How to Keep a Knowledge Base Synchronized with Its Encyclopedia Source

Jiaqing Liang[12], Sheng Zhang[1], Yanghua Xiao[134]*
[1]School of Computer Science, Shanghai Key Laboratory of Data Science
Fudan University, Shanghai, China
[2]Shuyan Technology, Shanghai, China
[3]Shanghai Internet Big Data Engineering Technology Research Center, China
[4]Xiaoi Research, Shanghai, China
l.j.q.light@gmail.com, {shengzhang16,shawyh}@fudan.edu.cn

Abstract

Knowledge bases are playing an increasingly important role in many real-world applications. However, most of these knowledge bases tend to be outdated, which limits the utility of these knowledge bases. In this paper, we investigate how to keep the freshness of the knowledge base by synchronizing it with its data source (usually encyclopedia websites). A direct solution is revisiting the whole encyclopedia *periodically* and rerun the entire pipeline of the construction of knowledge base like most existing methods. However, this solution is wasteful and incurs massive overload of the network, which limits the update frequency and leads to knowledge obsolescence. To overcome the weakness, we propose a set of synchronization principles upon which we build an **U**pdate **S**ystem for knowledge **B**ase (USB) with an update frequency predictor of entities as the core component. We also design a set of effective features and realize the predictor. We conduct extensive experiments to justify the effectiveness of the proposed system, model, as well as the underlying principles. Finally, we deploy USB on a Chinese knowledge base to improve its freshness.

1 Introduction

Knowledge bases (KBs) are playing an increasingly important role in many real-world applications. Many knowledge bases, such as Freebase [Bollacker *et al.*, 2008] and DBpedia [Auer *et al.*, 2007; Lehmann *et al.*, 2015], are extracted from encyclopedia website such as Wikipedia with many volunteers maintaining unstructured or semistructured knowledge every day. Knowledge bases extracted from encyclopedia websites usually have high *precision* and *coverage*, thus have been widely used in many real applications, such as search intent detection, recommendation and summarization.

*Corresponding author. This paper was supported by National Key Basic Research Program of China under No.2015CB358800, by the National NSFC (No.61472085, U1509213), by Shanghai Municipal Science and Technology Commission foundation key project under No.15JC1400900, by Shanghai Municipal Science and Technology project under No.16511102102, No.16JC1420401.

However, most of these knowledge bases tend to be outdated, which limits their utility. For example, in many knowledge bases, Donald Trump is only a business man even after the inauguration. Obviously, it is important to let machines know that Donald Trump is the president of the United States so that they can understand that the topic of an article mentioning Donald Trump is probably related to politics. Moreover, new entities are continuously emerging and most of them are popular, such as iphone 8. However, it is hard for a knowledge base to cover these entities in time even if the encyclopedia websites have already covered them.

One way to keep the *freshness* of the knowledge base is synchronizing the knowledge base with the data source from which the knowledge base is built. For example, encyclopedia websites such as Wikipedia are obviously good sources for update. Not only because many knowledge bases are extracted from these encyclopedia websites, but also due to their high-quality and updated information since many volunteers are manually updating existing articles or adding articles for new entities.

Thus, our problem becomes *how should the knowledge base be kept synchronized with online encyclopedia*.

In the best situation, the encyclopedia website may provide the access (such as live feed) to recent changes made in the website. The DBpedia live extraction systems [Hellmann *et al.*, 2009; Morsey *et al.*, 2012] rely on this access mechanism. However, most of the encyclopedia websites do not provide such access. In this case, a direct update solution is revisiting the whole encyclopedia *periodically* and rerun the entire pipeline of the construction of knowledge base like most existing methods. There are two alternative ways to access the encyclopedia articles. The first one is directly downloading the dump file provided by the encyclopedia websites, such as Wikipedia. However, the dump is usually updated monthly, which can not satisfy the real requirement about the knowledge freshness. Moreover, only few encyclopedia websites provide dump downloading. The second way is crawling all the articles in encyclopedia. But crawling the entire online encyclopedia with tens of millions of articles usually needs more than one month with a single machine. No matter which way we choose, we have to download data of GB size, which consumes too much network bandwidth. What's worse, the encyclopedia websites might ban the crawling if the access is too frequently. As a result, time delay of the update is still

significant, leading to the obsolescence of knowledge.

