Chapter 3
Automatic Extraction of Structured Data from a Single Webpage

3.1. Introduction
As much of the effort is being put on developing a Fully-automatic extraction method [Bohannon et al., 2012; Dalvi et al., 2011; Elmeleegy et al., 2011]. So, in this section we will discuss two categories of automatic data extraction approaches viz. Pattern search based methods and Visual signal aided methods. On pattern search based methods we try to identify the repeating tag tree patterns of data records. The two main techniques used in this method are String matching and Tree matching. One of the most popular String Matching Technique is string edit-distance also known as the Levenshtein distance [Liu, 2012]. String edit-distance and its other variations are used in MDR [Liu et al., 2003], ViPER [Simon, 2005] and ViNTs [Zhao et al., 2005]. String edit-distance between two strings is defined as the number of characters that need to be changed to convert one string into another. The operations include insert a character, change a character and delete a character. According to [Liu, 2012], the normalized edit-distance (ND) between two strings \( s_1 \) and \( s_2 \) is defined as edit-distance (d) divided by the length of the longest string.

\[
ND(s_1, s_2) = \frac{d(s_1, s_2)}{\max(|s_1|, |s_2|)}
\]  

Tree matching, the other technique for finding repetitive patterns in a source code is defined as the minimum set of operations required to transform one
ordered rooted tree to another tree. The operations include tree node insertion, tree node replacement and tree node removal. DEPTA [Zhai and Liu, 2006], NET [Liu, 2005] and G-STM [Jindal and Bing, 2012] were the first to use this technique. The normalized simple tree matching (NSTM) between two trees is defined as Simple Tree Matching (STM) divided by mean number of nodes in the two trees. [Liu, 2012]

\[
NSTM(t_1, t_2) = \frac{STM(t_1, t_2)}{\max(\text{nodes}(t_1), \text{nodes}(t_2))}
\] (3.2)

Visual signal aided methods use visual features of a webpage to detect the data records in a webpage. VIDE [Liu et al., 2010] and VIPS [Cai et al., 2003] fall in this category. Few methods depend on both HTML features and visual features to identify the main region containing the data records as in ViNTs [Zhao et al., 2005]. Figure 3.1 shows a typical visual block and its visual signal.

![Data Block and Visual Signal](image)

**Figure 3.1:** A Data Block and its Visual Signal

In this chapter, we propose a method for automatic extraction of structured data from a webpage known as Extraction of Structured Data (ESD). When a structured webpage is requested from the web server, web server in turn queries for the information from the underlying database and then returns the information to the browser. Many researchers propose different solutions to extract structured data from the Web. Pattern based methods compute the
similarity between tag trees to find the repetitive patterns in a webpage but such solution result in errors while dealing with Web 2.0 pages. It is because of the fact that, for an incorrect HTML code, web browsers have very high tolerance and thus many web pages do not follow World Wide Web Consortium (W3C) HTML specifications. HTML code should be correct for constructing correct HTML tag tree which in turn do not hamper the extraction process. Extracting structured data using only HTML code is thus a very tedious and difficult task. Some other techniques are available that uses the visual cues of a webpage for extracting data. Thus our method makes use of both visual and structural features of a web page to extract data records from a webpage. Our ESD depends on two basic observations.

- The structured data records are retrieved from some database and presented on web pages using some predetermined template. Figure 3.2 shows three data records showing books in an online store. These data records are structurally similar and are located in one region of a webpage known as data region.

- Website developers present their data in such a way that humans easily and quickly figure out and distinguish each data record.

Figure 3.2: Snapshot of an Amazon Store containing three Data Record
From these above two observations, it is clear that every data record of a webpage has some sort of resemblance.

3.2. The Proposed Method

The entire flow of ESD method is depicted in figure 3.3. If given a webpage, the ESD method works in the following phases:

- In the pre-processing phase the webpage is loaded in a browser for further processing.
- In segmentation phase the webpage is segmented into visually and semantically similar blocks and then the Hierarchical block tree is built.
- In data records identification phase block similarity is calculated using the normalized edit-distance algorithm.
- In data items extraction phase data items are extracted and displayed and are stored in a database.

