A Web Information Extraction Framework with Adaptive and Failure Prediction Feature

SUDHIR KUMAR PATNAIK and C. NARENDRA BABU, MS Ramiah University of Applied Sciences, India

The amount of information available on the internet today requires effective information extraction and processing to offer hyper-personalized user experiences. Inability to extract information by using traditional and machine learning techniques due to dynamic changes in website layout pose significant challenges to the technical community to keep up with such changes. The focus of existing machine learning-based information extraction framework is only on information extraction by using core extraction logic that is susceptible to website changes, thus missing out core features such as ability to handle proactive failure prediction and intelligent information extraction capabilities. The aim of this article is to build a robust and intelligent information extraction framework with the ability not only to proactively predict website failure but also automatically extract information using deep-learning techniques using You Only Look Once and Long Short-term Memory (LSTM) networks. The proactive detection using LSTM detects new location of the web page due to layout changes and enables automatic extraction of information of the new web page. A real-world case with retail website for intelligent information extraction and an offline experimentation environment is setup to demonstrate proactive failure prediction and automatic extraction resulting in high failure prediction, precision and recall of object detection and information extraction.

CCS Concepts: • **Information systems**;

Additional Key Words and Phrases: Adaptive, data mining, deep learning, prediction algorithm, predictive modeling

ACM Reference format:

Sudhir Kumar Patnaik and C. Narendrababu. 2022. A Web Information Extraction Framework with Adaptive and Failure Prediction Feature. *J. Data Inform. Quality* 14, 2, Article 12 (March 2022), 21 pages. <https://doi.org/10.1145/3495008>

1 INTRODUCTION

Predicting failures in websites has never been easy, be it crashes or memory outages or incidents. Traditional **Information Technology (IT)** operation teams have transformed to be **Site Reliability Engineering (SRE)** or **Development Operations (DevOps)**, to handle these special situations. On demand web application availability anytime, anywhere has made DevOps a specialized field for the future of IT industry. Although the application and infrastructure deployment strategies have evolved from on-premise data centers to cloud, failures continue to exist due to various

Sudhir Kumar Patnaik and C. Narendrababu contributed equally to this research.

Authors' address: S. K. Patnaik and C. N. Babu, MS Ramiah University of Applied Sciences, MSR Nagar, University House, Mathikere, Bangalore, Karnataka, India, 560054; emails: {skpatnaik9, narendrababu.c}@gmail.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

1936-1955/2022/03-ART12 \$15.00

<https://doi.org/10.1145/3495008>

unpredictable and some predictable situations [14, 30]. In the information extraction field, on-demand content use by the internet users [2] is even more challenging to predict the failures. Web applications continue to change dynamically meeting user expectations and hence failures do happen, unplanned or sometimes planned. Website failures occur for many reasons, such as: (i) page not found; (ii) website down for maintenance; (iii) user request to server timed out, i.e., could not be processed within specified time limit; (iv) system ran out of memory; (v) user login is invalid due to incorrect login name or password or other password related issues; and (vi) location of the page that user tries to access has changed. Mining server log file data has helped research community to design and develop algorithms for predicting such anomalies, either manually using statistical models [4, 12] or using machine learning techniques [3, 24]. System and infrastructure complexity grows every day, and hence it is obvious that the failures due to this also grows with undesirable outcomes that should be prevented; else it has huge impact on business outcome considering the downtime per hour. The byproduct of such outages has negative impact on the organization's brand with potential to impact revenue. It is for this very reason, building highly available, scalable, and secure systems has become ever more important. This opens the possibility of combining prediction with self-correction techniques to avoid crashes or malfunctions. The contributions of this article are:

- Develop a proactive failure prediction capability using **Long Short-term Memory (LSTM)** networks,
- After predicting the failure initiate **Uniform Resource Locator (URL)** prediction technique to construct the new or predicted URL, and
- Invoke state-of-the-art deep-learning-based automated information extraction framework to extract information from the new URL location [25, 26].

The rest of the article is organized as follows: Section 2 discusses related research work in the area of failure prediction in websites. The problem definition and proposed framework is explained in Section 3. Section 4 presents result from real world cases in the retail domain. The article is concluded in Section 5.

2 RELATED WORK

2.1 Failure Prediction

Infrastructure failures can occur due to human errors, software or hardware failures. There are many prediction techniques with capabilities, metrics, and costs. Predicting server crashes using machine learning techniques allow prediction of continuous (and not discrete values) [3]. Decision tree algorithm is one of the options to model anomalies, which is contrary to predicting infrastructure crashes in web applications using machine learning technique [4]. Emergence of clustering and virtualization technologies have alleviated such crashes [13]. The key is to understand the distinction between fault versus error versus bug [6]. Table 1 shows a list of failure types and corresponding algorithms used for predicting such failures. The section below delves in to providing details of these failure types:

- **Code failure:** Failures occur in software programs due to inadequate testing or validation [16]. A new machine learning-based failure prediction technique is implemented to predict failure in python code with recall 0.64 and false-positive rate 0.23 and compares with prior research on finding failures in C++/Java programs with similar precision 0.71 and false-positive rate 0.25 [37]. In this method, Naïve bayes classifier retrieves a list of python files from the code repository and builds an abstract syntax tree for each file and finally traversing the tree to find nodes representing source code units [37].

Table 1. Summary of Website Failure Types and Prediction Algorithms

Failure type	Algorithm
Server crash	Machine learning
Software programs	Naïve bayes classifier
Storage systems	Support vector machine (SVM) and back propagation neural network
Content change	Dynamic time warping (DTW)
Memory avoidance	OS-based approach
Website links	Markov chains
Website failures	Proposed approach

- **Drive failure:** Predicting an impending drive failure is performed by comparing the thresholding **algorithm with failure detection rates (FAR)** of 50% to 60% using **backpropagation (BP)** neural network model and improved **support vector machine (SVM)** algorithm [20, 38]. LIBSVM is used to implement the SVM model with a very large number of drives to validate the model and results (FAR @0:03% with SVM and BP neural network model detection rate >95%) show that the prediction accuracy of both models to be far higher than previous works [9]. Predicting content change is another area where an expert prediction system is developed to predict content such as degree and relationship among predicted pages, relation to other pages, and similarity in the types of changes [27]. Natural choices of relationship types (e.g., distance in the web graph, similarity in content, similarity in temporal change pattern) assess relative contributions over the page’s content by using cross correlation and **dynamic time warping (DTW)** algorithm [27]. Memory related issues in system infrastructure (e.g., operating system) using OS-based approach is implemented to proactively avoid memory failures using prediction techniques [1]. In this method, memory failure patterns are analyzed from the system log files by using failure patterns to predict likely memory failures. Performance monitoring and failure prediction method in optical networks based on machine learning is another technique for failure prediction, where SVM and **double exponential smoothing (DES)** are the algorithms used to predict the risk of an equipment failure with prediction accuracy of 95% [36].
- **Website link failure:** An evaluation on probabilistic link prediction and path analysis using Markov chains allows the system to model URL access patterns dynamically based on the URL’s previous state [28]. Experimental results indicate that web server requests can be predicted correctly, using the highest probability state, over 50% of the time, links predictions at almost 60% of the time. The algorithm is also validated with many other use cases such as tour generation, thus, proving that Markov chains are useful for web link sequence modeling and path analysis. Insufficient training data due to website visit is a limitation given its statistical nature and dimensionality factor limiting the approach to scale to very large number of websites [28].

