Analysis and comparison of deep learning models for user stories identification

Análisis y comparación de modelos de aprendizaje profundo para la identificación de historias de usuario

Francisco Javier Peña Veitía[1] https://orcid.org/0000-0002-0837-2285
María Luciana Roldán[1] https://orcid.org/0000-0002-4786-5592
María Marcela Vegetti[1] https://orcid.org/0000-0003-4016-1717

Recibido 01 de octubre de 2021, aceptado 03 de noviembre de 2023
Received: October 01, 2021 Accepted: November 03, 2023

ABSTRACT

Nowadays, most software companies have adopted agile development methodologies, which suggest the capture of requirements through user stories. Issues Management Systems allow development teams to manage user stories and other issues, such as errors, change requests, and others. Although these systems provide features for categorizing or labeling issue types, the user often needs to include or specify this information correctly. A poor issue categorization causes many user stories to end up buried in a large volume of data, making it difficult to identify them. This article presents and compares three neural network models to classify issues as User Stories. As the ultimate goal of this research is to improve the quality of the software development project documentation, the comparison is practical to select a model to be embedded in an IMS tool for automatically categorizing issues. The compared models are a BRNN-LSTM model, an Elmo-based model, and a BERT-based model. It applied the CRISP-MD methodology to train, validate, and test the three proposed neural network models. Then, a comparison was performed regarding their accuracy and performance. As a result, the article shows that the BERT-based model is the one that best fits the problem posed, managing to classify the issues as user stories with an accuracy of approximately 97%. This model can analyze the text syntactically and semantically with the best accuracy and performance.

Keywords: Natural language processing, machine learning, recurrent neural networks, software engineering, user story.

RESUMEN

Hoy en día, la mayoría de las empresas de software emplean metodologías de desarrollo ágiles, las cuales recomiendan la captura de requisitos a través de historias de usuario. Tanto las historias de usuario, como otros tipos de incidencias (errores, solicitudes de cambio, etc.) son gestionadas mediante Sistemas de Seguimiento de Incidencias (SSI). Aunque estos sistemas poseen características para el etiquetado o categorización de tipos de incidencias, esta información suele ser omitida o especificada incorrectamente por el usuario. Una mala categorización de las incidencias hace que muchas historias de usuario se pierdan en grandes volúmenes de datos, dificultándose así su posterior recuperación.

[1] INGAR - Instituto de Desarrollo y Diseño (CONICET-UTN), Ciudad de Santa Fe, Argentina.
E-mail: fpveitia@santafe-conicet.gov.ar; lroldan@santafe-conicet.gov.ar; mvegetti@santafe-conicet.gov.ar
* Autor de correspondencia: fpveitia@santafe-conicet.gov.ar

Este artículo compara tres modelos de redes neuronales para clasificar incidencias como Historias de Usuario. Siendo el objetivo final de esta investigación la mejora de calidad en la documentación de los proyectos de desarrollo de software, la comparación realizada es útil para la selección del mejor modelo a incorporar en una herramienta SSI para la categorización automática de las incidencias. Los modelos comparados son un modelo BRNN-LSTM, un modelo basado en Elmo y un modelo basado en BERT. Se aplicó la metodología CRISP-MD para entrenar, validar y probar los tres modelos de redes neuronales propuestos. Como resultado, el artículo muestra que el modelo basado en BERT es el que mejor se ajusta al problema planteado, consiguiendo clasificar los problemas como historias de usuario con una precisión de aproximadamente el 97%. Además, dicho modelo es capaz de analizar el texto tanto sintáctica como semánticamente con la mejor precisión y rendimiento.

Palabras clave: Procesamiento del lenguaje natural, aprendizaje automático, redes neuronales recurrentes, ingeniería de software, historias de usuario.

INTRODUCTION

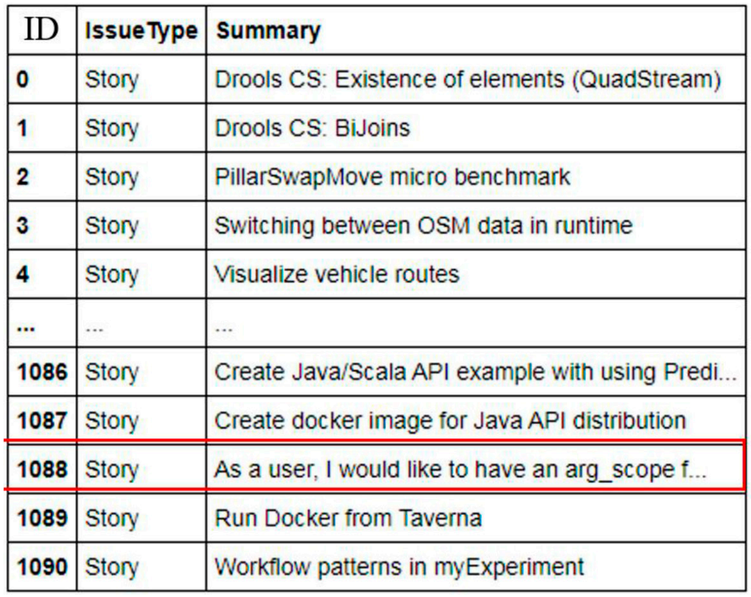
Today, most software development companies have adopted agile development methodologies such as SCRUM, Kanban, and XP. Most of these agile methodologies recommend the capture of requirements through user stories [1]. In this context, a user story is a short description of what some part of the software should do from the perspective of some stakeholder interested in the new feature that the software should provide or possess. Although, over the years, several structures have been proposed for writing user stories, most are now written in a strict and compact way that captures who it is for, what is expected as a system response, and optionally why it is relevant following the structure: "As a (type of user), I want (goal), so that (some reason)" [1].

In agile software development, requirements in the form of user stories are frequently managed in an Issue Management System (IMS). An Issue Management System is a computer application designed to help ensure software quality and support to programmers and other stakeholders in the tracking process. These systems include Jira, OpenProject, and Redmine, among others. An IMS can be configured as an issue tracker, a bug tracker, or a project management tool. Specifically, in agile software development, it is common to employ an IMS as a supporting tool for keeping track of the open development issues in a software project [2].The term "issue" is attributed to the unit of work improve a computer system. Therefore, this term can describe most of the kinds of tasks that are needed to track when developing a computer system [2].

IMS systems allow development teams to organize a collection of user stories in meaningful fragments like epics, themes and sprints. In addition, these systems manage other issues types, such as errors, change requests, and others. Although these systems allow the user to categorize or label an issue explicitly, selecting the right category for a new issue is up to the person creating it. That means that this information needs to be included or assigned correctly. Poorly categorizing issues causes many user stories to be buried in a large volume of data, making it difficult to identify them.

