Ryan Levering, Michal Cutler

Cost-Sensitive Feature Extraction and Selection in Genre Classification

Automatic genre classification of Web pages is currently young compared to other Web classification tasks. Corpora are just starting to be collected and organized in a systematic way, feature extraction techniques are inconsistent and not well detailed, genres are constantly in dispute, and novel applications have not been implemented. This paper attempts to review and make progress in the area of feature extraction, an area that we believe can benefit all Web page classification, and genre classification in particular. We first present a framework for the extraction of various Web-specific feature groups from distinct data models based on a tree of potentials models and the transformations that create them. Then we introduce the concept of cost-sensitivity to this tree and provide an algorithm for performing wrapper-based feature selection on this tree. Finally, we apply the cost-sensitive feature selection algorithm on two genre corpora and analyze the performance of the classification results.

1 Introduction

A classification task cannot achieve high performance without being provided a good set of features, regardless of the algorithm that is being used to train the computer. For instance, if you were attempting to train a computer to recognize dogs from other animals and the only measurement you took was the number of legs the animal had, it would be an impossible task. This paper is primarily concerned with the goal of obtaining thorough measurements, known as features, from a particular automatic classification domain: Web pages.

The features that we are discussing would most likely be useful for any number of classification tasks within the domain of Web pages. Ideally, there would be a general way, independent of classification task, to measure the effectiveness of a feature in representing the information on a Web page. For instance, when we are dealing with text documents, a bag-of-words model is a fairly good representation of the kind of content on the page. If word order was also included in the model, it would come much closer to representing all the information that is encoded in the document. However, Web pages are very high-dimensional concepts that layer semantic and visual annotation on top of the already rich semantics of text. This makes it much harder to theoretically verify the merit of a feature set. Therefore, the merit is more easily measured with practical experiments in which the features are used.

While we want to achieve a high accuracy with Web page classification, we recognize that practical Web-scale classification implementations cannot afford to compute every possible feature to achieve the maximal accuracy. Even if they could, they should prefer to minimize the time to achieve that accuracy. Therefore, the work presented in this paper focuses on finding a balance between accuracy and efficiency in feature extraction. As an example, if a particular genre class of Web pages is easily recognizable from the URL, it should not be necessary to actually perform a network fetch. While this can be decided manually, we believe that this sort of analysis can be incorporated into the machine learning process to better effect.

We have chosen to evaluate our feature extraction and selection methodology in the context of automatic genre classification tasks. The meaning of *genre* is quite complex and there are many different definitions in the information science literature. For the reader unfamiliar with genre in the context of Web page classification, we advise reading sections of more comprehensive works such as Santini (2007) or Boese (2005). Empirically, common examples of Web page genres include such labels as "FAQ", "News Article", "Company Home Page" that encompass a common perception of a document type.

Genre classification is a good choice to evaluate Web page features because genres of Web pages often need a larger amount of contextual information on the page to make a classification decision and therefore actually require more interesting features. Genre classification, on the machine learning side has always found merit in and been defined by features that traditional topical classification ignored. Stamatatos et al. (2000) found that stop words, which are typically thrown out in topical classification, were good indicators of genre. Toms and Campbell (1999) found that users could identify genre based on visual layout and spacing information. Later in our own research Levering et al. (2008), we corroborated that measuring this visual information could help with certain automatic genre classification tasks. As an example, HTML tag counts are almost always included in feature sets for Web genre classification tasks, when they are frequently stripped out in topical classification.

The general difference in the two types of automatic classification is that the feature spaces of the two types of classification are often orthogonal. From a human perspective, a particular topic will span multiple genres of representation and the genre of a document often does not imply a particular textual content. At the same time, certain genres are in practice correlated with particular topics. Additionally, there are features that are useful in both types of classification. The larger size and diversity of the feature space means that more care and sensitivity needs to be given to feature extraction and selection in genre classification.

This paper will start off in Section 2 with an introduction to a more descriptive methodology of feature extraction on Web pages. Once this method is presented, in Section 3 we will identify several useful data models that we can analyze to obtain Web page measurements. Then in Section 4 we will perform a review of current feature groups used in Web page classification using this more formalized methodology and the models from the previous section. In Section 5 we will introduce a way to use this extraction methodology to perform a measurement cost-sensitive feature selection across a large general set of feature extraction techniques. In the final section, we will demonstrate both the feature selection algorithm and our set of features on two previously used genre corpora.

