Electronic Preprint for Journal of Information Processing Vol.25

Regular Paper

Event.Locky: System of Event-Data Extraction from Webpages based on Web Mining

Chenyi Liao[1,a)] Kei Hiroi[2,b)] Katsuhiko Kaji[3,c)] Ken Sakurada[1,d)] Nobuo Kawaguchi[1,2,e)]

Received: May 12, 2016, Accepted: January 10, 2017

Abstract: Nearby event data, such as those for exhibitions and sales promotions, may help users spend their free time more efficiently. However, most event data are hidden in millions of webpages, which is very time-consuming for a user to find such data. To address this issue, we use web mining that extracts event data from webpages. In this paper, we propose and discuss the implementation of Event.Locky – a system for extracting event data from webpages in a user-defined area and displaying them to a user in a spatial-temporal structure. Furthermore, we design two core algorithms for event data extraction in Event.Locky: webpage-data-record extraction and event-record classification. The former is used to convert a semi-structural HTML document into processable structured data. The latter filters out non-event data from extracted data records using machine learning. We trained and evaluated Event.Locky with an actual dataset composed by 96 restaurants and shops at Nagoya train station. As a result, our event-classification algorithm achieved an F_1 score of 91.61%, an increase of 3.07% from current event-classification algorithms. The combination of our event-classification algorithm and our data-record-extraction algorithm achieved F_1 score 83.96% to extract event records from webpages. That increased 1.6% from current algorithm. Finally, we discuss the feasibility of Event.Locky in an actual online environment through the implementation of a demonstration application.

Keywords: event data, web mining, text classification, spatial-temporal visualization

1. Introduction

Most organizations publish event information (such as sales promotions or exhibitions) on their webpages with the aim of attracting more users. Providing event data within spatial-temporal information will benefit user offline participation. There are event search services (e.g., ATND [1] and Peatix [2] etc.) that collect spatial-temporal event data through user uploads. However, these services provide event registration platforms that are focused on individual events (almost about seminars or lectures). They do not provide certain valuable business events (such as sales promotions or happy hours), which are more attractive for users. Except event search APIs, users also voluntarily share event data they feel interesting through social network services (SNSs). Some approaches address event-data extraction on Twitter [3], [4], [5]. Unfortunately, most event data on SNSs are not official. Although some organizations also tweet event data on official blogs, most tend to share event data on their own webpages. Official webpages contain accurate event data of organizations. Nevertheless, the communication efficiency of webpages is limited because only loyal users check familiar websites regularly. For new users (such as tourists or passersby who are waiting for the next train at an unfamiliar train station), it is impossible to obtain event data from webpages at an unfamiliar area.

In order to solve these problems, we explore the new concept of a system extracting event data from organization webpages (including official websites and SNSs) so that it can push nearby event data to users according to current spatial-temporal conditions. We consider a specific question of this concept: can web mining techniques be applied? Web mining [6] is the process of extracting useful information from the content of web document. In this study, we focused on event data mining and developed a system called Event.Locky for event data extraction.

The process flow of Event.Locky is shown in **Fig. 1**. First, a user's device sends location information obtained from a GPS sensor or by a user indication to the server. Second, the URLs of nearby organizations are obtained through a search engine according to location information. Third, a crawler in the server downloads webpage documents. Forth, our *web-data-record extraction* then converts the HTML documents into processable structured data and our *event-record classification* filters out non-event data. The web-data-record extraction, which we discuss in Section 3, is divided into three steps: *Inline-level Element Pruning* in Section 3.1, *Partial Tree Matching* in Section 3.2, and *Backtracking* in Section 3.3. We discuss event-record classification in Section 4 about our paragraph vector generation and the classifier. Finally, the server pushes event data including locations, times, images, and contents to the user's client. A client application in the user's device displays these event records as

[1] Graduate School of Engineering, Nagoya University, Nagoya, Aichi 464–8601, Japan
[2] Institute of Innovation for Future Society, Nagoya University, Nagoya, Aichi 464–8601, Japan
[3] Faculty of Information Science, Aichi Institute of Technology, Toyota, Aichi 470–0392, Japan
[a)] liao@ucl.nuee.nagoya-u.ac.jp
[b)] k.hiroi@ucl.nuee.nagoya-u.ac.jp
[c)] kaji@aitech.ac.jp
[d)] sakurada@nagoya-u.jp
[e)] kawaguti@nagoya-u.jp

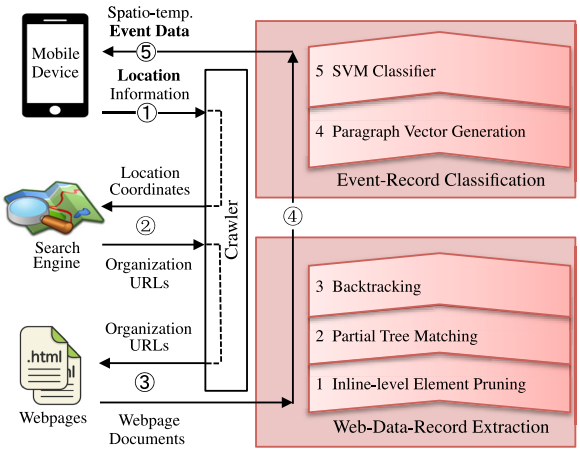


Fig. 1 The process flow of geographical-area event-data extraction.

spatial-temporal data.

In summary, the paper makes the following contributions:

- *We propose an event-data-extraction system called Event.Locky, which is a combination of our robust web-data-record-extraction algorithm and event-record-classification algorithm*, which converts webpages into data records apply to 83.96% event data on all types of webpages. We adopt HTML partial tree matching, which extracts data records on web documents with similar and continuous sub-structures. Event.Locky also uses our novel pruning process to remove unrelated elements for reducing sample noise and computation complexity, and our backtracking process that makes independent records extraction possible. Our Event.Locky increases F_1 score by 1.6% from current algorithms for event records extraction.
- *Event.Locky involves a high-performance event-record-classifier model.* For filtering out non-event data, we implemented a semantic-event-record-classifier that adopts a distributed representation model to generate paragraph vectors. Based on the neural network language model Word2Vec, we use a weighted optimization method to increase the classification performance. A support vector machine (SVM) is used with a radial basis function (RBF) kernel as the classifier. By cross validation, we achieved an F_1 score of 91.61%. Event.Locky currently can be applied to data in most Japanese websites.

