An Evidence-based Verification Approach to Extract Entities and Relations for Knowledge Base Population

Naimdjon Takhirov[1], Fabien Duchateau[2] and Trond Aalberg[1]

[1] Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
{takhirov,trondaal}@idi.ntnu.no
[2] Université Lyon 1, LIRIS, UMR5205, Lyon, France
fduchate@liris.cnrs.fr

Abstract. This paper presents an approach to automatically extract entities and relationships from textual documents. The main goal is to populate a knowledge base that hosts this structured information about domain entities. The extracted entities and their expected relationships are verified using two evidence based techniques: classification and linking. This last process also enables the linking of our knowledge base to other sources which are part of the Linked Open Data cloud. We demonstrate the benefit of our approach through series of experiments with real-world datasets.

Keywords: Linked Data, Knowledge Extraction, Machine Learning

1 Introduction

The Web, which includes databases, catalogs and all textual documents, is a wealthy and a primary source of information. Thus, there is a need for exploiting this tremendous growth of the amount of online information as a source of structured knowledge [22]. Unfortunately, computers are not able to interpret this information due to a lack of semantics. However, the emergence of knowledge bases such as DBpedia and Freebase[3] in the Linked Open Data cloud (LOD), nowadays contain billions of facts expressed as RDF triples representing instances of relations between entities [4]. Researchers from various domains are increasingly interested in making their data available as part of the LOD, because a proper semantic integration of this data enables advanced semantic services. Examples of such services include exploratory search, supporting sophisticated and semantically rich queries, interoperability, question answering, etc. Converting the unstructured information, mainly the textual documents, to semantic models is therefore crucial to reach the expected Web of Data [15]. For instance, one of the most widely spread data representation used in the cultural heritage domain is *MARC* and its alternative forms. However, the Functional

[3] The complete list of interconnected bases can be found at http://linkeddata.org/

Requirements for Bibliographic Records (FRBR) model has gained much attention during the last decade as an underlying and much needed semantic data model for the cultural heritage data [20]. In this context, one of the most significant challenge deals with the **extraction of semantic information** hidden in plain documents [6, 14, 8]. Indeed, the textual documents are interesting because they may contain information that is otherwise missing or incomplete in the existing knowledge bases in the LOD cloud. Sentences in the documents include named entities which are connected with a specific type of relationship, e.g. *Martin Scorsese directed the movie The Departed*. Besides, the **interconnection of the LOD data sources** brings benefit for sharing and inferring knowledge [12]. Thus, extracting related entities from documents is not sufficient, and they need to be connected to the LOD cloud.

In this paper, we propose to tackle these two challenges. Our approach, KIEV[4] first extracts examples for a given relationship from textual documents. Indeed, some relationships are rarely encompassed in the structured data sources, but they can be found in textual documents (such as the Web). Mining these relationships with a pattern-based technique involves the discovery of a large amount of examples. Thus, a verification of these examples is performed at two levels: (i) the type of relationship is checked with a machine learning approach and (ii) the extracted entities are matched to LOD for both verification and integration purposes. In addition to these challenges, our approach KIEV should perform reasonably well in terms of efficiency at the Web scale since every page is a potential source of examples and good patterns. As a summary, the contributions of this paper are the following:

- We designed a generic approach for extracting semantic relationships from a large text corpora which integrates a verification process;
- These relationships are filtered and verified with a classification technique and an entity matching process. In addition, the link from our generated entity to its corresponding LOD entity enables the connection and possible reasoning over all interconnected knowledges bases;
- Finally, we have conducted experiments with real-world datasets (about movies and sports) to evaluate the quality and the robustness of our approach.

The rest of this paper is organized as follows. Section 2 introduces the formalization of our problem and provides an overview of KIEV. Section 3 covers the first part of our approach, the discovery of examples by using patterns, while Section 4 and 5 focus on the evidence-based verification of these examples. The related work is described in Section 6. Our experiments are detailed in Section 7. Finally, we conclude in Section 8.

2 Overview of our Approach

Our goal can be seen as **the creation of a knowledge base of entities and relationships**. Simply assuming the existence of a repository of domain

[4] KIEV – **K**nowledge and **I**nformation **E**xtraction with **V**erification

entities would limit our approach. Rather, we extract entities from the textual documents, and as a consequence, our approach should also work with entities which have been previously identified (i.e., from a repository). A **relationship** is defined as a triple $<entity_1, type\text{-}of\text{-}relationship, entity_2>$. As an example, considering the 2006 *"The Departed"* movie directed by Martin Scorsese as a remake of the Andrew Lau's *"Infernal Affairs"* from 2002, the example would be represented as *<"Infernal Affairs", hasImitation, "The Departed">*.