2 Feature Framework

In order to compare different methods of classifying Web pages, the extracted features should be comparable. In basic text classification, this is not as great of an issue. Visually interpreted from a character level, text has two obvious forms of abstract representation. First, the visual model that is the concrete level on which authors generally write:

$$\text{glyph} \in \text{line} \in \text{page} \in \text{document}$$

On top of this is the semantic model that actually conveys the meaning of the document. Recognized visual cues (spacing/line breaks/punctuation) from the first textual level are used to produce a more semantic model, for example:

$$\text{character} \in \text{word} \in \text{sentence} \in \text{paragraph} \in \text{section} \in \text{document}$$

This model can produce most of the features that the field is familiar with: bag-of-words, sentence counts, punctuation counts, pattern matches. These features have generally established semantics and procedures for extraction that are agreed upon by a community and thus are comparable across classification experiment.

However, even these accepted features are often used without explanation and can often lead to obfuscated problems and results in classification. Boundary cases, noise, and parameters have to be dealt with in any algorithm implementation. One implementation might throw away any words with less than three letters plus stop words and not count sentences that cannot be parsed (like fragments within ellipses). These details are typically lost during most communication channels, such as papers and emails between researchers. It is also rare to find agreement on a particular software package used to perform feature extraction. While these details often will not make a difference, sometimes they will make analyzing features for classification a frustrating issue.

Another supplementary problem is that this fairly small list of models explodes when you attempt to analyze Web pages. HyperText Markup Language (HTML), the language in which most Web pages are written, is a semantic as well as visual expression language. It allows almost complete flexibility in the order and manner in which text is displayed. There are at least two obvious models that are layered on top of the already existing models for text representation: the HTML model itself, often called the Document Object Model (DOM) in interpreted, hierarchical form and the rendered model which is the way the document is drawn to the screen by the browser for the viewer. We could even go further and say there is a semantic model on top of that visual model that could represent how the author intends the page to be interpreted by the viewer. The point is that feature extraction in HTML becomes a much harder problem. This is because:

1. There are more models to work with and extract potential information from

2. The transformations between these models are sometimes very complex (in the rendering step for instance) and not agreed upon

3. The models themselves have more dimensions (for instance, graphically two dimensions) in addition to nested textual semantics

For all of the reasons above, we propose that a better framework for comparing feature extraction techniques needs to be established. This paper presents some established and new features we have found useful in the context of this feature framework. We envision feature extraction as a tree of model transformations that starts with a single initial data model and ends with a set of feature groups. In the diagram below, each of the circle nodes represent models that are used to potentially extract features from. The squares represent transformations that convert one type of model to another. The triangles are the actual feature groups that are extracted from each model to perform classification.

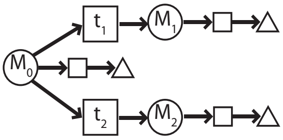

Figure 1: Feature extraction as tree of transformations.

A model is an abstract representation of the original data or a transformation of that data. Thus, the feature groups that result from the transformation steps are also instances of this generic model. A transformation is a higher-level function that takes a set of input models and produces a different set of output models.

These models and transformations are each identified by a unique URI (Uniform Resource Identifier). In the case of a transformation, this URI globally defines the algorithmic behavior and is ideally backed by a specification, or at least a detailed description, identifying working behavior and configuration parameters. In the case of a model, this URI identifies the structure and semantics of the model. This means that each feature in the end feature group is no longer just an identifier, but actually a transformation path that describes specifically how the feature was obtained. The worth of these identifiers are obviously dependent on the level of detail of the specifications or descriptions that define the transformations and models, but at the least they serve as a constrained language for discussing feature extraction.

3 Web Models and Transformations

Presented in Figure 2 is a summarization of the feature extraction tree that we use in our Web page analysis. Not included are all the leaf feature groups that are generated through extraction transformations.

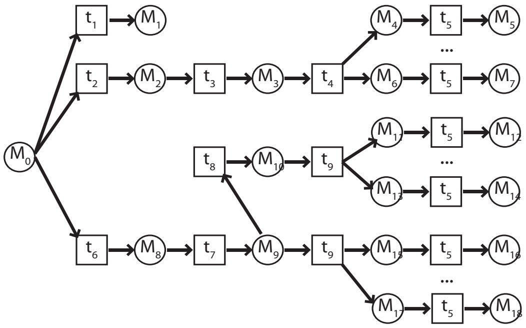

Figure 2: Feature extraction tree to convert an initial URL into models to use in feature extraction.