An analysis was performed on a dataset containing more than 1.5 million issues to support this claim [3]. Among other data, for each issue, the issue type and a summary description are stored in the dataset for each case. Using different kinds of string-matching patterns, we have found that a high percentage of issues have an incorrect type assigned, or their summary information needs to be correted. Figure 1 and Figure 2 illustrate the results of two searches performed on the dataset. Figure 1 shows 10 of the 10829 records obtained after filtering all the issues using "story" as the issue type. In Figure 1 it can be seen that although the issues were classified as user stories, only the labeled with ID 1088 comply with the compact format of a user story mentioned above. Also, in Figure 1, it can be seen that many issues that cannot be identified as requirements (issues labeled as 0,1,3, 1089, or 1099, for example).

We performed a second search, looking for the string "as a" in the "Summary" field of the issues. Figure 2 shows the records obtained, whose IDs range from 0 to 3058. This figure, shows that some issues were



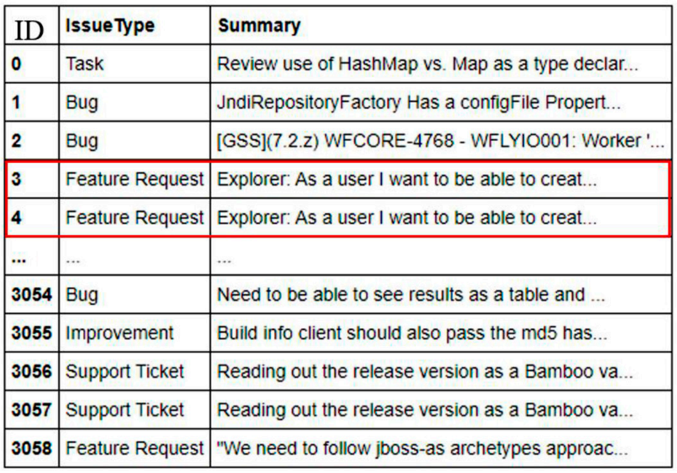
ID	IssueType	Summary
0	Story	Drools CS: Existence of elements (QuadStream)
1	Story	Drools CS: BiJoins
2	Story	PillarSwapMove micro benchmark
3	Story	Switching between OSM data in runtime
4	Story	Visualize vehicle routes
...	...	...
1086	Story	Create Java/Scala API example with using Predi...
1087	Story	Create docker image for Java API distribution
1088	Story	As a user, I would like to have an arg_scope f...
1089	Story	Run Docker from Taverna
1090	Story	Workflow patterns in myExperiment

Figure 1. Issues obtained when filtering with the search pattern “Story” applied to the Issue Type field.

not classified as user stories, although they were expressed using the compact format of a user story (se, for example, issues labeled as 3, and 4).

This preliminary analysis shows that while IMSs are helpful tools to support the management of software development projects, users can assign the wrong issue type or label to an issue or omit that information. Thus, it is necessary to have an efficient approach to classifying issues, which can be integrated into an IMS to provide it with the capabilities to identify the type of issue in an automated way.

Moreover, the correct identification of user stories interests' software engineering for several reasons. For the members of a software project team that employ an IMS, having a supporting tool for



ID	IssueType	Summary
0	Task	Review use of HashMap vs. Map as a type declar...
1	Bug	JndiRepositoryFactory Has a configFile Propert...
2	Bug	[GSS](7.2.z) WFCORE-4768 - WFLYIO001: Worker '...
3	Feature Request	Explorer: As a user I want to be able to creat...
4	Feature Request	Explorer: As a user I want to be able to creat...
...	...	...
3054	Bug	Need to be able to see results as a table and ...
3055	Improvement	Build info client should also pass the md5 has...
3056	Support Ticket	Reading out the release version as a Bamboo va...
3057	Support Ticket	Reading out the release version as a Bamboo va...
3058	Feature Request	"We need to follow jboss-as archetypes approac...

Figure 2. Issues obtained when filtering with the search pattern “as a” applied to the Summary field.

automatically categorizing of issues as user stories can save time and error occurrences, improving the whole quality of the project documentation.

For organizations that have multiple related projects, it is important to have an integrated requirements base. Requirements engineering activities are no longer associated with an individual system development process and, thus, an individual project [3]. In contrast, it is viewed as an independent activity executed across multiple projects and product developments. Therefore, an approach to identify the issues that constitute "user stories" in an IMS repository is helpful to retrieve them and feed an integrated requirements base, regardless of whether they were categorized as "user stories."

A recent research trend is the application of computational linguistic techniques to user stories to solve classic challenges in requirements engineering, such as the formulation of high-quality requirements or the creation of better models of system functionalities [2]. However, the success of these studies strongly depends on the correctness of the categories or labels assigned to the issues in an extensive IMS repository. Therefore, correctly identifying user stories is a starting point for applying these approaches.

This work presents three neural network models with different architectures to classify issues as User Stories. Then, we ran an experiment to evaluate the performance of each model.

This paper is organized as follows. The first section presents some related works. After that, some theoretical concepts of the methods and materials used to better understand this work are introduced. Then, details about the datasets generated and used are discussed, and the implemented models are presented. Subsequently, the main results obtained by testing the different models are described, and a comparison is offered that considers various aspects such as the accuracy of the models, syntactic analysis, and semantic analysis capability. Finally, the conclusions are drawn.

Related works

One of the features provided by most IMS is to assign a category or a set of labels to the generated issues with the aim, at least in theory, to facilitate their management and retrieval. Several authors have studied the use of labels to categorize issues in an IMS. In [4], the authors analyzed a population of more than three million GitHub projects and gave some insights on how labels are used in them. Their results reveal that, even if the label mechanism is scarcely used, using labels favors the resolution of issues. They also conclude that not all projects use labels similarly (e.g., for some, labels are only a way to prioritize the project, while others use them to signal their temporal evolution as they move along in the development workflow).

In a study conducted on closed issue reports of three open-source software systems from Jira, it has been observed that the label given to the issue reports about bugs or improvement is incorrect [5]. The authors manually classified more than 7000 closed issue reports from five popular open-source software systems to analyze the accuracy of already labeled reports. Their findings state that 33.8% of closed issue reports were misclassified.

The authors in [6] manually classified a dataset and applied machine learning algorithms for bug classification. In [7], an automated approach is proposed to label an issue either as a bug or other request based on fuzzy set theory. The labeling of bug reports is done in three phases. First, text from the bug reports is preprocessed. Second, the Fuzzy technique is applied, and third, the labeling is done using scores obtained after fuzzification. In [8], the authors selected seven projects in GitHub and built classification models based on issue information, text descriptions, and comments to improve the maintenance tasks for development teams. Text information was preprocessed with text data mining techniques and information retrieval. Then, they evaluated the performance of classifiers with several metrics. They conclude that very suitable classifiers may be obtained to label the issues or suggest the most suitable candidate labels.

These contributions employed datasets obtained from repositories of IMS configured for bug tracking and not for project management. For that reason, the focus of these works has been on the correct classification/labeling of defects or bugs. However, our work employs datasets obtained from IMS repositories used for project management and software development, and we focused on the identification of issues related to requirements definition, such as “user stories”.