![Figure 3.3: Block Diagram of ESD Method](image)
3.2.1. Web page Rendering and Pre-processing

Web Page Renderer accepts the URL of a webpage, fetches the webpage and is then displayed in a web browser. Thus the outcome of a whole rendering process is a webpage that we see in a browser. Prior to the extraction process, i.e. after the rendering phase the web page is modified to enhance and accelerate the extraction process. The modification consists of cleaning bad and ill formatted tags. This is done by using a utility known as HTML Tidy [Ragget et al., 1998]. Tidy detects and corrects the mismatched tags, corrects the tags which are in the wrong order, and adds missing slash in the end tags etc. For example consider the following HTML code segment:

```html
<html>
<body>
<h1>heading</h1>
   <h2>subheading</h2>
   <p>here is a para<b>bold</b> <i>bold italic</i> bold? normal?</p>
   <a href="http://.....">Books</a>
</body>
</html>
```

is mapped to

```html
<html>
<body>
   <h1>heading</h1>
   <h2>subheading</h2>
   <p>here is a para<b>bold</b> <i>bold italic</i> bold?</p>
   <a href="#http://.....">Books</a>
</body>
</html>
```
3.2.2. Segmentation of a Webpage

A webpage \( W \) can be represented as a triple set \( W = (B, S, R) \), where \( B = \{B_1, B_2, B_3... B_n\} \), a finite set of non-overlapped blocks, \( S = \{S_1, S_2, S_3... S_n\} \), a finite set of separators, each carrying its weight (both horizontal and vertical separators), and \( R \) is the relationship between every two blocks \( (B_i, B_j) \) in \( B \) and can be written as \( R = B_i \times B_j \rightarrow S \cup \{NULL\} \) as shown in figure 3.4.

\[
B = \{VB_1, VB_2\}
S = \{S_1\}
R = [(VB_1, VB_2)] = [S_1]
\]
and

\[
VB_2 = \{VB_2 - 1, VB_2 - 2, VB_2 - 3\}
S_2 = \{S_{12}, S_{22}\}
R_2 = [VB_2 - 1, VB_2 - 2] = [S_{12}], [VB_2 - 2, VB_2 - 3] = [S_{22}]
\]

Figure 3.4 (a): Different Blocks of a Webpage

Figure 3.4 (b): Corresponding Specification of Vision-Based Content Structure for an Amazon Webpage

Figure 3.4 (a) shows the layout structure and vision-based content structure for an Amazon webpage. At the first level, the webpage is divided into two blocks VB1 and VB2 with one separator S1. Then the visual block VB2 is
further divided into three blocks VB2-1, VB2-2 and VB2-3 with two separators S12 and S22 and the process is repeated as shown in figure 3.4 (b). In this phase, a webpage is divided into several smaller blocks which are visually and semantically similar. For this purpose segmentation algorithm is proposed. The most popular webpage segmentation methods are: Heuristic-based, DOM-based, Location based and Vision-based, each having their own advantages and disadvantages. Our segmentation method is an enhanced version of VIPS [Cai et al., 2003] known as eVIPS. The eVIPS first creates a Document Object Model (DOM) tree of a webpage. The Document Object Model is a very powerful and very flexible way of representing a document. DOM is platform-independent, browser and language neutral way of discovering, manipulating and changing the content of a document using programmatic techniques. The objects in the DOM are generically known as “nodes” and these nodes have a parent-child relationship. A DOM tree of a whole webpage is divided into several blocks starting from the root node. Every node is then analysed whether a smaller block is formed or not. These blocks are put in a pool. Separators between these blocks are determined and labelled with a weight. Finally a layout hierarchy for this round is constructed and each leaf node is monitored to meet the certain level of granularity. If that level of granularity is not met, then the leaf node is segmented again and the same process is repeated. Also similar data records are presented in one region known as a main data region. In eVIPS the visual block is segmented based on the following heuristic rules given in table 3.1.
### Table 3.1: Seven Heuristic Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule I.</td>
<td>If the DOM node has only one text node or invalid children, then the DOM node cannot be divided.</td>
</tr>
<tr>
<td>Rule II.</td>
<td>If the DOM node has only one valid child node and that child node is text node or virtual text node, then the DOM node cannot be divided.</td>
</tr>
<tr>
<td>Rule III.</td>
<td>If one of the child nodes of the DOM node has <code>&lt;HR&gt;</code> or <code>&lt;BR&gt;</code> html tag, then divide the DOM node.</td>
</tr>
<tr>
<td>Rule IV.</td>
<td>If the DOM node has some partially valid nodes and some valid nodes, then divide the DOM node.</td>
</tr>
<tr>
<td>Rule V.</td>
<td>If the DOM node has a line break child containing an image, divide the DOM node into two blocks. The first block will contain the nodes before the child along with the image and the second block will contain the remaining child.</td>
</tr>
<tr>
<td>Rule VI.</td>
<td>If the DOM node is a table and some of its columns have a different background color than others. Divide the node and construct a separate block for each column. Also the columns with the different bg color will not be divided in this round.</td>
</tr>
<tr>
<td>Rule VII.</td>
<td>If one of the child of the DOM node has greater size than its remaining children, divide the DOM node. Put the children before the bigger font size into one block and the remaining in the next block.</td>
</tr>
</tbody>
</table>

