a size that is even one pixel different is much less likely to be an ad.

Location of the image on the page is also informative: banner advertisements are commonly placed along the top and right side of web pages, and less often on the left side or in the center. However, location is a difficult attribute to work with, since it depends on the shape, size and layout of the browser window and the rest of the document. Maximizing the browser window, adding or removing UI elements to the browser window, or even changing the font, can change the location of images. AdZap currently ignores attributes specifying location.

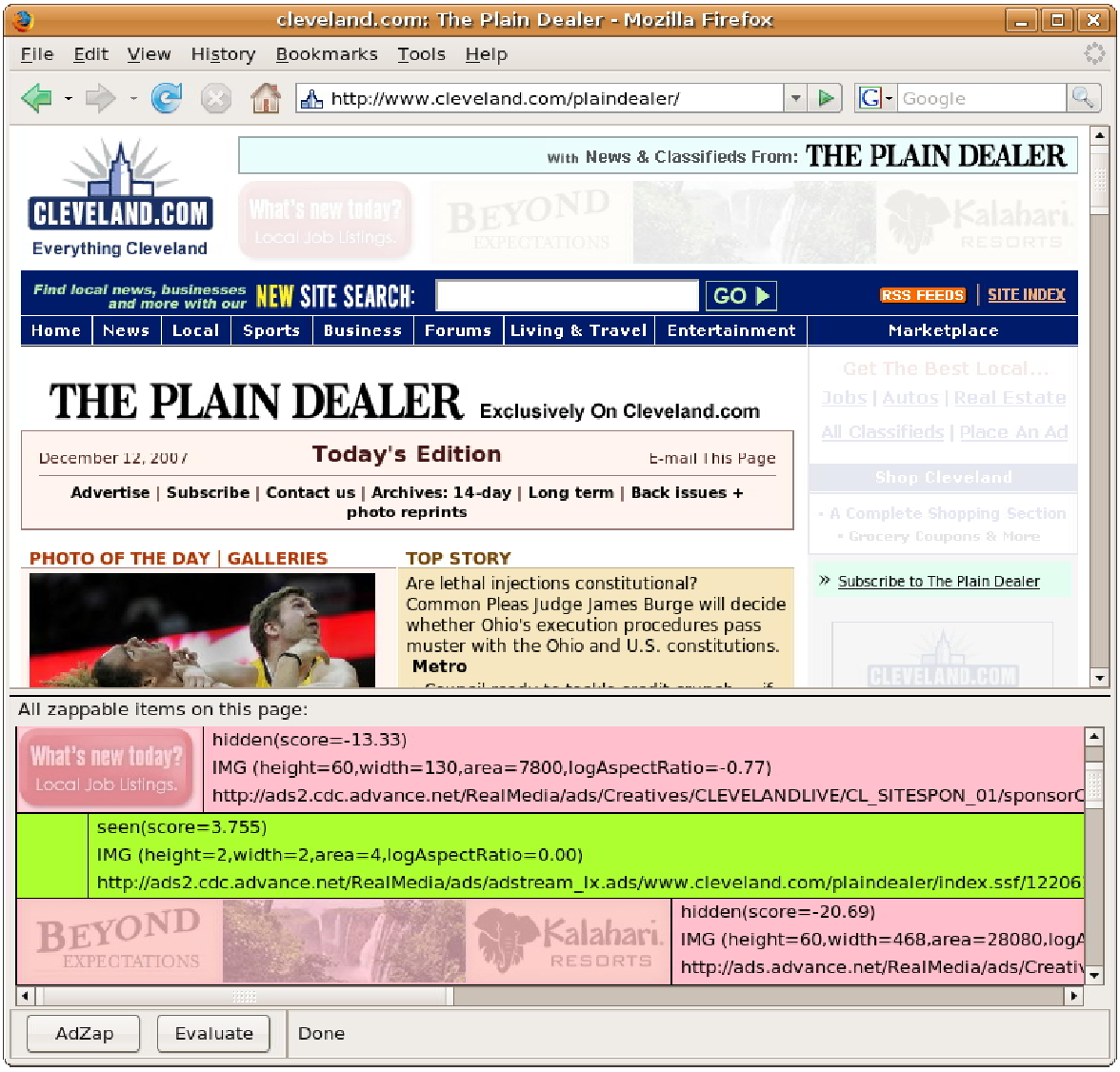
4. Implementation

AdZap is a Firefox browser plugin. Note that AdZap is a new piece of software and is unrelated to the "adzapper" package for squid proxies.