A summary of learnings from the failure prediction systems is as follows:

- When modeling any variable of interest as a function of time it is natural to consider the past observed values as a basic means of prediction
- When predicting whether a page will change is performed based on the change frequency for the page observed over some past historical window
- Improved prediction can be performed by measuring degree of change that a page is undergoing, and the relationship among these changes

- For a page that experiences vastly dissimilar changes on a regular basis, the changes may be considered more significant (since the content change implies a need for reindexing)
- Natural choices of relationship types include: distance in the web graph, similarity in content, and similarity in temporal change pattern
- Failure prediction in software systems is in existence for many years in the areas of detecting server crashes, memory leaks, content changes, and so on.
- Machine learning techniques emerged to improve the prediction rate (minimize failure variance) by enhancing statistical algorithms such as SVM, Naïve Bayes Classifier, Markov chains, and so on.
- Proactive failure prediction in the context of web information extraction is fairly recent and the proposed failure prediction framework including construction of predicted URL using LSTM is a state-of-the-art, and
- Information extraction framework should recognize different web page structures and act on this knowledge to modify the information extraction techniques employed. The framework should be customizable for a variety of domains and data object types.

2.2 Data Extraction: Background and Evolution

Web data extraction has evolved over the last three decades from statistical to more advanced machine learning methods and techniques. It has many synonyms such as web scraping, web crawling, data extraction, information extraction, and so on. For consistency in this study the authors use web data extraction including web page navigation, data extraction, enrichment and visualization [31]. With rapid developments in web applications technologies extracting data from web became increasingly challenging. Traditional methods such as **document object model (DOM)** trees, text density, tag ratios, visual strategies, and fuzzy algorithms are used for data extraction from web page [23]. Due to the complexity involved in the web page structure designing a single system or method is extremely challenging. The traditional web extraction was done with manually programming wrappers or frameworks using Perl or Python programming languages for every single web page, which required the user to understand each web page information or dataset to extract data [34]. The programmer needed substantial programming language experience to write wrappers or tools such as TSIMMIS [19], WebOQL [32], Minerva [5], and XWrap [21] to extract data within a single web page. The challenges for the user was to navigate to each web page within a website, write wrapper code and then create a batch file that can run to scrape the entire website and extract data page by page. This is a very labor intensive and cost ineffective method and does not scale beyond one website or domain. Few of these tools are briefly explained for better understanding as follows:

- TSIMMIS [17] is a framework for manually building wrapper code. The wrapper takes input parameter defined by the developer, with data of located on a page and defines the output to be packaged in to an object model.
- Minerva [34] is a declarative grammar-based technique with ability to handle exceptions or error conditions, written in specific language called EDITOR to use more advanced programming functions native to Minerva development environment.
- WebOQL [5] is a functional query language to extract semi structured data in web page. It is similar to “DOM” tree structure-based method, where the data is labeled with attributes such as tag, source and text.
- W4F (Wysiwyg web wrapper factory) [29] is a Java tool kit to generate web wrappers by retrieving, extracting and mapping layers. Data is retrieved, fed in to HTML parser to construct parser tree following DOM method. User developed extraction rules are applied on

Table 2. Comparison of Traditional, Machine Learning-, and Deep Learning-based Web Data Extraction Tools

Tool/Model	Extraction rule	Techniques/Algorithms
TSIMMIS	Wrapper-based	Traditional/ Statistical
WebOQL	Tag tree	Traditional/ Statistical
WHISK	Regular expression	Machine learning (supervised)
RAPIER	Logic rules	Machine learning (supervised)
SRV	Logic rules	Machine learning (supervised)
SoftMealy	Regular expression	Machine learning (supervised)
DEPTA	Tag tree	Machine learning (unsupervised)
Trinity	Regular expression	Machine learning (unsupervised)
DeLA	Regular expression	Machine learning (unsupervised)
OLERA	Regular expression	Machine learning (semi-supervised)
R-CNN	Object detection	Dataset:ImageNet Classification:BinarySVM
FAST R-CNN	Object detection	Dataset:ImageNet Classification:BinarySVM RoI Pooling
FASTER R-CNN	Object detection	Dataset:ImageNet Classification:BinarySVM RoI Pooling, RPN
MASK R-CNN	Object detection	Dataset:ImageNet Classification:BinarySVM RoI Pooling, RPN and RoIAlign (pixel level segmentation)
RetinaNet	Object detection	ResNet and FPN(Feature Pyramid Network)
Singleshot Detector (SSD)	Object detection	single deep neural network and feed forward convolutional network
Histogram of Oriented Gradients (HOG)	Object detection	Detection window and region of interest (RoI)
Region-Fully Convolutional Network (R-FCN)	Object detection	Convolutional and RoI
Spatial Pyramid Pooling (SPP-net)	Object detection	Pyramid pooling
You Only Look Once (Yolo)	Object detection	Classification and regression, deformable parts model (DPM) and R-CNN

parser tree for data extraction and stored in W4f proprietary format and finally exported to application using mapping rules defined by the user.

- XWrap [21] encodes hypothetical structure of the web page in to rule-based declarative language done by analyzing the structure in the first phase and then generating XML template in second phase. XWrap is classified as a supervised learning technique, where in there is no need to write rules, since extraction rules are based on DOM tree path and requires user to understand HTML parse tree, tags, rows and columns in a table thus demands user expertise on both programming and domain.

The traditional data extraction techniques along with extraction rules are shown in Table 2.

2.3 Adaptive Data Extraction

Dynamic changes in websites make information extraction from websites difficult in real life application scenarios. There are also challenges with navigating websites that have password protected features, cookies, non-HTML data formats, and so on [7]. Adaptive information extraction framework was developed to handle dynamic changes in websites [18]. Internet contains large amount

of information hidden in the web pages including noisy content (e.g., advertisements, page settings, navigation buttons, etc.) and extracting information is extremely challenging. **You Only Look Once (Yolo)** and LSTM models have been developed to filter such noisy content and extract hidden information from the web page and store it in the database [22]. Information extraction techniques using web wrappers continue to mature with the evolution of machine learning and deep-learning techniques. A new method Fast (region-convolutional neural network) R-CNN is implemented to learn wrappers that can extract data from previously unseen templates [33]. This does not need any site-specific initialization and extracts information from a single web page. A general site independent web wrapper is developed, though its hypothesis is yet to be proved with empirical results [33]. A new wrapper framework is proposed that is adaptive to dynamic changes in website layout by using script edit model [11]. In this method, when there is change in website layout, the extraction model trains the new wrapper to automatically extract information with improved accuracy.

