Ranking RDF Properties Using SPARK Framework

Akmal Khikmatullaev, Seyithan Dag

February 27, 2018

Abstract

This report summarizes the work done for the Distributed Big Data Analytics Lab course offered at the University of Bonn during winter semester 2017-18. The main goal of the implementation is to apply currently existing machine learning methods to rank properties of RDF data, which are available in massive amounts on the web with the help of semantic data representation. In order to familiarize the reader with the methods, structures and algorithms involved in our work, we firstly go over the concepts. Then we move on to giving information about the guiding literature followed for the implementation, clearly stating the adopted/implemented parts. After that, the implementation phase is explained. Then we continue by presenting and discussing the results and conclude our report.

Contents

1 Introduction

Semantic web is the way of sharing information on the web, where each piece of data is stored in a structure that can not only be read by machines themselves, but also humans. As the internet evolves every day, so does the amount of data contribution, and with every step making this happen, the importance of how to store the data and how to make the most out of it gains importance. This is the point where the W3C standard of data storage comes into play. With this standard, each piece of information is stored in RDF triples that contain a subject, an object and a predicate in between defining in what way the object is related to the subject (or vice-versa). With the help of SPARQL, which is an RDF query language for databases, these triples can be easily obtained from endpoints. A very well-known and popular one of these endpoints is DBpedia. DBpedia is basically an engine, where the data belonging to Wikipedia is extracted and stored as RDF triples, so as to make it available for the semantic web. Since every day the semantic web grows and gains importance, the statistical information extraction on the very RDF data it has becomes more valuable. As there are quintillion RDF triples that already exist, not all information they contain would be significant for a given querier at any time, where this querier could be a person or an application. Therefore, to get rid of this insignificance, labelling (or inferring) the available data in a way where they would be much more significant according to the intention of the querier is a promising way of elevating the usefulness of the semantic web.

One of the many labellings that try to get rid of query based irrelevance and insignificance, ranking of RDF properties also comes forward as a promising method. There are currently several applications tackling this problem, however, as with many other problems already existing in computer science, there exist multifarious ways of solving this very problem. Motivated by the intricacy and scalability of the problem, this report will be introducing a method of RDF property ranking, that is distributed and able to accommodate millions of entries. Throughout the report, the term predicate and property is used in an interchangeable manner. The collection of technologies used includes Apache SPARK and the MLLib library on it, DBpedia, Scala and Python.

The organization of the report starts with exploring the guiding literature, i.e., the article used. Then the details of the implementation, to which this report belongs, will be explained in detail, where the parts adopted from the article will be talked on. The implementation section will give information about the whole process, from collecting the data to processing that data and to the algorithms used. Finally, the results of the implementation will be discussed and then the report will be concluded.

2 Adoption of the Guiding Literature

The guiding literature used for the implementation is the work of Dessi and Atzori [1] titled as "A machine-learning approach to ranking RDF properties".

In this paper, the authors begin by explaining the importance of RDF ranking for the semantic web and mention the usage areas of it, while also exploring the already existing solutions. Then they move on to explaining their proposed method.

The summary of the work flow of the proposed method and the steps thereof is as follows:

1. Machine learning algorithm selection.

   The authors make use of the already available library called RankLib, that consists of following algorithms, all of which are specialized in learning to rank:

   - RankNet
   - RankBoost
   - AdaRank
   - Coordinate ascend
   - LambdaMART
   - MART
   - ListNet
   - Random forests

   The details of what these algorithms consist of will not be explored, as this is not within the scope of this report.

2. Feature creation phase.

   Since many of the concepts from this section were adopted, each suggested feature will be explained in detail. The adopted features that were used in the lab project are marked with an asterisk. The authors come up with the following features:

   - Frequency*: As the name suggests, the frequency feature measure the number of occurrence of a predicate, given other predicates belonging to the same domain.
   - NumberOfWords*: This feature is also straight forward and it basically counts the words in a predicate.
   - ContainsNumber*: This feature checks whether a given predicate has numbers in it.
   - IsEnglish*: This feature checks if the property name is English. The authors make use of the NLTK library, however, in the project, a modified version of this feature was implemented. That is, the characters are checked whether they belong to the English alphabet. This clearly narrows down the domain, though the effect thereof is not vital.

- OntOrRaw*: This feature determines whether the predicate has the ontology prefix or not.
- AnswIsLink*: This feature decides whether the object the predicate points to is a link or a literal.
- SuggestChars: This feature is the most complex one to implement and it determines how many chars of the predicate are needed to be entered to the DuckDuckGo search bar before the predicate comes up in the list of the autosuggests of the DuckDuckGo. The authors mention implementing their own API for simulating the entry of strings to the DuckDuckGo search bar, which the very reason why this feature was omitted in the project, as it is too complex and out of the focus of the project implementation.
- SWIPE: Another feature that exploits a different implemented system. SWIPE system is a search engine that is used for searching Wikipedia through examples. It also bares a suggest function and the authors make use of it, so as to see if the predicate is among the most popular properties according to the SWIPE. As this feature is also too complex, it was omitted.