Above solution is not only wasteful but also unnecessary. We notice that most entities have stable properties and are seldom changed, such as basic concepts like orange, or historical persons like Newton. In contrast, some very hot entities are subject to change, such as Donald Trump. Distinguishing the entities subject to change (hot entities) from others with stable properties and then only updating hot entities is clearly a smarter strategy, which not only saves network bandwidth but also shortens the time delay. Thus, the key will be *how to estimate the update frequency of an entity in an encyclopedia website*.

In this paper, we systematically study the update frequency of entities in encyclopedia. We find that only few of them follow Poisson distribution. Thus, we can only use the *change rate* (λ) of Poisson distribution to estimate the update frequency for quite few entities. This weakness motivates us to build a more effective predictor of update frequency that employs not only the historical update information but also the semantic information in the article page of the entity. Based on the predictor, we build a KB update system USB (**U**pdate **S**ystem for Knowledge **B**ase), whose effectiveness is justified in our experiments. The update system was further deployed on our knowledge base *CN-DBpedia*[1], which is created from the largest Chinese encyclopedia BaiduBaike[2]. The system updates 1K entities per day, and about 70% of them actually contain newly updated facts.

2 Framework

In this section, we present the basic framework of USB.

2.1 Problem Model and Solution Framework

Problem model Since the network resources are always limited and some encyclopedia websites have restricted access, we assume that there is an upper limit (K) on the number of entities we can access every day. In other words, a good update strategy should maximize the number of crawled entities with a newer version than which in our knowledge base with at most K entities to crawl. The formal objective is:

$$\arg \max_{R,|R|\leq K} |\{x|x \in R, t_n(x) > t_s(x)\}| \qquad (1)$$

where R is the entity set that we crawl, $t_n(x)$ is the last update time of x in online encyclopedia, and $t_s(x)$ is the last synchronization time of x. When x is a new entity that is not in the current KB, $t_s(x) = -\infty$.

A baseline solution Recall that the key of a smart update strategy is predicting whether an entity has been updated since its last synchronization. Assume that we have already built an effective predictor; the naive solution is running the predictor for each entity in the knowledge base. If the predictor says yes, we update the entities by re-crawling them. However, this naive solution still has two weaknesses:

- First, there are too many entities (tens of millions) in the knowledge base. Running the predictor for all of them is still time-consuming.
- Second, this method can only update the existing entities in KB, and it misses new entities that are not in KB.

To overcome weaknesses above, we highlight that many entities have stable properties and running predictor on these entities is wasteful. In contrast, the entities with changed facts actually are not too many. We find that hot entities mentioned on Web (in hot news or hot topics) are good candidates, because hot entities on Web in general are either new emerging entities or old entities that have a large chance to change. This rationale motivates us to use hot entities on Web as a seed set to start the updating procedure. This strategy is not good enough since the number of hot entities on Web per day is usually small. Hence, we further propose *entity expansion* to improve the recall.

Solution framework Based on the ideas above, we develop a framework to update a KB, namely USB, which is illustrated in Algorithm 1. USB mainly consists of four steps:

1. **Seed finding.** We find hot entities from Web.
2. **Seed synchronizing.** We visit/crawl the latest encyclopedia page for each seed entity, and synchronize (update or insert) the information in online encyclopedia with the knowledge bases via our knowledge extractor from online encyclopedia.
3. **Entity expanding.** We use the newest pages of the synchronized entity to find more related entities via the hyperlinks in the encyclopedia pages.
4. **Expanded entities synchronizing.** For the expanded entities, we synchronize the entities according to their priority. The priority is provided by a predictor of update frequency.

