Classifying Websites by Industry Sector: A Study in Feature Design

Giacomo Berardi, Andrea Esuli, Tiziano Fagni
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
56124 Pisa, Italy
E-mail: {firstname.lastname}@isti.cnr.it

Fabrizio Sebastiani[*]
Qatar Computing Research Institute
Qatar Foundation
Doha, Qatar
E-mail: fsebastiani@qf.org.qa

ABSTRACT

Classifying companies by industry sector is an important task in finance, since it allows investors and research analysts to analyse specific subsectors of local and global markets for investment monitoring and planning purposes. Traditionally this classification activity has been performed manually, by dedicated specialists carrying out in-depth analysis of a company's public profile. However, this is more and more unsuitable in nowadays's globalised markets, in which new companies spring up, old companies cease to exist, and existing companies refocus their efforts to different sectors at an astounding pace. As a result, tools for performing this classification automatically are increasingly needed. We address the problem of classifying companies by industry sector via the automatic classification of their websites, since the latter provide rich information about the nature of the company and market segment it targets. We have built a website classification system and tested its accuracy on a dataset of more than 20,000 company websites classified according to a 2-level taxonomy of 216 leaf classes explicitly designed for market research purposes. Our experimental study provides interesting insights as to which types of features are the most useful for this classification task.

Keywords

Website classification, Industry sectors

1. INTRODUCTION

Having companies and activities classified by industry sector is important to research analysts, fund managers, and investment managers, since this allows them to partition the market into homogeneous segments and to monitor and compare industry trends across different sectors and subsectors. Market trends indicate that stocks within the same industry sector often perform similarly during the same temporal interval; for this reason, it is important for investment managers to know the industry sector in which a given company operates. Additionally, this allows companies to have up-to-date views of who their competitors are, and to carry out competitive intelligence actions.

Classification by industry sector has a long tradition, and several taxonomies for classifying companies by industry sector have been developed and maintained over the years. One important such taxonomy is the ICB (Industry Classification Benchmark)[1], created in 2005 by Dow Jones and the FTSE Group, and used by the NASDAQ, NYSE and other stock markets worldwide. ICB consists of 184 nodes organised into a 4-level taxonomy; an example path (from root to leaf) of such a taxonomy is Financials $\rightarrow$ Insurance $\rightarrow$ Non-life Insurance $\rightarrow$ Property & Casualty Insurance. Other such taxonomies are the Global Industry Classification Standard (GICS)[2], a 4-level taxonomy of 256 nodes jointly developed by MSCI Inc. and Standard & Poor's and widely used in the financial sector, and the Thomson Reuters Business Classification (TRBC)[3], a 5-level taxonomy of 1065 nodes operated by Thomson Reuters.

Traditionally, the activity of classifying companies under these taxonomies has been performed manually, by dedicated specialists who perform an in-depth analysis of a company's public profile. One problem with this way of operating is that manual classification work is expensive, especially when carried out by senior employees. Even more importantly, manual classification work is increasingly unsuitable in nowadays's globalised market, in which new companies spring up, old companies cease to exist, and existing companies refocus their efforts to different sectors at an astounding pace. For some applications, manual classification work may also simply be too slow for responding to suddenly arisen business needs. As a result, tools for performing this classification automatically are sorely needed.

We address the problem of classifying companies by industry sector via the automatic classification of their websites. Website classification is different from webpage classification since the decision under which class the website should be classified should be taken based on the nature of the entire website, and not of a webpage alone.

We have built a website classification system based on

[*]Fabrizio Sebastiani is on leave from Consiglio Nazionale delle Ricerche.

[1]http://www.icbenchmark.com/
[2]http://www.msci.com/products/indexes/sector/gics/
[3]http://thomsonreuters.com/business-classification/

supervised learning techniques; the system can work both in "hard classification mode", placing each company website into exactly one leaf node in the taxonomy, and in "soft classification mode", returning a ranked list of the k leaf nodes that appear the most plausible for the website. The system thus lends itself to working as a fully autonomous classifier (hard classification), or as assistive technology to users who then need to take the final classification decision themselves (soft classification).

We have tested the accuracy and efficiency of our system on a dataset of more than 20,000 URLs identifying company websites. The URLs are classified according to a 2-level taxonomy of about 250 leaf classes explicitly devised for market research purposes. The results of our experiments indicate that website classification can be performed at a high accuracy level and with good efficiency. In reporting this experimental study we place particular emphasis on the description of the impact that specific types of features have on classification accuracy. In particular, we study the impact of *endogenous* features (i.e., features extracted from the website itself), and the impact of *exogenous* features (i.e., features extracted from sources external to the website itself, but somehow referring to the website). Our experimental study provides interesting insights as to which types of features are most useful for this classification task.

The rest of the paper is structured as follows. In Section 2 we describe our website classification system, especially focusing on the feature extraction phase. Section 3 describes our experimental setup, and Section 4 presents and discusses the results of our experiments. Section 5 discusses related work, while Section 6 concludes, pointing at avenues for future research.

