How does a text preprocessing pipeline affect ontology matching?

Anonymous ACL submission

Abstract

The generic text preprocessing pipeline, comprising Tokenisation, Normalisation, Stop Words Removal, and Stemming/Lemmatisation, has been implemented in many ontology matching (OM) systems. However, the lack of standardisation in text preprocessing creates diversity in mapping results. In this paper, we investigate the effect of the text preprocessing pipeline on OM tasks at syntactic levels. Our experiments on 8 Ontology Alignment Evaluation Initiative (OAEI) track repositories with 49 distinct alignments indicate: (1) Tokenisation and Normalisation are currently more effective than Stop Words Removal and Stemming/Lemmatisation; and (2) The selection of Lemmatisation and Stemming is task-specific. We recommend standalone Lemmatisation or Stemming with post-hoc corrections. We find that (3) Porter Stemmer and Snowball Stemmer perform better than Lancaster Stemmer; and that (4) Part-of-Speech (POS) Tagging does not help Lemmatisation. To repair less effective Stop Words Removal and Stemming/Lemmatisation used in OM tasks, we propose a novel context-based pipeline repair approach that significantly improves matching correctness and overall matching performance.

1 Introduction

Ontology matching (OM), also known as ontology alignment, is essential to enable interoperability between heterogeneous ontologies. An OM process usually takes two ontologies as input, discovers mappings between entities, and produces a set of correspondences (Euzenat et al., 2007). A classical OM system usually contains syntactic, lexical, and semantic matching. Syntactic matching captures "anchor mappings", providing the foundation for the latter lexical and semantic matching. This multi-layer architecture has been implemented in several successful OM systems, such as LogMap (Jiménez-Ruiz and Cuenca Grau, 2011; Jiménez-Ruiz et al., 2011), AgreementMakerLight (AML) (Faria et al., 2013, 2014), and FCA-Map (Zhao et al., 2018; Li et al., 2021).

There are many strategies to extract syntactic information from an ontology entity, including the older approach of Bag-of-Words (e.g. TF-IDF Sammut and Webb, 2010), popular word embedding models (e.g. Word2Vec Mikolov et al., 2013), and state-of-the-art language models (e.g. BERT Devlin et al., 2019). Despite the diversity of the models used, they all apply text preprocessing for cleaning the text data before fitting it into the model. Figure 1 shows an example of using the text preprocessing pipeline to process the ontology entity "cmt:reviewerBiddingStartedBy". The text preprocessing pipeline consists of a set of steps to segment, reconstruct, analyse, and process the information in the text, namely Tokenisation, Normalisation, Stop Words Removal, and Stemming/Lemmatisation (Anandarajan et al., 2019). Tokenisation is the process of breaking the text into the smallest units (i.e. tokens). We use whitespace to split the tokens in the example. Normalisation is the process of transforming these different tokens into a single canonical form. Stop Words Removal is the process of removing filler words that usually carry little meaning and can be omitted in most cases. Stemming/Lemmatisation is used to deal with the grammatical variation of words, applying rules to find the simplest common form of the word. This helps to capture the key information from the text and therefore improves efficiency.

While a number of OM systems use the text preprocessing pipeline for syntactic OM, few studies explain why a specific method is selected for a certain OM task. Our study tackles the challenge in two ways. Firstly, we conduct a comprehensive experimental analysis of the text preprocessing pipeline in syntactic OM across a wide range of domains, aiming to explain the behaviour of the text preprocessing pipeline in OM tasks at syn-

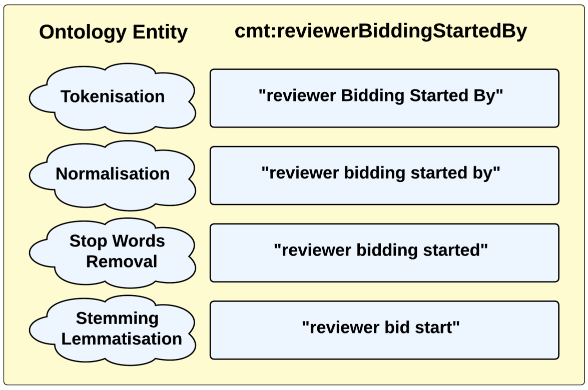


Figure 1: An example of the text preprocessing pipeline.

tactic levels. In each phase, a text preprocessing method is evaluated for its correctness and completeness. Secondly, we propose a novel context-based pipeline repair approach for syntactic OM. The method offers a customised way to fine-tune the text preprocessing pipeline for each domain-specific OM task and shows promising results for repairing false mappings. Specifically, this paper makes the following contributions:

- We categorise the text preprocessing pipeline used in syntactic OM into two phases. We find a significant improvement using Phase 1 text preprocessing methods. In contrast, Phase 2 text preprocessing methods are currently less effective. We compare the performance of (1) Stemming and Lemmatisation, (2) different stemming methods (Porter, Snowball, and Lancaster), and (3) Lemmatisation and Lemmatisation + Part-of-Speech (POS) Tagging. We find that inappropriate stop words removal, over-stemming, and over-lemmatisation are common on 8 Ontology Alignment Evaluation Initiative (OAEI) (OAEI, 2023) track repositories with 49 distinct alignments.
- We propose a simple and intuitive context-based pipeline repair method. This method is evaluated on the same OM tasks we analysed, showing promising results to improve the correctness of syntactic OM when inserted in the pipeline repair before Phase 2 text preprocessing methods.
- We provide our code and generated alignments from the experiment (submitted as a single .zip archive). They can be reused to benchmark new text preprocessing methods or fine-tune existing text preprocessing models used in OM systems.

The remainder of the paper is organised as follows. Section 2 reviews the related work. Section 3 analyses the text preprocessing pipeline used in OM. Section 4 proposes the context-based pipeline repair approach and experimentally validates its performance. Section 5 concludes the paper.

2 Related Work

Syntactic matching considers only the meaning of the entity's name or label, ignoring its lexical and structural context in an ontology (Liu et al., 2021). Correct syntactic matches are often implicit and usually require extra human observation and domain knowledge. Text preprocessing is introduced to automate this process.

The use of text preprocessing in syntactic matching can be traced back to the early stages of OM systems, having been initially developed and widely used as a basic component to generate linguistic-based mappings. AgreementMaker (Cruz et al., 2009) has a normaliser to unify the textual information of the entities. SAMBO (Lambrix and Tan, 2006) uses the Porter Stemmer for each word to improve the similarity measure for terms with different prefixes and suffixes. In RiMOM (Li et al., 2009), the context information of each entity is viewed as a document. The text in each document is preprocessed with tokenisation, stop words removal, and stemming.

Recently, machine learning (ML) models have emerged for modern OM systems. While text preprocessing remains useful, its role is more focused on normalising the text that becomes the input to the model. For example, BERTMap (He et al., 2022) uses BERT's inherent WordPiece tokeniser (Wu et al., 2016) to build the subword of each entity. A more recent approach DeepOnto (He et al., 2023) extends the normalisation to axioms using EL embedding models (Kulmanov et al., 2019). The ML extension of LogMap (Chen et al., 2021) reuses the seed mappings of the traditional system, where each entity is split into its component English word and the mapping is based on their similarity.

However, to the best of our knowledge, most of the literature implements a preprocessing method without explaining why a specific method is chosen, and no studies have been conducted to evaluate the effect of text preprocessing on syntactic OM.

3 Analysis of Text Preprocessing Pipeline

3.1 Method

Given a source ontology O_s and a target ontology O_t, OM establishes mappings between pairs of entities drawn from each of two ontologies. A correspondence (i.e. one instance of mappings) is defined as a 4-tuple (e_1, e_2, r, c), where $e_1 \in O_s$ and $e_2 \in O_t$. r is the relationship between two matched entities e_1 and e_2, and $c \in [0, 1]$ is the confidence for each correspondence. The relationship r in OM tasks can be equivalence ($\equiv$), subsumption ($\subseteq$), disjointness ($\perp$), or other more complex relationships. In this paper, we focus only on the equivalence relationship ($\equiv$) and evaluate the effect of text preprocessing on syntactic matching to produce the "anchor mappings" on which to base any subsequent lexical and semantic matching. We address only *equivalence* because subsumption and disjointness are typically dealt with in a later semantic and structural matching phase of OM. An alignment (A) is a set of candidate correspondences generated by tools, while a Reference (R) is a set of gold standard correspondences verified by domain experts (i.e. the ground truth alignment).