Algorithm 1 USB: Update System for KB

Input: K: upper limits of accesses to the online encyclopedia
Func: VISIT(x): access x's page in the online encyclopedia, can be called at most K times
Func: EXTRACT_SEED($hots$): extract entities from the sentence set $hots$
Func: EXPAND(x): return all hyperlinked entities of x

// Step 1: Seed finding
$hots \leftarrow$ crawl hot sentences/phrases from Web
$seeds \leftarrow$ EXTRACT_SEED($hots$)
// Step 2 & 3: Seed synchronizing and seed expanding
$pq \leftarrow \emptyset$ *//pq* is a priority queue
for $s \in seeds$ **do**
  $page(s) \leftarrow$ VISIT(s)
  if $page(s)$ exists in online encyclopedia **then**
    update s in KB with $page(s)$
    for x in EXPAND(s) **do**
      insert x into pq with priority value as $\mathbf{E}[u(x)]$
    end for
  end if
end for
// Step 3 & 4: Entity expanding and synchronizing
while access limit K is not reached **AND** pq is not empty **do**
  $s \leftarrow pq.pop()$ *// pop the entity with the largest value*
  $page \leftarrow$ VISIT(s)
  update s in KB with $page$
  for x in EXPAND(s) **do**
    insert x into pq with value $\mathbf{E}[u(x)]$
  end for
end while

[1] http://kw.fudan.edu.cn/cndbpedia/
[2] https://baike.baidu.com/

In the algorithm, entity expansion (Step 3) and the synchronization of expanded entities (Step 4) are repeated until the upper access limit is reached or the queue is empty (line 16). Since Step 2 is already clear, in the following text, we mainly elaborate the other three steps.

2.2 Seed Finding

In the first step, we use hot entities on Web as the seeds, motivated by the following principle:

Principle 1 *If an entity appears frequently on Webs (search engines, online communities or online news), its facts are likely to change.*

Specifically, we first collect the sentences or phrases from (1) hot news titles, (2) top search keywords of search engines, and (3) hot topics of online communities (line 2 in Algorithm 1). Then we extract entity names from these sentences and phrases (line 3 in Algorithm 1). We might directly run traditional named entity recognition (NER) methods to identify entities. However, the state-of-the-art NER methods, have less than 90% recall [Finkel *et al.*, 2005; Lample *et al.*, 2016]. In our case, recall is the key issue since we don't want to miss any new emerging entities. To achieve a high recall, we exhaustively enumerate all word segmentations in the sentences or phrases; then search with the word segmentations in the encyclopedia website to find the corresponding entity. For example, from the news title: *"Chinese comedian Zhou Libo was arrested"*, we get the segmentation Zhou Libo, which is an entity name. Another segmentation such as Libo was will also be searched, although it has no corresponding entity. Note that the exhaustive enumeration cost is acceptable since the total number of hot sentences/phrases is not large (each source only provides several tens of hot sentences/phrases per day), and we filter out trivial segmentations by a stop-word list.

Next we conduct an empirical study to justify the effectiveness of our seeding strategy. We find 86 seed entities from a hot query list of a search engine (Sogou search[3]) and a hot topic list of a BBS (Baidu Tieba[4]). As a comparison, we randomly sample the same number of entities from our KB. Then we compute the number of days (as d) since the last update in the encyclopedia website for each sampled entity. We show the CDF (cumulative distribution function) of d for the two sample sets in Figure 1. It can be seen that the CDF curve of seed entities grows faster than that of random entities. In the seed entity set, more than a half of them changed recently (in one month). In contrast, there are only three entities in the 86 random samples changed recently. Therefore, our strategy is effective to find entities recently changed.

