Semi-Structured Text Document Search using VSM

Project Report

Jinan El Itawi, Yara Aoun, and Amir Amine

Abstract— This project is part of IDPA course second project. It outlines the implementation of semi-structured XML documents similarity computations using Vector Space Model (VSM) similarity measures. The project allows to searching for the relevant documents in the documents database to the query submitted by the user. The query can be either an XML document or a flat text. The relevant documents are returned based on the similarity computations results. Finally, an evaluation of the results as well as a comparison with the TED approach implemented in project 1 are conducted.

Index Terms—XML, VSM, IR, TED, indexing

CONTENTS

1 Introduction

THIS project deals with semi-structured XML similarity-based search tool using Vector Space Model. The project allows the user to input a query which can be either an XML document or a flat text input into search component and it also permits him/her choose the similarity model as well as similarity measure based on the type of query submitted. It then returns the documents (from the documents dataset) ranked based on their similarity results with the query. An indexing part is also integrated into the project in order to speed up the process. The similarity process is also evaluated with and without indexing. Finally, the project allows to compare the time performance and results quality between VSM and TED solution presented in project 1. The project is implemented using python.

2 Procedure

2.1 Preprocessing Stage

2.1.1 Database of XML documents

Our database contained groups of documents, having different themes such as Netflix movies and series, menu of restaurants, details of research papers, and selections of songs. We adopted different themes in order to build differences as well as similarities on both structural and content levels. These XML documents will be preprocessed each into a vector accordingly based on whether we want to consider structure only, content only, or content and structure.

2.1.2 Structure only

In order to consider the structure of an XML document, we found that the most suitable approach was the all paths method. In order to implement this method, we used the lxml elementree library. We first parsed our document into a tree. Then, we iterated through each element of the tree using the tree.iter() method. In each iteration, we would get all the paths possible for each element using the .getparent() method. Once we stored all the paths available of our tree, we created a method that counts the repeated paths in order to compute the TF weights of our vector. Finally, our document would be processed as an array containing arrays where each stores the paths and its TF weight next to it.

2.1.3 Content and Structure

The document is pre-processed using the sklearn.feature_extraction.text library to a term-context vector model. The library tokenizes the content of each path of the document and removes stop-words using a built-in dictionary. The document is then transformed into a dataframe where the columns represent the content, the rows represent the paths of the document, and the values inside the dataframe represent the occurrence of each term (content) inside the path. The size of the dataframe is mxn where m is the number of columns and n is the number of rows,

2.1.3.1 Content and Structure for comparison

The dataframe is the transformed into an array of sub-arrays where each sub-array is in the form of [“*path*”, “*term*”, *occurrence*]. The value *occurrence* is the TF-weight of the term since it represents the number of times the term has occurred inside the given path.

For example, if the word "LAU" occurs inside the path "university/Lebanese" and inside "institution/Lebanese", the term-context vector becomes [["university/Lebanese", "LAU", 1], ["institution/Lebanese", "LAU", 1]].

2.1.3.2 Content and Structure for querying

The dataframe is also transformed to another array of sub-array that is only used when querying. This array is in the form of [["*path **or** term”, occurrence*]]. Since the dimenstion of the dataframe of the query (to be explained in 2.1.5) is mx1 (flat text query), the vector modeled against the query needs to be of the same dimension. Therefore, the dataframe of the document becomes mx1 and the values inside the dataframe represent the count of each term (content) and each tag (structure) inside the whole document.

2.1.4 Content only

After pre-processing the document and transforming it into a term-context vector model, the vector (as produced In 2.1.3.1) is transformed to a term-context vector that represents the occurrence of a term inside the whole document and not inside a single path only. The new vector model is represented in an array of sub-arrays in the form of [["*term*", occurrence]]. For example, using the previous example of the term "LAU" inside two different paths, the new content only term-context vector becomes [["LAU", 2]].

2.1.5 Flat text Query

The flat text query, which is input by the user, is tokenized the same way the document is tokenized in part 2.1.3 and then transformed to a term-context vector model. But instead, the dataframe includes one row only since the query is flat and is not constituted of multiple paths. The dataframe is the transformed to an array of sub-array In the form of [["*term*", *occurrence*]].

The term-context vector model of the Query is compared with the term-context vector model of the document produced as in part 2.1.3.1 when considering content and structure, and as produced in 2.1.4 when considering content only.