In the last years, a research trend has emerged regarding applying computational linguistic techniques to user stories to solve classic challenges in requirements engineering, such as the formulation of high-quality requirements or the creation of better models of system functionalities [9]. A research line is the extraction of conceptual models from natural language (NL) requirements, which can help to identify dependencies, redundancies, and conflicts between requirements from lengthy textual specifications. To extract meaningful models from requirements expressed in NL, researchers have been proposing heuristic rules for the identification of entities and relationships whenever the text matches particular patterns of the given language (usually English). For example, in [9], is proposed an automated approach based on natural language processing that extracts conceptual models from user story requirements. In another work [10], the authors proposed an approach to generate i* models from user stories. In [11], contributions are made toward mapping user stories and use case models. Also, in [12], user stories are used to extract quality attributes for early architecture decision-making. A common denominator of all these proposals is that they require user stories as input, so mislabeled user stories harm the results of such studies. Consequently, to anticipate better results from these user story studies, an approach that correctly identifies subjects as either “user stories” or “non-user stories” is required.

Background

In this section, the theoretical concepts on which this work is based are exposed. We describe the concepts and models used in this paper, such as User Stories, Recurrent Neural Networks, Bidirectional Long Short-Term Memory Recurrent Neural Networks, and Natural Language Processing with Neural Networks, among others.

User stories

Outside the world of software, a user story could be referred to as a customer’s testimonial or narrative; however, it has a whole different meaning for software professionals. In terms of software development, a user story is a short description of something or a piece of software it is supposed to do, told from the perspective of the person who desires the new feature. Although going back to its beginnings, user stories were proposed as unstructured text but with some size restrictions [1], nowadays, it follows a compact template for writing them. The template captures who it is, its expectations of the system function, and, optionally, why it is significant [13]. Although many different templates exist, 70% of practitioners use the template: “As a (type of user), I want (goal), [so that (some reason)]” [1]. Next, two examples of user stories using such a template are introduced.

- Example 1: As a visitor, I want to purchase an event ticket.
- Example 2: As an event organizer, I want to search for new events by favorited organizers, So that I know of events first.

Natural language processing with neural networks

Natural Language Processing (NPL) is a subfield of linguistic, computer, information engineering, and artificial intelligence sciences dedicated to interacting with computer equipment and human natural language, particularly how computer programs process and analyze large amounts of information. The problems often addressed with these techniques are speech recognition, understanding natural language such as sentiment analysis, text generation, automatic text summarization, and automatic entity recognition[14]. Although there exist several natural language processing techniques, in recent years, there has been a significant boom in the use of Deep Learning models [14] because of their ability to capture the syntactic and semantic information of words in large unlabeled bodies of text. Word vectors (word embeddings) are a standard component found in current NLP system architectures [14]. Word embeddings are vectors of real numbers representing terms correlating relative similarities with semantic similarities [15], generally learned by neural networks. They can represent the context of the word and can provide information about relations with other words. Hence, the meaning or semantic context of words can be predicted accurately as they can capture syntactic and semantic information about the words [16]. Following this trend, there are analyzed and used popular models at the moment of this work.

Recurrent Neural Network (RNN) architectures have become a typical and famous neural network model because of their capabilities to process sequential inputs and learn its dependencies [17], proving to be very helpful in NLP tasks. An RNN is a neural network where the connections between neurons form a directed graph, making a temporal sequence through X_t time steps, feeding each hidden state H_t to the next time step, as shown in Figure 3. The network thus has a dynamic temporal behavior. Unlike common networks, RNNs can use an internal state (memory state) to process sequences of inputs. However, they have problems with long-term dependencies due to gradient vanishing [17].

Otherwise, long short-term memory (LSTM) is a recurrent neural network architecture type that avoids the problem of gradient vanishing. LSTM is augmented by recurrent “forgetting” gates, preventing the backward propagation error from vanishing or exploding. In this type of network errors can go backward through a virtually unlimited number of layers unfolded in space. As shown in Figure 4, the internal memory cell C_t is controlled by a set of gate networks: a forget gate network f_t an input gate network i_t, and an output gate network o_t. The

Source: [17]

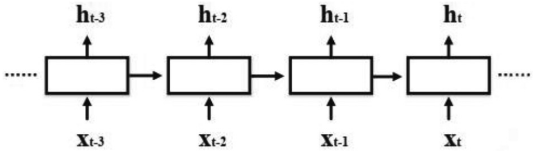


Figure 3. Recurrent Neural Network.

Source: [17]

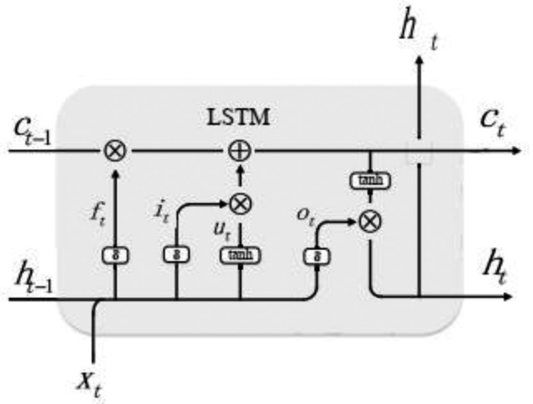


Figure 4. Schematic of the LSTM.

forget gate network controls how much information of the internal cell C_t should be passed into the next time step. The input gate network is used to scale the input block u_t to the internal cell. Consequently, LSTM can learn tasks that require memories of events that occurred thousands of times in previous training steps, thus making it capable of handling long-term dependencies [17].

On the other hand, Bi-directional Recurrent Neuronal Networks (BRNN) have a specific structure. The state neurons of a regular RNN are split into a part that is responsible for the positive time direction (forward states) and a part for the negative time direction (backward states), as shown in Figure 5. These outputs of two types of states are not necessarily connected to inputs in the opposite states [18]. Using time directions in the same network, input information in the past (t-1 in Figure 5) and the future (t+1 in Figure 5) of the currently evaluated time frame (t) can be used to minimize the objective function without the need for delays, unlike common RNN that require these “delays” to include future information. Using the LSTM and BRNN models, the model can handle long-term dependencies and analyze the whole sentence forward and backward [19].

Commonly, nowadays, different NLP tasks entail a great effort in terms of time and computing power consumption, so as an alternative to creating a model from scratch or too general, the transfer learning technology has emerged [20]. Transfer Learning (TL) is a machine learning method with the perspective of providing a better and faster solution with less effort for collecting the needed training information and reusing it in another similar model

Source: [17]

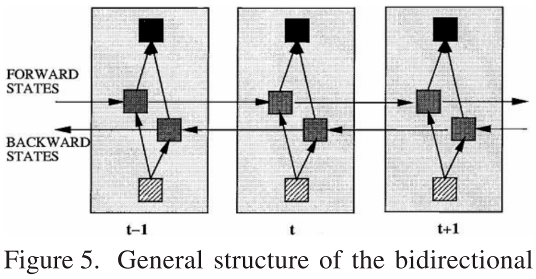


Figure 5. General structure of the bidirectional recurrent neural network (BRNN) shown unfolded in time for three-time steps.

