Requirements Engineering (2021) 26:67–96
https://doi.org/10.1007/s00766-020-00332-2

ORIGINAL ARTICLE

Model elements identification using neural networks: a comprehensive study

Kaushik Madala[1] · Shraddha Piparia[1] · Eduardo Blanco[1] · Hyunsook Do[1] · Renee Bryce[1]

Received: 29 June 2019 / Accepted: 2 May 2020 / Published online: 29 May 2020

Abstract

Modeling of natural language requirements, especially for a large system, can take a significant amount of effort and time. Many automated model-driven approaches partially address this problem. However, the application of state-of-the-art neural network architectures to automated model element identification tasks has not been studied. In this paper, we perform an empirical study on automatic model elements identification for component state transition models from use case documents. We analyzed four different neural network architectures: feed forward neural network, convolutional neural network, recurrent neural network (RNN) with long short-term memory, and RNN with gated recurrent unit (GRU), and the trade-offs among them using six use case documents. We analyzed the effect of factors such as types of splitting, types of predictions, types of designs, and types of annotations on performance of neural networks. The results of neural networks on the test and unseen data showed that RNN with GRU is the most effective neural network architecture. However, the factors that result in effective predictions of neural networks are dependent on the type of the model element.

Keywords Component state transition diagrams · Requirements analysis · Automated requirements modeling · Neural networks · Sequence labeling · Empirical study

1 Introduction

Requirements analysis is one of the important software development processes because it can prevent defects from propagating to later phases of development. One of the widely used requirements analysis approaches is the model-driven methodology, which provides a clear understanding of the system as well as an easy transformation into different types of software artifacts. To date, many researchers [1–9] have proposed various methods for model-driven requirements analysis. For example, Piras et al. [7] proposed Agon framework, which uses a multi-layer meta-model that represents the knowledge of gamification to analyze acceptance requirements. Another example is the technique proposed by Li et al. [9], which uses goal models to create three-layer framework for analyzing security in socio-technical systems. However, one major limitation of most of these approaches is that the models are created manually. To thoroughly create the model of a system manually, a significant amount of effort is required [10]. Furthermore, the overhead involving model creation might delay iterations in agile development, thereby limiting its use.

To address this limitation, some researchers [11–16] have proposed several methods to create models automatically from the requirements documents. For example, Robeer et al. [17] proposed Visual Narrator, a tool that generates conceptual models from user stories using a set of rules generated by combining heuristics proposed by various researchers [18–21]. While these approaches can aid in identifying model elements automatically, their usage is limited by the structure of sentences they process. For example, if the heuristics are created based on the assumption that a sentence is written in active voice, it cannot identify model

✉ Kaushik Madala
kaushikmadala@my.unt.edu

Shraddha Piparia
shraddhapiparia@my.unt.edu

Eduardo Blanco
Eduardo.Blanco@unt.edu

Hyunsook Do
hyunsook.do@unt.edu

Renee Bryce
Renee.Bryce@unt.edu

[1] University of North Texas, 3940 North Elm Street, Denton, TX 76207, USA

elements correctly for the sentences in passive voice. With recent breakthroughs in machine learning, deep neural networks have been adapted to perform various tasks in software engineering [9, 22, 23]. For example, in our previous work [22], we performed automatic model element identification to identify model elements of component state transition diagrams from unstructured and complex natural language requirements using recurrent neural network (RNN) with a long short-term memory (LSTM) architecture. While the goal was to reduce the effort needed to identify model elements in the long run, we learned from the previous study that the complexity of sentences affects the classifiers' performance and that we need more data to address this issue.

Based on the lessons learned and observations drawn from our previous study, we conducted a pilot study [14] to analyze whether there are trade-offs among neural networks in identifying model elements and whether their performance is affected by the style of the requirements documents (e.g., active voice vs. passive voice, and use cases vs. user stories). The results showed that there are trade-offs among neural networks and that their performance is affected when applied to unseen documents. We also observed that (as expected) simpler sentence representations used in structures such as use cases improve the performance of neural networks compared to complex natural language requirements when trained on a small amount of data. However, the validity of the results needs to be further investigated as it is possible that the neural networks in the pilot study suffered from the overfitting problem [24]. Moreover, we did not study the factors that can affect neural networks' performance for identifying model elements.

To address these shortcomings, in this paper, we perform an extensive empirical study to investigate the effectiveness of four types of neural networks (feed forward (FF) neural network [25, 26], convolutional neural network (CNN) [27, 28], recurrent neural network with long short-term memory (LSTM) [29, 30], and recurrent neural network with gated recurrent unit (GRU) [31, 32]) to identify model elements of component state transition diagrams automatically. We selected these neural network architectures (NNAs) for the following reasons:

1. Traditional natural language processing techniques [33, 34] do not consider the similarity between words based on their meaning, whereas neural networks consider similarity between words by using word embeddings.
2. Neural networks provide state-of-the-art results in tasks such as named entity recognition and relation extraction [35–37], which are similar to tasks involved in automated model generation from the requirements.
3. To understand whether neural networks can be useful for automated model element identification, we start with the four basic NNAs before applying advanced NNAs because it requires a lot of time and resources in applying the advanced NNAs. If the results with the basic NNAs are proven to be effective, then we plan to use the advanced NNAs such as XLNet [38] and BERT [39] for our task.

As part of our study, we also analyze the minimum percentage of an unseen document that needs to be annotated to produce a high F_1-measure on the rest of the unseen document. Using the state-of-the-art neural networks for automatic model element identification, we attempt to have a clear understanding on whether requirements documents have different characteristics compared to the text generally used in natural language processing research area. Moreover, studying different types of neural network architectures and their ability to learn and predict aids researchers in understanding how to adapt neural networks and data science to fit automatic model generation. The automatic model generation consists of two parts: (1) model element identification (2) automatic relation extraction among identified model elements. The first part, *model element identification*, refers to the task of extraction of model elements from natural language requirements. In this step, we try to identify components, states, actors, and transition conditions from the requirements. Once the model elements are extracted, it is necessary to identify the relations among them in order to understand the associated states of each component, and which actors and transition conditions affect the behavior change of a component. Moreover, without extracting relations among these model elements, we cannot know whether any details regarding the states of a component are omitted or if some information is missing in the requirements when creating a model. The second part of model generation, *automatic relation extraction*, is concerned with identifying the relationships among model elements and there by generating the model. In this paper, we focus only on the first part, i.e., model element identification. The results of our study show that model elements predictions using neural networks varied on the factors such as the way the data is split, the experimental design chosen to perform text classification, and the way annotations are done. Our results also indicate RNN with LSTM and RNN with GRU are top performing architectures for identifying model elements, but the results varied on different conditions.

The rest of paper is organized as follows. Section 2 discusses related work on automated model element identification. Section 3 details the research questions we investigate and the motivation behind the questions. Section 4 briefly introduces different neural networks, word embeddings, conditional random fields, and natural language annotation. Section 5 provides details about our empirical study and its results, and Sect. 6 discusses the threats to validity of our study. Section 7 presents our results, their implications, and limitations of our work. We conclude and discuss our future work in Sect. 8.

2 Related work

To date, many researchers [16, 40–42] have worked on automatic creation of models from natural language requirements. The proposed approaches range from using heuristic-based natural language processing techniques [17, 41, 43] to machine learning techniques [14, 22].

For example, Yue et al. [40] proposed an automated framework called aToucan, which uses rule sets generated based on a fixed set of sentence patterns for automatically creating unified modeling language (UML) models such as class diagrams, sequence diagrams, and activity diagrams from the descriptions of use cases that follow restricted structure. Their framework outperformed commercial activity diagram generation tools such as Visual Paradigm [44], Ravenflow [45], and CaseComplete [46]. Another approach [41] proposed AnModeler, a tool for automatically generating class and sequence diagrams from use case specification using features of natural language such as parts-of-speech tags [47] and type dependencies [48], which are generated using the Stanford's core natural language processing tool [49]. However, the identification of elements is restricted to sentences that follow Hornby's verb patterns [50]. The authors conducted a case study with two industrial experts and found that their tool gives results close to the diagrams drawn by experts. Further, Pudlitz et al. [51] proposed an approach that uses a combination of bidirectional LSTM (BiLSTM) and CNN architectures to extract states of system from requirements written in natural language. Sleimi et al. [52] also proposed a technique that automatically extracts semantic legal metadata from legal requirements using natural language processing that uses constituency and dependency parsing.

Many other researchers have proposed approaches [15, 16, 42, 53, 54] and tools [55, 56] to automatically create UML models from structured or semi-structured requirements. For example, Śmialek and Straszak [55] proposed ReDSeeDS tool, which uses requirements specification language (RSL) and provides an environment for generating component, class, and interaction diagrams. Their tool is also used to generate code structure from RSL specification. Another example is the mechanism proposed by Gutiérrez et al. [42] to automatically generate activity diagrams from use cases by using a meta-model of functional requirements and QVT-Relational language [57]. While their main goal is to eventually generate test cases, they found their method provides a highly usable representation, albeit with limited flexibility. Further, an approach proposed by Smialek et al. [53] automatically creates sequence diagrams from use case descriptions written with a restricted natural language. While the tool they developed received positive feedback from the industry, the tool needs the engineers to first manually annotate the elements of the sentences such as subjects, predicates, and verbs.

Unlike aforementioned approaches, some researchers [11, 20] have tried to automate the generation of conceptual models from natural language requirements. For example, Lucassen et al. [11] extended the Visual Narrator tool discussed in Sect. 1, which automatically generates conceptual models from user stories by combining the tool with other natural language processing tools such as AQUSA [58] and Interactive Narrator [59]. The results of the extended version of Visual Narrator shows an improvement in the accuracy compared to the original version of the tool. Another technique proposed by Vidya Sagar and Abirami [20] generates conceptual models from requirements by extracting natural language features such as parts-of-speech tags and dependencies among words. The extracted features are used to identify the relation types among entities. The entities and the relations among them are used to create and visualize the models.

A few other researchers [43, 60] have proposed approaches to automatically extract goals from textual descriptions. For example, Zhang et al. [60] proposed an approach to extract service goals from textual descriptions of services. The goal description is automatically created in two steps, both of which use dependency-based information among words to create an initial description and its refinement. Further, an approach, GaiusT, proposed by Zeni et al. [43] uses a semantic annotation mechanism to extract concepts from legal documents. These extracted concepts are used as goals for the software system that aid in satisfying legal the compliance standards. The results of the tool show it performs on par with human annotators. Chen et al. [61] proposed a tool called T-Star, which automatically generates iStar models from textual descriptions. The tool requires users to write descriptions by following semi-structured templates.

While all these approaches propose various mechanisms to automate models, most of them did not study the usage of state-of-the-art neural networks for automated model creation. As mentioned in Sect. 1, our earlier studies performed automated model element identification using deep neural networks from unstructured natural language requirements [22] and use cases [14], respectively. However, we did not study the effect of different factors on the performance of different neural network architectures to predict model elements. In our paper, we address this limitation by analyzing trade-offs among neural network architectures considering different factors that could affect their effectiveness.

3 Research questions

Our central research question in this study is: *Which neural network architectures are effective in performing automated model element identification and what are the factors affecting their performance?* To study our central research question in detail, we break it into a set of several research questions. The overview of research questions along with the relationships among them is shown in Fig. 1. The highlighted content in the black rectangle at the top of the figure is our central research question. The blue rectangles denote the major research questions in our study, and the green rectangles show the sub-questions for research question 2. The rest of this section discusses each RQ in detail.

RQ1 What is the effectiveness of the neural network architectures on test and unseen data?

Analyzing RQ1 will help requirements engineers to understand which architecture is effective in identifying model elements. However, a neural network architecture's performance can be affected by various factors such as the characteristics of its input, values of hyper parameters, and experimental design. This motivates our second research question.

RQ2 What are the factors that affect the performance of neural networks when predicting the model elements?

There is a wide variety of factors that can affect neural networks' performance, but it is not feasible to investigate all the factors, thus we focus on the following four factors that are our primary interests: (1) nature of splitting of data, i.e., how we partition data into training, validation, and test, (2) experimental design, i.e., how we plan to approach the prediction problem (e.g., sequence labeling vs. sentence classification), (3) dependencies among model elements, and (4) the nature of human annotations.

We considered partitioning of data as a part of our study (similar to [62]) to analyze how we can leverage benefits from model elements identification when there is only a small amount of data available in form of requirements documents.

RQ2.1 How does the partitioning of data into training, validation, and test affect the performance of neural networks?

RQ2.2 How does experimental design affect the performance of neural networks?

RQ2.3 Do neural networks predict model elements better when we use a single classifier for all model elements or separate classifiers for each type of model elements?

RQ2.4 Does the way we perform human annotations (e.g., identifying only the keyword of the model element vs. identifying the entire phrase representing the model element) affect the predictions of model elements by neural networks?

By analyzing RQ1 and RQ2, we can choose the best neural network architecture with factors that can enhance its performance. However, we might want to choose a combination of neural networks with the corresponding factors that boost the performance of neural networks rather than considering a single neural network architecture alone to predict all types of model elements. This motivates our third research question.

RQ3 Are there trade-offs among neural network architectures for automatic model element identification task? If so, what can we infer?

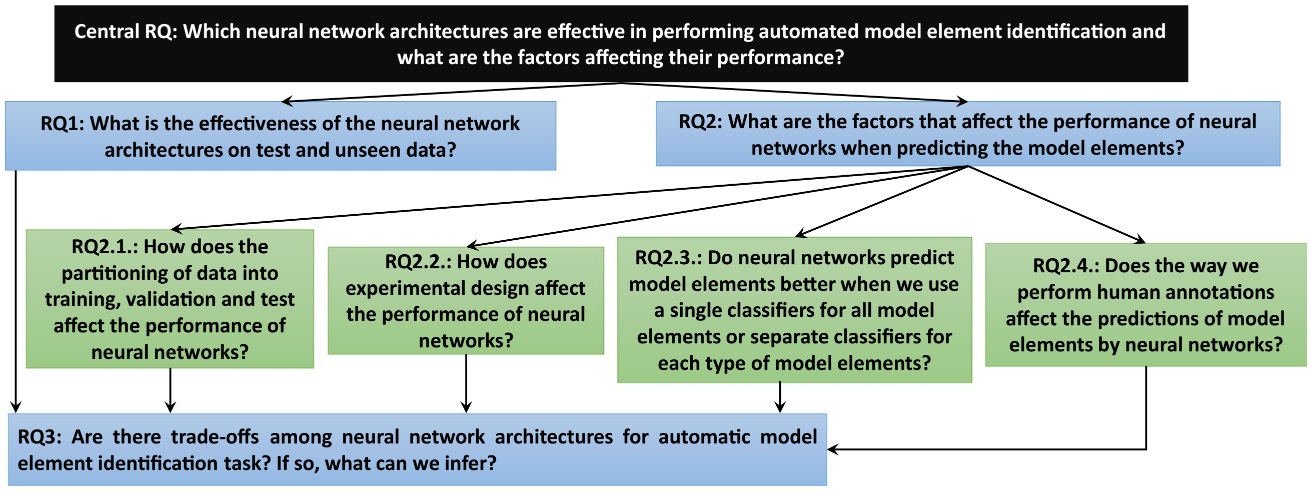

Central RQ: Which neural network architectures are effective in performing automated model element identification and what are the factors affecting their performance?
RQ1: What is the effectiveness of the neural network architectures on test and unseen data?
RQ2: What are the factors that affect the performance of neural networks when predicting the model elements?
RQ2.1.: How does the partitioning of data into training, validation and test affect the performance of neural networks?
RQ2.2.: How does experimental design affect the performance of neural networks?
RQ2.3.: Do neural networks predict model elements better when we use a single classifiers for all model elements or separate classifiers for each type of model elements?
RQ2.4.: Does the way we perform human annotations affect the predictions of model elements by neural networks?
RQ3: Are there trade-offs among neural network architectures for automatic model element identification task? If so, what can we infer?

Fig. 1 Relationships among research questions

4 Background

This section provides background information about component state transition diagrams, natural language annotation process, different neural network architectures, and word embeddings.