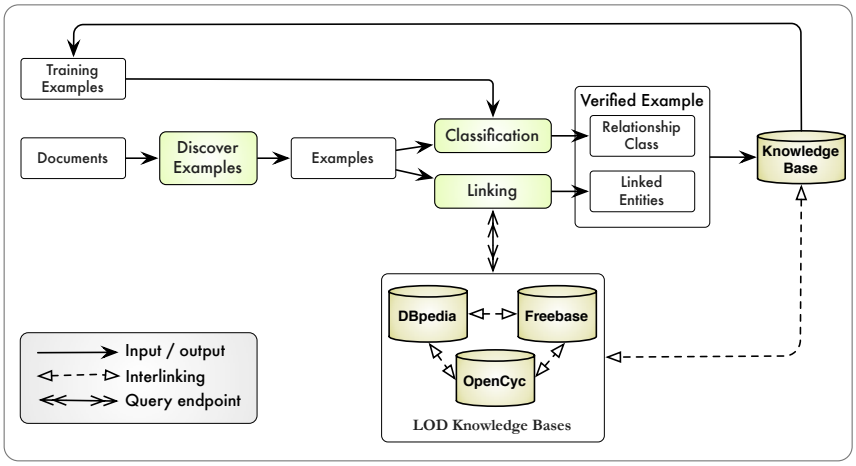


Fig. 1. Overview of our Approach.

Figure 1 depicts the global overview of KIEV. Given a type of relationship, KIEV requires a collection of documents and a few training examples (verifying the types of relationship) to bootstrap a possible infinite loop. The first step consists of **discovering examples** from the textual collection (see Section 3). It is based on semantic tagging which combines Named Entity Recognition and Part of Speech tagging, and it generates many examples for the concepts contained in a sentence. Thus, a verification of the relevance for these examples is performed with two other processes. The former checks if the extracted entities are effectively related with the type of relationship using a **machine learning classifier** (see Section 4). The latter process **links both extracted entities** of an example to their corresponding entities on the LOD cloud (see Section 5). Once an example is verified, it can be used as a training example to improve the classifier, but also to reinforce the confidence score of a pattern during the discovery process.

3 Discovering Examples

The core idea of our approach is to process the input as a stream of documents and to iteratively update our semantic knowledge base of entities. In this section, we describe the first part of our approach – discovering examples. An example for a given type of relationship is composed of two entities (e.g., for *imitation* type of relationship, an example is <*"Infernal Affairs", "The Departed"*>). Figure 2 provides the big picture of the example discovery workflow, whose goal is to generate a set of examples. Each process in the workflow of discovering examples is presented below.

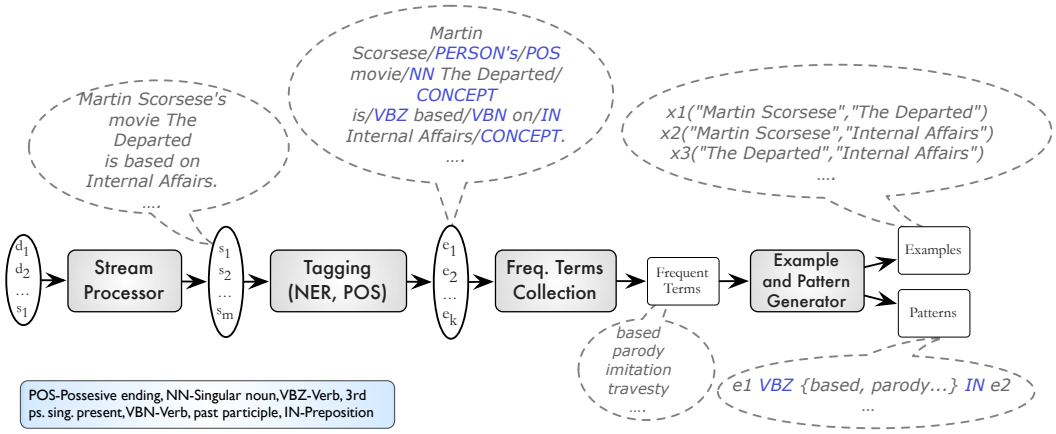


Fig. 2. Workflow of Processes for Discovering Examples

3.1 Stream Processing

Stream processor (SP) accepts as an input documents in textual form. The first task the SP performs is to pre-process the input. For example, this task may involve cleaning the html documents for tags, removing headers from emails, etc . At this point, we are interested in only obtaining text regardless of the quality. Each document $d \in \mathcal{D}$ is segmented into a list of sentences such that $d = \{S_i \mid i = 1 \dots N\}$ where N is the number of sentences. A sentence S_i is discarded if S_{i-1} and S_{i+1} contain no entities. This is because S_i may contain a personal pronoun referring to the previous sentence, e.g. *"**Martin Scorsese** is an American film director. **He** is the creator of Taxi Driver."*. Additionally, the sentences are filtered out to eliminate those that were likely to be noisy (broken and invalid sentences) and not useful for example discovery (e.g., non-English sentences, sentences missing verb, sentences with only uppercase letters

or only with lowercase letters, sentences without capital letters, etc.). The next step deals with the semantic tagging of the selected sentences with semantic information, namely named entity recognition (NER) and part-of-speech (POS) tags.