The limitations and challenges in adaptive information extraction framework are as follows:

- Commonly used position or structure-based web scraping tools need to be manually reconfigured as soon as the structure of the web page changes
- Contemporary and machine learning-based extraction frameworks usually rely on extraction rules or wrappers tailored to a particular information source [8]. Wrappers are specific to a given website and are tightly linked to the markup and structure of provider pages
- Most automatic information extraction framework can only cope with a limited set of document formats and do not adapt well to changes in document structure; HTML documents containing interesting data must be located
- Data of interest must be located within the Web page and rules that can be used to reliably extract the data must be created
- Mechanism used to create data extraction rules must either be sufficiently general or be easy to implement so that data can be extracted from the wide variety of page formats available on the web, and
- Information extraction framework must be able to cope with changes to web page structure, since web content providers frequently change the configuration and content of their pages.

3 PROBLEM DEFINITION AND PROPOSED SOLUTION

This section outlines the problem in building an end-to-end automated web information extraction framework shown in Figure 1 with the ability not only to predict website failures but also to construct the new URL location of the web page with product information and enable auto extraction of information. Figure 1 shows proposed automated web information extraction process including before and after a failure occurs. Section 3.1 outlines the problem in failure prediction in websites and Section 3.2 outlines the adaptive capability using a deep-learning method [26] to enable an automated information extraction from the webpage.

3.1 Failure Prediction in Website

The goal is to extract relevant product information attributes (e.g., “price,” “author,” etc.) referred as “region” and product page referred as “record” (e.g., “book”) and its corresponding location is referred to as “Target URL” denoted by U_t and it has a “peer” or “child” relationship with its parent URL referred as “Main URL” (denoted by U_p). In a real-world scenario, URL of the web page containing “record” or “region” change dynamically due to business requirements and therefore the location of “Target URL.” This relationship between “Main URL” and “Target URL” in a website

Fig. 1. Automated information extraction without failure and with proactive failure prediction and sample retail website with product attributes.

Fig. 2. “Main URL” and “Target URL” relationship.

is shown in Figures 2(a) and 2(b), where $U_t \subset U_p$ and is represented as $U = (U_p, U_t)$. The interaction between U_p and U_t is defined in terms of neighborhood and the relationship is defined as follows:

- 1 : “Main URL” and “Target URL” (peer or “child”) have peer relation i.e., both URLs are at the same level in the hierarchy;
- 2 : “Target URL” is parent of the “Main URL”;
- 3 : “Target URL” is child of the “Main URL”;

- 4 : “Target URL” becomes child of “Main URL,” which is child of a another “Target URL”;
 5 : “Target URL” becomes child of another “Main URL.”

The “Target URL” set is defined as $U = [u_1, u_2, \dots, u_l]$, where U_l is either peer or parent “Main URL” and “Target URL” are shown in Figures 2(a) and 2(b). The neighborhood of “Target URL” and “Main URL” is defined in Equation (1):

$$\mathcal{N}_l = [q | d_{ql} < \delta], \quad (1)$$

where δ is the threshold and d_{ql} is the distance between URL q (denoted as “Target URL”) to URL l (denoted as “Main URL”). If this distance is less than the threshold δ , then both q and l are considered to be neighbors. If the value of d_{ql} is beyond the threshold δ , then q and l are not neighbors. This is mathematically represented as follows:

$$\mathcal{N}_l = \begin{cases} 1(\text{Neighborhood Exists}), & \text{if } d_{ql} < \delta, \\ 0(\text{NoNeighborhood}), & \text{Otherwise.} \end{cases} \quad (2)$$

If “Neighborhood Exists,” then $E \exists ! U$, and the failure code (also denoted as E) is defined as follows:

$$E = \begin{cases} 1(\text{NochangeinURL}), & \text{if Success,} \\ 2(\text{NewURLlocation}), & \text{otherwise,} \\ 3(\text{Srvvertimeout}), \\ 4(\text{Outofmemory}), \\ 5(\text{Invalidlogin}), \\ 6(\text{Websitedown}). \end{cases} \quad (3)$$

The failure codes are based on literature in Section 2. The objective of this article is to predict the possibility of occurrence of $E = 2$ relevant to this study. Given it is difficult to predict the future, hence to test the LSTM algorithm’s accuracy, 80% of U is randomly singled out as a training set (U_{train}), the remaining 20% are used as test set to be predicted and no information in this set is allowed to be used for prediction ($y_{train}, y_{test} = 20\%$ of U). The problem definition is to find out if the proposed framework is able to predict a failure code based on the neighborhood prediction approach and the degree of accuracy measured by **mean-square error (mse)** will indicate performance of the algorithm. The proposed framework for proactive failure prediction is implemented in Algorithm 1 as follows:

- Preprocess data with Keras and NumPy,
- Create training and test dataset,
- Scale the data (to avoid overfitting) with MinMaxScaler on training/test dataset and normalize data range from 0 to 1,
- Pass 50 days of data (50,1) to LSTM model,
- Add Dense layer, which predicts 1 value,
- Predict output with test dataset, and
- Provide 10 days of failure code data and predict the 11th day failure code.

The proactive failure prediction algorithm is then integrated with the automated information extraction algorithm to build an end-to-end extraction framework shown in Algorithm 2 to extract product information by launching the “Target URL.” Website layout changes with every revamp. This will lead to the primary URL to change and hence causing the information extraction engine to break. In most cases, the new URL can be generated with static mapping as websites follow standard procedure of URL changes. For example, “v1/product/books/id” to “v2/product/books/id”

ALGORITHM 1: Failure prediction based on LSTM deep-learning network

```

procedure PREPROCESSERRORDATA(dataset)
  ▶ Input data pre-processing and training, test data creation
    define 80% [training_dataset]
    define 20% [test_dataset] excluding 50 data points
  end procedure
procedure CREATEERRORDATASET(u, y)
  ▶ Create error/failure data set
    define error_data:
  FOR i in range(50):
    u_train, y_train ← 80% [training_dataset]
    u_test, y_test ← 20% [test_dataset]
  end procedure
procedure TRAINANDPREDICTERRORCODE(u_train, u_test)
  ▶ Train LSTM model with dataset and predict the output
    select Sequential model, LSTM model
    add 'Dropout' and 'Dense'
    compile model (loss ← 'mean_squared_error', optimizer ← 'adam')
    fit model (u_train, y_train, epochs ← 100, batch_size ← 12)
    predictions = model.predict(u_test)
  end procedure

```

or “/product/search?type=articles&author=xyz” to “/product/articles?author=xyz”. It is easier to maintain static URL mapping for the pipeline to fallback. Whenever the algorithm predicts certain range of errors, the extraction engine script switches to the mapped URL for extracting information. This URL mapping is less overhead and more deterministic than predicting URL that may have high error rate due to randomness and historical data deficiency. Figure 3(a) shows the architecture of the proposed end-to-end web information extraction framework with proactive failure prediction and automated extraction capabilities using a deep-learning technique including subcomponents: (1) *Failure prediction*: The failure prediction model is implemented using LSTM networks shown in Algorithm 1, which has the ability to accurately predict the failure code, and (2) *Automated extraction*: This subcomponent handles automated extraction using Yolo and LSTM deep-learning network for image detection and image-to-text extraction in Algorithm 2 [26].