When a document is first loaded in the browser, each IMG tag is examined, and classified as "ad" or "not-ad" by the model. Ads are hidden by rendering them 90% transparent. This makes ads virtually invisible, but still makes the image visible enough for the user to recover from classification errors.

AdZap adds some controls to the browser window. Clicking on the AdZap button puts the browser in "zapping mode". In zapping mode, a left-click on an image labels that image as an "ad". Similarly, a right-click labels the image as "not-ad". The image list at the bottom of the window contains all the images in the current document, and left and right clicks in this list behave the same as zapping mode clicks. Labels created by these clicks are saved in a persistent store, and the model is retrained each time the user clicks.

Figure 1:AdZap User Interface



4.1 Training Data

The first design of AdZap only allowed the user to label ads. Every image on a page that was not explicitly labeled “ad” was implicitly labeled “not-ad” and used to train the model.

This implicit labeling works poorly. Experienced web surfers have trained themselves to identify and ignore ads at a subconscious level ("ad blindness"), so it is easy to overlook ads even when consciously looking for them. Beyond that, pages contain dozens, sometimes of hundreds of images that are small, unobtrusive, or even invisible. For example, CNN’s home page has more than 160 images. A small number are ads or news photos. The majority are company logos, invisible “web bugs”, partner logos, icons of unknown purpose, cobranding logos, and more. Web bugs are a particular problem, because their URLs are very similar to ad image URLs, but they never get marked as ads because they are invisible.

Finally, there are many images that are simply ambiguous. For example, Forbes’ home page contains a medium-sized image of a vehicle linked to ForbesAutos.com. On the one hand,

this is a legitimate navigation link to another section of Forbes' web site. ForbesAutos.com contains free auto reviews and other meaningful content. On the other hand, the image prominently features a Lincoln Towncar, and clicking the link takes you directly to the Lincoln section of ForbesAutos.com, complete with ads by Lincoln and links to purchase a Lincoln. It's purposely unclear where the dividing line between editorial content and advertising is.

This was a real problem until I realized that, for the great majority of images on the web, their proper classification is "don't know and don't care". The user cares about the large, flashing, annoying ads, and the large, interesting, meaningful photos and other actual content. Everything else is ignorable.

The second and current design of AdZap introduces a category of "ignore". All images smaller that a certain area are "ignore". This takes care of web bugs, company logos, and the like. Images that are very narrow or very short are also "ignore". This takes care of border art and lines. Images in "ignore" are not used as training data (unless explicitly relabeled as "ad" or "not-ad" by the user), and are not classified by the model. They are displayed normally in the browser.

Furthermore, the second design of AdZap allows the user to explicitly label images as "not-ad" as well as "ad". These labeled images are used as training data. Images that are not labeled are not used as training data, however they are classified by the model and hidden if classified as an "ad". Since "ad" images are rendered mostly transparent instead of totally hidden, this creates a simple feedback loop with the user. When the user notices misclassified images, they can explicitly label them and thus improve the model.

5. Experimental Results

It was not immediately clear what the best way to gather training data was. One could imagine enumerating all the web pages in existence and visiting a random sample of them. This has some logistical problems, and it's not clear that this would actually be representative of a real user experience anyway.

I decided to visit a selection of news sites. These sites offer a rich selection of news photos, advertisements, and miscellaneous images. My test procedure was to visit `news.google.com`, and for each story on the front page, follow the top three links for that story. Each story link goes to an article page on a news site like `nytimes.com`. On each article page, I classified every visible image as "ad", "not-ad" or "don't know and don't care" (by not clicking on that image). I repeated this procedure at intervals, since the stories on `news.google.com` change over time, until I had sufficient data points.

This training data was used to evaluate the model using different sets of features, as shown on Figure 2 (see next page). Each color represents a model that incorporates only the features listed in the legend above. Each feature set was evaluated using 20-fold cross-validation on training sets of various sizes.