2.2 Documents Weighting

//TF IDF TFIDF + their equations

2.3 Similarity Computations

The similarity computations were done based on the TF weights, IDF weights or TF and IDF combined and whether structure or content or both were considered based on the choice done by the user and either using the cosine correlation measure or PCC correlation measure. For cosine correlation measure we chose 0 being the threshold, meaning we ignored the values less than 0. For PCC correlation measure we chose -1 being the threshold, meaning we ignored the values less than -1.

For structure only, the similarity using cosine correlation measure was computed based on the formula below:

$$\mathbf{Sim}_{\mathbf{Cosine}}(\vec{A},\vec{B}) = \frac{\sum_{i=1\ldots N} A_i \times B_i}{\sqrt{\sum_{i=1\ldots N} A_i^2 \times \sum_{i=1\ldots N} B_i^2}}$$

The similarity using PCC correlation measure for structure only was computed based on the formula below:

$$Sim_{PCC}(\vec{A},\vec{B}) = \frac{\sum_{i=1...N}(A_i - \bar{A}) \times (B_i - \bar{B})}{\sqrt{\sum_{i=1...N}(A_i - \bar{A})^2 \times \sum_{i=1...N}(B_i - \bar{B})^2}}$$

For content only, or content and structure, the similarity using cosine correlation measure was computed based on the formula below:

$$Sim_{ECosine}(\vec{V_Q},\vec{V_D}) = \frac{\sum_{(t_i,c_i)\in Q}\sum_{(t_i,c_k)\in D} w_Q(t_i,c_i) \times w_D(t_i,c_k) \times Sim_{ED}(c_i,c_k)}{|\vec{V_Q}| \times |\vec{V_D}|}$$

The similarity using PCC correlation measure for content only, or content and structure was computed based on the formula similar to the cosine formula but by subtracting W_Q and V_Q by the average of Q, subtracting W_D and V_D by the average of D.

2.4 Indexing

The indexing was done by creating a function that takes as input a dictionary having as key the name of the document and as value an array of arrays containing its paths as first element, tf score as second, and IDF as a third element. The method created first computes TF*IDF of each element. Then, for each document the elements with the highest TF*IDF are selected. The number of element selected is the length of the document multiplied by the percentage of precision chosen by the user. For example, if a document has a length of 6 paths, and we choose a precision of 50%, 3 elements having the highest TF*IDF are selected. We then compare the chosen elements of each document with the others where indexing is done using the inverted indexing technique. This indexing technique permitted us to return a dictionary containing the common word or path as a key, and as a value a list of the names of the documents in which they were present.

2.5 GUI

3 RESULTS

3.1 Similarity Measure Evaluation

In order to evaluate the Cosine and PCC similarity measures, we have run our algorithm on different documents and recorded our obtained results. First, Cosine measure returns a similarity value between 0 and 1 having 0 as the minimum similarity (not similar at all) and 1 as the maximum similarity (same document, while the PCC similarity measure returns a similarity value between -1 and 1 with -1 being the minimum similarity and 1 being the maximum similarity. A sample run on a sample document, netflix6.xml (figure 1) from our dataset was performed while applying TF weighting and structural similarity. With our Knn measure discussed previously, the cosine measure returned 25 documents with the netflix xml documents being in the high rank while other documents shared minimal similarity near to 0. The PCC measure, on the other hand, returned all 40 documents with netflix documents having a positive similarity while other documents with negative similarity. This means that documents with negative similarity are irrelevant when comparing using PCC. The negative similarity was mainly due to the nature of PCC equation having to subtract or normalize towards the mean.

```
▼<netflix>
  ▼<movies>
    ▼<movie>
       <title>Die Hard 2</title>
       <producer>Joel Silvel</producer>
      ▼<actors>
         <actor>Bruce Willis</actor>
         <actor>Alan Rickman</actor>
       </actors>
     </movie>
    ▼<movie>
       <title>Mission Impossible 2</title>
       <producer>Bruce Geller</producer>
      ▼<actors>
         <actor>Tom Cruise</actor>
         <actor>Emmanuelle Beart</actor>
       </actors>
     </movie>
   </movies>
 </netflix>
```

Figure 1 - netflix6.xml Document

3.2 Documents Weighting Evaluation

Documents were weighted through different criteria: TF, IDF, and TFIDF. The same sample document (netflix6.xml) was also used to evaluate the accuracy of each weighting with structure only and Cosine similarity measure. When using TF weighting, netflix6.xml had maximum similarity with itself however while evaluating using TFIDF weighting, this similarity decreased to 0.98 with itself. However, using IDF weighting, netflix6.xml had 0.9 similarity with itself even though it is the same documents. All weighting measures returned the netflix documents from the dataset as the top relevant documents yet with different similarity values and different rankings. The IDF weighting produced some lesser similarity values than the TF and TFIDF weighting and sometimes irrelevant ordering.