2. Related Work

2.1 The Work of Web Data Extraction

Data on a webpage are composed of semi-structured data, which cannot be directly processed. Such data are generated by records in a database following particular rules. Intuitively, a region on webpage including one or more elements wraps a data record from a database. A web-data-record extraction algorithm generates wrappers that identify regions of data records and extract data records from webpages [7].

A number of approaches cover the theme of web-data-extraction based on *manual*, *tag*, *page-layout*, *vision* and *tree*. Manual [8], [9] and tag-based [10], [11] approaches are fundamental for web-data-extraction. By observing the structure of individual webpages, programmer writes a program (wrapper) to extract data records from regular expressions or particular paths. Manual approaches can accurately extract data records from an HTML document. They are frequently used to obtain periodic data (such as stock movements or price trends) from several webpages. Although programmers have to define different extraction patterns for each webpage, these approaches are not suitable for automatic data records extraction from heterogeneous webpages.

Table 1 Feature comparison of data-record-extraction approaches.

	Manual[8,9]	VIPS[13,14]	DEPTA[16,17]	ours
Automatic Webpage Records Extraction	×	○	○	○
Atomic Records Extraction	○	×	○	○
Unrelated Elements Pruning	×	×	×	○
Hidden Elements Extraction	○	×	○	○
Independent Elements Extraction	○	○	×	○

Kovacevic et al. [12] proposed a page-layout approach by observing the webpage layout. There are some design page-layout patterns (such as header, footer, left, right, and center) that allow a page to be segmented into fixed regions. This approach is effective for normal layout patterns of most web pages. However, it cannot be used on all webpages. The event webpage of a shop is most likely designed unconventionally; therefore, this page-layout-based approach is unsuitable.

Cai et al. [13], [14] proposed a vision-based page segmentation (VIPS) [15] approach, that simulates the human visual perception to segment webpage blocks, which distinguish different parts of a web page, such as lines, blanks, images, and colors. This approach can be applied to automatic webpages blocks segmentation. However, the extracted data records with VIPS are sometimes not atomic. For example, a list of titles in a ⟨*div*⟩ element may be estimated as one block (one record) by VIPS, rather than segmented into individual titles. In addition, some hidden items in a webpage (such as a slider bar) can not be extracted with VIPS; thus, it is not suitable for extracting event records from webpages.

Zhai and Liu [16], [17] developed a tree-based web-data-record-extraction approach called DEPTA [18]. This approach extracts web data from the viewpoint of webpage generation. Data records from a data table are typically presented in continuous regions on a webpage and formatted using fixed HTML templates. By matching partial tree structures, data records can be extracted; thus, it is suitable for converting from semi-structure data to structured data. However, this approach dose not take into account how to extract an independent data record without any similar continuous region on a page. It also does not take into account specific HTML elements, which are unrelated (such as inline elements) to partial tree matching but increase the complexity.

Event.Locky users our novel automatic web-data-record-extraction algorithm. **Table 1** shows feature comparison of what are the advantages in our algorithm compared to current approaches. Based on partial tree matching, we present a pruning

process to reduce un-related elements and a backtracking process that makes independent element extraction possible. We conducted a quantitative evaluation experiment, which is discussed in Section 5.2.

2.2 Filtering Out Unrelated Data

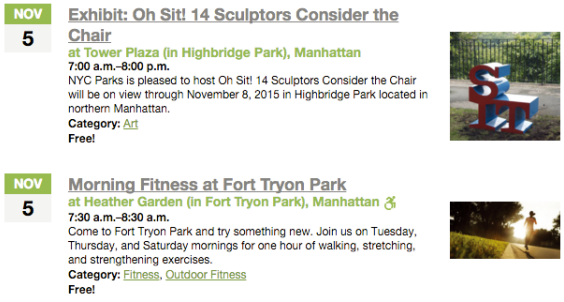
Data records extracted from webpages contain large amounts of unrelated data. A machine learning algorithm is commonly used unrelated data cleaning. In this study, we filtered out non-event data in a semantic text classifier. For a text classification task, mapping a paragraph to a dimensional vector is the key to maintaining classification performance. A one-hot paragraph vector [19] can solve many text classification problems. However, a one-hot paragraph may appear sparse and have high dimensionality, which may increase classification errors. Tomas et al. [20], [21], [22] proposed a neural network language model called Word2Vec, which clusters similar words (e.g. 'coupon' and 'campaign') with smaller cosine similarities. It effectively builds the semantic relationship between each word and solves the sparse problem of training a dataset. However, Word2vec works on the word level, which does not offer any direct implementation to obtain the paragraph vector.

Le et al. proposed a paragraph vector framework called PVDM [23]. It considers a paragraph token as another word called 'memory' in paragraph, and trains the paragraph token as the paragraph vector in Word2Vec processing. However, in this task, the paragraph as a record on webpages tends short. In training with few words, the convergence of PVDM is not complete. Repetitive training can solve this question but requires more computing resources. On the other hand, based on Word2Vec, PVDM solves some semantic problems. Nevertheless, it lacks an optimization for a specific classification task. Aiming at event records classification, we present a weighed vector method based on Word2Vec, which is more concise and achieves a higher F_1 score than existing methods.

We use a classifier to estimate data records belonging to event data. Some of the most frequently used methods for classification are the Naive Bayes classifier [24] and the k-Nearest Neighbors (k-NN) [25] classifier. However, because there is noise in training data, both Naive Bayes and k-NN easily appear overfit when the sample size increases. Thus, it is difficult to ensure their robustness. With Event.Locky, we use an SVM [26], [27] with a RBF kernel as the classifier. We conducted evaluation experiments, which are discussed in Section 5.1.

3. Web-Data-Record Extraction

Due to extracting event records from webpages, we developed our web-data-record-extraction algorithm. It identifies records as semi-structured data on webpages and converts them into structured data. The challenge is how to generate wrappers that segment an entire webpage into records and ensure each record includes an atomic data record. Partial tree alignment is an important concept in wrappers generation. **Figure 2** shows an example webpage [28], where each data record (in the list) has the date, address, description, categories, etc. They align continuously with a similar structure. Therefore, it is possible to detect



(a) Event-Record List on Webpage

Date	Title	Position	Description	Categories	Price
Nov 5 7:00-20:00	Exhibit...	Tower Plaza	NYC Parks...	Art	Free
Nov 5 7:30-8:30	Morning...	Heather Gar...	Come to Fort...	Fitness, Out...	Free

(b) Event Records in Database

Fig. 2 Event-records List in Webpage (a) is Generated from Event-record Table (b) in Database. Each row in the data table is filled with the same HTML structure to generate the list of event records. Therefore, event records have the same HTML structure. Wrappers can be segmented by matching HTML structures.