3.2 Information Extraction with Adaptive Capability

The deep-learning-based automated information extraction system referenced in this study solves the issue highlighted in Section 2.3 by eliminating the need for a “core extraction engine” making the automated web information extraction adaptive to dynamic changes in websites. It uses a deep-learning-based Yolo model for object or image detection and Tesseract LSTM deep-learning networks for the extraction of information from images or objects. As a result, the “core extraction engine” is eliminated by following a three-step process as follows:

- Step 1: Changes in website layout: Changes in website layout are introduced by simulating an experiment where the user accesses the specific product web page (i) without login (as a guest user) and (ii) with user login. With user login, the layout of the web page changes. In real-world scenarios, this change is a definite possibility due to other business needs, such as promotions and discounts.
- Step 2: Auto-navigation: The image detection feature of the Yolo model enables auto-navigation to the product page (also called “records” [product] and “regions” [product detail])

(a) Proactive and automated web information extraction framework

(b) Experiment (offline) setup for failure prediction and automated information extraction

Fig. 3. Architecture and offline experimentation environment of the proposed web information extraction framework.

and demands extensive training of the model for high accuracy in locating records and regions.

- Step 3: Adaptive and automated web information extraction: Using object detection-based deep-learning techniques such as Yolo and image to text extraction using Tesseract LSTM eliminates the core extraction engine; thus, the system demonstrates adaptability to dynamic changes in website layout.

The scope of this study aims to understand and apply the deep-learning-based Yolo model to enable the web data extraction system to demonstrate adaptive capability in the event of website layout changes. Once the Yolo is sufficiently trained with product images and the loss function is optimized, the proposed adaptive capability in the end-to-end web data extraction system is implemented in Algorithm 2 by implementing the steps as follows:

- Object detection: Yolo is used for object or image detection. Twenty-four CNN layers and two fully connected network layers are used to extract features from images and then predict bounding box output coordinates. Yolo CNN, which can identify multiple text objects on a web page image, is used. To identify text objects, Yolo DarkNet-19 architecture is considered. The DarkNet architecture is selected due to its lower processing requirement compared with other architectures such as ImageNet and GoogleNet.
- Information extraction: Once, the output image or product detail image is detected, Tesseract is used to extract the text. Tesseract is an optical character recognition open-source system that uses AI for text search and its image recognition; it also uses a two step approach that initiates adaptive recognition. In turn, Tesseract uses LSTM for text recognition. To predict text accurately from web page images, a stepwise strategy is used to identify the text objects on the page using the Yolo model. Once the bounding boxes with text are detected, the text is recognized. Several techniques can recognize text. A pretrained Tesseract neural nets LSTM

engine, which is an OCR engine, is used. One advantage of the Tesseract engine is its support for many languages. Yolo detects the required text regions and crops them out from the image. Later, these regions are passed one by one to Tesseract, which reads them and returns the corresponding text. In the experiment, sixteen images are used to validate multiple cases. The pipeline predicts different numbers of text objects for retail website, extracting the “book title,” “author,” and “price” from the products page in retail website.

Region-based CNN (R-CNN) techniques use regions to localize objects within an image, and do not consider the entire image, but only the parts of the images with a high chance of containing an object. By contrast, the Yolo framework handles object detection by taking an entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes [15]. Hence, the speed at which it processes (45 frames per second) coupled with the ability to understand generalized object representation provides the framework a unique advantage over the R-CNN family of object detection techniques [26]. Recently, Yolo has evolved to be one of the best algorithms for object detection with a comparatively similar performance to R-CNN algorithms. Hence, the proposed web data extraction system uses the Yolo architecture and an algorithm for object detection. Tesseract LSTM is an OCR technique used for extracting text from images [10]. It uses CNNs to recognize images by using recurrent neural networks [35] and LSTM. The input image is processed in boxes line by line and inserted into the LSTM model; then, the output is generated in the form of a text or Excel file. The proposed web data extraction system uses the Tesseract LSTM architecture and algorithm for image-to-text extraction. The section below explains the input data setup process, which includes two steps: (i) information gathering and (ii) training.

- **Information gathering:** Gathering a high quality dataset: Image files are crucial to training deep-learning systems and considerable amount of time is spent to create 100+ image files across retail and non-retail websites including single and multiple product specifications. The Yolo model is trained to detect single product (e.g., book) in the retail domain. After informations are gathered, bounding boxes around the objects are drawn. Eighty percent of the information set is then labeled and associated with corresponding object classes and validation; testing phases use 10% of the information set. The DarkNet expects the same information set for training and uses 80% of the total information set for training and the remaining 20% for testing.
- **Training:** Training the model starts with pretraining the weights of the DarkNet and fine-tuning the model by changing the weights of the last two layers. This procedure allows maintaining the same feature extraction layers and retraining only the decision part, thus reducing training time. With the scope of research in consideration, the information set is relatively smaller. The training, validation, and test sets are composed of 1,000 images at a ratio of 80:10:10. The information set is created by capturing images of product pages from different retail websites and then creating bounding boxes and label for each image in the information set. Yolo identifies the top bounding boxes per image based on the intersection over union measure and the confidence score of the bounding box for each text object class. In this study, three different classes of text objects (book title, author, and retail price) for identification are chosen. The model is trained for 30 epochs and the learning rate, weight decay, and momentum values are similar to the values used for training Yolo DarkNet-19 on the **common objects in context (COCO)** and **visual object challenge (VOC)** information set. Pascal VOC is an XML file, where one file is created for each image in the information set, whereas COCO is in the form of a JSON file, which has one file for the entire information set and the testing and validation information set.