4.1 Component state transition diagrams

A component state transition (CST) diagram [8, 63] is used to provide an abstract representation of behaviors of components in a system. CST diagram requires the system to be represented by finite components along with their corresponding states and the transition conditions. A component is defined as part of the system, and each component can have multiple states. The states of components can be changed because of actions known as transition conditions. We identify three major elements of a CST diagram: components, their states, and transition conditions to formulate component transition rules, which can be provided as input to requirements analysis approaches such as causal component model (CCM) [63].

There are two types of transition conditions: *system* transition conditions and *environmental* transition conditions. A *system* transition condition refers to components' states that result in the state change of other components. An *environmental* transition condition is an action performed by humans or other external entities that are not part of the system. To understand the complete behavior of the system, it is important to identify both transition conditions. System transition conditions can be identified easily, however, environmental transition conditions are difficult to identify because often environmental transition conditions are not explicitly defined in the requirements. Hence, in addition to three major model elements, we considered another model element, *actor*, in our approach to identify environmental transition conditions. An *actor* is any external agent who interacts with the system to perform an action.

Figure 2 illustrates an example of a CST diagram for the component ultrasonic sensor. The sensor has two states ('no detection' and 'detected'). There are two transition conditions: 'Switch(off)' and 'Switch(on) AND Obstacle(present).' 'Switch(off)' and 'Switch(on) are examples of system transition conditions and 'Obstacle(present)' is an example of environmental transition conditions. Because the obstacle is an external entity, it is considered as an actor.

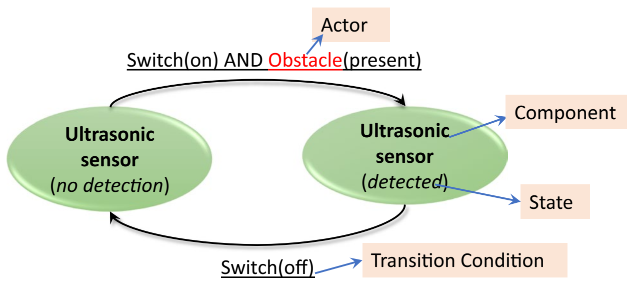

Actor
Switch(on) AND Obstacle(present)
Ultrasonic sensor (no detection)
Ultrasonic sensor (detected)
Component
State
Switch(off)
Transition Condition

Fig. 2 Example of a component state transition diagram for ultrasonic sensor

4.2 Natural language annotation process

Annotation process [64] involves two or more people labeling the text manually to reduce the possible errors or biases that can be occurred when it is done by a single annotator. The annotation process also aids in resolving conflicting opinions about the labeled entities. At least two annotators are required to perform annotation. For a given dataset and classification task, guidelines are created for labeling. The annotators label the entities (e.g., *sentences* in a sentence classification task, and *words* in sequence labeling) by following the guidelines. Once the annotators complete a preliminary sample of data, they check their inter-annotator agreement, i.e., a measure which gives an estimated number of instances the annotators agree and disagree. Examples of measures used for inter-annotator agreement are Cohen's Kappa [65, 66] and F_1-measure [67]. Based on the agreement, the guidelines are refined. The process is repeated until a reliable agreement is reached or until the agreement changes no more. Using the finalized guidelines, the rest of the documents are annotated and the corresponding inter-annotator agreement is calculated. Researchers in the natural language processing and computational linguistics communities [68–70] often consider the inter-annotator agreement value as a rough upper bound for a machine learning algorithm to perform on par with humans. Therefore, often a classifier's performance needs to be compared with the inter-annotator agreement to assess the performance of classifiers relative to human performance.

Once the annotations are done, the annotators finalize the labeling of instances. This process is called adjudication. During adjudication, if a disagreement or conflicting opinion rises among annotators on an instance, then it is resolved by consulting an additional annotator. After resolving the disagreements, the labels of data are finalized resulting in the gold standard. We train and evaluate classifiers with these labels in the gold standard.

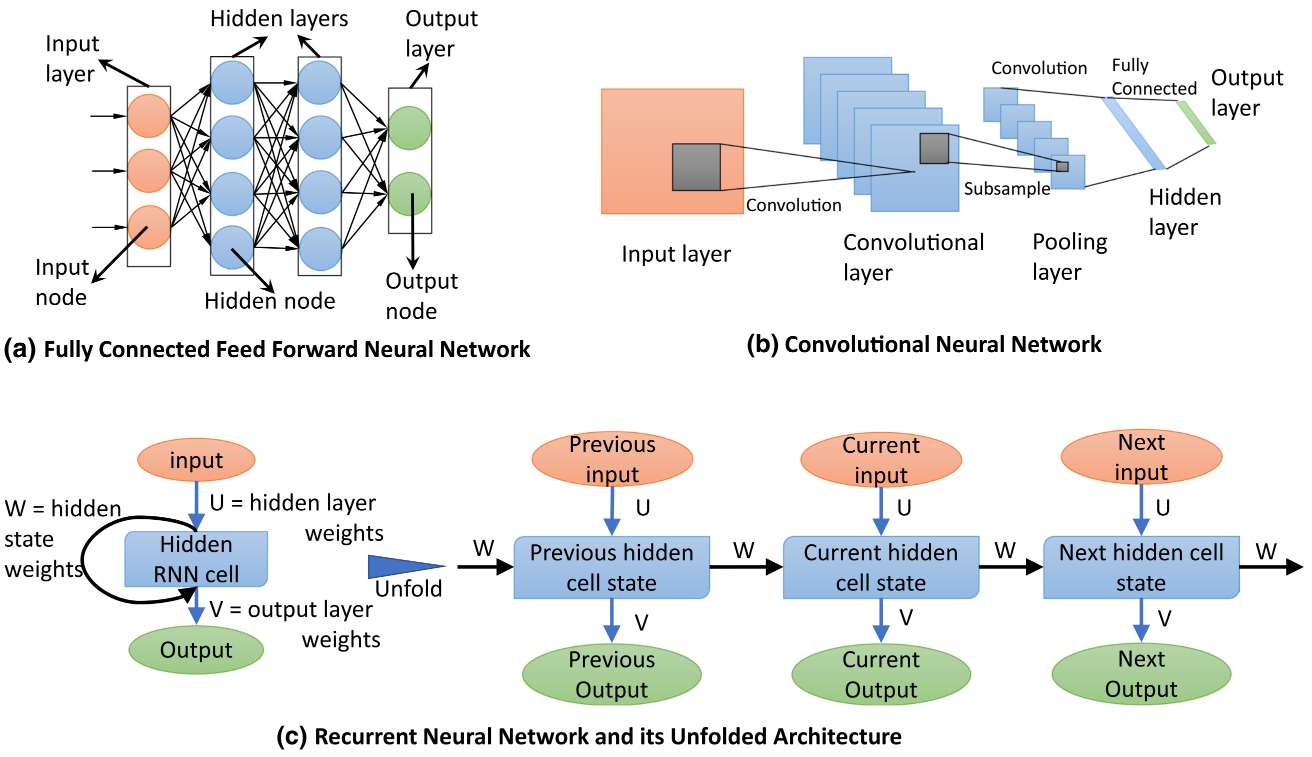


Fig. 3 Neural network architectures

4.3 Fully connected feed forward neural networks

Figure 3 illustrates different neural network architectures. The architecture in Fig. 3a represents the fully connected feed forward (FF) neural network [25, 26]. As shown in the figure, the architecture has an input layer, which has input nodes that are used to feed input into the neural network; an output layer, which has output nodes that have information on the output generated from the neural network; and hidden layers (one or more) with hidden nodes that carry out the internal operations which aid in predicting the labels of the input. The figure shows that every node from the preceding layer is connected to every node from the next layer (thus the name fully connected feed forward). This implies each node affects the learning based on the weights of their associations with other nodes.

4.4 Convolutional neural networks

Convolutional neural networks (CNN) [27, 28] produce state-of-the-art results in image recognition tasks [71, 72]. They also proved useful in natural language processing tasks such as sequence classification [73, 74]. The architecture of CNN is shown in Fig. 3b. As shown in the figure, CNN has input, hidden, and output layers similar to FF network discussed earlier. In addition, CNN also has one or more convolutional layers, which move a weight matrix along the input and perform a convolution operation (multiplying input values with corresponding weights and adding the resulting values). The parameters from a convolutional layer are further reduced by a pooling layer, which moves a window along the output from a convolutional layer and usually replaces the window with the maximum value of all the values within the window (thus, denoted as sub-sampling). The values from the pooling layer can be given as input to additional convolutional layers or to fully connected hidden layers.

4.5 Recurrent neural networks

Recurrent neural networks (RNN) [75] are the state-of-the-art neural networks for natural language processing tasks [76–78]. Unlike feed forward neural networks, RNNs consider previous inputs when learning the current inputs. The architecture of RNN is shown in Fig. 3c. The left-hand side of the figure shows the folded representation of RNN, and the right-hand side shows the unfolded presentation of RNN. The hidden layer has a recurrent input that considers previous information contextually to predict the current outcome. The bottom-left corner of the figure shows the folded RNN cell with multiple weights such as U, V, W, which are weights of a hidden layer, output layer, and hidden state, respectively. These weights are tuned during the training of a classifier. The two major state-of-the-art RNN architectures are long short-term memory (LSTM) [29, 30] and gated recurrent unit (GRU) [31, 32].

Long Short-Term Memory (LSTM) Long short-term memory (LSTM) [29, 30] is a RNN architecture that was proposed to overcome the vanishing gradient descent problem in RNN. The vanishing gradient descent problem results in short remembrance of previously fed inputs. LSTM tracks long-term dependencies by regulating the amount of new cell state to withhold, amount of existing memory to forget, and amount of cell state that needs to be exposed to next layers in the network.

Gated Recurrent Unit (GRU) Gated recurrent unit (GRU) [31, 32] is an RNN architecture similar to LSTM, but GRU does not control the amount of cell state, which is given as input to next layers in the network.

4.6 Word embeddings

Word embeddings [79] are vector representations of natural language words, in which words that are similar have smaller distance among them, whereas words that are dissimilar are located farther. They are proposed to avoid the curse of dimensionality and a highly sparse data problem faced when trying to convert text into vectors. For example, if a data set has a million unique words, traditionally these words are fed into neural networks by converting each word into a one-hot vector of length one million, where the index corresponding to the word has a value of one and the rest of indexes have a value of zero. Word embeddings reduce this representation into a smaller (e.g., 50, 100, 200, and 300) dimensional vector. Word embeddings are generated by training models on a large amount of corpora such as Wikipedia [80] or common crawl [81]. These models can be used as pre-trained models when only a small amount of data is available for the task at hand. There are multiple pre-trained word embedding models such as GloVe [82], word2vec [79], Dependency [83], Fasttext [84, 85], and ELMo [86]. In our study, we used GloVe embeddings to evaluate all four neural network architectures.

4.7 BIO tags

BIO (beginning, inside, and outside) tags [87, 88] are widely used to label a series of words in sequence labeling tasks such as named entity recognition. In the example shown in Fig. 4 in which we label the name of an organization, we can see that the phrase 'University of North Texas' is an organization name. The tag 'B' allocated to 'University' indicates that it is the beginning of the name of the organization. The tag 'I,' which indicates that the word is inside the name of an organization, is allocated to all other words in the name such as 'of,' 'North,' and 'Texas.' Any word that is not part of the name of an organization is considered as outside, which we represent with 'O' tag. In our study, we used BIO tags to label the model elements.

4.8 Conditional random fields

The statistical method conditional random field (CRF) [33, 34] models the decision boundary between different classes in the data. CRF uses probability distribution to estimate the label of the current word given the information such as current word, its features, and surrounding words, their features and labels. CRFs define a feature function, which is used to represent the characteristics of words in a sentence. For example, let us consider the application of parts-of-speech tagging by CRF. The feature function is denoted as $f(X, i, l_{i-1}, l_i)$, which provides a value of 0 or 1. The symbol 'X' in the feature function represents input vectors, 'i' represents the current position of word in the sentence, 'l_{i-1}' represents the label of preceding word, and l_i' represents the label of current word. For instance, in the case of parts-of-speech, a noun can follow an adjective. Thus, the value of feature function $f(X, i, l_{i-1}, l_i)$ is 1, where l_{i-1} is adjective, and l_i is noun. The value of 1 implies that a noun can be preceded by adjective. CRF also has a set of weights similar to neural networks, which represents the weight on each feature function. The weights are randomly initialized, learned, and iteratively updated as part of the learning process using gradient descent until the weights converge. Because CRF is the state-of-the-art technique for the sequence labeling task prior to advent of deep neural networks, we compared the performance of neural networks in identifying model elements with CRF. Our goal is to analyze if neural networks' performance is still better than CRF's when a small amount of data is used.

5 Study

To conduct an empirical study, we implemented neural network classifiers using Python language and keras library [89]. We conducted our experiments in a system with the Ubuntu operating system and 16 GB RAM. The research questions we considered for our study are detailed in Sect. 3. Further details of the study are as follows.

Fig. 4 BIO tags example: finding the named entity of an organization

Example:	University	of	North	Texas	is	a	good	university	.
Labels of organization:	B	I	I	I	O	O	O	O	O

5.1 Objects of analysis

Table 1 summarizes the use case documents we work with in this article. The table shows that the size of documents varies from 2 use cases to 21 use cases. The first four documents in the table are from the usecasedoc data set [90]. The fifth document is written as part of the academic course materials, and the sixth document is an industrial use case document. We used documents 1–5 to create training, validation, and test data, and document 6 as unseen data.

5.2 Variables

5.2.1 Independent variables

The list of independent variables for each research question is shown in Table 2. RQ1 manipulates one independent variable, neural network architecture, RQs 2.1–2.4 manipulate two different independent variables, and RQ3 manipulates all these independent variables. We describe each of the independent variables.

Neural network architecture In our study, we used four neural network architectures that were explained in Sect. 4. Table 3 explains each of them.

Splitting method Splitting refers to the process of dividing the data into training, validation, and test data. We considered two types of splitting method: intra- and inter-document splitting. Table 4 provides details about these methods, and Fig. 5 illustrates their process with 5 sample documents. Note that we use training data to train the neural network classifier, validation data for hyperparameter tuning, and test data to evaluate the performance of the classifier. We chose the two splitting methods as mentioned in Table 4 because we want to analyze how we can use automation effectively when we have a limited amount of data available. The intra-document splitting mentioned in Table 4 aims at identifying

Table 1 List of use case documents used

ID	Description	No. of use cases
1	A use case document on personalized health informatics (PHI) compliant system	21
2	A use case document on online shopping system	4
3	A use case document on automated guided vehicle system	2
4	A use case document on emergency monitoring system	4
5	A use case document on ambulance dispatch system [91]	10
6	A part of use case document on an system, which offers assisted mobility [92]	11

Table 2 Independent variables

Research question	Independent variable
RQ1	Neural network architecture
RQ2.1.	Neural network architecture and splitting method
RQ2.2.	Neural network architecture and text classification method
RQ2.3.	Neural network architecture and prediction method
RQ2.4.	Neural network architecture and annotation approach
RQ3	All the above-mentioned variables

Table 3 Types of neural network architectures being studied

#	Name	Mnemonic	Description
1	Fully connected feed forward Neural network [25, 26]	FF	A neural network architecture in which every node is connected to node in the next layer.
2	Convolutional neural network [27, 28]	CNN	A feed forward neural network where some fully connected Hidden layers are replaced by convolution layers.
3	Recurrent neural network with long short-term memory [29, 30]	RNN with LSTM	A neural network that considers previous input and keeps Track of long-term dependencies.
4	Recurrent neural network with gated recurrent unit [31, 32]	RNN with GRU	A neural network architecture similar to LSTM, but does Not use a memory unit to control the information as LSTM.

Table 4 Splitting methods

#	Name	Description
1	Intra-document splitting (withindoc)	As shown in Fig. 5a, this method divides each document in the data set in the ratio of 70:10:20 to generate training, validation, and test data for that document. Once the documents are divided, all the training splits of the documents are grouped to form the training data for the neural network. The other splits are also grouped into validation and test data, respectively, following a similar approach.
2	Inter-document splitting (between-doc)	As illustrated in Fig. 5b), this method does not divide each document into parts, rather it groups different documents into non-overlapping sets of ratio similar to 70:10:20 to create training, validation, and test data.

how predictions will turn out if we have annotations from a part of the document for which we are trying to identify model elements automatically. This scenario, however, is not ideal in practice because it requires engineers to annotate part of the a requirements document prior to automatically labeling the rest of the document. The latter inter-document splitting mentioned in Table 4 is more realistic as it predicts on a completely new document.