2. A SYSTEM FOR WEBSITE CLASSIFICATION

In this section we describe our website classification system in detail, focusing in particular on the feature extraction phase.

2.1 Crawling websites

The first step of the process consists in crawling the URLs in our dataset. The crawler we have used "pretends" to be the "googlebot" Google crawler, since some websites provide more information about themselves when the crawler is a search engine crawler, and this additional information may be useful to the classification process. Preliminary experiments that we had run by crawling the URLs in the standard way had shown inferior accuracy.

The crawler works in two phases:

1. It retrieves the homepage and the subpages of the websites[4]. In order to keep the computational cost of the modules downstream within reasonable limits, we limit the crawl to the first 10 subpages found for each website, in breadth-first order.

2. It retrieves exogenous information, such as Twitter profiles or Facebook pages, if linked from the homepage, and Alexa queries, if present; see Section 2.2.2 for details.

[4]We only retrieve HTML data, and do not instead download any image / video content, nor perform any Javascript code interpretation. Javascript interpretation could in principle be used by websites to perform AJAX-based dynamic content loading. However this is usually adopted to perform background loading of secondary information (e.g., widgets, social buttons, multimedia content) which is unlikely to contribute to our task, and is rarely used to load the main content of webpages, since this latter content should be loaded as fast as possible.

2.2 Feature Extraction

2.2.1 *Endogenous features*

The first type of features we extract are *endogenous* features, i.e., information to be found in the website itself. We extract features from the following fields marked up in the HTML structure:

- **URL**: Given a URL u, after downcasing all its characters we extract all *character n-grams* (for any $n = 3, 4, 5$) and all "words" contained in u, where a character n-gram is any sequence of n characters different from separators (defined as the characters ".", "/", "_") and a "word" is any maximal such sequence, i.e., a sequence of non-separator characters that occurs in u between separators. For instance, from the URL `http://www.google.com/`, the words extracted are `www`, `google`, and `com`, while the 4-grams extracted are `goog`, `oogl`, and `ogle`. The rationale is that some of these sequences may be highly indicative of class membership; for instance, the 3-gram `xxx` and the 4-gram `porn` clearly point to the Adult leaf class.

  The URL field is the only field from which we extract character n-grams; the rationale is that other fields, such as Title and Body, do not constrain the user to be as concise as the URL field does, which means that in fields other than URL it makes sense to consider the word as the minimal unit of content.

- **Title**: Given the title of a webpage, we extract all the words contained in it and perform stop word removal and stemming.

- **MetaDescription**: Same as for "Title".

- **MetaKeywords**: Same as for "Title".

- **Body**: Same as for "Title".

- **Headings**: Same as for "Title", and carried out only for tags `<h1>`, `<h2>`, `<h3>`, `<h4>`. The extracted features are pooled together and treated as belonging to generic "Headings" field.

In order to keep the computational cost of the modules downstream within reasonable limits, for each of "MetaDescription", "MetaKeywords", "Body", we limit the content extraction for each webpage to the first 2000 characters.

We also extract a number of "structural" features, such as

- **UrlLength**: Represents the length (number of characters) of the URL of the website home page. This might be useful, e.g., to identify websites (such as `https://bitly.com/`) belonging to class UrlShorteners, since such websites typically have very short URLs.

- **LinksCount**: Represents the maximum (across the pages in the website) number of outgoing links. For instance, a high number of outgoing links might be indicative of class WebDirectories.
- **HeadingsCount**: We compute the maximum (across the pages in the website) numbers of headings of type `<h1>`, `<h2>`, `<h3>`, `<h4>` (these are kept separate and thus give rise to four different subtypes of features).
- **ParagraphsCount**: Represents the maximum (across the pages in the website) number of paragraphs, as identified by the `<p>` tag.

All of these structural features are framed as binary features of the form $c \leq a_i$, where c is the count and a_i is a natural number. For instance, **UrlLength** ≤ 8 is a feature whose value is 1 if the length of the URL of the website home page is smaller or equal than 8, and 0 otherwise. For feature type **UrlLength**, we add one binary feature for each different home page URL length a_i instantiated in the dataset; same for the other three feature types.

2.2.2 Exogenous features

The second type of features we extract are *exogenous* features, i.e., information that is to be found outside the website but that refers to the website anyway. The exogenous features we extract are the following:

- **Alexa**: For any given website we query (the freely accessible version of) Alexa[5], a popular service for Web analytics, and obtain back from it the 5 most frequent queries among the ones addressed to major web search engines that have resulted in clicks on a link to our website. In other words, these queries are the ones which have sent the most traffic to this specific website during the months before we crawled it. The rationale of using this information is that, if we know that a user has issued, e.g., the query "replacement car battery" to Google and has clicked on a link, among the ones returned by Google, leading to our website, we may hypothesize that our website may belong to class CarParts. From these queries we obtain features by extracting all the words contained in them and performing stop word removal and stemming.
- **Facebook and Twitter**: Many companies now have a Facebook or a Twitter page which contains a short textual description of the company and of its mission; these pages are often referred to from within the company's website, typically from the home page. Whenever we find a link to a Facebook or a Twitter page within a website, we extract the univocal name the company adopts in the specific social network and we then exploit the specific web API for retrieving the desired data. We then extract all the words contained in the retrieved page and perform stop word removal and stemming.