3. Training set production.

   The authors come up with six ways of training set production/creation. In the implementation of the project, one of these was adopted, which, as previously, is marked with an asterisk. Short explanations of the methods used are as follows:

   - Expert-based training: Authors ask a semantic web expert to rank predicates.
   - Questionnaire-based training: Authors ask 5 CS graduate students to fill a questionnaire, and then the results of this is used as the basis of ranking. Figure 1 shows an example result of the questionnaire conducted by the authors about the domain Russelia.
   - Frequency-based training*: This method uses the frequency features obtained. Predicates are sorted according to their frequencies, and their position in the sorted list is used as a rank. In project implementation, this method was adopted.
   - Suggest training: This method uses SuggestChar feature.
   - Typicality based training: Authors calculate the typicality measures for each predicate, and use it as a feature. A typical typicality measure tells how typical an ontology is, given the attribute; how typical attribute is, given the instance; how typical instance is, given the attribute.
   - SWIPE-based training: This method uses SWIPE feature.

After this point on the authors move onto the mixing and matching every possible training set with every algorithm. However, this will not be talked on in this report.

Property	1 user	2 user	3 user	4 user	5 user
familia	4	4	2	4	4
ordo	3	3	3	3	4
tribus	4	4	3	4	4
genus	3	3	3	4	3
genus authority	1	3	2	4	3
subdivision ranks	4	3	3	4	4
unranked classis	1	1	2	3	1
unranked divisio	3	3	3	3	3
unranked ordo	3	3	3	3	3
regnum	1	3	3	4	4
Name	4	2	2	4	3
subdivision	2	3	3	4	4
image caption	1	3	2	4	1
family	4	4	3	4	4
Subject	1	3	2	4	3
class	1	4	3	4	3
kingdom	1	3	3	4	4
order	1	3	2	4	4
thumbnail	4	1	2	3	2
division	2	1	2	4	2
has abstract	1	2	2	3	4
Wikipage page ID	1	2	2	1	1
Wikipage revision ID	1	1	1	2	1
Link from a Wikipage to an external page	1	1	1	1	2

Figure 1: Results of questionnaire conducted by the authors for the domain (sphere) Russelia.

3 Implementation

In this section, the details of the implementation will be presented. This will include gathering data, preprocessing, feature extraction, algorithms and training.

3.1 Gathering Data

In order to narrow down the possibilities, three domains were chosen and worked on throughout the implementation. There are namely Language, Country and City. The data gathering process took a total of approximately **3 days**. Python was used as the interface for executing SPARQL queries. Following snippet shows an example SPARQL query execution used within Python:

Entry (RDF triple) and Distinct Predicate Amounts		
Domain	Number of Triples	Number of distinct predicates
Language	921,715	1,242
Country	1,178,959	2,545
City	5,000,000	3,852

Table 1: Gathered data.

```
spheres = ['City']

query = """
prefix dbpedia-owl: <http://dbpedia.org/ontology/>

select ?entry ?p ?o {{{{
select ?entry ?p ?o{{
?entry a dbpedia-owl:{sphere}.
?entry ?p ?o
}}
order by ?entry
}}}}

limit {limit}
offset {offset}
"""

total = 500000
for sphere in spheres:
i = 0
with io.open(data_path + "\\{}.txt".format(sphere.lower()), "w",
    encoding="utf-8") as f:
while i < total:
fetch = query.format(sphere = sphere, limit = str(10000), offset =
    str(i))
data = data_graph.query(fetch)
object = None
for entry in data:
if 'http://' in entry[2]:
object = 'link'
else:
object = 'literal'
rdf = ('<' + str(entry[0]) + '>' + ' '
+ '<' + str(entry[1]) + '>' + ' '
+ '<' + str(object) + '>' + '\n')
f.writelines(rdf)
i = i + 10001
print(sphere, i)
f.close()
```

Listing 1: Snippet of an example SPARQL query run within Python.

For each of the domains, the maximum number of entries (RDF triples) to be retrieved was set to be 5 million. However, this turned out not to be the case, as not all of the domains -apparently- have 5 million entries.

3.2 Feature Extraction

As mentioned in the previous section, the features extracted for each of the distinct predicates were *Frequency, NumberOfWords, ContainsNumber, IsEnglish, OntOrRaw*, and *AnswIsLink*. It should be noted that each feature was extracted using the predicate's own domain.

