Semantic Annotations in Clinical Guidelines

Fahrurrozi Rahman[0000−0001−9093−8518] and Juliana Bowles[0000−0002−5918−9114]

School of Computer Science, University of St Andrews, St Andrews KY16 9SX, United Kingdom {f27,jkfb}@st-andrews.ac.uk

Abstract. Clinical guidelines are evidence-based recommendations developed to assist practitioners in their decisions on appropriate care for patients with specific clinical circumstances. They provide succinct instructions such as what drugs should be given or taken for a particular condition, how long such treatment should be given, what tests should be conducted, or other situational clinical circumstances for certain diseases. However, as they are described in natural language, they are prone to problems such as variability and ambiguity. In this paper, we propose an approach to automatically infer the main components in clinical guideline sentences. Knowing the key concepts in the sentences, we can then feed them to model checkers to validate their correctness. We adapt semantic role labelling approach to mark the key entities in our problem domain. We also implement the technique used for Named-Entity Recognition (NER) task and compare the results. The aim of our work is to build a reasoning framework that combines the information gained from real patient data and clinical practice, with clinical guidelines to give more suitable personalised recommendations for treating patients.

Keywords: Therapy algorithms · Formal verification · Natural language processing · Machine learning · Text tagging

1 Introduction

Recent studies have shown that over the next 20 years there will be an increased expansion of morbidity, and particularly complex multimorbidity which occurs when individuals have several concurrent chronic conditions [12]. One of the challenges of treating patients with multimorbidity is that clinical guidelines are generally focused on single disease. It is hence difficult to understand treatment options, and their consequences in the long term, when patients have to follow a considerable number of single disease oriented treatments simultaneously.

Furthermore, there are also several challenges inherent in clinical guidelines: they are written mostly in natural language and hence prone to ambiguity; the way they are presented may be very different from one guideline to another; clinical practice varies and may at times deviate from clinical guidelines, and we need to take into account this variability. When there are multiple treatment options available, we want to know how likely these options are followed in practice.

We have built a framework to formalise the written text of a therapy algorithm for lowering blood glucose for people with type 2 diabetes (T2D) in our previous work [19]. The generated model is a network of timed automata, where each transition represents the medication taken by the patient and the state is the patient's condition after getting such treatment and over time. The model also takes into account how the value of HbA1c, i.e., the glycated haemoglobin that is commonly measured to determine the average blood sugar levels for patients with diabetes, may deteriorate over time, and force further intensifications in the treatment. The advantage of a formalisation is that it enables us to detect gaps or omissions in the textual algorithm, which can then be used to further clarify treatment steps.

Although this approach is promising, it depends entirely on their own curated controlled natural language (CNL) to design and hand-tune complex rules to extract information from the therapy algorithm written in a natural language string. This approach becomes problematic when recommendations are expressed in very different ways as either the CNL needs to be refined or the recommendation sentences need to be adjusted to the CNL, which in most cases it involves both. Expanding the breadth and complexity of the CNL also demands a lot of human development work, linguistic knowledge as well as a deep understanding in the specific domain. This rigid and unscalable approach is not suitable for the long term goal: to process any clinical guidelines automatically.

In this paper, we propose a different approach to address this issue of capturing the main concepts in clinical guideline sentences using semantic role labeling and named-entity recognition. This approach will limit the human effort needed to design the grammar for a CNL. The result can then be used for further linguistic analysis in clinical guideline domain. It can also be utilised to check the correctness of the guideline by transforming the main concepts in the guideline into formal representations such as UPPAAL [2], PRISM [13], or Z3 [16].

The remainder of this paper is structured as follows. In Section 2 we discuss related work that serve as the foundation of this paper. Section 3 describes the framework of the system that we want to build. Section 4 explains the semantic annotations covering the ontology concepts, the relationships between each concept and named-entity recognition. The learning process, including the syntactic and the semantic analysis as well as the features used and the learning models are discussed in Section 5. The outcome of the experiments using several learning models is discussed in Section 6. Finally, in Section 7, we draw some conclusions on our research.

2 Related work

Considerable research in modelling and formalising clinical guidelines has been done over the past years. Pérez et. al. [18] built a framework to enable authoring and verification of clinical guidelines. They use UML statecharts to represent the guidelines and provide a pattern-based approach to define commonly occurring types of requirements in guidelines to ease the non-expert to write

formal specifications. The statecharts are transformed into process meta language (PROMELA) and the specifications are translated into linear temporal logic (LTL). The verification of the guideline model and its specification is then performed using the SPIN model checker [10].