2.3 Entity Expanding

In general, the seed entities have a high precision but the recall still needs to be improved. We further propose an entity expansion step to find more recently updated entities. The expansion is based on Principle 2. To see the rationality of the principle, consider Donald Trump's wife. She becomes the

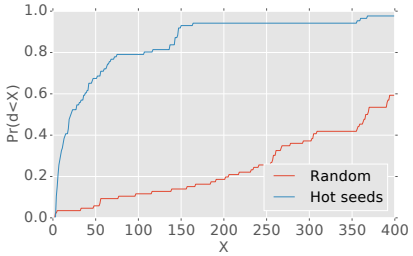

Figure 1: The CDF of seed entities and random entities. Most seed entities have a small d (#days since its last update), implying that they are recently updated. Our seeding strategy is effective to find recently updated entities.

First Lady after Donald Trump becomes American president. Although her name occurs much less frequently on Web than Donald Trump, the facts about her are likely to change.

Principle 2 *An entity that is semantically related to a recently updated entity is also likely to be updated recently.*

The inner hyperlinks in Wikipedia or similar websites have been widely used for finding semantically related entities. Hence, we use the hyperlinked entities in the encyclopedia articles of each seed entity as the expanded entity set. The expansion procedure is iteratively executed along all the entities synchronized (see line 20-22 in Algorithm 1).

2.4 Expanded Entities Synchronizing

After getting the expanded entities, each expanded entity either appears or not in the knowledge base. Any time in the expansion when we find an entity that is not in the knowledge base, we crawl and add it into the knowledge base, as stated in Principle 3. The reason is not only completing the KB with a new entity but also ensuring that there are no *dead links*. Note that the new entity might be referenced by other entities. The immediate update strategy for new entities will not pose a great burden because in general the new entities of the encyclopedia website are few. For example, there are 800 new articles per day in Wikipedia[5]. If the expanded entity is already in the knowledge base, we rely on a predictor to determine whether the entity needs to be updated.

Principle 3 *Any entity that is not in the current knowledge base has the highest priority to be synchronized.*

We continue our experimental study to justify the predictor based update strategy. From the 86 hot seeds, we get 687 expanded entities along the hyperlinks. We find that 269 of them (about 40%) are updated in the last month. The ratio is statistically significantly (with a z-score 18.03) larger than random samples (less than 3%). These results imply that (1) the expansion strategy is effective to find entities recently changed; (2) synchronizing all the expanded entities is still wasteful because the percentage (40%) is not high enough, thus requiring an oracle (predictor) for selection.

Given the network resource budget, we hope to synchronize the entities that have the largest *expected update times after the last synchronization* (denoted by $\mathbf{E}[u(x)]$, where $u(x)$

[3]http://top.sogou.com/home.html

[4]http://tieba.baidu.com/hottopic/browse/topicList?res_type=1

[5]https://en.wikipedia.org/wiki/Wikipedia:Statistics

is the update times after the last synchronization of entity x) as earlier as possible. The principle we used here is:

Principle 4 *The entity that has large expected update times in encyclopedia websites after its last synchronization, deserves a high priority to be synchronized under the network access budget constraint.*

Next, we show how to evaluate $\mathbf{E}[u(x)]$. It is obvious that

$$\mathbf{E}[u(x)] = P(x) \times (t_{now} - t_s(x)) \quad (2)$$

where $P(x)$ is the expected update frequency produced by our predictor (See the next section), and $t_s(x)$ is the last synchronization time of x. Moreover, we define $t_s(x)$ as $-\infty$ if x is not in our knowledge base. As a result, a new entity will always have the highest priority to be synchronized, which reflects Principle 3. We use $\mathbf{E}[u(x)]$ as the priority to insert an entity into the synchronization queue, and stop the KB updating procedure if the network resource budget runs out.

3 Update Frequency Predictor

In this section, we present our solution for the update frequency prediction (i.e., estimating $P(x)$ in Equation 2). We first present a baseline solution based on Poisson distribution assumption about the update frequency of an entity. However, the predictor still has many weaknesses, which motivates us to propose a learning model, which employs more information than the baseline.