Figure 2 shows the method used to analyse the text preprocessing pipeline. Alignment (A) is generated via the text preprocessing pipeline. Firstly, for both O_s and O_t, we retrieve the entities from the named classes (i.e. owl:Class) and named properties (i.e. object properties owl:ObjectProperty and data type properties owl:DatatypeProperty). For those ontologies where the names of the concepts are not textual (e.g. a numerical identifier), instead, we retrieve the meaningful text from entity labels (i.e. rdfs:label). Then, we apply the text preprocessing pipeline method $f(.)$ on each entity $e_1 \in O_s$ and $e_2 \in O_t$. If $f(e_1) = f(e_2)$, we store the correspondence in the corresponding alignment file. Finally, we compare the generated Alignment (A) with Reference (R) to evaluate the performance of the text preprocessing pipeline on OM tasks.

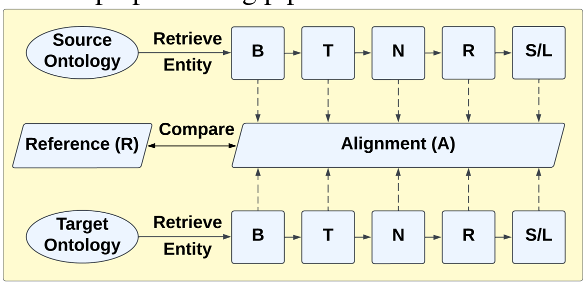


Figure 2: Method used to analyse the text preprocessing pipeline. B-Base Entity without Text Preprocessing, T-Tokenisation, N-Normalisation, R-Stop Words Removal, S/L-Stemming/Lemmatisation.

3.1.1 Selected OAEI Track Repositories

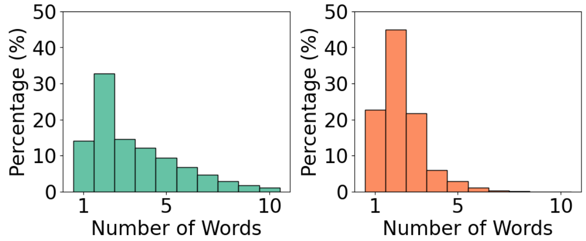
The Ontology Matching Evaluation Toolkit (MELT) (Hertling et al., 2019) is a powerful framework for OM evaluation. We retrieve 17 OAEI normal track repositories stored in the MELT public repository (date accessed: 2024-06-01). The repositories are categorised as schema matching or instance matching. 12 of the 17 track repositories are schema-matching and applicable for evaluating OM. Only 8 of these track repositories are selected because the other 4 have noisy data or miss required files. Specifically, the *knowledgegraph* repository contains both schema and instance matching. The *complex* repository is primarily focused on detecting complex correspondences, while the *multifarm* repository is an extension of the *conference* repository to multilingualism. The *laboratory* repository lacks reference files at the time of writing.

Table 1 shows the details of the selected track repositories. We consider only the equivalence mappings contained in the reference files. Each track repository may contain more than one alignment corresponding to different pairs of ontologies. For example, the *largebio* track repository has 6 reference files, pairing the whole and small versions of FMA (Rosse and Mejino, 2003) and NCI (Golbeck et al., 2003), FMA and SNOMED (Donnelly et al., 2006), and SNOMED and NCI, respectively. The number of references for each track repository is given in the table. There are 49 distinct alignments evaluated in this study.

Name	Domain	Number of References
anatomy	Human and Mouse Anatomy	1
biodiv	Biodiversity and Ecology	9
commonkg	Common Knowledge Graphs	3
conference	Conference	24
food	Food Nutritional Composition	1
largebio	Biomedical	6
mse	Materials Science & Engineering	3
phenotype	Disease and Phenotype	2

Table 1: Selected OAEI track repositories.

Compound words are frequently used in ontology naming conventions. For example, compound words "art gallery" can be formatted as the Pascal case "ArtGallery" used in YAGO (Suchanek et al., 2007) or the Snake case "art_gallery" used in Wikidata (Vrandečić and Krötzsch, 2014). Figure 3 shows the proportion of compound words in the selected OAEI track repositories. For comparisons between entities with compound words, one approach is to use their tokens, where "XY" is equivalent to "YX" because they share the same concepts X and Y. However, our comparison considers both concepts and their order, so that "XY" is not equivalent to "YX". Although concurrent "XY" and "YX" are rare in the matching process, we assume that it could happen and so we take account of the order of concepts in compound word names to ensure a fair comparison in our experiments.



(a) Named Classes (b) Named Properties

Figure 3: The proportion of compound words.

3.1.2 Selected Subtasks of Pipeline Methods

There are a variety of subtasks that can be used in a general text preprocessing pipeline, but not all of them are applicable to OM tasks. We select the following subtasks in each text preprocessing method:

(1) Tokenisation: includes word tokenisation only. We do not use sentence tokenisation (also known as segmentation) because text retrieved from the entity's name or label is commonly short.

(2) Normalisation: includes lowercasing, HTML tags removal, separator formatting, and punctuation removal. Other subtasks that may potentially change the word semantics, such as removal of special characters and numbers, are excluded.

(3) Stop Words Removal: includes the most common English stop words defined in the Natural Language Toolkit (NLTK) (Bird et al., 2008).

(4) Stemming/Lemmatisation: Stemming methods include Porter Stemmer, Snowball Stemmer, and Lancaster Stemmer. Lemmatisation uses the NLTK Lemmatiser (Bird et al., 2008) based on WordNet (Miller, 1995), and the word categorisation uses POS Tagging.

3.1.3 Selected OM Evaluation Measures

In information retrieval, there are four primitive measures: true positive (TP), false positive (FP), false negative (FN), and true negative (TN). In the context of OM, evaluation compares an alignment (A) returned by the OM system with a gold standard reference (R). Figure 4 illustrates that the four primitive measures in OM can be interpreted as $TP = A \cap R$, $FP = A - R$, $FN = R - A$, and $TN = (C \times C') - (A \cup R)$, where $C \times C'$ refers to all possible correspondences $\in \{O_s, O_t\}$.

Accuracy (Acc), Specificity (Spec), Precision (Prec), Recall (Rec), and F_β Score (F_β) are the most common evaluation measures based on TP, FP, FN, and TN. In the context of OM, since $C \times C'$ is extremely large (the Cartesian product of $e_1 \in O_s$ and $e_2 \in O_t$), TN is often much larger than TP,

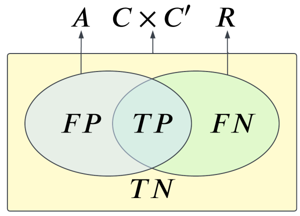


Figure 4: OM evaluation measures (Euzenat, 2007).

FP, and FN. This means that Accuracy (Acc) and Specificity (Spec) are close to 1, and they have no statistically significant difference across different alignments. We note that Precision (Prec) and Recall (Rec) contribute equally to F_β. Therefore, we choose Precision (Prec), Recall (Rec), and F1 Score ($\beta = 1$) in this study. They are defined as:

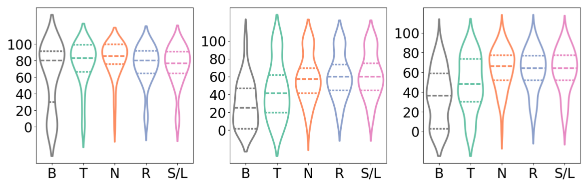
$$Prec = \frac{|A \cap R|}{|A|} \quad Rec = \frac{|A \cap R|}{|R|} \tag{1}$$

$$F_1 = \frac{2}{Prec^{-1} + Rec^{-1}} \tag{2}$$

3.2 Results

3.2.1 Comparison of Pipeline Methods

Figure 5 summarises the comparison of the text preprocessing methods Tokenization (T), Normalisation (N), Stop Words Removal (R), and Stemming/Lemmatisation (S/L). The result indicates that the vast majority of correct correspondences are found by T and N. We do not see R and S/L playing a prime role in the OM tasks. Details can be found in the Appendix B.1.