Bäumler et. al. [1] also apply formal modelling and verification to improve the quality of medical guidelines. To model the guidelines, they must be written in Asbru language [21], a predefined language for guideline-application tasks. With the properties formulated in the Action Computational Tree Logic (ACTL) language, the model is then verified using the Cadence SMV model checker [14].

Another implementation of model checking to verify clinical guidelines is done by Giordano et. al. [7]. They use the GLARE language [22], a domain-independent prototypical system for acquiring, representing and executing clinical guidelines. With an XML intermediary layer which then is translated to PROMELA, the model and its specification written in LTL are verified using the SPIN model checker.

In software engineering, Carvalho et. al. [4] have created a framework to formally generate test cases from the written software requirements into several formalisms using natural language processing (NLP) techniques. Written in a controlled natural language, the requirements are transformed into data flow reactive system (DFRS), where inputs and outputs are modelled as signals, with timers to capture the time-based behaviour.

Another work by Diamantopoulos et. al. [5] shows a system that automatically maps software requirements into formal representations to detect problems hidden in the written texts at the early stage of development process. The system is built upon ontology class hierarchies to represent the semantic roles in the requirement texts. The hierarchies are built by training a semantic role labelling system from software requirements project classes in Europe. The inference process is done after the ontology is represented in the web ontology language (OWL).

In this research, we modify our previous work [19] following the approach used by Diamantopoulos et. al. [5] and NER. Concretely, we will adapt the work in [5] and NER to annotate the key concepts in guideline sentences domain so it becomes less labour intensive, more scalable, and more general purpose.

3 The framework

Figure 1 shows the whole framework in our research. We simplify our previous work by adding the syntactic and semantic analysis module. Instead of manually crafting a CNL, we use machine learning to achieve the goal.

Firstly, we provide the guideline sentences that we want to map into some formal representations. Some guidelines would have an implicit orders on how therapy should be given while others are orderless. Next, our syntactic and semantic analysis module will mark the key concepts in the sentences. The learning process for this module is further explained in Section 5.

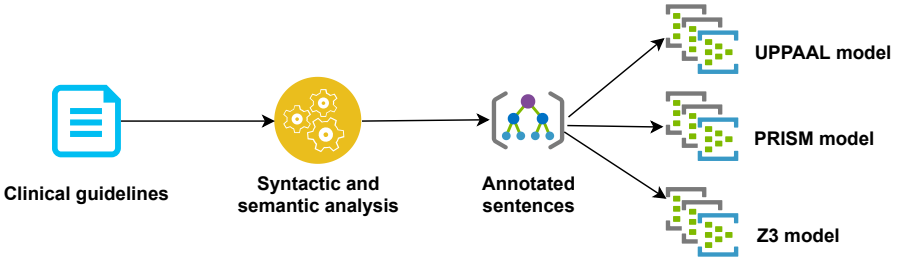

Fig. 1. The mapping of clinical guidelines framework

The result of this process will be annotated sentences as illustrated in Figure 2, i.e., some words in the sentences are given semantic markers. Knowing these semantic markers means that we can move on to transforming the guideline into any further modelling approach we have in mind, for example, a formal representation. We believe it would be beneficial to help the transformation process from guidelines to UPPAAL models as done in [19].

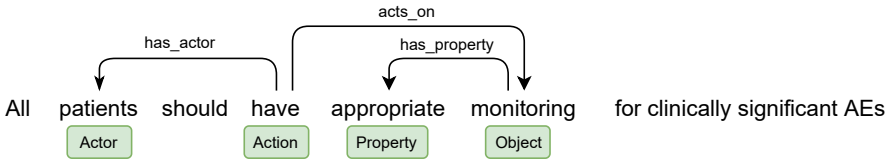

Fig. 2. Example of an annotated sentence

4 Semantic annotations

4.1 A hierarchy of concepts

In order to build a learning model to mark the roles of a word or a phrase in a sentence, we first need to define the classes of roles that we allow in our domain. Following [5], we made several ontology concepts to represent the static aspects of the guidelines. The design focuses on the concept of an actor doing some action(s) on some object(s) with some properties. Figure 3 shows our current ontology class hierarchy for our domain.

The ontology hiearchy in Figure 3 states that every class is a `Concept`. They are furthermore diversified into `ThingType` and `OperationType`. `OperationType` refers to the operations performed by an actor to another entity, whereas `ThingType` refers to any entity that can be an actor of an action, an object that is acted upon, or a property that can further explain an action, an actor, or an object.