In order to reuse techniques for feature extraction, we often will create transformations to convert to existing model types. For instance, all the leaf models in Figure 2 are all the same type (fragmented text) even though they represent different textual contents (i.e. extracted URL tokens vs. content tokens).

Many of these transformations deal with two different implementations of the core text extraction path. One of these paths uses a browser-driven rendered DOM version and the other uses a more simplistic DOM version. The general idea is illustrated in Figure 3, which is a simplified view of Figure 2 that collapses similar transformations (listed below the node title) into recognizable steps in a common Web page feature extraction path. We relate some of the underlying details of these models and transformations. Included in them are URI references relative to our own namespace.

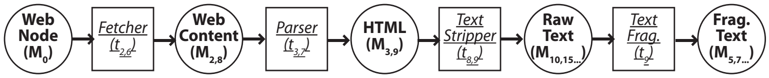

Figure 3: Core transformation path.

Web Node (`<model/www-node>`): The model all the way to the left is the Web node model. This is a more realistic form of a URL expanded to include state that is needed for the HTTP request. Any sophisticated general-purpose crawler can possibly track cookie state or issue POST requests that will drastically influence the produced Web page. Therefore, our model of a Web page location needs to be expanded to deal with this.

Fetcher: We use two different types of fetchers to fetch Web nodes. The first is a modified version of Apache Nutch's fetcher that does single-threaded pure HTML fetching (`<transformation/fetcher/nutch>`). This is very fast and in addition to the standard URL fetching it uses intelligent parsing to handle meta-redirects without a full document parse. The second is a rendered document fetcher that downloads the complete Web page (`<transformation/fetcher/swt>`). This fetcher loads the document using Mozilla's backend rendering engine and then saves the complete page with image, style, and script resources very accurately. By interpreting the page scripts, the HTML output by this fetcher is generally much more accurate than the output produced by the Nutch fetcher.

Web Content (`<model/www-content>`): This is the raw content found at a webnode. It is composed of the resulting URL (which may differ from that requested) and raw bytes, possibly including resources from the fetched URL depending on the fetcher.

Parser: For our lightweight transformation path, we convert the results of a single HTML page into an HTML DOM model using JTidy (`<transformation/generator/tidy>`). This library makes similar heuristic decisions to those made by a browser about how to deal with poorly formed HTML. To generate our rendered HTML model, we use the same rendering fetcher library mentioned above and then build a DOM based on Mozilla's own internal DOM model (`<transformer/generator/swt>`). This includes visual positions used to display the nodes in addition to useful style annotations that we can use to pass on extra information like element visibility.

HTML (`<model/html>`): A parsed HTML document, represented as a hierarchical in-memory Document Object Model.

Rendered HTML (`<model/rendered-html>`): A HTML DOM model annotated with position (height, width, x, y) for each one of the DOM nodes.

Text Stripper: To generate the text, we traverse the DOM model and output text nodes, generating line breaks when certain text-breaking HTML nodes are encountered. The only difference between the rendered HTML text stripper (`<transformer/renderedhtml-text>`) and the basic HTML text stripper (`<transformer/html-text>`) is the extra visibility information that is used so that non-visible text is not converted.

Text (`<model/text>`): Basic string text, decoded bytes interpreted as a character array.

Text Fragmenter: This is a lookahead parser that fragments English text by using punctuation heuristics along with abbreviation lookups (`<transformer/text-ftext>`). It is most likely not as accurate as a heavy-weight statistical parser, but it is very fast and provides much better results then simply splitting on punctuation/delimiters. Web pages often do not have well-formatted sentence structure and it is not uncommon to have Web pages without any complete sentences in the text. Being able to handle fragments as well as common word punctuation (dates/urls/etc) keeps us from losing this extra information when we extract features from the text.

Fragmented Text (`<model/ftext>`): Heuristically, fragmented text that is broken down into a linear sequence of word and punctuation, grouped by sentence or fragment if the parser did not think it was a valid sentence. This model is much more accurate for token analysis than delimiter split text.

4 Web Feature Extraction Groups

Generally, many features and models have been extracted from Web pages at some point in the literature, though most of them have been understated or implicit. In this section, we hope to highlight explicitly some of the main categories of Web page features in the context of the models they are derived from, along with some examples of how they can be used. We use our formal notation to explain how we represent those features in our system.