3.2 Semantic Tagging

For each sentence $s \in S_i$, named entity recognition is performed to detect the set of entities $\mathcal{E}$ (person, location, organization, dates, etc.). Consider a document containing the following sentence: *Infernal Affairs was followed by a 2006 American remake by Martin Scorsese entitled The Departed*. From this sentence, two *concepts* are detected and one *person*. Traditionally, NER is focused around the detection of common entities such as people, organization or geographic location. However, the recognition of domain specific entities poses a particular challenge because the NER tools usually require training examples for the types of entities to recognize. In our context of textual documents from the Web, providing such examples is not possible.

To avoid missing entities, a POStagger is first applied on all sentences. Our assumption is that entities are POStagged as *"noun"*. Thus, we consider that **all nouns in the sentences are entities**. A NER tool can confirm some of these entities. Although this assumption implies the identification of many incorrect entities, the next steps are in charge of discarding those irrelevant entities. The output of the semantic tagger is a set of semantically and structurally tagged sentences, from which we can extract frequent terms.

3.3 Frequent Terms Collection

Terms that appear frequently in the same sentence with a pair of entities are likely to be **highly relevant to the pair of entities**. For example, in the sentence *"Martin Scorsese's movie The Departed is based on Internal Affairs"*, frequent terms are *movie* and *based on* because they appear frequently together with the entities in the sentence.

In order to collect these frequent terms, all possible word n-grams are first identified in the sentence s. The top thousand most common words on the Web[5] are excluded and cannot be part of frequent terms. Then, the sentence s is splitted into a set of words. A list of n-grams is constructed out of this list. After the list of n-grams has been obtained, we look up Wordnet lexical database to obtain the list Φ of semantically related words. These words are grouped into unordered sets (synsets). Stopwords (e.g., "the", "a", "but" etc.) are removed and stemming is performed. The following Wordnet relations are used:

- synonymy (e.g., "writer" and "novelist"), words that denote the same concept and are interchangeable in many contexts.
- hyponym, a word whose semantics are included within that of another word[6], e.g., "The Departed is a movie".

[5] This list is available from Microsoft Web N-gram Service: `http://bit.ly/bFKSxz`

[6] This is similar to *is-a* relationship

Since the synsets obtained from Wordnet have a shared information content, i.e., hierarchy of is-a concepts, this list of semantically similar words can be larger than desired. Thus, to control the level of granularity of this list of concepts, we employ the Resnik similarity to prune those that are below a given threshold [16]. This similarity measure is applied between the segmented n-grams and each of the synsets in Φ. For example, the distance between "novel" and "book" is 0.29.

These frequent terms are generated for different objectives such as the classification of examples through features, but also to generate the examples as explained in the next part.

3.4 Example and Pattern Generator

Having obtained the lists of named entities and frequent terms, a **set of candidate examples** is built. One of our goals is to populate a knowledge base that can serve as a repository of distinct entities. First, a set of unique pair of entities Θ is constructed such that $\Theta = \{(e_i, e_j) | e_i \neq e_j, e_i \in \mathcal{E}, e_j \in \mathcal{E}\}$. At first glance, it appears that we generate overly many examples and this most likely leads to a fair number of false positives. But we will show in section 4 that our classification approach effectively discards these irrelevant examples. Our basic assumption with generating so many examples is to reduce the likelihood of low recall.

At this time, we can **generate patterns** based on the information from the frequent terms collector. That is, we mask the named entities (e.g. "Infernal Affairs" $\Rightarrow e_1$, "The Departed" $\Rightarrow e_2$). The idea is to obtain entity and word independent patterns, as shown in the Figure 2. At the end of each iteration, a list of patterns is generated from the candidate examples. If the pattern had been generated before, its statistics are updated from the current iteration. For patterns $\{p_1, \ldots, p_n\}$, we compute the pattern similarity using the Levenshtein distance and those above a given threshold are merged into a set of patterns P_p. By now, we know the amount of patterns generated in this iteration (P_i). We note the list X_p of examples that support this pattern. The patterns generated at iteration i are ranked according to the following scoring function:

$$score(p) = \frac{\alpha \frac{occ(p)}{i} + \beta \frac{|P_p|}{|P_i|} + \gamma \frac{|X_p|}{|X|}}{\alpha + \beta + \gamma}$$

where $occ(p)$ is the number of iterations this pattern has been discovered out of total number of iterations i. X denotes the number of total examples in the system. The scores are normalized in the range $[0, 1]$. The patterns generated during this iteration will be used to discover new examples in the next iteration. These patterns will also be used as features during the classification process.

As previously explained, all of the examples discovered so far may not be correct. In the next section, we will show how a classifier effectively discards false positives.