  Facebook company pages have several fields, of which we consider the following: `about` (information about the page); `category` (the page's main category, e.g. Product/Service, Computers/Technology); `company_overview` (the company overview); `description` (the description of the page); `general_info` (general information provided by the page); `mission` (the company mission); `name` (the name of the page); `products` (the products of this company). Twitter company pages only have `name` and `description` fields.

We qualify all the features we extract, endogenous and exogenous, by a prefix that indicates which field the feature comes from, with the goal of placing different emphasis on features coming from different fields (e.g., a word may have more importance if it was extracted from the page title). For example, the word `google` extracted from the URL `http://www.google.com/` is represented as `URL:google`, while when the same word is encountered in the title of the webpage it is represented as `TITLE:google`.

2.3 Language Identification

We analyse each webpage in the website, and each page downloaded from Facebook or Twitter, in order to classify it according to the language the page is worded in. We detect language using the Language Detection library[6]. This library implements a naïve Bayesian learner trained on a corpus of Wikipedia pages in different languages; the learner exploits the character n-grams extracted from the texts as features, and builds a single-label multi-class classifier for the languages in the training set. For the purposes of this experimentation we discard from consideration all websites none of whose pages are deemed to be in English by this language detector. For all other websites, only pages deemed to be in English are retained.

2.4 Feature Selection

The feature extraction process described in Section 2.2.1 returns a number of features too high to be used in practice. In order to keep the computational effort to a manageable level, and in order to avoid overfitting, we perform feature selection via the usual "filtering" approach [6], which consists in (a) scoring each feature t_k according to its estimated contribution to discriminating class c_j from the other classes, and (b) retaining only the highest-scoring ones. As the scoring function we use *information gain* (also known as *mutual information*), defined as

$$\begin{aligned} IG(t_k, c_j) &= H(c_j) - H(c_j|t_k) = \\ &= \sum_{c \in \{c_j, \bar{c}_j\}} \sum_{t \in \{t_k, \bar{t}_k\}} P(t,c) \log_2 \frac{P(t,c)}{P(t)P(c)} \end{aligned}$$

Here $H(\cdot)$ indicates entropy, $H(\cdot|\cdot)$ indicates conditional entropy, and probabilities are evaluated on an event space of training instances, where $P(t_k)$ and $P(\bar{t}_k)$ represent the fractions of instances that contain feature t_k and do not contain feature t_k, respectively, and $P(c_j)$ and $P(\bar{c}_j)$ represent the fractions of instances that belong to class c_j and do not belong to class c_j, respectively. $IG(t_k, c_j)$ measures the reduction in the entropy of c_j obtained as a result of observing t_k, i.e., measures the information that t_k provides on c_j.

We then adopt a "round robin" policy [4] in which the n (internal and leaf) classes take turns in picking a feature from the top-most elements of their class-specific orderings, until the desired number x of features are picked. In this way, for each class c_j the final set of selected features con-

[5] `http://www.alexa.com/`

[6] `http://code.google.com/p/language-detection/`

tains at least[7] the $\frac{x}{n}$ features that are best at discriminating c_j, which means that all the classes are adequately championed in the final feature set.

In the experiments reported in this paper we retain 30% of the original features, bringing the total number of 1,325,645 features to a slightly more manageable number of 397,693 features.

2.5 Feature Weighting

All the selected features are weighted by means of the well known BM25 weighting function (see e.g., [10]); initial experiments carried out on a development set and using standard $tf * idf$ had returned inferior results. In BM25, the weight $w(t, d)$ of feature t for website d is defined by

$$w(t,d) = \frac{tf(t,d) \cdot (k_1 + 1)}{tf(t,d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avgdl})} \cdot idf(t) \qquad (1)$$

where $tf(t, d)$ is the number of occurrences of feature t in website d, $avgdl$ is the average number of non-null features (endogenous and exogenous) for a website, k_1 and b are parameters, and $idf(t)$ is

$$idf(t) = log\frac{(N - |\{d : t \in d\}|) + 0.5}{|\{d : t \in d\}| + 0.5} \qquad (2)$$

where N is the total number of websites in the collection. In our experiments we have set k_1 and b to the values suggested in [10], namely, $k_1 = 1.2$ and $b = 0.5$.

2.6 Classifier Training

As the learning algorithm we have used the TreeSVM configuration described in [3]. Essentially, this consists of running a binary SVM learner[8] at each nonroot node x in the hierarchy, using as negative examples for x all examples belonging to the parent node of x that belong to some sibling of x and not to x itself. As the binary SVM-based learner we have used the freely available `libsvm` software package[9]. We have used a linear kernel with the parameter `C` (the penalty parameter of the error term) set to 1; this parameter setting, while simple, achieves a good trade-off between model complexity and effectiveness, and is the default setting used in the `libsvm` software.