4.1 Web Node Feature Groups

Genre classification can often be surprisingly effective with URL driven features. As several very rough examples, a tilde could imply a personal home page, short URL length could imply corporate home page, digit count could imply time-sensitive or version sensitive genres. This is likely more true in genre classification than in topical text classification which is more dependent on a richer topical vocabulary.

We generate simple character statistics on the URL (`<extractor/www-content/url>`) and then tokenize the URL intelligently (`<transformer/www-ftext>`) and apply all of our fragmented text feature extraction techniques (see Section 4.3).

4.2 Text Feature Groups

Because most of the features we use are based on our fragmented text model, we do not do much analysis on the raw text. We count characters of different types such as digits, alphanumeric, and punctuation (`<extractor/text/simple>`) and create a dynamic punctuation vocabulary (`<extractor/text/punctuation>`).

4.3 Fragmented Text Feature Groups

The core of fragmented text features is the dynamic vocabulary extraction (`<extractor/ftext/dynamic-vocab>`), which gives us the very common bag-of-words text word counts. Because of the fragmentation, these have already had punctuation dealt with. We do not eliminate stop words because they are often good genre indicators (Stamatatos et al., 2000), but we do perform stemming. Stemming has long been used in text classification as a simple way to combine frequencies of words with similar roots and thus similar semantic concepts. These features are very important to classifier performance as they pick up on genre-sensitive vocabulary.

To add some extra abstraction ability to the classifier, we also include a number of common token patterns such as several different date formats, integer and decimal numbers, several common time formats, words in upper case, and words in title case (`<extractor/ftext/pattern>`). Finally we also have some basic fragmentation statistics like sentence counts, word counts, etc. (`<extractor/ftext/simple>`).

4.4 HTML Feature Groups

The primary HTML features frequently used in Web page have been tag counts (`<extractor/html/tag>`). These have been mentioned in many classification papers such as Karlgren et al. (1998), Joachims et al. (2001) and Boese's 2005 feature review. Generally, most people at least intuitively agree that high tag counts can be indicative of the type of a Web page. For instance, heavy TABLE (a layout annotation) tag usage may indicate complex layout or many IMGs (representing page images) may indicate a visually complex page and hubpages are essentially defined by having many A (hypertext link) tags. They can also be useful for discovering special cases of Web pages that rely on the presence of a more specific tag (like a file upload tag).

To these basic features we add several sets of slightly more interesting abstractions. Form elements get their own treatment, so we count the number of checkboxes, radio buttons, etc (`<extractor/html/form>`). These features can point to interactive genres (contact, feedback). We also count table and HTML depth that can be used to detect complex page layouts (`<extractor/html/depth>`). Counts of event handlers on tags (`<extractor/html/events>`) can point to more interactive, JavaScript-laden genres. Finally, we analyze the link tags to see what type of file and what domain they are pointing to (`<extractor/html/link>`). This is especially useful for a download or link type genre.

4.5 Rendered HTML Feature Groups

One advantage of a rendered DOM comes when you use it to "clean" standard DOM features. The DOM produced is structurally the same as a non-annotated DOM so it can be used in the same way. For instance, often there are parts of a Web page that are dynamically generated or that are listed in the HTML source but are not actually visible on the page. With rendering information, you can very easily tell that these elements are not included and thus have a potentially cleaner version of the page contents. This especially applies to site home pages, where dynamic visual updating often plays a large part of the user experience. Most features generated without rendering are essentially guesses or approximations of what is actually going on when the page is rendered.

Other features can be generated by examining overall statistics of the extra visual positions. These were discussed in Levering et al. (2008) in the context of genre classification as being useful for identifying certain genre that were dependent on content-type statistics (like containing an image heavy header followed by a lot of text).

Finally, we use the rendered HTML to generate subtree models of the HTML that occur in important parts of the visual document (`<transformer/html-section>`). This allows us to apply all of the same feature extraction techniques to generate location-aware feature counts. For instance, it allows the classifier to have such information as "a date appears near the top of the document". In this paper, we use this to analyze the center of the page as in Kovacevic et al. (2002). Because these are fairly specific features, they lend themselves to finer grained genre classification tasks and tend not to show up in feature extraction on broad genres.