After the segmentation is done, Hierarchical block tree of a webpage is build to extract the data records. Each Segmented block of a webpage can be Text block (T), Text Link block (TL), Image block (I), Image Link Block (IL), Visual block (VB), drop down menu bock (DM), Textbox block (TB) and Action button block (AB). With the help of these nodes we create a block tree. By the end, each node has its type attached to it and if a node is of type VB then
only it is further subdivided into smaller blocks. Figure 3.5 shows the block tree of a visual block (VB2-2) of a web page and figures 3.6, 3.7, 3.8 and 3.9 shows the four main procedures of eVIPS.

Figure 3.5: Block Tree of a Visual Block (VB2-2) of an Amazon Webpage
**Algorithm: Main Module of eVIPS**

**Input:** root node  
**Output:** HCS
1. **Begin**  
2. eVIPS(root node)  
3. Pool ← Xtractblocks(root node)  
4. S ← DetectS(Pool)  
5. HCS ← Construct_CS(Pool, S)  
6. return HCS  
7. **End**

**Figure 3.6:** Main module of eVIPS

**Procedure: Set of Raw Blocks**

**Input:** root node of DOM tree  
**Output:** Pool
1. **Begin**  
2. Xtractblocks (root node)  
3. if (root node is dividable) then
4. for each child of root node do
5. Pool ← Pool + Xtractblocks (child)
6. endfor
7. else
8. Block ← Subtree (root node);
9. Pool ← Pool + Block
10. endif
11. return Pool  
12. **End**

**Figure 3.7:** Get Blocks Procedure
**Procedure: List of separators**

Input: Pool
Output: $S$

1. **Begin**
2. Detect $S$(Pool);
3.  
   for each Block $\in$ Pool do
4.      Split or update or remove $S_i$
5. endfor
6.  
   for each $S_i$ in $S$ do
7.      weight $\leftarrow S_i$
8. endfor
9. return $S$
10. **End**

**Figure 3.8:** Get Separators Procedure

**Procedure: Hierarchical Vision-Based Content Structure**

Input: Pool, $S$
Output: HCS

1. **Begin**
2. Construct_CS (Pool, $S$)
3. $S \leftarrow$Sort($S$); // sorting done on weight of Separators
4.  
   for each $S_i \in S$ do
5.      Getblocks $\leftarrow$($S_i$, Neighboring blocks, Pool)
6.      Mblock $\leftarrow$ merge (Getblocks)
7.      Set DoC $\leftarrow$Mblock
8.      Construct_H(Mblock,, HCS)
9. endfor
10.  
    for each leaf $\in$Structure.leafs do
11.     if leaf. DoC$\leq$ PDoC then
12.        new_CS $\leftarrow$VBPS(leaf, root node)
13.        Replace((leaf, new_CS), HCS)
14.     endif
15. endfor
16. return HCS
17. **End**

**Figure 3.9:** Hierarchical Vision-Based Content Structure Procedure
3.2.3. Identification of Data Records

In order to identify data records, we have to compute the similarity between the blocks. Block similarity can be calculated using string similarity metric (a metric that measures the distance between the two strings for approximate string matching). There are various string similarity metrics available in [Bilenko et al., 2003]. These metrics can be broadly divided into two categories viz. character-based metrics and token-based metrics. Some of the prominent string similarity metrics which are used in information extraction systems are:

I). Jaro-Wrinkler Distance:

Jaro-Wrinkler metric [Wrinkler, 1999] is a character-based metric and is an extension of Jaro distance metric [Jaro 1989; Jaro 1995]. For two strings s and t, the Jaro metric is given in equation 3.1

\[
\text{Jaro}(s, t) = \frac{1}{3} + \left( \frac{|s|}{|s|} + \frac{|t|}{|t|} + \frac{|s| - T_{s,t}}{2|s|} \right)
\]  

(3.3)

Where \( s' \) is the number of matching characters in \( t \) and \( t' \) is the number of matching characters in \( s \) and \( T_{s,t} \) is half the number of transpositions for \( s' \) and \( t' \) [Bilenko et al., 2003]. The Jaro-Wrinkler is defined as in equation 3.2

\[
\text{Jaro-Wrinkler}(s, t) = \text{Jaro}(s, t) + \left( l p \left( 1 - \text{Jaro}(s, t) \right) \right)
\]  

(3.4)

where \( l \) is the length of the common prefix of \( s \) and \( t \), and is taken as 4 and \( p \) is the constant scaling factor and taken as 0.1.

II). Jaccard Similarity:

Jaccard similarity is an effective token-based string similarity metric [Bilenko et al., 2003]. The Jaccard similarity between two strings \( s \) and \( t \) is defined as the size of the intersection divided by the size of the union of \( s \) and \( t \) as shown below in equation 3.3.

\[
\text{Jaccard Similarity}(s, t) = \frac{|s \cap t|}{|s \cup t|} = \frac{|s \cap t|}{|s| + |t| - |s \cap t|}
\]  

(3.5)
III. Cosine Similarity:
Cosine similarity is mainly defined between two same dimensional vectors. And it is the ratio of the scalar product of the vectors to the multiplication of their norms. The cosine of 0° is 1, and it is less than 1 for any other angle. It is thus a judgement of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. Given two vectors of attributes, x and y, the cosine similarity, \( \cos(\theta) \), is represented using a dot product and magnitude as:

\[
\text{Cosine\_Similarity}(x, y) = \cos(\theta) = \frac{xy}{||x|| ||y||}
\]  

(3.6)

In order to calculate the block similarity, we will use the Levenshtein edit-distance algorithm. Edit-distance is an algorithm to calculate the similarity between two strings (sequences). As shown in Figure 3.5, the string representation of block VB2-2-1 is: \( IL - T - T - T - TL - TL - TL \). The basic idea is to calculate the minimum number of single character edits required to change one string into other with the help of insertion, deletion and substitution operations. So this string metric is used for measuring the difference between two sequences. The bigger the return value is, the less similar the two strings are because different strings take more edits than similar strings. The equation 3.7 is used to calculate the edit-distance between two blocks say \( S_i \) and \( S_j \)

\[
\text{Ed}(s_i, s_j) = \min \{ \text{Ed}(s_i, s_{i-1}) + 1, \text{Ed}(s_{i-1}, s_j) + 1, \text{Ed}(s_{i-1}, s_{j-1}) + s \}
\]

where \( s = 1 \) if \( s_i \neq s_j \) and \( s = 0 \) if \( s_i = s_j \)

(3.7)

In our method we will use normalized edit-distance [Liu et al., 2000] because equation 3.7 is not appropriate for our problem. The normalized string edit-distance is expressed in equation 3.8.
Chapter 3: Automatic Extraction of Structured Data from a Single Webpage

\[ \text{Ned} = 1 - \frac{\text{Ed}(s_i, s_j)}{\text{Max}(|s_i|, |s_j|)} \]  

We calculate the block similarity using the above normalized edit-distance [Liu, 2000]. Also the similarity threshold is set to 0.5. That means, if \( \text{Ned} \) (Normalized edit-distance) of two blocks of a webpage is greater than 0.5 they are considered to be similar blocks and are treated as data records. Otherwise, if the threshold is smaller than 0.5 the blocks are dissimilar and are not treated as data records.

