Relevanseek: Determining relevant images from HTML source

David Zhang
drz@cs.stanford.edu

Abstract

Webpages of online articles typically contain one or more images which are *relevant* to the article, such as a photo of a key person or a graph of figures mentioned in an article, and many images which are *irrelevant*, such as links to related articles, "share" buttons, and advertisements. Given the HTML source to an article webpage, but without actual image data, we are able to train a model able to distinguish the two classes of images with high accuracy.

Relevanseek was the winner of Diffbot's Machine Learning Challenge for this classification problem.[1]

1 Introduction

Diffbot[1], a company specializing in automated content extraction of webpages, posted a challenge: given the HTML source to an article, determine which images are *relevant* to the article content.

An example article is shown in Figure 1. This particular *Slate* article has text content about some flying gadgets, and includes a relevant image of one such toy at the top of the page. Other images on the page, such as social media buttons, the author's photo, and related article thumbnails, are irrelevant to this article.

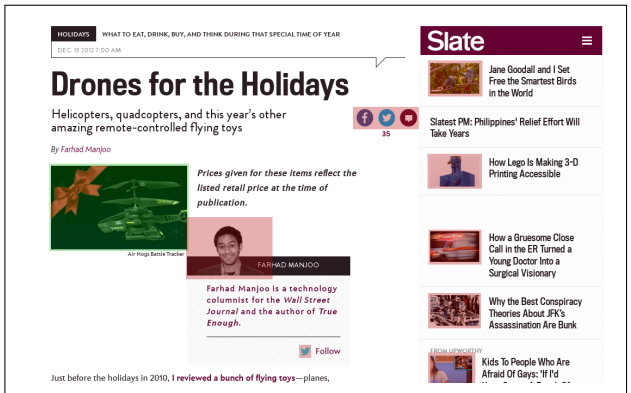

Figure 1: Example of an article with relevant images highlighted in green and irrelevant images highlighted in red.

It is easy to see why an effective algorithm for this task is valuable. For example, when sharing a URL on a service like Facebook or Skype, a thumbnail might be generated from one of the relevant images on the target webpage. As another example, a browser user agent (either a local client or a hosted service) can try to strip out irrelevant information, intrusive styles, etc and leave the reader with just relevant text and images with plain formatting. Determining which images are relevant is a key component of both of these systems.

Diffbot has provided a training set consisting of the raw HTML source for several articles. Because the actual images are not included in the data set, all analysis is performed on the HTML source only. However, Diffbot has annotated the source with a small amount of helpful information. Specifically,

[1]Sadly, Relevanseek was the only submission in the contest though many teams signed up. Whether this means others were unable to produce competitive programs, or whether they dropped out for some other reason, is unknown to the author.

selected HTML element tags are annotated with an extra element describing how the element is rendered in Diffbot's browser implementation. For example, the image of the helicopter is described by the following annotated HTML tag:

```
<img src="..." alt="..." _="x=54,y=421,w=250,h=153,mbo=5px,dis=inline">
```

The extra "_" attribute describes the rendered (x, y) position and size, the bottom margin (5px), and the display mode (inline). This information can be used to obtain valuable features.

2 Architecture

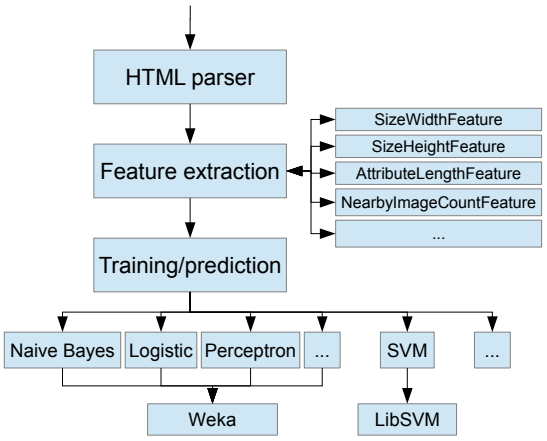

Figure 2: Processing flow of one document through Relevanseek's pipeline.

Relevanseek is written entirely in Java. Its core pipeline consists of three principal components. The same pipeline is used whether the intent is to train, evaluate training using cross-validation or other methods, or to predict unknown samples.

2.1 Page parser

As the first step in the pipeline, the parser builds a DOM using jsoup[2], and then sanitizes and normalizes the resulting tree. The resulting well-formed DOM is then converted to Java's native DOM structure. From this, the the parser builds a list of images and extracted image features (more on this in §3). If labels are known, then the parser associates each image with its label. The parser produces a strongly-typed data structure to avoid potential bugs caused by inconsistent interpretation of matrix values.

2.2 Classifier interfaces

The parser hands off extracted information to one of the many classifiers. The classifiers are designed to be pluggable and extend a common interface `IClassifier` in order to facilitate rapid development feedback and short iterations. Switching from one classifier to another to test different models requires only swapping half a line of code. Most of the classifiers (e.g. `NaiveBayesWekaClassifier`, `LibSVMClassifier`) simply perform translation between Relevanseek's native format and the underlying library's required format, but it is also possible to write custom classifiers. For example, the `LeastSquaresLinearRegression` OLS classifier[2] is implemented using only JAMA[3], a generic matrix library.

The classifier architecture also allows parameters (e.g. bias for unbalanced data set) and composition (e.g. AdaBoost) or other arbitrary refinements without any changes in the application layer.