2.7 Classification

At the end of the learning process, TreeSVM has generated a hierarchical classifier in the form of a tree of binary classifiers, one for each nonroot node. The classifiers work in "Pachinko machine" style, i.e., only the instances that have been attributed to class x by the associated binary classifier are fed to the binary classifiers associated with the children of x. For a given instance a binary classifier returns both a binary decision ("assign" or "not assign") and a classification score, where high score means high probability of belonging to the class. In "hard classification" mode the highest-scoring leaf class is assigned to the instance, while in "soft classification" mode the k highest-scoring leaf classes are ranked in decreasing order of their score and associated to the instance.

[7]The "at least" here means that, since the same feature may be a high-scoring feature for more than one class, strictly more than $\frac{x}{n}$ features per class may eventually get selected.

[8]In preliminary experiments we had tried to use MpBoost [2] as the base learner, with inferior results.

[9]`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

We have modified the original behaviour of TreeSVM to better handle the simultaneous presence of the hard and soft classification modes. The original TreeSVM is designed to work as a multi-label classifier, so that zero, one, or several leaf classes at the same time can be attributed to an instance. This causes problems in our application context, since the original TreeSVM (a) does not guarantee that the instance is attributed to at least one leaf class, thus making hard classification problematic, and (b) does not even guarantee that classification scores for at least k leaf classes are generated for an instance[10], thus making soft classification problematic.

To overcome these two problems, when classifying an instance we heuristically force, at each level in the hierarchy, the assignment of at least three internal nodes. The instance can thus be fed to all the children of these internal nodes, and this tends to guarantee that a large enough number of leaf classifiers are invoked and thus return a classification score for the instance. When in hard classification mode, the top-scoring class is selected; when in soft classification mode, a ranked list of the top-scoring k classes is returned (in our experiments we take $k = 5$).

3. EXPERIMENTS

For our experiments we have used a classification scheme and a dataset of URLs that were provided to us by a customer. Note that we had no control on the design of the classification scheme and on the choice of the dataset; we thus take both as given[11].

3.1 The Classification Scheme

The classification scheme consists of a taxonomy of industry sectors of depth 2, with 27 internal (depth-1) classes and 248 (depth-2) classes. All leaf classes are at depth 2, and all depth-1 classes have at least two children classes. The taxonomy is severely imbalanced, with internal nodes having from a minimum of 2 to a maximum of 21 children nodes[12].

One aspect that makes classification under this taxonomy difficult is that the taxonomy mixes issues of topic and genre[13]. For instance, the leaf classes ShoppingSearch and ShoppingBlogs have two different parent classes (WebPortals&Search and Blogging, respectively); some distinctions at level 1 are based on topic (e.g., Automotive vs. ComputersAndInternet), while some others are based on genre (e.g., Blogging vs. NewsAndMedia). Classification within this taxonomy is thus very challenging, since it is sometimes the case that different nodes are very similar in meaning, making the task of telling them apart extremely difficult.

[10]To see this, assume that the taxonomy has two levels, that the instance is rejected by all level-1 nodes but one, and that this one node has only $(k - 1)$ leaf classes as its children; in this case, only for these $(k - 1)$ leaf classes a classification score is generated.

[11]Unfortunately the data and the classification scheme are property of this customer, and we are not in a position to make them publicly available.

[12]For reasons internal to their organization, our customer decided not to avail themselves of the standard taxonomies discusses in the introduction, and thus developed their own custom taxonomy.

[13]Classification by genre, especially in the web domain, is sometimes called "functional classification"; see e.g., [9].

3.2 The Dataset

Our dataset of URLs originally contained 69,100 URLs, each labelled with exactly one leaf class from our taxonomy. After the crawling phase we discarded from consideration the 46,893 websites that proved unreachable. Most of the errors returned by unreachable hosts were due to bad HTTP status (e.g., "404" errors) or unknown IP address[14]. We further discarded the 2,169 websites that our language recognition module deemed void of English-language content. This left us with (69,100-(46,893+2,169)) = 20,038 websites to experiment with. Of the original 248 leaf classes in the taxonomy we discarded the 32 that proved empty (i.e., none of the 20,038 websites was classified under them), thus leaving 216 leaf classes to work with.

3.3 Evaluation measures

As mentioned in previous sections, our website classification system can work both in "hard classification mode", placing each company website into exactly one leaf class in the taxonomy, and in "soft classification mode", returning a ranked list of the k leaf classes that appear the most plausible for the website. We thus evaluate our system according to two different evaluation measures, each suitable for a different mode.