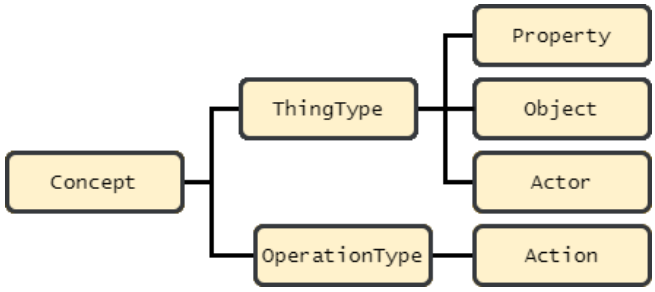

Fig. 3. Ontology class hierarchy

The class `OperationType` covers all operations performed in the sentence, either transitive or not. Its subclass is:

- `Action` denotes an operation performed on some `Object` by an `Actor` (if exists). Different from [5], we also consider the ownership type as an `Action`. E.g. "All patients should *have* appropriate monitoring for clinically significant AEs."

A `ThingType` can furthermore be classified as:

- `Actor` refers to the explicit performer of an `Action`. In many cases, the actor is invisible from the guideline sentences. E.g. "All *patients* should have appropriate monitoring for clinically significant AEs."
- `Object` denotes the entity that an `Action` is performed on. E.g. "All patients should have appropriate *monitoring* for clinically significant AEs."
- `Property` describes all modifiers of an `Action`, an `Actor`, or an `Object`. E.g. "All patients should have *appropriate* monitoring for clinically significant AEs."

Although our ontology classes can still be further diversified into lower subclasses as in [5], we find that they are not needed and too complex for our problem at the moment.

4.2 Relationship between classes

When designing the concept classes, we also need to introduce the relationship between them. This relationship defines the allowed interactions between one concept to another, and possibly from different level of concept. Table 1 shows the set of relationship between classes and concept in general.

`acts_on` defines that an `Action` is performed on either an `Object` or a `Property`. The inverse relation is `receives_action` that connects an `Object` or a `Property` to an `Action`. From here, we can say that *monitoring* `receives_action` from *have*.

The performer of an `Action` is defined by the `has_actor` relation to an `Actor`. Likewise, the `Actor` of an `Action` is defined by the `is_actor_of` relation. E.g. *have* `has_actor` *patients*.

The last two relations can cover the whole `ThingType` concept as their participants. This is because a `Property` can be used to modify an `Actor`, an `Object`, or a `Property` itself. Hence, we set the rule that any `ThingType` can have the `has_property` to a `Property` or a `Property` is connected to any `ThingType` by the `is_property_of` relation. E.g. *monitoring* `has_property` *appropriate*.

Table 1. Relationship between classes

Concept class	Relationship	Concept class
`Action`	`acts_on`	`Object, Property`
`Object, Property`	`receives_action`	`Action`
`Action`	`has_actor`	`Actor`
`Actor`	`is_actor_of`	`Action`
`ThingType`	`has_property`	`Property`
`Property`	`is_property_of`	`ThingType`

As each pair of the relations is an inverse of themselves, we will only use three of them in our end system, namely: `acts_on`, `has_actor`, and `has_property`.

4.3 Named-entity recognition

We also investigate a different approach to mark the important part of the sentence using named-entity recognition (NER) technique. NER is a task in NLP to detect the entity in the text that can be referred to with a proper name such as a person, a location, an organisation, or even things that are not proper entities such as dates, times, or prices [11].

In NER, the entities are usually marked using IOB format. The beginning of an entity type is marked with B-prefix tag, and I-prefix tag marks every token inside an entity type. An O tag is used for tokens that do not belong to any entity. Figure 4 shows a sentence marked with IOB format.

Fig. 4. A sentence marked with IOB format

We use the same ontology concepts for NER in IOB format. We consider four entities, i.e., `Action`, `Actor`, `Object`, and `Property` (as shown in Figure 3). As there are two tags for each entity, i.e. the B-tag and the I-tag, our label set size becomes 9 (from $2n + 1$ where n is the number of entities) namely `B-action`, `I-action`, `B-actor`, `I-actor`, `B-object`, `I-object`, `B-property`, `I-property`, and `O`. In reality, we only use 7 tags, namely `B-action`, `B-actor`, `B-object`,

`I-object`, `B-property`, `I-property`, and `O` as we do not have instances for either `I-action` or `I-actor`.

