Charset Encoding Detection of HTML Documents
A Practical Experience

Shabanali Faghani[1], Ali Hadian[1], and Behrouz Minaei-Bidgoli[1]

[1] Department of Computer Engineering, Iran University of Science and Technology, Tehran
{shabanali.faghani,ali.hadian}@gmail.com, b_minaei@iust.ac.ir

Abstract. Charset encoding detection is a primary task in various web-based systems, such as web browsers, email clients, and search engines. In this paper, we present a new hybrid technique for charset encoding detection for HTML documents. Our approach consists of two phases: "Markup Elimination" and "Ensemble Classification". The Markup Elimination phase is based on the hypothesis that charset encoding detection is more accurate when the markups are removed from the main content. Therefore, HTML markups and other structural data such as scripts and styles are separated from the rendered texts of the HTML documents using a decoding-encoding trick which preserves the integrity of the byte sequence. In the Ensemble Classification phase, we leverage two well-known charset encoding detection tools, namely Mozilla CharDet and IBM ICU, and combine their outputs based on their estimated domain of expertise. Results show that the proposed technique significantly improves the accuracy of charset encoding detection over both Mozilla CharDet and IBM ICU.

Keywords: Charset Encoding; HTML Markups; Multilingual Environments

1 Introduction

Since the beginning of the information age, many coding schemes have been designed to support different languages. However, early character encodings were designed to support a specific language, or a family of similar languages, hence almost all of them were bilingual, supporting Latin characters plus the specific local script. Those schemes were incompatible with each other, and were dependent on the operating system and the software being used. With the advent of globalization and development of the Internet, the variety of existing coding schemes had become a barrier in front of the information exchange. For a web-based application, e.g. a web browser, it is hard to support and integrate documents from different types of encodings. To address this issue, everyone should use a standard multilingual coding scheme, so that any application can easily decode it. Unicode is the de-facto universal charset encoding which provides a universal coding scheme, particularly in the web. However, despite many efforts to use Unicode as universal charset encoding, not every web page is converted to Unicode so far, and other local coding schemes are still being widely used.

From the language point of view, charset encodings can be categorized into three types: universal, multilingual, and local. Universal encodings, such as UTF-8 and UTF-16, are extensively used for the majority of contents in many languages. Likewise, multilingual charset encodings, such as ISO-8859 (for European languages) and ISO-2022 (for East Asian languages) families, are used for a class of languages whose individual orthographies are sufficiently similar. The most well-known multilingual charset encoding is ISO-8859 which supports languages with Latin scripts, such as European languages [1]. Local charset encodings are specific to a single language. For instance, Shift_JIS, EUC-KR, GB18030 and Windows-1256 are local encodings used for Japanese, Korean, Chinese and Arabic languages respectively.

In order to process a HTML web page, the charset encoding of the page should be detected at first. Charset encoding detection is the single point of failure in many web-based systems; if the detected charset is wrong, the results of any further processing on the page turns to be unreliable. In some web pages, the character encoding is explicitly specified in the Meta tag. Moreover, some HTTP servers provide clients with the information about the charset encoding of the requested web pages in the HTTP headers. As shown further in Section 5, approximately half of the famous web sites do not explicitly declare the encodings. In these situations, automatic identification of the charset encoding of the web pages is inevitable.

The remainder of this paper is organized as follows: A brief review of related works is presented in Section 2. We give an overview of the charset encodings in Section 3. Section 4 describes the new approach presented in this paper. Section 5 states test bed specification and results of the evaluations. Finally, Section 6 provides some concluding remarks.

2 Related Work

Only few works have been done on charset encoding detection for HTML documents, of which the most are based on machine leaning techniques [2]. For example, Russell et al. propose a Language and Encoding Identification (LEI) system, named SILC. They use Naïve Bayes classifier along with a combined trigram/unigram model [1]. Kim et al. investigated different feature sets, such as byte-level and character-level N-grams, for language and encoding detection. Also, they tested different learning algorithms, such as Support Vector Machine (SVM) and Naïve Bayes (NB). Their results show that a NB classifier with character-level features yields the best accuracy for charset detection and SVM with character-level features is the best configuration for language detection [2]. Similarly, Kikui suggests a similar approach and states that using word-unigrams for single-byte charsets and character-unigram for multi-byte charsets gives the best accuracy [5].