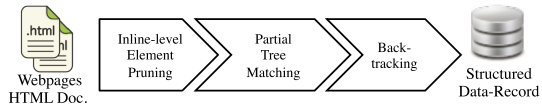


Fig. 3 The process flow of web-data-record extraction.

records according to matching similar structures on the webpage. We begin this section by explaining inline-level element pruning, which helps us prune the HTML tree to improve the accuracy of the wrapper generation and the reduce computational complexity of the web-data-record extraction algorithm. After that, we explain the partial tree matching algorithm. Finally, by the backtracking process, we can also extract some independent records without any continuous siblings. The process flow of these three steps is shown in **Fig. 3**.

3.1 Inline-level Element Pruning

The pruning process is used to remove the elements, which are not related to the HTML structure matching. A target of web designing is to enable users to identify data records on a webpage as easily as possible. Designers keep data records identifiable through space separations on webpages. Some HTML elements affect the spatial structure of webpages; however, others do not. In the HTML 2.0 [29] standard, HTML elements are divided into *block-level elements* and *inline-level elements*. Because inline elements normally do not significantly affect the webpage structure, we argue that they should not be used for partial tree matching on a webpage. In this case, inline elements may affect the result of partial tree matching. Consequently, pruning of inline elements is done to reduce the computational complexity for partial tree matching.

Pruning is started from scanning a HTML document D in breadth-first search at lines 2 to 7. As shown in Algorithm 1, we initialize a queue $Q[] = \{D.Body()\}$ with a single element, which is the $\langle body \rangle$ element of the HTML document D, and initialize a cursor i of Q from 0 at line 1. At line 3, children elements of $Q[i]$ are stored in the queue $E[]$. At lines 4 to 6, $E[]$ are iterated. At lines 5 and 6, if the child element is a block-level element, it is added into the end of the queue $Q[]$. At line 7, the i increases by

Algorithm 1: Inline-level Element-Pruning

input : An HTML document D.
output: A pruned elements queue Q[].

```
1 Q[]={D.Body()}, i=0;
2 while i < Q[].Size() do
3     E[] = Q[i].Children();
4     foreach element e of E do
5         if e.isBlock() then
6             Q[] ⟵ e;
7     i = i + 1;
8 return Q[];
```

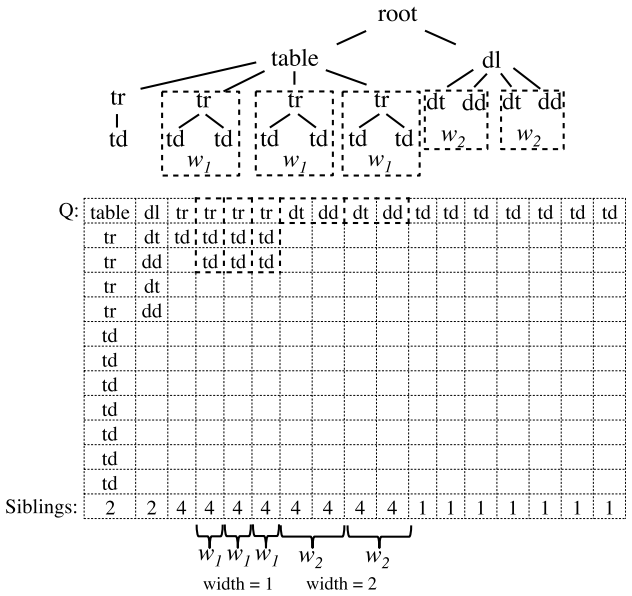


Fig. 4 Example of Partial Tree Matching with 2 Types of Wrapper Structures, as Shown in HTML Tree. The map below the tree structure extends each element in Q as first row. From the second row, there are corresponding children elements. The last row shows the number of siblings of the element that determines the width of the matching window. It was observed that wrappers can be extracted by matching columns in each window (the dotted region in the table).

1. The pruned tree is stored in the Q[], which is used for partial tree matching the next step.

3.2 Partial Tree Matching

The reason of adopting breadth-first search is that a webpage is designed from outline to detailed, namely from a large to a small area. A breadth-first search precisely scans a webpage from a large area to a small area. The Q[] from the pruning process happens to be a sequence arranged as a breadth-first search, which can be directly used for partial tree matching.

We explain the process of partial tree matching and wrappers extraction in an example of a tree structure, which is shown at the top of **Fig. 4**. We assume there are two wrappers W_1 and W_2 of six partial trees. For ease of explanation, we extend the pruned tree Q[] into a first row of the map, as shown at the bottom of Fig. 4. The first row is assigned each element in Q[] as a breadth-first search. From the second row, there are corresponding descendant elements, which are also aligned through breadth-first search, namely the *partial tree structure* of the element. The matching process is from left to right. Obviously, the three ⟨*tr*⟩ records can be extracted by matching each of their columns.

Algorithm 2: Partial Tree Matching

input : Q[] is from inline-level element-pruning.
output: R[][] stores extracted records, and Q[] with marked elements.

```
1  R[][] = {}, i = 0;
2  while i < Q[].Size() do
3      if Q[i].isExtracted() = true then
4          continue;
5      else
6          w = 1;
7          while w < Q[i].SiblingsNumber() / 2 do
8              l =Match({Q[i] to Q[i + w − 1]}, {Q[i − w] to Q[i − 1]});
9              r =Match({Q[i] to Q[i + w − 1]}, {Q[i + w] to
                 Q[i + 2w + 1]});
10             if l or r then
11                 R[] ⟵ {Q[i] to Q[i + w − 1]};
12                 {Q[i] to Q[i + w − 1]}.MarkExtracted();
13                 {Q[i] to Q[i + w − 1]}.MarkAllChildrenExtracted();
14                 {Q[i] to Q[i + w − 1]}.MarkAllParentsExtracted();
15                 i = i + w − 1;
16                 break;
17             else
18                 w = w + 1;
19     i = i + 1;
20 return R[][], Q[];
```

However, matching each of the elements is not suitable in some exceptional cases. Some records may be constructed with multiple elements in the same level, such as two ⟨*div*⟩s or more. Figure 4 gives an example in which a ⟨*dt*⟩ element and a ⟨*dd*⟩ element construct one record. In this case, the matching function needs to be compared to both elements. We designed a matching window mechanism to address this case. It determines how many columns of same-level-elements are combined to match. The width of the window loops from one to half the number of siblings elements, which is shown as the last row in Fig. 4. In this case, when the window width increases to two, wrapper W_2 is generated.