5 Cost-Sensitive Feature Analysis

It is impossible to talk about many of these more complex features without also talking about their measurement cost. Any implementation of a genre classification system applied to the Web has to be efficient and most likely scalable to a large number of documents. Luckily, viewing feature extraction as a transformation tree lends itself very well to parallel processing. Each transformation is a discrete functional process and can be batched for large-scale runs in a distributed architectural paradigm like MapReduce (Dean and Ghemawat, 2008).

There is often a diminishing return on the calculation of more complex features, particularly for certain fine grained genres with dominant features. For instance, if a genre or set of genres in a multiclass problem were detectable to some level of accuracy from the URLs and it cost ten times as much computational power to calculate rendered HTML features that would improve accuracy a small percentage, it may not be worth it to calculate those features for a one million document universe let alone the entire Web.

This measurement cost can be factored into the classification process itself using techniques from other classification areas where measurement cost is even more crucial. In Paclík et al. (2002), they use a greedy wrapper-based algorithm to do feature selection based on classifier performance with different groups of features that share computation cost. This was designed for independent feature group measurement costs in the image classification domain, but can be adapted to our Web feature extraction tree paradigm fairly well.

In the following paragraphs, we present an algorithm that performs cost-sensitive feature selection on a feature extraction tree. We chose to simplify the analysis by assuming that once a model is selected all its features can be added at zero cost. We also assume that all these features are presented to the classification algorithms (which may or may not perform additional feature selection). We plan to expand the algorithm shown here to work with more general transformation graphs and to include also the cost of feature extraction for some more costly features.

We assume also that each transformation t of the feature extraction tree has a cost c_t. This is the average computation time required by the transformation t of a model to a child model in the tree. The algorithm depends on a performance evaluation function, $Eval$, that evaluates the performance of a classification task when all features of a set of models S, $features(S)$, are presented to it. The first model included in S is the WebNode model M_0. The goal of the algorithm is to return a set of models S, with a low cumulative computation cost, whose features will produce a performance $P = Eval(features(S)) \geq P_{goal}$.

Let $adjacent(S)$ be the set of models that are not yet in S and are adjacent to at least one model in S. For each model $M \in adjacent(S)$ we can compute the performance improvement per unit of cost achieved by adding M to S. Let $S' = S \cup M$, and let P' be the performance measure of S', the performance improvement $R_{S'} = (P' - P)/c_t$. The next model added to S by the algorithm is the model $M_{max} = argmax\{R_{S'} | M \in adjacent(S)$ and $S' = S \cup M\}$. Using a more complex ratio function or a more complex evaluation of the cost would allow for a more customized feature selection.

Algorithm 1 Greedy feature extraction tree search

```
 1: $S \leftarrow \{\}$
 2: $P = Eval(features(S))$
 3: while $P < P_{goal}$ do
 4:     $R_{max} = -\infty, P_{max} = 0, S_{max} = \{\}$
 5:     // Select best model in $adjacent(S)$
 6:     for all $M$ in $adjacent(S)$ do
 7:         $S' = S \cup M$
 8:         $P' = Eval(features(S'))$
 9:         if $R_{S'} > R_{max}$ then
10:             $R_{max} = R_{S'}, P_{max} = P', S_{max} = S'$
11:         end if
12:         $S = S_{max}, P = P_{max}$
13:     end for
14: end while
```

Another possibility with a higher analysis cost is to flatten the feature extraction tree by consolidating all ancestor models into a single model set that contains the cumulative models up to a certain point in a tree. Then we could evaluate every node path at the same time and iteratively choose the one with the best performance gain to measurement cost ratio. This approach would translate to the more greedy previous algorithm if you evaluated any descendant path as opposed to just an adjacent edge. This would also translate the problem more closely into the type of problem in Paclík et al. (2002).

Finally, at the far end of the complexity of analysis scale, an exhaustive search of every combination of feature paths could be done to guarantee the best choice of models for the highest classification performance to cost ratio. If the classification task was fully offline (not recalculated frequently) or the tree was very simple, this would be the best option. For our purposes, this search was not practical due to training cost.

The first two techniques do have the caveat that they depend on classifiers that are resistant to noisy, useless features. Otherwise, greedy choices may not find an optimal solution when an intermediate transformation produces poor features. Paclík et al. (2002) approached this by proposing a nested feature subset selection step while evaluating a feature group. We use SVMs for evaluation, which we find in practice are very good at ignoring useless features.