4 Classification

The first part of the verification is to check that the candidate entities (represented with a label) are related with a type of relationship. Indeed, a sentence may contain different entities and the discovery process generates in that case incorrect examples, mainly because of the pattern-based matching. The classification aims at discarding these incorrect examples without prior knowledge about the two entities. To fulfill this goal, the verification process can be seen as a **classification problem** [13]. Given a set of features (properties), the idea is to find the correct class for a given example (extracted from a sentence). Each class represents a type of relationship (e.g., *imitation, adaptation*). For instance, the example *(James Cameron, Avatar)* should be classified in the class *creatorOf*. A specific class named *unknown relationship* is added as a garbage class to collect all incorrect examples or those that cannot be classified in another class. To select the correct class for an example, a classifier is trained using training examples, i.e., examples for which the correct class is already known. Although the training process depends on the type of classifier (e.g., decision tree, Bayes network), it mainly consists of minimizing the misclassification rate when classifying the training examples according to the values of their features [13]. To compute these values, each training example is used as a query over the document collection and all sentences containing the two entities of the example are analyzed given the following features: the frequency and the presence of any frequent terms (e.g., *parody*), the length and structure of the best-ranked pattern which generated the example (see Section 3.4), the average spamscore of the documents from which the pattern is extracted [7]. Note that this paper does not aim at designing a new classifier, but we rather use existing ones from the Weka environment [9]. More formally, an example $x \in \mathcal{X}$ is defined by a set of features $\mathcal{F}$. We note the set of training examples $\mathcal{T}$, with $\mathcal{T} \subseteq \mathcal{X}$. Each example can be assigned a class $c \in \mathcal{C}$. Given a (type of) classifier Γ, we formulate the training as a process to obtain an instance γ of this classifier as follows:

$$\Gamma(\mathcal{T}, \mathcal{F}, \mathcal{C}) \rightarrow \gamma$$

The advantage of building a generic classifier rather than many binary classifiers (for each type of relationship) is that the former enables the verification of different types of relationships. Consider a query for *"imitation"*, we could obtain the pair of entities <*"Infernal Affairs", "The Departed"*> and <*"The Departed", Martin Scorsese"*>. With a binary classifier for "imitation", we would only keep the first example. With a generic classifier, we would store both examples (classified in different classes). When an instance of a classifier which best minimizes the misclassification rate is trained, we can use this instance γ for assigning classes to the unclassified examples:

$$\gamma(\mathcal{X}, \mathcal{F}, \mathcal{C}) \rightarrow <(x_1, c_1), (x_2, c_1), (x_3, c_4), \ldots, (x_k, c_n)>$$

In our context, we cannot assume that the user provides many initial training data. A set of 5 to 10 examples for each class is realistic. However, some

classifiers are robust with a few training examples while other classifiers achieve better results with more training data. Two problems arise from these remarks: the former is about selecting which examples should be added as training data while the latter deals with the choice of the classifier for each iteration. Let us discuss **the choice of the training data** first. To improve the robustness of the classifier, one has to train it with more data. To add new examples as training data, we have to select them among the sets of discovered examples from the previous iterations. We propose two strategies to achieve this goal. The first one (linking based) consists in selecting all examples that have been verified (with the classification step and the linking process) during any previous iterations. The second strategy (frequency based) is based on a frequency constraint: all examples which have been discovered in half of the previous iterations are added as training data during the current iteration. We believe that this selection of training data could be investigated further, e.g., when combining the two described strategies.

As for **the selection of the classifier**, the idea is the following: with the selected training examples, we generate instances of different types of classifiers (decision trees such as *J48* or *NBTree*, instance-based such as *KStar* or *IBk*, rule-based such as *NNge* or *JRip*, etc.). We perform cross-validation against the set of training examples for each instance of a classifier, and we compute the misclassification rate for each of them. The instance of classifier which achieves the minimal misclassification rate is selected to classify the examples discovered at this iteration. Such a strategy enables us to ensure that the best classifier is used for each iteration, but it also brings more flexibility to our approach.

We will show the impact of the training data and the type of classifiers in Section 7. The result of the classifier is a set of pairs, each of them composed of an example and its verified relationship class. The next step is to check whether the two extracted entities have a corresponding LOD entity.

5 Entity Linking

Entity Linking is the task of discovering local entity's correspondence in another data source [19]. The interest in linking entities is increasing rapidly due to the LOD movement. Note that linking does not imply coreference resolution is performed, but linking partially solves the coreference resolution problem. For example, the local entities "Martin Scorcese" and "Scorcese" are both linked to the same DBpedia entity *Martin_Scorsese*. The kind of linking we are performing here differs from structure-based linking as we **only have labels** at our disposal. The core of the idea is to **match the entity against existing general purpose semantic knowledge bases** such as DBpedia or Freebase to obtain corresponding LOD entities. Namely, we build various queries by decomposing the initial label and we query in the *descriptive text* attributes of knowledge bases (i.e., *common.topic.article* for Freebase, *dbpedia-owl:abstract* for DBpedia, etc.). In most cases, several candidate entities are returned and the task deals with automatically selecting the correct one. To fulfill this goal, the intuition is

based on the hypothesis that the document about entity e and the descriptive text of LOD entity l should be fairly similar. Linking is performed for each entity of each document. That means that each document where e is mentioned serves as a context for disambiguation and matching against LOD knowledge bases. We note $\boldsymbol{\xi}$ the vector of terms in e's document, while $\boldsymbol{\Lambda}$ represents the vector of terms of l. Terms in both documents are treated using *bag-of-words* method and both the context of e and the descriptive text of l are represented as a point in an n-dimensional term space. The cosine similarity score between the vectors $\boldsymbol{\xi}$ and $\boldsymbol{\Lambda}$ is calculated as follows:

$$sim(\boldsymbol{\xi}, \boldsymbol{\Lambda}) = \frac{\sum_{i=1}^{n} \boldsymbol{\xi}_i \times \boldsymbol{\Lambda}_i}{\sqrt{\sum_{i=1}^{n} (\boldsymbol{\xi}_i)^2} \times \sqrt{\sum_{i=1}^{n} (\boldsymbol{\Lambda}_i)^2}}$$

where n is the size of the vocabulary. The terms in both vectors are based on classical *tf/idf* scores while the vocabulary is created out of the whole document collection. The top ranked entities are chosen as candidates for further comparison. This last comparison is performed on labels (and optionally "redirects" property) of the two entities to ensure a reasonable similarity in the label of e and one of the labels of l (e.g. "rdfs:label" and "dbpedia-owl:wikiPageRedirects" for DBpedia). This comparison is necessary because even though the similarity function returns a sufficiently high cosine similarity score, the labels should also be lexically similar. At this stage, the three well-known similarity measures are applied (Jaro Winkler, Monge Elkan and Scaled Levenshtein) as described in [19]. The top linked LOD entity is stored and is considered as a candidate until the end of the iteration. At the end of an iteration, all verified relationships (both by the classification and the linking) are **converted into triples**: for each entity, some triples express the link to LOD, the different labels and other possible attributes. One triple represents the relationship between the two entities and the type of relationship. Thus, the knowledge base is populated iteratively and can run continuously.

6 Related Work

In the field of knowledge extraction, various works have been proposed to discover relationships for specific domains [22]. For instance, **Snowball** associates companies to the cities where their headquarter is located [1] while **DIPRE** focuses on books and authors [5]. To increase the quality and the consistency of generated facts, systems may either be based on general ontologies such as Yago [14] or on logical rules associated with a SAT solver [18]. The last trend in this domain deals with **Open Information Extraction**, in which the large scale aspect of the Web is taken into account [8]. However, none of these works clearly aim at building a semantic knowledge base, thus there is no linking with the LOD cloud.

DBpedia is one of the first initiative to automatically extract structured content from Wikipedia [4]. It relies on the infoboxes provided by the knowledge-sharing community. Since, many companies and organizations have added their own knowledge base to the LOD cloud, from generic ontologies such as **Yago** and **Freebase** to specialized bases such as **MusicBrainz** or **LinkedMDB** [10]. The process for converting unstructured or semi-structured data sources into facts is called *Triplification*[7]. For instance, **Triplify** has been designed to extract triples from Relational databases and expose them on LOD [2] while **Catriple** builds a store of triples from Wikipedia categories [11]. Similarly to most of these approaches, we generate triples and store them in our knowledge base.

In the Information Retrieval domain, researchers have studied the discovery of the corresponding LOD entities for a given task, such as in the **TREC challenge** [3]. Due to the large scale application and the uncertainty of the results, a ranking of the most probable entities which correspond to the query (usually with target categories) is computed [21, 17]. The linking to LOD for disambiguation and enrichment has also been studied for any bag of words [12] as well as for FRBR entities [19]. In our context, the entities are extracted from textual documents and usually represented with a label. The surrounding context of the label in the sentence is the main information available for discovering the corresponding LOD entity.

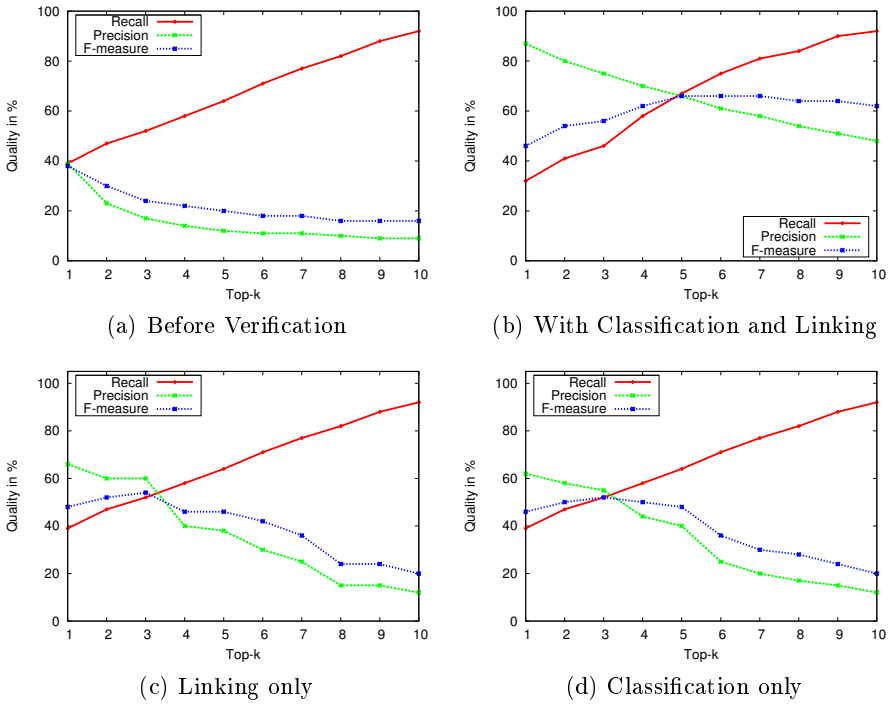
7 Experimental Evaluation

To assess the effectiveness of our approach, we have conducted a number of experiments which are presented below.