Algorithm 2 shows the details of the partial tree matching. The Q[] is from the inline-level element-pruning process as the input. At line 1, because a record may include multiple elements, we initialize a two-dimensional array R[][] to store extracted records, and an i for Q[]. At line 3, if $Q[i]$ has been marked as an extracted element, skips it and continue to the next loop. The extracted-element-marking process is shown at lines 12 to 14. At line 6, it initializes a w of the matching window size from value 1. At line 7, the w increases by 1 (at line 18) until it reaches half the number of $Q[i]$ siblings. At lines 8 and 9, it matches the columns in the current window with the left and right windows, and stored the results in two boolean variables l and r. At line 10, if l or r is true, elements of the current window are added into array R[] as a record (line 11). At line 12, all the elements of the current window are marked as 'extracted'. At line 13, all the child elements of the current window are marked as 'extracted'. At line 14, all the parent elements of the current window are marked as 'extracted'. Line 15 skips the extracted elements of the current window and updates i to $i + w - 1$. Line 16 breaks the loop in line 7 and goes to line 19. If it is not matched at line 10, the w

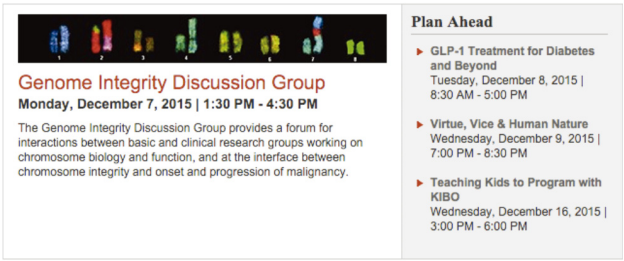


Fig. 5 Common Structure of Independent Record: 3 Event Records are Aligned as a List on the Right Side. They can be extracted by partial tree matching. Another highlight event record is drawn on the left side independently, which cannot be extracted. All have the same parent element. We mark the parent element when those 3 records are extracted. The independent record can be extracted by observing its parent element.

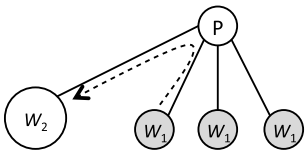


Fig. 6 Wrapper Generation by Backtracking the Marked Parent Element. When wrapper W_1 is generated, the parent element P is marked as 'extracted'. After that, all tree elements are backtracked to generate wrapper W_2, which has not been extracted yet but has marked the parent element.

increases to 1 at line 18. At line 19, i increases to 1. The extracted records in the array $R[][]$ and marked $Q[]$ are returned at line 20.

This partial tree matching algorithm extracts major data records on a webpage. However, independent records without any continuous and similar sibling are not extracted. At the next stage, we focus on independent records extraction through a backtracking process.

3.3 Backtracking: Independent Record Extraction

When a web designer tries to emphasize an important event data record, the record probably can be drawn as a special structure on the page, although it may be selected from the same data table. **Figure 5** shows a typical independent record on a webpage [30]. We can see that the designer draws the event record 'Genome Integrity Discussion Group' in a larger area than the others. Because this event element has no sibling, the partial tree matching algorithm does not work in this case.

Fortunately, by observing a large amount of samples like this, we found a feature of independent records can be used. Records from a data table tend to be drawn at the same depth in the HTML tree with the same parent element. If we mark each extracted parent element during partial tree matching, some independent wrappers with a marked parent element can be extracted. Because this process is after partial tree matching and scans the tree in breadth-first search one more time, we call it a *backtracking* process. Therefore, by backtracking extracted records, these independent records may be extracted to a certain extent. As shown in **Fig. 6**, the wrapper W_1 is generated by partial tree matching and W_2 wraps an independent record. W_2 can be generated by marking parent element P, which is marked when W_1 is generated.

As shown in Algorithm 3, we scan queue Q again as breadth-first search. The $Q[]$ and extracted $R[][]$ are outputted from partial

Algorithm 3: Backtracking

input : $R[][]$ and $Q[]$ are from partial tree matching.
output: $R[][]$ stores extracted records.

```
1 i = 0;
2 while i < Q[].Size() do
3     if Q[i].isExtracted() = false then
4         if Q[i].Parent().isExtracted() = true then
5             R[] ⟵ Q[i];
6             Q[i].MarkExtracted();
7             Q[i].MarkAllChildrenExtracted();
8     i = i + 1;
9 return R[][];
```

tree matching as the input. At line 1, an i for Q is initialized. At line 2, i is looped from 0 until the size of $Q[i]$. At line 3, if $Q[i]$ has not been extracted yet, the program runs lines 4 to 6. At line 4, if the parent element of $Q[i]$ is marked as an extracted element, $Q[i]$ is extracted and added into $R[]$ at line 5. Note that, the marking is done during partial tree matching as shown in Algorithm 2 at lines 12 to 14. At line 6, $Q[i]$ is marked as 'extracted'. At line 7, all the children elements of $Q[i]$ are marked as 'extracted'. At line 11, all the extracted records in array $R[]$ are returned.

In summary, the above three processes extract data records on a webpage. However, these primary processed records can not be pushed to users yet. There is a vast amount of non-event data; therefore, it is necessary to filter out such data. In the next section, we present our classification algorithm to filter out non-event data using machine learning.

4. Event-Record Classification

4.1 The Background of Paragraph Vector Generation

We filter out non-event data as a text classification task. We call the text of an event record a 'paragraph'. As a semantic classifier for text, it is essential to start from mapping the paragraph into a mathematical model. *One-hot representation* [19] is the most commonly used model. A paragraph X is mapped into a vector as

$$X = [w_1, w_2, ..., w_i, ..., w_n] \qquad \omega_i \in \{0, 1\}$$

where the dimensionality n is predefined to the size of a dictionary. If a word w_i appears in the paragraph, it is set to the binary value 1; otherwise 0. Although this model can solve many problems in text classification, it has two serious disadvantages. First, due to sparseness, each word is independent. The one-hot representation model cannot build semantic relationships between words (e.g., 'coupon' and 'campaign' are unrelated in the one-hot representation model, although they have similar linguistic functions). Therefore, the classification accuracy is restricted to the coverage of training data. Second, paragraph vectors are mapped to a high-dimensional vector with the dimensionality n ($n > 200,000$ in Japanese). The model with a high-dimension vector may reduce the efficiency of the classifiers. This is a disadvantage for some classifiers.