In addition to academic research, there are two open source implementations for charset encoding detection, namely Mozilla CharDet [3] and IBM ICU [4], which are widely used in practice. However, when applied to HTML web pages, their accuracy does not meet the requirements of the target applications.

Russell et al. also discuss about the relation between encoding and language detection. They state that in many cases, the language can be determined if the charset encoding is known. If the detected encoding is EUC-KR, for example, we are confident that the language is Korean. Therefore, in some specific cases, the encoding and language are two sides of the same coin. However, since language detection falls outside the scope of this paper, we only focus on charset encoding detection. Interested readers are referred to [1, 2] for more details. Also of note, we do not use machine learning techniques in this paper. Instead, we suggest a pre-processing phase, i.e. markup elimination, to improve the accuracy of charset detection tools. Using the markup-eliminated version of HTML documents, we suggest that the two charset detectors can be combined in a way that significantly improves the accuracy.

3 Charset Encodings

A character set, or so-called charset, is defined as a set of characters to be used in a special coding scheme, while encoding is defined as a mapping from an abstract character repertoire to a serialized sequence of bytes. In some cases a single charset is used in various encodings. For instance, the Unicode charset is used for UTF-8, UTF-16, etc. However, in many cases charsets and encodings have one-to-one mapping. Therefore, while the two terms are used interchangeably, they are different in nature. Nonetheless, charset encoding detection and charset detection refer to the same task, which is to heuristically guess the character encoding of a series of bytes that represent text [6].

In practice, charset encoding is more complex than a simple mapping from a character repertoire to the corresponding codes. IBM, for example, uses a three level encoding model named CDRA (Character Data Representation Architecture) to organize and catalog its own vendor-specific array of character encodings [7]. Similarly, Unicode uses a four level coding architecture [8]. In this section, we describe Unicode character encoding model based on a technical report by Unicode organization [8]. Further details on the literature and terminology about character encoding issues and a deep study about glyph, character and encoding can be found in [9].

A generic text T is composed of a sequence of characters $\{c_1, c_2, \dots, c_n\}$ which can be encoded as a serialized sequence of bytes $\{b_1, b_2, \dots, b_m\}$. The transition between characters and bytes in Unicode is mediated by four levels of representation, namely ACR, CCS, CEF, and CES.

3.1 ACR

Abstract Character Repertoire is the set of characters to be encoded, i.e. some alphabet or symbol set. It determines which potential characters can be represented in a special charset. The word ‘abstract’ implies that the characters typically have varying graphical representations.

3.2 CCS

Coded Character Set is a mapping from an abstract character repertoire to a set of nonnegative integers. A coded character set may also be known as a character encoding, a coded character repertoire, or a code page. The name 'code page' is used for East Asian character inventories. A code page is customarily presented in a tabular 'row-cell' form, where each <row, cell> index pair corresponds to a distinct integer.

3.3 CEF

Character Encoding Form is a mapping from a set of nonnegative integers that are elements of a CCS to a set of sequences of particular code units. A code unit is an integer occupying a specified binary width in computer architecture, which can be an 8-bit byte or a 16-bit and even a 32-bit integer. The sequences of code units do not necessarily have the same length. The code units of UTF-8, for example, vary from one to four 8-bit units.

3.4 CES

Character Encoding Scheme is a reversible transformation from a set of sequences of code units (i.e. from one or more CEFs) to a serialized sequence of bytes. A CES can be simple, compound or compressing. A simple CES uses a mapping of each code unit of a CEF into a unique serialized byte sequence in order, while a compound CES uses two or more simple CESs plus a mechanism to shift between them. A compressing CES maps a code unit sequence to a byte sequence while minimizing the length of the byte sequence.