3.2.4. Extraction of Data Items

The main aim of ESD is to extract all data records from a given webpage. The method must extract all data records of a main content region and no data item should be missed or incorrectly extracted. In a webpage it has been observed that the data records are structurally similar and are present in one region of a webpage known as data region. Data records are usually contained in the same internal node and thus are the siblings in a block tree. There are few situations where the similar data records are not at the same level. So, we apply some transformations to extract such data records. As shown in figure 3.10 (a) three similar data records VB2-1-1-1, VB2-1-1-2 and VB2-1-1 are at different levels. So, we first compare VB2-1-1-1, and VB2-1-1-2 and mark them as data records. These data records are compressed to VB2-1-1-1(2) as shown in figure 3.10 (b). where number 2 in parenthesis represents the number of occurrences of that block and then we finally perform collapse operation to VB2-1-2 as shown in figure 3.10 (c). In another situation three data records are similar as shown in figure 3.11 (a) VB2-1-1, VB2-1-2-1 and VB2-1-2-2. When we reach VB22-1-1, there is no similar node and hence it is not marked as a data record. But then when VB2-1-2-1 and VB2-1-2-2 are compared and they are marked as data records we need to go back to check if VB2-1-1 is a data record. So we again compress VB2-1-2-1 and VB2-1-2-2 to VB2-1-2-1(2) as shown in figure 3.11 (b) and finally perform collapse as
shown in figure 3.11 (c). The overall algorithm for data record extraction is shown in figures 3.12, 3.13 and 3.14.

Figure 3.10: Three Data Records at Different Levels

Figure 3.11: Three Data Records at Different Levels
**Algorithm: Data Record Extraction**

Input: node(p), t=0.5  
Output: list of data records

1. Begin
2. if(node(p) != NULL)
3.   if(node(p) has next sibling )
4.     Set node(q) ← next sibling of node (p)
5.     if(node(q) != leaf node )&& (Ned (p,q) >= t))
6.       if(node(p)has at least two children and no leaf child)
7.         for all children of node (p) and node (q) do
8.            Neighbor Comparison
9.         endfor
10.        if(all neighbor comparison of node (p) are similar) &&(all neighbor comparison of node (q) are similar too)
11.           Mark all children of (p) & (q) as data records
12.           Set their NOC=1;
13.           Mark Their Ancestors;
14.           Add all children of node (p) and node (q) to the list, if they are not in list;
15.           Compress the child of node (p) ← last child of node (p);
16.           Compress the child of node (q) ← last child of node (q);
17.           Call Fun2();
18.        endif
19.        Set node (p) and node (q) as data records;
20.        Set their NOC=1;
21.        Mark their ancestors;
22.        Add them to the list, if they are not in list;
23.        Call Fun2();
24.      else
25.        Call Fun1();
26.      endif
27.   else
28.     Call Fun2();
29.   endif
30. endif
31. endif
32. End

**Figure 3.12:** Data Record Extraction Algorithm
Procedure: Fun1

1. Begin
2. if((node (p) has Grand Child) && ( node (p) != List))
3. Set node(p) ← First child node (p) and repeat
4. else
5. Set node(p) ← highest internal node of node (p) and repeat
6. endif
7. End

Figure 3.13: Fun1 procedure

Procedure: Fun2

1. Begin
2. Set node(q) ← Compress node (p) and node (q)
3. Set node(q)’s NOC ← NOC’s of node(p) + node(q)
4. if collapse needed
5. collapse and compare
6. endif
7. Set node ( q) ← node ( p)
8. End

Figure 3.12: Fun2 procedure

3.2.5. Storing Data Items

By knowing the block id of data records, we can easily iterate and gather the appropriate information. Extraction of data items is very straight forward as we just have to locate and extract all data records. Within each data record, we can extract all or few data items with the help of parsers (Regular expressions). Figure 3.15 shows the snapshot of how ESD extracts data from the webpage and Figure 3.16 shows the snapshot of data being stored.
Chapter 3: Automatic Extraction of Structured Data from a Single Webpage

Figure 3.15: Snapshot of Data being Extracted

Figure 3.16: Snapshot of Data being Stored
3.3. Experimental Setup and Evaluation

The proposed work is implemented using Windows XP and JDK 1.5. In this section, we experimentally verify our method. We extract several data items from each webpage. For example, we choose product name and price from an e-commerce website. The two performance measures that we used to evaluate our method are recall and precision. Precision measures how many data records that are retrieved are actually relevant, e.g., 65% precision rate means that 65% of the data records retrieved are relevant while as 35% have been misidentified as irrelevant while as Recall measures how much data records in a collection have been actually found e.g., 40% recall rate means that 40% of data records in a collection have been found and 60% have been missed as shown in equations 3.9 and 3.10.