For evaluating the effectiveness of the system at hard classification we have used the well-known F_1 function, defined as

$$F_1 = \frac{2\pi\rho}{\pi+\rho} = \frac{2TP}{2TP+FP+FN} \tag{3}$$

F_1 corresponds to the harmonic mean of *precision* (π) and *recall* (ρ); here, TP, FP, and FN, stand for the numbers of true positives, false positives, and false negatives, respectively. Note that F_1 is undefined when $TP = FP = FN = 0$; in this case, consistently with most other works in the literature, we take F_1 to equal 1.0, since the classifier has correctly classified all instances (as negative examples). It is customary to average the class-specific F_1 scores across the classes by computing both *microaveraged* F_1 (denoted by F_1^μ) and *macroaveraged* F_1 (F_1^M). F_1^μ is obtained by (i) computing the class-specific values TP_i, (ii) obtaining TP as the sum of the TP_i's (same for FP and FN), and then (iii) applying Equation 3. F_1^M is obtained by first computing the F_1 values specific to the individual classes, and then averaging them across the c_j's. Both averages are computed across *all* classes, internal and leaf alike.

For evaluating the effectiveness of the system at soft classification we instead use a variant of *reciprocal rank* (RR). The RR of an instance d_i is defined as

$$RR(d_i) = \frac{1}{r(d_i)} \tag{4}$$

where $r(d_i)$ is the rank attributed by the system to d_i's true class. We use a variant of RR that we call RR$_k$, defined as

$$RR_k(d_i) = \begin{cases} \dfrac{1}{r(d_i)} & \text{if } r(d_i) \leq k \\ 0 & \text{if } r(d_i) > k \end{cases} \tag{5}$$

That is, no credit at all is given to the system for placing d_i's true class at rank $(k+1)$ or higher, since we only return the top k classes for each website.

Again, we compute both microaveraged and macroaveraged versions of RR_k, noted RR_k^μ and RR_k^M, respectively. RR_k^μ is obtained as the average of $RR_k(d_i)$ across all the test instances, while RR_k^M is obtained by first computing the class-specific averages of RR_k and then averaging the results across the classes. Again, both averages are computed across *all* classes, internal and leaf alike.

4. RESULTS

The results of our experiments are reported in Table 1; all results were obtained by 10-fold cross-validation (10FCV).

The first observation that can be made by looking at the microaveraged figures (F_1^μ and RR_k^μ) reported in the table is that the accuracy is surprisingly high. For instance, the system equipped with all feature types at the 30% reduction level (last row, columns 7-11) obtained $F_1^\mu = 0.808$ and $RR_k^\mu = 0.852$, which are very high values. Just to put these values into context, the $RR_k^\mu = 0.852$ value resulted from the true class of the website being ranked in 1st place 80.8% of the cases, 5.9% of the cases in 2nd place, 2.9% in 3rd place, 1.4% in 4th place, 0.9% in 5th place, and in only 8.1% of the cases it was ranked outside the top 5 positions. This is a very good result especially in the light of the fact that there are no less than 216 classes to pick from, and picking the single one that applies to the website certainly qualifies as finding a needle in a haystack.

Figures for macroaveraged measures (F_1^M and RR_k^M) are lower, but this was to be expected in the light of the severely imbalanced nature of the dataset. In fact, the micro- and macro-averaged versions of a measure yield the same value only when the dataset is perfectly balanced, while the former yields higher values than the latter when the dataset is imbalanced. In our case, the most frequent class (BloggingServices) totals 3347 instances while the least frequent one (AccountancyAndTaxServices) has only 1 instance, which is an indication of the severe level of imbalance of this dataset.

4.1 Effects of Different Feature Types

One of the key aspects in designing a website classification system is feature design, i.e., (a) deciding what type of features to extract, and (b) which among the extracted features to retain. The reason is that the size of the full set of features extracted via a process like the one of Section 2.2 may be daunting, even for datasets of modest size. For the dataset that we use in the present work, no less than 1,325,645 features are generated from the 20,038 websites the dataset consists of.

In an attempt to better understand the quality of the features contributed from each of the fields identified in Section 2.2, for each such field we have run experiments by using the features extracted from that field only; the results are reported in Table 1, each row representing a specific feature type. While these experiments do not account for the subtle interactions that may take place as a result of the copresence of features of different types, they nonetheless give an indicative idea of the relative contribution that the different types of features can give to the overall classification task. We have run such experiments (a) with the full feature set (Columns 2-6 of Table 1), and (b) with the set resulting from the feature selection process (Columns 7-11 of Table 1). This latter process consisted in putting into a common pool all the extracted features of all types, and then selecting the

[14]The dataset contains a considerable number (20,911) of URLs directed to Google static content, i.e., `gstatic.com`, which turned out to be unreachable during the crawling.

Table 1: Accuracy of classifiers obtained by using all extracted features (Columns 2 to 6) or by using selected features only (Columns 7 to 11). In Column 7, percentages indicate the survival rate of the specific feature type. Boldface indicates best results for a specific combination of feature selection level (no selection or 30% selection) and evaluation measure.