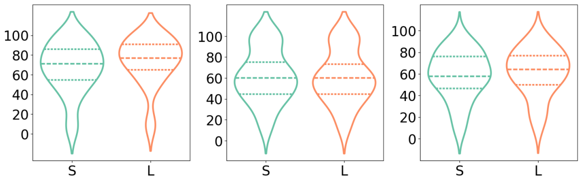


(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 5: Comparison of the generic text preprocessing pipeline: Base Entity without Text Preprocessing (B), Tokenisation (T), Normalisation (N), Stop Words Removal (R), Stemming/Lemmatisation (S/L). The methods are always applied sequentially in the pipeline. Three horizontal lines inside each violin plot show three quartiles: Q1, median, and Q3. The extension of the curves beyond 0% and 100% is an artefact of the seaborn (Waskom, 2021) violin plot. (a) Precision: The median increases with T and N but decreases with R and S/L. After T, the shape of the distribution is unchanged by R and S/L. (b) Recall: After T, the median increases slightly with each of N, R, and S/L. The shape of the distribution does not change after N. (c) F1 Score: the median increases with T and N but then decreases with R and S/L. The shape of the distribution does not change after N.

3.2.2 Stemming vs. Lemmatisation

Figure 6 compares Stemming (S) and Lemmatisation (L) on 49 alignments after Tokenisation (T) and Normalisation (N) have been applied. L is commonly better than S.

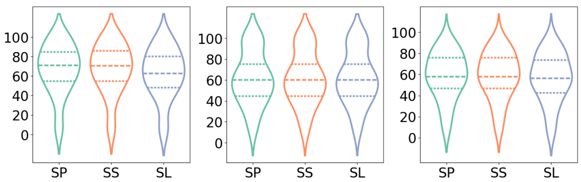


(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 6: Comparison of Stemming (S) and Lemmatisation (L). (a) Precision: The median after L is greater than that achieved by S. The shape of the distribution is slightly different. (b) Recall: The median and the shape of the distribution are identical after S and after L. (c) F1 Score: The median after L is greater than for S. The shape of the distribution is identical.

3.2.3 Porter Stemmer vs. Snowball Stemmer vs. Lancaster Stemmer

Figure 7 compares Porter Stemmer (SP), Snowball Stemmer (SS), and Lancaster Stemmer (SL) in 49 alignments. SP and SS have been found to perform better than SL.

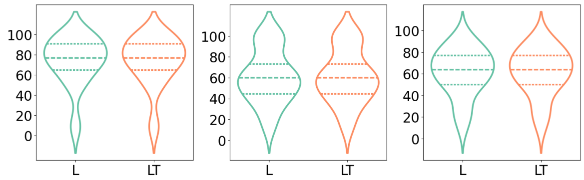


(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 7: Comparison of different stemmers: Porter Stemmer (SP), Snowball Stemmer (SS) and Lancaster Stemmer (SL). (a) Precision: The median number in SP and SS is greater than that in SL, and there is no difference between SP and SS. The shape of the distribution is identical. (b) Recall: The median number and the shape of the distribution are identical in SP, SS, and SL. (c) F1 Score: The median number in SP and SS is greater than that in SL, and there is no difference between SP and SS. The shape of the distribution is identical.

3.2.4 Lemmatisation vs Lemmatisation + POS Tagging

Figure 8 summarises the comparison of Lemmatisation (L) and Lemmatisation + POS Tagging (LT) in 49 alignments. The result indicates that POS Tagging does not help with Lemmatisation in Precision, Recall, and overall F1 Score in the total of 49 alignments we analysed.



(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 8: Comparison of Lemmatisation (L) and Lemmatisation + POS Tagging (LT). For each of (a) Precision, (b) Recall and (c) F1 Score, the median and the shape of the distribution are identical for L and LT.

3.3 Discussion

For syntactic OM, the text preprocessing pipeline can be categorised into two phases. Phase 1 text preprocessing pipeline methods contain Normalisation and Tokenisation, whereas Stop Words Removal and Stemming/Lemmatisation are assigned to Phase 2. Phase 1 text preprocessing pipeline methods do not change word semantics; instead, they only change syntactic features such as formatting and typography. Conversely, in Phase 2 of the text preprocessing pipeline, words are changed to better capture semantic similarity, such as removing prefixes and suffixes or substituting common etymological roots.

3.3.1 Phase 1 Text Preprocessing Methods

We observe that matching performance usually increases with each Phase 1 text preprocessing method, indicating the benefit of these “rules of thumb” for syntactic OM. If two terms can be matched using a heuristic or intuitive technique, there is no need to leverage the more complex Phase 2 methods.

However, Phase 1 methods could also benefit from customisation in OM tasks. Some traditional methods originating for natural language processing (NLP) are not useful for OM and need to be adjusted. For example, sentence segmentation (i.e.splitting long text into sentences) is not applicable for ontology entities because they are generally short text fragments and do not need such operations. Word tokenisation (i.e. breaking text into single words) could be rewritten to detect the abbreviations that are common practice in ontology concept names and to tackle the conflicting use of camel case and snake case.

3.3.2 Phase 2 Text Preprocessing Methods

We observe no benefit from Phase 2 text preprocessing methods. In some cases, Stop Words Removal and Stemming/Lemmatisation may even hamper the mapping. There are plausible explanations for this behaviour arising from the nature of ontologies as distinct from natural language text, as follows:

(1) Stop Words Removal: "AND" and "OR" are stop words in English and usually carry little useful information, whereas these two words express logical operations in ontology entities and therefore may carry important semantics.

(2) Stemming vs. Lemmatisation: Based on our experiments, Lemmatisation is better than Stemming. While lemmatisation tends to avoid generating FPs, it may also miss some implicit TPs. The complexity of finding a missing TP is equal to the size of (C'). On the other hand, stemming is more aggressive in finding TPs, but the aggression can lead to more FPs as well. The complexity of finding FPs is equal to the smaller size of (A). The workload of discovering FPs after stemming is generally much lighter than detecting missing TPs after lemmatisation, but this may also depend on the accuracy of post-hoc corrections that can be performed.

(3) Porter Stemmer vs. Snowball Stemmer vs. Lancaster Stemmer: Porter Stemmer and Snowball Stemmer (also known as Porter 2 Stemmer) have been found to perform better than Lancaster Stemmer, and we cannot see a significant difference between Porter and Snowball. Although the Lancaster Stemmer is more aggressive in detecting more TPs, it does not lead to performance improvement as more FPs are matched synchronously.

(4) Lemmatisation vs. Lemmatisation + POS Tagging: Lemmatisation + POS Tagging is generally expected to have better results than Lemmatisation alone because tagging can help detect more precise root words. However, we do not observe such a performance improvement when using POS Tagging in our study. The reason may be that ontology classes are usually nouns or gerunds, and in such cases, we could expect the simpler grammatical assumption to have similar results.

4 Context-based Pipeline Repair

4.1 Motivation

Experimental results demonstrate that only Tokenisation and Normalisation help with syntactic OM. The use of Stop Words Removal and Stemming/Lemmatisation does not improve performance and may even have negative impacts.

Phase 1 text preprocessing methods (Tokenisation and Normalisation) do not change the text meaning. For example, *isReviewing* is equivalent to *is_reviewing*. This means that applying Phase 1 methods only helps detect TPs, while the number of FPs remains unchanged. For this reason, Phase 1 methods always have a positive effect on Precision, Recall, and overall F1 Score.

$$Prec \uparrow = \frac{|A \cap R|}{|A|} = \frac{TP}{TP + FP} = 1 - \frac{FP}{TP \uparrow + FP} \quad (3)$$

$$Rec \uparrow = \frac{|A \cap R|}{|R|} = \frac{TP \uparrow}{|R|} \quad (4)$$

$$F_1 \uparrow = \frac{2}{Prec \uparrow^{-1} + Rec \uparrow^{-1}} \quad (5)$$

In most cases, OM only requires minor preprocessing using Phase 1 text preprocessing methods. This is because entity names in the ontology are often compound words that do not occur in natural language, but they are partially formalised by agreement. There have been well-defined conventions established over the years, e.g. singularity, positive names, nouns for classes and verbs for properties (Schober et al., 2007; Taylor et al., 2015).