3.3 Similarity Methods Evaluation

3.3.1 XML Document Query

When uploading and XML document to query upon, the user has the choice between structure only or structure and content. In this part of the report, we will consider the document music8.xml (figure 2) as our reference and we will be fixing the weighting to be TF and the similarity measure to be Cosine measure. When querying using structure only, music8.xml document had maximum similarity with other documents such as music9.xml, music4.xml, and music2.xml since the tags and attributes keys are the same. However, when querying using structure and content, this similarity with the previously mentioned documents decreased to become 0.5, 0.175, and 0.224 respectively while remaining to have maximum similarity with its own self document. The similarity with music9.xml (figure 3) was found to be higher than others due to the fact that this document share similar content as in music8.xml as seen between figures 2 and 3.

```
▼<Song>
   <Title>Wahdi Ana</Title>
   <Release year="1981">Released</Release>
  ▼<Artist type="solo">
     <Name>Melhem Barakat</Name>
     <Origin>Lebanon</Origin>
   </Artist>
   <Genre>Classic</Genre>
   <Subgenre>Tarab</Subgenre>
 </Song>
```

Figure 2 - music8.xml Document

```
▼<Song>
   <Title>Saalouni El Nass</Title>
   <Release year="1973">Released</Release>
  ▼<Artist type="solo">
     <Name>Fairuz</Name>
     <Origin>Lebanon</Origin>
   </Artist>
   <Genre>Classic</Genre>
   <Subgenre>Tarab</Subgenre>
 </Song>
```

Figure 3 - music9.xml Document

3.3.1 XML Flat Text Query

The user has also the choice to enter his query as words or sentences and to evaluate his similarity based on content only or content and tags. The similarity measure is also fixed to be Cosine measure and the weighting is fixed to be TF weighting. If we consider querying the word song which represents the music xml documents in the dataset, nothing is returned while considering content only, while all the music xml documents are returned when considering tags and content as the word song is only mentioned in the tags (figures 2 and 3). Moreover, when querying the word "Joseph", if the choice was content only, a single document (family7.xml) is returned, yet if the choice was content and tags, the same family7.xml is returned as well as for menu7.xml document which is a tag value representing the name of the restaurant (figures 4 and 5). This means that tags may need to be considered in our model if we need more accurate results.

```
▼<family>
  ▼<hanna>
     <mother>Rima</mother>
     <father>George</father>
    ▼<children>
       <daughter>Maria</daughter>
       <son>Maroun</son>
     </children>
    ▼<cousins>
       <cousin>Rami</cousin>
       <cousin>Lama</cousin>
     </cousins>
     <aunt>Jana</aunt>
     <aunt>Mira</aunt>
     <uncle>Sami</uncle>
     <uncle>Joseph</uncle>
   </hanna>
 </family>
```

Figure 6 - family7.xml Document

```
▼<restaurant>
  ▼<joseph>
    ▼<appetizers>
       <appetizer>fries</appetizer>
     </appetizers>
    ▼<burgers>
       <burger>chicken</burger>
       <burger>beef</burger>
     </burgers>
    ▼<sandwiches>
       <sandwich>shawarma</sandwich>
       <sandwich>taouk</sandwich>
       <sandwich>fries</sandwich>
     </sandwiches>
   </joseph>
 </restaurant>
```

Figure 5 - menu7.xml Document

```
▼<a>
  ▼<b>
     <c>IDPA course</c>
     <d>hello</d>
   </b>
 </a>
```

Figure 4 - query.xml Document

3.3 Indexing Evaluation

3.2.1 Time Evaluation

There was a significant time while comparing between indexed and non-indexed queries. We uploaded an XML document (figure 6) with structure and content different than those found in all documents in our dataset. While using structure and content comparison with no indexing, it took 26.55 ms to return empty the ranked results whereas it took 12.22 ms to evaluate with indexing. A flat text was also submitted with the word "hello" which is not contained in any of the documents, the empty result was returned after 21.02 ms when no indexing while it took 8.92 ms with indexing. Therefore, the time needed to process a query decreases as we shift towards indexing. Although the time difference was not too big, however, if we consider bigger query and larger dataset, indexing would help a lot in saving time.