\[
\text{Precision} = \frac{\text{total number of records correctly extracted}}{\text{total number of data records extracted}} \tag{3.9}
\]

\[
\text{Recall} = \frac{\text{total number of records correctly extracted}}{\text{total number of records in the web page}} \tag{3.10}
\]

The summary of the results is shown in table 3.2. The table clearly shows a variation in recall value among data sets while, as the precision of extraction is good for all web pages.
Table 3.2: Experimental Results of ESD Method

<table>
<thead>
<tr>
<th>Domain</th>
<th>Records in a Webpage</th>
<th>Records Extracted</th>
<th>Records Correctly Extracted</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.flipkart.com/mobables">www.flipkart.com/mobables</a></td>
<td>58</td>
<td>56</td>
<td>52</td>
<td>92.85</td>
<td>89.65</td>
</tr>
<tr>
<td><a href="http://www.ebay.in/sch/Laptops">www.ebay.in/sch/Laptops</a></td>
<td>50</td>
<td>43</td>
<td>41</td>
<td>95.34</td>
<td>82.00</td>
</tr>
<tr>
<td><a href="http://www.myntra.com/mensuits">www.myntra.com/mensuits</a></td>
<td>48</td>
<td>43</td>
<td>41</td>
<td>95.34</td>
<td>85.41</td>
</tr>
<tr>
<td><a href="http://www.homeshop18.com/gifts-flower">www.homeshop18.com/gifts-flower</a></td>
<td>24</td>
<td>22</td>
<td>20</td>
<td>90.90</td>
<td>83.33</td>
</tr>
<tr>
<td><a href="http://www.bigbuy.com.bd/">www.bigbuy.com.bd/</a></td>
<td>18</td>
<td>17</td>
<td>15</td>
<td>88.23</td>
<td>83.33</td>
</tr>
<tr>
<td><a href="http://www.homeshop18.com/jeans">www.homeshop18.com/jeans</a></td>
<td>16</td>
<td>13</td>
<td>12</td>
<td>92.30</td>
<td>75.00</td>
</tr>
<tr>
<td><a href="http://www.ebay.in/rts/hub/Fragrances">www.ebay.in/rts/hub/Fragrances</a></td>
<td>72</td>
<td>69</td>
<td>64</td>
<td>92.75</td>
<td>88.88</td>
</tr>
<tr>
<td><a href="http://www.snapdeal.com/products/stationery">www.snapdeal.com/products/stationery</a></td>
<td>24</td>
<td>21</td>
<td>20</td>
<td>95.23</td>
<td>83.33</td>
</tr>
<tr>
<td><a href="http://www.flipkart.com/bag">www.flipkart.com/bag</a></td>
<td>58</td>
<td>54</td>
<td>50</td>
<td>92.59</td>
<td>86.20</td>
</tr>
</tbody>
</table>

As can be seen from the table 3.2 the precision is good for all web pages. Hence, in future we can upgrade this value by enhancing our segmentation method.

3.4. Conclusion

In this chapter we propose an ESD method for web data extraction. Although the problem has been studied by other researchers before, but their works are more or less limited to real-life application for being based on many restrict assumptions. Thus, ESD method is more flexible and pragmatic. In ESD method, firstly, we use HTML Tidy to clean bad and ill
formatted tags of a webpage and then eVIPS is used to convert the HTML webpage into an XML file. XML file is then converted into a DOM tree using a DOMfromXML.java class and finally Levenshtein distance.java is used to calculate the edit-distance between the blocks and finally data records are extracted, displayed and stored using parsers. Various data extraction methods that are available works fine on a specific domain of test data sets. If they will be allowed to run on the other data set the result will turn out to be unsatisfactory. So, the main aim of the ESD method is to extract data from a structured webpage that would work satisfactorily on various data sets. Various algorithms are present in the literature that extracts data from a single webpage, but only few of them are practically implemented. Our proposed method automatically extracts data from a webpage. Every time a webpage is loaded, it generates the DOM tree for that particular webpage, segments the webpage using enhanced vision-based segmentation method, detects data region and finally extracts all the text data items. The ESD algorithm acts as a milestone for the main goal of our research work, i.e. to develop a Fully-automatic web data extraction method to extract data from multiple web pages. This method does not extract the images of data records from a webpage. So, in future work we plan to modify the ESD method to extract all data items of a data record.