Text classification method Experimental design refers to how we planned to approach the model elements prediction problem. In this study, we considered three types of design: *wordlevel*, *seq2seq*, and *seqlvlword*. Their details are presented in Table 5.

Figure 6 shows examples of these classification methods. The example sentence is ‘When a user presses switch, the motor runs.’ For *wordlevel* shown in Fig. 6a, when the word ‘user’ is given as input, the neural network predicts the label of ‘user.’ We can see that no preceding or following word is fed into the neural network. In the case of *seq2seq* in Fig. 6b, when the entire example sentence is fed into the neural network, we obtain an output of the same length of the input sentence with a label for each word in the sentence. The first output label ‘O’ is the label of word ‘When,’ the second output label ‘O’ is the label of ‘a,’ and so on. The example of *seqlvlword* is shown in Fig. 6c. In seqlvlword design, for a given sentence, we generate sentences by highlighting each token in the input sentence individually by adding ‘##’ symbols before and after the token. For the example sentence with 10 tokens in Fig. 6, we generate 10 highlighted sentences. We add the symbol ‘##’ before and after a word to highlight the word in the sentence. An example of a sentence with a highlighted word (‘user’) is shown at the top of Fig. 6. When this sentence with ‘user’ highlighted is given as input to the neural network, the label of the word ‘user’ will be predicted by the neural network such as label ‘B’ as shown in Fig. 6.

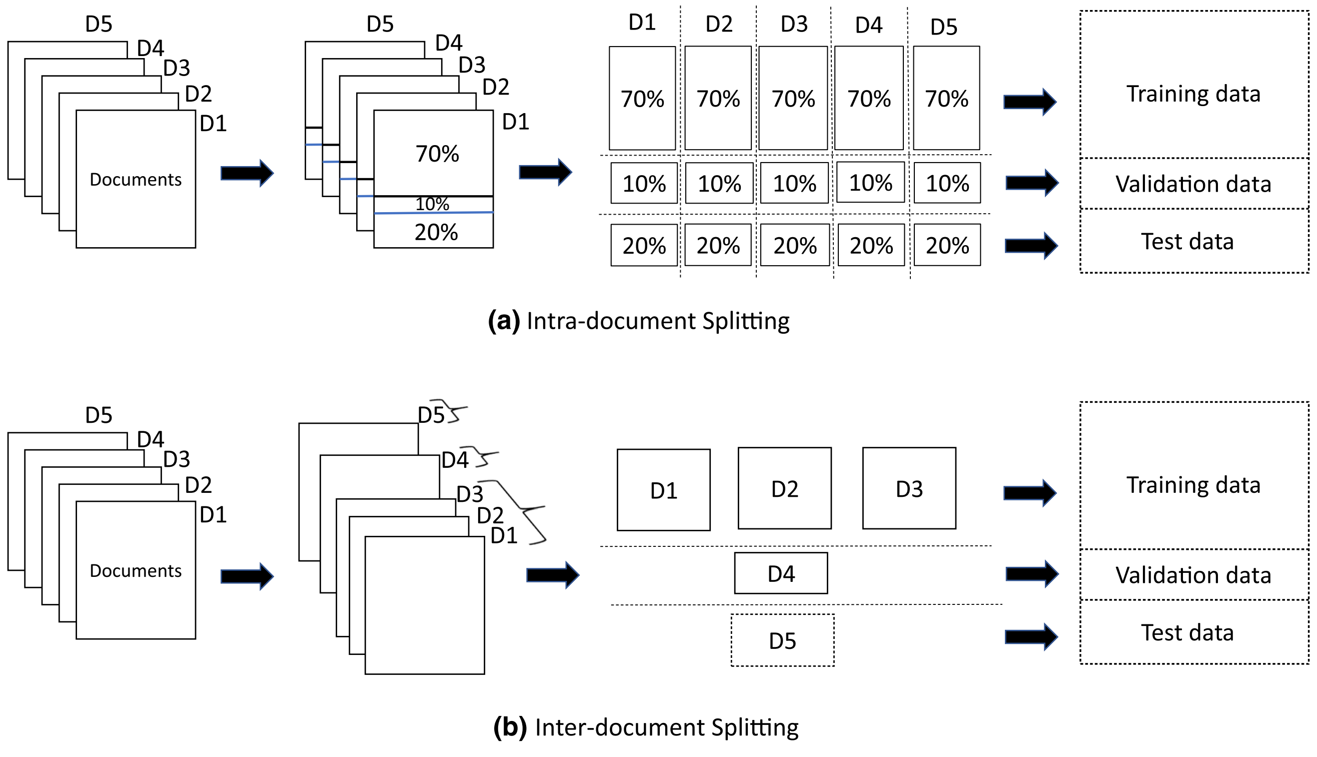
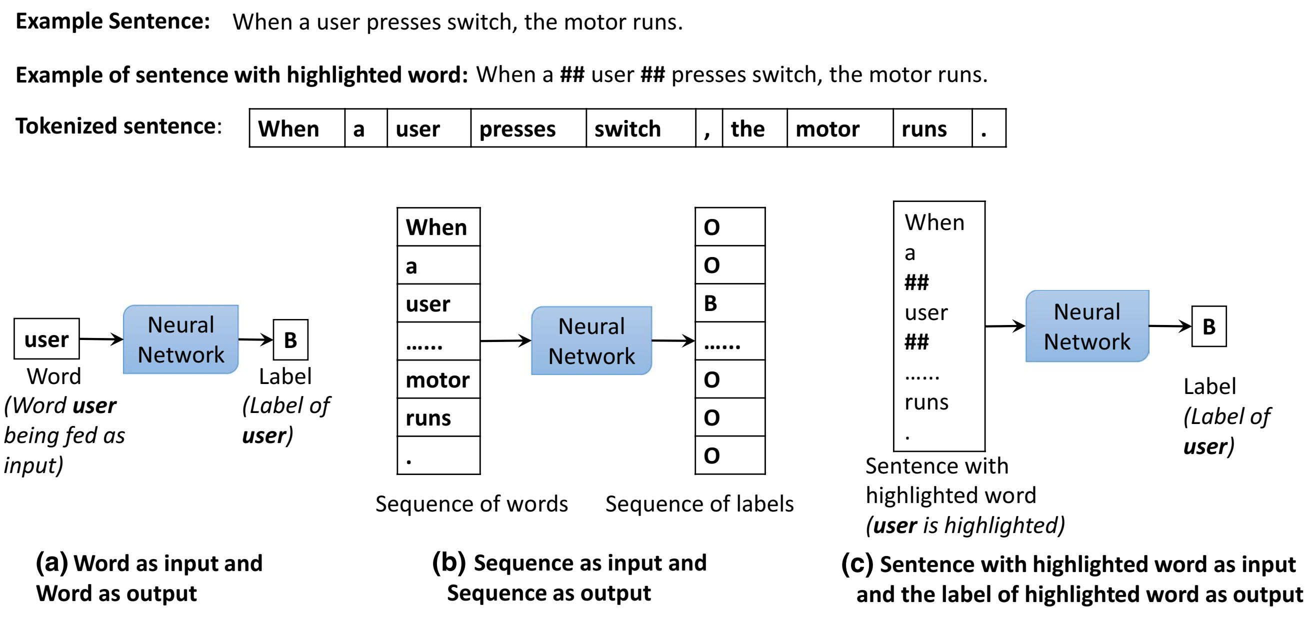


Fig. 5 Splitting methods

Table 5 Text classification methods

#	Name	Description
1	Word as input and word as output (wordlevel)	In this design, the neural network is trained on each word without any context. The network takes a word as input and generates its label, respectively. We consider this as baseline to our approach. Because this design does not consider the context of the words, we use only a fully connected neural network to identify model elements
2	Sequence as input and sequence as output (seq2seq)	This design is similar to a sequence labeling task. We feed the entire sequence into the neural network and obtain a sequence of labels, i.e., every word has its respective label predicted in the sequence. In this design, the context dependencies among words are considered. We use all four neural architectures for this design
3	Sequence with highlighted word as input and the label of highlighted word as output (seqlevelword)	Unlike the previous design, this design translates the problem of sequence labeling to a token classification task. To do so, we highlight each word in the sentence and train the neural network by feeding the highlighted sentence and making it learn the label of the highlighted word. We highlight a word by adding the '##' symbol before and after the word. Similar to seq2seq design, we consider all four neural network architectures in this design



Example Sentence: When a user presses switch, the motor runs.
Example of sentence with highlighted word: When a **##** user **##** presses switch, the motor runs.
Tokenized sentence:
When | a | user | presses | switch | , | the | motor | runs | .
user
Neural Network
B
Word
(Word user being fed as input)
Label
(Label of user)
When, a, user, ……, motor, runs, .
Neural Network
O, O, B, ……, O, O, O
Sequence of words
Sequence of labels
When, a, ##, user, ##, ……, runs, .
Neural Network
B
Sentence with highlighted word (user is highlighted)
Label (Label of user)
(a) Word as input and Word as output
(b) Sequence as input and Sequence as output
(c) Sentence with highlighted word as input and the label of highlighted word as output

Fig. 6 Text classification methods

Table 6 Prediction methods

#	Name	Description
1	Predicting each type of model elements separately (separate predictions)	In this method (illustrated in Fig. 7), we train the neural network for each type of model elements to learn and predict only that specific model element type (Fig. 7a)
2	Predicting all model element types together (combined predictions)	In this method, we train a single neural network classifier to predict all types of model elements (Fig. 7b)

Prediction method We considered two types of predictions. Their details are shown in Table 6, and examples with different types of predictions are illustrated in Fig. 7.

We use BIO tags (explained in Sect. 4) for *separate* predictions, but use a set of labels { A, TC, C, S, O } for *combined* predictions, where 'A' denotes actor, 'TC' denotes

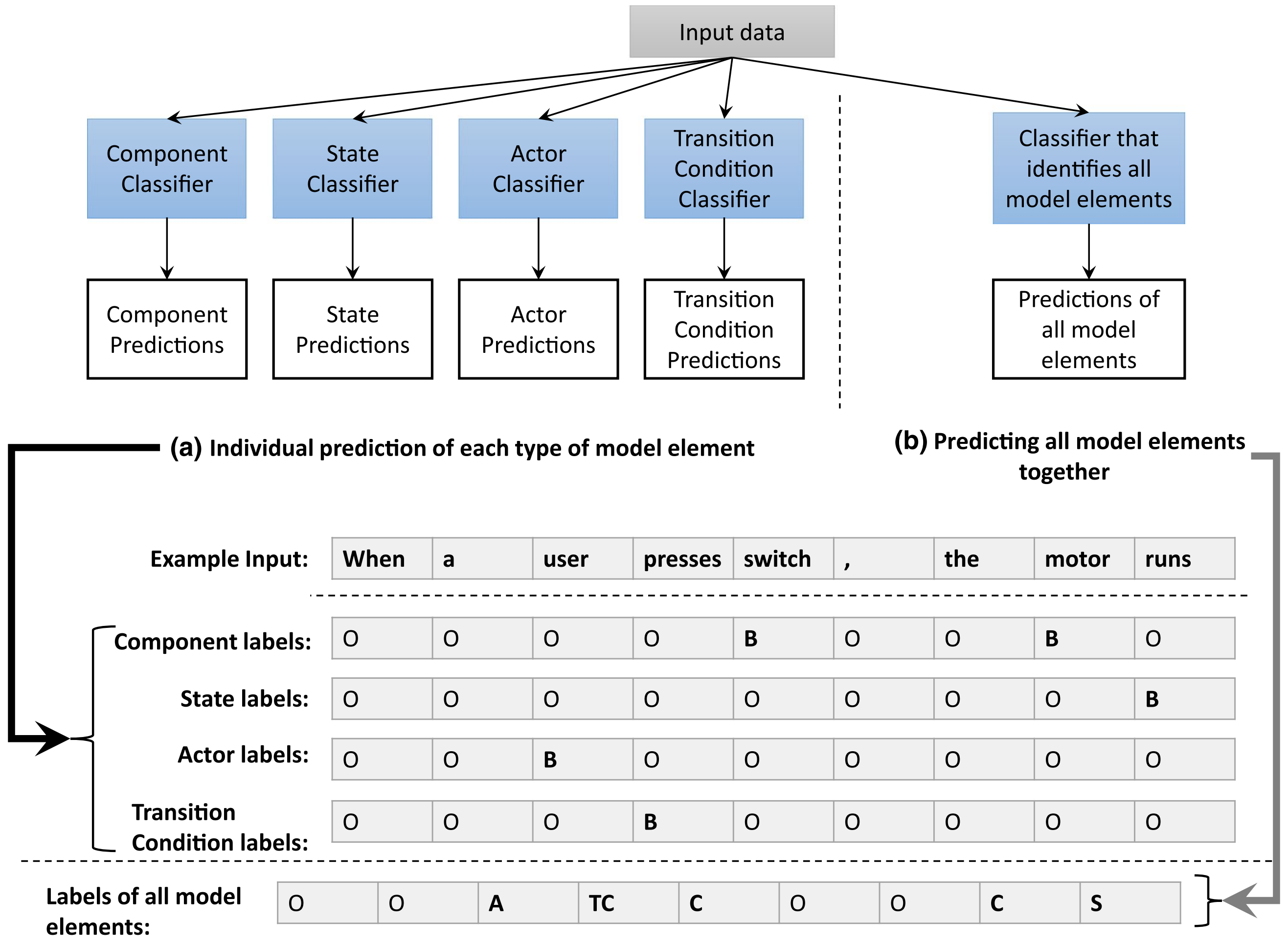

Input data
Component Classifier
State Classifier
Actor Classifier
Transition Condition Classifier
Classifier that identifies all model elements
Component Predictions
State Predictions
Actor Predictions
Transition Condition Predictions
Predictions of all model elements
(a) Individual prediction of each type of model element
(b) Predicting all model elements together
Example Input: When a user presses switch , the motor runs
Component labels: O O O O B O O B O
State labels: O O O O O O O O B
Actor labels: O O B O O O O O O
Transition Condition labels: O O O B O O O O O
Labels of all model elements: O O A TC C O O C S

Fig. 7 Examples of prediction methods

transition condition, ‘C’ denotes component, ‘S’ denotes state and ‘O’ denotes anything that is not a model element. An example of prediction types is illustrated in Fig. 7. The figure shows that when we apply separate predictions to our example (‘When a user presses switch, the motor runs.’), the labels of components, states, actors, and transition conditions are predicted using different classifiers. Each classifier predicts whether or not a token in the sentence is the type of the model element for which the classifier is trained. For example, the component classifier identifies ‘switch’ and ‘motor’ as components (shown with label B, i.e., beginning of component) whereas the state, ‘runs,’ is predicted as not a component (shown with label O, i.e., outside the component, which implies it is not a component). The combined predictions for the example sentence are shown at the bottom of Fig. 7. The main goal of using combined predictions is to evaluate whether there are associations among model element types that can improve the effectiveness of model element prediction. The classifier not only identifies ‘switch’ and ‘motor’ as components (denoted as C), but also identifies other types of model elements such as ‘user’ as an actor, ‘presses’ as a transition condition, and ‘runs’ as a state.

Annotation approach In our study, we used two types of annotations as shown in Fig. 8, which we defined for our component state transition model because there are no existing guidelines for identifying component state transition model elements. The first type of annotations is based on the guidelines we established in our previous work [14], and the second type of annotations is a refinement of the first type, which aims at improving the effectiveness of predicting model elements. Their details are described in Table 7.