ALGORITHM 2: Automated information extraction based on Yolo and LSTM model

```

1: procedure E(v)alPipeline(imagelist):
2:   Tesseract  $\leftarrow$  InitializeTesseractmodelwithlstm
3:   YoloModel  $\leftarrow$  InitializetrainedYoloModelwithweights
    $\triangleright$  Predict bounding box around images
4:   FOR-each imagename in imagenameslist:
5:     predictedboundingboxes, predictedclasses  $\leftarrow$  YoloModel.predict(image)
6:     imagewithboxesclasses  $\leftarrow$  drawboxesclasses(image)
7:     FOR-each index, box in enumerate( predictedboundingboxes) :
8:       predictedlocation  $\leftarrow$  image.crop(box)
9:       predictedclassname  $\leftarrow$  classes[predictedclasses]
10:      predictedtext  $\leftarrow$  Tesseract.imagetostring(predictedlocation)
11:     END For-each
12:   END For-each
13: end procedure
14: procedure U(R)LMapping
    $\triangleright$  Launch the URL, capture image and save
15:   browse(URL $\leftarrow$ First.html)
    $\triangleright$  Extract the product data using Yolo and Tesseract LSTM-based model
16:   call evalPipeline(parameters)
    $\triangleright$  Assume First.html URL has been changed and results in error
    $\triangleright$  Launch the target URL based on static mapping for product and extract the product data
17:   imagepath  $\leftarrow$  'Second.jpg'
18:   browse(URL  $\leftarrow$  Second.html)
19:   call evalPipeline(parameters)
20: end procedure

```

4 RESULTS AND DISCUSSION

A two-step approach is implemented in the experiment as follows: (1) using Algorithm 1 to proactively predict failure in website and (2) using Yolo and LSTM (Procedure “evalPipelin” in Algorithm 2) to auto extract information from the newly mapped web page (Procedure “URLMap” in Algorithm 2). This is performed in a simulated environment where a static website is hosted on an Apache server shown in Figure 3(b). The reason is that, in a real-world scenario, there is wait time for the failure to occur, which is an unpredictable event with no definitive timeline.

4.1 Evaluation Metrics

- **Mean-average precision (mAP) error:** For all the experiments in Section 4, mAP of a set of objects or images is defined in Equation (4) and used to measure performance of models performing information retrieval and object detection. In Equation (4), I_k = number of images in the data set and AvgP = Average Precision (AvgP) for a given image, I_k . For a given image, I_1 , AvgP is calculated and mean of the all AvgP scores gives mAP indicating the model's performance in identifying the image or object. In Equation (4), R_n and P_n are the precision and recall at the n th threshold. mAP is the mean of AvgP over all the images or objects explained in Section 4.3:

$$\text{AvgP} = \sum_n (R_n - R_{n-1}) P_n. \quad (4)$$

- **Information retrieval accuracy:** Information retrieval accuracy is measured by precision, recall, and accuracy for detection of objects (e.g., product) and for character retrieval from the

Table 3. Input Dataset

Date	Failure code	Error	URL	Time	Frequency	Weight	Distance
30 Sep	0	0	999	0	4	1.1	0
30 Sep	1	1	1999	0.04	4	1.5	0.5
30 Sep	2	1	1999	0.08	4	2.5	1.0
30 Sep	3	1	1999	13	4	3.5	1.5
30 Sep	4	1	1999	17	3	4.5	2.0
30 Sep	5	1	1999	21	3	5.5	2.5
30 Sep	6	1	1999	25	3	10.5	3.0

object. In Equation (5), precision, recall, and accuracy are calculated where PRA = Precision, Recall, and Accuracy and O_{rel} = Relevant images or objects, O_{ret} = Retrieved images or objects, C_{rel} = Relevant characters, C_{ret} = Retrieved characters and explained in Section 4.3:

$$PRA = \begin{cases} (|O_{rel} \cap O_{ret}|)/O_{rel}, Precision(Objectdetection), \\ (|O_{rel} \cap O_{ret}|)/O_{ret}, Recall(Objectdetection), \\ (|C_{rel} \cap C_{ret}|)/C_{rel}, Precision(Characterextraction), \\ (|C_{rel} \cap C_{ret}|)/C_{ret}, Recall(Characterextraction), \\ (|O_{ret} + C_{ret}|)/O_{rel} + C_{rel}, Accuracy. \end{cases} \quad (5)$$

- Model loss: YOLO uses binary cross-entropy loss function for each label, shown in Equation (6), reducing computation complexity and predicts objectness score for each bounding box using logistic regression. Loss is cost function value for training data shown in Figure 5(a):

$$loss = \begin{cases} -\log Object_{predicted}, & \text{if } Object_{actual} == 1, \\ -\log(1 - Object_{predicted}), & \text{otherwise.} \end{cases} \quad (6)$$

- Mean-square error: From the target failure codes within specific range, error rate is calculated in Equation (7). It means that if target failure code is 2, then the predicted code is 1.9; if failure code is 3 and the predicted code is 2.9, then the average of the sum of the difference between the predicted and the actual data point is calculated as shown in Figure 5(b), where $F_{measured}$ is the actual failure code and $F_{expected}$ = expected failure code:

$$mse = \left(\frac{1}{n}\right) \sum_{k=1}^n (F_{measured} - F_{predicted}). \quad (7)$$

4.2 Experiment-1 (Proactive Failure Prediction)

Input dataset is shown in Table 3 with each input data field described in Table 4 comprising of 1,467 rows, 8 fields, 80% training dataset, 20% test dataset, 6 failure codes (Equation (3)), and 3 months dataset. The experimental setup with algorithm, tools, parameters and technology details is shown in Table 5 along with the proposed failure prediction model summary including parameters and hyperparameters.

The proposed model is trained with 80% training dataset and 100 epochs with each epoch executed in less than 1 second (48 ms). Subsequently, the model predicts failure code with 20% test dataset (not seen or trained earlier) and then predicts failure code on the (n+1)th day where n = day of occurrence of failure. The actual and failure code is plotted for the dataset till 1,200th data row (of 1,467 data rows/ 61 days) from input data file taken as training dataset shown in Figure 4(a) and with last 12 days data (300 data rows) as testing dataset to predict the 1,468th data row/13th (n+1) and overall, 61st day failure code as shown in Figure 4(b).

Table 4. Input Data Field Details

Input variables	Details
Date	Website running date
Failure code	Value between 0 and 6 where 0 refers to success and no failure while the remaining values are assigned specific failure type referenced in Equation (3) mapped to actual retail web page URL
URL	Location of the product specific web page encoded in to specific codes (“1999”—URL with failure, “999”—URL with no failure/Success)
Time	Time log of website in running state
Frequency	Number of occurrence of failure type in a single day
Error weightage	Numerical value assigned to each failure code depending on the severity of the failure type
Distance	Distance between “Main URL” and “Target URL” (denoted as d_{ql})

Table 5. Model Parameters for Failure Prediction Experiment

Environment setup	Details	
Epoch	100	
Activation function	Softsign	
Loss function	Mean-square error	
Memory avoidance	Dense	
Optimizer	Adam	
Training dataset	80%	
Test dataset	20%	
Language	NumPy	
Failure prediction	LSTM	
Hardware	1 TB HDD, 16 GB GPU	
Layer (type)	Output shape	Value
Model	Sequential_11	
LSTM_33	(None,50,96)	37,632
LSTM_33	(None,50,96)	37,632
Dropout_33	(None,50,96)	0
LSTM_34 (Dropout)	(None,50,96)	74,112
Dropout_34	(None,50,96)	0
LSTM_35	(None,96)	74,112
Dropout_35	(None,96)	0
Dense_11	(None,1)	97
Total parameters	185,953	
Trainable parameters	185,953	
Non-trainable parameters	0	

The loss from the model during the training phase is showing a declining trend with each epoch indicating that the model is improving the prediction shown in Figure 5(a), where the x-axis is the number of epochs and y-axis is the %loss. The failure variance or mean-square error is 0.6% to 1.0% as shown in Figure 5(b), indicating that the prediction accuracy is between 99% and 99.4%. The x-axis is the number of epochs and the y-axis is % failure rate.

