Negation Conflict Detection in Agile User Story Requirements using Machine Learning Approach

Fahd Alsahli

Abstract

This paper introduces a machine learning based approach toward requirements validation and verification. It considers conflicts in agile user story requirements (user story). More precisely, it focuses on negation conflicts. For example, the pair of user stories “As a user, I want to be able to have multiple accounts” and “As a developer, users should not have multiple accounts” is considered as a negation conflict.

The approach is to deploy paragraph vectors model to encode user story requirements as vectors. The vectors are used as input features to a Support Vector Machine (SVM) classifier. Each input consists of two vectors corresponding to two different user stories. In addition to the vectors, two other features are used. The features are Negation and Jaccard Coefficient. Negation is a binary feature, and it is set to 1 if a sentence includes a negation, and it is set to 0 otherwise. Jaccard Coefficient is the ratio of common words in two sentences to the total number of words.

A problem with this approach is the lack of annotated user story dataset. I utilized transfer learning to overcome this issue. I used Stanford’s Recognizing Textual Entailment (RTE) dataset to train and test the introduced model. Then, I validated the model on a small manually annotated user story dataset.

The model was built and tested using RTE data. It achieved an average accuracy of 0.5045, an average precision of 0.5283, and an average recall value of 0.4705 when tested with RTE data. The model achieved an average accuracy of 0.8708, an average precision of 0.8779, and an average recall value of 0.8585 when tested with the manually

labeled data. The most important limitation of the model was that it did not use Natural Language Processing (NLP) methods to analyze data along with the utilized models (e.g., paragraph vectors and SVM). This is due to the fact that text data is highly correlated and needs some processing before using them with learning approaches.

Keywords

Negation conflicts, user story requirements, paragraph vectors.

1. Introduction

Requirements validation and verification is an essential part of requirements analysis. This is due to the fact that resolving issues in requirements elicitation phase saves time in designing and developing systems. Many approaches toward requirements validation and verification have been proposed. In [7] a conflict resolution strategy that filters requirements, analyzed requirements, and resolved conflicts was proposed. In [8] a conflict detection method was introduced. The method considers goal-oriented requirements. In [5] a combination of natural language processing and artificial neural networks was proposed to detect conflicts in requirements.

Requirements are written in several different formats such as goal-oriented requirements and use case requirements. One such format is agile user story (user story). User stories are short and small description of a desirable feature. It is written as “As a (type of user), I want (some goal) so that (some reason)” [16]. Type of user could be user, admin, owner, etc. As an example, “as a user, I want a fast-responsive website, so that I do not waste time waiting” is a user story.

In this paper, I am going to provide a method for detecting negation conflicts in user story requirements. The method utilized paragraph vectors proposed in [17] to represent user stories as vectors. Paragraph vectors is a technique that converts text into vector representation and preserving semantic meanings of encoded text. These vectors are used as inputs to a Support Vector Machine (SVM) classifier, and each input includes two

vectors that encode two different user story requirements. Two other features are used along with the vectors. The features are Negation and Jaccard Coefficient proposed in [1]. Negation is set to 1 if a sentence includes a negation, and it is set to 0 otherwise. Jaccard Coefficient is a similarity measure, and it is computed as the number of common words between two sentences divided by the total number of words.

The rest of the paper is organized as the following. In section 2, I discuss some of the existing approaches. In section 3, I illustrate the proposed approach. In section 4, I provide an empirical validation. In section 5, I discuss results of experiments. In section 6, I discuss and compare the proposed approach and two other approaches. In section 7, I discuss threats to validity of the research. In section 8, I conclude the research.

2. Related Work

This section discuses some research done in conflict detection in requirements. Also, it includes some discussion on some Natural Language Processing (NLP) techniques. Reviewed literature includes several approaches toward detecting inconsistencies in requirements, and it also includes several types of requirements.

In [1], a deep learning approach toward contradiction detection in text was proposed. The proposed model considers three types of contradictions namely negation, antonyms and numeric mismatch. The model utilizes Long short-term memory (LSTM) and Global Vector (GloVe) to encode input text as vectors. The researchers developed four features to enhance their model. The developed features are:

a. Jaccard Coefficient:

Jaccard Coefficient is a measure of similarity between two sentences. It is calculated as the number of common words between two sentences divided by the total number of words.

b. Negation:

Negation is a binary feature. It is set to 1 if a sentence includes a negation, and it is set to 0 otherwise.

c. IsAntonym:

This feature is binary also. It is set to 1 if a pair of sentences includes antonym words, and it is set to 0 otherwise.

d. Overlap Coefficient:

Overlap Coefficient is another similarity measure. It is the ratio of common words of two sentences to the size of the smaller sentence.

The encoded text and the four features are fed to an Artificial Neural Network (ANN) classifier. The model was trained and tested on three datasets. The datasets were Stanford, SemEval, and PHEME datasets. The training samples were 1073, 3822, and 906 respectively. The testing samples were 1189, 3513, and 387 respectively. The model achieved 71.9% accuracy on Stanford dataset, 91.2% on SemEval dataset, and 96.85% on PHEME.