[20]. In [20], it is defined as: "Given a Ds domain and a source Ts learning task, and a Dt domain and target Tt learning task, the TL aims to enhance the learning of the target predictive function f(x) in Dt using the knowledge in Ds and Ts, where Ds ≠ Dt, or Ts ≠Tt." Word embeddings are a good example of transfer learning since neural networks generally learn them in a domain for a learning task, and these learned word embeddings can be applied in a different domain for other learning tasks. Hence, those vectors of real numbers are transferred from one model to another.

Word representations, such as Word Embeddings, are a crucial component in many neural language models [21]. ELMo (Embeddings from Language Models) incorporates a form of deep word representation based on a feature-based approach, where each token is assigned a representation that is a function of the entire input sequence [21]. The vectors derived from a trained LSTM network with a pair of linguistic models are used in an extended text corpus. These representations are a function of all the layers of a Bidirectional Linguistic Model (biLM) [21]. ELMo looks at the entire sentence before assigning each word in its embedding. It uses a bi-directional LSTM trained on a specific task to create contextual word embedding. The ELMo LSTM, once trained on a massive dataset, could be used as a constituent in other NLP models aimed at language modeling. In [22], an implementation of a module with this architecture and an application trained in 1 billion words is presented. This module returns as output a set of fixed embeddings for each LSTM layer, the learned aggregation composed of 3 layers, and a mean-pooled vector representation of the input.

There are two strategies for applying pre-training in linguistic models: the characteristics-based approach and the parameter adjustment approach [23]. Feature-based models such as ELMo [21] use architectures that include pre-trained representations as additional features. On the other hand, models that use parameter resetting introduce parameters to specific tasks, trying to simplify and adjust all the pre-trained parameters. However, current techniques based on the parameter-matching approach use unidirectional linguistic models [23]. BERT (Bidirectional Encoder Representations from Transformers) [23] alleviates this problem using a masked linguistic model. The linguistic model masks some of the input tokens and aims to predict the original id of the vocabulary by linking the contexts from the right and left; hence it is bidirectional.

BERT uses a masked language modeling objective to pre-train the transformer network on an extensive unlabeled data [24]. In [25], it can be found an implementation and examples of the use of a module that fits this architecture trained in Wikipedia and BookCorpus. Assuming that the entries are pre-processed as required by this module implementation, it returns as output representations of each token in the input sequence and an entire grouped representation of the entry.

Attention mechanisms have become an integral part of sequential modeling in various tasks, allowing the modeling of dependencies regardless of the distance between input and output sequences. These mechanisms are generally used with some RNN[26]. These models use the so-called attention functions, which are nothing more than a function that can be described as the mapping of a query and a set of identifier-value pairs to an output, where the query, the identifiers, and the values are all vectors. The output is calculated as the weighted sum of the values, where the weight of each value is calculated by a query compatibility function with the corresponding identifier [26]. In [26], various types of attention functions, such as "Scaled Dot-Product Attention," "Multi-Head Attention," and "Self- Attention," are presented and explained.

A model called "Transformer" [26] is completely based on the Self-Attention and Multi-Head Attention models. This model does not use alienated RNNs or convolutions; it follows an encoder-decoder architecture completely connected between its layers. That means that the encoder maps an input sequence of symbol representations to a continuous representation. Then, the decoder generates an output of the symbols for each element at a time [27].

MATERIAL AND METHODS

This section introduces the models proposed to identify user stories in issue management systems. For the resolution of the problem presented in this work, the CRISP-DM methodology was followed [28]. The main steps of the methodology are listed below:

Business understanding. This initial phase focuses on understanding the problem establishing the data mining goals and the success criteria.

Data understanding. The data understanding phase starts with initial data collection and proceeds with activities to get familiar with the data and to identify data quality problems.

Data preparation. The data preparation phase covers all activities to construct the final dataset, which will be fed into the models.

Modeling. In this phase, various modeling techniques are selected and applied, and their parameters are calibrated to optimal values.

Evaluation. Before proceeding to the final deployment of the model, it is essential to evaluate the model more thoroughly, reviewing its metrics and behavior in the real application.

Deployment. This task takes the evaluation results and concludes a strategy for deployment of the data mining result(s) into the business.

Data understanding and preparation

The models were trained by taking data from public sources containing real software development project problems [29] [30]. These sources contain positive examples of user stories (sentences in the format described previously) and negative examples (erroneous user stories or sentences with a similar syntaxis to user stories but with a different purpose). An algorithm was implemented to generate additional examples by splitting and mixing positive examples into random parts using the TensorFlow Tokenizer to obtain a more extensive data set suitable for testing the models. This implementation is available in [31]. A manual classification work was performed to differentiate the examples to which each classification class belonged, thus introducing into the model an index of human error, given that there was no record of the previously classified data. The resulting dataset includes a total of 7997 positive and negative examples, of which 2618 are positive, as shown in Table 1, and the remainder are negative, as shown in Table 2.

Therefore, a binary classification problem is presented, where the issues classified as user stories belong to the positive class (1) and the rest to the negative class (0). The entire dataset obtained can be found in [32].

A BRNN-LSTM model for User Story issues classification

The first model proposed for User Stories classification is based on an architecture for a bidirectional LSTM neural network (Figure 6). The model has a maximum of Word Embeddings equal to the vocabulary length, with 300 dimensions each and 125 bidirectional LSTM layers. A dropout layer was used to prevent overfitting, and a sigmoid activation function in the output layer.

For the implementation of this model, Python 3 and TensorFlow 2.0.0-rc0 for GPUs were used. The implemented model is available in [33].

An ELMo-based model for User Story issues classification

Furthermore, a custom Keras layer for TensorFlow, whose implementation was taken from [35] and

Table 1. Samples of positive examples in the dataset.

No.	Issue	Class
1	As a Carequality implementer, I want CONNECT to leverage the Carequality framework so I can exchange with other Carequality participants	1
2	As a Carequality implementer, I want CONNECT to leverage the Carequality framework so I can exchange with other Carequality participants	1
3	As a CONNECT administrator, I want CONNECT to push audits and events via web services	1
4	As a CONNECT Adopter I need CONNECT to be database independent and support different databases such as Oracle	1
5	As a CONNECT Adapter, I want to be able to respond to requests and receive responses to requests asynchronously in addition to synchronously	1
6	As a CONNECT Adapter, I want to be able to respond to requests and receive responses to requests asynchronously in addition to synchronously	1

Table 2. Samples of negative examples in the dataset.