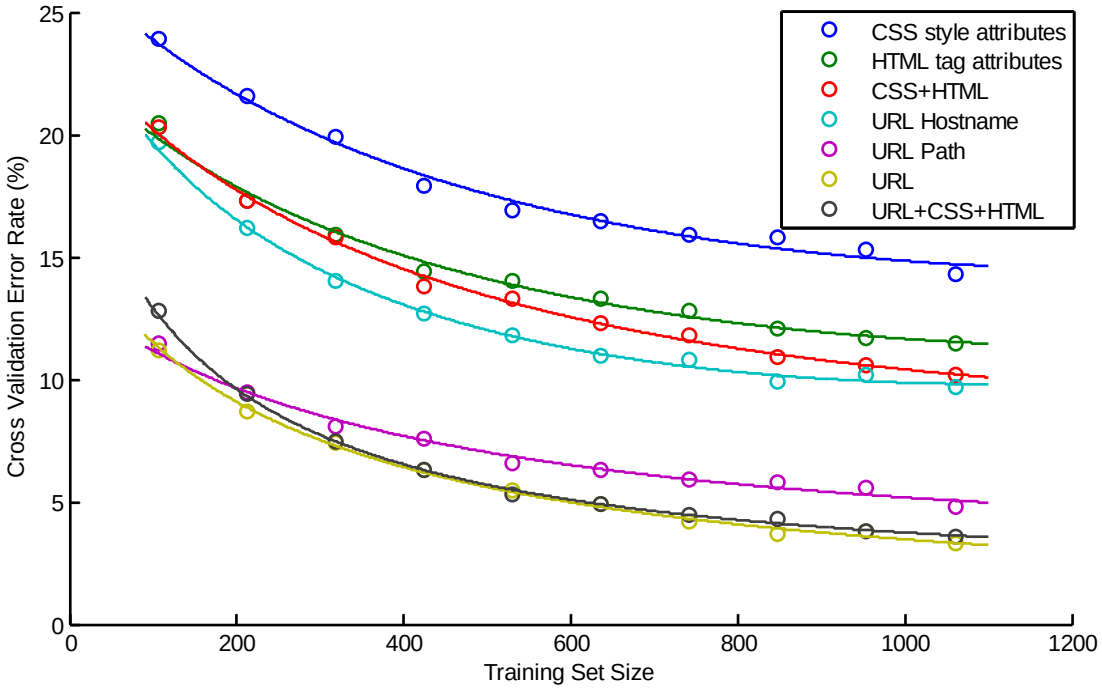
6. Conclusions

Training a Naïve Bayes model on image URLs is very effective at identifying ads. The few errors that occur have little negative effect on the user's browsing experience. Some error is unavoidable due to the inherent ambiguity of certain images.

Training a model on HTML and CSS attributes is not very effective at identifying ads. There are a number of standardized shapes and positions for banner ads, and these types of ads are detected easily. However, images in unusual places, such as the center of the page, or images with unusual shapes, are poorly classified.

Training on both URLs and HTML attributes isn't any better than training on just URLs. This might be due to overtraining.

Figure 2:Evaluation on various feature sets



7. Future Work:

Currently, ad URLs are distinctively different from non-ad URLs. However, if large numbers of users started using ad-blocking software, ad companies would probably respond by making their URLs indistinguishable from other image URLs, which would be fairly easy. If this happened, then ad classification based on HTML attributes like size and location would become relatively more useful, since these attributes can't be obfuscated like the URL can.

It might be possible to improve the performance of HTML features by more complex analysis of web pages. For example, most advertisements are clickable links, meaning that the IMG element is contained inside an A element. The A element has its own attributes, and its own URL, describing where the user will be sent if they click on the advertisement.

8. References

Nielsen, Jakob, "Banner Blindness: Old and New Findings", Jacob Nielson's Alertbox, http://www.useit.com/alertbox/banner-blindness.html (accessed December 15, 2007).

Ragget, Dave, Arnaud Le Hors, and Ian Jacobs, ed. "HTML 4.01 Specification", W3C, http://www.w3.org/TR/REC-html40/ (accessed December 15, 2007).

"Extensions", Mozilla Foundation, http://developer.mozilla.org/en/docs/Extensions (accessed December 15, 2007)

"adblock", The Adblock Project, http://adblock.mozdev.org (accessed December 15, 2007)