3.3 Algorithms

- Multinomial Logistic Regression

  This algorithm is the multiclass version of the classic logistic regression. Multinomial logistic regression is used to predict categorical placement in or the probability of category membership on a dependent variable based on multiple independent variables. The independent variables can be either dichotomous (i.e., binary) or continuous (i.e., interval or ratio in scale). Like binary logistic regression, multinomial logistic regression uses maximum likelihood estimation to evaluate the probability of categorical membership.

- Naive Bayes

  Goal: Learning $p(y|x)$

  Bayes formula gives:

  $$p(y|x) = \frac{p(x|y)\,p(y)}{p(x)}$$

  The Naive Bayes method consists in assuming that the features x_i are all conditionally independent from the class, hence:

  $$p(x|y) = \prod_{i=1}^{p} p(x_i|y)$$

  Then, the Bayes formula yields:

  $$p(y|x) = \frac{p(y)\prod_{i=1}^{p} p(x_i|y)}{p(x)} = \frac{p(y)\prod_{i=1}^{p} p(x_i|y)}{\sum_{y'} p(y')\prod_{i=1}^{p} p(x_i|y')}$$

- Decision Tree

  Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data [2].

- Isotonic Regression

  Isotonic regression belongs to the family of regression algorithms. Formally isotonic regression is a problem where given a finite set of real numbers $Y = y_1, y_2, ..., y_n$ representing observed responses and $X = x_1, x_2, ..., x_n$ the unknown response values to be fitted finding a function that minimises

  $$f(x) = \sum w_i(y_i - x_i)^2$$

  with respect to complete order subject to $x_1 \leq x_2 \leq ... \leq x_n$ where w_i are positive weights. The resulting function is called isotonic regression and it is unique. It can be viewed as least squares problem under order restriction. Essentially isotonic regression is a monotonic function best fitting the original data points [3].

3.4 Training

After feature extraction step was completed, the algorithms mentioned in the previous subsection were trained. For all of the training runs, 60 percent of the available data in every domain was used for training, and 40 percent for testing respectively. Since SPARK turns the data into RDDs, this distributivity enabled the algorithms to be trained in a timely fashion.

4 Results and Discussion

Among all of the algorithms trained, Decision Tree proved to be the most useful in terms of accuracy. As expected, isotonic regression was unable to be trained, due to it being capable only in binary classification. Apart from this, multinomial logistic regression turned out to be the least effective. Table 2 presents the accuracy rates of each algorithm on each domain. In addition to the accuracy

Accuracy (%) Rates of Algorithms in Each Domain				
Domain	Multinomial Logistic Regression	Naive Bayes	Decision Tree	Isotonic Regression
Language	0,30	0,66	0,83	0
Country	0,36	0,65	0,85	0
City	0,36	0,70	0,84	0

Table 2: Test accuracy of algorithms on different domains.

results, following are the confusion matrices of each of the algorithms (Multinomial Logistic Regression, Naive Bayes and Decision Tree. Isotonic Regression was omitted, due to incapability) on Language domain as an example:

Language domain confusion matrices:

$$\begin{bmatrix} 81 & 0 & 10 & 22 \\ 80 & 0 & 14 & 39 \\ 70 & 0 & 12 & 39 \\ 63 & 0 & 7 & \end{bmatrix} \begin{bmatrix} 96 & 48 & 0 & 0 \\ 0 & 103 & 34 & 1 \\ 0 & 1 & 70 & 55 \\ 0 & 0 & 15 & 107 \end{bmatrix} \begin{bmatrix} 112 & 2 & 0 & 0 \\ 1 & 120 & 0 & 0 \\ 0 & 11 & 56 & 1 \\ 0 & 0 & 65 & 130 \end{bmatrix}$$

As can clearly be seen, and as previously mentioned, Decision Tree proved to be the most useful. Had the features been more and extracted in a different way, whether would it give the same results or not is another topic. However, juding by the number of data and availability of different domains, it is clear that these are somewhat context-free for Decision Trees. Therefore, it would be advisable to test any future RDF "big data" related machine learning tasks on first on Decision Trees.

5 Conclusion

In this report, the work adopted from [1] has been shown. Ranking of RDF properties is no doubt useful, and the walkthrough of a way implementing this has been shown. The stages of data gathering, preprocessing, future extraction and selected algorithm training and the results there of have been talked on. Since the big data of the web will -from this point on, given the current stage of the internet and technology- never grow any smaller, it is definitely a clever practice to try to come up with more ways of making the readily available (and ever-growing) linked data as meaningful as possible.

6 References

[1] Dessi, A., Atzori, M (2015). A Machine-Learning Approach to Ranking RDF Properties. *Future Generation Computer Systems.* pp 366-377

[2] https://spark.apache.org/docs/2.2.0/mllib-isotonic-regression.html

[3] http://www.saedsayad.com/decision$_t ree.htm$