It is important not to confuse a CEF and a CES; the CEF maps code points to code units, while the CES transforms sequences of code units to byte sequences. Also it should be noted that the CES must take into account the byte-order serialization of all code units used in the CEF that are wider than a byte. Table 1 presents four HTML elements saying 'Hello' in different languages along with the hexadecimal representation of three charset encodings for each element. The texts exhibited by h1 elements and their corresponding code units are shown in boldface. Also, the code units of non-ASCII characters are underlined. For each h1 element in the Table 1, we use charset encodings that support characters of the corresponding language. For example, GB18030 supports Chinese characters, but ISO-8859-1 does not.

4 The Proposed Method

The intuition behind our proposed method is that structural data in HTML documents, i.e. HTML markups and scripts, can drastically decrease the precision of the charset detector. This is due to the fact that most charset detection algorithms use statistical analysis to detect the charset of the given byte stream. At the same time, a HTML document contains HTML markups along with the contents of the document, each of which have different statistical properties. Markups are composed of English characters and specific symbols, such as wickets, slash, etc. The HTML markups use ASCII-supported symbols and characters and can be stored using ASCII character encoding. Besides, ASCII characters are stored with the same byte patterns in every character encoding scheme for the sake of compatibility. As a result, if there are lots of HTML markups in the text, it becomes hard to detect the encoding of the rest of the content.

Table 1. Four h1 elements along with their hexadecimal representation in three charset encodings

HTML Elements	**Charset Encoding Details**	
	Charset Encodings	***Sequences of bytes (hexadecimal representation)***
<h1 lang="en-US">**Hello world**</h1>	UTF-8	3c 68 31 20 6c 61 6e 67 3d 22 65 6e 2d 55 53 22 3e **48 65 6c 6c 6f 20 77 6f 72 6c 64** 3c 2f 68 31 3e
	ISO-8859-1	3c 68 31 20 6c 61 6e 67 3d 22 65 6e 2d 55 53 22 3e **48 65 6c 6c 6f 20 77 6f 72 6c 64** 3c 2f 68 31 3e
	Windows-1252	3c 68 31 20 6c 61 6e 67 3d 22 65 6e 2d 55 53 22 3e **48 65 6c 6c 6f 20 77 6f 72 6c 64** 3c 2f 68 31 3e
<h1 lang="fa-IR"> **سلام علیکم**</h1>	UTF-8	3c 68 31 20 6c 61 6e 67 3d 22 66 61 2d 49 52 22 3e **d8 b3 d9 84 d8 a7 d9 85 20 d8 b9 d9 84 d9 8a d9 83 d9 85** 3c 2f 68 31 3e
	Windows-1256	3c 68 31 20 6c 61 6e 67 3d 22 66 61 2d 49 52 22 3e **d3 e1 c7 e3 20 da e1 ed df e3** 3c 2f 68 31 3e
	MacArabic	3c 68 31 20 6c 61 6e 67 3d 22 66 61 2d 49 52 22 3e **d3 e4 c7 e5 20 d9 e4 ea e3 e5** 3c 2f 68 31 3e
<h1 lang="de-DE">**Grüß Gott**</h1>	UTF-8	3c 68 31 20 6c 61 6e 67 3d 22 64 65 2d 44 45 22 3e **47 72 c3 bc c3 9f 20 47 6f 74 74** 3c 2f 68 31 3e
	Windows-1252	3c 68 31 20 6c 61 6e 67 3d 22 64 65 2d 44 45 22 3e **47 72 fc df 20 47 6f 74 74** 3c 2f 68 31 3e
	ISO-8859-9	3c 68 31 20 6c 61 6e 67 3d 22 64 65 2d 44 45 22 3e **47 72 fc df 20 47 6f 74 74** 3c 2f 68 31 3e
<h1 lang="zh-CN">**你好**</h1>	UTF-8	3c 68 31 20 6c 61 6e 67 3d 22 7a 68 2d 43 4e 22 3e **e4 bd a0 e5 a5 bd** 3c 2f 68 31 3e
	GB18030	3c 68 31 20 6c 61 6e 67 3d 22 7a 68 2d 43 4e 22 3e **c4 e3 ba c3** 3c 2f 68 31 3e
	Big5	3c 68 31 20 6c 61 6e 67 3d 22 7a 68 2d 43 4e 22 3e **a7 41 a6 6e** 3c 2f 68 31 3e

For example, in Table 1, each HTML snippet is encoded in different encodings, but the standard ASCII characters of the snippets are encoded with the same code points and code units, regardless of the encoding being used. The first ten bytes in all charset encodings, i.e. **3c 68 31 20 6c 61 6e 67 3d 22** which corresponds to the substring **<h1 lang="**, are equally encoded in every character encoding. This is why the structure of a web page do not become thoroughly corrupted when the browser detects a wrong charset encoding. Note that there are some exceptions for some charset encodings, such as UTF-16 and UTF-32 in which though ASCII characters have the same code points; ASCII characters have different code units due to the different CEFs of these encodings.