Distributed representation models have been recently presented. Word2Vec [20], [21], [22] is the most common. By observing a word ω_i that appears near its context $[\omega_{i-c}, \omega_{i-1}]$ and $[\omega_{i+1}, \omega_{i+c}]$ with the probability $P(\omega_i|context(\omega_i))$, it clusters sim-

ilar words with smaller cosine similarities in a low-dimensional vector space (usually $n = 50$, 100 or 200). It mitigates the disadvantages with the one-hot representation model. However, word2vec works on the word level, which does not offer any direct implementation to obtain the paragraph vector. Furthermore, as a word clustering algorithm, Word2Vec has no optimization for specific classification tasks. In this paper, we present a weighted optimization paragraph vector mapping method that works on word vectors and improves the classification of paragraph vectors.

4.2 Optimization Method: Weighted Paragraph Vectors

First, we obtain a paragraph vector from word vectors. We map a paragraph into an n-dimensional vector X using the centroid of word vectors, which constitute the paragraph. This is described as

$$X = \frac{1}{m}\sum_{i=1}^{m}\omega_i \qquad \omega_i \in \mathbb{R}^n$$

where m is the number of words in a paragraph, ω_i are the words constituting the paragraph, and n is the dimensionality of a paragraph vector, which equals the dimensionality of the word vector because we map paragraphs to the same vector space as words. It can be seen that the paragraph vector is mapped to the mean of words vectors. With this method, each word vector has a uniform weight of 1. In other words, each word vector gives the same support for the paragraph vector.

Second, we explain the concept and give a description of a weighted paragraph vector. We assume some words in the dictionary are strongly related to the polarity of event-data classification (such as the words 'firework show', 'exhibition' etc). In comparison, some words are weakly related to classification (such as 'the' and 'and' etc). If a word is strongly related for classification, it should be given a higher weight. Geometrically, the paragraph vector should 'drift' close to the higher-weight word vectors; otherwise, far from the lower-weight word vectors. This is the concept of a weighted paragraph vector. Therefore, the equation of a weighted paragraph vector is given as

$$X = \frac{\sum_{i=1}^{m}\theta_i\omega_i}{1+\sum_{i=1}^{m}\theta_i} \qquad \theta_i \in [0,1] \tag{1}$$

where θ_i is the weight of the ith word ω_i in a paragraph. The definition domain of θ_i is a closed interval from 0 to 1. Because in some cases, the sum of weights may be 0 in a paragraph, we add 1 to the denominator to avoid an infinite value.

The training event dataset is given as $\{(p^{(j)}, y^{(j)}); j = 1, 2, ..., M\}$, where $p^{(j)}$ is the jth paragraph in the dataset (with size M) corresponding to a target variable $y^{(j)} \in \{0, 1\}$. We obtain the weight θ_i by using Naive Bayes, which calculates the probability of an ω_i that belongs to a target variable $y^{(i)}$ as

$$p(y|\omega_i) = \frac{p(\omega_i|y)p(y)}{p(\omega_i)} \tag{2}$$

where the target variable y is set to 1 if the paragraph is an event record; otherwise, set to 0. When the training dataset is large enough, we approximatively consider

$$\begin{aligned} p(\omega_i) &\approx \frac{M(\omega_i)}{M} \\ p(y) &\approx \frac{M_y}{M} \\ p(\omega_i|y) &\approx \frac{M(\omega_i, y)}{M_y} \end{aligned} \tag{3}$$

where the integer M is the number of training datasets. $M(\omega_i)$ is that in which the word ω_i appears. M_y is that with y, and $M(\omega_i, y)$ is that including ω_i also with y. By substituting Eq. (3) into Eq. (2), we obtain the probability of y given ω_i as

$$p(y|\omega_i) = \frac{M(\omega_i, y)}{M(\omega_i)} \qquad p(y|\omega_i) \in [0,1]$$

Note that, $p(y = 1|\omega_i)$ and $p(y = 0|\omega_i)$ are complementary. Therefore, any y can be used to obtain the probability $p(y|\omega_i)$. When $p(y|\omega_i)$ approximates to 0.5, ω_i is weekly related for classification; When it approximates to 0 or 1, ω_i is strongly related for classification. Therefore, we give the mapping function to calculate the weight θ_i.

$$\theta_i = |1 - 2p(y|\omega_i)| \qquad \theta_i \in [0,1] \tag{4}$$

Substituting Eq. (4) into Eq. (1), the weighted paragraph vector can be obtained.

We implement the classifier using an SVM with an RBF kernel. The SVM is trained by weighted paragraph vectors, which result in a better classification. The details of the results are given in the next section.

5. Evaluation Experiments

In this section, we evaluate our event-record-classification algorithm, the capability of event data extraction by a combination of the web data extraction algorithm and the event-record-classification algorithm, and we show a demonstration experiment to discuss the feasibility of Event.Locky. First, the event classifier must be trained. For event-record-classification evaluation, we manually labeled 23,000 records as training data, which are extracted from top-pages and their sub-pages of 96 shops in our web data extraction algorithm. These 96 shops are located in UNIMALL and ESCA [31], two underground shopping streets at the railway station in Nagoya, Japan. We use these training data to pre-training Word2Vec and our Event-Record-Classification algorithms. Second, we use about 4,000 of 23,000 records, which are extracted from top-pages of 96 shops, to evaluate event-data-extraction algorithms. Third, we develop a demonstration system to evaluate the feasibility when Event.Locky runs at actual Internet environment.

5.1 Results Comparison of Event-Record-Classification Algorithms

Note that we re-checked data set from previous studies [32], [33]. We adopted cross validation that sets 90% as the training dataset and 10% as the test dataset. We used Japanese morphological analyzer Kuromoji [34] for word segmentation and an open source library SVM, LIBSVM [35], developed by Chang et al. [36] as the classifier. The evaluation of the classification algorithms involved precision, a recall, and a F_1 score.

Table 2 The evaluation of precision, recall and F_1 score of each model (We do parameter optimizations for each C and γ. It scan logarithm of C and γ until achieve best F_1).

Model	$log_2 C$	$log_2 \gamma$	Prec.	Rec.	F1
ONE-HOT	4	−6	81.62%	79.42%	80.51%
W2V MEAN	5	1	86.92%	88.70%	87.80%
PVDM	5	1	88.46%	88.94%	88.70%
PVDM x10	8	1	88.73%	89.32%	89.03%
Ours	6	1	92.48%	90.46%	**91.61%**

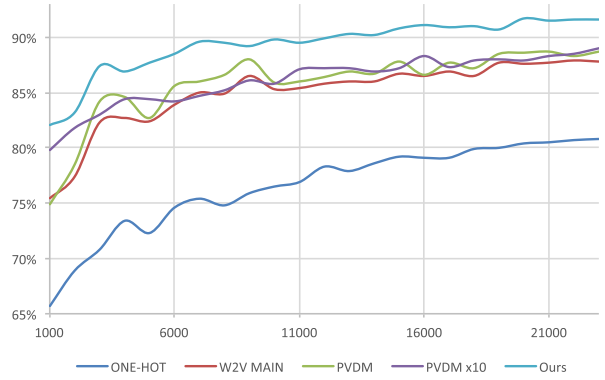


Fig. 7 The F_1 Score Transition of Models in each 1,000 Training Data. The vertical axis is F1-score of each algorithm. The horizontal axis is size of training set.