5 Learning

5.1 Syntactic analysis of guideline sentences

In this section, we will use the following common terminology. Part-of-speech (POS) is a category of words that have similar grammatical properties. Noun (e.g., noun NN or plural noun NNS), verb (VB), adjective (JJ), determiner (DT), adverb (RB), and punctuation (PUNCT) are some common POS in English language. The complete POS tags set that we use and their description can be found on Penn Treebank POS Tags[1].

A lemma is a word that can be inflected into several forms. E.g. *eat* as a verb is the lemma for *eat*, *eats*, *eating*, *ate*, and *eaten*.

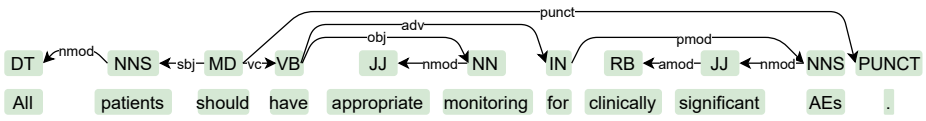

Fig. 5. An example of dependency tree

In order to build the features for the learning model which will be explained further in the next section, we need to perform syntactic analysis tasks on the sentences. These tasks are encapsulated as a pipeline which consists of several steps, namely:

- tokenisation that splits every component in the sentence into a single token. In *All patients should have appropriate monitoring for clinically significant AEs.*, there will be ten tokens: *All*, *patients*, *should*, *have*, *appropriate*, *monitoring*, *for*, *clinically*, *significant*, *AEs*, and *.*.
- POS tagging that marks up the words corresponding to a particular part of speech. Following the previous example, the POS tags are as follow: *All/DT*, *patients/NNS*, *should/MD*, *have/VB*, *appropriate/JJ*, *monitoring/NN*, *for/IN*, *clinically/RB*, *significant/JJ*, *AEs/NNS*, *./PUNCT*.
- lemmatisation which groups the same uninflected base form of each word into the same cluster. Using the previous example, the lemmas are as follow: *All/all*, *patients/patient*, *should/should*, *have/have*, *appropriate/appropriate*, *monitoring/monitor*, *for/for*, *clinically/clinically*, *significant/significant*, *AEs/aes*, *./.*.

[1] `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

– dependency parsing which parses the sentence based on the dependency relation of the words, i.e. every word is connected to each other by a direct link. Figure 5 shows the dependency parse tree for the sentence *All patients should have appropriate monitoring for clinically significant AEs.*. The dependency relationship marks the link between two words. For example, the link connecting *monitoring* to *appropriate* is marked by the relation *nmod*, or $\langle nmod \rangle \rightarrow \langle monitoring, appropriate \rangle$, which means that *appropriate* is a noun modifier for *monitoring*.

We follow the approach used in [5] to utilise the Mate Tools[2] [3] to perform the steps in the syntactic analysis. This tool has achieved state of the art performance on the shared task for syntactic analysis [8] so we can incorporate it in our system.

5.2 Semantic analysis of guideline sentences

Similar to the syntactic analysis, we adapt the approach done in [5] into our semantic analysis. This step is analogous to the semantic role labelling pipeline in [3], namely the *predicate identification*, *predicate disambiguation*, *argument identification*, and *argument classification*. In relation to our problem domain, each of these steps in the pipeline deals with one particular task as follows:

1. identifying words that are either `Action` or `Object`, which corresponds to the *predicate identification*. The reasoning behind choosing these two concepts is because they govern the relationship to other ontology concepts in the hierarchy. For example, by knowing if a word is an `Action` or an `Object`, we can further find the rest of the concepts through the relationships `acts_on`, `has_actor`, and `has_property`.
2. classifying words identified in step 1 to their correct concept, similar to the *predicate disambiguation*. For every verb and noun that can be either an `Action` or an `Object`, this step classifies them into the actual ontology concept, e.g. *have*/`Action`, *monitoring*/`Object`.
3. identifying words that are related to the instances in step 1, which corresponds to the *argument identification*. The instances that we are looking for in this step are the `Actor` of an `Action` and the `Property` related to any `TypeThing` concept. For example, this step will recognise *patients* as an `Actor` and *appropriate* as a `Property`.
4. classifying the relationship holds between a pair of instances from step 1 and step 3, which corresponds to the *argument classification*. The input of this step is a pair of words and its corresponding pair of concepts such as $\langle patients, have \rangle \rightarrow \langle$`Actor`, `Action`$\rangle$ and $\langle have, monitoring \rangle \rightarrow \langle$`Action`, `Object`$\rangle$.

Table 2. Feature sets and their usage

	`Action` and `Object`		Related concepts	
	Identification	Classification	Identification	Classification
word form	•	•	•	•
word lemmata	•	—	—	—
word POS	•	—	•	•
dependency relation	•	—	•	•
parent POS	•	•	—	—
child words	•	—	—	—
child POS	•	—	—	—
dependency words	—	—	•	•
position	—	—	•	•
word embedding	•	•	•	•

5.3 Features

In order to do the semantic analysis, we build one learning model for every step in the pipeline. This means we need to have a set of features for every learning model as it is more likely that one set of features for a task will not perform as well as when it is used for a different task. We based our feature sets on the intersection between the approach used by [5] and [6] for semantic role labeling task.

Most of the basic features have been implemented by Mate Tools as explained in Section 5.1. Furthermore, our additional features can be derived from the ones that have been provided. These features are as follow:

1. *affected word form*, which is the original word in the sentence
2. *affected word lemmata* taken from the lemmatisation step in syntactic analysis
3. *affected word part-of-speech* taken from the part-of-speech tagging in syntactic analysis
4. *relation to parent*, which is taken from the relation of dependency parsing in syntactic analysis
5. *parent part-of-speech*. The parent word can be derived from the dependency parsing in syntactic analysis
6. *child words*, the same with *affected word form* but for all children of current word. This is derived from the dependency parsing in syntactic analysis
7. *child part-of-speech*, the same with *parent part-of-speech* but for all children of current word
8. *dependency between words*, i.e. the words in dependency relations between the action and its object, the action and its actor, or the property and its action/actor/object.
9. *position of affected words*, e.g. *before* or *after* the predicate

[2] `http://code.google.com/p/mate-tools/`

10. *word vector representation*, which is the word embedding or the numerical representation for every word

Vectorising categorical features into numerical will inevitably generate a very sparse feature matrix. To compensate this phenomenon, we add the last feature 10 which is a dense feature matrix in nature. This feature is also used to add more generalisation to other features, for instance the POS features, which are specifically important for the semantic analysis task. Slightly different from the approach used in [5], we utilise the GloVe 6 billion tokens[3] [17] and the fastText 16 billion tokens[4] [15] as our word vector representation.

Table 2 shows the features and their usage in each semantic analysis steps.

5.4 Learning algorithm

To get the best learning model for our semantic analysis steps, we run our dataset against several classifiers. To achieve this, we annotate our guideline sentences following the ontology concepts needed for each particular step. For example, in the first and second step, we only annotated words in the sentences as either `Action` or `Object`. Then we give the label for those words as either 1 (for potential `Action` or `Object`) or 0 (for others). For the second step, the classifier will learn to distinguish the words recognised in step 1 as either 1 (for `Action`) or 0 (for `Object`).

After comparing several classifiers, we choose perceptron [20] as our learning algorithm as it shows the best result compared to the rests, e.g. decision tree, and random forests. We use the free perceptron library from scikit-learn. In perceptron, during the training step, for every input $\mathbf{x}_j$ and the expected output y_j in the training set, the algorithm will calculate the output $\hat{y}_j(t)$ using the weight matrix $\mathbf{w}(t)$ and activation (also called *step*) function f as in Eq. (1). In every iteration, the weight matrix is updated following Eq. (2) where w_i is the weight for feature i, $x_{j,i}$ is the ith feature value of jth training data, and η is the learning rate.

$$\hat{y}_j(t) = f[\mathbf{w}(t) \cdot x_j] \tag{1}$$

$$w_i(t+1) = w_i(t) + \eta \cdot (\hat{y}_j - y_j(t))x_{j,i} \tag{2}$$

The learning process will stop until it reaches a converging point, i.e. the value of $|\hat{y}_j - y_j| \leq \epsilon$ where ϵ is a very small threshold value. Otherwise, it will stop until it passes the maximum number of learning iteration.

5.5 Long short-term memory for NER

For our NER approach, we built a neural network learning model using long short-term memory (LSTM) [9]. LSTM is an architecture in recurrent neural

[3] `https://nlp.stanford.edu/projects/glove/`
[4] `https://fasttext.cc/docs/en/english-vectors.html`

network (RNN). RNNs are commonly used to analyse sequence and time series data. However, unlike RNNs, LSTMs can also capture long-term dependencies in the data. This is due to an LSTM unit/cell is made up of an input gate, output gate, and forget gate that make an LSTM cell can learn an important input, keep it as long as it is deemed important, and extract it when it is required.

Figure 6 illustrates our NER model using LSTM for the first 4 words in the sentence "*All patients should have appropriate monitoring for clinically significant AEs.*" Although Figure 6 shows that the input is represented by word embeddings and part-of-speech features, we also ran many experiments using the combination of all possible feature sets in Table 2. We also conducted experiments to see the effect of using different embedding dimensions.

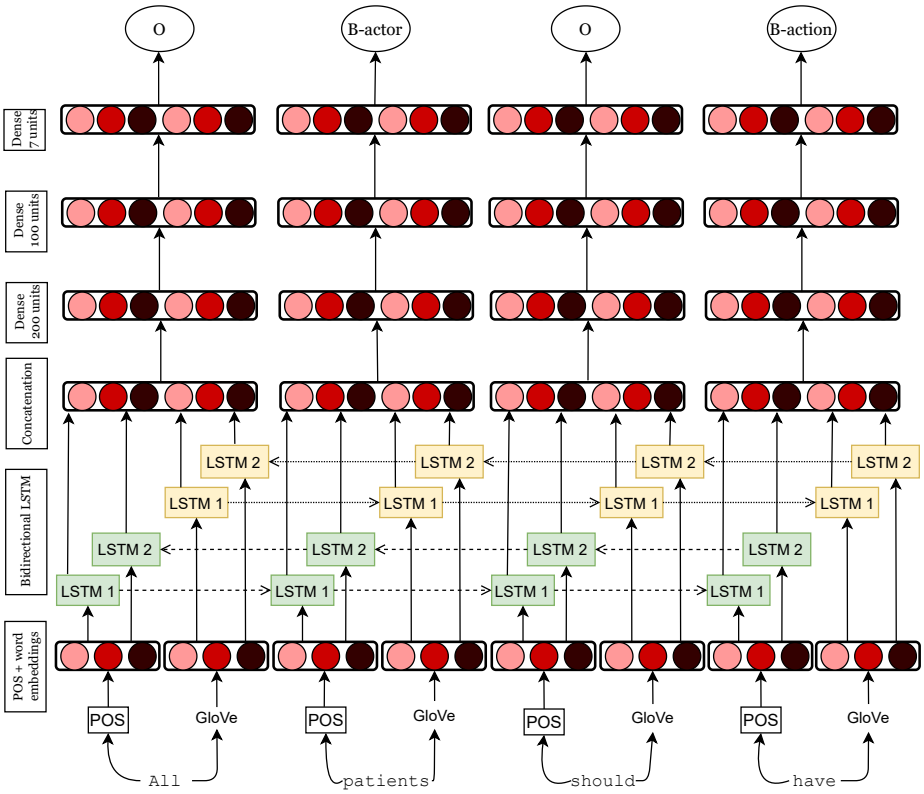

Fig. 6. Named-entity recognition using LSTM

For every word token and its part-of-speech in the sentence, we use bidirectional LSTM with 128 units to recognise the pattern in both forward and backward directions. The outputs of the LSTM layers are merged in the next layer. We added two subsequent dense layers with 200 and 100 units respectively.

These additional layers added the depth of our model to learn better from the inputs. Finally, the output layer contains 7 units for each label in our data domain as described in Section 4.3.

6 Evaluation

6.1 Dataset analysis

Our dataset has a total of 379 sentences, 216 of them are taken from The National Institute for Health and Care Excellence (NICE)[5], 98 are from The Scottish Intercollegiate Guidelines Network (SIGN)[6], and 65 are from Annals of the Rheumatic Diseases (ARD)[7]. These sentences are gathered from guidelines for various diseases to capture the nature of the sentences in a clinical guideline setting. Overall there are 7967 tokens and 1414 types, i.e. one sentence would have 21 words in average. The shortest sentence has 9 words in it whereas the longest has 66.

The annotation of the dataset was performed by a single annotator to mark the ontology of concepts on words following the hierarchy in Figure 3. Some difficulties became evident when dealing with an `Actor` or an `Object` as well as a `Property`. For example, the annotator sometimes mixed up tagging a word as an `Actor` in a passive sentence where it should be an `Object`, and vice versa. Determining if there is a `Property` in a phrase can also be challenging. For example, in the phrase *adjuvant therapy*, the annotator initially marked both words as `Object`s. On a further examination, it was then revised so that now *adjuvant* is the `Property` of the `Object` *therapy*. We believe that it also becomes more difficult if we want to have a more fine-grained concepts in our annotations. Although all considerations have been taken into account, we may not be surprised if there are still several inconsistencies and/or ambiguities in our final dataset.