From the statistical point of view, the statistical properties of characters in an HTML documents is a linear combination of the encoding of the contents and the encoding of the markups. Since the markups are usually encoded in US-ASCII, the two distributions are essentially different when the content has an encoding different from US-ASCII. The markups disturb the statistical charset detectors because they bias all HTML document towards the distribution of US-ASCII. For example, if a HTML documents have a small non-English UTF-8 content with huge lines of HTML codes, a charset detector will classify the documents as ASCII, since it is statistically closest to ASCII. In short, when a large portion of document is markups, it is hard to detect the encoding of the document. Moreover, with the increase of Internet bandwidths and the processing speed of computers, the portion of structural data like scripts, styles, menus, navigational links, etc. in HTML web pages is increasing. Therefore, the large amount of structural data can potentially affect the precision of charset detection tools.

Our core idea is to remove the markups from the HTML document, so that only the content is considered for character detection. Markup elimination is not trivial, because in order to remove the markups, one should know in advance the charset of the page. In this regard, we propose two methods for eliminating the markups without knowing the encoding of the page.

4.1 Direct Markup Elimination

As mentioned earlier, almost all charset encoding schemes use the same code points and the same code units for the traditional US-ASCII characters. Merely knowing the ASCII code of the characters and keywords being used in HTML markups, we can identify and remove the byte sequences corresponding to the markups. However, this is practically a challenge, because there are many malformed HTML documents in the web, e.g. pages with broken tags, odd structures, etc. For example, there are pages with more than one body elements. Even though this method essentially works, it is a highly error-prone method because the program may face many unseen conditions in HTML documents.

4.2 ISO-8859-1 Decoding-Encoding Markup Elimination

Among the existing charset encoding schemes, ISO-8859-1 has an interesting feature. Having a byte sequence of a document with an arbitrary encoding, if we decode the byte sequence using ISO-8859-1 and then re-encode the content with ISO-8859-1, we exactly get the same byte sequence. In other words, ISO-8859-1 preserves code units of any charset encoding in the decoding-encoding process, while others do not necessarily do so. We may leverage this feature in order to remove the markups from the document without caring about the encoding of the page. It can be done in three steps. In step 1, the document is decoded using ISO-8859-1. Then, in step 2 the HTML markups are removed using a HTML parser. The out-of-the-box benefit of using a custom HTML parser is that it can repair malformed HTML documents, so we feel free about exceptions that potentially could be caused by these documents in HTML markup elimination process. Having the markup-eliminated document, we re-encode it using ISO-8859-1 in step 3. Then, a regular charset encoding detector can be used to identify the charset of the resulting contents. Table 2 illustrates markup elimination from four HTML snippet using ISO-8859-1 decoding-encoding method.