We implemented and compared the One-Hot Representation model, Word2Vec Mean Vector model, PVDM, and our Weighted Mean Vector model. The mapped paragraph vectors and their target variables were inputted into the SVM for supervised training. The kernel function we adopted was the RBF. The RBF kernel has two undetermined parameters - the penalty factor C and the influence factor γ. Each parameter was tuned by parameter optimization to find the best F_1 score in each model.

Table 2 lists the experiment results. There was a significant improvement when a distributed representation was adopted instead of one representation. From the data, we can see PVDM is significantly better than One-hot Representation, and perform at the same level as Word2Vec Mean Vector. There has to be mentioned an important feature of event records on the webpage. These event records are short as one or two sentences, even a phrase. From **Fig. 7**, we can see when the model is trained in few words, the convergence of PVDM is not complete. The classification results are not satisfactory. For a complete convergence, we did a repetitive training experiment (PVDM x10) that trains each paragraph in 10 loops. We can see a marked improvement in a small training size. On the other hand, based on Word2Vec, PVDM solves some semantic problems. Nevertheless, it lacks an optimization for a specific classification task. Our method solves this lack through weighting for each word on a given class. As a result, our method achieves a higher F1-score than others. Our algorithm is more specifically suited to our event data classification task.

5.2 Results Comparison of Web-Data-Extraction Algorithms

The crawler of Event.Locky is developed in a Java library jsoup [37]. It is used to download webpages from the Internet as HTML documents. The downloaded webpages are sent to our web-data-record-extraction algorithm.

Table 3 The evaluation event record extraction capability. We calculate the precision, recall and F_1 score of each web-data-extraction algorithm.

Model	Prec.	Rec.	F1
VIPS	89.89%	40.23%	55.58%
DEPTA	98.72%	70.98%	82.36%
ours without backtracking	98.72%	70.98%	82.36%
ours with backtracking	98.76%	73.02%	**83.96%**

The web-data-record extraction algorithm is a module in the Event.Locky system. The evaluation of the web-data-record extraction algorithm should aim to be independent of other modules and based on final event records extraction results. All other modules being equal, we evaluate the web-data-record extraction by comparing each event records extraction result on each web-data-record extraction algorithm. Therefore, from results comparison of event-record-classification algorithms in Section 5.1, we choose the most suitable one, our algorithm, to be a combination with each web-data extraction algorithm. Then, we evaluate the final event extraction result on each web-data-record extraction algorithm. In this experiment, we compare VIPS, DEPTA and our web-data-record extraction algorithm. For quantitative evaluation of 'backtracking', we divide our algorithm into two experiments: the algorithm without backtracking and another algorithm with backtracking.

As shown in **Table 3**, we can see our algorithm got the similar F1-score from DEPTA without the backtracking process. Without backtracking, our algorithm achieves the same level of DEPTA. The pruning processing and partial tree matching before backtracking provide the formatted data structure for backtracking. The backtracking processing improves the recall by 2.04% and the F1 score by 1.6% from DEPTA. Analyzing the reason of a higher F1-score depends on the independent event-record extraction by backtracking processing. For example, on the webpage 'komeda [38]', when the list on the right side is extracted by partial tree matching, a larger banner on the left side can be extracted by backtracking.

We compare with with the experiment of the event-record-classification algorithm, and analyze the reason the reason why the precision is higher than the event record classification experiment but the recall is lower than it. Because the density of event record in top-pages is much greater than that in sub-pages, it has a higher hit ratio. In summary, this experiment verified the combination method we proposed is more suitable to an event data extraction task.

5.3 Demonstration Experiment

We developed an application of Event.Locky to validate its feasibility. Our aim is to publish event data for mobile users at anytime and anyplace throughout Japan. **Figure 8** shows the system flow. The application on mobile devices sends device location information, which is obtained with the built-in GPS sensor or indicated by the user to the server (step 1). Then the server searches nearby organizations' information including their coor-

Fig. 8 Flow of Event.Locky Demonstration Application.

dinates, addresses, website URLs, and the type from the search engine Google Places [39] (step 2). Next, the server requests these websites and downloads their webpages by using an inner crawler (steps 3 and 4). Because there may be multiple search results, the crawler is designed to be a multi-threading program that can download webpages from all the organizations simultaneously. After that, our event-record-extraction algorithm works on the web documents (step 5). Finally, the server sends the event records to the user client and displays these records on the mobile device (step 6). It is worth mentioning that except implementing texts of event records extraction, we also implemented the extractions of times, images, and hyperlinks from event records.

The server of Event.Locky is deployed at a public data center with two Intel Xeon E3 CPUs and 2 GB memory. The programs were written in Java and run in Apache Tomcat 8 with Open-JDK 8.0. The maximum network throughput is 100 Mbps. The client runs on the iPhone 6 plus with 54-Mbps Wifi network. The communication between the server and the client passes through the Internet.

Figure 9 shows the client application main interfaces of Event.Locky. We categorized event data according to the type of organization into four main categories – *Exhibiton* (museums, parks, galleries etc.), *Gourmet* (restaurants, cafes, bars etc.), *Shopping* (malls, stores, markets etc.) and *Amusement* (bowling alleys, cinemas, clubs etc.). We began by testing the processing time of Event.Locky at train stations of ten cities in Japan. At each area event data is extracted in the four categories (Exhibition, Gourmet, Shopping and Amusement) and the processing time is counted.

The retrieval processing times [s] are shown in **Fig. 10** in seconds. The average retrieval time was 2.1 s for Exhibition, 1.84 s for Gourmet, 2.44 s for Shopping and 4.61 s for Amusement for each organization. We argue that these are acceptable results for users. The geographical distance does not evidently impact retrieval time. The bottleneck is on the crawler when analyzing the delay in the retrieval time. Some organizations were unable to provide normal services due to the fact that their websites were not updated, which lead to crawler timeout. By upgrading the bandwidth and network performance, this problem can be properly solved. Nonetheless, our event-data-extraction algorithm has a sufficient capacity to support high-speed online retrieval.