No.	Issue	Class
1	Add enable/disable exchange refresh function to Exchange Manager GUI	0
2	Add details should anchor tag you back to the expanded section that you added from.	0
3	Add JUnit tests for mail classes for Mail package	0
4	i want to take a dataset offline so that i can perform a long running maintenance or migration procedure	0
5	as a url to social networks so that i can	0
6	as necessary including title dates languages and other facets	0

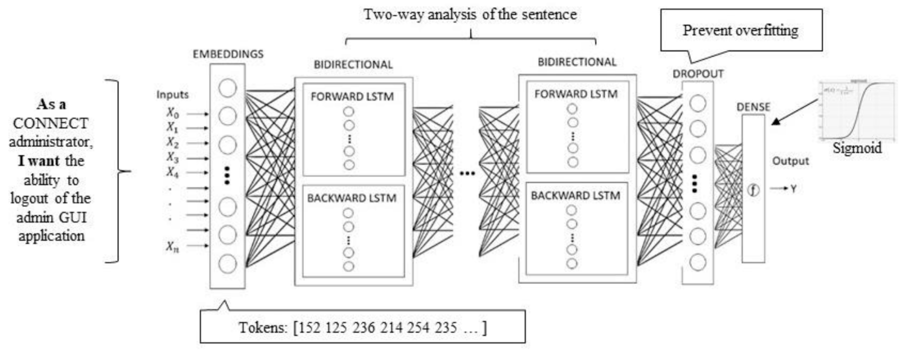

Two-way analysis of the sentence
Prevent overfitting
EMBEDDINGS
BIDIRECTIONAL
BIDIRECTIONAL
DROPOUT
As a CONNECT administrator, I want the ability to logout of the admin GUI application
Inputs
FORWARD LSTM
BACKWARD LSTM
FORWARD LSTM
BACKWARD LSTM
DENSE
Output
Sigmoid
Tokens: [152 125 236 214 254 235 …]

Figure 6. The BRNN-LSTM model.

subsequently integrated and adapted to our model, was used to build the ELMo-based model using the ELMo2 module available through the Tensorflow Hub platform [34]. Besides, a dropout layer was added to the model to prevent overfitting and a sigmoid activation function. Figure 7 illustrates a general view of the sequential model using the ELMo module.

For this implementation, Tensor Flow 1.14 was used due to support and compatibility problems of the module with TensorFlow2.0 and the Tensor Flow-Hub library [36]. The model implementation is available in [37].

A BERT-based model for User Story issues classification

The BERT module **bert_uncased_L-12_H-768_A-12/1**, available through the Tensorflow Hub platform [34], which provides a simple way to share Tensorflow models, was used to implement the BERT-based model. We used Keras with Tensorflow backend to build our BERT-based model. Before Keras can use the core TensorFlow model, a customized Keras layer must be defined to render it in the appropriate format [38] correctly.

As shown in Figure 8, after the inputs are preprocessed, the ids for the tokens and their respective masks are obtained, which fed the BERT layer. Finally, to avoid overfitting, a dropout layer is placed at the output of the BERT module, and subsequently, a sigmoid function is used.

The implementation of this model used Tensor Flow 1.14 to avoid some compatibility issues with TensorFlow2.0 and the Tensor Flow-Hub library. The implemented model is available in [39].

EVALUATION AND DISCUSSION OF THE RESULTS

In this section, the main results obtained by the different models are presented, and then a comparison is made between them. For each model, once the dataset was loaded, it was randomly divided into 70% for training and 30% for testing using the function **train_test_split** from [40]. Also, during training, the 70% was divided again into 30% for

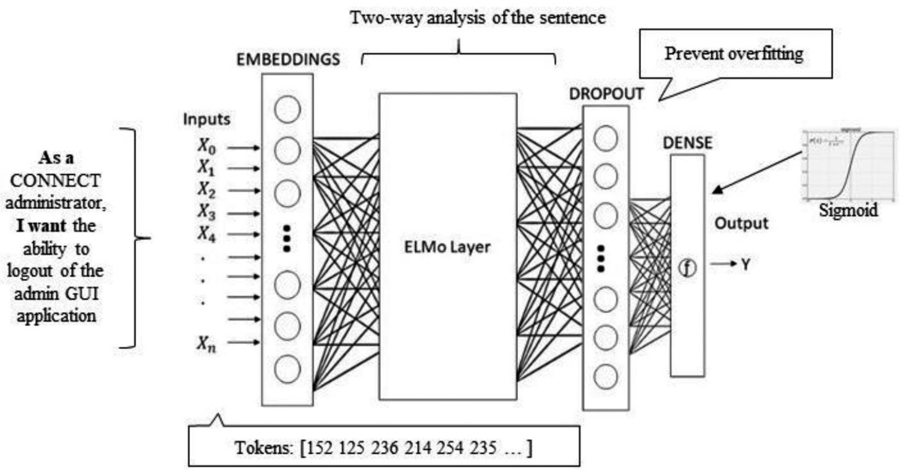

Two-way analysis of the sentence
EMBEDDINGS
Prevent overfitting
DROPOUT
Inputs
DENSE
As a CONNECT administrator, I want the ability to logout of the admin GUI application
ELMo Layer
Output
Sigmoid
Tokens: [152 125 236 214 254 235 …]

Figure 7. The proposed model using ELMo module.

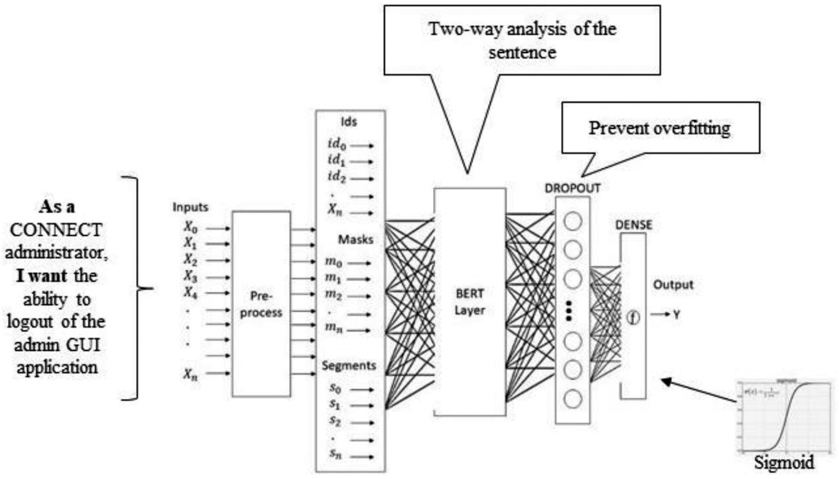

Two-way analysis of the sentence
Prevent overfitting
DROPOUT
DENSE
Inputs
Ids
Masks
Segments
Pre-process
BERT Layer
Output
As a CONNECT administrator, I want the ability to logout of the admin GUI application
Sigmoid

Figure 8. The proposed model using BERT module.

validation using the **validation_split** parameter available when training TensorFlow models.

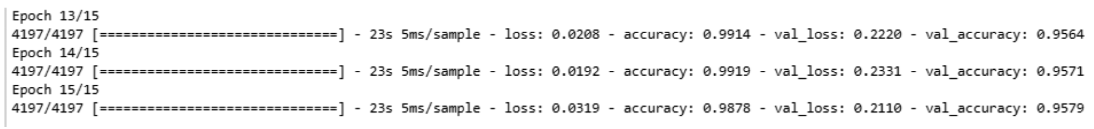
Training and validating the models

For the BRNN-LSTM model, an Adam optimizer with a learning rate of 0.01 and a batch size of 35 was used, as shown in Figure 9 after 15 epochs of 23 seconds each, accuracies of 0.9579 and 0.9624 were achieved in validation and testing, respectively.