We use *AT2* type of annotations because we aim to reduce the disagreement regarding the scope of what needs to be considered as a model element during the annotation process, which we faced during *AT1* type of annotations in our previous work [14]. Figure 8a, b shows *AT1*- and *AT2*-type annotations, respectively. We explain them using the example sentence shown in Fig. 8: ‘When an industrial user presses power switch, the motor controller also turns

Table 7 Annotation approaches

#	Name	Description
1	Annotation type 1 (AT1)	In this method (illustrated in Fig. 8a), if a model element is a phrase, i.e., the model element comprises of more than one word, we consider the entire phrase as part of the model element. We have used this type of annotations in our previous work [14]
2	Annotation type 2 (AT2)	This method (illustrated in Fig. 8b) requires each model element to be restricted to one word. We achieve this by choosing the head word in a phrase

Example:	When	an	industrial	user	presses	power	switch	,	the	motor	controller	also	turns	on

(a) Annotation Type 1 (used in previous work)

	When	an	industrial	user	presses	power	switch	,	the	motor	controller	also	turns	on
Components:	O	O	O	O	O	B	I	O	O	B	I	O	O	O
States:	O	O	O	O	O	O	O	O	O	O	O	O	O	B
Transition Conditions:	B	I	I	I	I	I	I	O	O	O	O	O	O	O
Actors:	O	O	B	I	O	O	O	O	O	O	O	O	O	O

(b) Annotation Type 2 (proposed in current work)

	When	an	Industrial	user	presses	power	switch	,	the	motor	controller	also	turns	on
Components:	O	O	O	O	O	O	B	O	O	O	B	O	O	O
States:	O	O	O	O	O	O	O	O	O	O	O	O	O	B
Transition Conditions:	O	O	O	O	B	O	O	O	O	O	O	O	O	O
Actors:	O	O	O	B	O	O	O	O	O	O	O	O	O	O

Fig. 8 Example of annotation approaches

on.' We used BIO tags (explained in Sect. 4) to indicate the scope of model elements. The component 'power switch' in the example is a noun with two words. As shown in Fig. 8, in *AT1*-type annotations, 'power' is marked as the beginning of component (label B) and 'switch' is marked as inside the component (label I), whereas in *AT2*-type annotations, the last word of 'power switch,' 'switch,' is labeled as the beginning of component (label B). They also recognize the transition condition differently. For example, when using *AT1*-type annotations, the entire phrase 'When an industrial user presses power switch' is considered as a transition condition; depicted by starting the beginning of the transition condition (label B) at 'When' and using inside transition condition labels (label I) for the rest of the phrase. However, in *AT2*-type annotations, the head word, which is the verb 'presses,' in the given sentence as a transition condition (depicted with label B). Note that *AT2* is a refinement of *AT1*. *AT2* annotates the head word of *AT1*, where the head is the most important or informative word [93]. In this paper, we focus more on *AT2*-type annotations than *AT1* because the latter involves phrase detection which often is predicted incompletely because of a small amount of data available. However, we will still compare the partial phrase matching result of *AT1*-type annotations to the results of *AT2*-type annotations. We aim to compare *AT1*-type annotations to *AT2*-type annotations because we want to analyze if model elements are identified effectively when we just identify the keyword of the model elements or when they are detected as a phrase or partial prediction of a phrase.

???

5.2.2 Dependent variable

The dependent variable for all research questions is the effectiveness of neural network architectures in predicting the model elements. We measured the effectiveness of neural networks using F_1-measure, which is the harmonic mean of precision and recall.

$$F_1\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Precision and Recall are defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives. True positives refer to correctly identified model elements. False positives refer to the entities that are incorrectly classified as model elements. False negatives are the occurrences of model elements that are incorrectly classified as not model elements.

We adopted F_1-measure because our data is imbalanced [94], i.e., the number of entities that are not model elements is higher than the number of model elements. We calculate F_1-measure for each model element to assess the performance of neural networks on each model element, which aids in choosing suitable architecture(s).

5.3 Experimental procedure

An overview of the approach we followed for our study is illustrated in Fig. 9. The ovals represent the processes, and the rectangles represent the outcomes of the processes as well as inputs to the processes. The numbers over the ovals represent the step numbers. Before we describe each step in detail, we provide a brief overview. We first annotate requirements documents by following the guidelines we established (Step 1). Once the annotations are complete, we create a gold standard from which we generate labeled data (Step 2). The generated data are partitioned into training, validation, and test sets (Step 3). We use training data to train the neural networks to predict model elements and use validation data to tune the hyperparameters of neural networks (Step 4). Once the hyperparameters are finalized, the generated classifier is treated as the finalized classifier and can be used to predict model elements on test data and unseen data (out of domain data) (Step 5). The final outcome of the approach is the predictions of model elements of data being evaluated by neural network classifiers. We now discuss each of these steps in detail.

Table 8 Inter-annotator agreement values for overall data (in Cohen's Kappa)

#	Annotation type	Inter-annotator agreement			
		Component	State	Actor	Transition condition
1	AT1	0.76	0.74	0.91	0.71
2	AT2	0.79	0.75	0.91	0.74

5.3.1 Step 1: Annotate requirements

The first step of our approach is to annotate requirements documents to generate data, which can be used to train and evaluate neural network classifiers. The general annotation process is described in Sect. 4. In our study, two doctoral students, who are familiar with component state transition (CST) diagrams and causal component model (CCM) annotated the requirements documents. However, unlike our previous annotations (*AT1*-type annotations) [14], we modified the guidelines and annotated the documents from scratch with new guidelines (*AT2*-type annotations). Our aim is to

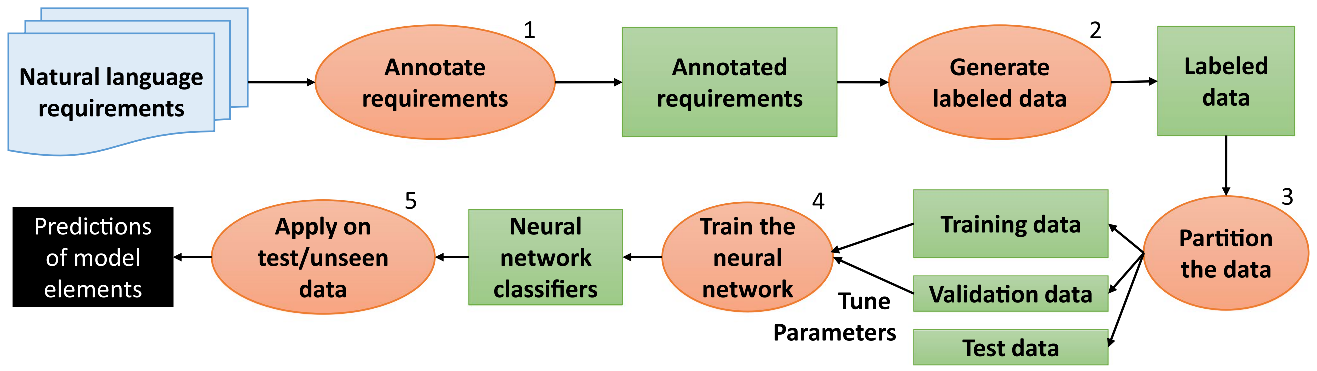


Fig. 9 Overview of approach used to conduct experiments

improve to the inter-annotator agreements as well as the effectiveness of neural networks. As mentioned earlier, our new annotation mechanism requires users to consider only the head word rather than the entire phrase. We used GATE annotation tool [95] to annotate the use case documents. The inter-annotator agreements we achieved using our old (*AT1*-type annotations) and new guidelines (*AT2*-type annotations) on the entire data set are shown in Table 8.

The agreement coefficients in the table indicate that there is not much difference in inter-annotator agreements of *AT1*-type and *AT2*-type annotations. Kappa values not only identify the observed agreement (i.e., percentage of times they agree) between annotators, but also take into account the agreement by chance. The small difference between Kappa values of *AT1* and *AT2*-type annotations is because of the reduction in disagreement between annotators regarding what constitutes a component and a transition condition. We believe that the Kappa value of component states is slightly increased because the annotators tend to agree on what constitutes a component more frequently, thereby making it easy to identify their corresponding states. There are no changes in the scores for actor because most of the actors are one word and the new guidelines do not affect the actors with one word. According to the literature [96, 97], a Kappa value between 0.6 and 0.8 indicates the annotations have substantial agreements and between 0.8 and 1.0 indicates the annotations are nearly perfect or perfect. Moreover, the high inter-annotator agreement score means the data is highly reliable [97]. Thus, we can conclude that our annotations are highly reliable for components, states, and transition conditions, and nearly perfect quality for actors.

5.3.2 Step 2: Generate labeled data

After completing the annotation process, the annotators reviewed and adjudicated the annotations, and decided what should be included in the data set that serves as our gold standard. If there were disagreements among annotators, a third annotator who is an expert in natural language processing resolved them. The finalized annotations serve as a gold standard, which is used as a reference for training and evaluation purposes. However, we cannot use annotations directly to train the data as the annotated information is in the form of meta-information and must be converted into the labeled format for the neural networks to learn. Thus, we generated labeled data by considering two prediction types (separate predictions and combined predictions) and the 3 experimental designs (*wordlevel*, *seq2seq* and *seqlevelword*). As mentioned earlier, we used BIO tags for *separate* predictions and {A, C, TC, S} tags for the *combined* predictions. An example illustrated in Fig. 7 shows the labeling method for *separate* predictions and *combined* predictions. We also created data sets suitable to evaluate different types of experimental designs.

5.3.3 Step 3: Partition the data

When evaluating neural network classifiers, due to the small size of data, we applied 5-fold cross-validation for both types of splitting using the first five use case documents in Table 1. For each iteration in the 5-fold cross-validation, we generated the corresponding training data, validation data, and test data. This is because *k*-fold cross-validation [62, 98–100] requires data to be partitioned into training and test sets, where for each iteration we change the training and test sets, ideally by using 1/*k*th part of data as test data. This test data is left out until training is finalized. But as explained in [99, 100], we cannot tune parameters on the test set because it can result in skewed results. Thus, we created a set called a validation set which is used for the purpose of tuning parameters. We used the sixth document as our unseen data, which we used to assess the performance of neural network architectures given new data. *Training data* is used for classifiers to learn the patterns and information that can identify model elements. *Validation data* is used for tuning hyperparameters and optimization parameters in the neural networks. Optimization parameters and hyperparameters that we tuned in our study are: (1) the number of epochs: the number of iterations in which the entire training data is passed to neural networks, (2) the learning rate: a float value that controls how we adjust the weights of the neural networks by considering the loss gradient, (3) the loss function: a function used to measure inconsistency between the actual value and predicted value, (4) the number of neural network layers, (5) the number of nodes within the layer, (6) the activation function: a function that determines the output for neural network nodes, and (7) the optimization function (optimizer): a function that is used to minimize the loss value.

Test data is the data, which is kept aside and is used to analyze the performance of neural networks. We performed two types of splitting: *inter-document* splitting (betweendoc) and *intra-document* splitting (withindoc), which are shown in Fig. 5.

5.3.4 Step 4: Train the neural network

Once the partitioning of data is completed, we trained the neural networks using training data and GloVe embeddings. Because neural networks accept only numerical input, we used word embeddings to convert the textual representation of words into a vector representation. For example, the word 'that' will be replaced by the following vector of 300-dimensions: $[-0.18256, 0.49851, -0.1639, \ldots, -0.19107, -0.094104]$. Also, we converted all labels to one-hot vectors, i.e., vectors in which the index representing the label has a value of 1, and the values in the rest of the indexes are 0. For example, in the case of separate predictions, we converted the value of ‘B’ into [0, 1, 0], ‘I’ into [0, 0, 1], and ‘O’ into [1, 0, 0]. Similarly, for combined predictions, we converted ‘O’ into [1, 0, 0, 0, 0]], ‘C’ into [0, 1, 0, 0, 0], ‘S’ into [0, 0, 1, 0, 0], ‘A’ into [0, 0, 0, 1, 0], and ‘TC’ into [0, 0, 0, 0, 1]. We tuned the hyperparameters mentioned in the previous step using validation data. We finalized the hyperparameters of the neural network classifiers by choosing the hyperparameters that produced the best results (in terms of F_1-measure) on validation data. The classifiers trained with these finalized tuned hyperparameters are treated as finalized classifiers. We performed this step for each fold and each classifier by considering the different types of neural networks, data splittings, experimental designs, and prediction types.

5.3.5 Step 5: Apply the finalized classifiers on test or unseen data

After the classifiers are finalized, we applied each of them to corresponding test data as well as the unseen data (sixth use case document shown in Table 1) to find the effectiveness of neural networks.

5.4 Results

To measure the performance of each classifier, we averaged F_1-measures of 5-fold cross-validation on test data that is evaluated in each iteration (represented as AVG 5-fold) and on unseen data (represented as AVG unseen). We also identified the F_1-measure on unseen data given by the classifier trained on the fold that produced the best results on test data during 5-fold cross-validation (represented as *BEST unseen*). The overall results of components, component states, actors, and transition conditions are shown in Table 9. For each element, the table lists the results considering three factors: the types of experimental design (labeled as ‘Design Type’), the types of machine learning algorithm used (labeled as ‘Alg. Type’), and the types of splitting and predictions (labeled as ‘Splitting and prediction types’). ‘Splitting and prediction types’ has two sub-columns: inter-document splitting (aka betweendoclevel splitting), and intra-document splitting (aka withindoclevel splitting). Each splitting type (inter-document splitting and intra-document splitting) is further divided into two columns, which contain the types of predictions: separate predictions and combined predictions. These predictions are again divided into three columns which represent the F_1-measures: AVG 5-fold, AVG Unseen, and BEST Unseen.

To provide a better understanding of the results for our research questions, we extract a subset of data from this table that is relevant to each research question and explain its results in the following subsections.

5.4.1 RQ1 results

To evaluate RQ1 and find the effectiveness of neural network architectures, we analyzed their performance on test data and unseen data. The ranges of F_1-measures for each neural network architectures are shown in Table 10. Overall, RNN with LSTM and RNN with GRU perform better on both test and unseen data when compared to FF and CNN. RNN with LSTM and RNN with GRU produced similar F_1 -measures for test data, but for unseen data, the results varied on the type of model elements. For example, for states, they produced similar results, but for actors and transition conditions, RNN with GRU performed slightly better and for components, RNN with LSTM performed slightly better.

Similarly, in the case of CNN and FF, they produced similar results for test data, but for unseen data, the results varied on the type of elements. For states and actors, FF performed better, but for components and transition conditions, CNN outperformed.

5.4.2 RQ2.1 results

For RQ2.1, we analyzed the effect of splitting methods on the performance of neural networks. We summarized the results of splitting methods in Table 11. Overall, we can observe that inter-document splitting produced a smaller range of F_1-measure values compared to intra-document splitting for all neural network architectures and all model elements. This result is expected as test data in inter-document splitting may not be similar to any of the instances in training data, whereas instances in test data can have some similarity with training data as they are extracted from same documents in intra-document splitting. We found that the type of splitting method has a negligible effect on performance of all neural networks and for all model element types with a few exceptions. However, we found some exceptions to this behavior. For example, CNN and RNN with GRU produced better results for actors with intra-document splitting than with inter-document splitting. Another example is that for FF, in which the range of F_1-measure values is higher for inter-document splitting than intra-document splitting for actors.