Phase 2 text preprocessing methods (Stop Words Removal and Stemming/Lemmatisation) are actually relaxations of matching rules in OM tasks. Moving through the text preprocessing pipeline tends to detect more TPs and FPs in the derived alignment (A). For example, *isReviewing* and *isReviewedBy* may be object properties with distinctly different meaning, but removing the common stop words "is" and "by", and using Stemming/Lemmatisation to retrieve the same root word "Review", could cause a false match. For this reason, Recall is always increasing, but Precision and overall F1 Score are less reliable.

$$Prec\ ? = \frac{|A \cap R|}{|A|} = \frac{TP \uparrow}{TP \uparrow + FP \uparrow} \quad (6)$$

$$Rec \uparrow = \frac{|A \cap R|}{|R|} = \frac{TP \uparrow}{|R|} \quad (7)$$

$$F_1\ ? = \frac{2}{Prec\ ?^{-1} + Rec \uparrow^{-1}} \quad (8)$$

If we define ΔTP and ΔFP as the increase in TP and FP for a preprocessing method, then the threshold to increase Precision and F1 Score is:

$$\frac{TP + \Delta TP}{TP + \Delta TP + FP + \Delta FP} > \frac{TP}{TP + FP} \Rightarrow \frac{\Delta TP}{\Delta FP} > \frac{TP}{FP} \quad (9)$$

In our experiments, we actually observe a reduction in Precision and overall F1 Score. This means that Phase 2 methods produce more FPs than TPs, and this proportion is less than the original number of TP/FP. So performance does not improve, unless the benefit of each TP is considered more valuable than the disbenefit of each FP. This could apply, for example, if we are expecting a post-hoc correction phase where removing FPs is considered to be an easier human task than adding missing TPs.

4.2 Method

Phase 2 text preprocessing pipeline methods (Stop Words Removal and Stemming/Lemmatisation) have been shown to be less effective in OM tasks, caused mainly by FPs. We propose a pipeline repair approach that aims to differentiate FPs and therefore improve Precision and F1 Score.

One critical step in our approach is to retrieve a reserved word set that may cause FPs after the text preprocessing, and these words will be excluded before the text preprocessing. The selection criteria are based on two widely agreed assumptions: (1) there are no duplicate entities within a single ontology; and (2) ontologies that represent the same domain knowledge tend to use similar terminologies (we call our approach context-based here because pairs of words may have the same or different meanings in different contexts). Based on these two assumptions, we propose a simple and intuitive Algorithm 1 to retrieve the reserved word set for context-based pipeline repair. For both O_s and O_t, the algorithm first iterates on all pairs of entities e_i, e_j in each of them. For a specific text preprocessing method $f(.)$, if $f(e_i) = f(e_j)$, we retrieve the different words between e_i and e_j and store them in the reserved word set. If a word appears in the reserved set, the text preprocessing pipeline skips the operation for this word. To simplify the reserved word set, we also remove the words where $f(w) = w$ from the final set because either skipping or keeping these words in the reserved word set would not change the mapping results.

An example of generating and using a simple reserved word set is illustrated below. (1) Two object properties *was_a_member_of* and *has_members* from a single ontology have the same result "member" via the traditional text preprocessing. Because there are no duplicate entities within a single ontology, we use a reserved word set in our proposed pipeline repair approach to determine that they are distinguishing entities. The initial step (i.e. Phase 1 of Algorithm 1) is to add ["was", "a", "member", "of", "has", "members"] to the reserved word set so that these two object properties would not have the same text preprocessing results. *was_a_member_of* preprocessed with skipping the reserved word set

Algorithm 1 Finding the reserved words

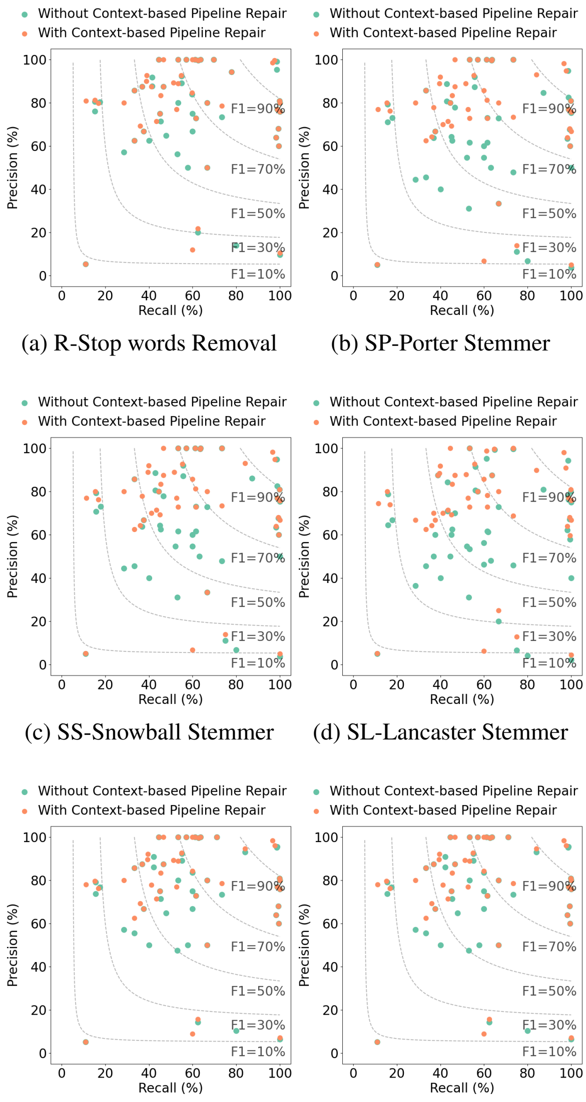
Input: Source Ontology O_s, Target Ontology O_t, Text Preprocessing Pipeline Method $f(x)$
Output: $Reserved\_Word\_Set$

```
/* Phase 1: Find duplicates in O_s and O_t */
/* Phase 1.1: Find duplicates in O_s */
for Entity e_i, e_j ∈ O_s do
    if f(e_i) = f(e_j) then
        Reserved_Word_Set ← differ(e_i, e_j)
    end if
end for
/* Phase 1.2: Find duplicates in O_t */
/* Same procedure applies ...*/
/* Phase 2: Find duplicates in Reserved_Word_Set */
for Word w ∈ Reserved_Word_Set do
    if f(w) = w then
        Reserved_Word_Set → w
    end if
end for
return Reserved_Word_Set
```

is "was a member of", while *has_members* preprocessed with the reserved word set is "has members". The revision step (i.e. Phase 2 of Algorithm 1) is to check whether there are duplicates in the reserved set. We can observe that the word "member" is a duplicate because it is the same before and after text preprocessing. Removing this word from the reserved word set still makes the two object properties different. Therefore, the final reserved word set is ["was", "a", "of", "has", "members"]. (2) The generated reserved word set can be used to repair false mappings between entities within the same domain context but coming from different ontologies. For example, we expect that the two object properties *has_a_steering_committee* and *was_a_steering_committee_of* are non-identical. While a false mapping may occur when they both have the same result "steer committe" after the traditional text preprocessing, using the reserved word set can repair this false mapping. As the words "has", "a", "was", and "of" are listed as reserved words, these two named properties are preprocessed as "has a steer committe" and "was a steer committe of", respectively.

4.3 Evaluation

We apply our context-based pipeline repair approach to the same OAEI track repositories and alignments as above. Figure 9 compares with and without context-based pipeline repair in 8 track repositories with 49 distinct alignments. The text preprocessing pipeline methods implemented include all the Phase 2 text preprocessing methods: Stop Words Removal (R), Porter Stemmer (SP), Snowball Stemmer (SS), Lancaster Stemmer (SL), Lemmatisation (L), and Lemmatisation + POS Tagging (LT). We can see that our context-based repair approach can significantly improve the Precision. As a trade-off, it may cause a slight decrease in Recall, but the overall F1 Score is still increasing in the majority of the alignments. For example, this repair approach applied to the MaterialInformation-EMMO alignment using Lancaster Stemmer improved Precision by 8.95%, with only a 3.17% decrease in Recall, and the overall F1 Score also increased by 2.92%. Details are in Appendix B.2.



(a) R-Stop words Removal (b) SP-Porter Stemmer

(c) SS-Snowball Stemmer (d) SL-Lancaster Stemmer

(e) L-Lemmatisation (f) LT-L + POS Tagging

Figure 9: Testing the context-based pipeline repair approach on Phase 2 text preprocessing methods (the total number of each category can appear to be less than 49 when data points overlap).