Table 2. ISO-8859-1 decoding-encoding markup elimination

HTML Elements	**Charset Encoding Details & Steps of Markup Elimination Using ISO-8859-1**				
	Charset Encoding details		***Step1***	***Step2***	***Step3***
	Charset Encoding	***Sequence of bytes (hex. Rep.)***	***Decoded text using ISO-8859-1***	***Extracted content using a html parser***	***Encoded content using ISO-8859-1***
<h1>**Hello world**</h1>	UTF-8	3c 68 31 3e **48 65 6c 6c 6f 20 77 6f 72 6c 64** 3c 2f 68 31 3e	<h1>Hello world</h1>	Hello world	**48 65 6c 6c 6f 20 77 6f 72 6c 64**
	ISO-8859-1	3c 68 31 3e **48 65 6c 6c 6f 20 77 6f 72 6c 64** 3c 2f 68 31 3e	<h1>Hello world</h1>	Hello world	**48 65 6c 6c 6f 20 77 6f 72 6c 64**
<h1> **سلام عليكم**</h1>	Windows-1256	3c 68 31 3e **d3 e1 c7 e3 20 da e1 ed df e3** 3c 2f 68 31 3e	<h1>ÓáÇã Úáíßã</h1>	ÓáÇã Úáíßã	**d3 e1 c7 e3 20 da e1 ed df e3**
	MacArabic	3c 68 31 3e **d3 e4 c7 e5 20 d9 e4 ea e3 e5** 3c 2f 68 31 3e	<h1>ÓäÇå Ùäêãå</h1>	ÓäÇå Ùäêãå	**d3 e4 c7 e5 20 d9 e4 ea e3 e5**
<h1>**Grüß Gott**</h1>	UTF-8	3c 68 31 3e **47 72 c3 bc c3 9f 20 47 6f 74 74** 3c 2f 68 31 3e	<h1>GrÃ¼ÃŸ Gott</h1>	GrÃ¼ÃŸ Gott	**47 72 c3 bc c3 9f 20 47 6f 74 74**
	Windows-1252	3c 68 31 3e **47 72 fc df 20 47 6f 74 74** 3c 2f 68 31 3e	<h1>Grüß Gott</h1>	Grüß Gott	**47 72 fc df 20 47 6f 74 74**
<h1>**你好**</h1>	GB18030	3c 68 31 3e **c4 e3 ba c3** 3c 2f 68 31 3e	<h1>ÄãºÃ</h1>	ÄãºÃ	**c4 e3 ba c3**
	Big5	3c 68 31 3e **a7 41 a6 6e** 3c 2f 68 31 3e	<h1>§A¦n</h1>	§A¦n	**a7 41 a6 6e**

Note that, by using ISO-8859-1 to decode a byte sequence we might get gibberish or mojibake text in many cases; however we still have a valid HTML string in which we could look for a potential charset encoding in its Meta tags. This can be considered as an option just to use in practical environment and we did not use Meta tag information in the evaluations presented in Section 5.

Approximately 80% of the websites use UTF-8 as their charset encoding [10], although this ratio varies greatly across regions, from the low ratio in East Asian countries to a very high ratio, say near the 100%, in countries like Iran. Anyway, due to the high usage rate of UTF-8, we should be as accurate as possible about UTF-8 pages. In this regard, our primitive tests show that Mozilla CharDet works very accurate for UTF-8 documents, but IBM ICU shows a better performance when dealing with other formats. Based on primitive tests, to maximize the precision we combine these tools in a particular way as shown in Fig. 1.

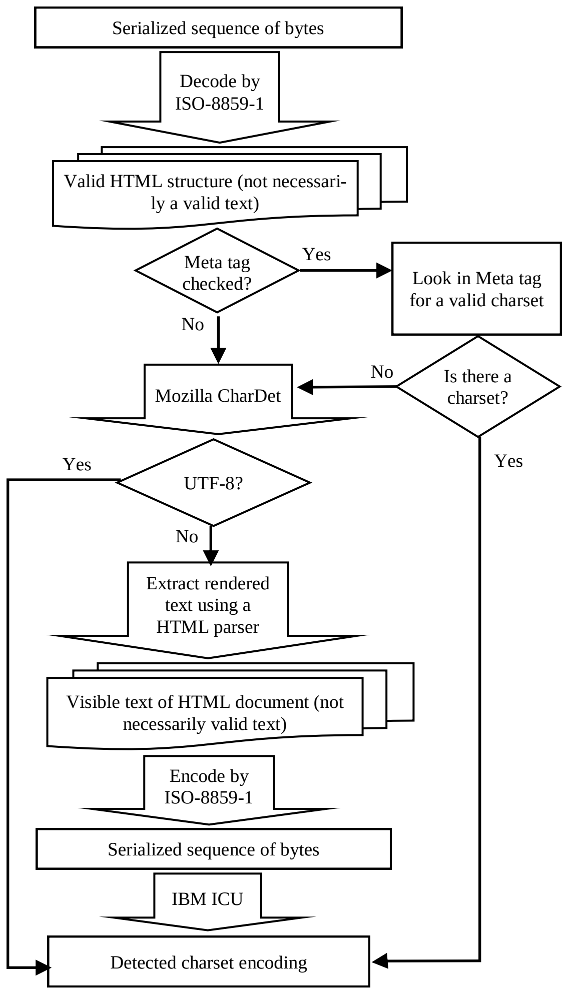
Serialized sequence of bytes
Decode by ISO-8859-1
Valid HTML structure (not necessari-ly a valid text)
Meta tag checked?
Yes
Look in Meta tag for a valid charset
No
Mozilla CharDet
No
Is there a charset?
Yes
Yes
UTF-8?
No
Extract rendered text using a HTML parser
Visible text of HTML document (not necessarily valid text)
Encode by ISO-8859-1
Serialized sequence of bytes
IBM ICU
Detected charset encoding