3.1 A Baseline Predictor

The first idea comes to our mind is that *whether we can use the historical update information to predict when an incoming update will come*. Many encyclopedia websites, such as Wikipedia, BaiduBaike, have a complete update history, making the exploration of the question above possible. If the answer to the question is yes, it means that the *change rate* (or *update frequency*) is independent of the time window we select, i.e. the change rate, or simply rate, is a constant. If it is the case, we can assume that *the number of updates occur in an interval follows Poisson distribution* (See Assumption 1).

Assumption 1 *The number of updates in an interval (denoted by N) for an entity in encyclopedia websites follows the Poisson distribution.*

$$P(N = k) = \frac{\lambda^k e^{-\lambda}}{k!} \quad (3)$$

Based on the Poisson distribution assumption, the rate λ characterizes the update frequency. Thus, our problem will be how to estimate λ. It was shown that the *total number of updates* divided by *time period* is an effective estimator of λ [Cho and Garcia-Molina, 2003; Umbrich *et al.*, 2010]. This allows us to use the historical update information to estimate λ. More specifically, in our settings, we estimate λ as *the total number of updates* $C(x)$ in the encyclopedia website, divided by the number of weeks since its creation.

$$\hat{\lambda}(x) = \frac{C(x)}{t_{now} - t_{create}(x)} \quad (4)$$

$\hat{\lambda}(x)$ can be used as the expected update frequency $P(x)$ in Equation 2 and helps us finding the entity that has a large expected number of updates.

Time unit	Amount	Confidence	Test passed	Proportion
Month	94873	0.050	4253	4.48%
Month	94873	0.010	6182	6.52%
Month	94873	0.005	6796	7.16%
Week	94873	0.050	313	0.33%
Week	94873	0.010	436	0.46%
Week	94873	0.005	490	0.52%

Table 1: The result of Poisson distribution test. Only few of entities obey Poisson distribution.

Weakness of the baseline However, the baseline above still has weaknesses. First, some new entities have a short history. Thus, the historical update based estimation is inaccurate. Second, the update frequency of an entity might change with time. Thus the update frequency does not necessarily obey Poisson distribution for many entities. For example, Donald Trump is not so popular before his presidential campaign. The update frequency when the election result is announced is certainly larger than before and afterwards.

We further conduct an empirical study to test whether the update frequency follows a Poisson distribution. Specifically, we use K-S (Kolmogorov-Smirnov) test [Lilliefors, 1967] for this purpose. We set the interval with both month and week. The K-S test results on randomly selected 94,873 entities of BaiduBaike under three confidence levels (0.05, 0.01, 0.005) are shown in Table 1. We can see that less than 8% of the encyclopedia entities obey Poisson distribution.

Hence, Poisson distribution based assumption does not necessarily hold for an entity. We still need more features to build a more effective predictor for the update frequency.

3.2 Supervised Update Frequency Predictor

Next, we present some more effective features for the prediction of the update frequency. These features are further fed into a supervised regressor, which predicts the update frequency according to these features. In order to train the regressor, we need labeled dataset and effective features.

Labeled dataset and models We first need the labeled data to supervise the learning of our model. Since we have the entire changing logs for lots of entities in the encyclopedia website and the objective of our model is the "future" update frequency, we use a labeling method similar to time series prediction problems. We first collect the whole changing logs for 94k entities in BaiduBaike. Then we select a time stamp T (in our study, we set T as one month before now). And we use the snapshots of the entities at T to generate the features. Moreover, the response variable $y(e)$ is the average of weekly update frequency after T of entity e. Thus, we construct a sample as $< \langle x_1(e), x_2(e), ...x_k(e)\rangle, y(e) >$, where each $x_i(e)$ is one of the feature about e, which will be elaborated in the next subsection. Finally, the training samples are fed into two regressor models: linear regression [Kutner *et al.*, 2004] and random forest regression [Liaw and Wiener, 2002]. A regressor is obviously more advantageous than a binary classifier since a regressor can be trivially transferred into a binary classifier by a certain threshold. A regressor is also necessary since we need to sort the candidate entities by the predicted update frequency. We use the implementations

in scikit-learn [Pedregosa *et al.*, 2011]. Specifically, for the linear model, we use ridge linear regression, which uses linear least squares loss and l2 regularization.