4.3 Experiment-2 (URL Mapping and Automated Information Extraction)

After predicting the failure code in experiment-1 (Section 4.2), an automated information extraction framework using Yolo and LTSM deep-learning technique is invoked to learn the newly

Fig. 4. Actual and 61st day predicted failure code with last 12 days as test dataset.

Fig. 5. Model loss and failure prediction rate.

Fig. 6. Static product page without and with failure condition with associated URL change.

mapped URL and extract relevant product information. The base and predicted product page URL of the website are shown in Figures 6(a) and 6(b). Without failure condition, using Yolo and LSTM deep-learning technique, the model detects the product page as an image or object and draws bounding boxes around the product attributes that need to be extracted as shown in Figure 7(a).

The framework uses Tesseract image-to-text technique to extract product information from the detected images with bounding box shown in Figure 7(a) and the corresponding bounding box coordinates shown in Figure 8(a). After failure occurs, the automated information extraction

(a) Product page attributes with bounding box: before failure condition

(b) Product page attributes with bounding box: after failure condition

Fig. 7. Static website product page without and with failure condition with associated URL change.

<pre> <annotation> <filename>Second.jpg</filename> - <object> <name>Author</name> <pose>Unspecified</pose> <truncated>0</truncated> <difficult>0</difficult> - <bndbox> <xmin>136</xmin> <ymin>339</ymin> <xmax>168</xmax> <ymax>514</ymax> </bndbox> </object> - <object> <name>Price</name> <pose>Unspecified</pose> <truncated>0</truncated> <difficult>0</difficult> - <bndbox> <xmin>274</xmin> <ymin>357</ymin> <xmax>319</xmax> <ymax>408</ymax> </bndbox> </object> - <object> <name>Title</name> <pose>Unspecified</pose> <truncated>0</truncated> <difficult>0</difficult> - <bndbox> <xmin>92</xmin> <ymin>356</ymin> <xmax>126</xmax> <ymax>939</ymax> </bndbox> </object> </pre>	<pre> - <annotation> <filename>Second.jpg</filename> - <object> <name>Author</name> <pose>Unspecified</pose> <truncated>0</truncated> <difficult>0</difficult> - <bndbox> <xmin>136</xmin> <ymin>339</ymin> <xmax>168</xmax> <ymax>514</ymax> </bndbox> </object> - <object> <name>Price</name> <pose>Unspecified</pose> <truncated>0</truncated> <difficult>0</difficult> - <bndbox> <xmin>274</xmin> <ymin>357</ymin> <xmax>319</xmax> <ymax>408</ymax> </bndbox> </object> - <object> <name>Title</name> <pose>Unspecified</pose> <truncated>0</truncated> <difficult>0</difficult> - <bndbox> <xmin>92</xmin> <ymin>356</ymin> <xmax>126</xmax> <ymax>939</ymax> </bndbox> </object> </pre>
(a) Product page bounding box coordinates: before failure condition	(b) Product page bounding box coordinates: after failure condition

Fig. 8. Static website product page without and with failure condition with associated URL change.

framework points to the newly mapped URL of the product page in the website as shown in Figure 6(b) and the corresponding bounding box coordinates shown in Figure 8(b). Subsequently the model detects the product page accurately with bounding boxes around specific product information as shown in Figure 7(b). The framework then uses Tesseract image-to-text technique to extract text from the detected images as shown in Figures 9(b) and 10. At record level, 11 of 12 total objects and 84 of 86 characters were detected correctly (i.e., 2 characters were not detected or detected wrongly), and hence leading to low precision shown in Table 6. In deep-learning technique, mAP is used as an evaluation metric to measure model performance in web information extraction, which in this experiment was at 74%. It is established in the experiment that the extracted information is correct as per the specific product details measured by precision, recall at the record level (extraction accuracy) at 47.8%, 91%, and precision, recall at region level (character

Table 6. Performance Measurement of the Proposed Web Information System

Parameters	Value
Mean-average precision (mAP)	74%
Precision, Recall (Object extraction)	48.8%, 97%
Precision, Recall (Character extraction)	49.5%, 99%
Overall extraction accuracy	96.9%
Execution time	48 ms
Failure prediction accuracy	99%
Mean-square error	1.0%

File Name:demo\url\input\First.jpg
Author:by Monisha Rajesh
Price:'486
Title:Around the World in 80 Trains: A 45,000-Mile Adventur

(a) Product page attribute information extracted: before failure condition

(b) Product page attributes with bounding box: after failure condition

Fig. 9. Product page attribute information extracted: before and after failure condition.

```
Anaconda Prompt (tf1)
(tf1) C:\git\yolo>python url_based_pred.py -i demo\url\input\ -o demo\url\output\
Using TensorFlow backend.
Evaluating :demo\url\input\First.jpg
file Name:First
Author 1.00 (339, 136) (514, 168)
Price 0.98 (357, 274) (408, 319)
Title 0.98 (356, 92) (939, 126)
by Monisha Rajesh
'486
Around the World in 80 Trains: A 45,000-Mile Adventur

DevTools listening on ws://127.0.0.1:57301/devtools/browser/09b1ed1f-1646-41ca-8c5d-b12c3795d4c8
Evaluating :demo\url\input\Second.jpg
file Name:Second
Author 1.00 (339, 136) (514, 168)
Price 0.98 (357, 274) (408, 319)
Title 0.98 (356, 92) (939, 126)
by Monisha Rajesh
'486
Around the World in 80 Trains: A 45,000-Mile Adventur
```

Fig. 10. Data extracted from the product page with and without failure condition.

extraction accuracy) at 49.4%, 97%, respectively, with an overall extraction accuracy of 96.9%. This is shown in Table 6.

Yolo and LSTM model execution time is higher in the initial run, but decreases in subsequent cycles, thus reducing the overall execution time. The model training time is a function of the system GPU, because it involves graphic image processing, which is computation intensive. The execution time starts from the initiation of the experiment (i.e., locating the product specification page) up to the extraction of data. The evaluation time is high in the first iteration due to the initial model training time, but decreases in subsequent iterations shown in Figure 11.

Fig. 11. Evaluation time (training, execution, and extraction) of the proposed framework.

(a) Product page with failure condition (user login)

(b) Bounding box around product page with failure condition (user login)

Fig. 12. Product page with failure condition demonstrating adaptive capability.

4.4 Experiment-3 (Adaptive and Automated Information Extraction)

After predicting the failure code in experiment-1 (Section 4.2), an automated information extraction framework using Yolo and LSTM deep-learning technique is invoked to learn the newly mapped URL and extract relevant product information. The predicted product page and corresponding URL of the website is shown in Figure 12(a). With failure condition, using Yolo and LSTM deep-learning technique, the model detects the product page as an image or object and draws bounding boxes around the product attributes that need to be extracted and the corresponding text extracted are shown in Figures 12(b) and 13.