5 Conclusion

In this paper, we conduct a comprehensive study on the effect of the text preprocessing pipeline on syntactic OM. 8 OAEI track repositories with 49 distinct alignments are evaluated. Despite the importance of text preprocessing in syntactic OM, our experimental results indicate that the text preprocessing pipeline is currently ill-equipped to handle OM tasks. (1) We find that Phase 1 text preprocessing methods (Tokenisation and Normalisation) help with both matching completeness (i.e. Recall) and correctness (i.e. Precision). Phase 2 text preprocessing methods (Stop Words Removal and Stemming/Lemmatisation) are less effective. They can improve matching completeness (i.e. Recall), but matching correctness (i.e. Precision) is relatively low. (2) We propose a novel context-based pipeline repair approach to repair the less effective Phase 2 text preprocessing methods. By using a reserved word set to reduce false positive samples detected, our approach outperforms the traditional text preprocessing pipeline, in particular, the matching correctness (i.e. Precision) and overall matching performance (i.e. F1 Score). Figure 10 illustrates the mechanisms of two-phase text preprocessing and how our novel context-based pipeline repair approach successfully repairs the Phase 2 text preprocessing methods.

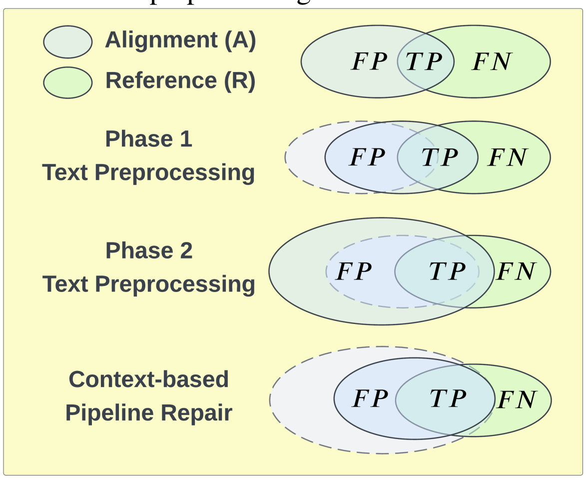


Figure 10: Two-phase text preprocessing and our context-based pipeline repair approach. (1) Phase 1 methods **shift Alignment (A) towards Reference (R)**. The number of TPs increases, while FPs decrease. (2) Phase 2 methods **expand Alignment (A)**. The number of TPs increases, but FPs also increase. (3) Our context-based pipeline repair approach **collapses Alignment (A)**. The number of FPs significantly decreases, with a slight decrease in TPs.

Our future work will focus on handling class axioms and complex relationships to evaluate the text preprocessing pipeline for OM tasks. We will also study the pipeline and pipeline repair approach working with both traditional knowledge-based OM systems and modern ML-based OM systems.

Limitations

OAEI track repositories are comprehensive but do not cover all real-world scenarios. Further, the reference files contained in the OAEI track repositories are not "complete" gold standards and some need further development. For example, in the *conference* track repository, a pair of exact matches (cmt:Paper, conference:Paper, ≡, 1) may be missing from the Reference (R). In this study, we only deal with equivalence mappings between named classes, object properties, and data type properties. Disjoint mappings are not considered. Finding widely used baseline data with ground truth matches for subsumptions is empirically difficult. Only a few OAEI track repositories contain subsumption mappings, and they are not enough to robustly demonstrate text preprocessing used in subsumption mappings.

Broader Impacts

Ontologies provide formalised conceptualisations of knowledge graphs (KGs). Incorporating ontologies for KG-related NLP tasks can enhance the capability to handle complex concepts and relations, thereby enabling accessible KGs from the text (e.g. Text-to-KG), or coherent text generation from KGs (e.g. KG-to-Text). However, aligning and integrating heterogeneous ontologies remains a challenge when using them for NLP tasks. OM aims to bridge the gap by capturing similar concepts that occur in different ontologies.

OM is a complex process that combines syntactic, lexical, and semantic matching. Each level of matching may use different matching techniques, for example, text preprocessing for syntactic matching, text vectorisation for lexical matching, and logical reasoning for semantic matching. This paper concentrates on text preprocessing, a common practice in syntactic matching. While generic text preprocessing pipeline methods in syntactic OM are usually applied on the basis of intuition or extrapolation from other experience, this work advances the state of knowledge towards making design decisions objective and supported by evidence.

(1) Our study shows (i) *whether* to use, or not use, and (ii) *how* to use these text preprocessing methods. Such experimental results will benefit decision making when selecting appropriate text preprocessing methods for syntactic OM. For large-scale OM, it can significantly reduce unnecessary trial costs.

(2) Our context-based pipeline repair approach is proposed to repair the less effective Phase 2 text preprocessing methods. Its broader value is showing *how to maximise true mappings and minimise false mappings throughout the text preprocessing pipeline*. From a statistical perspective, it is a local optimisation for syntactic matching that will benefit the global optimisation for OM, as syntactic matching provides "anchor mappings" for lexical and semantic matching.

In addition, this study also demonstrates that the nature of text preprocessing used in NLP applications can be context-based. Not all pipeline methods are effective, and some of them may even hamper the overall performance of downstream tasks. Modifications are required before applying text preprocessing to specific NLP tasks.

Ethical Considerations

In this study, we use the datasets from OAEI track repositories. According to the OAEI data policy (date accessed: 2024-06-01), "OAEI results and datasets, are publicly available, but subject to a use policy similar to the one defined by NIST for TREC. These rules apply to anyone using these data." For more details, please check the official link: `https://oaei.ontologymatching.org/doc/oaei-deontology.2.html`

Acknowledgements

Not available during the review process.

References

Murugan Anandarajan, Chelsey Hill, and Thomas Nolan. 2019. *Text Preprocessing*, pages 45–59. Springer International Publishing, Cham.

Anthropic. 2024. Claude Models.

Steven Bird, Ewan Klein, Edward Loper, and Jason Baldridge. 2008. Multidisciplinary Instruction with the Natural Language Toolkit. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*, pages 62–70, Columbus, Ohio. Association for Computational Linguistics.

Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, Denvar Antonyrajah, Ali Hadian, and Jaehun Lee. 2021. Augmenting Ontology Alignment by Semantic Embedding and Distant Supervision. In *The Semantic Web*, pages 392–408, Cham. Springer International Publishing.

Isabel F. Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. 2009. AgreementMaker: Efficient Matching for Large Real-World Schemas and Ontologies. *Proc. VLDB Endow.*, 2(2):1586–1589.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Kevin Donnelly et al. 2006. SNOMED-CT: The advanced terminology and coding system for eHealth. *Studies in health technology and informatics*, 121:279.

Jérôme Euzenat. 2007. Semantic precision and recall for ontology alignment evaluation. In *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 348–353. AAAI Press.

Jérôme Euzenat, Pavel Shvaiko, et al. 2007. *Ontology Matching*, volume 18. Springer-Verlag, Berlin, Heidelberg.

Daniel Faria, Catia Pesquita, Emanuel Santos, Isabel F Cruz, and Francisco M Couto. 2014. AgreementMakerLight 2.0: Towards Efficient Large-Scale Ontology Matching. In *ISWC (Posters & Demos)*, pages 457–460, Cham. Springer International Publishing.

Daniel Faria, Catia Pesquita, Emanuel Santos, Matteo Palmonari, Isabel F. Cruz, and Francisco M Couto. 2013. The AgreementMakerLight Ontology Matching System. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, pages 527–541, Berlin, Heidelberg. Springer.

Jennifer Golbeck, Gilberto Fragoso, Frank Hartel, Jim Hendler, Jim Oberthaler, and Bijan Parsia. 2003. The National Cancer Institute's thesaurus and ontology. *Journal of Web Semantics First Look 1_1_4*.

Yuan He, Jiaoyan Chen, Denvar Antonyrajah, and Ian Horrocks. 2022. BERTMap: A BERT-Based Ontology Alignment System. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):5684–5691.

Yuan He, Jiaoyan Chen, Hang Dong, Ian Horrocks, Carlo Allocca, Taehun Kim, and Brahmananda Sapkota. 2023. DeepOnto: A Python Package for Ontology Engineering with Deep Learning.

Sven Hertling, Jan Portisch, and Heiko Paulheim. 2019. MELT - Matching Evaluation Toolkit. In *Semantic Systems. The Power of AI and Knowledge Graphs*, pages 231–245, Cham. Springer International Publishing.

Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. 2011. LogMap: Logic-Based and Scalable Ontology Matching. In *The Semantic Web – ISWC 2011*, pages 273–288, Berlin, Heidelberg. Springer.

Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Yujiao Zhou. 2011. LogMap 2.0: Towards Logic-Based, Scalable and Interactive Ontology Matching. In *Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences*, SWAT4LS '11, page 45–46, New York, NY, USA. Association for Computing Machinery.

Maxat Kulmanov, Wang Liu-Wei, Yuan Yan, and Robert Hoehndorf. 2019. EL Embeddings: Geometric construction of models for the Description Logic EL ++.

Patrick Lambrix and He Tan. 2006. SAMBO—A system for aligning and merging biomedical ontologies. *Journal of Web Semantics*, 4(3):196–206. Semantic Web for Life Sciences.

LangChain Inc. 2024. LangChain.

Guoxuan Li, Songmao Zhang, Jiayi Wei, and Wenqian Ye. 2021. Combining FCA-Map with representation learning for aligning large biomedical ontologies. In *OM@ ISWC*, pages 207–208, Berlin, Heidelberg. Springer.

Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. 2009. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Transactions on Knowledge and Data Engineering*, 21(8):1218–1232.

Xiulei Liu, Qiang Tong, Xuhong Liu, and Zhihui Qin. 2021. Ontology Matching: State of the Art, Future Challenges, and Thinking Based on Utilized Information. *IEEE Access*, 9:91235–91243.

Meta. 2024. Meta Llama 3 Models.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space.

George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM*, 38(11):39–41.

Mistral AI. 2024. Mistral AI Models.

OAEI. 2023. Ontology Alignment Evaluation Initiative.

Ollama. 2024. Ollama.

OpenAI. 2024. OpenAI Models.

Cornelius Rosse and José L.V. Mejino. 2003. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. *Journal of Biomedical Informatics*, 36(6):478–500.

Claude Sammut and Geoffrey I. Webb, editors. 2010. *Encyclopedia of Machine Learning*. Springer US, Boston, MA.

Daniel Schober, Waclaw Kusnierczyk, Suzanna E Lewis, Jane Lomax, et al. 2007. Towards naming conventions for use in controlled vocabulary and ontology engineering. In *The 10th Annual Bio-Ontologies Meeting*.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 697–706, New York, NY, USA. Association for Computing Machinery.

Kerry Taylor, Simon Cox, and Linda van den Brink. 2015. Spatial Data on the Web Working Group: Ontology Design Principles.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Michael L. Waskom. 2021. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.

Mengyi Zhao, Songmao Zhang, Weizhuo Li, and Guowei Chen. 2018. Matching biomedical ontologies based on formal concept analysis. *Journal of biomedical semantics*, 9(1):1–27.

A Supplementary Material

The experiment code used in this paper has been submitted as a single .zip archive. The camera-ready version will use a GitHub link instead.

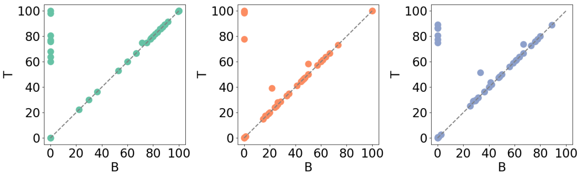
B Extended Experiment Details

B.1 Analysis of Text Preprocessing Pipeline

Figures 11, 12, 13, and 14 show the details of the experiment to compare the text preprocessing pipeline in syntactic OM.

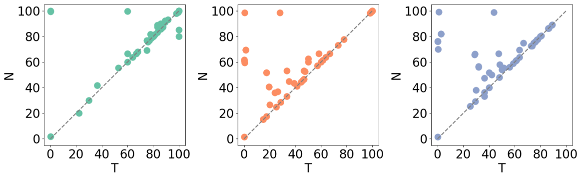
(1) For Phase 1 text preprocessing methods (Tokenisation and Normalisation), most of the data points located above the equivalent line in Precision, Recall, and F1 Score indicate that they help syntactic OM.

(2) For Phase 2 text preprocessing methods (Stop Words Removal and Stemming/Lemmatisation): Some data points are located above the equivalent line in Recall, but the majority of data points located below the equivalent line in Precision and F1 Score indicate that they do not help syntactic OM.



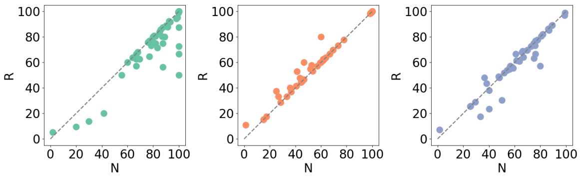
(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 11: Comparison of Base Entity without Text Preprocessing (B) vs. Tokenisation (T).



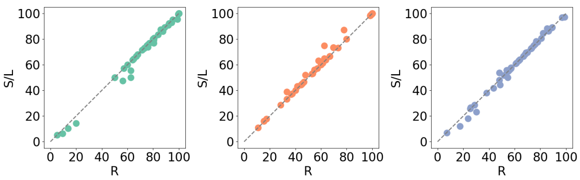
(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 12: Comparison of Tokenisation (T) vs. Normalisation (N).



(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

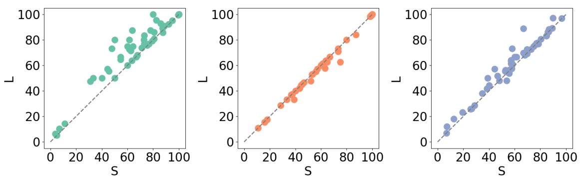
Figure 13: Comparison of Normalisation (N) vs. Stop Words Removal (R).



(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 14: Comparison of Stop Words Removal (R) vs. Stemming/Lemmatisation (S/L).

Figure 15 shows the details of the experiment to compare Stemming (S) and Lemmatisation (L). Most of the data points located above the L=S line in Precision and F1 Score indicate that using Lemmatisation is better than Stemming in syntactic OM (assuming that the post hoc correction is excluded).

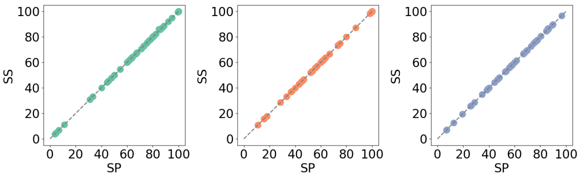


(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 15: Stemming vs. Lemmatisation.

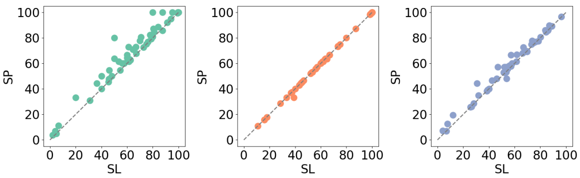
Figures 16, 17, and 18 show the details of the experiment to compare Porter Stemmer, Snowball Stemmer, and Lancaster Stemmer.

(1) For Porter Stemmer vs. Snowball Stemmer, all data points located in the equivalent line indicate that there is no difference in using Porter Stemmer and Snowball Stemmer in syntactic OM.
(2) For Porter/Snowball Stemmer vs. Lancaster Stemmer, most of the data points the data points located above the equivalent line in Precision and F1 Score indicate that Porter/Snowball Stemmer is more effective than Lancaster Stemmer in syntactic OM.



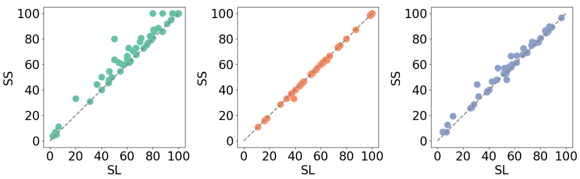
(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 16: Comparison of Porter Stemmer (SP) vs. Snowball Stemmer (SS).



(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

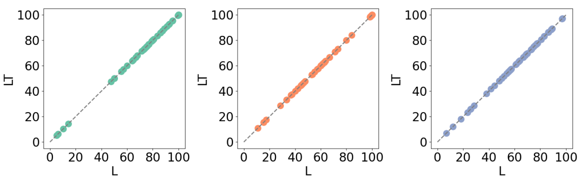
Figure 17: Comparison of Porter Stemmer (SP) vs. Lancaster Stemmer (SL).



(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 18: Comparison of Snowball Stemmer (SS) vs. Lancaster Stemmer (SL).

Figure 19 shows the details of the experiment to compare Lemmatisation vs. Lemmatisation + POS Tagging. All data points located in the equivalent line indicate that using POS Tagging in Lemmatisation does not help syntactic OM.