5.4.3 RQ2.2 results

To evaluate RQ2.2, we analyzed the performance of neural networks using three types of text classification methods (wordlevel, seqlvlword, and seq2seq). The wordlevel uses only a FF neural network, whereas the other two classification methods use all four neural network architectures. The

Table 9 Results of classifiers on model elements (in F_1-measure)

Model element	Design type	Alg. type	Splitting and prediction types											
			Inter-document splitting (betweendoclevel)						Intra-document splitting (withindoclevel)					
			Separate predictions			Combined predictions			Separate predictions			Combined predictions		
			AVG 5f	AVG Un	BST Un	AVG 5f	AVG Un	BST Un	AVG 5f	AVG Un	BST Un	AVG 5f	AVG Un	BST Un
Components	Wordlevel	FF	0.59	0.34	0.30	0.60	0.32	0.27	0.80	0.34	0.32	0.80	0.34	0.35
	Seqlvlword	FF	0.25	0.13	0.10	0.33	0.15	0.18	0.72	0.14	0.2	0.74	0.18	0.17
		CNN	0.19	0.11	0.12	0.3	0.25	0.22	0.55	0.24	0.22	0.47	0.26	0.23
		LSTM	0.69	0.38	0.41	0.66	0.43	0.44	0.89	0.41	0.52	0.89	0.40	0.44
		GRU	0.64	0.39	0.35	0.70	0.40	0.42	0.9	0.38	0.41	0.89	0.43	0.43
	Seq2seq	FF	0.56	0.34	0.34	0.63	0.33	0.33	0.83	0.36	0.38	0.82	0.35	0.31
		CNN	0.63	0.35	0.4	0.59	0.33	0.36	0.82	0.36	0.39	0.82	0.34	0.38
		LSTM	0.59	0.34	0.34	0.60	0.36	0.36	0.88	0.32	0.38	0.87	0.33	0.31
		GRU	0.61	0.36	0.41	0.59	0.32	0.32	0.89	0.37	0.38	0.87	0.31	0.27
States	Wordlevel	FF	0.53	0.33	0.32	0.56	0.36	0.43	0.80	0.27	0.32	0.81	0.39	0.41
	Seqlvlword	FF	0.23	0.07	0.12	0.26	0.07	0.04	0.61	0.11	0.10	0.76	0.09	0.14
		CNN	0.16	0.09	0.24	0.33	0.20	0.17	0.45	0.10	0.24	0.45	0.20	0.17
		LSTM	0.72	0.31	0.2	0.69	0.29	0.23	0.83	0.36	0.34	0.90	0.33	0.35
		GRU	0.76	0.42	0.31	0.71	0.34	0.26	0.82	0.40	0.38	0.91	0.31	0.36
	Seq2seq	FF	0.48	0.31	0.34	0.51	0.30	0.34	0.84	0.32	0.38	0.83	0.31	0.32
		CNN	0.51	0.32	0.34	0.52	0.30	0.34	0.83	0.30	0.37	0.84	0.28	0.26
		LSTM	0.66	0.42	0.38	0.68	0.38	0.42	0.90	0.45	0.46	0.89	0.42	0.46
		GRU	0.75	0.44	0.40	0.7	0.43	0.4	0.92	0.45	0.4	0.89	0.44	0.45
Actors	Wordlevel	FF	0.35	0.56	0.80	0.25	0.14	0.11	0.88	0.59	0.60	0.90	0.55	0.17
	Seqlvlword	FF	0.05	0.07	0.11	0.09	0.11	0.13	0.67	0.05	0.00	0.76	0.13	0.08
		CNN	0.09	0.15	0.03	0.09	0.26	0.19	0.59	0.26	0.47	0.31	0.36	0.47
		LSTM	0.28	0.38	0.42	0.55	0.6	0.72	0.91	0.48	0.61	0.92	0.61	0.72
		GRU	0.38	0.35	0.14	0.75	0.54	0.75	0.92	0.67	0.73	0.91	0.70	0.72
	Seq2seq	FF	0.00	0.12	0.17	0.00	0.13	0.16	0.88	0.12	0.16	0.88	0.13	0.16
		CNN	0.00	0.12	0.15	0.13	0.13	0.16	0.88	0.13	0.16	0.88	0.13	0.16
		LSTM	0.16	0.36	0.43	0.23	0.51	0.13	0.90	0.40	0.37	0.90	0.57	0.74
		GRU	0.22	0.64	0.12	0.18	0.52	0.1	0.91	0.80	0.84	0.91	0.70	0.65
Transition conditions	Wordlevel	FF	0.32	0.30	0.33	0.37	0.34	0.35	0.78	0.29	0.26	0.79	0.33	0.35
	Seqlvlword	FF	0.05	0.08	0.07	0.10	0.07	0.10	0.73	0.06	0.05	0.72	0.09	0.11
		CNN	0.08	0.06	0.05	0.20	0.19	0.10	0.58	0.03	0.08	0.55	0.2	0.11
		LSTM	0.42	0.43	0.4	0.35	0.45	0.49	0.86	0.48	0.42	0.85	0.43	0.44
		GRU	0.48	0.51	0.46	0.44	0.46	0.49	0.87	0.49	0.43	0.84	0.46	0.46
	Seq2seq	FF	0.37	0.32	0.29	0.41	0.34	0.29	0.81	0.35	0.37	0.81	0.35	0.36
		CNN	0.38	0.37	0.34	0.40	0.30	0.29	0.80	0.35	0.34	0.81	0.32	0.32
		LSTM	0.41	0.39	0.39	0.44	0.46	0.47	0.86	0.37	0.42	0.85	0.49	0.47
		GRU	0.44	0.37	0.23	0.52	0.55	0.46	0.87	0.44	0.44	0.86	0.54	0.57

ranges of F_1-measures for each type of text classification method are summarized in Table 12. Overall, we found that the performance of text classification methods varied on the type of neural network architectures and model elements. The results of wordlevel FF are comparable to the results of seqlvlword and seq2seq classification methods using RNN with LSTM and RNN with GRU, for both test data and unseen data. Seqlvlword and seq2seq methods produced higher ranges of results using RNN with LSTM and RNN with GRU. While their results on test and unseen data are similar to those for transition conditions, seqlvlword method produced slightly higher results

Table 10 Range of F_1-measures for neural network architectures (considering all factors affecting the performance of neural networks)

Neural network	Model element	Range of F_1 values for test data	Range of F_1 values for unseen data
FF	Component	0.25–0.83	0.10–0.38
	State	0.23–0.84	0.04–0.43
	Actor	0.00–0.90	0.00–0.80
	Transition condition	0.05–0.81	0.05–0.37
CNN	Component	0.19–0.82	0.11–0.40
	State	0.16–0.84	0.09–0.37
	Actor	0.00–0.88	0.03–0.47
	Transition condition	0.08–0.81	0.03–0.47
RNN with LSTM	Component	0.59–0.89	0.31–0.52
	State	0.66–0.90	0.20–0.46
	Actor	0.16–0.92	0.13–0.74
	Transition condition	0.35–0.86	0.39–0.49
RNN with GRU	Component	0.59–0.90	0.27–0.43
	State	0.70–0.92	0.26–0.45
	Actor	0.18–0.92	0.10–0.84
	Transition condition	0.44–0.87	0.23–0.57

Table 11 Range of F_1-measures for splitting methods

Neural network	Model element	Range of F_1 values			
		Inter-document splitting		Intra-document splitting	
		Test data	Unseen data	Test data	Unseen data
FF	Component	0.25–0.63	0.10–0.37	0.73–0.83	0.14–0.38
	State	0.23–0.56	0.04–0.43	0.61–0.84	0.09–0.41
	Actor	0.00–0.35	0.07–0.80	0.67–0.90	0.00–0.60
	Transition condition	0.05–0.41	0.07–0.35	0.72–0.81	0.05–0.37
CNN	Component	0.19–0.63	0.11–0.34	0.47–0.83	0.22–0.38
	State	0.16–0.51	0.09–0.34	0.45–0.83	0.10–0.37
	Actor	0.00–0.13	0.03–0.26	0.31–0.88	0.13–0.47
	Transition condition	0.08–0.40	0.05–0.37	0.55–0.81	0.03–0.35
RNN with LSTM	Component	0.59–0.69	0.34–0.44	0.87–0.89	0.31–0.52
	State	0.66–0.72	0.20–0.42	0.83–0.90	0.33–0.46
	Actor	0.16–0.55	0.13–0.72	0.90–0.92	0.37–0.74
	Transition condition	0.35–0.44	0.39–0.49	0.85–0.86	0.37–0.49
RNN with GRU	Component	0.59–0.70	0.32–0.42	0.87–0.90	0.27–0.43
	State	0.70–0.76	0.26–0.44	0.82–0.92	0.31–0.45
	Actor	0.18–0.75	0.10–0.75	0.91–0.92	0.65–0.84
	Transition condition	0.44–0.52	0.23–0.55	0.84–0.87	0.43–0.57

for components, and seq2seq method produced slightly higher results for states. In the case of actors, while both the seq2seq and seqlvlword methods produced similar results on test data for RNN with LSTM and RNN with GRU, seq2seq RNN with GRU produced higher results on unseen data. Furthermore, wordlevel FF also produced slightly lower scores compared to seq2seq RNN with GRU for actors.

5.4.4 RQ2.3 results

To answer RQ2.3, we analyzed the effect of prediction methods on neural networks and the performance of classifiers by training them for separate predictions and combined predictions. We summarized the results in Table 13. Overall, we can observe that all neural network architectures produced similar results for separate and combined predictions on both

Table 12 Range of F_1-measures for text classification methods

Text classification method	Neural network	Model element	Range of F_1 values	
			Test data	Unseen data
Wordlevel	FF	Component	0.59–0.63	0.27–0.35
		State	0.53–0.81	0.27–0.43
		Actor	0.25–0.90	0.11–0.80
		Transition condition	0.32–0.79	0.26–0.35
Seqlvlword	FF	Component	0.25–0.74	0.10–0.20
		State	0.23–0.76	0.04–0.14
		Actor	0.05–0.76	0.00–0.13
		Transition condition	0.05–0.73	0.05–0.11
	CNN	Component	0.19–0.55	0.11–0.26
		State	0.16–0.45	0.09–0.24
		Actor	0.09–0.59	0.03–0.47
		Transition condition	0.08–0.58	0.05–0.20
	RNN with LSTM	Component	0.66–0.89	0.38–0.52
		State	0.69–0.90	0.20–0.36
		Actor	0.28–0.92	0.38–0.72
		Transition condition	0.35–0.86	0.40-0.49
	RNN with GRU	Component	0.64–0.90	0.35–0.43
		State	0.71–0.91	0.26–0.42
		Actor	0.38–0.92	0.14–0.75
		Transition condition	0.44–0.87	0.43–0.51
Seq2seq	FF	Component	0.56–0.83	0.31–0.38
		State	0.48–0.84	0.30–0.38
		Actor	0.00–0.88	0.12–0.17
		Transition Condition	0.37–0.81	0.29–0.37
	CNN	Component	0.59–0.82	0.32–0.40
		State	0.51–0.84	0.26–0.37
		Actor	0.00–0.88	0.12–0.16
		Transition Condition	0.38–0.81	0.29–0.37
	RNN with LSTM	Component	0.59–0.88	0.31–0.38
		State	0.66–0.90	0.38–0.46
		Actor	0.16–0.90	0.13–0.74
		Transition condition	0.41–0.86	0.37–0.49
	RNN with GRU	Component	0.59–0.89	0.27–0.41
		State	0.70–0.92	0.40–0.45
		Actor	0.22–0.91	0.10–0.84
		Transition condition	0.44–0.87	0.23–0.57

test and unseen data with a few exceptions. For example, for FF, separate predictions produced a higher upper bound value (0.00–0.80) than combined predictions (0.08–0.55) for unseen data for actors. In the case of transition conditions, for both RNN with LSTM and RNN with GRU, the range values are higher and with a small variance for combined predictions than separate predictions. In the case of actors, for RNN with LSTM, combined predictions produced a higher upper bound value (0.74) than separate predictions (0.61), whereas for RNN with GRU, separate predictions produced a higher upper bound value (0.84) than combined predictions (0.75).

5.4.5 RQ2.4 results

To investigate RQ2.4, we used our annotated data from our previous paper [14], which followed the guidelines that are different from the ones used for our new annotation data. Because our old annotation data has only 6 of 11 use cases of unseen data, we applied the experiments for RQ2.4 by only considering 6 use cases. Furthermore, because our previous results show better results with GRU and LSTM architectures, and they are the state-of-the-art architectures in natural language processing research, we investigated only those architectures for this research

Table 13 Range of F_1-measures for prediction methods

Neural network	Model element	Range of F_1 values			
		Separate predictions		Combined predictions	
		Test data	Unseen data	Test data	Unseen data
FF	Component	0.25–0.83	0.10–0.38	0.33–0.82	0.15–0.38
	State	0.23–0.84	0.07–0.38	0.26–0.83	0.04–0.43
	Actor	0.00–0.88	0.00–0.80	0.0–0.90	0.08–0.55
	Transition condition	0.05–0.81	0.05–0.37	0.10–0.81	0.07–0.36
CNN	Component	0.19–0.82	0.11–0.40	0.30–0.82	0.22–0.38
	State	0.16–0.83	0.09–0.37	0.33–0.84	0.17–0.34
	Actor	0.00–0.88	0.03–0.47	0.09–0.88	0.13–0.47
	Transition condition	0.08–0.80	0.03–0.37	0.20–0.81	0.10–0.32
RNN with LSTM	Component	0.59–0.89	0.32–0.52	0.60–0.89	0.31–0.44
	State	0.66–0.90	0.20–0.46	0.68–0.90	0.23–0.46
	Actor	0.16–0.91	0.36–0.61	0.23–0.92	0.13–0.74
	Transition condition	0.41–0.86	0.37–0.48	0.35–0.85	0.43–0.49
RNN with GRU	Component	0.61–0.90	0.35–0.41	0.59–0.89	0.27–0.43
	State	0.76–0.92	0.31–0.45	0.70–0.91	0.26–0.45
	Actor	0.22–0.92	0.12–0.84	0.18–0.91	0.1–0.75
	Transition condition	0.44–0.87	0.23–0.51	0.44–0.86	0.46–0.57

Table 14 Results of classifiers using different types of annotations (in F_1-measure)

Alg. type	Model element	Types of annotations					
		Annotations type 1				Annotations type 2	
		Strict F_1		Lenient F_1		F_1 (Strict = Lenient)	
		Test	Unseen	Test	Unseen	Test	Unseen
LSTM	Component	0.79	0.44	0.86	0.52	0.81	0.40
	State	0.75	0.27	0.75	0.27	0.81	0.35
	Actor	0.73	0.75	0.73	0.75	0.95	0.66
	Transition condition	0.25	0.00	0.76	0.51	0.85	0.42
GRU	Component	0.74	0.51	0.85	0.60	0.83	0.43
	State	0.77	0.29	0.77	0.29	0.86	0.32
	Actor	0.74	0.75	0.74	0.75	0.93	0.80
	Transition condition	0.25	0.00	0.76	0.51	0.87	0.39

question. However, to have a better understanding on how annotation methods can affect results, we fixed the text classification method to seq2seq, splitting method to withindoclevel, and predictions method to separate predictions. The values we obtained using different annotations are shown in Table 14. Overall, we found that annotation type 2 performs well on test data for all model element types except for components and on unseen data for states (for both LSTM and GRU) and actors (only for GRU). Annotation type 1 (AT1) performs well on components and transition conditions for unseen data when we consider lenient F_1-measure, i.e., when we consider partial matches of model elements.

As the table shows, annotation type 1 has a strict and lenient score, whereas annotation type 2 has only one score in which strict and lenient scores are considered to be the same. This is because, in annotation type 1, which is our previous annotation type, we can annotate more than one word to represent model elements. If a model element has more than one word, predicting all words of the model element is considered a strict match, whereas predicting at least one word of the model element is considered a lenient match (also known as a partial match). In annotation type 2, the current annotations type, the guidelines allow the annotators to choose only one word to represent any model element. Therefore, we do not have strict and lenient matches for annotation type 2.

The results show that the strict values of transition condition (0.25 and 0.00 in LSTM and GRU, respectively) are very low compared to a lenient match for annotation type 1 (0.76 and 0.51 for LSTM and GRU, respectively). This is because the classifiers are able to predict part of the transition conditions but not the entire phrase. The case of components is similar to transition conditions, however the difference between strict (0.79 and 0.44 in LSTM and 0.74 and 0.51 in GRU) and lenient scores is smaller when compared to transition conditions. In the case of states and actors, which mostly have single words, strict and lenient scores are not different. When we compared the results of annotation types 1 and 2 in Table 14, we can see that both LSTM and GRU produced better prediction results for states with annotation type 2 (for both test and unseen data). In GRU, for example, the strict as well as lenient scores on test data and unseen data with annotation type 1 for the model element state are 0.77 and 0.29, respectively; the scores for test and unseen data with annotation type 2 for the model element state are 0.81 and 0.35, respectively.