4.5 Discussion and Future Scope

A combined framework to address proactive failure prediction and automated information extraction is a challenging area to solve. While research in application of traditional and machine learning-based predictive algorithms in areas such as server crash, software programs, memory utilization and website links, and so on, is performed extensively, more research is needed in predicting changes in website layout or content. Emerging technologies such as deep learning will help advance research in predicting such changes, while also constructing the new URL link as

File Name:C:\git\yolo\demo\temm_input\uc1-we-1.JPG Author:by Monisha Rajesh Price:394 Title:Around the World in 80 Trains: A 45,000-Mile Adventure

Fig. 13. Product information extracted from web page with failure condition.

outlined in Section 2.1. In automated information extraction, technology emerged rapidly in achieving more than 95% recall (information accuracy rate). However, the challenge is when modern techniques in failure predictions are applied to build a comprehensive end-to-end automated information extraction framework and deep-learning techniques will radically transform the process of automated web information extraction framework in the coming years.

5 CONCLUSION

A long short-term memory algorithm is implemented for proactive prediction of failures in websites, constructing the new URL or location of the specific product page, and allowing the creation of an end-to-end web information extraction framework. The proposed approach resulted in 99% success rate on failure prediction. Incorporating a deep-learning-based extraction framework with 94% information accuracy rate enabled us to build an end-to-end web information extraction framework. We recommend conducting the experiment in an online or live environment with websites in non-retail domain as a next step to make the proposed idea into a generic framework adaptable to dynamic changes in websites. Further research is recommended to validate the proposed failure prediction model against real-time environments.

REFERENCES

- [1] H. A. Carlos, C. Yoonho Park, Bryan S. Rosenburg, C. Cher, and K. Dong Ryu. 2014. A system software approach to proactive memory-error avoidance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*. 1291–1300. <https://doi.org/10.1109/BigData.2016.7840733> Retrieved from <https://dl.acm.org/doi/10.1109/SC.2014.63>.
- [2] E. Adar, J. Teevan, S. Dumais, and J. Elsas. 2009. The web changes everything: Understanding the dynamics of web content. *Proc. WSDM* (2009), 282–291. <https://doi.org/10.1145/1498759.1498837> Retrieved from <https://dl.acm.org/doi/10.1145/1498759.1498837>.
- [3] J. Alonso, J. L. Berral, R. Gavald, and J. Torres. 2009. Predicting web application crashes using machine learning. In *Proceedings of the 5th International Conference on Autonomic and Autonomous Systems (ICAS'09)*. <https://doi.org/10.1109/ICAS.2009.56> Retrieved from https://www.researchgate.net/publication/221039808_Predicting_Web_Server_Crashes_A_Case_Study_in_Comparing_Prediction_Algorithms.
- [4] J. Alonso, J. Torres, and R. Gavald. 2009. Predicting web server crashes: A case study in comparing prediction algorithms. In *Proceedings of the 5th International Conference on Autonomic and Autonomous Systems (ICAS'09)*. 264–269. <https://doi.org/10.1109/ICAS.2009.56> Retrieved from <https://ieeexplore.ieee.org/document/4976614>.
- [5] G. O. Arocena and A. O. Mendelzon. 1998. WebOQL: Restructuring documents, databases, and webs. In *Proceedings of the 14th IEEE International Conference on Data Engineering (ICDE'98)*. 24–33. <https://doi.org/10.1109/ICDE.1998.655754> Retrieved from <https://ieeexplore.ieee.org/document/655754>.
- [6] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Depend. Secure Comput.* 1, 1 (2004), 11–33. <https://doi.org/10.1109/TDSC.2004.2> Retrieved from <https://ieeexplore.ieee.org/document/1335465>.
- [7] R. Baumgartner, M. Ceresna, and G. Ledermuller. 2006. DeepWeb navigation in web data extraction. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Vienna. 698–703. <https://doi.org/10.1109/CIMCA.2005.1631550> Retrieved from <https://ieeexplore.ieee.org/document/1631550>.
- [8] I. Bifulco, S. Cirillo, C. Esposito, R. Guadagni, and G. Polese. 2021. An intelligent system for focused crawling from big data sources. *Expert Syst. Appl.* 184 (2021), 115560. <https://doi.org/10.1016/j.eswa.2021.115560> Retrieved from <https://www.sciencedirect.com/science/article/abs/pii/S0957417421009660>.