In [2], a tool for ambiguity in user stories was proposed. Researchers claimed that near-synonym words and incomplete user stories are main sources for ambiguity. So, the proposed tool considers near-synonym words and incomplete user stories. The tool parses text to extract conceptual models and semantic similarity. Then, it visualizes the results to ease analyzing them. An algorithm was developed to parse texts. The algorithm extracts nouns from all user stories. Then, for every two nouns, it computes the similarity between them, and then it finds their contexts (e.g., user stories where they appear at). The similarity score is obtained after converting text to vectors. The authors utilized the model presented in [15] to encode text as vectors. After that, the proposed algorithm computes similarity between contexts. Finally, it provides an ambiguity score. Finally, the tool visualizes the results. To validate the developed tool, 57 participants did a manual analysis. The participants were divided into 28 groups, and each group considered a data set of user story requirements. The results of the manual analysis were compared to the tool's results. The two analysis provided similar results.

In [3], a technique for extracting conceptual models from user stories requirements was proposed. The technique utilized NLP methods to analyze the requirements text. Based on the analysis, the technique produced conceptual models describing the system to be developed. Two datasets with sizes 73 and 32 (e.g., the included number of user stories) were used to test the system. The system could achieve a precision of 91% in analyzing the first dataset. It could achieve a precision of 81.4% in analyzing the second dataset. An advantage of the system was that it was relatively simple. A disadvantage was that the system worked with user stories only.

In [4], a deep learning approach toward conflict detection in norm contracts was proposed. The approach consists of two parts. First, contracts are parsed and classified as norm or non-norm. A Support Vector Machine (SVM) is used for the classification task. Second, conflicts in norm contracts are detected. A Convolutional Neural Network (CNN) is utilized for this task. A Dataset of contracts was collected and annotated to train and test the models. SVM model was trained on 954 sentences and tested on 238 sentences. The training data included 559 norm sentences while the testing data had 139 norm sentences. SVM achieved 90% accuracy. CNN was trained and tested on a dataset with 208 pairs of norm sentence. The dataset had 104 conflicting pairs. It was divided into 80% training set, 10% validation set, and 10% testing set. CNN achieved 95% accuracy.

In [5], a combination of NLP and Artificial Neural Networks (ANNs) was developed to parse the requirements document. Requirements were in unstructured language format. The goal of the model was to extract actions and actors from the document. The used NLP parser was Stanford-CoreNLP. The ANN was built from scratch. The parser analyzed the text and extracted tokens (e.g., nouns and verbs). These tokens were prepared (e.g., encoded) to be passed to the ANN. The ANN were trained to learn a mapping between a token and its encoded semantic meaning (e.g., action or actor). Five case studies were utilized to test the system. Out of the total number of actions and actors, 50% were correctly identified in the first, third, and fifth case studies, 63% were correctly identified in the second case study, and 17% were correctly identified in the fourth case

study. An advantage of the system was that it did not require stakeholders' participations. A disadvantage was that it required a relatively large amount of data for training.

In [6] a model for classifying non-functional requirements (NFRs) was presented. The model was a machine learning based classifier. Its goal is to trace requirements documents and detect NFRs. The classifier detects NFRs by considering key terms of NFR types. For example, an NFR types is security, and some of its key terms are confidentiality, integrity, and encryption. The data set utilized to train the model had 684 requirement statements, and 326 of them were NFRs. The model was validated using cross-validation technique. The proposed model had low precision values (e.g., ranging from 11.11% to 27.27%) although it had good recall values (e.g., ranging from 51.43% to 98.39%). An important limitation of the model was that it was not trained on a relatively large data, so the model is not general.

Butt [7] proposed a Conflict Resolution Strategy (CRS) which filtered requirements, analyzed requirements, and resolved conflicts. Requirements were represented in unstructured language format. Requirements were filtered (classified) into Mandatory Requirements (MRs), Essential Requirements (ERs), and Optional Requirements (ORs). MRs were detected and resolved in elicitation phase. ERs were detected and resolved in requirements analysis phase. Conflicts in ORs were marked in requirements analysis phase but not resolved. Conflict detection was achieved by applying a divide and conquer approach. For example, a requirement is divided into some parts, and these parts were checked against parts of other requirements. To test the proposed approach, it was applied in the requirements phase in National University of Sciences & Technology, Pakistan. 25 requirements were collected, and 13 of them were conflicting with each other. 44 people were involved, and their feedback about the system was as the following. 25 said that the system was excellent, 10 said very good, 1 responded with good, 5 responded with average, and 3 believed the system is poor. An advantage of the system was that it did not require training of the system. A disadvantage of it was that it required several stages to accomplish the task.