	No Feature Selection					With Feature Selection				
	#Features	F_1^M	F_1^μ	RR_5^M	RR_5^μ	#Features	F_1^M	F_1^μ	RR_5^M	RR_5^μ
URL	670,340	0.497	0.797	0.532	0.843	211,250 (31.5%)	0.491	0.797	0.534	0.842
Title	38,801	0.367	0.639	0.420	0.705	12,076 (31.1%)	0.331	0.621	0.390	0.694
Body	319,355	0.329	0.604	0.370	0.670	78,913 (24.7%)	0.328	0.618	0.373	0.686
Headings	153,552	0.319	0.597	0.364	0.653	46,858 (30.5%)	0.320	0.606	0.361	0.660
MetaDescription	42,897	0.299	0.492	0.321	0.542	15,297 (35.7%)	0.291	0.505	0.316	0.553
MetaKeywords	32,139	0.201	0.297	0.232	0.341	11,464 (35.7%)	0.200	0.322	0.229	0.371
LinksCount	96	0.019	0.161	0.034	0.215	96 (100.0%)	0.019	0.161	0.034	0.215
ParagraphsCount	39	0.014	0.171	0.028	0.198	39 (100.0%)	0.014	0.171	0.028	0.198
HeadingsCount	39	0.012	0.141	0.029	0.185	39 (100.0%)	0.012	0.141	0.029	0.185
UrlLength	12	0.002	0.076	0.009	0.088	12 (100.0%)	0.002	0.076	0.009	0.088
All endogenous feature types	1,257,270	0.469	**0.791**	0.517	**0.841**	376,044 (29.9%)	0.472	0.797	0.523	0.844
Alexa	36,061	**0.552**	0.754	**0.567**	0.804	9,309 (25.8%)	**0.522**	0.696	**0.548**	0.773
Facebook	24,164	0.201	0.204	0.196	0.241	9,689 (40.1%)	0.203	0.254	0.197	0.300
Twitter	8,150	0.181	0.112	0.174	0.131	2,641 (32.4%)	0.175	0.156	0.172	0.197
All exogenous feature types	68,375	0.527	0.726	0.554	0.792	21,639 (31.6%)	0.499	0.679	0.540	0.764
All feature types	1,325,645	—	—	—	—	397,693 (30.0%)	0.490	**0.808**	0.539	**0.852**

best ones irrespective of their type, so that different feature types compete with each other. In Column 7 we report, for each feature type, the *survival rate* of a given feature type, i.e., the percentage of features of that type that made it into the final selected set.

The first observation we can make is that, as it could be expected, the "All feature types" setting is overall the best, i.e., there is no single feature type, or group of feature types, that outperforms it; while the "Alexa" setting slightly outperforms it for macro measures, it is substantially outperformed by "All feature types" for micro measures.

A second observation is that some individual endogenous feature types (e.g., "URL", "Title", "Body") deliver very good accuracy by themselves. Using URL features alone is actually competitive with using *all* feature types, which is somehow surprising, given that URLs are hardly a convenient means for authors to convey semantics. The success of this feature type has likely to do with the need, on the part of authors, to cram as much information as possible in a short string that must be both descriptive and evocative to prospective customers. "Title" features also perform well, and this is intuitive, since authors tend to use highly significant words in page titles; the same argument applies, although to a slightly smaller degree, to "Headings" features. Also the fact that "Body" features are helpful is unsurprising, since the body of the page is where content is meant to be conveyed.

"MetaDescription" and "MetaKeywords" features deliver smaller accuracy levels than the previously discussed types. We believe the main reason for this to be the fact that, for many webpages, these fields are left completely empty by their authors. This means that, if all pages of the website have these fields empty, the website is classified completely at random by the "MetaDescription" and "MetaKeywords" classifiers. As for the remaining endogenous feature types ("LinksCount", "UrlLength", "HeadingsCount", "ParagraphsCount"), each of them in isolation delivers very little accuracy, but this is obviously due to the fact that these feature types only contain a few dozen features each.

As for the exogenous features, the "Alexa" features deliver extremely high performance, even surpassing (as already noted above) "All feature types" when it comes to macro measures. The fact that these features excel for macro measures means that they particularly excel for infrequent classes; this is intuitive, since queries tend to be highly descriptive of the content of websites that they send clicks to, and the information they convey can thus make up for the scarcity of endogenous information, a problem that especially affects infrequent classes.

The "Facebook" and "Twitter" features are much less helpful than the "Alexa" features. However, this need not mean that pages from social media are less informative than queries. Indeed, one of the problems here is that, while "Alexa" queries are available for most websites in the dataset (18,478 websites), only 4,370 websites in the dataset have a corresponding Facebook page, and only 4,237 have a Twitter page. This means that, similarly to what happens for the "MetaDescription" and "MetaKeywords" classifiers, the "Facebook" and "Twitter" classifiers classify websites that do not have a Facebook and Twitter page completely at random. So, the percentage of websites that have a Facebook and Twitter page is the *de facto* upper bound to the percentage of websites that get correctly classified by the "Facebook" and "Twitter" classifiers.

4.2 Effects of Feature Selection

As for feature selection, the results indicate that it is beneficial when applied to endogenous features (where small improvements are observed for all four measures), but the contrary is true for exogenous features, where deteriorations are observed for all four measures as a result of feature selection. This is likely due to the fact that endogenous features are almost 20 times as many as exogenous features, which means that the former are potentially responsible for overfitting much more than the latter are, and are thus much more in need of trimming.

Some feature-specific classifiers tend to lose accuracy as a result of feature selection. One of them is "Title"; the likely reason for this is that, since titles consist of few words only, if all the words contained in the title of a website have been filtered out by the feature selection process, this website will be classified at random by the "Title" classifier. Also the "Alexa" classifier loses accuracy as a result of feature selection, which is yet another confirmation of how informative queries are, especially for infrequent classes.

Other feature-specific classifiers instead benefit from feature selection. One such example is "Body"; this confirms well-known results from text classification [11], which had shown that bag-of-words classifiers (i.e., classifiers using as features the words in the body of the text) can benefit from moderately aggressive feature selection. Even more interestingly, two other such examples are "Facebook" and "Twitter"; their case is similar to "Body", in the sense that company Facebook and Twitter pages consist of free text, which obviously contains many non-discriminative words too.

As for "LinksCount", "UrlLength", "HeadingsCount", and "ParagraphsCount" features, none of them is filtered out by the selection process (they obtain 100% survival rate), which means that, even if they are very few, all of them are important for the learning process.

4.3 Learning curve

We have run additional experiments in order to plot a learning curve for our classifiers, with the goal of finding out if we are likely to obtain improvements should we provide additional training data. To run these tests, in each of the 10 "folds" of the 10FCV experiment the test set is retained in its entirety while only a randomly selected percentage p of the training set is retained. We have run tests with $p \in \{20\%, 40\%, 60\%, 80\%\}$, in "All feature types" mode; if the results obtained with $p = 100\%$ (which we have reported in the preceding sections) are higher than the ones for $p = 80\%$, this means that we probably have not yet reached an effectiveness plateau, and that there are still likely margins of improvement.

The learning curve displayed in Figure 1 (see also Table 2) indeed shows that effectiveness is still on the rise, both for the hard classification (F_1) and the soft classification (RR) modes. This indicates that the results discussed in the previous sections, while already good in themselves, could likely still be improved upon if more training data were available.

	F_1^M	F_1^μ	RR_5^M	RR_5^μ
20%	0.168	0.635	0.211	0.701
40%	0.270	0.706	0.303	0.763
60%	0.365	0.751	0.389	0.802
80%	0.423	0.781	0.454	0.828
100%	0.490	0.808	0.539	0.852

Table 2: Accuracy of classifiers (using all feature types) obtained by using increasing percentages of training data.

4.4 Efficiency issues

Our experiments have been run on a commodity machine Intel Core i7-980X Processor Extreme Edition equipped with

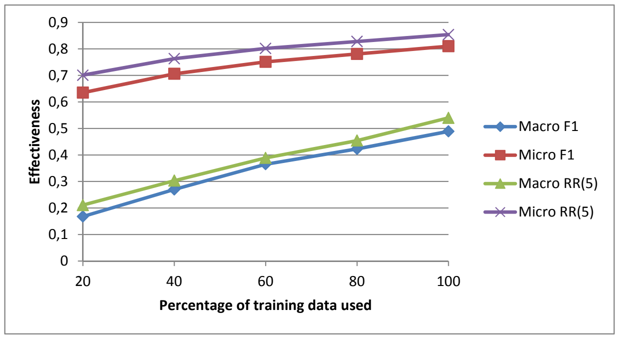

Figure 1: Learning curves for F_1^μ, F_1^M, RR_k^μ, RR_k^M, expressed as a function of p.

six 3.33Ghz cores and 24GB RAM. All experiments were run with sequential code except for the crawling part, where a multithreaded process was employed.

From the point of view of processing time:

1. Crawling the entire dataset (69,343 websites + info about them crawled from Alexa, Twitter, Facebook) required ≈ 3 days and 4 hours (at a rate of ≈ 5 websites per minute). This time also includes the effort wasted for attempting to crawl the 47,136 websites that, in the end, proved unreachable.

2. Content extraction for the entire dataset required less than 2 hours (at a rate of ≈ 185 websites per minute); this includes noise removal, feature extraction, indexing, and language identification.

3. For each of the 10 folds in the 10FCV experiment, training (on 90% of the dataset, i.e., 18,034 websites) required less than 4 hours, while classification (of 10% of the dataset, i.e., 2,003 websites) required ≈ 18 mins (at a rate of ≈ 108 websites per minute).

The above figures allow us to estimate that coding 1,000 new websites (after the classifier has already been trained) requires ≈ 3 hours and 35 mins. Of these, ≈ 3 hours and 20 mins (93.0% of the total effort) are needed for crawling, ≈ 6 mins (2.8%) are needed for extracting content, and 9 minutes (4.2%) are needed for classification. So, the largest amount of time is spent for crawling, a phase which is not entirely under our control since the speed at which it can be performed depends e.g., on the throughput of the server on which the website is hosted, and on the speed of the network connection. However, the crawling process is parallelized (each task is entrusted with a host, i.e., a website), so it can be sped up almost at will by increasing the level of parallelism. It should be remarked that the system is still in prototype form, and that margins of improvement in terms of efficiency are thus still large.

5. RELATED WORK

Website classification. Website classification is tackled in the work of Yang et al. [12] by framing the problem as one of *webpage* classification: for each website, the first 50 pages (in breadth-first fashion) from the site are crawled and merged into a single page, which is classified via webpage classification methods. In their work the emphasis is comparing different methods for making sense of hyperlinks pointing out of the website, and no sophisticated technique is attempted for either content extraction or feature selection; the problem of language identification is not mentioned in their paper, which seems to suggest that all the websites in their datasets are from the English-speaking world.

Webpage classification. A survey of techniques for webpage classification can be found in [9]. The issue of crawling and preprocessing webpages for their classification is akin to that of doing the same for their retrieval or for their clustering; on crawling and preprocessing for webpage clustering, see [1, Section 4]. The already mentioned work by Yang et al. [12] indeed tackles webpage classification according to industry sector, using a set of 4,285 URLs classified under a coarse-grained taxonomy of 28 classes and a more fine-grained one of 255 classes called Hoovers-28 and Hoovers-255, respectively; the datasets used in this research had originally been assembled by Ghani et al. [5]. Such a dataset, being almost 15 years old, is unfortunately unusable nowadays, since it may safely be assumed that many of the URLs in that dataset point to websites that do not exist anymore, or whose content has radically changed since then. As a consequence, results obtained nowadays are not comparable with the results obtained then, even if obtained on the same dataset; this is, of course, one of the problems of performing experiments on a moving target such as the Web.

Another attempt at webpage classification according to industry sector is the one in [8], which uses a depth-2 hierarchy of 71 classes and a dataset of 6,440 webpages. It should be noted, however, that [8, 12] are not real attempts at maximizing the accuracy of an operational website classifier, since they only use the datasets mentioned for investigating one specific aspect of this task (text classification under hierarchically-organized classification schemes in the case of [8], hypertext classification in the case of [12]), leaving other aspects unexplored. The use of information extracted from the URL for purposes of webpage classification is studied in [7].

6. CONCLUSIONS

We have presented a study on automatically classifying company websites by industry sector; this is a challenging classification task since it combines issues of topic with issues of genre, the two aspects being closely intertwined in classification schemes used for this task. The experiments we have carried out on a dataset of more than 20,000 websites classified according to a 2-level taxonomy of 216 classes have shown that the system can obtain very high accuracy values, guessing the true class of the website more than 80% of the times; this is an important feat, since picking the correct class from no less than 216 available classes amounts to finding the classic needle in the haystack.

We are currently working on extending the system to address the classification of websites expressed in languages other than English. This is an important step for today's globalised market.

7. REFERENCES

[1] Claudio Carpineto, Stanisiaw Osiński, Giovanni Romano, and Dawid Weiss. A survey of Web clustering engines. *ACM Computing Surveys*, 41(3), 2009.

[2] Andrea Esuli, Tiziano Fagni, and Fabrizio Sebastiani. MP-Boost: A multiple-pivot boosting algorithm and its application to text categorization. In *Proceedings of the 13th International Symposium on String Processing and Information Retrieval (SPIRE 2006)*, pages 1–12, Glasgow, UK, 2006.

[3] Tiziano Fagni and Fabrizio Sebastiani. Selecting negative examples for hierarchical text classification: An experimental comparison. *Journal of the American Society for Information Science and Technologies*, 61(11):2256–2265, 2010.

[4] George Forman. A pitfall and solution in multi-class feature selection for text classification. In *Proceedings of the 21st International Conference on Machine Learning (ICML 2004)*, pages 38–45, Banff, CA, 2004.

[5] Rayid Ghani, Rosie Jones, Dunja Mladenić, Kamal Nigam, and Seán Slattery. Data mining on symbolic knowledge extracted from the web. In *Proceedings of the KDD Workshop on Text Mining*, Boston, US, 2000.

[6] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference on Machine Learning (ICML 1994)*, pages 121–129, New Brunswick, US, 1994.

[7] Min-Yen Kan and Hoang O. Nguyen Thi. Fast webpage classification using URL features. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM 2005)*, pages 325–326, Bremen, DE, 2005.

[8] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 15th International Conference on Machine Learning (ICML 1998)*, pages 359–367, Madison, US, 1998.

[9] Xiaoguang Qi and Brian D. Davison. Web page classification: Features and algorithms. *ACM Computing Surveys*, 41(2), 2009.

[10] Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[11] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning (ICML 1997)*, pages 412–420, Nashville, US, 1997.

[12] Yiming Yang, Seán Slattery, and Rayid Ghani. A study of approaches to hypertext categorization. *Journal of Intelligent Information Systems*, 18(2/3):219–241, 2002.