6 Experimental Results

We chose to evaluate our feature extraction tree on two datasets to emphasize several different outcomes of feature selection. We used the same transformation costs on both of the datasets. In reality, the transformation costs are dataset dependent, but the magnitudes tend to be fairly consistent. In addition, our corpora are cached HTML pages and so getting accurate fetch times was difficult. In addition, one of the datasets does not include dependent resources (scripts, styles, and images) and therefore accurate renderings were not possible.

We use the greedy tree search discussed above to determine the order of models to generate for feature extraction. This produces a graph with an increasing performance over cost. By setting the P_{goal} to 1.0, we can see the maximum performance of the search algorithm.

6.1 KI-04 Dataset

The KI-04 dataset presented in Eissen and Stein (2004) is a universal set of Web super-genres that can theoretically encompass any Web page. Though it does suffer from some granularity issues (Santini, 2006), it is one of the most complete genre corpora. The lack of a large negative class makes it a genre palette where a multi-class classification makes sense.

It is composed of eight genres: download, article, help, FAQ, private portrayal, non-private portrayal, shop, and linked list. It has over one hundred of each genre type, a palette that was created via user survey.

To evaluate both the feature selection algorithm and the worth of various features on the dataset, we use a ten-fold cross-validating multiclass linear SVM as our *Eval* function. This is the Weka (Witten and Frank, 2005) implementation that uses a pair-wise voting scheme to choose a single class after comparing the results of multiple binary SVMs. At several points, we also ran a multiclass J4.8 classification to validate our results manually by examining the generated decision tree.

Performance in this experiment was measured using accuracy, or number of correct classifications divided by the number of total classifications. This was just to make it consistent with the KI-04 results. However, in a multiclass problem with no large negative class, the accuracy results tend to be consistent with more balanced metrics such as F_1, which we use in the next experiment.

The results are presented in Figure 4. The graph highlights the tradeoffs in classification performance with increasing computation cost. Each data point in the graph corresponds to the cost and performance (in F_1-measure) of a particular feature set that the feature selection algorithm selected as it traversed the cost-weighted feature transformation tree. A labeled version of that tree, showing the evaluated performance

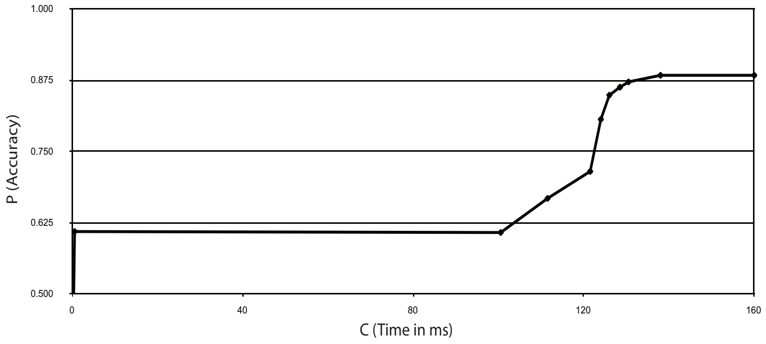

Figure 4: KI-04 feature extraction performance vs. costs.

as we include additional models is shown in Figure 5. Those performance numbers are obtained by measuring the classification performance of the union of the labeled model with every model with a lesser performance value. Also note that having continuously incrementing performance values by adding models is not common. Often there is no gain by including additional models and sometimes they will actually lower the performance, even with noise-resistant classifiers like SVMs.

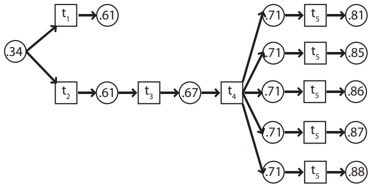
$$\begin{tikzcd}
 & t_1 \arrow[r] & .61 & & & .71 \arrow[r] & t_5 \arrow[r] & .81 \\
 & & & & & .71 \arrow[r] & t_5 \arrow[r] & .85 \\
.34 \arrow[uur] \arrow[r] & t_2 \arrow[r] & .61 \arrow[r] & t_3 \arrow[r] & .67 \arrow[r] & t_4 \arrow[uu] \arrow[u] \arrow[r] \arrow[d] \arrow[dd] & .71 \arrow[r] & t_5 \arrow[r] & .86 \\
 & & & & & .71 \arrow[r] & t_5 \arrow[r] & .87 \\
 & & & & & .71 \arrow[r] & t_5 \arrow[r] & .88
\end{tikzcd}$$

Figure 5: KI-04 feature extraction performance graph.