In [8], an extended version of Goal Oriented Requirements Elicitation technique called Attributed Goal Oriented Analysis (AGORA) was introduced. Requirements were represented in goal model. The technique classified conflicts into two types. The first type included the conflicts occurring as a result of different interpretations by stakeholders. The second type included the ones happening because of different evaluations of preferences of a requirement. The technique required each stakeholder to not only evaluate their preferences for a requirement but also to evaluate preferences of other stakeholders. Based on stakeholders' evaluations, the technique detected conflicts. For example, if a stakeholder preferred a requirement and another did not, then the system would detect the introduced conflict. To test the technique, an experiment was set up. Participants were selected such that they did not have any political or personal relationships with each other. The participants were told to provide requirements of a given product. An expert was invited to analyze the requirements. Also, AGORA was used to analyze the requirements. Then, the results obtained from the expert were compared with the ones obtained from AGORA. The two sets of results were similar. An advantage of the system was that it did not require training of the system. A disadvantage was that it required stakeholders' involvements.

Kaiya [9] proposed a software requirements analysis method based on domain ontology technique. Requirements were represented in unstructured language format. Ontology system consisted of a thesaurus and inference rules. The thesaurus included domain specific concepts. Three types of semantic processing were demonstrated. They were detecting incompleteness and inconsistency included in a requirements specification, measuring the quality of a specification with respect to its meaning, and predicting requirements changes based on semantic analysis on a change history.

Procedure was as the following. First, requirements text was processed to extract relationships between words. Rules were applied to the relationships to detect the three mentioned semantic processing. One type of sematic processing of the system was tested. This type was predicting requirements changes based on semantic analysis on a change history. The system was presented with 35 versions of requirements of a product. The

system could detect more than 50 changes, and the total number of actual changes was not reported.

An advantage of the system was that it did not deploy sophisticated Natural Language Processing methods, and this is a result of considering domain ontology. But, this introduced a disadvantage which is that the system was specific to the terminology of software requirements.

Ali [10] developed a system to detect and analyze modelling errors in contextual goal model requirements. The system consisted of two analysis mechanisms to detect two kinds of modelling errors. The first mechanism was used for the detection of inconsistent specification of contexts in a goal model. The second mechanism concerned about the detection of conflicting context changes. The system analyzed texts to extract tokens (e.g., nouns and verbs). Then, it applied logic operations on the tokens to check for conflicts. To evaluate the developed system, 5 experts in Requirements Engineering were invited to produce requirements of a product and analyze the requirements. The system could detect conflicts that were not detected by the experts. The experts were asked to use the system and give their feedback about how easy to use the system and understand produced results, and time taken by the system to accomplish the task. Their response was that the system was easy to use, and results were easy to understand, but they claimed that the system was time consuming. An advantage of the system was that it considered the context at which the software would operate. Based on that, the system decided whether or not a given requirement would be applicable in a given context. A disadvantage of the system was that it included relatively large numbers of definitions and logic operations.

In [11], a conflict detection method based on ontology was presented. The method considers conflicts in requirement evolution. For instance, stakeholders define the initial set of requirements, and they make sure that the set is consistent. Then, the method is used as requirements are as added, deleted or replaced. Requirements are written in a specific format illustrated in the literature, so that processing them becomes easier. Conflicts are detected by using some rules presented in the literature. The method was

applied to a case study, and it could achieve its goal. A disadvantage of the method is that it is hard to use.

In [12], an automatic method for classifying user requests was proposed. The method processes user requests to detect qualities wanted by users. The method could help in improving products. The researchers defined some quality types and associated some terms with each type. For instance, security was defined as “Security refers to requirements discussing attacks on purpose that threaten either the system or information safety in all kinds of ways”. Some of associated terms with security are safe, sensitive, and protected. Several combination of NLP techniques and machine learning methods were developed and compared. NLP techniques investigated in the literature were word unigram and keyword unigram, word frequency and keyword frequency, TF-IDF, and heuristic property of user request. The literature considered three machine learning algorithms namely K-nearest neighbor (KNN), Naïve Bayes (NB), and Support Vector Machine (SVM). The developed models were trained and tested on three datasets. The datasets were user requests of three open source projects. The projects were KeePass, Mumble and Winmerge. The results of comparing the models is as the following. For KeePass dataset, the combination was TFIDF plus all keyword (TFIDF+ALL) and SVM multi-class (SVM-M), and the accuracy was 74.8%. For Mumble dataset, the combination was TFIDF+ALL and SVM binary-class (SVM-B), and the accuracy was 74.9%. For Winmerge dataset, the combination was TFIDF+ALL and SVM-B, and the accuracy was 78.4%. An important limitation of the approach was that the models were not general. For instance, the models were specific to the data that they were trained on. So, one needs to repeat the presented methodology to find the best combination for a dataset that is different from the presented ones.

In [13], a definition of contradiction in text was proposed. Also, text corpora were developed. The definition and corpora were utilized to ease NLP tasks. the system consisted of four stages to analyze text, filter text, and detect conflicts.

The system was validated by testing it with 8 datasets. The highest precision achieved was 74.07%. The sizes of the datasets were not reported. An advantage of the system was

that it was relatively simple. A disadvantage was that the system had relatively low accuracy.