Feature engineering For each entity, we extract features from its snapshot at time T. We highlight that all the following metrics are calculated from the data before time T. The features are listed below:

1. **#Weeks of existence.** This is used to quantify how long the entity exists in the encyclopedia website. Intuitively, a new entity has a larger chance to be updated.
2. **#Total updates.** This quantifies how many times the entity has been updated. A historically frequently updated entity in some cases tends to be updated in the future as discussed in the baseline method.
3. **#Times viewed by users.** The encyclopedia website contains the times that an entity is viewed. Intuitively, a popularly viewed entity has a large probability to be updated.
4. **#All hyperlinks.** The encyclopedia article of an entity usually contains hyperlinks to external sources or other entities in encyclopedia. The more such hyperlinks are, the more possibly the entity is influenced by an external source or other entities.
5. **#Hyperlinks to entities.** The hyperlinks to other entities in encyclopedia deserve an individual metric. Because any change in one of hyperlinked entities in encyclopedia website might propagate to the entity.
6. **Page length.** It is the length of the article page about an entity, except the ADs and banners. A longer article usually has a large chance to be updated since it contains rich content.
7. **Main content length.** It is the length of the main text of an entity in encyclopedia.
8. **Historical update frequency.** It is the weekly average update frequency. The rationale has been discussed in the baseline method.

In addition, for the 2-7 features, we use $g(x) = log(1 + x)$ to normalize them.

4 Experiments

In this section, we conduct extensive experiments[6] to show the effectiveness of our update frequency predictor and our updating framework.

Exp 1: Effectiveness of our features At first, we show the effectiveness of our features. We use the labeled dataset proposed in Section 3.2 to train a binary classifier that predicts whether an entity will be changed in one month after T or not. In the classifier version, the label is true when $y(e) > 0$, and the label is false when $y(e) = 0$. We fed the model with the same features as the regressor. The classifier models allow us to compute χ^2 and IG (information gain) to evaluate the effectiveness of each feature. The result is in Table 2.

The result shows that our features are effective to distinguish the recently updated entities from other entities. In general, the number of updates either the total version or divided by interval is among the most effective features. However, many other features, such as the number of hyperlinks, show better effectiveness than historical update frequency in χ^2. These results suggest that other features also contribute to the identification of recently updated entities.

[6]The code and data are available at http://kw.fudan.edu.cn/resources/data/bdupd.zip

#	Feature	χ^2	IG(10^{-3})
1	#Weeks of existence	41.8	19.1
2	#Total updates	**481.1**	**55.9**
3	#Times viewed by users	203.5	46.2
4	#All hyperlinks	460.9	35.8
5	#Hyperlinks to entities	444.9	32.1
6	Page length	131.9	32.9
7	Main content length	202.1	19.1
8	Historical update frequency	287.6	54.7

Table 2: χ^2 and IG of the features

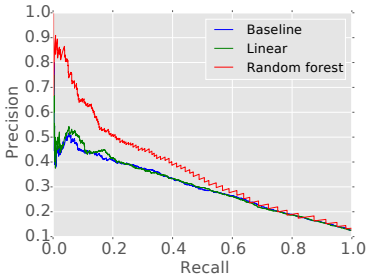

Figure 2: PR curves of hold-out test

Exp2: Effectiveness of the predictor: hold-out evaluation In this experiment, we evaluate the effectiveness of our predictor of update frequency on the labeled dataset. We randomly split our labeled 94K entities into training set (90%) and test set (10%). We report the MSE (mean square error) between the predicted update frequency and the real update frequency for the test entities. Moreover, by using different thresholds for predicting whether each test entity is updated after time T, we get a precision-recall curve, and we report the AUC (area under the curve) as the measures of the regressor's goodness. The results are shown in Figure 2 and Table 3. The random forest model has the best result. Our models outperform the baseline, which justifies that historical update based features together with other features perform better than models with only historical update based features.