7.1 Experimental settings

Our **document collection** is the English subset of the ClueWeb09 dataset[8] which consists of 500 million documents. This dataset is used by several tracks of the TREC conference [3]. For **semantic tagging**, several text processing tools have been used, including OpenNLP[9] (for tokenization and sentence splitting), the StanfordNLP[10](for POS tagging). For **classification**, six classifiers of different types were applied, namely the classic Naïve Bayes, the rule-based (NNge, DecisionTable), tree-based (J48, RandomForest) and lazy (KStar). These classifiers are included in the Weka software [9]. As for **linking**, we have used the DBpedia[11] dataset version 3.7 which contains 3,550,567 triples. Apache Lucene was employed for the **backend indexing**. Running KIEV for one type of relation on a subset of the collection took roughly 20 minutes.

[7] `http://triplify.org/Challenge/`
[8] `http://lemurproject.org/clueweb09/`
[9] `http://opennlp.apache.org/`
[10] `http://nlp.stanford.edu/`
[11] `http://wiki.dbpedia.org/Downloads37`



(a) Before Verification (b) With Classification and Linking

(c) Linking only (d) Classification only

Fig. 3. Quality Results for the Remakes Dataset

7.2 Quality of Discovery Process

In this experiment, we focus on the movie dataset (remakes). Examples of relationships of interest include *imitation*, *adaptation* and *creator*. The ground truth for this dataset was obtained from the IMDb[12] movie database. This ground truth contained 545 entries out of the total 1052 remake pairs. For the remaining 507 we could not find suitable documents in our collection. The reason for this is twofold. First, a number of movies were in non-English language. Second, a significant number of movies were created before the *Information Age*, i.e., those produced earlier than 1970s. Additionally, some examples were only mentioned in a few documents.

Figure 3 demonstrates the results of our experiments **with or without the evidence-based verifications**. The quality is presented in terms of well-known information retrieval measures - recall, precision and F-measure. Extracted examples are ranked and thus presented by top-k. In our context, the recall (at top-k) is the fraction of extracted correct examples (at top-k) out of the total number of correct examples (at top-k), while the precision (at top-k) is the number of extracted correct examples (at top-k) out of the total number of extracted examples (at top-k). F-measure is the harmonic mean of precision and recall.

[12] `http://imdb.com`

We first notice that before the verification process, the precision score is quite low (≈39%) at top-1. This is because the discovery process extracts quite a lot of incorrect examples (false positives). As we increase the top-k, the recall also increases and eventually peaks at 87% at top-10. This trend illustrates that our approach achieves fairly high recall value but at the expense of precision. We tackle this issue with our verification techniques, i.e., classification and linking. To show the **benefit of both verification steps**, the individual results of these steps are depicted in Figure 3(c) and 3(d). The recall value for both classification only and linking only is very similar to the values before verification, thus confirming that the individual verification do not discard many correct relationships. And the precision values for both steps, which are lower than the precision score after verification at any iteration, indicate that classification and linking do not discard the same incorrect examples. Thus, they enable a higher precision when they are combined.

Figure 3(b) illustrates that the **verification process** is effective to discard incorrect examples (precision score reaching ≈85% at top-1). However, a few correct relations were also discarded (a ≈6% decrease of recall at top-1), mainly due to the missing of a link to LOD of one of the entities. Furthermore, this phenomenon involves changes in the ranking of the extracted examples. Correct relationships can be promoted to a higher top, thus increasing the recall value of the highest top (e.g., at top-5). Finally, the benefit of the verification process clearly appears at top-10, since the plots have a close recall value (≈87%) but the verification discarded half of the incorrect examples (50% precision).

7.3 Impact of the Training Data

The example discovery process *feeds* the classifier with new training data for the subsequent iteration. In this experiment, we have studied the impact of the selection of this training data by comparing the two strategies described in Section 4.

Frequency based strategy. The frequency based strategy accounts for the frequency of a given example being discovered in all iterations. Initially, the user provides a set of 20 training examples (5 per relation type). If a given example is discovered repeatedly on each iteration, the intuition behind this strategy is that this example is most likely valuable and is promoted as a training example in the next iteration. Figure 4 illustrates the impact of the training data at the i-th iteration. On the y axis, we have the number of training examples that is used by our classifiers. On the right $y2$ axis, we have the harmonic mean F-measure obtained by the best performing classifier at the i-th iteration. The best performing classifier is the one with the highest F-measure during the classification with 10-fold cross-validation against the training data. Note that from one iteration to the other, the best performing classifier may be different because the set of training data evolves. For example, *KStar* was selected as the best classifier for the first iteration, but *J48* performed better in the second iteration.