In [14], several approaches toward NLP were presented. One approach was Embeddings. This approach mapped a piece of text (e.g., a word) to a vector representation such that the semantic meaning of the text is preserved. So, analysis based on similarities between words could be done. Two types of Embeddings were discussed. These types are Skip Gram and Continues Bag of Words. The difference between them were that Skip Gram considered words whereas Continues Bag of Words considered sentences. An advantage of the approach was that it was relatively simpler than other NLP methods. A disadvantage was that the model needed relatively large amount of time for training.

Mikolov [15] could utilize a Recurrent Neural Network (RNN) to learn vector representations of words. These vectors were one type of Embeddings. So, they preserved similarities in meaning between words and relationships between them. A model called Vector Offset was proposed. The model manipulated words by applying vector operations on the learned vectors. To illustrate its performance, the model was compared with other models via conducting several experiments, and the proposed model outperformed the other models. An advantage of the model was that it preserved sematic meaning of words and relationships between them. A disadvantage was that the model needed relatively large amount of time for training. The existing work has not applied machine learning based approaches (learning approaches) in conflict detection in user story requirements. So, this paper introduces a learning approach toward conflict detection in user story requirements. The following section discusses the proposed approach in detail.

3. Proposed Approach

The approach toward detecting negation conflicts in user story requirements takes a pair of user stories (e.g., two user stories at a time) as an input and produces a binary value of 1 or 0. A value of 1 indicates a conflicting pair, and a value of 0 indicates a consistent pair. The approach is divided into three parts. First, each user story is represented as a vector so that it would be possible to utilize them in a machine learning based model. Two features are used along with the encoded user stories. Second, encoded user stories and the two features are concatenated and fed to an SVM classifier. Third, the SVM classifier is utilized to classify inputs as contradiction or non-contradiction. A problem with the approach is that it requires a user story dataset that includes conflicts and is labeled. The following subsections discuss the approach in detail.

3.1 Vector Representation and Features

In order to use text information in a machine learning approach, it must be represented by numeric values. There are many ways to encode text as numeric values such as word vectors [15] and paragraph vectors [17]. In the proposed approach, paragraph vectors method is utilized since it is designed to work with paragraphs (e.g., sentences). In addition, I use two features which are negation and Jaccard coefficient [1]. These features are obtained from the original text (e.g., user stories before encoding them). Negation is a binary feature, and it is set to 1 if a user story in a pair of user stories includes a negation and the other user story does not, and it is set to 0 otherwise. Jaccard Coefficient is the ratio of common words in a pair of user stories to the total number of words in the pair. So, the output of this step consists of three features which are a vector representation of a pair of user stories (encoded user stories), a negation feature, and a Jaccard coefficient.

3.2 Concatenation

Concatenation is a very simple process. It takes the three features namely encoded user stories, negation, and Jaccard coefficient, then it stores them in a single data structure

(e.g., a list). So, the result of this step is a vector consisting of the three features. This vector can be conceptualized as a feature vector for its corresponding pair of user stories.

3.3 Classifier

The last step of the model is a Support Vector Machine (SVM) classifier. SVM takes a feature vector produced by concatenation as an input, and it produces a binary value of 1 or 0. A value of 1 means that the input feature vector corresponds to a conflicting pair of user stories, whereas a value of 0 means that the input feature vector corresponds to a consistent pair of user stories. Figure 1 shows a block diagram illustrating the steps of the approach.

3.4 Lack of Labeled User Story Data

The proposed approach requires a labeled dataset of user story requirements. A possible solution to solve this issue is to perform transfer learning. Transfer learning refers to utilizing a trained model on data that is similar to the training data of the model. Because user stories are similar to normal text (e.g., it does contain technical terminologies or low-level descriptions of a product), a normal contradiction dataset could be utilized. One such dataset is Stanford's Recognizing Textual Entailment (RTE) dataset [18]. So, to overcome the issue of lacking the data, I utilized RTE dataset to train and validate the proposed approach. Then, I used a dataset from [19] to validate the model as the following. First, I obtained 75 user stories the dataset. Then, I injected 25 inconsistencies. Each inconsistency is paired with a single user story of the obtained 75 user stories. So, there are 50 pairs of user story requirements with 25 conflicting pairs and 25 consistent pairs. I manually labeled this small dataset. Finally, I used the dataset to test the proposed model.