In Figure 10, it is analyzed the performance graphs by accuracy and loss. As it can be observed, there exist some overfitting, which could lead to missing classifications.

For validating the automatic generation algorithm, the same training and validation process was run for the BRNN-LSTM model using the original dataset. After that, the F1 score was 0.8796 against 0.9545 for the model trained using the enhanced dataset, as shown in Figure 11. As can be seen, the model using the enhanced dataset has a better score; hence, it was decided to continue the training and validation of the rest of the models using the enhanced data set only.

For the ELMo-based model, an SGD optimizer and a batch size of 35 were used, and after 34 epochs of 23 seconds each, an accuracy of 0.9607 in validation was obtained, as shown in Figure 12.



```
Epoch 13/15
4197/4197 [==============================] - 23s 5ms/sample - loss: 0.0208 - accuracy: 0.9914 - val_loss: 0.2220 - val_accuracy: 0.9564
Epoch 14/15
4197/4197 [==============================] - 23s 5ms/sample - loss: 0.0192 - accuracy: 0.9919 - val_loss: 0.2331 - val_accuracy: 0.9571
Epoch 15/15
4197/4197 [==============================] - 23s 5ms/sample - loss: 0.0319 - accuracy: 0.9878 - val_loss: 0.2110 - val_accuracy: 0.9579
```

Figure 9. Training of the BRNN-LSTM model.

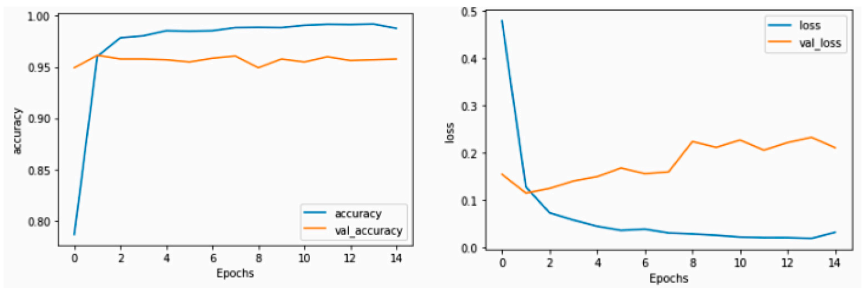
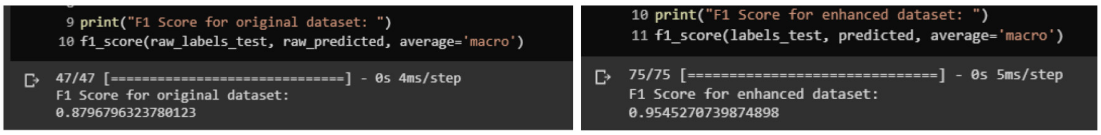


Figure 10. Accuracy and Loss analysis of the BRNN-LSTM model.



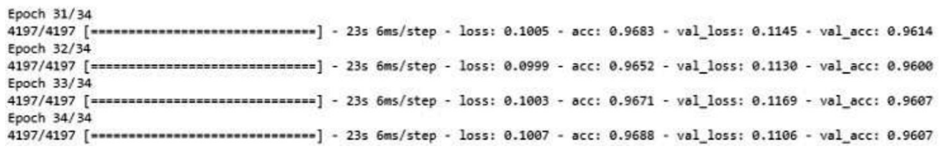
```
print("F1 Score for original dataset: ")
f1_score(raw_labels_test, raw_predicted, average='macro')

47/47 [==============================] - 0s 4ms/step
F1 Score for original dataset:
0.8796796323780123
```

```
print("F1 Score for enhanced dataset: ")
f1_score(labels_test, predicted, average='macro')

75/75 [==============================] - 0s 5ms/step
F1 Score for enhanced dataset:
0.9545270739874898
```

Figure 11. Training BRNN-LSTM model with original vs enhanced dataset.



```
Epoch 31/34
4197/4197 [==============================] - 23s 6ms/step - loss: 0.1005 - acc: 0.9683 - val_loss: 0.1145 - val_acc: 0.9614
Epoch 32/34
4197/4197 [==============================] - 23s 6ms/step - loss: 0.0999 - acc: 0.9652 - val_loss: 0.1130 - val_acc: 0.9600
Epoch 33/34
4197/4197 [==============================] - 23s 6ms/step - loss: 0.1003 - acc: 0.9671 - val_loss: 0.1169 - val_acc: 0.9607
Epoch 34/34
4197/4197 [==============================] - 23s 6ms/step - loss: 0.1007 - acc: 0.9688 - val_loss: 0.1106 - val_acc: 0.9607
```

Figure 12. Training the ELMo-based model.

An analysis of the performance of this model shows a better performance than the previous one without a relevant overfitting (Figure 13).

On the other hand, for the BERT-based model, an SGD optimizer and a batch size of 35 were used, and after 7 epochs of 4 minutes each, an accuracy of 0.9676 in validation is obtained, as shown in Figure 14. An analysis of the performance of this model shows a better performance than the previous one without relevant overfitting (Figure 15).

Comparison between the models

We evaluated the proposed models using a set of new issues that do not belong to the training or the validation datasets (Table 3). The results obtained by testing the different models are listed in the column titled "Probability of being a User Story." From these results, several observations are made:

The identification of short user stories improves as the complexity of the applied model increases (the BRNN-LSTM being the simplest and the BERT-based the most complex), as observed in the first example of Table 3.

Considering the User Story 5 example, the BRNN-LSTM model is not able to recognize this example as unfavorable. In contrast, others return a lower probability, ensuring this is not a positive example.

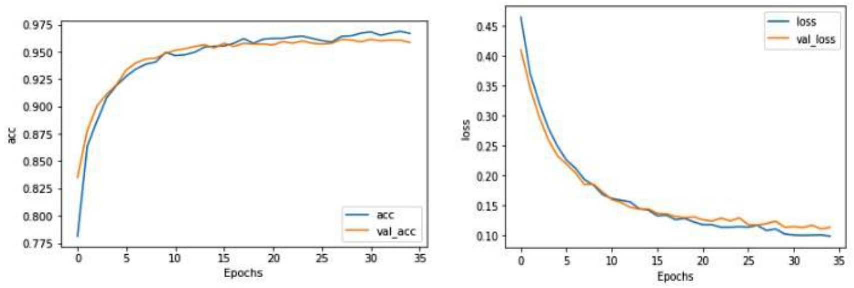
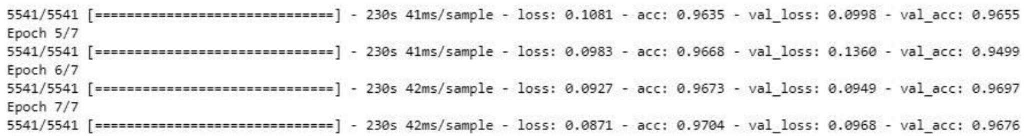


Figure 13. Accuracy and Loss analysis for the ELMo-based model.



```
5541/5541 [==============================] - 230s 41ms/sample - loss: 0.1081 - acc: 0.9635 - val_loss: 0.0998 - val_acc: 0.9655
Epoch 5/7
5541/5541 [==============================] - 230s 41ms/sample - loss: 0.0983 - acc: 0.9668 - val_loss: 0.1360 - val_acc: 0.9499
Epoch 6/7
5541/5541 [==============================] - 230s 42ms/sample - loss: 0.0927 - acc: 0.9673 - val_loss: 0.0949 - val_acc: 0.9697
Epoch 7/7
5541/5541 [==============================] - 230s 42ms/sample - loss: 0.0871 - acc: 0.9704 - val_loss: 0.0968 - val_acc: 0.9676
```

Figure 14. Training the BERT-based model.