Table 3. Counts of instances of concepts and relations

Concept	Instances	Relations	Instances
`Action`	630		
`Actor`	261	`has_actor`	128
`Object`	691	`acts_on`	676
`Property`	825	`has_property`	823
Total	**2407**	**Total**	**1627**

Table 3 shows the counts of instances of concepts and relations. It should be noted that there are many actions without explicit actors in our dataset.

[5] https://www.nice.org.uk/
[6] https://www.sign.ac.uk/
[7] https://ard.bmj.com/

Furthermore, some actors do not involve in any action, i.e. they just have some properties to modify them. As for our NER approach, the counts of named-entity instances can be seen in Table 4.

Table 4. Counts of named-entity instances

IOB tag	Instances
`O`	3785
`B-action`	472
`B-actor`	281
`B-object`	1507
`I-object`	769
`B-property`	928
`I-property`	225
Total	**7967**

6.2 Experiments

After finalising our dataset, we ran several machine learning classifier algorithms to evaluate the performance of our semantic annotations approach as we briefly mentioned in Section 5.4. We use the common evaluation metrics precision and recall. Precision is defined as the percentage of predicted instances that are correct whereas recall is defined as the percentage of correct instances that are predicted by the model. It is also often that we combine precision and recall into a single metric called F_1-score, particularly to simplify the comparison between several classifiers. The F_1-score is computed as the harmonic mean of precision and recall.

Table 5. Evaluation F_1-score values for several classifiers

Classifier	Precision	Recall	F_1-score
Decision tree	0.745	0.512	0.603
Random forest	**0.792**	0.488	0.604
Perceptron	0.603	**0.678**	**0.638**
GloVe 300 + POS	0.854	0.881	**0.867**
GloVe 200 + POS & Lemma	0.855	0.879	0.867
GloVe 300 + POS & Parent	0.846	**0.888**	0.866
Wiki 300 + POS & Dependency	0.853	0.876	0.864
Wiki 300 + POS & Parent & Dependency	**0.856**	0.872	0.864

Table 6 shows the performance of several classifiers for our semantic annotation approach. For each classifier, we perform evaluation using tenfold cross-

Table 6. Evaluation F_1-score values for several relation classification

Classifier	F_1-score
300d 300/200/100/100	**0.885**
100d 100/100/100	0.881
100d 100/100/100	0.877

validation setting, i.e. in every fold, there will be ten equal portions of data where one portion out of ten will be used as testing. We can see variations of trend for each performance metric. Random forests has the best performance (79%) for correctly predicting the concepts and relations in the sentences, i.e. 4 out of 5 annotations are correct. Meanwhile, perceptron is the best for predicting all correct concepts and relations (67%), roughly 7 out of 10 correct annotations can be predicted. Using the F_1-score, the best one is achieved by perceptron (64%) with mean 0.97 and $\pm$ 0.007 standard deviation.

Table 6 also shows some experiments of NER using LSTM for our domain. Here, we only show 5 experiments although in reality we ran many more to get the best result. We tried using every possible combinations of features in Table 2. We also investigated the effect of using pre-trained word embeddings with varying dimension size. We found the best F_1-score of 87% using the combination of POS feature and GloVe embeddings of size 300.

As we adapted the approach done by [5], it would be interesting to see how our performance would be if we run it against their dataset. It will also answer the question on how much the domain used to build the model affects its performance when used in a different one. This was outside the scope of the present paper, but will be explored in future work.

7 Conclusion

In this paper, we presented our work to annotate semantic information in guideline sentences. We began by collecting guideline sentences from the English, Scottish, and European guideline corpora. These sentences serve as the preliminary dataset for applying linguistic analysis in the domain. Although we only have 379 sentences in our dataset, we have done around 4000 annotations for the concepts we are interested in.

Following the approach in [5], we annotated the dataset using a hierarchy of concepts. We adapted their ontology concepts using only concepts that we found useful for our problem at present. We also conducted a named-entity recognition task using the same ontology concepts to compare the results. Furthermore, the more fine-grained concepts we want to apply, the more challenging it becomes. As our current development only has one annotator for the whole dataset, to increase the accuracy of our annotation we should consider adding one or more annotators in the future.