Fig. 1. Flowchart of the proposed hybrid mechanism

5 Evaluations

We evaluated our hybrid mechanism via two test scenarios: encoding-wise and language-wise. In the former test scenario we used a corpus of HTML documents with various encodings and in the latter test scenario we used a collection of URLs pointing to pages with different languages. Also of note, we tried to select these test pages and URLs from a real-word sample, and not from specific domains like Wikipedia.

Typically, special websites and domains have uniform structure and special features which can potentially affect the credibility of evaluations. The source code and test data we used for our experiments is freely available[1].

5.1 Encoding-Wise Comparison

In this test scenario we test Mozilla CharDet, IBM ICU and the hybrid mechanism against a corpus of HTML Documents. Unfortunately, there is no evaluation dataset for charset encoding detection. It is because, as we mentioned in Section 2, there are few works in this area and each of the existing works used its own specific benchmark for evaluation. Hence, we collected a corpus of nearly 2700 HTML pages with various charset encoding types.

To create the corpus, we wrote a multi-threaded crawler and used a simple version of Tunneling [12] to crawl the Web. Tunneling enables us to gather more diverse web pages, consequently the more diverse web pages to be in the corpus, the more reliable results will be produced. For the sake of diversity, we used various seeds from different TLDs (Top Level Domains). By the way, the main criteria for adding a page to the corpus is the information provided in the charset field of the HTTP header. The evaluation results on our corpus is presented in Table 3.

Table 3. Histogram of true vs. detected charsets using IBM ICU, Mozilla CharDet and Hybrid

Encoding-Wise Comparison	**True Encodings**														
	UTF-8			***Windows-1251***			***GBK***			***Windows-1256***			***Shift_JIS***		
Detected Encodings	***ICU***	***Ch.D.***	***Hyb.***	***ICU***	***Ch.D.***	***Hyb.***	***ICU***	***Ch.D.***	***Hyb.***	***ICU***	***Ch.D.***	***Hyb.***	***ICU***	***Ch.D.***	***Hyb.***
UTF-8	549	**657**	**657**					3	3	5	14	14			
Windows-1251						**313**									
GB18030							**414**	145	407		1			636	
Windows-1256										26		**616**			
Shift_JIS					10						227		**639**	3	636
ISO-8859-1	108			311			5			573		6			
Windows-1252				3		1				31	28	8			2
Big5								73	4						
GB2312								197							
UTF-16LE											4			1	
UTF-16BE								1			28				
Other Encodings									5	10		1	1		2
No Match					304						343				
Accuracy	%84	%**100**	%**100**	%0	%0	%**100**	%**99**	%35	%97	%4	%0	%**96**	%**100**	%1	%100

Results show that the mean average precision of the hybrid mechanism in detecting charset encoding of pages that are encoded by Windows-1251, Windows1256 and UTF-8 is improved. The average overall accuracy of IBM ICU, Mozilla CharDet, and our hybrid mechanism are 61%, 30%, and 99%, respectively.