As an example of that, we did not include any rendered DOM-driven features including visual distributions in this result as their cost-performance ratio was very poor on this corpus and they were added at the very end for no gain. This is due to the previously mentioned point that these features do not work well on very broad non-visual genres in addition to the fact that the dataset was not collected with external resources (scripts, styles, and images), so the renderings were not always very complete.

The long line that stretches most of the graph is the fetch time. One interesting point is that we were able to achieve over sixty-percent accuracy with only the URL. A combination of URL vocabulary like ‘FAQ’, ‘thread’ and ‘download’ combined with URL path lengths did a moderately successful job of classification without any of the overhead of actually fetching the page. If we had trained binary classifiers to just distinguish FAQ or download genres in this corpus, these results would have been much higher.

After the fetch time, which clearly dominates the computation time, the algorithm did not have too many choices in our current graph without the useless visual features. It added HTML features, then text statistics, then it chose between fragmenting different extracted text types (headings, links, title, emphasized, and all). Interestingly, it found the tradeoff worst on the *all text* category and had better luck with categories like link and heading text. This suggests looking at particular sets of annotated text as opposed to entire vocabulary lists in this type of genre classification.

The research in which this dataset was first presented achieved an overall 70% accuracy on the entire dataset. Thus, our ideal feature set with a maximum F_1-measure of close to 0.9 shows a dramatic overall improvement on the corpus. We theorize that the primary reason for this is the inclusion of URL-based features that appear to have not been used in their analysis.

6.2 Retail Store Dataset

To contrast with several aspects of the KI-04 dataset, we also performed an evaluation on the retail store dataset that we used in previous research (Levering et al., 2008). This dataset has a genre palette fully within the retail store universe of Web pages. It has three positive, labeled genres - store home page, store product list, and store product page - with over a hundred examples of each. It also has a large negative class of other pages in the retail store Web page universe. The goal is to train binary classifiers to be able to recognize those types of pages out of the noise of the entire Web site.

In that paper, we concluded that visual features were useful for this particular problem and raised the accuracy of classification by a significant amount. We wanted to revisit that now with measurement cost factored in to figure out what that gain is costing.

Performance in this experiment was measured as F_1-measure. This is a standard classification metric that balances precision and recall. Generally, there is an inverse relationship between these two metrics, since you can always overtrain a classifier to improve precision at the cost of recall. F_1 provides a single metric that penalizes either measurement being poor.

We evaluated the dataset using a ten-fold cross-validating linear SVM like for the previous experiment but with a single classifier for each of the three positive classes against every other class. This way, we were able to generate a separate performance-cost graph for each classification. The graphs are shown in Figure 6.

One of the most dramatic things about this graph is, like in the previous dataset, just how well a URL alone predicts genre with an extremely low computation cost. You could never use a system with an F_1-measure around 0.5 as shown in the figure for *store-products*, but it does show the feature group's worth. On the other hand, it was intuitively obvious that *store-fronts* were easy to figure out from the URL and the classifier agreed with a nearly perfect F_1-measure.

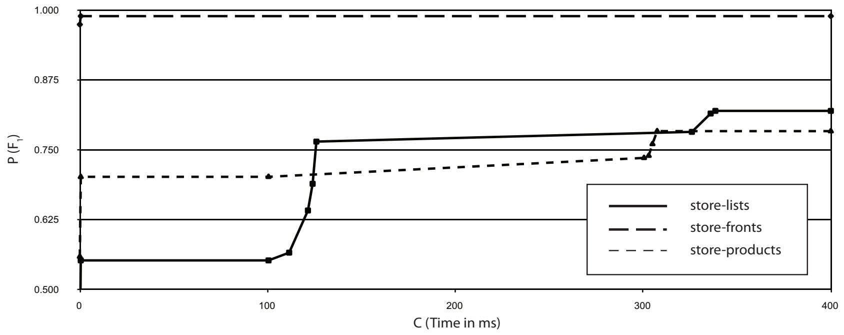

Figure 6: Retail store feature extraction performance vs. costs.

The *store-lists* genre favored non-rendered textual models until it stopped getting improvements, whereupon the search algorithm finally paid the performance cost to render the page in exchange for slightly increased performance. It then produced several more inexpensive models for visually central feature groups that were slightly more accurate than the general feature groups.