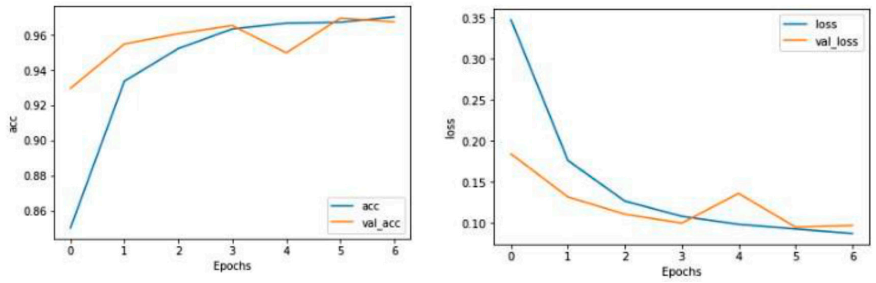


Figure 15. Accuracy and Loss analysis for the BERT-based model.

Table 3. Testing new examples in the implemented models.

No.	Issue	Probability of being a User Story		
		BRNN-LSTM	Elmo-based	BERT-based
1	As a developer, I want to implement tests.	0.1088	0.9155	0.9935
2	As a tester, I want to implement tests so i can assure the softwares quality.	0.9999	0.6258	0.9910
3	as an administrator i want a gui admin for configuration options.	0.9999	0.9973	0.8623
4	A tester want to implement tests so he can assure the software quality.	0.0053	0.0038	0.0012
5	I want a developer as much as good tester so I have a good team.	0.9664	0.1686	0.0151
7	As a IA tester, I want to wrtie with ortografics errors to test efficiency.	0.9999	0.8843	0.9195
8	An administrator will audit event via the system administration module.	0.0047	0.0044	0.0021
9	As a developer the default build should take less than 5 minuts.	0.0507	0.0008	0.0166

Table 4. Comparing the models.

Model	F1-score	complexity	tr-effort	parsing	semantic
BRNN-LSTM	0.946	Low	low	middle	low
ELMo-based	0.949	High	high	high	middle
BERT-based	0.965	High	high	high	high

Besides orthographic errors or unknown words in a user story, all the models can generalize it correctly, as seen in the User Story 7 example.

After implementing the models and assessing their results, a comparison can be made. The metrics considered are the F1-score obtained during the validation through the library Sklearn [40], the complexity, the training time (tr-effort), the syntactic analysis (parsing), and the semantic analysis (semantic). Table 4 shows the results of the comparison.

As observed in Table 4, the models obtained similar values for the validation metrics; however, the most notable difference lies in the ability to semantically and syntactically analyze user stories. The BERT-based model has a slightly superior generalization capability. Besides, although the BERT-based model complexity is higher (in terms of the times and number of training epochs), it can be observed that there exists an improvement in the parsing and semantic interpretation. In contrast, the parsing of issues is similar for the BERT-based and ELMo-based models.

CONCLUSIONS

In this work, three different neural network models were implemented to identify user stories in large volumes of data. From the results obtained using these models, it was analyzed which is better for the classification of issues records. Also, it was concluded that the BERT-based model can analyze the text syntactically and semantically with higher accuracy and performance. Future work will involve improving the dataset used by increasing the number of cases, finding a better balance between positive and negative classes, and then retraining the models to enhance the obtained results. A limitation of the approach is that a previous loading and extraction process of issues of any type in an IMS is needed to have a dataset and then feed the model to extract the user stories. In other words, some coding knowledge is still required to use the proposed models.

This work can be the first step to applying other techniques to analyze user stories within Issues Management systems. The approach can be embedded in an IMS tool for automatically categorizing of incidents as user stories, which would save time and avoid error occurrences, therefore improving the quality of the software development project documentation. Therefore, this proposal allows performing any study based on user stories and locating possible requirements or requests for new functionalities in a large repository, even if the incidents are not labeled as such.

REFERENCIAS

[1] G.G. Lucassen, Understanding User Stories, Universität Utrecht, Graad van doctor, 2017. [Online]. Available: http://dspace.library.uu.nl/handle/1874/356784 (accessed Oct. 2, 2019).

[2] C. Henderson, *Building Scalable Web Sites*, USA: O'Reilly Media, 2006.

[3] F.J. Peña Veitía, "Issues_logs_data_exploration/data-exploration&user stories classification.ipynb at master fjpena35226/issues_logs_data_exploration". [Online]. Available: https://github.com/fjpena35226/issues_logs_data_exploration/blob/master/data-exploration%26user stories classification.ipynb (accessed Sep. 21, 2020).

[4] J. Cabot, J.L.C. Izquierdo, V. Cosentino, and B. Rolandi, "Exploring the use of labels to categorize issues in Open-Source Software projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, Reengineering (SANER)*, Montreal, QC, Canada, 2015, pp. 550-554, doi: 10.1109/SANER.2015.7081875.

[5] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *2013 35th International*

Conference on Software Engineering (ICSE), San Francisco, CA, USA, 2013, pp. 392-401, doi: 10.1109/ICSE.2013.6606585.

[6] F. Thung, D. Lo and L. Jiang, "Automatic Defect Categorization," *2012 19th Working Conference on Reverse Engineering*, Kingston, ON, Canada, 2012, pp. 205-214, doi: 10.1109/WCRE.2012.30.

[7] I. Chawla and S.K. Singh, "An automated approach for bug categorization using fuzzy logic," in *ACM International Conference Proceeding Series*, New York, New York, USA: Association for Computing Machinery, Feb. 2015, pp. 90-99, doi: 10.1145/2723742.2723751.

[8] J.M. Alonso-Abad, C. López-Nozal, J. M. Maudes-Raedo, and R. Marticorena-Sánchez, "Label prediction on issue tracking systems using text mining," *Progress in Artificial Intelligence*, vol. 8, no. 3, pp. 325-342, Sep. 2019, doi: 10.1007/s13748-019-00182-2.

[9] G. Lucassen, M. Robeer, F. Dalpiaz, J.M. E. M. van der Werf, and S. Brinkkemper, "Extracting conceptual models from user stories with Visual Narrator," *Requir Eng*, vol. 22, no. 3, pp. 339-358, 2017, doi: 10.1007/s00766-017-0270-1.