[1] https://github.com/shabanali-faghani/IUST-HTMLCharDet

Note that for a fair judgment on the accuracy of a charset detection tool in a multi-encoding environment, like our case in Table 3, one should consider all charset encodings together. For example, while the accuracy of Mozilla CharDet for UTF-8 is 100%, it performs poorly for other charset encodings. On the other hand, if IBM ICU is used instead, it works very accurate for East Asian languages, which use charsets like GBK, Shift_JIS and also EUC-KR, but it is very inaccurate for Cyrillic-based and Arabic languages which extensively use Windows-1251 and Windows-1256 charsets respectively. All this, while the accuracy of the hybrid mechanism in detecting Windows-1251, Windows1256 and UTF-8 is near 100%, its accuracy in detecting GBK and Shift_JIS do not meaningfully drop from the accuracy of IMB ICU on raw HTML pages.

5.2 Language-Wise Comparison

In the language-wise test scenario, we compare our hybrid mechanism with the two other charset detector tools from language point of view. We collected a list of URLs that are pointing to various web pages with different languages. The URLs are selected from the top one million websites visited from all over the world, as reported by Alexa (www.alexa.com). In order to collect HTML documents in a specific language, we investigated webpages with the internet domain name of that language. For example, Japanese web pages are collected from .jp domains. The results of evaluation for eight different languages are shown in details in Table 4.

Table 4. Accuracy of three charset encoding detectors in monolingual and multilingual environments

Language-Wise Comparison	**Details of Fetched Web Pages & Accuracy of Charset Detector Tools**					
Language(s)	***TLD(s)***	***# primitive URLs***	***# sites with valid charset in HTTP header***	***IBM ICU***	***Mozilla CharDet***	***Hybrid***
English	.us, .uk	21565	13223	9895 = %75	12373 = %94	**12586 = %95**
Russian	.ru	39453	28257	21716 = %77	21840 = %77	**27900 = %99**
Japanese	.jp	20339	6833	6461 = %95	6051 = %86	**6741 = %99**
Arabic	.iq, 11 more	1904	1093	888 = %81	1015 = %93	**1064 = %97**
Chinese	.cn	10086	3776	3596 = %95	3449 = %91	**3640 = %96**
German	.de	35318	23225	20115 = %87	20137 = %87	**21371 = %92**
Persian	.ir	7396	4018	3896 = %97	3971 = %99	**4003 = %100**
Indian, English	.in	12236	4867	3203 = %66	4640 = %95	**4677 = %96**
Total (multilingual)	all above	148297	85292	69770 = %82	73476 = %86	**81982 = %96**

In our test set, the English pages are selected from `.uk` and `.us` domains. Similarly, Arabic pages are selected from 12 different country-specific TLDs. Note that, the `.in` domain is quite different from the others, because the pages in this domain are not purely in Indian, but also include contents in English and some pages have both Indian and English contents. As illustrated in Table 4, for all languages there is a significant difference between the number of primitive URLs and the number of sites with valid charset in HTTP header. This difference is mainly due to that the HTTP servers do not provide clients with the information about charset encodings for approximately half of all requested web pages, and sometimes a result of networking problems.

As illustrated in Table 4, the accuracy of our hybrid mechanism is significantly higher than the two other tools. Besides, unlike IBM ICU and Mozilla CharDet, the accuracy of this mechanism is not volatile in different languages and regions. Also, it should be noted that the high accuracy of Mozilla CharDet over the IBM ICU in this test is due to the high usage of UTF-8 in various languages and regions. In absence or low usage of UTF-8, Mozilla CharDet is not likely to outperform IBM ICU.

In addition to UTF-8 there are three charset encodings GB2312, GBK and GB18030 which are extensively used in China. GBK is backward compatible with GB2312, because it was defined as an extension of GB2312 in 1993. Later on, GB18030 became compatible with GBK in 2000. Hence, there is forward compatibility between both GB2312 and GBK with GB18030. It seems that neither IBM ICU nor Mozilla CharDet has GBK in its charset list, but this charset is frequently appearing in HTTP header of the Chinese web pages. Therefore, to have a fair comparison we considered the actual direction of compatibility among these charsets as accurate detection in this evaluation.