Our *store-products* genre did not perform well on non-rendered HTML features (the F_1-measure actually decreased), so the search algorithm opted to render the page and then had several more textual feature groups that it found useful. This is a case of the algorithm choosing incorrectly. If the algorithm included a look-ahead or the second approach was used, it most likely could have found gains with textual features after generating the HTML model without paying the rendering cost.

In all, this experiment verified that while visual features did improve performance on this classification task, simpler features could get nearly the same level of performance. We theorize that it could be useful to have a multi-tiered classifier that first extracts lightweight features. If these features strongly indicate a certain genre, then we can stop extraction; otherwise, we extract more complex features. This is left to future work.

7 Conclusion

The goal of this research was neither to point out interesting facets of the experimental datasets nor to show improvement on classification tasks. Yet at the same time, we made some gains on both of these fronts. Both of our experiments showed that genre is particularly sensitive to URL features and some effort should be spent on more interesting tokenizations and patterns of URLs. Web pages are not just text documents and in a classification task every bit of relevant information should be used.

We improved the accuracy on the KI-04 dataset by using some dynamic URL tokens and textual vocabulary. More importantly, it was done without any focused effort. The same process was applied on both classification tasks to extract features. By having a process to analyze a new classification task on a very generic and powerful feature extraction platform and then perform a cost-sensitive feature selection, we insure that we get acceptable performance while not using unnecessary transformation or extraction techniques.

The real goal of this research was to attempt to break down the process of Web page feature extraction into smaller functional units (transformations) that could be more easily compared and analyzed. We proposed a tree-based abstraction for feature extraction where intermediate models can have their feature groups extracted. Transformations between these models are represented by URIs that convey the semantics of a particular transformation.

One of the benefits of this feature extraction tree is that we can perform a cost-sensitive feature selection as described in this paper. Having intermediate models also would allow researchers to more easily build complex features without repeating earlier work. Finally, by using methods backed by URIs that represent certain algorithmic choices, researchers can more quickly and accurately communicate results.

This work has many directions for improvement. A methodology without a concrete tool that supports the methodology is quickly forgotten. A tool or library to allow sharing of common transformation and extraction techniques would be very productive, much in the same way that Weka has improved the ease of classification research.

Also, there are always more interesting features to be discovered. Visual features may not improve performance on all genre classification tasks, but the Web is becoming a more visual, media-intensive environment and Web classification researchers need to find ways to use these extra layers of information.

References

Boese, E. (2005). Stereotyping the web: Genre classification of web documents. Master's thesis, Colorado State University.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Eissen, S. M. and Stein, B. (2004). Genre classification of web pages: user study and feasibility analysis. *KI-2004: Advances in Artificial Intelligence*, pages 256–269.

Joachims, T., Cristianini, N., and Shawe-Taylor, J. (2001). Composite kernels for hypertext categorisation. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 250–257, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Karlgren, J., Bretan, I., Dewe, J., Hallberg, A., and Wolkhert, N. (1998). Iterative information retrieval using fast clustering and usage-specific genres. In *Eighth DELOS Workshop - User Interface in Digital Libraries*, pages 85–92.

Kovacevic, M., Diligenti, M., Gori, M., and Milutinovic, V. (2002). Recognition of common areas in a web page using visual information: a possible application in a page classification. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining*, page 250, Washington, DC, USA. IEEE Computer Society.

Levering, R., Cutler, M., and Yu, L. (2008). Using visual features for fine-grained genre classification of web pages. In *HICSS '08: Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 131, Washington, DC, USA. IEEE Computer Society.

Paclík, P., Duin, R. P. W., Kempen, G. M. P. v., and Kohlus, R. (2002). On feature selection with measurement cost and grouped features. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, pages 461–469, London, UK. Springer-Verlag.

Santini, M. (2006). Common criteria for genre classification: Annotation and granularity. In *Proceedings of the workshop on text-based information retrieval.*

Santini, M. (2007). *Automatic Identification of Genre in Web Pages.* PhD thesis, University of Brighton.

Stamatatos, E., Fakotakis, N., and Kokkinakis, G. (2000). Text genre detection using common word frequencies. In *Proceedings of the 18th conference on Computational linguistics*, pages 808–814, Morristown, NJ, USA. Association for Computational Linguistics.

Toms, E. G. and Campbell, D. G. (1999). Genre as interface metaphor: Exploiting form and function in digital environments. In *HICSS '99: Proceedings of the 32nd Annual Hawaii International Conference on System Sciences.*

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco.