Ontology Embeddings with ontowalk2vec: an Application to UI Personalisation

Blerina Gkotse[a][b][*], Pierre Jouvelot[a] and Federico Ravotti[b]

[a] *Mines Paris, PSL University, Paris, France*

[b] *Experimental Physics Department, CERN, Geneva, Switzerland*

E-mails: blerina.gkotse@cern.ch, pierre.jouvelot@minesparis.psl.eu, federico.ravotti@cern.ch

January 31, 2022

Abstract. Within software applications, user experience is greatly improved when user interface (UI) personalisation is possible, and even more so when recommender systems can help users find the set of settings best suited for their skills and goals. In this paper, we suggest that such recommender systems should be based on ontologies dedicated to describing both software traits and user preferences, an example of which is the Ontology-based Web Application Generation ontology (OWAO) that specifies what web applications and their UI are. The key scientific contribution of our approach is ontowalk2vec, an algorithm that maps instances of ontologies to feature vectors (embeddings) that can be later on used for classification purposes, a process inherent to recommender systems. In addition to OWAO, we validate ontowalk2vec on two other significant ontologies, namely MUTAG and DBpedia, where we demonstrate it outperforms existing techniques. We finally discuss how using ontowalk2vec on OWAO can form the basis of personalised UI recommender systems, stressing, in particular, the importance of properly setting the many hyperparameters that typically characterise embedding-generation algorithms.

Keywords: Web Semantics, ontology, embeddings, user interface, personalisation

1. Introduction

User experience is an important aspect to consider when aiming to provide intuitive and user-friendly user interfaces (UI) for software applications. Since ontologies can be used to provide formal and standard definitions of concepts pertaining to any domain of knowledge, they can, in particular, be used for the precise description of the UI components that form the fabrics of the vast majority of UIs. The literature reports on several attempts to specify (part of) this kind of knowledge [1–4]. These UI ontologies generally fulfil specific requirements and describe certain parts of a UI or web application. For example, the Semantic UI ontology includes concepts related to the interface elements of its own UI framework [1]. Another example is GenAppi, a methodology for generating web applications based on a web application ontology [5].

UI personalisation has been shown to be a key ingredient of positive user experience for complex software [6]. Such a service can be even more valuable to both advanced and not-so-advanced users when recommender systems can help them find the set of settings best suited for their current skills and goals. In addition to providing finely tuned UI parameters for each user, recommender systems for UI personalisation can keep track of user activities as their skills evolve over time, and thus suggest UI adaptations along the way, leading to a sustainable increase of their productivity [7].

In addition to the many existing approaches used to design recommender systems (see for instance [8]), we suggest in this paper that the design *and implementation* of UI recommender systems should be based on ontologies dedicated to describing both software traits

[*]Corresponding author. E-mail: blerina.gkotse@cern.ch

and user preferences. We present here a case study about this approach using an Ontology-based Web Application Generation ontology (OWAO) developed to specify web applications and their UI [5].

Since recommender systems rely heavily on Machine Learning classification techniques to map any user to the set of users that have similar preferences, an important issue is the extraction of classification features from ontologies and their instances. Natural Language Processing (NLP) models, such as word2vec, have shown consistent results on predicting the contexts of a text fragment or a word by representing them as feature vectors (also named, in such a setting, *embeddings*) [9]. Providing some prior knowledge to these models by the introduction of ontologies or knowledge graphs has been shown to help improve their accuracy [10].

In this paper, we introduce ontowalk2vec, a new model for generating ontology embeddings that can be used for classification purposes and recommender systems. We test our model using two benchmark ontologies (MUTAG and DBpedia) and OWAO, and we obtain results suggesting that our approach improves the current state of the art. We finally focus on how to best configure the ontowalk2vec hyperparameters for OWAO in order to provide the best embeddings for proper UI personalisation.

The main contributions of this paper are therefore:

– ontowalk2vec, a NLP-based model that maps ontologies to textual feature vectors;
– the validation of ontowalk2vec on three ontologies, namely MUTAG, DBpedia and OWAO;
– an OWAO-based use case to illustrate how ontowalk2vec can be efficiently used as the basis of a personalised UI recommender system.

The structure of the paper is as follows. After this introduction, Section 2 describes some state-of-the-art NLP models for specifying ontology embeddings. In Section 3, we present the metrics generally used for the performance assessment of embedding-generation algorithms. In Section 4, ontowalk2vec is described, providing a pseudo-code of the algorithm. Section 5 details the experimental evaluation of ontowalk2vec by the use of three baseline ontologies. In Section 6, we focus on the application of the ontowalk2vec model for UI personalisation, in particular discussing optimisation techniques and the final results. We conclude in Section 7, summarising our contributions and discussing possible future work.

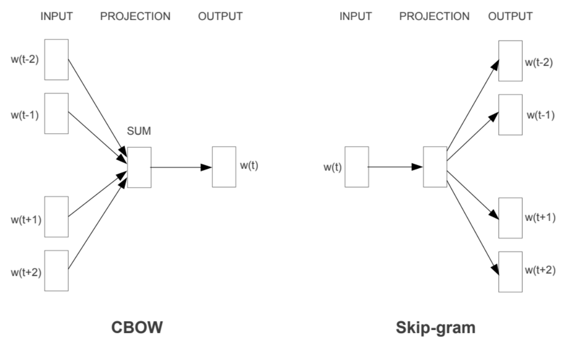

Fig. 1. word2vec architectures. The CBOW architecture predicts the current word based on the context; the Skip-gram mode predicts surrounding words, given a current word [9].

2. Related Work

Ontology-based recommender systems rely on the inferences they can make from the knowledge present in ontologies and knowledge graphs (i.e., ontologies and their instances). In a Machine Learning setting, ontology embedding-generation algorithms (or models) such as ontowalk2vec are used to represent this information as feature vectors, usually based on NLP techniques.

2.1. word2vec

One of the most influential works concerning embeddings in the recent years is the NLP model called word2vec [9]. This model takes as input sentences, seen as sequences of words, and computes a vector for each word. In word2vec, two different architectures are proposed: the Continuous Bag of Words (CBOW) and Skip-gram (see Figure 1). Provided a set of words as context, CBOW predicts a word that could fit next in that context. In the Skip-gram approach, when a word is provided, its fitting in a specific classification is used to predict the context.

2.2. node2vec

The word2vec model is efficient for extracting word features in text, for example from a newspaper. However, it is not sufficient when the input is a structured object such as a graph. As an ontology can be assimilated to a directed graph, one needs to extract embeddings from graphs. For such graph embeddings, a reference model is node2vec [11]. This model uses two classic traversal strategies in graphs: Breadth-first

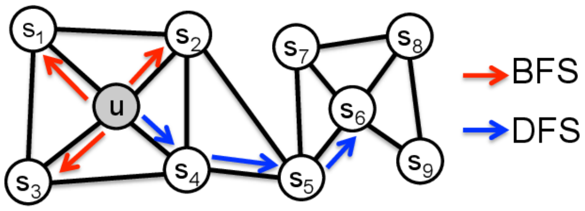

Fig. 2. Starting from node u, a Breadth-First Search (BFS) traversal is depicted using red arrows, while a Depth-First Search (DFS) path is depicted by blue arrows (from [11]).

Search (BFS) and Depth-first Search (DFS). These two algorithms provide two different ways of visiting the nodes of a graph. In BFS, starting from a specific node u, all the nearest neighbouring nodes s_i are visited first, as depicted by red arrows in Figure 2, in contrast to BFS, DFS visits the nodes in depth, as depicted there by blue arrows. These traversal algorithms are then used to extract random walks from any graph in order to build node sequences that can then be considered as artificial "sentences" and, as such, input to word2vec.

2.3. *RDF2Vec*

Resource Description Framework (RDF) is a common format for encoding ontologies. Therefore, RDF2-Vec has been introduced for extracting embeddings from RDF graphs [10]. In order to generate adequate random walks, RDF2Vec uses a BFS approach up to a certain depth, as in node2vec; the length of the paths generated by these random walks is fixed. In addition, RDF2Vec also relies upon the Weisfeiler-Lehman algorithm, which computes sub-tree kernels for graph comparison [12]. RDF2Vec plays an important role in the generation of random walks for ontologies and is thus, as node2vec, integrated in our ontowalk2vec model, described below.

2.4. *Hyperparameters*

All these models rely on several “hyperparameters” for fine tuning. For instance, the node2vec algorithm relies on two hyperparameters: p, which is the probability that a node in a walk is revisited, and q, which specifies whether a BFS or DFS approach should be employed. All hyperparameters (e.g., iteration number, learning rate, window size, etc.) need to be initialised before starting training the model and providing embeddings. Even though in many works these hyperparameters are initialised with their default values, studies have shown that the best values for them strongly depend on the type of data the models are trained on. These parameters should thus be chosen with caution; otherwise they may affect the final accuracy of the modelling [13, 14]. For these reasons and in order to provide a fair assessment of our proposed methodology, an experimental investigation about the choice of their initial values is presented in Section 6.

3. Classification Evaluation Metrics

To formally assess the accuracy of embeddings, classification tasks are often used (see, for instance, [10]). Yielding a high-quality classification model is important to ensure that our new model ontowalk2vec is robust; in our case, this would ensure that we managed to encode high-dimensional ontology instances as representative, yet automatically generated, feature vectors of lower dimension.

In this paper, two classical methodologies, namely Support Vector Machine (SVM) and Random Forest (RF), implemented in a Scikit-learn python module [15], were used for classification. Provided that some labelled training data are available, SVM outputs the hyperplane[1] that best classifies its training set and, likely, any new samples [17]. A Random Forest classifier uses instead a number of decision trees on smaller groups of data contained in its input data set and averages the results in order to improve accuracy [18].

Common evaluation metrics for classifiers are the accuracy score, confusion matrix and t-distributed Stochastic Neighbor Embedding (t-SNE) plot, briefly summarised in the following subsections.

3.1. *Accuracy Score*

For a given classification technique, its accuracy score over a labelled test data set of N samples, also implemented in the Scikit-learn module [15], is the ratio of correctly predicted labels over N. If $\hat{y}_i$ is the predicted value for the i-th sample and y_i the actual value, then accuracy$(y, \hat{y})$ is the fraction of correct predictions over N, defined as:

$$\text{accuracy}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} 1(\hat{y}_i = y_i), \tag{1}$$

[1]A hyperplane is any co-dimension-1 vector subspace of a vector space [16]. Models compute multidimensional vectors, and therefore a hyperplane can be used to try to separate these for classification purposes.

Model	Actual Negative	Actual Positive
Predicted Negative	$C_{0,0}$ (True Neg.)	$C_{0,1}$ (False Neg.)
Predicted Positive	$C_{1,0}$ (False Pos.)	$C_{1,1}$ (True Pos.)

Table 1
Confusion matrix for binary classification.

where, as usual, $1(\hat{y}_i = y_i)$ is equal to 1 if its argument is true, and 0 otherwise.

According to Equation 1, a high accuracy, i.e., a large number of correct predictions, is such that the value of the accuracy score should approach 1. On the contrary, low accuracy means that the value approaches 0 (note that a random predictor has an accuracy of 0.5).

3.2. Confusion Matrix

Another metric to evaluate classification performance is the confusion matrix. By definition, the element $C_{i,j}$ of a confusion matrix C is the number of observations known to be in group i and predicted to be in group j. We focus here on binary classification; a confusion matrix then includes the count of true negatives $C_{0,0}$, false negatives $C_{0,1}$, false positives $C_{1,0}$ and true positives $C_{1,1}$.

According to this definition, the quality of a model is considered to be high when the numbers $C_{0,0}$ and $C_{1,1}$ of true negatives and positives are high (approaching the number of actual negative and positive data points, respectively, and depicted in Table 1 in green), while the numbers $C_{1,0}$ and $C_{0,1}$ (depicted in Table 1 in red) of false positives and negatives should be as low as possible, approaching a value of 0.

3.3. t-distributed Stochastic Neighbor Embedding

In order to provide a visual evaluation for high-dimensional data classification, the notion of t-distributed Stochastic Neighbor Embedding (t-SNE) is often used [19]. It converts similarities between data points to joint probabilities and minimises the Kullback-Leibler divergence[2] between the joint probabilities of the low-dimensional embedding results and the high-dimensional data [20]. A classifier is considered as providing accurate results when the points of a t-SNE diagram corresponding to the same class (in this case, same label) are grouped together.

[2]The Kullback-Leibler divergence, also known as relative entropy, denotes the "distance" between the probability distributions being compared.

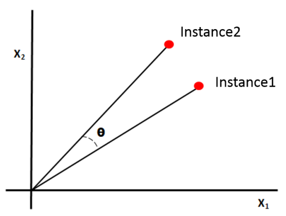

Fig. 3. Cosine similarity.

3.3.1. Cosine Similarity

By comparing feature vectors using the cosine similarity metric, one can get an approximation of the proximity and relatedness of the corresponding data instances. Assuming that an ontology instance is represented by the embedding as vector A and a second instance, as vector B, the cosine similarity is $\cos(\theta)$, where θ is the angle between the two vectors, defined as

$$\cos(\theta) = \frac{A.B}{\|A\|\|B\|}, \tag{2}$$

where $A.B$ denotes the scalar product of the two vectors A and B, and $\|x\|$ is the norm of vector x.

According to Equation 2, the closer the vectors are, the more the similarity value approaches 1, and the more similar the instances should be. When the cosine similarity value approaches 0, the two instances are considered less similar.

3.3.2. F1 Score

The most pertinent metrics for evaluating recommender systems are used here [21]. The first metric is precision, p, defined as:

$$p = \frac{\|true\ positives\|}{\|true\ positives\| + \|false\ positives\|}. \tag{3}$$

where $\|L\|$ denotes the number of elements in set L.

The second evaluation metric is recall, r, defined as:

$$r = \frac{\|true\ positives\|}{\|true\ positives\| + \|false\ negatives\|}. \tag{4}$$

A good model is one for which both precision and recall values are high (approaching 1). For this reason, a combined metric is commonly used for globally evaluating recommender systems, namely the F1 score F_1, providing a normalised average and defined as:

$$F_1 = 2 * \frac{p * r}{p + r}. \tag{5}$$

4. ontowalk2vec

To perform UI personalisation for ontology-based web applications, we introduce ontowalk2vec, a new technique for computing ontology embeddings.

4.1. Presentation

The ontology embedding model ontowalk2vec is inspired by the two main models detailed in the sections above, i.e., RDF2Vec [10] and node2vec [11]. Both models focus on creating different random walks on an ontology or a graph, utilising them as input sentences to word2vec [9]. These two methodologies are considered complementary within our method. The node2vec model focuses on the structural part of the ontology by treating it as a directed graph and extracting random walks - sentences of different lengths - from it, including the taxonomy and inheritance properties of the ontology classes. On the other hand, RDF2Vec focuses on the RDF triples, which include the instances and their relations in a subject-predicate-object format; this method also relies on word2vec, used for generating the final feature vectors.

As previously described, word2vec supports two specific training architectures: CBOW and Skip-gram. In the CBOW architecture, the order of the word instances does not influence the result. However, in ontologies and knowledge graphs, the structure and order of the instances is intrinsically linked to their semantic meaning, and therefore CBOW is not suitable for our purpose. For example, in a RDF triple, the subject and object components are strictly non-interchangeable, otherwise the whole semantic meaning changes. In contrast to CBOW, the Skip-gram architecture focuses on the context of instances and provides better semantic relevance in comparison to CBOW [9]. For these reasons, Skip-gram has been selected as our training architecture of choice.

As displayed in Figure 4, in ontowalk2vec, random walks are extracted from the input ontology and its instances using both the node2vec and RDF2Vec models and fed to word2vec. After training word2vec with specific random walks, the feature vectors are generated and can be then used in a recommender system.

4.2. Algorithm

The pseudo-code for the ontowalk2vec model is given as Algorithm 1. Note that this is a simplified version of the actual code, intended to provide here a mostly high-level understanding of ontowalk2vec. The full code as well as the data used for the experiments detailed in the following sections can be found in the accompanying online resources[3].

Once started, the algorithm reads the file of the ontology for which the embedding generation will be performed. In Line 3, the ontology is translated as a directed node2vec graph object, using the parameter *directed*, which specifies that the graph is directed, and the two hyperparameters p and q. The latter are used for pre-processing the transition probabilities that guide the computation of the random walks, as indicated in Line 4. Based on these probabilities, random walks on the graph are gathered in $node2vec\_walks$.

Subsequently, random walks using the Breadth-First Search (BFS) method are generated by RDF2Vec (bfs_walks). The ontowalk2vec embedding algorithm is thus assumed here to have access to a data set *data* that includes data points (ontology instances) on which random walks can be performed. Both $node2vec\_walks$ and bfs_walks are used as input sentences of word2vec, which produces one set of final embeddings ($bfs\_embeddings$). The same process is performed for the Weisfeiler-Lehman (WL) method, used as an alternative to BFS in RDF2Vec for generating random walks. Finally, the generated embeddings are stored in $wl\_embeddings$. Both embeddings are then returned as result of Algorithm 1.

4.3. Classification

The accuracy of both types of embeddings is evaluated in Algorithm 2, based on the evaluation process of pyRDF2Vec [22], by the use of the two classifiers Random Forest and SVM, detailed in Section 3. They rely on the same data set *data* and their associated classification labels. These data points and labels are split into different subsets ($training\_data$ and $test\_data$ for the data points, and $training\_labels$ and $test\_labels$ for the labels), to be used for the training process. After the training, the classifiers are expected to predict accurately the labels of the $test\_data$. The accuracy score and confusion matrix are computed, while a t-SNE plot is also generated.

Given a pair of instances ($instance_1$, $instance_2$), the similarity value is computed (in Line 15) in order to evaluate how similar these two instances are. The motivation for such a test is that, for providing embedding-

[3]https://gitlab.cern.ch/irrad/ontowalk2vec

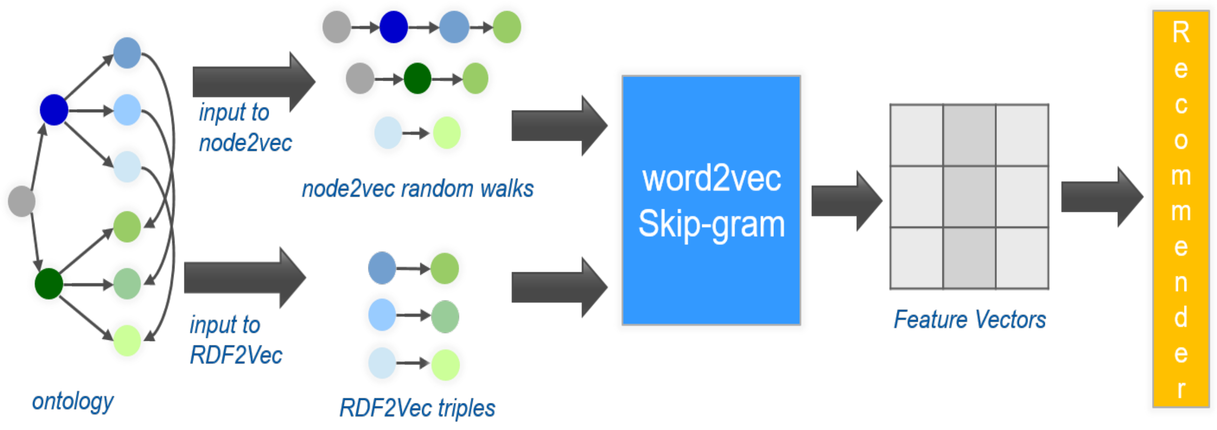

Fig. 4. Embeddings generation with ontowalk2vec.

Algorithm 1 ontowalk2vec model
Input: *ontology* file, *p* and *q* hyperparameters, ontology instances *data*
Output: ontology-based feature vectors

```
1: Read $ontology$
2: $graph \leftarrow ontology\_to\_graph\_conversion(ontology)$
3: $node2vec\_graph \leftarrow node2vec.graph(graph, directed, p, q)$
4: $node2vec\_graph.preprocess\_transition\_probabilities(p, q)$
5: $node2vec\_walks \leftarrow node2vec\_graph.simulate\_walks()$
6: $bfs\_walks \leftarrow RDF2VecBFS(graph, data)$
7: $bfs\_embeddings \leftarrow word2vec(bfs\_walks \cup node2vec\_walks)$
8: $wl\_walks \leftarrow RDF2VecWL(graph, data)$
9: $wl\_embedding \leftarrow word2vec(wl\_walks \cup node2vec\_walks)$
10: Return $(bfs\_embeddings, wl\_embeddings)$
```

based recommendations, given an ontology instance $instance_1$, the instances exhibiting the highest similarity value with $instance_1$ will be selected.

5. Experimental Evaluation

In this section, we apply Algorithms 1 and 2 to evaluate ontowalk2vec with two reference ontologies, namely MUTAG and DBpedia, and OWAO. For each ontology, after the training phase, the RF and SVM classifiers are used to predict the labels of the test data. The prediction quality is measured by accuracy scores and confusion matrices, while a classification in two categories is visualised through t-SNE plots, as detailed in Section 3. These results suggest that ontowalk2vec outperforms the current state of the art models, node2vec and RDF2Vec.

5.1. MUTAG

The MUTAG ontology is part of DL-Learner, a framework for supervised machine learning based on the OWL language, RDF and Description Logics [23]. MUTAG contains information about 340 complex molecules that could be carcinogenic or, as it is called in the ontology, "MUTAGenic". An instance of such a molecule (here, in this specific example, the molecule d30) and its structured composition is shown in Figure 5. The classification of these molecules is here relative to the boolean data property `mutag:isMutagenic` present in MUTAG. Classification is based on labels, 0 or 1, set according to the value of the data property `mutag:isMutagenic`. This value is removed from the data set so that the classification is only performed through the embedding of the molecules' instances.

Algorithm 2 ontowalk2vec evaluation by classification
Input: *ontology*, *p* and *q*, *training_data*, *test_data*, *training_labels*, *test_labels*, $instance_1$, $instance_2$
Output: classification analysis data

```
1:  $(bfs\_embeddings, wl\_embeddings) \leftarrow ontowalk2vec(ontology, p, q, training\_data \cup test\_data)$
2:  $rf \leftarrow RandomForestClassifier()$
3:  $svm \leftarrow SVM()$
4:  for $e$ in [$bfs\_embeddings$, $wl\_embeddings$] do
5:      Separate $e$ to $training\_e$ and $test\_e$
6:      for $c$ in [$rf$, $svm$] do
7:          $c.fit(training\_e, training\_labels)$
8:          $c.predictions \leftarrow c.predict(test\_e)$
9:          $c.accuracy \leftarrow c.accuracy\_score(test\_labels, c.predictions)$
10:         $c.confusion\_matrix \leftarrow confusion\_matrix(test\_labels, c.predictions)$
11:     end for
12:     $tsne \leftarrow TSNE().fit\_transform(e)$
13:     Plot $tsne$
14:     $top\_similarities \leftarrow e.most\_similar(instance_1)$
15:     $similarity\_value \leftarrow e.similarity(instance_1, instance_2)$
16: end for
```

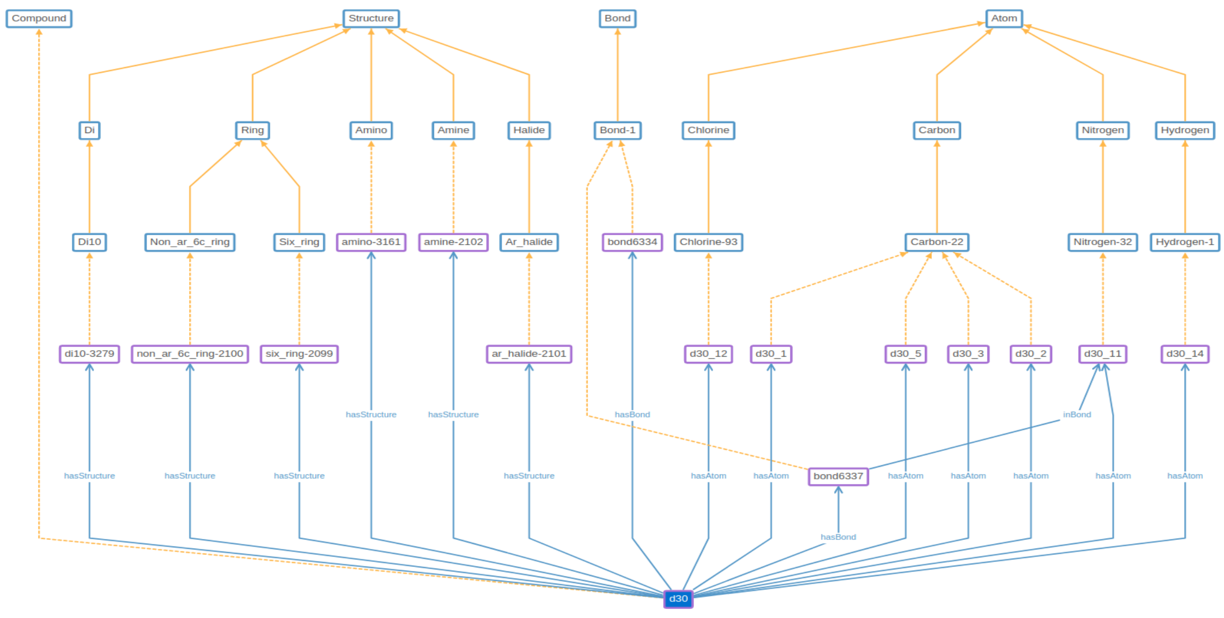

Fig. 5. Excerpt from MUTAG depicting the instance d30 of the Compound class.

Model	Random Forest	SVM
node2vec	0.69 (±0.03)	0.71 (±0.02)
RDF2vec BFS	0.69 (±0.03)	0.71 (±0.02)
RDF2vec WL	0.71 (±0.01)	0.72 (±0.02)
ontowalk2vec BFS	**0.74** (±0.02)	**0.76** (±0.02)
ontowalk2vec WL	**0.74** (±0.02)	**0.74** (±0.01)

Table 2
Models' accuracy scores using the MUTAG data set.

For this experiment, a dedicated data set was used, containing the 340 molecules with their labels. The data set was divided in 80% training data and 20% test data, before running Algorithms 1 and 2. We performed embeddings using node2vec, RDF2Vec Breadth-First Search (BFS), RDF2Vec Weisfeiler-Lehman (WL), ontowalk2vec BFS and ontowalk2vec WL. In order to obtain statistically significant results, according to standard techniques in Machine Learning (extracting random walks is a stochastic process), the same experiments were run 10 times [24].

The accuracy scores for the RF and SVM classification methods are summarised, via their average value and standard deviation, graphically in Figure 6, while the exact values are also provided in Table 2. Both ontowalk2vec BFS and WL provide, for the MUTAG ontology, on average, higher accuracy, in both classification problems, RF and SVM. More specifically, one can observe in Figure 6 and Table 2 that the accuracy using RF classification for both ontowalk2vec BFS and WL is 0.74, and the same value is output by the ontowalk2vec WL with SVM classification. The best result in this experiment (0.76) is provided by ontowalk2vec BFS with SVM classification.

For a more exhaustive comparison, we also present the confusion matrices for understanding which predictions are affecting positively or negatively the results. According to Table 3, ontowalk2vec appears to have high prediction scores, on average, for predicting the negative data points, which are actually the non-mutagenic molecules. This is outlined in Table 3 for the RF classification with BFS (41.1) and with WL (41.2), while using the SVM classification, one gets 41.2 and 43.1 for BFS and WL, respectively. Furthermore, the false positives are low, as depicted in Table 3 for RF classification with BFS (4) and with WL (3.8), while using the SVM classification, one gets 3.8 and 1.9 for BFS and WL, respectively. However, the results are in line with the other tested models for predicting the molecules that are considered mutagenic. This is probably due to the fact the MUTAG ontology contains less data points representing mutagenic molecules; thus the model does not get well trained to learn with high accuracy to predict the positive labels.

Our generated embedded vectors for all the given data points are also visualised using t-SNE plots. In Figures 7, 8 and 9, the red points represent the molecules of the test data that were classified as non-mutagenic while the green points show the molecules that were classified as mutagenic. With a successful classification, t-SNE should be able to separate the points of different colours in different colour groups, and there should be some significant distance among the colour groups.

Figure 7 is the t-SNE visualisation produced when using node2vec. As can be observed from the figure, the classified data points appear quite mixed, and there is no clear distinction between the colour groups. This means that the vectors representing mutagenic and non-mutagenic molecules do not have high similarity distance. The same observation can be made for Figure 8, where the results from RDF2Vec BFS are shown. The t-SNE suggests that the RDF2Vec-based embedding-generation model does not seem to provide any clear classification of the two vector categories.

Figure 9 depicts the t-SNE plot of the ontowalk2vec BFS method. Looking at it, there seems to exist some similarity among the embedded vectors; the non-mutagenic molecules tend to be more numerous in the lower part of the plot, while the mutagenic ones appear more often in the top part.

Overall, this analysis seems to suggest that ontowalk2vec is a better model for classifying the MUTAG data than node2vec and RDF2Vec. Admittedly, this improvement is somewhat limited, around 5 %, based on our classification analysis of the MUTAG data.

5.2. *DBpedia*

In order to further challenge our hypothesis that ontowalk2vec provides better feature vectors than current approaches as well as to also check that it can tackle larger ontologies, a second experiment is set up with DBpedia [25] as input. DBpedia stores triples of general-purpose knowledge extracted from Wikipedia. As in the RDF2Vec experimental setup [10], a data set of cities is used for performing classification on their quality of life[4]. According to the provided ranking,

[4]https://mobilityexchange.mercer.com/insights/quality-of-living-rankings

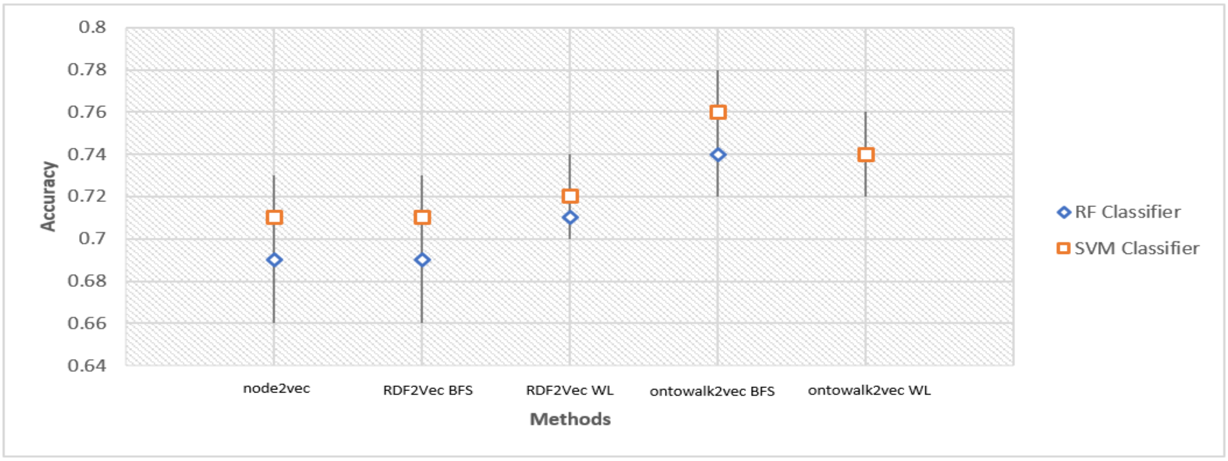

Fig. 6. Accuracy plots for classification based on the various models.

Actual Value	Negative	Negative	Positive	Positive
Predicted Value	Negative	Positive	Negative	Positive
node2vec - RF	35.4 (±2)	9.6 (±2)	11.4 (±0.5)	11.6 ((±2))
node2vec BFS - SVM	40.7 (±2.3)	4.3 (±2.3)	14.8 (±1.9)	8.2 (±1.9)
RDF2vec BFS - RF	35.9 (±2.2)	9.1 (±2.2)	10.4 (±1.9)	12.6 (±1.9)
RDF2vec BFS - SVM	40.7 (±1.8)	4.3 (±1.8)	14.5 (±1.3)	8.5 (±1.3)
RDF2vec WL - RF	36.8 (±2.1)	8.2 (±2.1)	13.2 (±1.5)	9.8 (±1.5)
RDF2vec WL - SVM	37.7 (±3.1)	7.3 (±3.1)	14.1 (±1.9)	8.9 (±1.9)
ontowalk2vec BFS - RF	**41.1** (±1.1)	**4** (±1.1)	14 (±1.1)	9 (±1.1)
ontowalk2vec BFS - SVM	**41.2** (±2.1)	**3.8** (±2.1)	12.8 (±2.3)	10.2 (±2.3)
ontowalk2vec WL - RF	**41.2** (±1.7)	**3.8** (±1.7)	14.7 (±1.1)	8.3 (±1.1)
ontowalk2vec WL - SVM	**43.1** (±1.4)	**1.9** (±1.4)	16.4 (±0.8)	6.6 (±0.8)

Table 3
Models' confusion matrices for the MUTAG ontology.

the cities are separated into three categories (excellent, good and bad) depending on their position in the ranking. As in the MUTAG experiment above, we perform tests with RDF2vec, node2vec and ontowalk2vec and repeat the experiments 10 times.

In the pyRDF2Vec python implementation of RDF2Vec [22], SPARQL queries are used for extracting data from DBpedia where the subject of the RDF triple is the instance for which a feature vector is sought. After some preliminary experiments, it was observed that node2vec is not able to compute meaningful vectors from such data, since the random walks are too short. To benefit from the full capabilities of node2vec, we included in our experimentation not only the DBpedia triples that have the required instances as subject, but also the triples that have these instances as object. Moreover, we also included the DBpedia ontology itself in the training phase. This allows for longer random walks and enables ontowalk2vec to take full advantage of both node2vec and RDF2Vec approaches.

Model	Random Forest	SVM
node2vec	0.46 (±0.06)	0.55 (±0.01)
RDF2vec BFS	0.55 (±0.03)	0.57 (±0.03)
RDF2vec WL	0.56 (±0.06)	0.55 (±0.04)
ontowalk2vec BFS	**0.62** (±0.05)	**0.62**(±0.05)
ontowalk2vec WL	**0.63** (±0.03)	**0.6** (±0.04)

Table 4
Accuracy scores using DBpedia.

The accuracy scores provided in Table 4 for both ontowalk2vec BFS (RF - 0.62, SVM - 0.62) and ontowalk2vec WL (RF - 0.63, SVM - 0.6) and the confusion matrix in Table 5 suggest that ontowalk2vec

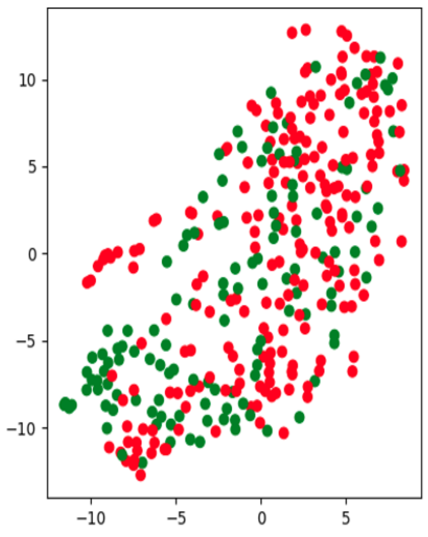

Fig. 7. node2Vec t-SNE.

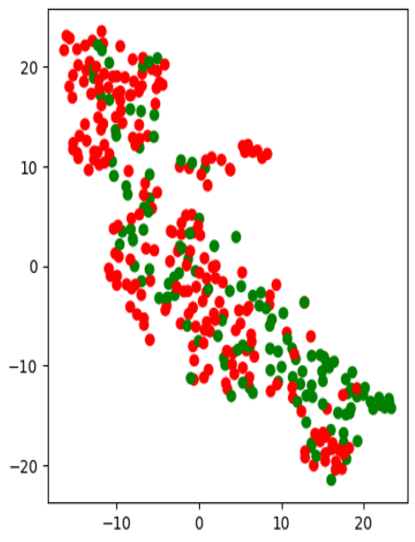

Fig. 8. RDF2Vec t-SNE.

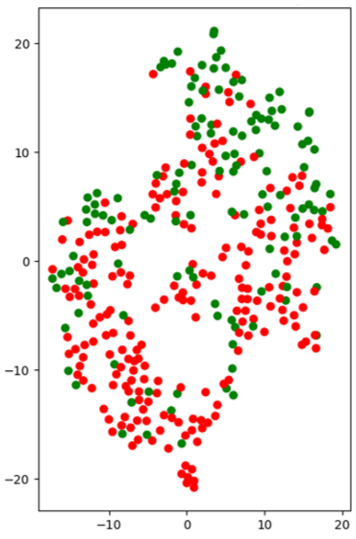

Fig. 9. ontowalk2vec t-SNE.

Actual	Excellent	Good	Bad	Excellent	Good	Bad	Excellent	Good	Bad
Predicted	Excellent	Excellent	Excellent	Good	Good	Good	Bad	Bad	Bad
node2vec									
RF	0.8 (±1.4)	3.9 (±1.5)	1.3 (±1.2)	0.9 (±1.1)	14.8 (±2.2)	6.3 (±2.4)	0.3 (±0.4)	8.9 (±1.9)	2.8 (±1.9)
SVM	0 (±0)	5.7 (±0.9)	0.3 (±0.9)	0 (±0)	21.7 (±0.9)	0.3 (±0.9)	0 (±0)	11.6 (±1.2)	0.4 (±1.2)
RDF2Vec BFS									
RF	2 (±1)	3.3 (±0.8)	0.7 (±0.4)	2.3 (±0.8)	13.7 (±2.9)	6 (±2.6)	0.1 (±0.3)	5.3 (±2)	6.6 (±1.8)
SVM	0.5 (±0.9)	5.5 (±0.9)	0 (±0)	0.3 (±0.4)	20.5 (±2.1)	1.2 (±1.9)	0.1 (±0.3)	9.4 (±3.3)	2.5 (±3.2)
RDF2Vec WL									
RF	0.4 (±0.6)	4.8 (±0.4)	0.8 (±0.4)	0.9 (±0.7)	16.6 (±2.2)	4.5 (±2.2)	0 (±2.2)	6.3 (±2.1)	5.7 (±2.1)
SVM	0.4 (±0.6)	5.6 (±0.6)	0 (±0)	1.1 (±1.5)	17.6 (±2.2)	3.3 (±1.5)	0 (±0)	7.2 (±3.3)	4.8 (±3.3)
ontowalk2vec BFS									
RF	3.1 (±1)	1.8 (±1)	1(±0)	2.4 (±1.8)	12.9(±1.9)	6.7(±0.7)	0(±0)	3(±1.9)	8.8(±1.9)
SVM	0(±0)	5.7(±0.4)	0.3(±0.4)	0(±0)	15 (±2.5)	7(±2.5)	0(±0)	2.4(±2.9)	9.6(±2.9)
ontowalk2vec WL									
RF	0.7 (±0.6)	4.3 (±0.6)	1 (±0)	1.6 (±0.8)	14.8 (±1.0)	5.6 (±1.5)	0.1 (±0.3)	2.5 (±0.7)	9.4 (±0.5)
SVM	1.3 (±1.2)	4.2(±1.6)	0.5(±0.7)	2.8(±2.25)	15.3(±3.4)	3.9(±1.7)	0.8(±1)	3.7(±3.8)	7.5(±3.3)

Table 5
Models' confusion matrices for DBpedia.

is again a more accurate model than node2vec and RDF2Vec, this time using DBpedia. These results also confirm our result of 5% for the quantitative improvement brought by ontowalk2vec over state-of-the-art techniques. A visual proof is also provided in Figure 12 where we can observe a better distinction among the three classes (one colour per class) with ontowalk2vec in comparison to node2vec in Figure 10 and RDF2Vec in Figure 11.

5.3. *OWAO*

OWAO is the basis of the GenAppi framework, used for generating, from any domain ontology, a full-fledged web application to manage data relevant to the domain covered by the ontology [5]. We briefly describe OWAO and its data in the following paragraphs, before using it to further evaluate ontowalk2vec.

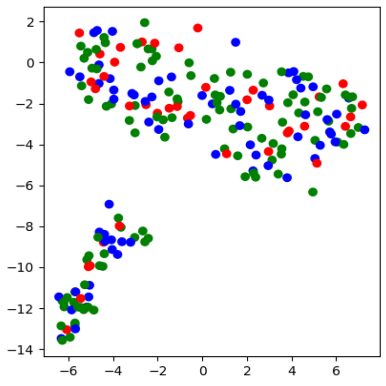

Fig. 10. node2Vec t-SNE for DBpedia cities classification.

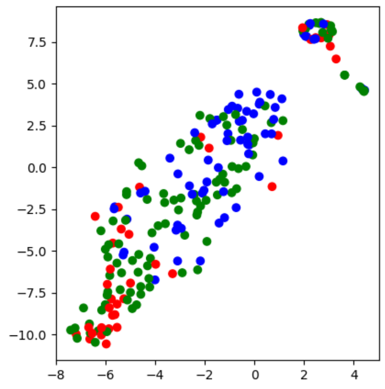

Fig. 11. RDF2Vec t-SNE for DBpedia cities classification.

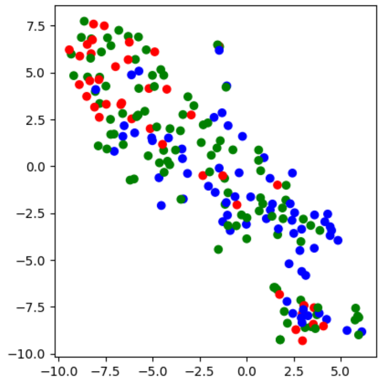

Fig. 12. ontowalk2vec t-SNE for DBpedia cities classification.

5.3.1. *Definition*

OWAO provides means to formally describe the relationships between web application entities and their corresponding UI fragments, to be displayed by GenAppi-generated front-end clients. To make the automatically generated UI more customisable and appealing to final users, concepts and relations representing some key UI preference concepts are introduced in the OWAO ontology.

For example, Figure 13 focuses on the `owao:Form` UI fragment (we omit the `owao` prefix in this section). A specific instance of `Form`, generated by GenAppi, will have a specific `UIStyle` element, composed of instances of the `UIStyleComponent` class. Some subclasses of `UIStyleComponent` are `Colour` and `FontSize`. A `UIStyle` adopted by a user is seen as a user preference, of the class `UIPreference`, that corresponds to a specific UI fragment, such as the `Form` in this case. Each `User` is linked to a set of elements of `UIPreference`.

In the GenAppi-based approach to UI personalisation, for any domain ontology structure and instances, the corresponding OWAO-related instances will form an input data set from which recommendations will be provided to users (see Figure 14). In particular, note that from all the UI choices made by the users, different statistical values can be gathered to help the recommendation process; they are instances of the `UIPreferenceStatistic` class.

5.3.2. *Data*

Since OWAO provides no class instances, we use automatic synthesis to populate it, using current research findings related to UI preferences [26] to create artificially relevant data. More specifically, we focus on four UI components: background colour, font colour, font size and text alignment. Since GenAppi interfaces are built using the Semantic UI framework [27], the discrete values of the previously mentioned UI style components are those defined via the Semantic UI classes. For instance, for the font size, the discrete defined values can be: tiny, small, medium, large, etc. In order to build instances of the class `UIStyle`, all the combinations of font size, text alignment, background and font colour are generated. However, only the combinations that, according to state-of-the-art UI research, are generally more appealing to users are, logically, assumed to be preferred by them.

For example, Figure 16 shows a simplified excerpt of OWAO instances related to UI preferences. The fig-

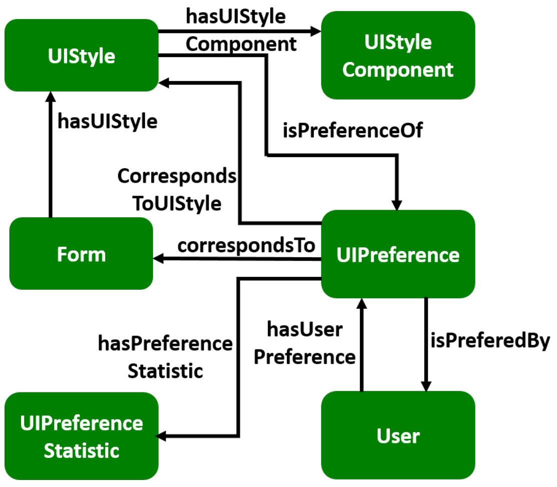

Fig. 13. OWAO excerpt for describing UI user preferences.

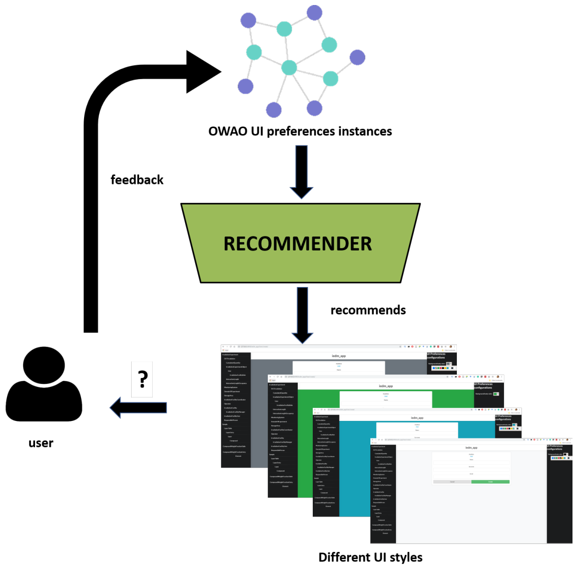

Fig. 14. Recommender model. The user is shown different UI styles for a specific UI and chooses his/her preference, which is provided as a feedback to OWAO.

	Random Forest	SVM
BFS	1.00 (±0.00)	1.00 (±0.00)
WL	1.00 (±0.00)	0.99 (±0.01)

Table 6
Accuracy for OWAO preferences.

ure shows the instance `User1` of the `User` class, associated to an instance of `UIPreference` named `white_black_center_Form1_pref_1` corresponding to an instance of `Form` class named `Form1`. This preference corresponds to the `UIstyle` instance `ui_style_white_ black_mini_center`.

5.3.3. *Evaluation*

Following the same experimental setup as above, a simple binary classification problem is addressed. Instead of the data property `mutag:isMutagenic`, in OWAO, a boolean data property `preferred` is introduced and associated to a `UIStyle` in order to denote if the UI style composed from specific UI components is preferred by users or not. This means that if at least one user prefers the specific `UIStyle` instance, the `preferred` data property is true, otherwise false. The ontowalk2vec model should predict, by exploiting the OWAO ontology data, whether a specific UI style is preferred by the users.

Actual Value	Negative	Negative	Positive	Positive
Predicted Value	Negative	Positive	Negative	Positive
BFS - RF	343 (±0)	0 (±0)	0 (±0)	57 (±0)
BFS - SVM	343 (±0)	0 (±0)	0 (±0)	57 (±0)
WL - RF	343 (±0)	0 (±0)	0 (±0)	57 (±0)
WL - SVM	343 (±0)	0 (±0)	2 (±2)	55 (±2)

Table 7
Confusion matrix for OWAO preferences.

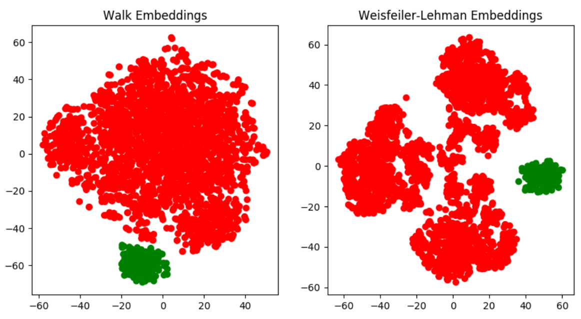

Fig. 15. t-SNE visualisation for OWAO preferences.

Table 6 shows the accuracy score based on the binary classification using as label the data property `preferred` in order to evaluate the ontowalk2vec model on whether a UI style is preferred by any user or not. As in the MUTAG experiment, also in this one, the algorithm is run 10 times, and values presented are the mean values while the errors shown are the standard deviation. According to the table, for both methods BFS and Weisfeiler-Lehman (WL) and with both classifiers, Random Forest and SVM, the accuracy value is 1 or approaching 1. This means that the model is almost always accurate on the predictions, except for the case of the WL method for the SVM classification where the value is 0.99, being still quite high. Given such high accuracy provided by ontowalk2vec, performance data for node2vec and RDF2Vec on OWAO are omitted here.

An analysis of the predictions presented by Table 7, depicting the confusion matrix of the model, shows that the false positives and false negatives are, as expected, in most of the cases equal to 0. Moreover, the t-SNE plots show a clear classification of the two categories, preferred and not preferred. The points of preferred values (in green) are well separated from the not-preferred ones (in red), and no mixture of the two categories is observed. Thus, according to the metrics previously defined, the ontowalk2vec model, when applied to both actual (MUTAG, DBpedia) and synthetic (OWAO) data, provides good classification results.

6. Hyperparameter Settings for UI Personalisation

In the previous section, data suggested that ontowalk2vec provides, on average, better classification results and thus better ontology embeddings than current models. Yet, one key metric that remains to be assessed is the sensitivity of ontowalk2vec to hyperparameters' selection (all models studied in Section 5 were run using the default hyperparameter settings). Since, as all other models, ontowalk2vec depends on many adjustable variables, most of which are imported from the models upon which ontowalk2vec is built, the proper choice of these settings and its impact are addressed in this section, where we focus only on OWAO, our ontology of interest for UI personalisation.

6.1. *Selection*

The word2vec model integrated in ontowalk2vec for generating the embedded vectors, but also RDF2Vec and node2vec used for generating random walks, contain several hyperparameters that need to be tuned before efficiently training these models. These hyperparameters depend on the type and volume of data and can provide higher accuracy results once they are properly initialised [13, 14].

Not all the hyperparameters are useful for getting better results. For example a hyperparameter of word2vec is *workers*, which denotes the number of computer threads needed to train the model, obviously aiming for faster training with multi-core machines. Another hyperparameter defines whether the model will use the Skip-gram or the Continuous Bag of Words (CBOW) technique, but as already explained in Section 4, for ontology problems such as ours, it is more suitable to use Skip-gram, since with the CBOW, the order of the entities appearing in a walk would be lost.

In the literature, the most common hyperparameters that are targeted for optimisation are: negative sampling, iterations and window size (described below) [13]. In this work, a larger set of hyperparameters is taken into account in order to seek even better performance. The fine tuning of these hyperparameters is presented below, and illustrated by the use of these optimisation techniques for finding the best combination of hyperparameters for OWAO. More specifically, the hyperparameters that are being optimised here are the following.

- **learning rate (alpha)**. This hyperparameter controls the amount of change of the weights of the model in each iteration needed for reaching convergence. In word2vec, this hyperparameter denotes the initial value for the training phase, and after each iteration this value changes to a smaller one until reaching the minimum value 0.0001.
- **iterations**. This is the number of iterations (epochs) of training over the corpus of data.
- **vector size**. This denotes the dimensionality of the vector space used to represent the data.
- **window size**. This is the maximum distance allowed between the current and predicted words within a sentence.
- **minimum counts**. The words with a total frequency lower than `min_count` are ignored by the model.
- **hierarchical softmax**. Softmax is a normalised exponential function used as the last layer of a neural network to normalise the output. Hierarchical softmax is an optimised softmax function representing the words as the leaves of a binary tree. The hierarchical softmax hyperparameter is a boolean parameter specifying whether hierarchical softmax is used or not.
- **negative**. An alternative to hierarchical softmax is the negative sampling approach. The idea behind negative sampling is to separate the words that do not contribute in the output (noise) and the useful data. This word2vec hyperparameter indicates whether negative sampling is used or not. If the value is more than 0, negative sampling will be used, while the specified integer is the number of noisy words that should be drawn. If it is set to 0, no negative sampling is used.
- **depth**. This is the maximum level to be reached when traversing the ontology-derived graph.

6.2. *Evaluation*

In order to evaluate how the values assigned to the hyperparameters affect ontowalk2vec and which ones should be used, an exhaustive grid search, detailed in [26], was performed using hyperparameter combinations uniformly sampled from the range of hyperparameter values shown in Table 8. While it is difficult to conclude in a single rule taking into account all the hyperparameters, by running the on-

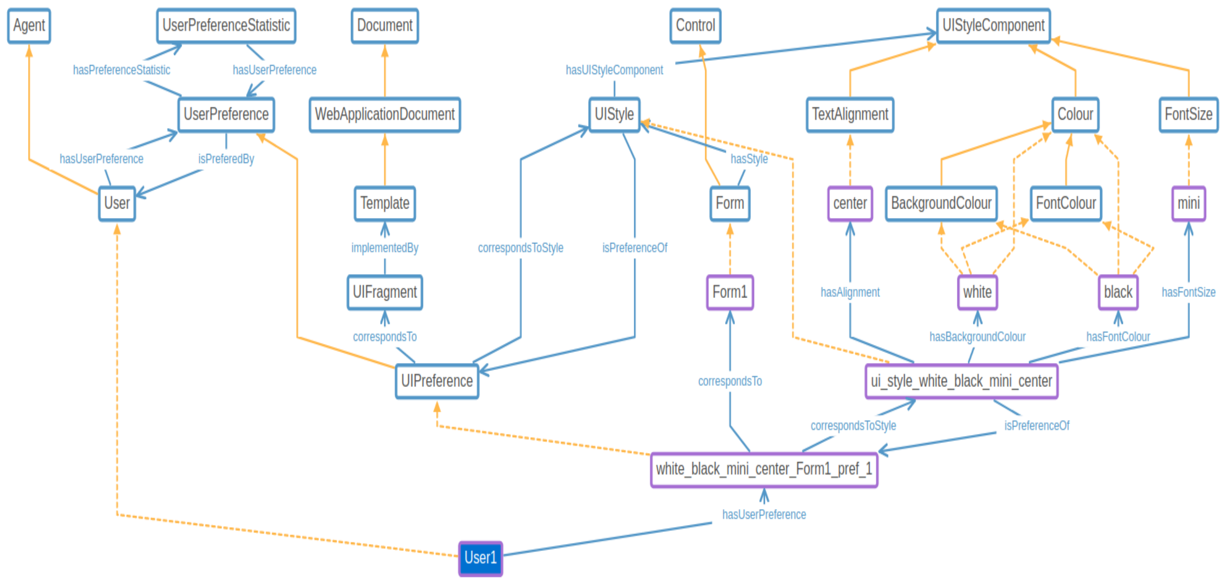

Fig. 16. Excerpt from OWAO depicting a User instance (`User1`) and his/her preferences for a given UI component.

Hyperparameter	Range	Default Value
learning rate (lr)	[0.1, 0.05,0.025,0.01]	0.025
iterations (iter)	[1,5,20,50]	5
vector size (vs)	[20,100, 200, 500]	100
window (win)	[3,5,7]	5
min_count (mc)	[0,1]	5
negative (neg)	[0,1,5,10]	5
hierarchical softmax (hs)	[0,1]	0
depth (dep)	[1,2,3]	1

Table 8
Hyperparameter and their range, used to form combinations.

towalk2vec experiments with each of the combinations of Table 8 and calculating the F1 score, a specific combination of hyperparameters appeared to provide the best F1 score for the recommendation results for both ontowalk2vec BFS and WL. This combination is given by the following hyperparameter setting: `window size = 7`, `vector size = 500`, `learning rate = 0.0025`, `iteration = 1`, `min count = 1`, `depth = 2`, `negative = 10` and thus `hierarchical softmax = 0`. The standard deviation of the F1 score recommendation results over the all grid is 0.27, which shows that about 1/4 of the performance of an embedding-based recommender system built on top of ontowalk2vec can be attributed to proper hyperparameter setting.