Except for Chinese language, equality between the charset in HTTP header and the detected charset by a tool was the accuracy measure in this test. However as we know there is a pretty cool difference between **equality** and **equity**. Equity for two charsets on a web page can be defined as the validity of both of them for decoding that page, i.e. if we use either charset for decoding that web page the generated html documents are quite equal. Unfortunately the validity check for a charset on a web page could not be done automatically, because often, a charset validator does not complain even if a wrong encoding is detected or selected. Hence, the validator cannot decide whether the decoded text makes sense or not [11]. The reason behind this is that there is a close similarity between many of charset encoding types. Windows-1252, for example, is an extension on ISO-8859-1. On the other hand there is quite some overlap between ISO-8859-1 (Western Europe) and ISO-8859-2 (Eastern Europe) and other charset encodings in this series. Since typically the difference of similar charsets falls on unused code points and obsolete characters, as about the Windows-1252 and ISO-8859-1, in many cases either charset is valid to decoding a text that is encoded by one of them. Altogether, in addition to the charsets in HTTP headers and considering the right direction of compatibility between charsets, only visual inspection can make sure that whether a detected charset is valid or not. However, visual inspection is impossible for large collections like our case in Table 4.

6 Conclusions and Future Work

In this paper, we propose a hybrid mechanism for charset encoding detection. The proposed method heavily relies on the fact that removing structural data from HTML documents can significantly increase the precision of the charset detector tools. In addition to high accuracy, our method is able to cope with normal HTML web pages, as well as the noisy and malformed ones, in multi-encoding and multilingual environments. The proposed method is compared against two test scenario with real-world

HTML. Results show a significant improvement over the two most famous charset encoding detection tools, namely IBM ICU and Mozilla CharDet.

Charset detection is not a foolproof process, because it is essentially based on statistical data and what actually happens is guessing not detecting. Both IBM ICU and Mozilla CharDet provide the probability of accuracy along with the guessed charset. Using these probabilities in a special way seems to be useful to yet increasing precision of the hybrid approach. Also, since the more critical mission web applications brings with it the need to more reliability and precision, developing special-purpose charset validator may assure these applications. In this connection, inspecting special characters in decoded text, like the replacement character (U+FFFD: �) in Unicode, could help the validator.

Acknowledgements. The authors would like to thank Hamed Kordestanchi, who proposed the language-wise evaluation, and Mojtaba Akbarzade for his guidance during this work.

References

1. Russell, G., Lapalme, G., Plamondon, P.: Automatic identification of language and encoding. In: Rapport Scientifique, Laboratoire de Recherche Appliquée en Linguistique Informatique (RALI), Université de Montréal, Canada (2003)
2. Kim, S., Park, J.: Automatic Detection of Character Encoding and Language, Technical Report, Machine Learning, Stanford University (2007)
3. Tang, F.Y.F.: Mozilla Charset Detectors, Mozilla (2008). http://www-archive.mozilla.org/projects/intl/chardet.html
4. ICU - International Components for Unicode, IBM (2014). http://site.icu-project.org
5. Kikui, G.I.: Identifying, the coding system and language, of on-line documents on the internet. In: Proceedings of the 16th Conference on Computational linguistics, vol. 2, pp. 652–657 (1996)
6. Charset detection, Wikipedia (2014). http://en.wikipedia.org/wiki/Charset_detection
7. Character Data Representation Architecture, IBM (2013). http://www.ibm.com/software/globalization/cdra/index.jsp
8. Whistler, K., Davis, M., Freytag, A.: Unicode Technical Report# 17, Character Encoding Model, The Unicode Consortium (2008). http://www.unicode.org/reports/tr17
9. Dürst, M.J., Yergeau, F., Ishida, R., Wolf, M., Texin, T.: Character Model for the World Wide Web 1.0: Fundamentals, W3C (2005). http://www.w3.org/TR/charmod
10. Historical trends in the usage of character encodings for websites, W3Techs (2015). http://w3techs.com/technologies/history_overview/character_encoding
11. Dürst, M.: Checking the character encoding using the validator, W3C (2003). http://www.w3.org/International/questions/qa-validator-charset-check.en
12. Bergmark, D., Lagoze, C., Sbityakov, A.: Focused crawls, tunneling, and digital libraries. In: Agosti, M., Thanos, C. (eds.) ECDL 2002. LNCS, vol. 2458, pp. 91–106. Springer, Heidelberg (2002)