5.4 Limitations

We focused on event-data extraction by using web mining techniques. The main methods of information extraction are based on text classification. We also found that some event information on webpages is presented as multimedia data (such as event

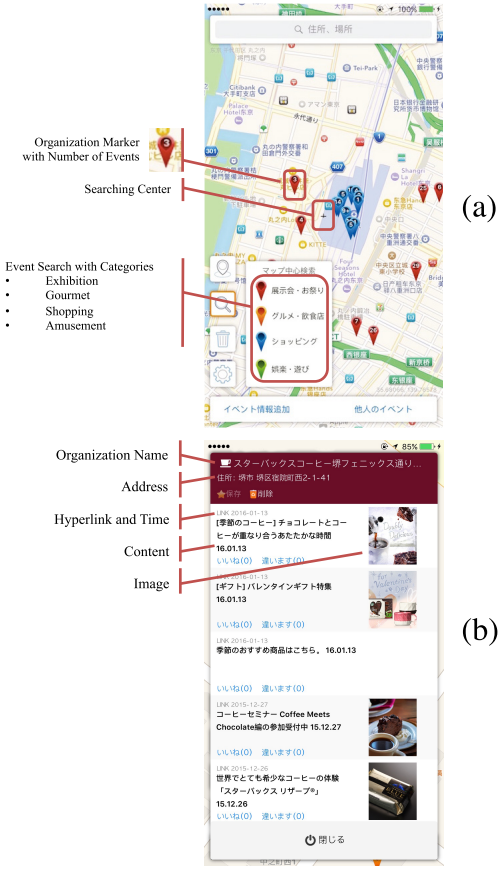


Fig. 9 Main Interfaces of Event.Locky Demonstration Application. (a) Markers with the number of event records of organizations on electronic map; Searching coordinate set at the center of the map; defaults to the current location obtained from a GPS sensor. Can also be indicated by dragging on the map. (b) Detailed event records list from touching a marker; event records are sorted by time. We also implemented time extraction by the regular expression and the image extraction from HTML tags.

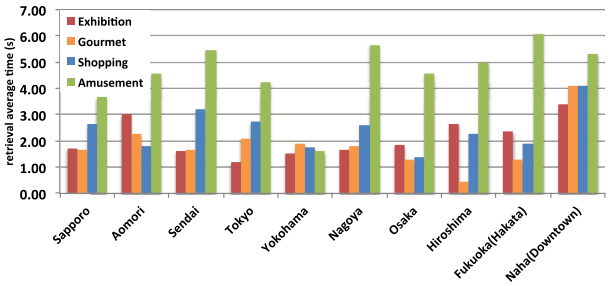


Fig. 10 Processing Time at Train Stations or Downtowns of 10 Cities in Japan.

images and animation of event advertisement). In this case, a limitation of the text method is that multimedia event data can not be extracted. Combining image processing and deep learning with web mining may be promising to address this limitation. Our algorithm may provide mass training data about images and the related text in records. It properly supports image capture approaches, which automatically generate the description text from images.

Another limitation is regarding language. Tourist who have no knowledge of the area find it difficult to obtain event data. For instance, as the 2020 Tokyo Olympic Games approach, the foreign tourists will increase. However, due to the fact that most web-

pages in Japan are in Japanese, the event data will not directly benefit foreign tourists. We need to solve this language problem by adopting machine translation techniques.

6. Conclusion

We proposed a feasible online event data extraction system called Event.Locky, which extracts event data from organization webpages and displays event records on mobile devices as spatial-temporal data. Event.Locky makes it possible to collect and reuse event data from orgnizations webpages in a geographical area.

We obtained three key experiments to evaluate the feasibility of Event.Locky. First, for converting semi-structured web documents into processable structured data, we implemented our web-data-record extraction algorithm.

Through an experiment involving webpages of 96 shops at Nagoya station, our event-classification algorithm achieved an F_1 score of 91.61%, an increase of 3.07% from current event-classification algorithms. The combinations of our event-classification algorithm and our data-record-extraction algorithm achieved the F_1 score 83.96% to extract event records from webpages. That increased 1.6% from the current algorithm. Finally, we discuss the feasibility of Event.Locky in an actual online environment through the implementation of a demonstration application.

For future work, we will investigate multimedia event-data extraction from webpages and attempt to combine image processing techniques (such as deep learning and optical character recognition) with web mining. We will also implement event-image extraction. For non-Japanese speakers, we will investigate machine translation techniques that will help them obtain valuable event data.

References

[1] ATND (online), available from ⟨https://atnd.org/⟩ (accessed 2016-04-27).

[2] Peatix (online), available from ⟨http://peatix.com/⟩ (accessed 2016-04-27).

[3] Hila, B., Mor, N. and Luis, G.: Beyond Trending Topics: Real-world Event Identification on Twitter (2011).

[4] Watanabe, K., Ochi, M. and Okabe, M.: Jasmine: A Real-time Local-event Detection System Based on Geolocation Information Propagated to Microblogs, *Proc. 20th ACM International Conference on Information and Knowledge Management*, pp.2541–2544 (2011).

[5] Hamed, A., Christian, S. and Michael, G.: Eventweet: Online Localized Event Detection from Twitter, *Proc. VLDB Endowment*, pp.1326–1329 (2013).

[6] Cristobal, R. and Sebastian, V.: Educational Data Mining: A Survey from 1995 to 2005, *Expert Systems with Applications*, Vol.33, No.1, pp.135–146 (2007).

[7] Ferrara, E., Meo, P., Fiumara, G. and Baumgartner, R.: Web Data Extraction, Applications and Techniques: A Survey, *Knowledge-Based Systems*, pp.301–323 (2014).

[8] Jussi, M.: Effective Web Data Extraction with Standard XML Technologies, *Computer Networks*, Vol.39, No.5, pp.635–644 (2002).

[9] Robert, B., Sergio, F. and Georg, G.: Visual Web Information Extraction with LIXTO, *VLDB*, pp.119–128 (2001).

[10] Lin, S. and Ho, J.: Discovering Informative Content Blocks from Web Documents, *Proc. ACM SIGKDD'02* (2002).

[11] Crivellari, F. and Melucci, M.: Web Document Retrieval Using Passage Retrieval, Connectivity Information, and Automatic Link weighted, *9th Text Retrieval Conference* (*TREC-9*) (2000).