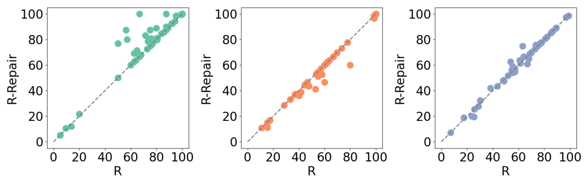
(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 19: Comparison of Lemmatisation (L) vs. Lemmatisation + POS Tagging (LT).

B.2 Context-based Pipeline Repair

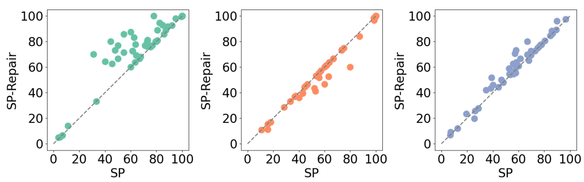
Figures 20, 21, 22, 23, 24, and 25 show the details of the experiment to consider the benefit of using context-based pipeline repair in Phase 2 text preprocessing methods (Stop Words Removal and Stemming/Lemmatisation).

(1) For Precision, most of the data points are located above the equivalent line, indicating that context-based pipeline repair significantly improves matching correctness.
(2) For Recall, most of the data points are located in the equivalent line, and only a few data points are located below the equivalent line, indicating that context-based pipeline repair slightly reduces matching completeness.
(3) For F1 Score, most of the data points are located above the equivalent line, indicating that context-based pipeline repair also improves overall matching performance.
(4) Experimentally, the wider ellipse around the data points indicates that the matching performance improvement ranking in Phase 2 text preprocessing methods is Stemming (S) $>$ Lemmatisation (L) $>$ Stop Words Removal (R).
(5) Experimentally, the wider ellipse around the data points indicates that the matching performance improvement ranking in different stemmers is Lancaster Stemmer (SL) $>$ Porter Stemmer (SP) = Snowball Stemmer (SS).
(6) Experimentally, the same ellipse around the data points indicates that the matching performance improvement ranking in lemmatisation is Lemmatisation (L) = Lemmatisation + POS Tagging (LT).



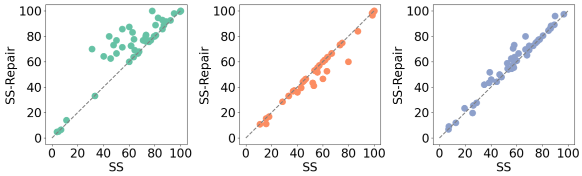
(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 20: Comparison of using and without using context-based repair in Stop Words Removal (R).



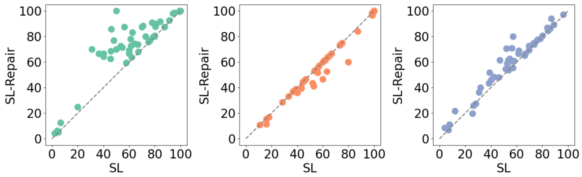
(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 21: Comparison of using and without using context-based repair in Porter Stemmer (SP).



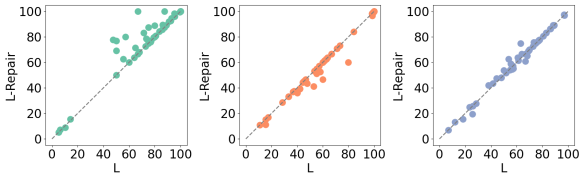
(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 22: Comparison of using and without using context-based repair in Snowball Stemmer (SS).



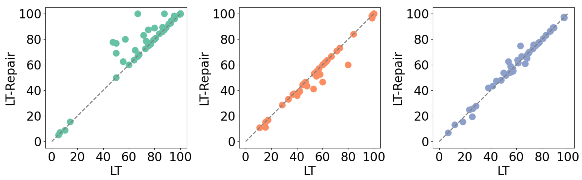
(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 23: Comparison of using and without using context-based repair in Lancaster Stemmer (SL).



(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 24: Comparison of using and without using context-based repair in Lemmatisation (L).



(a) Precision (%) (b) Recall (%) (c) F1 Score (%)

Figure 25: Comparison of using and without using context-based repair in Lemmatisation + POS Tagging (LT).

C Extended Discussion on LLMs

The use of large language models (LLMs) is a breakthrough in NLP, but it is not a one-size-fits-all solution for OM tasks. OM is a complex multi-level matching mixed with syntactic, lexical, and semantic matching. LLMs may not outperform traditional methods in performing these sub-matching tasks. The development and implementation of an LLM-based framework that incorporates different matching techniques is a new research direction for OM, but it is beyond the scope of this paper. In some scenarios where the information is sensitive or domain-specific, the models need to run locally. LLMs are difficult to run locally and retrain due to constraints on storage and computational power in some cases, and constraints on API access in others. Classical and ML-based OM systems (which can run locally and retrain) are still very useful. These two types of systems rely heavily on "seed mappings" produced by syntactic matching using classical text preprocessing.

C.1 LLMs and OM Text Preprocessing

We observe that some LLMs can perform syntactic matching without preprocessing, but their robustness remains questionable. We conduct a preliminary study on the use of LLMs for syntactic matching with and without text preprocessing. The experiment settings are described as follows:

(1) We select the same example described in Section 3.1.1. The compound word "art gallery" can have different naming conventions in different ontologies. For example, the Pascal case "ArtGallery" used in YAGO and the Snake case "art_gallery" used in Wikidata.

(2) We choose 10 LLMs from 4 different families. These include two OpenAI models (gpt-3.5-turbo and gpt-4-turbo) (OpenAI, 2024), three Mistral AI models (mistral-large, mistral-medium, and mistral-small) (Mistral AI, 2024), three Anthropic Claude models (claude-3-opus, claude-3-sonnet, and claude-3-haiku) (Anthropic, 2024), and two open-source Meta Llama models (llama-3-70b and llama-3-8b) (Meta, 2024). We use LangChain (LangChain Inc., 2024) to build the chats with OpenAI, Mistral AI, and Anthropic Claude models, and Ollama (Ollama, 2024) to access Meta Llama models. The versions of LLMs used in the experiment are listed in Table 2. All the model temperatures are set to 0 to minimise the random results generated by LLMs.

Series	LLMs	Versions
GPT	gpt-4-turbo	gpt-4-turbo-2024-04-09
	gpt-3.5-turbo	gpt-3.5-turbo-0125
Claude	claude-3-opus	claude-3-opus-20240229
	claude-3-sonnet	claude-3-sonnet-20240229
	claude-3-haiku	claude-3-haiku-20240307
Mistral	mistral-large	mistral-large-2402
	mistral-medium	mistral-medium-2312
	mistral-small	mistral-small-2402
Llama	llama-3-70b	date accessed: 2024-06-01
	llama-3-8b	date accessed: 2024-06-01

Table 2: Versions of LLMs used in the experiment.

(3) We generate the prompt template as "Is ArtGallery <KEYWORD> to/with art_galley?" (without text preprocessing) and "Is art galley <KEYWORD> to/with art galley?" (with text preprocess-

ing). We only use T and N to preprocess the text because R and S/L are shown to be less effective in the main content of the paper, and also R and S/T needs are not present in the actual naming of the ontology entities. The <KEYWORD> is a placeholder for a collection of words that can be used to describe an equivalence relationship. In this study, we experiment with three common words namely "identical", "interchangeable", and "equivalent".
(4) Considering the complexity of prompts, we also test an additional case where the prompts have a self-reflection phase (i.e. add "Write a short explanation" to the prompt). We use parentheses to indicate that the different responses generated by the prompts have a self-reflection phase.

Table 3 shows the results of using LLMs for syntactic matching without text preprocessing. We expect the LLMs to generate "Yes" answers for this TP sample. However, the majority of the LLMs output the "No" answers (marked with the red colour in the table) without text preprocessing. Table 4 shows the results of using LLMs for syntactic matching with text preprocessing. We can see that the error rate can be significantly reduced, but some LLMs still continue to produce incorrect results. The result also shows that LLMs with a larger number of parameters (e.g. llama-3-70b) and prompts with a self-reflection phase (labelled in parentheses) offer "too much of a good thing". They do not help with syntactic matching and may even produce more errors for OM tasks.

This preliminary study does not cover all possible triggers used in prompt engineering, but it is adequate to highlight the importance of proper preprocessing before fitting the text into the LLMs. We believe that the classical programming-based text preprocessing pipeline is more stable and reliable than using LLM's implicit text preprocessing or prompt-based text preprocessing for OM tasks.