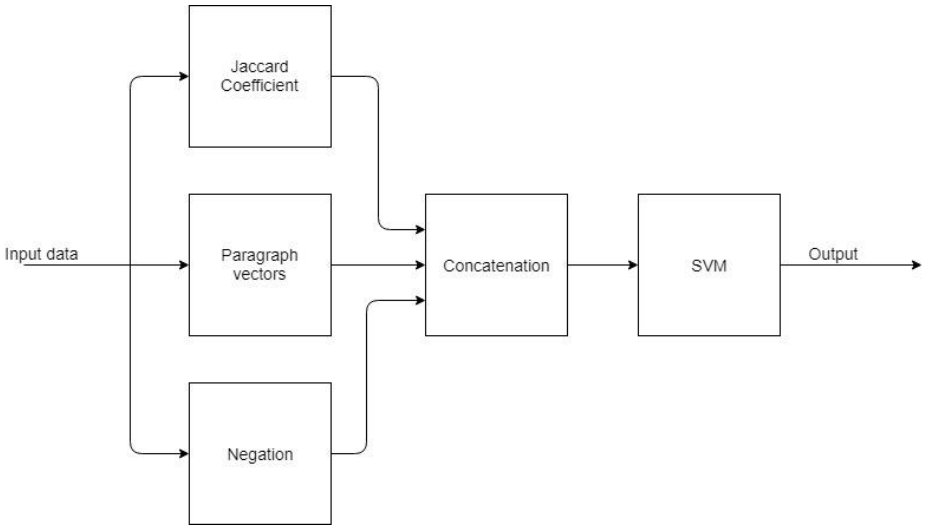

Figure 1. Block diagram showing the main steps of the proposed approach.

4. Empirical Validation

This section discusses the experiment developed to apply the proposed approach. First, data was obtained and pre-processed. Then, development of the approach is discussed. For example, deployed models and methods.

4.1 Dataset and Pre-processing

Data was obtained from two sources. The first data source was Stanford's RTE data [18]. It was annotated for contradiction. It was marked for a 3-way decision in terms of entailment. The labels are "YES" (entails), "NO" (contradicts) and "UNKNOWN" (doesn't entail but is not a contradiction).

The second data was collected from [19]. It includes 1948 user story requirements divided into 22 files. It was collected from the web and some software companies. The goal of collecting it was to utilize it in an ambiguity detection software. It was not labeled. Some pre-processing was needed before using the obtained data.

Pre-processing of the first data was as the following. I converted the "UNKNOWN" labels to the "YES" label. Then, I inverted the labels. For instance, I converted the "NO" labels to "YES" and the "YES" labels to "NO". This because the goal of the classifier is

to classify text to two classes namely contradiction (e.g., yes) and no contradiction (e.g., no). Finally, I encoded the labels as binary numeric labels. For example, I encoded the "YES" labels as ones and the "NO" labels as zeros.

The second data was pre-processed as the following. I combined all user stories in a single file. Then, I chose 75 user story requirements randomly. I injected 25 inconsistencies manually to 25 user stories out of the 75 user stories. This leaded to 25 conflicting pairs and 25 consistent pairs. In addition, I removed the first part of a user story (e.g., as a (type of user)) since it does not contribute well (e.g., it does not have specific meaning).

4.2 Examples of the Data

The pair of sentences "iTunes software has seen strong sales in Europe" and "Strong sales for iTunes in Europe" is a consistent pair from RTE data. The pair of sentences "Oracle had fought to keep the forms from being released" and "Oracle released a confidential document" is a conflicting pair from RTE data. The pair of user stories "As a User, I want to authenticate locally" and "As User, I want to be able to log in." is a consistent pair from the manually labeled data. The pair of user stories "As a user, I want to create a protocol and assign metadata to any stag" and "As an admin, users are not allowed to create protocols" is a conflicting pair from the manually labeled data.

4.3 Development of the approach

In order to implement the approach, I developed and trained two models. The models were paragraph vectors and SVM. I developed two methods to provide Negation and Jaccard Coefficient features. I developed a method to provide concatenation of vectors and features. I utilized Python programming language in the development of the approach. This section illustrates the development of the models and methods.

The first developed model was paragraph vectors. It was developed using Gensim library [20]. Gensim is a Python library, and it is used for Natural Language Processing (NLP)

tasks. It included implementations of several NLP methods. One such method is paragraph vectors model. Paragraph vectors itself has two models namely distributed memory (DM) and distributed bag of words. DM model was chosen since it considers the context at which words appear.

DM model has several parameters to be set. The most important parameters, in terms of influence on the model, are window, vector size, and epochs. The descriptions of these parameters are as the following. First, the parameter “window” tells the model how many words to look at before and after a word to be predicted. Second, the parameter “vector size” sets the dimensions of the output vectors. Finally, the parameter “epochs” tells the model how many times the model should go over the data.

The second model was SVM. It was developed using scikit-learn [21]. Scikit-learn is a Python library that includes implementations of various machine learning algorithms. The model was used to provide the binary classification (e.g., contradiction and non-contradiction). The most important parameter in SVM is the kernel.

Two methods were developed to provide Negation and Jaccard Coefficient features. Both methods take two strings as inputs, each string is a user story. The method providing Negation parses each input to find one of defined negation words. Defined negation words are no, never, not, nothing, no one, without, and nobody. The method returns a Boolean true value if a string has a negation word and the other does not. The method returns a Boolean false value otherwise.

The method that provides Jaccard Coefficient computes the common words between the input strings and sizes of the strings. Then, it computes the ratio of the number of common words to the total size of the two strings. Finally, it returns the ratio.