In the case of LSTM, while the actors are predicted better on test data when we used annotation type 2 compared to both strict and lenient F_1 values of annotation type 1, a higher value was achieved on unseen data for annotation type 1 for both strict and lenient F_1 values. For components, the F_1 values on test data using annotation type 2 are higher than strict values but lower than lenient values of annotation type 1 for both LSTM and GRU. However, the F_1 values for unseen data using annotation type 2 (0.40 for LSTM and 0.43 or GRU) are lower than both strict and lenient values of component (0.44 and 0.52 for LSTM and 0.51 and 0.60 for GRU) when using annotation type 1. In the case of transition conditions, neural networks trained using annotation type 2 have higher values than the strict scores of annotations type 1. Conversely, the lenient scores for unseen data is higher for annotation type 1 (0.51 for both LSTM and GRU) when compared to annotation type 2 (0.42 for LSTM and 0.39 for GRU).

The numbers of model elements for each type we annotated using annotation types 1 and 2 are shown in Table 15. As the table shows, the results from two annotation types are quite different. For example, annotation type 1 has 64 components in test data and 64 components in unseen data, and annotation type 2 has 96 components in test data and 127 components in unseen data. The reason for having a higher number of components with annotation type 2 is that we changed the guidelines and narrowed the scope of what can be a component. While annotation type 2 helps in finding the core word of each model element, we need to use a dependency parser for post-processing to get the entire information associated with the model element. For example, suppose we have a component named *Device Controller Monitor (DCM)*. In this case, annotation type 2 identifies a single word, 'Monitor,' as a component ignoring two other words, 'Device' and 'Controller.' To resolve this problem, a dependency parser should be applied to identify them. A high strict match used in annotation type 1 reduces this post processing effort. However, when we consider a lenient match, if the word detected is not a head word or core word from which we can identify dependencies, we might need human effort or post processing. Therefore, there are trade-offs between types of annotations and post processing effort.

Table 15 Count of model elements in each annotation type

Model element	Annotation type 1		Annotation type 2	
	Test	Unseen	Test	Unseen
Component	64	64	96	127
State	42	34	56	33
Actor	29	40	47	53
Transition condition	25	32	45	59

5.4.6 RQ3 results

From the results of RQ1 and RQ2, we can say that every neural network architecture is influenced by various factors such as the types of text classification methods, splitting methods, annotation methods, and predictions methods. Thus, choosing the same neural network architecture without considering the factors might not be ideal. We need to select appropriate neural networks considering suitable factors that can increase the model element prediction results. Overall, the splitting methods can affect the results of model elements on test data and it is preferable to use withindoclevel partitioning, in particular when the data size is small. For components, whether it is betweendoclevel splitting or withindoclevel splitting, we can obtain better results using seqlvlword LSTM or GRU with combined predictions. For states, we can achieve better results using seq2seq GRU or LSTM with separate predictions. For actors, using seq2seq GRU with separate predictions, we can get higher scores on test and unseen data for withindoclevel splitting. However, we can achieve higher scores at betweendoclevel partitioning using seqlvlword GRU or LSTM with combined predictions. In the case of transition conditions, seq2seq GRU with combined predictions will aid in getting higher scores.

5.4.7 Central RQ results

Our central research question seeks to find the most effective neural network architectures and factors affecting their performance. Based on the findings from our previous research questions, we answer our central research question as follows:

1. RNN with LSTM and RNN with GRU are the neural network architectures with the highest performance. However, there are different factors that must be considered for different model elements. Wordlevel FF also performed similar but slightly worse than those.
2. We found that the splitting method we used might affect performance of classifiers on test data but not on unseen (out-of-domain) data.
3. We found that text classification methods can affect prediction scores. We found that wordlevel FF performs on par with top performing classifiers and produced results whose F_1 values are higher than the ones produced by seqlvlword FF and seq2seq FF. We also found that seqlvlword LSTM is suitable for learning components and seq2seq GRU is suitable for states and actors.
4. Regarding the prediction methods used, we found that there is not much difference in F_1-measures between prediction methods for all neural network architectures.
5. We observed that the way in which annotations are performed can affect the results. However, it comes with a trade-off. The effective annotation mechanisms can require additional post-processing and analysis.
6. From our results, we also found that the classifier trained on the fold that produced the best result on test data during 5-fold cross-validation does not necessarily produce the best results on unseen data. We found that some other folds can produce better results on unseen data. For example, from Fig. 9, we can observe that in the case of seq2seq RNN with LSTM, for actors, we found that the F_1-measure using best fold is 0.13, whereas the average of all F_1-measures on unseen data by all 5 folds is 0.51. This points us the necessity to identify ways to choose the best possible fold for unseen data depending on the unseen data characteristics.

6 Threats to validity

Our first threat to validity involves the annotation procedure performed in our study. The model element identification is subject to human judgment, and thus, the results can differ from the annotations produced by other annotators. To reduce this threat, we carefully examined and validated our guidelines eliminate any confusions and use two annotators. Moreover, we used a third annotator, who is also an expert in the field to resolve any conflicts between two annotators. We also note that this threat is virtually eliminated given the high Kappa coefficients. The second threat to validity is the generalization of results of neural networks to other requirements documents and industrial documents. The results are dependent on the data being used as well as the size of the data. Thus, we might not be able to generalize the results for other documents and application domains. We plan to address this by considering large data sets and by testing them on multiple requirements documents including industrial documents.

7 Discussion

Our results indicate that the performance of neural network architectures can be affected by various factors such as splitting methods, annotation methods, and text classification methods. We found that for a small amount of data, it is better to use intra-document splitting method because it follows the principle of independent and identically distributed (iid) data [101, 102]. We also found that RNN with GRU and RNN with LSTM perform better in predicting all model elements. Further, we also learned a particular text classification method or annotation approach can work better for a certain model element. Thus, we need to apply different text classification methods or annotation approaches for different model elements to produce desirable prediction results. The results also show that to achieve overall better results, one should not restrict the same design for all model elements.

Industrial feedback: We contacted industrial engineers who work on requirements and analyze them for safety issues and received feedback from them regarding the automated model elements identification. The engineers responded positively about our results, but they also suggested that the results need to be improved further especially for inter-document splitting. They suggested to use the method of annotating a part of document manually and use it to retrain a model and predict the elements in the remaining of the document when we want to gather more labeled data in a smaller amount of time. The engineers mentioned that it would be more realistic to construct a model of a complete requirements document rather than to construct a model using a part of the document. Moreover, the engineers also pointed out that all the details related to the model might not necessarily exist in the requirements. Thus, they suggested we improve the results further before moving our direction to automatic relation extraction. One engineer also pointed out that providing a means to automatically suggest missing model elements would be beneficial.

In our study, we analyzed the performance of various neural network architectures when given a small amount of data. However, the results raised additional questions such as: (1) How do traditional machine learning techniques that use natural language features perform when compared to neural networks, especially given a small amount of data? (2) What makes model element identification task challenging for neural networks? (3) How much of unseen data or out-of-domain data must be manually labeled to obtain desirable results on the rest of unseen or out-of-domain data? To address these questions, we compared the performance of neural network architectures with a sequence labeling algorithm: conditional random fields (CRF) [33, 34], performed error analysis, and conducted additional analyses on the effect of the amount of manually labeled unseen data on the performance of neural networks. We describe each of these analyses in the following subsections.

7.1 Comparison with conditional random fields (CRF)

Prior to the advent of deep neural networks, conditional random field (CRF) [33, 34] (explained in Sect. 4) is the machine learning algorithm that produced state-of-the-art results for sequence labeling task. The design of CRF is similar to *seq2seq* text classification method. The major advantage of CRF is that it takes less time to train than neural networks. However, it requires feature engineering that usually involves complex heuristics generated manually, whereas neural networks neither need feature engineering nor require manual generation of complex heuristics.

We compare CRF with neural networks in terms of their model element prediction results using a small data set. We used Mallet simple tagger [103], a third party tool, to implement CRF classifiers. We trained CRF classifiers following the procedure mentioned in Sect. 5, similar to neural networks. The results of CRF classifiers for detecting model elements are shown in Table 16. The results indicate that the effectiveness of the best performing neural network classifiers (GRU and LSTM) for finding model elements is on par with CRFs. Moreover, similar to all neural network architectures, CRF resulted in higher AVG 5-fold values for withindoclevel splitting, and the effect of type of splitting is negligible on its performance on unseen data.

We discuss the results of each model element shown in Tables 9 and 16.

Components For components, CRF produced similar AVG 5-fold and AVG Unseen values as seqlvlword LSTM and GRU. For example, the AVG Unseen value for betweendoclevel CRF with combined predictions is 0.41, whereas the AVG Unseen values for seqlvlword LSTM and GRU for betweendoclevel splitting and combined predictions are 0.44 and 0.42, respectively. However, Best Unseen is the highest value for CRF for betweendoclevel separate predictions and for LSTM for the rest of the cases. Furthermore, CRF outperformed most of the neural network architectures with seq2seq and wordlevel designs.

States In the case of states, CRF produced higher AVG Unseen and BEST Unseen values than all neural network architectures for combined predictions. However, seq2seq GRU and LSTM outperformed CRF for separate predictions. Overall, the highest values achieved by CRF are close to the best performing values of LSTM and GRU. For example, the highest AVG Unseen value of CRF is 0.47, whereas the highest AVG Unseen values for LSTM and GRU (for seq2seq design) are 0.45 and 0.45, respectively.

Actors For actors, GRU and LSTM outperformed CRF for betweendoclevel splitting for both test and unseen data. For withindoclevel splitting, CRF produced similar results as GRU and LSTM for AVG 5-fold of test data but it did not perform well compared to GRU and LSTM on unseen data (AVG Unseen and BEST Unseen values). For example, for betweendoclevel splitting and separate predictions, the AVG 5-fold, AVG Unseen, and BEST Unseen values for CRF are 0.91, 0.13, and 0.16, respectively, whereas the values for GRU are 0.91, 0.80, and 0.84, respectively.

Transition conditions For transition conditions, CRF produced the highest BEST Unseen values and seqlvlword GRU produced the highest AVG 5-fold and AVG Unseen values are highest for separate predictions. In the case of combined predictions, seq2seq GRU outperformed CRF on both test data (higher AVG 5-fold values) and unseen data (higher AVG Unseen and BEST Unseen values) with a few exceptions. One exception is that the AVG 5-fold value of CRF for withindoclevel combined predictions with a value of 0.90, which is slightly higher than the AVG 5-fold value of seq2seq GRU (0.86).

Table 16 Results of CRF for identifying model elements(in F_1-measure)

Design type	Model elements	Splitting and prediction types											
		Inter-document splitting						Intra-document splitting					
		Separate predictions			Combined predictions			Separate predictions			Combined predictions		
		AVG 5-fold	AVG Unseen	BEST Unseen	AVG 5-fold	AVG Unseen	BEST Unseen	AVG 5-fold	AVG Unseen	BEST Unseen	AVG 5-fold	AVG Unseen	BEST Unseen
CRF	Component	0.63	0.38	0.49	0.64	0.40	0.41	0.91	0.35	0.33	0.92	0.40	0.37
	State	0.58	0.27	0.2	0.67	0.46	0.48	0.88	0.3	0.38	0.92	0.47	0.48
	Actor	0.04	0.13	0.09	0.19	0.47	0.55	0.91	0.13	0.16	0.92	0.56	0.52
	Transition condition	0.37	0.47	0.48	0.39	0.47	0.51	0.83	0.46	0.47	0.90	0.50	0.51

7.2 Error analysis

In this section, we analyzed the source of errors for each neural network as well as the effect of unseen words in our data on the predictions of classifiers. The goal of our error analysis is to address *'when the model element identification task is most challenging.'* By answering this question, we can identify plausible reasons for hindering the performance of neural networks and identify possible solutions for them.

Procedure for error analysis In order to perform error analysis [104, 105], we manually analyzed the results of each neural network classifier. Because we have more than one thousand experimental data sets, which require significant manual efforts and time, we analyzed a subset of results, the first two folds in the five folds used for cross-validation. The steps we followed for error analysis are as follows:

1. For each experiment, we identified the words and model elements that are incorrectly classified. We grouped the incorrect predictions into two types: (1) model elements that are incorrectly classified as other types of model elements or non-model elements, and (2) words that are not model elements but are identified as model elements.
2. Once we grouped the results, we identified the common patterns among incorrect predictions within and between groups.
3. To identify more causes of incorrect predictions, we considered parts-of-speech tags, possible semantics of the word, and dependencies among words. We used this information to identify additional patterns and reasons that can result in incorrect predictions. During this analysis, we linked the predictions to instances in training data to understand why certain patterns occurred among incorrect predictions.
4. To analyze the relationship between incorrect predictions and training data, we examined training data to identify potential causes that might have resulted in wrong predictions in test data.
5. We repeated aforementioned steps for each model element for both separate predictions and combined predictions and further filtered the common patterns that result in incorrect predictions for each design type of neural network architectures. We identified these common patterns as the potential sources of errors that make the model element identification task challenging.

Results of error analysis Table 17 shows the results of error analysis. We only report the most frequently occurred errors. In the table, each row represents a neural network architecture, its design type, and their sources of incorrect predictions. Each source of incorrect predictions is discussed as follows:

1. *Learning without context* Wordlevel FF learns the predictions without considering the context in which the input is used. This resulted in wrong predictions as the context of words in the test or unseen documents does not fit the context of words in training data. For example, the 'user' in 'user manual' is identified as an actor although according to guidelines, it is not.
2. *Unseen words and data* Words that are model elements (e.g., actors such as citizen and supervisor, and components such as application) and that are not part of instances in training data are not predicted well. For example, the highlighted word 'citizen' in 'The ## citizen ## must have Internet connection' is incorrectly predicted as not an actor because it is not a part of training data. This outcome is expected as the data size is small, in which unknown instances can affect predictions of classifiers.
3. *Words with similar semantics* Words with similar semantics to model elements are also predicted as model elements irrespective of their usage. For example, the word 'details,' which is semantically closer to the word 'information' is often incorrectly predicted as

Table 17 Sources of incorrect predictions for different designs of neural network architectures

Neural network	Text classification method	Sources of incorrect predictions
FF	Wordlevel	Learning without context, unseen words, words with similar meanings
	Seqlvlword	Unseen data, incorrect contextual learning
	Seq2seq	Incorrect contextual learning, unseen words
CNN	Seqlvlword	Incorrect pattern learning, unseen context
	Seq2seq	Frequently used associations, homographs, states that are not verbs
RNN with LSTM	Seqlvlword	unseen words, unseen context
	Seq2seq	Compound nouns, homographs, words with similar context
RNN with GRU	Seqlvlword	unseen context, homographs
	Seq2seq	Compound nouns, missing subjects

a component. The reason is because the word 'information' is considered as a component in training data.

4. *Incorrect contextual learning* We found that in FF, because every node is connected to every other node, the classifier is unable to learn contextual dependencies properly, which affected the outcomes of the classifiers. For example, the word 'dates' in the sentence 'The supervisor enters the period start and end ## dates ## .' is incorrectly classified as a component. We attribute these results to the way of learning is done via FF. Because we use extra symbols such as '##' and consider all words in a sentence to predict the label of the highlighted word, it is possible that words other than surrounding words of the highlighted word have higher weights, thereby affecting the outcome.
5. *Incorrect pattern learning* While seqlvlword CNN considers the context of words in a sentence, it does not consider the long-term order in which the words are in a sentence. This led to wrong predictions. For example, in the sentence 'The Dispatcher ## evaluates ## the criticality of the situation,' the highlighted word 'evaluates' is incorrectly predicted as an actor instead of a transition condition. This is because of association of the surrounding word 'Dispatcher,' which is an actor. The use of '##' for the word after or before a model element is often being confused as a feature associated with the model element.
6. *Unseen context* Unseen association of words (words that co-occurred in test data or unseen data but not training data) resulted in incorrect predictions for seqlvlword design of CNN, RNN with LSTM, and RNN with GRU. For example, in the sentence 'The system displays the missing fields to be ## completed ##.', the highlighted word 'completed' is not identified as a state because the words 'to,' 'be,' and 'completed' never co-occurred together in the training data.
7. *Frequently used associations* Unlike seqlvlword CNN, seq2seq CNN does not have a problem with unseen co-occurrences of words. Rather, seq2seq CNN does not predict the model element types correctly if the words in the model element are used as different types of model elements in the training data. Seq2seq CNN identifies the model element with most frequently used label rather than considering the subject of the model element. For example, the word 'requests' in the sentence 'The system requests the caregiver to accept the community profile information.' is not identified as a state because the word is often used as a transition condition in association with actors such as an operator and dispatcher.
8. *Homographs* The presence of homographs (words with same spelling but different meanings) affected the outcomes of seq2seq CNN, seq2seq RNN with LSTM, and seqlvlword RNN with GRU. For example, in the sentence 'The display shows the stops in the route and details the pending time until the arrival to the nearest stop.', the word 'details' is a verb and a state but is identified as a component because the word 'details' is considered as a noun.
9. *States that are not verb* Seq2seq CNN is not able to find states that are not verbs. This might be due to the fact that the majority of states are verbs, which makes it difficult to learn about states that are not verbs. An example is the word 'severe' in the sentence 'If the alarm is severe, display flashing warning.' is a state that is an adjective but is classified as a non-model element.
10. *Compound nouns* Often words in compound nouns are incorrectly classified as model elements such as components and actors. This can be because each word in the compound noun is identified as a different model element. For example, in the sentence 'tracking information is updated.', the word 'information' is a component. However, the phrase 'tracking information' is a compound noun, which makes the classifier identify 'tracking' as a component even though it is not a component according to the annotation guidelines. Although this incorrect prediction can affect the F_1-measure, it does not affect the creation of transition rules because the entire model element is correctly detected without requiring a further analysis of dependencies among words in the sentence and model elements.
11. *Missing actors* The sentences with missing actors often resulted in incorrect classification of model elements. For example, in the sentence 'The status of ambulances can be changed.', the word 'changed' is a transition condition because the action is performed by the actor 'Dispatcher.' However, 'changed' is identified as a state as the word 'Dispatcher' is not included in the sentence. (This can be done through inference.)