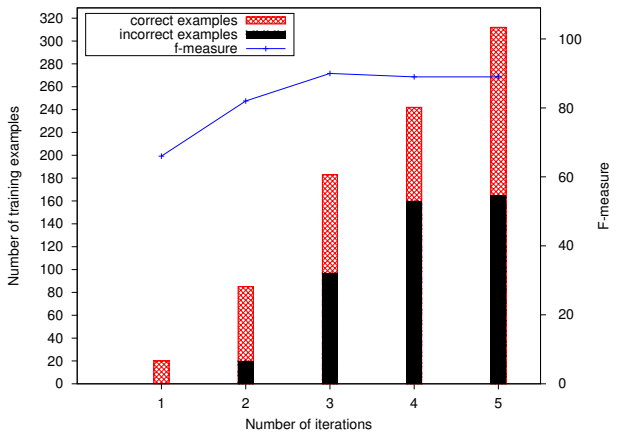


Fig. 4. Impact of Training Examples with Frequency based Strategy

On the plot, 85 examples discovered during the first iteration are selected for training for the second iteration. However, 20 of them are incorrect, i.e. false positives (shown as a black bar). Both the number of correct and incorrect examples increases as we move towards the 5-th iteration, eventually reaching 312 and 165 examples respectively for correct and incorrect examples. The high number of examples can be explained as follows. The frequency based strategy promotes as training data examples which appear at least 50% of the time in the previous iterations. Thus the number of added examples can potentially grow high. Yet, the F-measure obtained on the remakes dataset does not suffer much from the presence of incorrect examples (stable around 89% after the 3-rd iteration).

Linking based strategy. The linking based strategy provides a harder constraint than the frequency based strategy when selecting the training data. Indeed, the candidate examples have to be verified both by the classification and by the linking process. Let us study the impact of this strategy over the quality of results by analyzing Figure 5. It presents the F-measure value achieved by the best generated classifier and the evolution of the number of training examples for five iterations.

The first remark about this plot deals with the F-measure scores, which are higher than those of the frequency based strategy from iterations 1 to 5. Another interesting phenomenon with this strategy is that the number of examples selected as training data (y axis) is lower than the one of the frequency based strategy. Indeed, the linking based strategy requires that both entities of an example are linked to LOD. Thus, this number is dramatically reduced, i.e., 200 in linking based strategy versus 312 in the frequency-based strategy at the fifth iteration. Finally, the number of incorrect examples is much lower in the linking based strategy too.

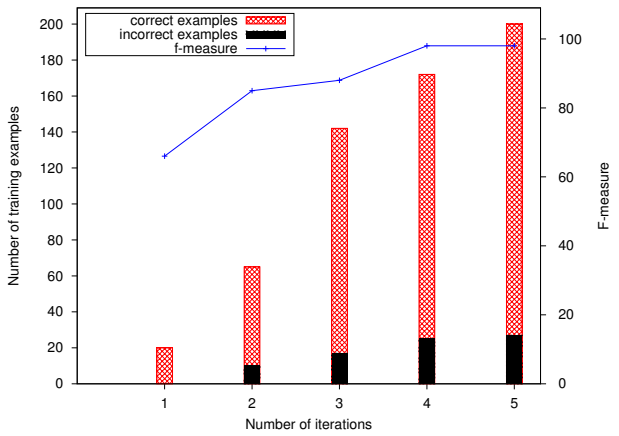


Fig. 5. Impact of Training Examples with Linking based Strategy

These remarks about the total number of correct examples together with the higher F-measure value are clear indicators that the linking based strategy is quality oriented while the frequency based strategy is performance oriented (simple and fast computation). The latter strategy is more appropriate for quickly generating training examples.

7.4 Comparative Evaluation

Finally, the evaluation of KIEV would not be complete without a **comparison with similar knowledge extraction systems**. Two systems, Prospera and NELL, are publicly available along with their dataset about *sports*. The results

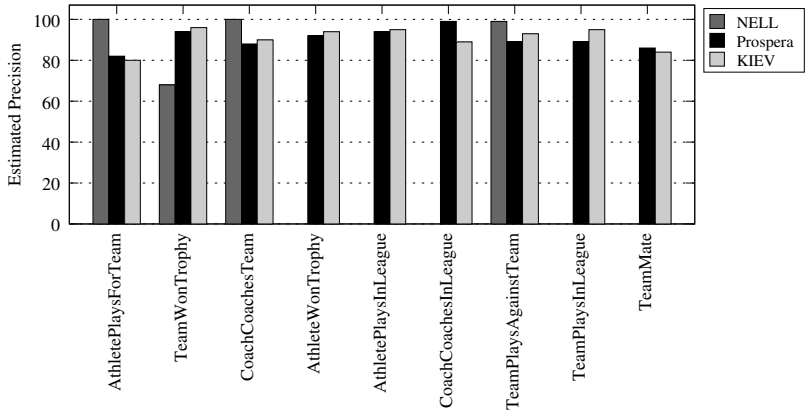


Fig. 6. Comparison to NELL and Prospera

of these systems over the *sport* dataset are reported in [6, 14]. To be fair in this evaluation, we have used the same set of training examples, and we also validated 1000 random types of relationship, as explained in the experiments reported in [6, 14]. This means that similarly to Prospera and NELL, our precision is an estimation, due to the amount of relationships to be validated.