[10] R. Mesquita, A. Jaqueira, C. Agra, M. Lucena, and F. Alencar, "US2StarTool: Generating i∗ models from user stories," *CEUR Workshop Proc*, vol. 1402, pp. 102-108, 2015.

[11] Y. Wautelet, S.Heng, D. Hintea, M. Kolp, and S. Poelmans, "Bridging User Story Sets with the Use Case Model," in *ER 2016 workshops*, S. Link and J.C. Trujillo, Eds., 2016, pp. 127-138, doi: 10.1007/978-3-319-47717-6 11.

[12] F. Gilson, M. Galster, and F. Georis, "Extracting Quality Attributes from User Stories for Early Architecture Decision Making," *2019 IEEE International Conference on Software Architecture-Companion, ICSA-C 2019*, 2019, pp. 129-136, doi: 10.1109/ICSA-C.2019.00031.

[13] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, "Unifying and Extending User Story Models," in *Advanced Information Systems Engineering. CAiSE 2014, Lecture Notes in Computer Science*, M. Jarke. *et al.*, Eds. vol. 8484, doi: 10.1007/978-3-319-07881-6_15.

[14] D.W. Otter, J.R. Medina, and I. Mirbel, "A Survey of the Usages of Deep Learning for Natural Language Processing," *IEEE Trans Neural Netw Learn Syst*, vol. 32, no. 2, pp. 604-624, Feb. 2021, doi: 10.1109/TNNLS.2020.2979670.

[15] M. Sahlgren, "A brief history of word embeddings (and some clarifications) | LinkedIn". [Online]. Available: https://www.linkedin.com/pulse/brief-history-word-embeddings-some-clarifications-magnus-sahlgren/ (accessed Oct. 29, 2019).

[16] A. Yadav and D.K. Vishwakarma, "Sentiment analysis using deep learning architectures: a review," *Artif Intell Rev*, vol. 53, no. 6, pp. 4335-4385, Aug. 2020, doi: 10.1007/S10462-019-09794-5/METRICS.

[17] W. Xia, W. Zhu, B. Liao, M. Chen, L. Cai, and L. Huang, "Novel architecture for long short-term memory used in question classification," *Neurocomputing*, vol. 299, pp. 20-31, 2018, doi: 10.1016/j.neucom.2018.03.020.

[18] M. Schuster and K.K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673-2681, 1997, doi: 10.1109/78.650093.

[19] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional LSTM networks for improved phoneme classification and recognition," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3697, pp. 799-804, 2005.

[20] Y.P. Lin and T.P. Jung, "Improving EEG-based emotion classification using conditional transfer learning," *Front Hum Neurosci*, vol. 11, 2017, Art no. 334, doi: 10.3389/fnhum.2017.00334.

[21] M. Peters *et al.*, "Deep Contextualized Word Representations," in *NAACL. Association for Computational Linguistics (ACL)*, pp. 2227-2237, 2018, doi: 10.18653/v1/n18-1202.

[22] Tensorflow Hub-Google, "Elmo-Tensorflow Hub". [Online]. Available: https://tfhub.dev/google/elmo/3 (accessed Mar. 31, 2020).

[23] J. Devlin, M.W. Chang, K. Lee, and K. Toutanova, "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), J. Burstein, C. Doran, and T. Solorio, Eds.

2019, pp. 4171-4186, doi: 10.18653/v1/N19-1423.

[24] S. González-López, S. Bethard, F.C.E. Orozco, and A.P. López-Monroy, "Consumer cynicism identification for spanish reviews using a Spanish transformer model", *Procesamiento del Lenguaje Natural*, vol. 66, pp. 111-120, 2021, doi: 10.26342/2021-66-9.

[25] Tensorflow Hub - Google, "Bert_uncased - TensorFlow Hub". [Online]. Available: https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1 (accessed May. 25, 2020).

[26] A. Vaswani *et al.*, "Attention is All you Need," *in Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Nov. 3, 2017, doi:10.48550/arXiv.1706.03762.

[27] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. Rush, "OpenNMT: Open-Source Toolkit for Neural Machine Translation," in *Proceedings of ACL 2017, System Demonstrations*, M. Bansal and H. Ji, Eds. Vancouver, Canada, 2017, pp. 67-72.

[28] P. Chapman *et al.*, "CRISP-DM 1.0: Step-by-step data mining guide". 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:59777418 (accessed: Oct. 28, 2023).

[29] CONNECT, "Navegador de incidencias - Issue Tracker CONNECT". [Online]. Available: https://connectopensource.atlassian.net/issues/?jql=orderbycreatedDESC&startIndex=50 (accessed Oct. 30, 2019).

[30] F. Dalpiazo, "Requirements data sets (user stories)," Mendeley Data, 2018. [En línea]. Disponible: https://data.mendeley.com/datasets/7zbk8zsd8y/1

[31] F.J. Peña Veitía, "Fjpena35226/augmenting dataset_userstories". [Online]. Available: https://github.com/fjpena35226/augmentingdataset_userstories (accessed Oct. 1, 2020).

[32] F.J. Peña Veitía, "Identifying User Stories in Issues records". [Online]. Available: https://data.mendeley.com/datasets/bw9md35c29/1/files/2515d495-62f4-4b59-a738-4dcdcef03efe/dataSetUserStoriesRecognition.csv?dl=1 (accessed Jul. 02, 2020).

[33] F.J. Peña Veitía, "Fjpena35226/rnn_simplre_lstm_userstories_recognition". [Online]. Available: https://github.com/fjpena35226/rnn_simplre_lstm_userstories_recognition (accessed Jul. 31, 2020).

[34] T. Hub, "TensorFlow Hub". [Online]. Available: https://www.tensorflow.org/hub (accessed Oct. 31, 2019).

[35] J. Zweig, "Keras-elmo/Elmo Keras.ipynb at master strongio/keras-elmo". [Online]. Available: https://github.com/strongio/keras-elmo/blob/master/Elmo Keras.ipynb (accessed Oct. 31, 2019).

[36] G. Issues, "Using ELMo with TensorFlow 2.0 Issue #412 tensorflow/hub". [Online]. Available: https://github.com/tensorflow/hub/issues/412 (accessed Oct. 31, 2019).

[37] F.J. Peña Veitía, "Fjpena35226/rnn_ElMo_userstories_recognition". [Online]. Available: https://github.com/fjpena35226/rnn_ElMo_userstories_recognition (accessed Oct. 31, 2019).

[38] J. Zweig, "Keras-bert/keras-bert.ipynb at master strongio/keras-bert". [Online]. Available: https://github.com/strongio/keras-bert/blob/master/keras-bert.ipynb (accessed Oct. 31, 2019).

[39] F.J. Peña Veitía, "Fjpena35226/bert_userstories_recognition". [Online]. Available: https://github.com/fjpena35226/bert_userstories_recognition (accessed Mar. 24, 2020).

[40] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, M. Braun, Ed. vol. 12, no. 85, pp. 2825-2830, 2011.