Recommendations to reduce errors Our error analysis indicates that most prediction errors of neural networks come from homographs, compound nouns, and improper learning of context. The problem with homographs can be addressed by using context-based embeddings such as ELMo [86], which use different embeddings for homographs based on their context. The problem with improper learning of context can be addressed by using variations of recurrent neural networks over other types and using more data sets,

Effect of out of vocabulary words The data we used in our study has a total of 8435 words. These 8435 words originated from a vocabulary of 770 unique words. Out of 770 words, 53 words are not present in the pre-trained GloVe embeddings. These 53 words occurred a total of 188 times in the data. Some examples of out of vocabulary words in our data are: smart-phone, CI_11, Post-conditions, communitary, and datahosting. While most of the out of vocabulary words did not affect our results, those out of vocabulary words that are model elements are misidentified as non-model elements. For example, the word 'smart-phone' is a component in our data, but it is identified as a non-model element.

We can address the problem of out of vocabulary words by generating word-embeddings that are specific to the data used rather than using pre-trained embeddings. However, this might affect the generalizability of the classifiers. An approach we can recommend for a small amount of training data is to add the data to a large corpus and retrain word embeddings. Moreover, using domain adaptation techniques can also enhance results. Another solution is to use character embeddings and append them to word embeddings as a part of training. In addition to these solutions, the recently proposed ELMo [86] embeddings are considered to have the ability to handle out of vocabulary words.

7.3 Amount of manually labeled unseen data and its effect on performance of classifiers

To use an inter-document splitting method and to meet the assumptions of the principle of identical and independently distributed (iid) data [101, 102], we need a large amount of data. However, generating a large amount of labeled training data can be challenging in industry. Thus, we further analyzed the relationship between the percentage of unseen data added to training data and its outcome to identify a threshold that can produce acceptable and reliable results on unseen data. To perform this analysis, we combined five use case documents to form training data except the unseen document (use case document 6). We used 10% of the training data as our validation data. We divided the unseen data document into 10 parts. We calculated the performance of classifiers while incrementally adding the data of unseen data trained using RNN with GRU and CRF with combined predictions.

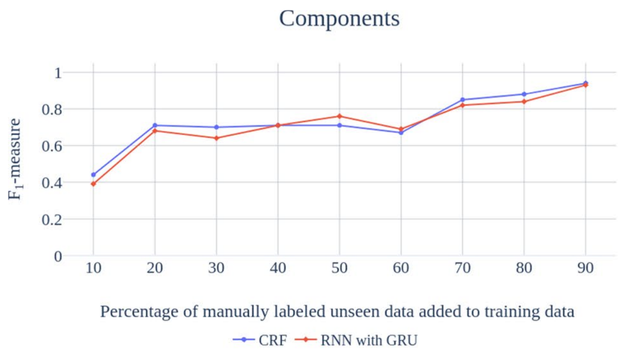
The results of our analysis are shown in Table 18. Each cell in the table denotes the F_1-measure value produced for a given type of model element, type of machine learning algorithm, and a given percentage of unseen data added to training data. Overall, for both CRF and RNN with GRU, the F_1-measures increased with the size of unseen data added to the training data for all model element types except for actors, for which RNN with GRU has the maximum F_1-measure at 30% of addition of manually labeled data. For the percentages greater than 30%, there is a decrease in F_1-measures. The graphs representing these trends can be seen in Fig. 10.

Components For components, to reach a performance equivalent to humans, F_1-measure must be close to the inter-annotation score of components, which is 0.79. We observe from Table 18 and Fig. 10a that for CRF and RNN with GRU to perform on par or better than human annotators, at least 70% of the unseen data must be added to the training data. We attribute these results to the fact that we have a small size of data. We also observe that when the percentage of unseen data changes from 40 to 50%, RNN with GRU outperformed CRF, but in the rest of the cases, CRF outperformed. Further, for some model elements, we noticed that F_1-measures increased by 30% using 20% of unseen data for both the machine learning techniques by 30% (e.g., F_1-measure of 0.39 at 10% to 0.68 at 20% for RNN with GRU).

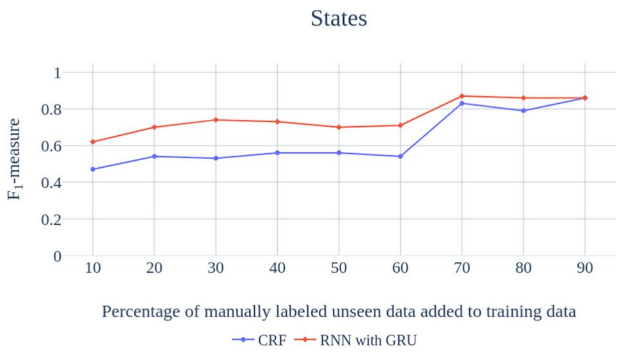
Table 18 Percentage of unseen data used in the data on which the classifiers are trained and the corresponding results of classifiers on remaining unseen data in (F_1-measure)

	Alg. type	Model element type	% of unseen data manually labeled and added to training data								
			10%	20%	30%	40%	50%	60%	70%	80%	90%
F_1 values on remaining data	CRF	Component	0.44	0.71	0.70	0.71	0.71	0.67	**0.85**	0.88	0.94
		State	0.47	0.54	0.53	0.56	0.56	0.54	**0.83**	0.79	0.86
		Actor	**0.92**	0.95	0.96	0.97	0.99	0.99	0.99	1.00	1.00
		Transition condition	0.65	0.71	0.68	0.68	0.71	0.68	0.72	**0.96**	0.95
		Not a model element	0.92	0.94	0.94	0.94	0.94	0.94	0.93	0.97	0.97
	RNN with GRU	Component	0.39	0.68	0.64	0.71	0.76	0.69	**0.82**	0.84	0.93
		State	0.62	0.70	**0.74**	0.73	0.70	0.71	0.87	0.86	0.86
		Actor	0.90	**0.92**	0.95	0.93	0.94	0.92	0.91	0.90	0.93
		Transition condition	0.63	0.71	0.70	**0.86**	0.79	0.80	0.84	0.96	0.96
		Not a model element	0.91	0.94	0.93	0.94	0.94	0.93	0.96	0.96	0.97

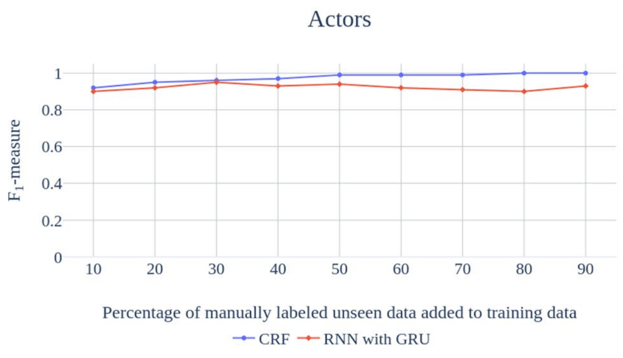
The bold values highlights the F_1 - measures which are on par or greater than inter-annotator agreement. The percentage of the data corresponding to the bolded values represent the minimum percentage of manually annotated data that needs to be added to reach a performance similar to or greater than inter-annotator agreement



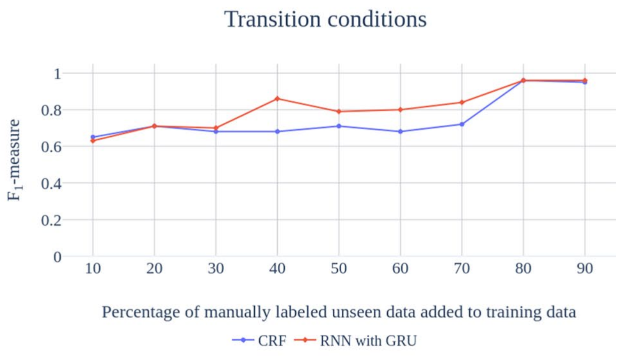
(a) Components



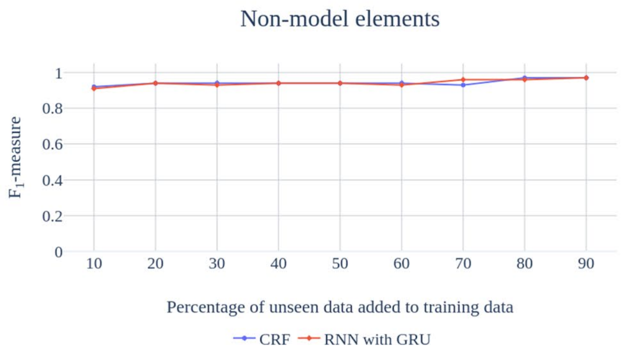
(b) States



(c) Actors



(d) Transition conditions



(e) Words that are not model elements

Fig. 10 Trends of F_1-measures of model elements based on amount of manually labeled unseen data added to training data

States For states, we can observe from Table 18 and Fig. 10b that RNN with GRU outperform CRF for all percentages of additions of unseen data to training data. Compared to human annotation score of states (0.75), we can achieve an F_1-measure of 0.74 for RNN with GRU by adding only 30% of manually labeled data, while for CRF, we need to add at least 70% of data to achieve a score beyond the human annotation score.

Actors In the case of actors, as shown in Table 18 and Fig. 10c, the value of F_1-measure for CRF increased as the percentage of unseen data is increased. However, for RNN with GRU, the maximum F_1-measure is obtained when 30% of labeled unseen data is added. Beyond 30%, F_1-measure decreased. Comparing the F_1-measure values to human annotation agreement on actors (0.91), we can see that CRF and RNN with GRU are able to outperform human annotations with the addition of 10% and 20% manually labeled unseen data to training data, respectively.

Transition conditions For transition conditions, we found that RNN with GRU performed better than or similar to CRF for each addition of unseen data. The results (Table 18 and Fig. 10d) also show that the F_1-measure of RNN with GRU increased up to 40% addition of the data and then decreased slightly up to 70% addition of unseen data. At 80% addition, its performance reached the highest F_1-measure. On the other hand, in the case of CRF, the F_1-measure varied between 0.65 and 0.72 for 10–70% addition of the data. It produced the highest F_1-measure when 80% of the manually labeled unseen data is added. When we compare the F_1-measures of techniques to human annotation value of 0.74, CRF outperforms human annotators when we added 80% of manually labeled unseen data. RNN with GRU, on the other hand, outperformed human annotators when we added 40% of manually labeled unseen data.

Words that are not model elements Similar to all model element types, we also investigated the trends of F_1-measures of classifiers for words that are not model elements. As shown in Table 18 and Fig. 10e, the F_1-measures of both the classifiers are always greater than 0.9. For the percentage of addition ranging from 20 to 60%, the F_1-measures for both the classifiers lie between 0.93 and 0.94. The values increased to 0.96 for RNN with GRU with a 70% addition of the data and to 0.97 for CRF with an 80% addition of data.

From our analysis, we can conclude that the amount of unseen document that needs to be manually labeled for effective predictions can vary with the choice of machine learning algorithms, the type of model elements, and the size of the data. The results also indicate that actors are relatively easy to predict but predicting components can be challenging. We can say that for components, we need more manually labeled unseen data when compared to other model elements to produce a similar or better performance result than human annotators.

7.4 Limitations

Our study contains several limitations. First, we did not handle out-of-vocabulary words (words without word embeddings) in our study. We plan to address this by using ELMo [86] embeddings, which can handle the out-of-vocabulary words. Second, we only considered use case documents and did not consider domain-specificity. We plan to address these issues by annotating user stories and unstructured natural language requirements and by using domain-adaptation techniques. Third, we did not perform statistical significance test because the number of data points for the model elements are smaller than the number of data points for non-model elements. We plan to address this limitation by generating more data points and performing statistical tests on those collected data points. Fourth, we did not have practitioners conduct a comprehensive evaluation of our approach in an industrial setting. We plan to collaborate with industrial partners in the near future in order to evaluate out approach in an industrial setting.

8 Conclusion and future research

This paper proposes various novel factors specific to requirements analysis that can affect the performance of neural network architectures for automated model element identification, and empirically evaluates different neural network architectures on automated model element identification. We analyzed four different types of neural networks (FF, CNN, RNN with LSTM, and RNN with GRU) and investigated the trade-offs among them using six use case documents. We also considered various factors such as types of splitting, design, predictions, and annotations to analyze the impact of these factors on the performance of neural networks in finding model elements. Our results show that each type of model elements is affected by these factors, and that RNN with GRU performed the best when compared to other neural network architectures.

Our future research will address the limitations discussed in Sect. 7 and the threats to our study discussed in Sect. 6. For example, we plan to collect more data points and perform statistical tests to determine which neural networks will be more likely to provide better results. Currently, we limited our analysis to use case documents. We plan to extend our study to documents with unstructured natural language requirements and to user stories. In this study, we concentrated only on automated model element identification but not the complete model generation. We plan to add automatic relation extraction among model elements, thereby constructing the component state transition models.

Acknowledgements This work was supported, in part, by NSF CAREER Award CCF-1564238 to University of North Texas.