Using these best hyperparameters on a particular synthetic UI-preferences database (see [26] for a precise description of how this data set was obtained), Table 9 illustrates what would be the top recommendations (i.e., the top similarity-based predictions) for the UI style instance `ui_style_light_grey_black_medium_center`, when using the ontowalk2vec BFS method. One can easily check that the instances recommended by ontowalk2vec, i.e., that have high similarity values with this particular instance, do indeed have common UI style elements such as the same font size, similar background colour or font colour and text alignment.

7. Conclusion

In this paper, we introduced the ontowalk2vec model for computing ontology embeddings and experimentally assessed its capabilities for classification purposes, in particular for UI preference recommendation. Several classification methods and evaluation metrics have been, first discussed, and then used for evaluating its performance on the MUTAG, DBpedia and OWAO ontologies. Our preliminary results suggest that, overall, ontowalk2vec performs on average 5% better than state-of-the-art techniques, although the selection of good hyperparameters has a significant

Instances	Cosine Similarity
owao:ui_style_white_black_medium_center	0.956
owao:ui_style_light_grey_black_medium_left	0.955
owao:ui_style_light_grey_black_large_left	0.941
owao:ui_style_white_black_large_left	0.931
owao:ui_style_light_grey_black_medium_center	0.924

Table 9

Top cosine similarities for `ui_style_white_black_medium_left` using BFS.

impact on its performance. We assessed this last issue by performing a grid search for the optimal values using the OWAO use case.

Using a high-accuracy vector embedding model such as ontowalk2vec should allow for building a solid recommender system that can suggest to users UI styles similar to their current preferences or to those of users who have similar UI preferences or features. More generally, this means that our approach can be used for both content- and collaborative-filtering-based recommender systems, if there is enough data. Finally, though more software development work would be needed, these embeddings could be directly integrated in GenAppi-generated web applications, thus improving automatically user experience when using such systems.

The source code of ontowalk2vec containing all the ontologies used and techniques for hyperparameter optimisation can be found in the accompanying online resources[5].

8. Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 654168.

References

[1] Semantic User Interface ontology, Available: https://old.datahub.io/dataset/ui.

[2] Linked Open Vocabularies (LOV) User Interface ontology, Available: https://lov.linkeddata.es/dataset/lov/vocabs/ui.

[3] A. Haller, J. Umbrich and M. Hausenblas, RaUL: RDFa User Interface Language – A Data Processing Model for Web Applications, in: *Web Information Systems Engineering – WISE 2010*, L. Chen, P. Triantafillou and T. Suel, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 400–410. ISBN 978-3-642-17616-6.

[4] S.K. Shahzad, Ontology-based User Interface Development: User Experience Elements Pattern, *J. UCS* **17** (2011), 1078–1088.

[5] B. Gkotse, P. Jouvelot and F. Ravotti, Automatic Web Application Generation from an Irradiation Experiment Data Management Ontology (IEDM), in: *17th International Conference on Accelerator and Large Experimental Physics Control Systems, ICALEPCS 2019*, JACoW Publishing, 2019, pp. 687–693, Available: http://icalepcs2019.vrws.de/papers/tubpl01.pdf. ISSN 2226-0358. ISBN 978-3-95450-209-7. doi:doi:10.18429/JACoW-ICALEPCS2019-TUBPL01.

[6] A. Darejeh and D. Singh, A Review on User Interface Design Principles to Increase Software Usability for Users with Less Computer Literacy, *JCS* **9** (2013), 1443–1450.

[7] S.L.T. Hui and S.L. See, Enhancing User Experience Through Customisation of UI Design, *Procedia Manufacturing* **3** (2015), 1932–1937, 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015. doi:https://doi.org/10.1016/j.promfg.2015.07.237. http://www.sciencedirect.com/science/article/pii/S2351978915002383.

[8] J. Bobadilla, F. Ortega, A. Hernando and A. Gutiérrez, Recommender systems survey, *Knowledge-Based Systems* **46** (2013), 109–132. doi:https://doi.org/10.1016/j.knosys.2013.03.012. http://www.sciencedirect.com/science/article/pii/S0950705113001044.

[9] T. Mikolov, K. Chen, G.S. Corrado and J. Dean, Efficient Estimation of Word Representations in Vector Space, *CoRR* **abs/1301.3781** (2013).

[10] P. Ristoski, J. Rosati, T.D. Noia, R.D. Leone and H. Paulheim, RDF2Vec: RDF graph embeddings and their applications, *Semantic Web* **10**(4) (2019), 721–752. doi:10.3233/SW-180317.

[11] A. Grover and J. Leskovec, node2vec: Scalable Feature Learning for Networks, *KDD : proceedings. International Conference on Knowledge Discovery & Data Mining* (2016), 855–864.

[12] G.K.D. de Vries, A Fast Approximation of the Weisfeiler-Lehman Graph Kernel for RDF Data (2013), 606–621. ISBN 978-3-642-40988-2.

[13] H. Caselles-Dupré, F. Lesaint and J. Royo-Letelier, Word2vec applied to recommendation: hyperparameters matter, *Proceedings of the 12th ACM Conference on Recommender Systems* (2018).

[14] J. Dillenberger, Evaluation of Model and Hyperparameter Choices in word2vec, 2019, Bachelor's Thesis, Koblenz University, Landau, Germany. Available: https://west.uni-koblenz.de/assets/theses/evaluation-model-hyperparameter-choices-word2vec.pdf.

[5] https://gitlab.cern.ch/irrad/ontowalk2vec

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and É. Duchesnay, Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research* **12**(85) (2011), 2825–2830, Available: http://jmlr.org/papers/v12/pedregosa11a.html.

[16] Hyperplane, From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/Hyperplane.html.

[17] T. Wu, C. Lin and R.C. Weng, Probability estimates for multi-class classification by pairwise coupling, *Journal of Machine Learning Research* **5**(Aug) (2004), 975–1005.

[18] L. Breiman, Random forests, *Machine learning* **45**(1) (2001), 5–32.

[19] L.v.d. Maaten and G. Hinton, Visualizing data using t-SNE, *Journal of machine learning research* **9**(Nov) (2008), 2579–2605.

[20] S. Kullback, Letters to the Editor: The Kullback–Leibler distance, *The American Statistician* **41**(4) (1987), 338–341, Available: https://doi.org/10.1080/00031305.1987.10475510. doi:10.1080/00031305.1987.10475510.

[21] G. Shani and A. Gunawardana, Evaluating Recommendation Systems, in: *Recommender Systems Handbook*, Springer US, Boston, MA, 2011, pp. 257–297.

[22] G. Vandewiele, B. Steenwinckel, T. Agozzino, M. Weyns, P. Bonte, F. Ongenae and F. De Turck, pyRDF2Vec: A python library for RDF2Vec, IDLab, 2020. https://github.com/IBCNServices/pyRDF2Vec.

[23] DL-Learner Homepage, Available: http://dl-learner.org/.

[24] A. Harms and S. Spinder, A comprehensive view of machine learning techniques for CPI production, *Statistics Netherlands* (2019), Available: https://www.cbs.nl/en-gb/background/2019/47/machine-learning-techniques-for-cpi-production. doi:10.1088/1748-0221/3/08/s08003.

[25] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer and C. Bizer, DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia, *Semantic Web* **6** (2015), 167–195.

[26] B. Gkotse, Ontology-based Generation of Personalised Data Management Systems: an Application to Experimental Particle Physics. Génération de systèmes de gestion de données personnalisés fondée sur les ontologies : une application à la physique expérimentale des particules, PhD thesis, Mines Paris, PSL University, 2020, Presented 25 Sep 2020. https://cds.cern.ch/record/2743492.

[27] Semantic UI framework, Available: https://semantic-ui.com.