The last developed method was the concatenation method. It takes as inputs two vectors and two features. It returns the concatenation of the inputs. The two models and the three methods were connected as illustrated in figure 1.

5. Experiments and Results

This section illustrates the procedure followed to develop the approach. For example, it describes how parameters of DM and SVM are set. Also, it shows the results of the introduced approach and a comparison with two other approaches.

5.1 Parameters setting

This section illustrates the procedure followed in setting parameters of DM and SVM models. The parameters were set by defining a set of values for each considered parameter and then, for each combination of parameter values, comparing the resulted models. I considered three parameters of DM namely window size, vector size, and epochs. I considered one parameter of SVM which was the kernel function. The reason behind considering these parameters is that these parameters have the most influence on the models.

Sets of values corresponding to DM's parameters are as the following. The set of window sizes included the numbers 4, 6, 8, and 10. The set of vector sizes had the numbers 5, 10, 15, and 20. The set of epochs had the numbers 50, 100, 150, and 200.

SVM provided by [21] has four different functions. The functions are linear, polynomial, radial basis function (RBF), and sigmoid. The set of kernels included all four kernel functions. Table 1 summarizes the sets and their values.

I utilized Stanford's RTE data for this task. It has 4567 pairs of sentences, and 2263 pairs out of the total number of pairs are conflicting sentences. I divided the data into 90% for training and 10% for testing. So, there were 4110 pairs for training and 457 pairs for testing.

Table 1. Sets and their values.

Parameters	sets
Window size	{4, 6, 8, 10}
Vector size	{30, 40, 50, 60}
Epochs	{50, 100, 150, 200}
Kernels	{linear, polynomial, RBF, sigmoid}

There are four sets, and each has four values. So, there are 256 different combinations of these values. For each combination, I created a model of the proposed approach, and I saved its accuracy. After creating all the models, I chose the models with the highest five accuracies. Table 2 shows the models with their parameters and accuracies.

Table 2. Best five models with parameters and accuracies.

Models	Parameters {window size, vector size, epochs, kernel}	Accuracies
Model 1	{4, 20, 100, RBF}	0.5667
Model 2	{4, 10, 50, RBF}	0.5645
Model 3	{6, 15, 200, RBF}	0.5645
Model 4	{6, 15, 150, RBF}	0.5601
Model 5	{10, 15, 100, linear}	0.5601

I analyzed each of these models separately. For example, I tested every model 30 times, and I averaged its accuracies. I chose the model with the best average accuracy. The model with the best average accuracy had the set of parameters {6, 15, 150, RBF} and an average accuracy of 0.5054.

After this process and the best model is achieved, I tested it with the small user story data. This data includes 50 pairs of user story requirements. I utilized all this data to validate the model, so there was no retraining for the model. This was due to that lack of data (e.g., only 50 pairs).

5.2 Results

This section illustrates the results obtained when deploying the model achieved. The model has the set {6, 15, 150, RBF} of parameters. The model achieved an average accuracy of 0.5045, an average precision of 0.5283, and an average recall value of 0.4705 when tested with the 10% test data of RTE data. The model achieved an average accuracy of 0. 8708, an average precision of 0. 8779, and an average recall value of 0. 8585 when tested with the manually labeled data. Table 3 summarizes the results.

Table 3. Results of testing the model on the two datasets.

Dataset	accuracy	precision	recall
RTE	0.5045	0.5283	0.4705
Manually labeled	0.8708	0.8779	0.8585

The results showed that the model performed better on the manually labeled data. The main reason behind this was that the labeled data was smaller. It included only 50 pairs of user story requirements, while RTE's test set had 457 pairs. This resulted in that the manually labeled data were less correlated. Figure 2 shows the encoded data points of the manually labeled data reduced to 3-dimensinal space, and it shows that the data are less correlated. On the other hand, RTE's test set data points were more correlated. Figure 3 shows the encoded data points of the RTE's test set reduced to 3-dimensinal space, and it shows that the data is more correlated. I reduced the data to 3-dimensional space by utilizing an implementation of t-SNE method [22] provided by [21].

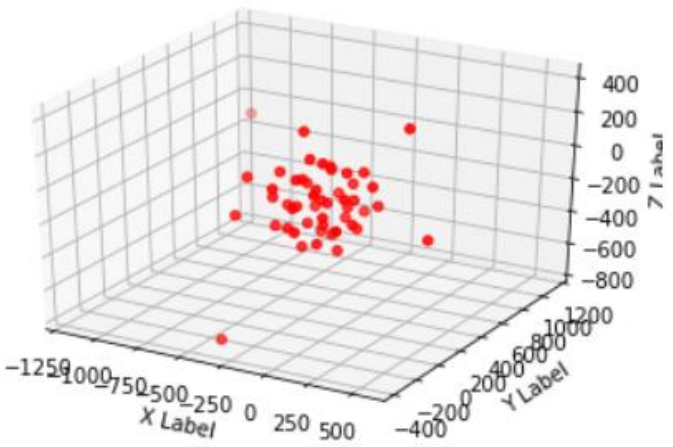

Figure 2. Encoded data points of the manually labeled data reduced to 3-dimensinal space.