3.2.2 Union or Intersection

As per of indexing, the returned documents were evaluated based on if they have exactly all the words in the query, which is the intersection of each term's documents value, or if they have any of the words mentioned in the query which is evaluated based on the union. The software presented had also the chance to pick between intersection or union. When choosing the union and querying "David Bryan", both music1.xml and netflix2.xml are returned. However, when choosing intersection, only music1.xml is returned as it contains exactly the term. Therefore, for more accuracy and exact result, intersection might be considered.

3.4 TED and VSM Comparison

3.4.1 Time Evaluation

As an aim to evaluate the time between TED and VSM, we started by considering only 4 documents in our dataset, and then ran the comparison found in our software interface and finally recorded the time for both TED and VSM. The number of documents found in our dataset was increased each time by 4 and the same steps were repeated again until 10 iterations were made (40 documents were reached in our dataset). The results were recorded in the graph (figure 7). The time complexity for TED starts increasing rapidly as the number of documents increase while that of VSM remains increasing slightly.

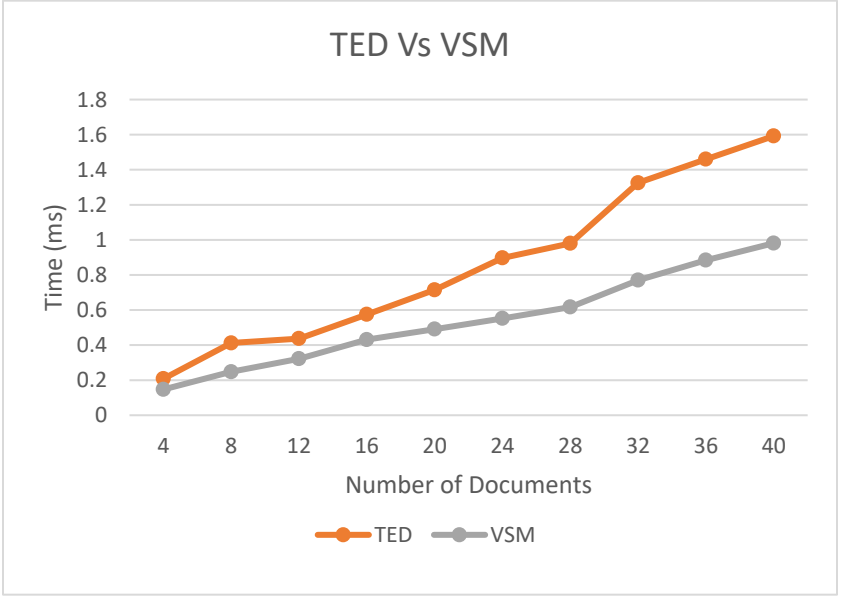

Figure 7 - Variation of time complexity of VSM and TED as with the number of Documents

3.4.2 Results Evaluation

In order to evaluate the similarity results obtained from TED and VSM, the comparison algorithm was run while choosing base document as netflix6.xml (figure 1). In terms of maximum similarity, both TED and VSM returned a result of 1 while comparing the same document. However, TED approach returned similarity values minimal with the other netflix documents which was not the case with VSM. The netflix5.xml has a similarity of 0.997 with netflix6.xml using VSM yet by TED it was 0.333. Therefore, VSM produces more accurate results in comparison with the TED approach.

4 Conclusion

REFERENCES

[1] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In Proceedings of the 5th ACM SIGMOD International Workshop on the Web and Databases (WebDB), (2002) pp. 61-66W.-K. Chen, *Linear Networks and Systems.* Belmont, Calif.: Wadsworth, pp. 123-135, 1993. (Book style)

[2] J. Tekli and R. Chbeir, "A novel XML document structure comparison framework based-on sub-tree commonalities and label semantics," *Journal of Web Semantics*, vol. 11, pp. 14–40, 2012.

[3] J. Tekli, R. Chbeir, and K. Yetongnon, "An overview on XML similarity: Background, current trends and Future Directions," *Computer Science Review*, vol. 3, no. 3, pp. 151–173, 2009.