Exp3: Effectiveness of the predictor: real dataset We use this experiment to test how our regressors perform in real-world applications. We first make a snapshot of our knowledge base on Dec 15, 2016. At the same time, we got 86 hot seeds from Web. From the seed, we get 687 expanded entities. We run our regressor on the expanded entities, and then rank all the expanded entities in terms of their predicted update frequency. We further collect the changing logs between Dec 16, 2016 and Jan 15, 2017 of these expanded entities, and we find that 269 of them are updated in this month.

Model	MSE	AUC
Baseline	0.0400	0.2992
Linear	0.0367	0.3021
Random forest	**0.0315**	**0.3692**

Table 3: Comparison of models, for hold-out test

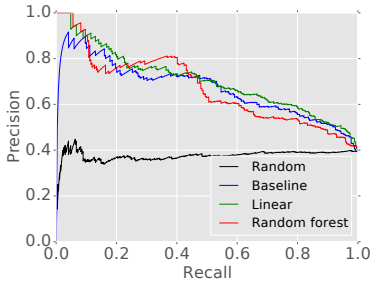

Figure 3: PR curves of expanded entities

Metric	Random	Baseline	Linear	RF
MAP	0.379	0.670	**0.704**	0.671
nDCG	0.800	0.916	**0.941**	0.933
AUC	0.376	0.666	**0.700**	0.667
Precision@20	0.400	0.850	0.900	**0.950**
Recall@20	0.030	0.063	0.067	**0.071**
F1@20	0.055	0.118	0.125	**0.131**
Precision@100	0.350	0.730	0.750	**0.790**
Recall@100	0.130	0.271	0.279	**0.294**
F1@100	0.190	0.396	0.407	**0.428**

Table 4: Comparison of models, for expanded entities

Thus, we can use ranking metrics, such as MAP, nDCG, etc, to check whether our regressor can rank these 269 updated entities higher. We compare the ranks produced by our regressor (linear regression and random forest regression) to random shuffling and ranking by historical update frequency. The results are shown in Figure 3 and Table 4.

The result shows that, our two models generate better rankings than the random ordering and the historical frequency based ordering. Moreover, we can see that random forest model in general has a better performance when we are only interested in the top sublist. In contrast, the linear model has a better performance when the entire ranked list is considered. In real applications, we prefer random forest model, since the upper access limit forces us to drop tailed entities.

Exp4: Performance of our system: USB In this experiment, we report the performance of our system USB. We deployed the system on our knowledge base, a DBpedia-like Chinese knowledge base extracted from BaiduBaike. We set K (upper access limit) as 1000 so that the crawling will not be banned by the website. We report the proportion of web visits that find a newer version of entities to all web visits. The result is shown in Table 5.

The result shows that, our system is effective to find the entities that have to be synchronized. Most of our visits crawl entities that have a newer version. Hence, our system is effective to use the network resource budgets. By setting a reasonable upper access limit, we can keep most of our KB up-to-date, without incurring overloads on the encyclopedia site.

Total visits	Success updates	Success ratio
50	46	92.0%
100	90	90.0%
200	175	87.5%
500	398	79.6%
1000	687	68.7%

Table 5: Real-world update experiment

5 Related Work

Recently, knowledge bases have attracted wide research interests. Many knowledge bases are automatically constructed. Specifically, most of them use encyclopedia websites such as Wikipedia as main data sources, such as DBpedia [Auer *et al.*, 2007; Lehmann *et al.*, 2015], WikiTaxonomy [Ponzetto and Strube, 2008], and Freebase [Bollacker *et al.*, 2008].