- [9] C.-C. Chang and C.-J. Lin. 2011. A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2, 3 (2011), 27:1–27:27. <https://doi.org/10.1145/1961189.1961199> Retrieved from <https://dl.acm.org/doi/10.1145/1961189.1961199>.
- [10] C. Wang and P. Wei. 2019. A novel web page text information extraction method. In *Proceedings of the IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC'19)*. pp. 2213–2218. <https://doi.org/10.1109/ITNEC.2019.8729329> Retrieved from <https://ieeexplore.ieee.org/document/8729329>.
- [11] D. Liu, L. Ma, and X. Liu. 2015. Research on adaptive wrapper in deep web data extraction. In *Proceedings of the Conference on Internet of Vehicles (IOV'15)*. 409–423. https://doi.org/10.1007/978-3-319-27293-1_36
- [12] E. Marcus and H. Stern. 2018. Machine learning approach using MLP and SVM algorithms for the fault prediction of a centrifugal pump in the oil and gas industry. *Int. J. Comput. Appl.* 180, 19 (2018), 0975–8887. <https://doi.org/10.3390/su12114776> Retrieved from <https://www.mdpi.com/2071-1050/12/11/4776>.
- [13] H. Galih, A. Azhari, and M. Khabib. 2003. *Blueprints for High Availability*. Wiley, New York, NY.
- [14] J. Gao, H. Wang, and H. Shen. 2020. Task failure prediction in cloud data centers using deep learning. *IEEE Trans. Serv. Comput.* (2020), 1111–1116. <https://doi.org/10.1109/TSC.2020.2993728> Retrieved from <https://ieeexplore.ieee.org/document/9006011>.
- [15] G. Huang, Z. Liu, and K. Weinberger. 2016. Densely connected convolutional networks. Retrieved from <https://arxiv.org/abs/1608.06993>.
- [16] M. Hajiaghayi and E. Vahedi. 2019. Code failure prediction and pattern extraction using LSTM networks. In *Proceedings of the IEEE 5th International Conference on Big Data Computing Service and Applications (BigDataService'19)*. 55–62. <https://doi.org/10.1109/BigDataService.2019.00014> Retrieved from <https://arxiv.org/abs/1812.05237>.
- [17] J. Hammer, J. McHugh, and H. Garcia-Molina. 1997. Semistructured data: The TSIMMIS experience. *Proceedings of the 1st East-European Symposium on Advances in Databases and Information Systems (ADBIS'97)*. <https://doi.org/10.14236/ewic/ADBIS1997.22> Retrieved from <https://www.scienceopen.com/hosted-document?doi=10.14236/ewic/ADBIS1997.22>.
- [18] M. Harish Rao and D. R. Sashikumar. 2016. A survey on automated web data extraction techniques for product specification from E-commerce web sites. *Int. J. Adv. Res. Comput. Sci. Softw. Eng.* 6, 8 (2016).
- [19] C. N. Hsu and M. T. Dung. 1998. Generating finite-state transducers for semi-structured data extraction from the web. *Info. Syst.* 23, 8 (1998), 521–538. Retrieved from <https://www.sciencedirect.com/science/article/abs/pii/S0306437998000271>.
- [20] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan. 2002. Improved disk-drive failure warnings. *IEEE Trans. Reliabil.* 51, 3 (2002), 350–357. <https://doi.org/10.1109/TR.2002.802886> Retrieved from <https://ieeexplore.ieee.org/document/1028408>.
- [21] L. Liu, C. Pu, and W. Han. 2000. XWRAP: An XML-enabled wrapper construction system for web information sources. *Proceedings of the 16th International Conference on Data Engineering*. 611–621. <https://doi.org/10.1109/ICDE.2000.839475> Retrieved from <https://ieeexplore.ieee.org/document/839475?arnumber=839475>.
- [22] S. Nagarajan and K. Perumal. 2017. A deep neural network for information extraction from web pages. In *Proceedings of the IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPSCSI'17)*. 918–922. <https://doi.org/10.1109/ICPSCSI.2017.8391845> Retrieved from <https://ieeexplore.ieee.org/document/8391845>.
- [23] N. Kushmerick and B. Thomas. 2003. Adaptive information extraction Core technologies for information agents, *Lecture Notes in Computer Science book series (LNCS)*, vol. 2586. https://doi.org/10.1007/3-540-36561-3_4 Retrieved from https://link.springer.com/chapter/10.1007/3-540-36561-3_4.
- [24] P. Francesco Orrù, A. Zoccheddu, L. Sassu, C. Mattia, R. Cozza, and S. Arena. 2020. Machine learning approach using MLP and SVM algorithms for the fault prediction of a centrifugal pump in the oil and gas industry. *Sustainability* 12, 4776 (2020). <https://doi.org/10.3390/su12114776> Retrieved from <https://www.mdpi.com/2071-1050/12/11/4776>.
- [25] S. K. Patnaik and C. N. Babu. 2021. Information retrieval from web with faster R-CNN deep learning networks: A new perspective. In *Proceedings of the 6th International Computer Science and Engineering Conference (UBMK'21)*. 61–66. <https://doi.org/10.1109/UBMK52708.2021.9558956> Retrieved from <https://ieeexplore.ieee.org/document/9558956>.
- [26] S. K. Patnaik, C. N. Babu, and M. Bhawe. 2021. Intelligent and adaptive web data extraction system using convolutional and long short-term memory deep learning networks. *IEEE Big Data Mining Anal.* 4 (2021), 279–297. <https://doi.org/10.26599/BDMA.2021.9020012> Retrieved from <https://ieeexplore.ieee.org/document/9523501/>.
- [27] K. Radinsky and Paul N. Bennett. 2012. Predicting content change on the web. In *Proceedings of the (WSDM'13)*. 415–424. <https://doi.org/10.1145/2433396.2433448> Retrieved from <https://dl.acm.org/doi/abs/10.1145/2433396.2433448>.
- [28] Ramesh R. Sarukkai. 2000. Link prediction and path analysis using Markov chains. *Elsevier J. Comput. Netw.* 33, 1–6 (2000), 377–386. [https://doi.org/10.1016/S1389-1286\(00\)00044-X](https://doi.org/10.1016/S1389-1286(00)00044-X) Retrieved from <https://www.sciencedirect.com/science/article/abs/pii/S138912860000044X>.
- [29] A. Sahuguet and A. Fabien. 2001. Building intelligent web applications using lightweight wrappers. *Data Knowl. Eng.* 36, 3 (2001), 283–316. [https://doi.org/10.1016/S0169-023X\(00\)00051-3](https://doi.org/10.1016/S0169-023X(00)00051-3).

- [30] F. Salfner, M. Lenk, and M. Malek. 2010. A survey of online failure prediction methods. *Comput. Surveys* 42, 3 (2010), 1–42. <https://doi.org/10.1145/1670679.1670680> Retrieved from <https://dl.acm.org/doi/10.1145/1670679.1670680>.
- [31] S. Li, C. Chen, K. Luo, and B. Song. 2019. Review of deep web data extraction. In *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI'19)*. 1068–1070. <https://doi.org/10.1109/SSCI44817.2019.9002877> Retrieved from <https://ieeexplore.ieee.org/document/9002877>.
- [32] S. Soderland. 1999. Learning information extraction rules for semi-structured and free text. *J. Mach. Learn.* (1999), 233–272. Retrieved from <https://link.springer.com/article/10.1023/A:1007562322031>.
- [33] T. Gogar, O. Hubacek, and J. Sedivy. 2016. Deep neural networks for web page information extraction. In *Proceedings of the Conference on Artificial Intelligence Applications and Innovations (AIAI'16)*. IFIP Advances in Information and Communication Technology, Springer, Cham 475 (2016), 154–163. https://doi.org/10.1007/978-3-319-44944-9_14 Retrieved from <https://hal.inria.fr/hal-01557648>.
- [34] V. Crescenzi, G. Mecca, and P. Merialdo. 2001. RoadRunner: Toward automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*. 109–118. <https://doi.org/10.1.1.28.9727> Retrieved from <http://vldb.org/conf/2001/P109.pdf>.
- [35] Liu Wei, Xiaofeng Meng, and Weiyi Meng. 2006. Vision-based web data records extraction. In *Proceedings of the 9th International Workshop on the Web and Databases (WebDB'06)*. Retrieved from https://www.researchgate.net/publication/221035622_Vision-based_Web_Data_Records_Extraction.
- [36] Z. Wang, M. Zhang, D. Wang, C. Song, M. Liu, J. Li, L. Lou, and Z. Liu. 2017. Failure prediction using machine learning and time series in optical network. *Optics Express* 25, 16 (2017), 18553–18565. <https://doi.org/10.1364/OE.25.018553> Retrieved from <https://www.osapublishing.org/oe/fulltext.cfm?uri=oe-25-16-18553&id=370117>.
- [37] Bartłomiej Wójcicki and Robert Dabrowski. 2018. Applying machine learning to software fault prediction. *Inform. Softw. Eng. J.* 12, 1 (2018), 199–216. <https://doi.org/10.5277/e-Inf180108> Retrieved from https://www.e-informatyka.pl/attach/e-Informatica_-_Volume_12/eInformatica2018Art8.pdf.
- [38] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma. 2013. Proactive drive failure prediction for large-scale storage systems. In *Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST'13)*. 1–5. <https://doi.org/10.1109/MSST.2013.6558427> Retrieved from <https://ieeexplore.ieee.org/document/6558427>.

Received May 2021; revised July 2021; accepted October 2021