Figure 6 summarizes the comparison between the three systems in terms of estimated precision. We notice that the average precision of the three systems is the same (around 0.91). However, the total number of facts discovered by KIEV (71,921) is 36 times higher than NELL (2,112) and 1.3 times higher than Prospera (57,070). As a consequence, KIEV outperforms both baselines. Prospera provides slightly better quality results than our approach on the *AthletePlaysForTeam* relationship. However, several factors have an influence on the precision results between Prospera, NELL and KIEV. First, Prospera is able to use seeds and counter seeds while we only rely on positive examples. On the other side, Prospera includes a rule-based reasoner combined with the YAGO ontology and KIEV mainly uses the LOD cloud for verification purposes. Yet, the combination of POS-tagged patterns and NER techniques supported by the two verification steps achieves outstanding precision values.

8 Conclusion

We have presented our novel approach KIEV for **populating a knowledge base** with entities and relationships. Our approach enables the analysis of a large amount of documents to extract examples (of entities) with their expected type of relationships after each iteration. A **verification step** ensures an acceptable quality for these extracted relationships by discarding irrelevant examples (classification) and by discovering the corresponding LOD entities (entity linking). Experiments performed on different datasets confirm the significant benefit of the verification step, thus enabling our approach to run continuously and to use new examples as training data to strengthen both the produced classifier and consequently the verification process.

The outcome of this work provides several interesting perspectives. First, we plan to run more experiments to **analyze the impact of parameters** (e.g., selection of the training data, number of iterations on the long term). We could associate a confidence score (based on provenance, number of patterns, number of occurrences, etc.) to each discovered relationship to rank them and help discarding the incorrect ones. Another objective is to study the **architecture and implementation of the knowledge base** in terms of infrastructure and support for RESTful and SPARQL queries. When our knowledge base will be publicly available, we plan to **integrate user feedback** to address the potentially contradictory cases between the two verification steps (classification and linking). Then, an extension could be proposed to **discover any type of relationship**, from an ontology for instance, by automatically defining the features and the training examples.

References

1. E. Agichtein and L. Gravano. Snowball: Extracting Relations from Large Plain-Text Collections. In *Proceedings of ACM DL*, pages 85–94, 2000.
2. S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: light-weight linked data publication from relational databases. In *Proceedings of WWW*, pages 621–630, 2009.
3. K. Balog, P. Serdyukov, and A. P. de Vries. Overview of the TREC 2011 entity track. In *Proceedings of TREC 2011*. NIST, 2012.
4. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal of Semantic Web and Information Systems*, 5(3):1–22, 2009.
5. S. Brin. Extracting patterns and relations from the world wide web. In *Proceedings of WebDB*, pages 172–183, 1998.
6. A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of AAAI*, pages 1306–1313, 2010.
7. G. V. Cormack, M. D. Smucker, and C. L. A. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Inf. Retr.*, 14(5):441–465, 2011.
8. O. Etzioni, A. Fader, J. Christensen, S. Soderland, and Mausam. Open Information Extraction: The Second Generation. In *Proceedings of IJCAI*, pages 3–10, 2011.
9. S. R. Garner. WEKA: The Waikato Environment for Knowledge Analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference*, pages 57–64, 1995.
10. O. Hassanzadeh and M. P. Consens. Linked Movie Data Base. In *Proceedings of LDOW*, 2009.
11. Q. Liu, K. Xu, L. Zhang, H. Wang, Y. Yu, and Y. Pan. Catriple: Extracting triples from wikipedia categories. In *Proceedings of ASWC*, pages 330–344, 2008.
12. D. Milne and I. H. Witten. Learning to link with wikipedia. In *Proceedings of CIKM*, pages 509–518, 2008.
13. T. Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), Oct. 1997.
14. N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of WSDM*, pages 227–236, 2011.
15. R. Parundekar, C. A. Knoblock, and J. L. Ambite. Linking and building ontologies of linked data. In *International Semantic Web Conference*, pages 598–614, 2010.
16. P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research (JAIR)*, 11:95–130, 1999.
17. H. Rode, P. Serdyukov, and D. Hiemstra. Combining document- and paragraph-based entity ranking. In *Proceedings of ACM SIGIR*, pages 851–852, 2008.
18. F. Suchanek, M. Sozio, and G. Weikum. SOFIE: a self-organizing framework for information extraction. In *Proceedings of WWW*, pages 631–640, 2009.
19. N. Takhirov, F. Duchateau, and T. Aalberg. Linking FRBR Entities to LOD through Semantic Matching. In *Proceedings of TPDL*, pages 284–295, 2011.
20. The International Federation of Library Associations and Institutions. Functional requirements for bibliographic records. In *UBCIM Publications - New Series*, volume 19, 1998.
21. A.-M. Vercoustre, J. A. Thom, and J. Pehcevski. Entity ranking in Wikipedia. In *Proceedings of ACM SAC*, pages 1101–1106, 2008.
22. G. Weikum and M. Theobald. From information to knowledge: harvesting entities and relationships from web sources. In *Proceedings of PODS*, pages 65–76, 2010.