References

1. Maiden NAM, Jones SV, Manning S, Greenwood J, Renou L (2004) Model-driven requirements engineering: synchronising models in an air traffic management case study. In: Anne P, Janis S (eds) Advanced information systems engineering. Springer, Berlin, pp 368–383
2. Baudry B, Nebut C, Traon YL (2007) Model-driven engineering for requirements analysis. In: 11th IEEE international enterprise distributed object computing conference (EDOC 2007), pp 459–459
3. dos Santos SM, Vrancken J, Verbraeck A (2011) User requirements modeling and analysis of software-intensive systems. J Syst Softw 84(2):328–339
4. Anton AI (1996) Goal-based requirements analysis. In: Proceedings of the second international conference on requirements engineering, pp 136–144
5. Mylopoulos J, Chung L, Eric Y (1999) From object-oriented to goal-oriented requirements analysis. Commun ACM 42(1):31–37
6. Horkoff J, Eric Y (2016) Interactive goal model analysis for early requirements engineering. Requir Eng 21(1):29–61

7. Piras L, Paja E, Giorgini P, Mylopoulos J (2017) Goal models for acceptance requirements analysis and gamification design. In: Heinrich CM, Giancarlo G, Hui M, Oscar P (eds) Conceptual modeling. Springer, Cham, pp 223–230
8. Madala K, Do H, Aceituna D (2018) A combinatorial approach for exposing off-nominal behaviors. In: 2018 IEEE/ACM 40th international conference on software engineering (ICSE), pp 910–920
9. Li T, Horkoff J, Mylopoulos J (2018) Holistic security requirements analysis for socio-technical systems. Softw Syst Model 17(4):1253–1285
10. Baltes S, Diehl S (2014) Sketches and diagrams in practice. In: Proceedings of the 22Nd ACM SIGSOFT international symposium on foundations of software engineering, FSE 2014, New York, NY, USA, ACM, pp 530–541
11. Lucassen G, Robeer M, Dalpiaz F, van der Werf JMEM, Brinkkemper S (2017) Extracting conceptual models from user stories with visual narrator. Requir Eng 22(3):339–358
12. Dalpiaz F, van der Schalk I, Brinkkemper S, Aydemir FB, Lucassen G (2018) Detecting terminological ambiguity in user stories: tool and experimentation. Inf Softw Technol 110:3–16
13. Echeverría J, Pérez F, Pastor Ó, Cetina C (2018) Assessing the performance of automated model extraction rules. In: Nearchos P, Marios R, Chris B, Michael L, Henry L, Christoph S (eds) Advances in information systems development. Springer, Cham, pp 33–49
14. Madala K, Piparia S, Do H, Bryce R (2018) Finding component state transition model elements using neural networks: an empirical study. In: 2018 5th international workshop on artificial intelligence for requirements engineering (AIRE), pp 54–61
15. Ratba SS, Ghoshal B (2018) Automatic extraction of structural model from semi structured software requirement specification. In: 2018 IEEE/ACIS 17th international conference on computer and information science (ICIS), pp 543–58
16. Elallaoui M, Nafil K, Touahni R (2018) Automatic transformation of user stories into UML use case diagrams using NLP techniques. Proc Comput Sci 130: 42–49. In: The 9th international conference on ambient systems, networks and technologies (ANT 2018)/The 8th international conference on sustainable energy information technology (SEIT-2018)/affiliated workshops
17. Robeer M, Lucassen G, Werf JMEM, Dalpiaz F, Brinkkemper S (2016) Automated extraction of conceptual models from user stories via NLP. In: 2016 IEEE 24th international requirements engineering conference (RE), pp 196–205
18. Harmain HM, Gaizauskas R (2003) Cm-builder: a natural language-based case tool for object-oriented analysis. Autom Softw Eng 10(2):157–181
19. Omar N, Hanna JRP, McKevitt P (2004) Heuristic-based entity-relationship modelling through natural language processing. In: Artificial intelligence and cognitive science conference (AICS). Artificial Intelligence Association of Ireland (AIAI), pp 302–313
20. Vidya S, Vidhu BR, Abirami S (2014) Conceptual modeling of natural language functional requirements. J Syst Softw 88:25–41
21. Peter Pin-Shan Chen (1983) English sentence structure and entity-relationship diagrams. Inform Sci 29(2):127–149
22. Madala K, Gaither D, Nielsen R, Do H (2017) Automated identification of component state transition model elements from requirements. In: 2017 IEEE 25th international requirements engineering conference workshops (REW), pp 386–392
23. Tong H, Liu B, Wang S (2018) Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Inf Softw Technol 96:94–111
24. Tetko IV, Livingstone DJ, Luik AI (1995) Neural network studies. 1. Comparison of overfitting and overtraining. J Chem Inf Comput Sci 35(5):826–833
25. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp 249–256
26. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
27. Kalchbrenner N, Grefenstette E, Blunsom P (2014) A convolutional neural network for modelling sentences. In: Proceedings of the 52nd annual meeting of the association for computational linguistics (Volume 1: Long Papers), vol 1, pp 655–665
28. Kim Y (2014) Convolutional neural networks for sentence classification. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp 1746–1751
29. Sutskever I, Vinyals O, Le Quoc V (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp 3104–3112
30. Ma X, Hovy E (2016) End-to-end sequence labeling via bi-directional LSTM-CNNS-CRF. In: Proceedings of the 54th annual meeting of the association for computational linguistics (Volume 1: Long Papers), vol 1, pp 1064–1074
31. Chung J, Gulcehre C, Cho K, Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS 2014 workshop on deep learning, December 2014
32. Jozefowicz R, Zaremba W, Sutskever I (2015) An empirical exploration of recurrent network architectures. In: International conference on machine learning, pp 2342–2350
33. Sutton C, McCallum A et al (2012) An introduction to conditional random fields. Found Trends Mach Learn 4(4):267–373
34. Witten IH, Frank E, Hall MA, Pal CJ (2016) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Burlington
35. Miwa M, Bansal M (2016) End-to-end relation extraction using LSTMS on sequences and tree structures. In: Proceedings of the 54th annual meeting of the association for computational linguistics (Volume 1: Long Papers), pp 1105–1116
36. Chiu JPC, Nichols E (2016) Named entity recognition with bidirectional LSTM-CNNS. Trans Assoc Comput Linguist 4:357–370
37. Wang S, Jiang J (2016) Learning natural language inference with lstm. In: Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies, pp 1442–1451
38. Yang Z, Dai Z, Yang Y, Carbonell J, Salakhutdinov RR, Le Quoc V (2019) Xlnet: generalized autoregressive pretraining for language understanding. In: Advances in neural information processing systems, pp 5754–5764
39. Devlin J, Chang M-W, Lee K, Toutanova K (2019) Bert: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, Volume 1 (Long and Short Papers), pp 4171–4186
40. Yue T, Briand LC, Labiche Y (2015) Atoucan: An automated framework to derive UML analysis models from use case models. ACM Trans Softw Eng Methodol 24(3):13:1–13:52
41. Thakur JS, Gupta A (2016) Anmodeler: a tool for generating domain models from textual specifications. In: 2016 31st IEEE/ACM international conference on automated software engineering (ASE), pp 828–833
42. Gutiérrez JJ, Nebut C, Escalona MJ, Mejías M, Ramos IM (2008) Visualization of use cases through automatically generated activity diagrams. In: Krzysztof C, Ileana O, Jean-Michel B, Axel U, Markus V (eds) Model driven engineering languages and systems. Springer, Berlin, pp 83–96

43. Zeni N, Kiyavitskaya N, Mich L, Cordy JR, Mylopoulos J (2015) Gaiust: supporting the extraction of rights and obligations for regulatory compliance. Requir Eng 20(1):1–22
44. Ideal modeling and diagramming tool for agile team collaboration. https://www.visual-paradigm.com/
45. Ravenflow. http://www.ravenflow.com/
46. Casecomplete. http://casecomplete.com/
47. Petrov S, Das D, McDonald R (2012) A universal part-of-speech tagset. In: Proceedings of the eighth international conference on language resources and evaluation (LREC-2012)
48. De Marneffe M-C, Manning CD (2008) The Stanford typed dependencies representation. In: Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation. Association for Computational Linguistics, pp 1–8
49. Manning C, Surdeanu M, Bauer J, Finkel J, Bethard S, McClosky D (2014) The Stanford Corenlp natural language processing toolkit. In: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, pp 55–60
50. Hunston S, Francis G (2000) Pattern grammar: a corpus-driven approach to the lexical grammar of English, vol 4. John Benjamins Publishing, Amsterdam
51. Pudlitz F, Brokhausen F, Vogelsang A (2019) Extraction of system states from natural language requirements. In: 2019 IEEE 27th international requirements engineering conference (RE), IEEE, pp 211–222
52. Sleimi A, Sannier N, Sabetzadeh M, Briand L, Dann J (2018) Automated extraction of semantic legal metadata using natural language processing. In: 2018 IEEE 26th international requirements engineering conference (RE). IEEE, pp 124–135
53. Śmiałek M, Kalnins A, Kalnina E, Ambroziewicz A, Straszak T, Wolter K (2010) Comprehensive system for systematic case-driven software reuse. In: van Leeuwen J, Muscholl A, Peleg D, Pokorný J, Rumpe B (eds) SOFSEM 2010: theory and practice of computer science. Springer, Berlin, pp 697–708
54. Elallaoui M, Nafil K, Touahni R (2015) Automatic generation of UML sequence diagrams from user stories in scrum process. In: 2015 10th international conference on intelligent systems: theories and applications (SITA), pp 1–6
55. Miaek M, Straszak T (2012) Facilitating transition from requirements to code with the redseeds tool. In: 2012 20th IEEE international requirements engineering conference (RE), pp 321–322
56. Erazo L, Martins E, Greghi JG (June 2017) Maritaca: from textual use case descriptions to behavior models. In: 2017 47th annual IEEE/IFIP international conference on dependable systems and networks workshops (DSN-W), pp 83–90
57. Song H, Huang G, Chauvel F, Zhang W, Sun Y, Shao W, Mei H (2011) Instant and incremental QVT transformation for runtime models. In: International conference on model driven engineering languages and systems, Springer, New York, pp 273–288
58. Lucassen G, Dalpiaz F, van der Werf JMEM, Brinkkemper S (2015) Forging high-quality user stories: towards a discipline for agile requirements. In: 2015 IEEE 23rd international requirements engineering conference (RE), IEEE, pp 126–135
59. Slob G-J, Dalpiaz F, Brinkkemper S, Garm L (2018) Effective requirements exploration and discussion through visualization. In: REFSQ workshops, The interactive narrator tool
60. Zhang N, Wang J, Ma Y (2018) Mining domain knowledge on service goals from textual service descriptions. IEEE Trans Serv Comput 1–1
61. Chen Y, Wang Y, Hou Y, Wang Y (2019) T-star: a text-based istar modeling tool. In: 2019 IEEE 27th international requirements engineering conference (RE), pp 490–491
62. Bengio Y, Grandvalet Y (2004) No unbiased estimator of the variance of k-fold cross-validation. J Mach Learn Res 5:1089–1105
63. Aceituna D, Do H (2015) Exposing the susceptibility of off-nominal behaviors in reactive system requirements. In: 2015 IEEE 23rd international requirements engineering conference (RE), pp 136–145
64. Pustejovsky J, Stubbs A (2012) Natural language annotation for machine learning: a guide to corpus-building for applications. O'Reilly Media Inc, New York
65. Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20(1):37–46
66. Ben-David A (2008) About the relationship between ROC curves and Cohen's Kappa. Eng Appl Artif Intell 21(6):874–882
67. Powers DMW (2011) Evaluation: from precision, recall and f-measure to ROC, informedness, markedness and correlation. J Mach Learn Technol 2:37–63
68. Veronis J (1998) A study of polysemy judgements and inter-annotator agreement. In: Programme and advanced papers of the senseval workshop, Herstmonceux, pp 2–4
69. Artstein R, Poesio M (2008) Inter-coder agreement for computational linguistics. Comput Linguist 34(4):555–596
70. Petra Saskia Bayerl and Karsten Ingmar Paul (2011) What determines inter-coder agreement in manual annotations? A meta-analytic investigation. Comput Linguist 37(4):699–725
71. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds) Advances in neural information processing systems, vol 25. Curran Associates Inc, New York, pp 1097–1105
72. Chen L, Papandreou G, Kokkinos I, Murphy K, Yuille AL (2018) Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE Trans Pattern Anal Mach Intell 40(4):834–848
73. Lai S, Xu L, Liu K, Zhao J (2015) Recurrent convolutional neural networks for text classification. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence, AAAI'15. AAAI Press, pp 2267–2273
74. Kim Y (2014) Convolutional neural networks for sentence classification. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). Association for Computational Linguistics, pp 1746–1751
75. Mandic DP, Chambers J (2001) Recurrent neural networks for prediction: learning algorithms. Architectures and stability. Wiley, New York
76. Ma X, Hovy E (2016) End-to-end sequence labeling via bi-directional LSTM-CNNS-CRF. In: Proceedings of the 54th annual meeting of the association for computational linguistics (Volume 1: Long Papers), Association for Computational Linguistics, pp 1064–1074
77. Zaheer M, Ahmed A, Smola AJ (2017) Latent LSTM allocation: joint clustering and non-linear dynamic modeling of sequence data. In: Proceedings of the 34th international conference on machine learning, volume 70 of proceedings of machine learning research, international convention centre, Sydney, Australia, PMLR, pp 3967–3976
78. Reimers N, Gurevych I (2017) Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In: Proceedings of the 2017 conference on empirical methods in natural language processing. Association for Computational Linguistics, pp 338–348
79. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ (eds) Advances in neural information processing systems, vol 26. Curran Associates Inc, New York, pp 3111–3119
80. Wikimedia. Wikipedia data set. https://dumps.wikimedia.org/backup-index.html

81. Common Crawl. Common crawl corpus. http://commoncrawl.org/the-data/
82. Pennington J, Socher R, Manning CD (2014) Glove: global vectors for word representation. In: Empirical methods in natural language processing (EMNLP), pp 1532–1543
83. Levy O, Goldberg Y (2014) Dependency-based word embeddings. In: Proceedings of the 52nd annual meeting of the association for computational linguistics (Volume 2: Short Papers), vol 2, pp 302–308
84. Bojanowski P, Grave E, Joulin A, Mikolov T (2017) Enriching word vectors with subword information. Trans Assoc Comput Linguist 5:135–146
85. Joulin A, Grave E, Bojanowski P, Mikolov T (2017) Bag of tricks for efficient text classification. In: Proceedings of the 15th conference of the European chapter of the association for computational linguistics: Volume 2, short papers. Association for Computational Linguistics, pp 427–431
86. Peters M, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L (2018) Deep contextualized word representations. In: Proceedings of the 2018 conference of the North American chapter of the association for computational linguistics: human language technologies, Volume 1 (Long Papers). Association for Computational Linguistics, pp 2227–2237
87. Ramshaw LA, Marcus MP (1999) Text chunking using transformation-based learning. In: Natural language processing using very large corpora, Springer, New York, pp 157–176
88. Van Halteren H (2000) Chunking with WPDV models. In: Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on computational natural language learning-Volume 7. Association for Computational Linguistics, pp 154–156
89. Chollet F et al (2015) Keras. https://keras.io
90. Liu S, Sun J, Liu Y, Zhang Y, Wadhwa B, Dong JS, Wang X (2014) Automatic early defects detection in use case documents. In: ASE'14, New York, NY, USA, ACM, pp 785–790
91. Ambulance dispatch system requirements specification. http://www.utdallas.edu/~chung/CS6354/CS6354_U07_source/Team_2/delivarable_2_final.doc
92. SIMON: D2.1 Use case Specification Document. http://simon-project.eu/wp-content/uploads/2014/02/simon_D2_1_Use-case-specification-document_PU_v1.0.pdf, 2014
93. Huang Z, Thint M, Qin Z (2008) Question classification using head words and their hypernyms. In: Proceedings of the conference on empirical methods in natural language processing, EMNLP '08, Stroudsburg, PA, Association for Computational Linguistics, pp 927–936
94. Saito T, Rehmsmeier M (2015) The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PloS One 10(3):e0118432
95. Cunningham H, Tablan V, Roberts A, Bontcheva K (2013) Getting more out of biomedical documents with gate's full lifecycle open source text analytics. PLOS Comput Biol 9(2):1–16
96. Viera AJ, Garrett JM (2005) Understanding interobserver agreement: the kappa statistic. Family Med 37(5):360–363
97. McHugh ML (2012) Interrater reliability: the kappa statistic. Biochem Med 22(3):276–282
98. Kohavi R et al (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. Ijcai. vol 14. Montreal, Canada, pp 1137–1145
99. Domingos P (2012) A few useful things to know about machine learning. Commun ACM 55(10):78–87
100. Mitchell TM (1997) Artificial neural networks. Mach Learn 45:81–127
101. Tillman RE (2009) Structure learning with independent non-identically distributed data. In: Proceedings of the 26th annual international conference on machine learning, ICML'09, New York, NY, USA, ACM, pp 1041–1048
102. Cesa-Bianchi N, Conconi A, Gentile C (2004) On the generalization ability of on-line learning algorithms. IEEE Trans Inf Theory 50(9):2050–2057
103. McCallum AK (2002) Mallet: a machine learning for language toolkit. http://mallet.cs.umass.edu
104. Leaman R, Khare R, Zhiyong L (2015) Challenges in clinical natural language processing for automated disorder normalization. J Biomed Inf 57:28–37
105. Tseytlin E, Mitchell K, Legowski E, Corrigan J, Chavan G, Jacobson RS (2016) Noble—flexible concept recognition for large-scale biomedical natural language processing. BMC Bioinf 17(1):32