2.3 Classifier backends

To avoid reinventing the wheel, we reuse existing high-quality machine learning libraries. Several classifiers are wrappers for Weka[4] and LibSVM[5], which both have pure Java interfaces and implementations. By placing the backends behind this abstraction layer, arbitrary backends can be used by Relevanseek without any changes to the application layer.

[2]While the OLS classifier was useful for rapid early prototyping, its results are not included here because the features being used led to singular matrices.

3 Feature extraction

Relevanseek reduces each image to a vector of features along with a binary label. About 30 of these features are static, including:

- Size of image
- Distance between image and the four edges of the rendered webpage
- Length of the `alt` and `title` attribute values
- Similarity of `alt` and `title` value values compared to header and title text
- Number of other images rendered close by
- Number of other identically-sized images
- Whether image's bounding box overlaps with elements like `<p>` or `<article>`
- Whether the image is a hyperlink
- The size of the smallest rendered ancestor
- The position of the first text-heavy ancestor element

Two of the features are generated based on the training data:

- Whether the image or its ancestor elements contain certain strings in their `class` and/or `id` attributes
- Whether the image has an ancestor of some specific tag name (e.g. `<figure>`)

Each feature is implemented as a class extending a common `IFeature` interface. Similar to the common interface for all classifiers mentioned previously, this allows rapid feedback for feature performance and enables automated feature selection.

4 Results

4.1 Classifier performance

Because of the unbalanced data, even the trivial "always irrelevant" classifier achieves 96.4% accuracy. Thus, we use the F1 score to measure performance.

Classifier performance is determined using 4-fold cross validation, training each classifier on 75% of the labeled documents and predicting image relevance in the remaining documents. The training set consists of 600 documents containing 25509 images, of which 906 are relevant.

The cross-validation performance is as follows:

Classifier	Accuracy	Precision	Recall	F-Score
SVM (RBF)	0.972	0.601	0.625	0.613
Logistic*	0.971	0.614	0.461	0.527
Perceptron*	0.963	0.475	0.464	0.469
Naive Bayes*	0.933	0.320	0.796	0.457
Always irrelevant	0.964	-	0.000	-
Always relevant	0.036	0.036	1.000	0.069

* With AdaBoost

4.2 Learning curve

The learning curve for the SVM classifier is shown here. While precision and recall on the training set remains high throughout the curve, accuracy on the test set during cross-validation plateaus after about 300 documents. Although the learning curve may suggest high variance and overfitting, each one of the many features contributes positively towards testing accuracy and removing any of them reduces testing accuracy. One challenge of this particular problem is the large amount of information included in each HTML page; it is difficult to represent all of the relevant information into a concise set of real numbers.

Given more time to continue this strategy and implement more features, we believe the testing accuracy plateau can be shifted up and to the right, and an F-score of 0.75-0.80 is reasonably achievable.

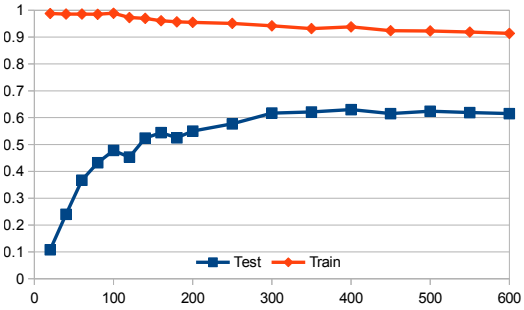

Figure 3: SVM testing and training F-scores during cross-validation with 4 folds

5 Limitations and future work

Relevanseek compresses information about each image into a single feature vector, which may be overly restrictive. In addition, this creates the implicit assumption that the classification of each image is independent of others in the same document. Building more complex models could improve performance. For example, the popular AdBlock program detects advertisements primarily by using simple pattern matching against URLs, which is not well-suited for classification using numeric features; a similar strategy could be added to Relevanseek to efficiently eliminate several classes of irrelevant images.

Another potential improvement is filtering dirty data. Diffbot provides a training set of about 600 labeled documents, but the labels are not always accurate (some labels are malformed, some are missing, some are duplicated, some are wrong based on manual judging by the author). Exploring options to either clean up the data as a preprocessing step or to filter out bad training data may improve performance.

Finally, one quirk of this task is that Diffbot provides the `<img src="...">` attribute in the training data, but not the evaluation test set, even though the URL of the image provides useful information regarding the relevance of the image. For example, a simple image feature describing the end of the image URL (e.g. `.png` vs `.html` etc) improved accuracy in cross-validation, but had to be removed for classifying the test set. Similarly, another possible feature could be generated by comparing the domain of the image source to the domain of its containing webpage, or to look for certain keywords in the URL (e.g. "share").

6 Conclusion

We have described a supervised learning classifier for determining whether images in an HTML file are relevant to their containing article. The documents are parsed and images are processed into vectors of carefully chosen features, and several learning algorithms are compared for accuracy. By choosing an appropriate classification algorithm, relevant images can be chosen with high precision and recall.

References

[1] Diffbot Technologies. Diffbot's Machine Learning Challenge, 2013. `http://diffbot.com/robotlab/DiffbotContest/` [Online; accessed 17 Nov 2013].

[2] Jonathan Hedley. jsoup: Java HTML Parser. 2013. `http://jsoup.org/`.

[3] Joe Hicklin, Cleve Moler, Peter Webb, Ronald F. Boisvert, Bruce Miller, Roldan Pozo, and Karin Remington. JAMA: Java Matrix Package. 2012. `http://math.nist.gov/javanumerics/jama/`.

[4] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 2009. `http://www.cs.waikato.ac.nz/ml/weka/`.

[5] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.