C.2 LLMs and OM Text Preprocessing Pipeline Repair

Using LLMs for OM text preprocessing is not without merit. With strong background knowledge, they can be used to facilitate several subtasks in OM. For example, LLMs can be used as a repair tool for the text preprocessing pipeline, as an alternative to the approach we proposed in Section 4.2. The slight difference is that the context-based pipeline repair approach is an ad hoc repair prior to syntactic matching, while the LLM-based approach is a post hoc repair after syntactic matching. We conduct a preliminary study on LLMs used in OM text preprocessing pipeline repair. The experiment settings are described as follows:

LLMs	<KEYWORD>		
	"identical"	"interchangeable"	"equivalent"
gpt-4-turbo	No	No	No
gpt-3.5-turbo	No	No	No
claude-3-opus	No	Yes	Yes
claude-3-sonnet	No	No	No
claude-3-haiku	No	No	No
mistral-large	No	No	Yes
mistral-medium	No	No	No
mistral-small	No	Yes	Yes (No)
llama-3-70b	No	No	No
llama-3-8b	Yes	Yes	Yes

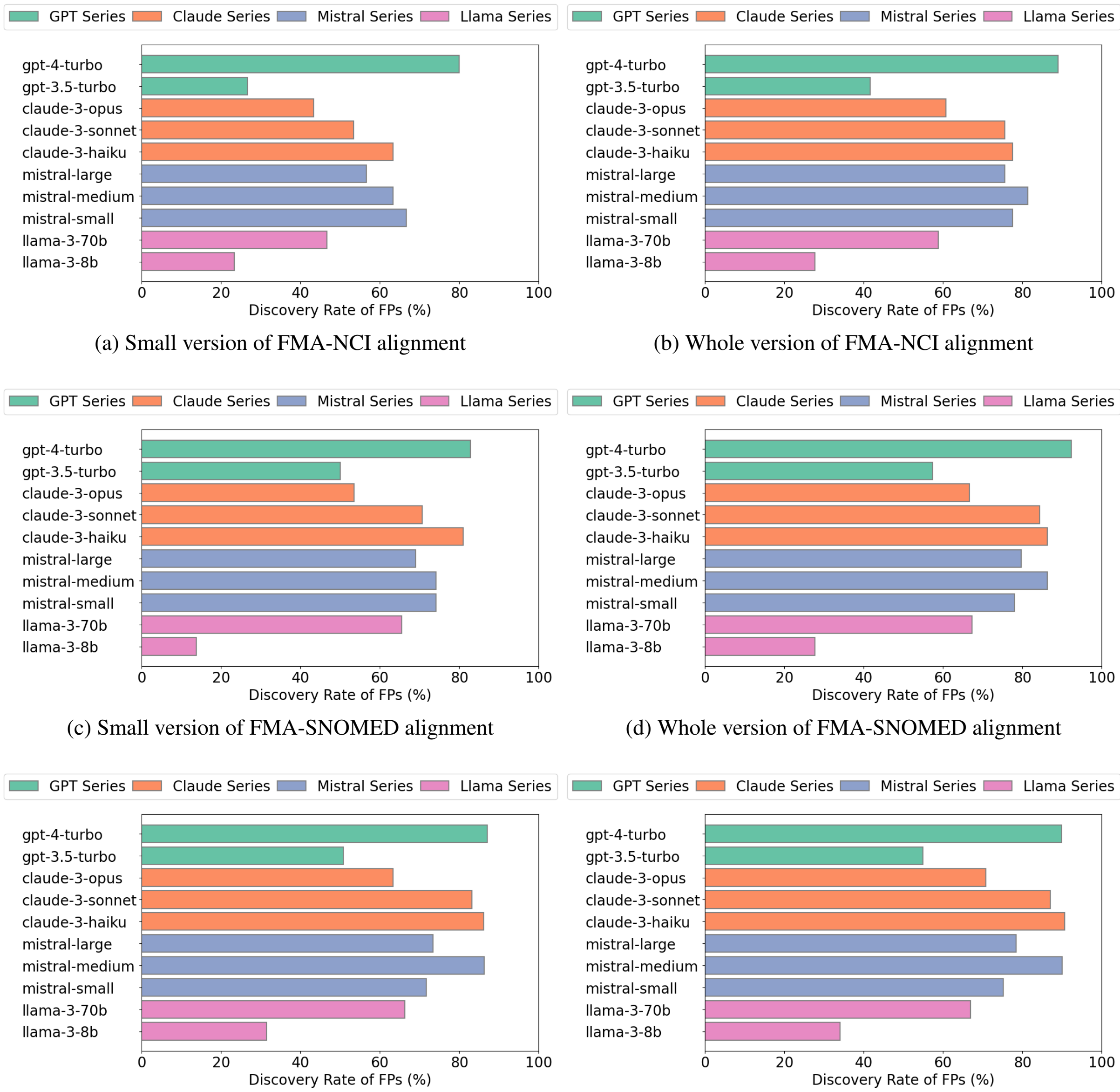
Table 3: Using LLMs for syntactic matching without text preprocessing. Prompt: "Is **ArtGallery** <KEYWORD> to/with **art_gallery**? Answer yes or no. (Write a short explanation.)"

LLMs	<KEYWORD>		
	"identical"	"interchangeable"	"equivalent"
gpt-4-turbo	Yes	Yes	Yes
gpt-3.5-turbo	Yes	Yes	Yes
claude-3-opus	Yes	Yes	Yes
claude-3-sonnet	Yes (No)	No	Yes (No)
claude-3-haiku	No	No	Yes (No)
mistral-large	Yes	Yes	Yes
mistral-medium	No	Yes	Yes
mistral-small	Yes	Yes	Yes
llama-3-70b	Yes	Yes (No)	Yes (No)
llama-3-8b	Yes	Yes	Yes

Table 4: Using LLMs for syntactic matching with text preprocessing. Prompt: "Is **art gallery** <KEYWORD> to/with **art gallery**? Answer yes or no. (Write a short explanation.)"

(1) The experiment is set up on the *largebio* track repository, a biomedical track repository requires higher Precision with low rates of FPs. The Lancaster Stemmer always shows lower Precision across 6 pairs of alignments, meaning that the results contain a significant number of FPs.
(2) We choose the same 10 LLMs used in the previous experiment in Appendix C.1.
(3) We use the prompt "Is X equivalent to Y?" The previous experiment in Appendix C.1 shows the keyword "equivalent" produces fewer errors, and a self-reflection phase does not help the OM tasks. We also preprocess the entity names X and Y with N and T before fitting them into the LLMs.

Figure 26 shows the discovery rate of FPs using 10 different LLMs. We expect the LLMs to generate "No" answers for these FP samples. The GPT model gpt-4-tubo, the Claude model claude-



(a) Small version of FMA-NCI alignment

(b) Whole version of FMA-NCI alignment

(c) Small version of FMA-SNOMED alignment

(d) Whole version of FMA-SNOMED alignment

(e) Small version of SNOMED-NCI alignment

(f) Whole version of SNOMED-NCI alignment

Figure 26: The discovery rate of FPs using LLMs on the *largebio* track repository.

3-haiku, the Mistral model mistral-medium, and the Llama model llama-3-70b achieve better performance in detecting FPs. For the GPT and Llama series, LLMs with a larger number of parameters perform better than those with a relatively smaller number of parameters. For the Claude and Mistral series, LLMs with larger parameters do not improve performance. In the 6 test cases we analysed, they are even less effective than those with a relatively smaller number of parameters.

C.3 Implications of LLMs used in OM

LLMs were originally developed for the purpose of question-answering (QA). OM is not only a QA task, but also a knowledge-intensive task that relates to a comprehensive understanding and reasoning of domain knowledge. The preliminary studies in this section demonstrate the limitations and opportunities of using LLMs for OM tasks. From our experience, we highlight two F&T pathways to customise LLM for a new domain task. One is functional tooling, which refers to teaching LLMs using external tools. In LLM-based text preprocessing (Appendix C.1), we could package the programming-based text preprocessing pipeline as a tool for LLMs to use. Another approach is fine-tuning, where LLMs are customised with local data. In LLM-based text preprocessing repair (Appendix C.2), we could use human-in-the-loop to validate the repaired results generated by LLMs and feed the validated data back into LLMs, so that they can better understand the context of the task.