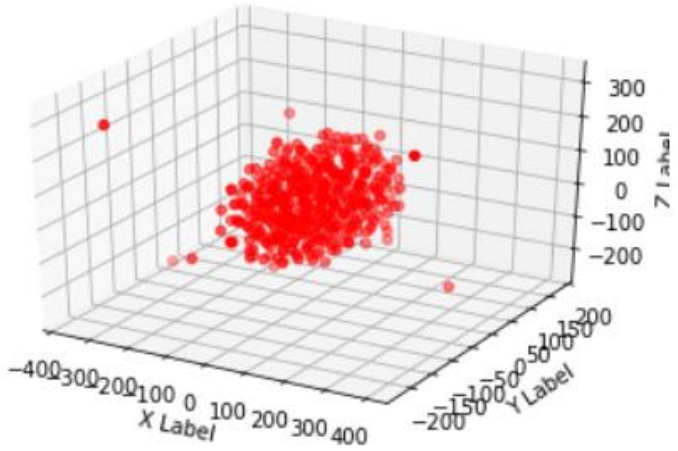

Figure 3. Encoded data points of the RTE's test set reduced to 3-dimensinal space.

6. Discussion

In this section, discussion and comparison of the proposed approach and approaches in [2] and [3] is provided. The two approaches do not consider conflicts in user story requirements, but they are considered to compare various ways for processing user story requirements in general.

The approach developed by Turkheimer et al. [2] extracts nouns from all user stories. Then, for every two nouns, it computes the similarity between them, and then it finds their contexts (e.g., user stories where they appear at). After that, the developed algorithm

computes similarity between contexts. The similarity score is computed after converting text to vectors. The model presented in [15] was utilized to encode text as vectors. Finally, the method provides an ambiguity score. The approach could achieve an average precision of 0.51 and an average recall value of 0.25.

Robeer et al. [3] presented a method for extracting conceptual models from user stories requirements was proposed. For instance, it extracts from a user story the user and the desired feature. The method utilized NLP techniques to analyze user story text. Based on the analysis, the method produces conceptual models describing the system to be developed. The method was tested on a dataset with size 73. It achieved a precision of 0.91 and a recall of 0.929.

The proposed approach outperformed the approach in [2] in terms of both precision and recall. On the other hand, the approach in [3] outperformed the proposed approach in terms of both precision and recall. The main reasons behind this is that the approach in [3] utilized NLP techniques, which are more robust than learning approaches. Table 4 summarizes the comparison.

Table 4. Comparison of the presented approach with two other approaches.

Approach	Type of the approach	Precision	Recall
Presented in [2]	Word vectors	0.51	0.25
Presented in [3]	NLP techniques	0.91	0.929
Proposed model	Paragraph vectors	0.8779	0.8585

7. Threats to Validity

7.1 Internal validity

Internal validity refers to how well the experiment is designed [23]. In other words, is the experiment valid? The parameters of DM and SVM models were set by comparing all models resulted from each combination of a set of parameters. Then, the best five models out of all models (e.g., 256 models) were chosen. The comparison was based on

accuracies of models. After that, the five models were analyzed and compared based on average accuracies. Finally, the best model was chosen. So, the experiment was valid.

7.2 External validity

External validity refers to how general the results are [23]? In the context of this experiment, can the proposed model be applied in real world data? The model is not general due to two reasons. First, the model was not trained on user story requirements data due to the lack of annotated data. Second, the manually labeled user story data is very small (e.g., only 50 pairs of user stories), so the data is not representative.

8. Conclusions

Requirements validation and verification is important phase of requirements analysis. This is because resolving issues in requirements elicitation phase improves designing and developing systems in terms of time needed to accomplish them. Different approaches toward requirements validation and verification have been proposed. Validation and verification process includes detecting conflicts in requirements.

In this paper a method for detecting conflicts in user story requirements was proposed. The method utilized paragraph vectors proposed in [17] to encode user stories as vectors. The encoded user story requirements were used as inputs to a Support Vector Machine (SVM) classifier. Two other features were used along with the vectors. The features were Negation and Jaccard Coefficient proposed in [1].

The parameter setting of distributed memory (DM), which is an instance of paragraph vectors, SVM models followed some procedure. First, three parameters of DM were considered, and one parameter of SVM was considered. The DM's parameters were window size, vector size and epochs. SVM's parameter was the kernel fiction. Second, an instance of the introduced approach was created for each combination of values of these parameters. Finally, the best five models in terms of accuracy were chosen.

Each model of the best five models was analyzed. Then, every model was tested 30 times, and its accuracies were averaged. The model with the best average accuracy was chosen.