The main aim for this work is to help people retrieve the key information in clinical guidelines. As shown in [19], we built a system to do formal verification

of a therapy algorithm for type 2 diabetes. As guidelines are expressed in natural language, they are prone to ambiguity, incompleteness, and inconsistency. We expect that our work will help further the development of clearer and better clinical guidelines.

In future work, our aim is to build a framework that integrates the whole process defined in [19]. It means that we will also add functionalities to produce some formal models from the annotated guideline sentences. We need to further assess the performance of our approach when compared to different datasets. Finally, we also plan to build a user interface to help the annotation process and to visualise the annotation result.

References

1. Bäumler, S., Balser, M., Dunets, A., Reif, W., Schmitt, J.: Verification of medical guidelines by model checking – a case study. In: Valmari, A. (ed.) Model Checking Software. pp. 219–233. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer (2004)
3. Björkelund, A., Hafdell, L., Nugues, P.: Multilingual semantic role labeling. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task. pp. 43–48. Association for Computational Linguistics, Boulder, Colorado (June 2009), https://www.aclweb.org/anthology/W09-1206
4. Carvalho, G., Carvalho, A., Rocha, E., Cavalcanti, A., Sampaio, A.: A formal model for natural-language timed requirements of reactive systems. In: Merz, S., Pang, J. (eds.) Formal Methods and Software Engineering. pp. 43–58. Springer International Publishing (2014)
5. Diamantopoulos, T., Roth, M., Symeonidis, A., Klein, E.: Software requirements as an application domain for natural language processing. Lang. Resour. Eval. **51**(2), 495–524 (Jun 2017). https://doi.org/10.1007/s10579-017-9381-z
6. Gildea, D., Jurafsky, D.: Automatic labeling of semantic roles. Computational Linguistics **28**(3), 245–288 (2002), 10.1162/089120102760275983
7. Giordano, L., Terenziani, P., Bottrighi, A., Montani, S., Donzella, L.: Model checking for clinical guidelines: an agent-based approach. AMIA Annual Symposium pp. 289–93 (2006)
8. Hajic, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M.A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., et al.: The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task. pp. 1–18 (2009)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**, 1735–1780 (12 1997). https://doi.org/10.1162/neco.1997.9.8.1735
10. Holzmann, G.J.: The spin model checker: primer and reference manual. Addison-Wesley (2003)
11. Jurafsky, D., Martin, J.H.: Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition. Pearson Prentice Hall, Upper Saddle River, N.J. (2009)

12. Kingston, A., Robinson, L., Booth, H., Knapp, M., Jagger, C.: Projections of multimorbidity in the older population in England to 2035: estimates from the population ageing and care simulation (pacsim) model. Age and Ageing **47**, 374–380 (2018)
13. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)
14. McMillan, K.L., Qadeer, S., Saxe, J.B.: Induction in compositional model checking. In: Emerson, E.A., Sistla, A.P. (eds.) Computer Aided Verification. pp. 312–327. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
15. Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., Joulin, A.: Advances in pretraining distributed word representations. In: Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018) (2018)
16. Moura, L.D., Bjørner, N.: Z3: An efficient smt solver. In: TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
17. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543 (2014), `http://www.aclweb.org/anthology/D14-1162`
18. Pérez, B., Porres, I.: Authoring and verification of clinical guidelines: A model driven approach. Journal of Biomedical Informatics **43**(4), 520 – 536 (2010). https://doi.org/10.1016/j.jbi.2010.02.009
19. Rahman, F., Bowles, J.K.F.: Formal verification of cnl health recommendations. In: Polikarpova, N., Schneider, S. (eds.) Integrated Formal Methods (iFM 2017). pp. 357–371. Springer International Publishing (2017)
20. Rosenblatt, F.: The perceptron — a perceiving and recognizing automaton. Tech. Rep. 85-460-1, Cornell Aeronautical Laboratory (1957)
21. Shahar, Y., Miksch, S., Johnson, P.: The asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. Artificial Intelligence in Medicine **14**(1), 29 – 51 (1998). https://doi.org/10.1016/S0933-3657(98)00015-3
22. Terenziani, P., Molino, G., Torchio, M.: A modular approach for representing and executing clinical guidelines. Artificial Intelligence in Medicine **23**(3), 249 – 276 (2001). https://doi.org/10.1016/S0933-3657(01)00087-2