However, even if there are many volunteers maintaining the unstructured or semistructured data in the Wikipedia every day, most of these KBs are fixed in only one version, and the knowledge in them is outdated. For example, Freebase is stopped for maintenance. Although they can build a new version with their constructing methods form the latest Wikipedia dump, as we argued that, it is time-consuming [Brewington and Cybenko, 2000], and wasteful.

There are some works focusing on the update of KBs, but most of them have a severe limitation. That is, the encyclopedia website should provide the access to recent changes, as an update stream. For example, the Wikipedia foundation kindly provided the Wikipedia OAI-PMH7[7] live feed. Based on these information, [Hellmann *et al.*, 2009] creates a live extraction framework, and provides a mechanism that allows the Wikipedia community to maintain the DBpedia ontology collaboratively. [Morsey *et al.*, 2012] provides a newer version, which adds new features, e.g. abstract extraction, ontology changes, and changesets publication. However, most of encyclopedia websites do not provide the update stream.

There are also many previous literatures that investigated the statistical update behavior of online data, such as web page [Brewington and Cybenko, 2000; Cho and Garcia-Molina, 2003] and linked open dataset [Umbrich *et al.*, 2010]. Among these researches, it is widely assumed that the change frequency of online web data obeys Poisson distribution [Brewington and Cybenko, 2000; Cho and Garcia-Molina, 2003]. In this paper, we use this assumption to develop a baseline predictor, which unfortunately is still not good enough yet to help us find the entity in an encyclopedia that recently changed.

6 Conclusion

In this paper, we focus on the knowledge bases using encyclopedia websites as the data source. Most of these KBs tend to be outdated, and simply rebuilding the whole KB is wasteful and incurs massive network overload. To overcome the weakness, we propose a set of synchronization principles upon which we build a KB update system with an update frequency predictor of entities as the core component. We also realize the predictor with a set of effective features. Moreover, we conduct extensive experiments to justify the effectiveness of the proposed system, model, as well as the underlying principles. Finally, we deploy the system on a Chinese knowledge base to improve the freshness of the KB. The system updates 1K entities per day, and about 70% of them actually contain newly updated facts.

[7] http://www.mediawiki.org/wiki/Extension:OAIRepository

References

[Auer *et al.*, 2007] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[Bollacker *et al.*, 2008] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[Brewington and Cybenko, 2000] Brian E Brewington and George Cybenko. Keeping up with the changing web. *Computer*, 33(5):52–58, 2000.

[Cho and Garcia-Molina, 2003] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology (TOIT)*, 3(3):256–290, 2003.

[Finkel *et al.*, 2005] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[Hellmann *et al.*, 2009] Sebastian Hellmann, Claus Stadler, Jens Lehmann, and Sören Auer. Dbpedia live extraction. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1209–1223. Springer, 2009.

[Kutner *et al.*, 2004] Michael H Kutner, Chris Nachtsheim, and John Neter. *Applied linear regression models*. McGraw-Hill/Irwin, 2004.

[Lample *et al.*, 2016] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[Lehmann *et al.*, 2015] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[Liaw and Wiener, 2002] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[Lilliefors, 1967] Hubert W Lilliefors. On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American statistical Association*, 62(318):399–402, 1967.

[Morsey *et al.*, 2012] Mohamed Morsey, Jens Lehmann, Sören Auer, Claus Stadler, and Sebastian Hellmann. Dbpedia and the live extraction of structured data from wikipedia. *Program*, 46(2):157–181, 2012.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Ponzetto and Strube, 2008] Simone Paolo Ponzetto and Michael Strube. Wikitaxonomy: A large scale knowledge resource. In *ECAI*, volume 178, pages 751–752. Citeseer, 2008.

[Umbrich *et al.*, 2010] Jürgen Umbrich, Stefan Decker, Michael Hausenblas, Axel Polleres, and Aidan Hogan. Towards dataset dynamics: Change frequency of linked open data sources. In *3rd International Workshop on Linked Data on the Web (LDOW2010) at WWW2010*. CEUR, 2010.