The model with the highest average accuracy was validated with a user story data. The data had 50 pairs of user story. 25 pairs were conflicting pairs. The model achieved average accuracy of 0. 8708, an average precision of 0.8779, and an average recall value of 0.8585.

The model was compared to approaches presented in [2] and [3]. The proposed model performed better than the model in [2]. On the other hand, it performed worse than the model presented in [3].

The most important limitation of the approach was that the model does not deploy NLP techniques in addition to the used models. This is because text data are highly correlated and needs processing before utilizing them in learning approaches.

References

1. Lingam, V., Bhuria, S., Nair, M., Gurpreetsingh, D., Goyal, A., Sureka Corresp, A., & Sureka, A. (2018). Deep learning for conflicting statements detection in text Deep Learning for Conflicting Statements Detection in Text 2. https://doi.org/10.7287/peerj.preprints.26589v1
2. Turkheimer, E., & Waldron, M. (2019). Detecting terminological ambiguity in user stories: Tool and experimentation. Psychological Bulletin, 126(1), 21. https://doi.org/.1037//0033-2909.I26.1.78
3. Robeer, M., Lucassen, G., Werf, J. M. E. M. van der, 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). IEEE. http://doi.org/10.1109/RE.2016.40

4. Aires, J. P., & Meneguzzi, F. (2017). A deep learning approach for norm conflict identification. Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, 3, 1451–1453.
5. Al-Hroob, A., Imam, A. T., & Al-Heisa, R. (2018). The use of artificial neural networks for extracting actions and actors from requirements document. Information and Software Technology, 101, 1–15. http://doi.org/10.1016/j.infsof.2018.04.010
6. Cleland-Huang J, Settimi R, Zou X, Solc P (2007) Automated classification of non-functional requirements. Requirements Engineering 12:103–120. doi: 10.1007/s00766-007-0045-1
7. Butt, W. H., Amjad, S., & Azam, F. (2011). Requirement Conflicts Resolution: Using Requirement Filtering and Analysis (pp. 383–397). Springer, Berlin, Heidelberg. http://doi.org/10.1007/978-3-642-21934-4_31
8. Kaiya, H., Shinbara, D., Kawano, J., & Saeki, M. (2005). Improving the detection of requirements discordances among stakeholders. Requirements Engineering, 10(4), 289–303. http://doi.org/10.1007/s00766-005-0017-2
9. Kaiya, H., & Saeki, M. (n.d.). Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach. In Fifth International Conference on Quality Software (QSIC'05) (pp. 223–230). IEEE. http://doi.org/10.1109/QSIC.2005.46
10. Ali, R., Dalpiaz, F., & Giorgini, P. (2013). Reasoning with contextual requirements: Detecting inconsistency and conflicts. Information and Software Technology, 55(1), 35–57. http://doi.org/10.1016/J.INFSOF.2012.06.013
11. Liu C-L (2016) CDNFRE: Conflict detector in non-functional requirement evolution based on ontologies. Computer Standards & Interfaces 47:62–76. doi: 10.1016/j.csi.2016.03.002
12. Luo, B., Ge, J., Huang, L., Ng, V., & Li, C. (2017). Automatically classifying user requests in crowdsourcing requirements engineering. Journal of Systems and Software, 138, 108–123. https://doi.org/10.1016/j.jss.2017.12.028

13. De Marneffe, M.-C., Rafferty, A. N., & Manning, C. D. (2008). Finding Contradictions in Text. Retrieved from https://nlp.stanford.edu/pubs/contradiction-acl08.pdf
14. Singh, P. (2019). Natural Language Processing. In Machine Learning with PySpark (pp. 191–218). Berkeley, CA: Apress. http://doi.org/10.1007/978-1-4842-4131-8_9
15. Mikolov, T., Yih, W.-T., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. Association for Computational Linguistics. Retrieved from http://research.microsoft.com/en-
16. Cohn M (2015) User stories applied for agile software development. Addison-Wesley, Boston
17. Distributed Representations of Sentences and Documents. https://cs.stanford.edu/~quocle/paragraph_vector.pdf
18. The Stanford NLP Group. In: The Stanford Natural Language Processing Group. https://nlp.stanford.edu/projects/contradiction/. Accessed 21 Mar 2019
19. Data.mendeley.com. (2019). Requirements data sets (user stories). [online] Available at: https://data.mendeley.com/datasets/7zbk8zsd8y/1 [Accessed 3 Mar. 2019].
20. Radimrehurek.com. (2019). gensim: topic modelling for humans. [online] Available at: https://radimrehurek.com/gensim/index.html [Accessed 3 Mar. 2019].
21. learn. In: scikit. https://scikit-learn.org/stable/index.html. Accessed 21 Mar 2019.
22. Van Der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. Journal of Machine Learning Research (Vol. 9). Retrieved from http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf
23. Feldt, R., & Magazinius, A. (n.d.). Validity Threats in Empirical Software Engineering Research-An Initial Survey. Retrieved from http://www.robertfeldt.net/publications/feldt_2010_validity_threats_in_ese_initial_survey.pdf