[12] Kovacevic, M., Diligenti, M., Gori, M. and Milutinovic, V.: Recognition of Common Areas in A Web Page Using Visual Information: A Possible Application in A Page Classification, *ICDM 2003, Proc. 2002 IEEE International Conference*, pp.250–257 (2002).

[13] Cai, D., Yu, S., Wen, J. and Ma, W.: VIPS: A Vision-based Page Segmentation Algorithm, Microsoft Technical Report, MSR-TR-2003-79 (2003).

[14] Cai, D., Yu, S., Wen, J. and Ma, W.: Extracting content structure for web pages based on visual representation, *Web Technologies and Applications*, pp.406–417, Springer Berlin Heidelberg (2003).

[15] VIPS (online), available from ⟨https://github.com/tpopela/vips_java⟩ (accessed 2015-09-11).

[16] Zhai, Y. and Liu, B.: Web Data Extraction Based on Partial Tree Alignment, *Proc. International Conference on World Wide Web* (*WWW 2005*), pp.76–85 (2005).

[17] Zhai, Y. and Liu, B.: Structured Data Extraction From the Web Based on Partial Tree Alignment, *IEEE Trans. Knowledge and Data Engineering*, pp.1614–1628 (2006).

[18] DEPTA (online), available from ⟨https://github.com/seagatesoft/sde/⟩ (accessed 2015-09-11).

[19] Jie, J., Chan, T. and Zhao, Q.: Clustering Large Sparse Text Data: A Comparative Advantage Approach, *Journal of Information Processing*, pp.242–251 (2010).

[20] Mikolov, T., Chen, K., Corrado, G. and Dean, J.: Efficient Estimation of Word Representations in Vector Space, *arXiv preprint arXiv*, pp.1301–3781 (2013).

[21] Mikolov, T., Sutskever, I., Chen, K. and Corrado, G.: Distributed Representations of Words and Phrases and Their Compositionality, *Advances in Neural Information Processing Systems*, pp.3111–3119 (2013).

[22] Mikolov, T., Le, Q.V. and Sutskever, I.: Exploiting Similarities Among Languages for Machine Translation, *arXiv preprint arXiv*, pp.1309–4168 (2013).

[23] Le, Q. and Mikolov, T.: Distributed Representations of Sentences and Documents, *Proc. 31st Intl. Conference on Machine Learning*, pp.1188–1196 (2014).

[24] Lewis, D.D.: Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval, *Machine Learning ECML-98*, pp.4–15 (1998).

[25] Yiming, Y. and Xin, L.: A Re-examination of Text Categorization Methods, *Proc. 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.42–49, ACM (1999).

[26] Thorsten, J.: *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*, Springer Berlin Heidelberg (1998).

[27] Fabrizio, S.: Machine Learning in Automated Text Categorization, *ACM Computing Surveys* (*CSUR*), Vol.34, No.1, pp.1–47 (2002).

[28] NYC Park (online), available from ⟨http://www.nycgovparks.org/events/⟩ (accessed 2015-11-01).

[29] RFC1866 (online), available from ⟨http://www.ietf.org/rfc/rfc1866.txt⟩ (accessed 2016-04-27).

[30] NYAS (online), available from ⟨http://www.nyas.org/⟩ (accessed 2015-11-01).

[31] Nagoya Station (online), available from ⟨http://www.meieki.com/station_sa.php⟩ (accessed 2015-04-20).

[32] Liao, C., Kaji, K., Hiroi, K. and Kawaguchi, N.: An Event Data Extraction Method Based on HTML Structure Analysis and Machine Learning, *Computer Software and Applications Conference* (*COMPSAC*), *2015 IEEE 39th Annual*, pp.217–222, IEEE (2015).

[33] Liao, C., Kaji, K., Hiroi, K. and Kawaguchi, N.: HTML A Proportion of Event Data Extraction based on HTML Structure Analysis and Machine Learning, *Ubiquitous Computing System*, pp.1–7 (2015). (In Japanese)

[34] Kuromoji (online), available from ⟨http://www.atilika.org/⟩ (accessed 2015-09-03).

[35] LIBSVM (online), available from ⟨http://www.csie.ntu.edu.tw/_cjlin/libsvm/⟩ (accessed 2015-09-27).

[36] Chang, C.C. and Lin, C.J.: LIBSVM: A Library for Support Vector Machines, *ACM Trans. Intelligent Systems and Technology* (*TIST*), pp.1–39 (2011).

[37] JSOUP (online), available from ⟨http://jsoup.org/⟩ (accessed 2015-09-03).

[38] Komeda (online), available from ⟨http://www.komeda.co.jp/⟩ (accessed 2016-03-07).

[39] Google Places (online), available from ⟨https://developers.google.com/places/⟩ (accessed 2015-09-01).

Chenyi Liao was born in 1987. He is a doctoral student in Graduate School of Engineering, Nagoya University and has been engaged in the Information Processing Society of Japan since 2011. His research interest is data mining and natural language processing. He is a member of the IEEE.

Kei Hiroi received her B.S. degree in Engineering in 2004 from Tohoku University. She worked NTT EAST from 2004 to 2011. She received her Master of Media Design and Ph.D. in Media Design in 2011 and 2014, respectively from Keio University. From 2014, she has been a designated assistant professor in the Institute of Innovation for Future Society, Nagoya University.

Katsuhiko Kaji received his Ph.D. in information science from Nagoya University in 2007. He became a RA at NTT Communication Science Laboratories in 2007 and an assistant professor in Nagoya University in 2010. Currently, he is associate professor of Faculty of Information Science, Aichi Institute of Technology from 2015. His research interests include indoor positioning and remote interaction. He is a member of JSSST.

Ken Sakurada received the B.E., M.E., and Ph.D. degrees from Tohoku university, Japan, in 2009, 2011, and 2015 respectively. He is currently an Assistant Professor at the Graduate School of Engineering, Nagoya university. From April 2013 to March 2014, he was a visiting researcher at Robotics Institute, Carnegie Mellon University, PA, USA. His research interests are in computer vision, remote sensing, and robotics. From April 2012 to March 2014, he was a recipient of the Research Fellowship of JSPS.

Nobuo Kawaguchi received his B.E., M.E. and Ph.D. in Computer Science from Nagoya University, Japan, in 1990, 1992, and 1995, respectively. From 1995 he was an associate professor in the Department of Electrical and Electronic Engineering and Information Engineering, School of Engineering, Nagoya University. Since 2009, he has been a professor in the department of Computational Science and Engineering, Graduate School of Engineering, Nagoya University. His research interests are in the areas of Human Activity Recognition, Smart Environmental System and Ubiquitous Communication Systems.