Identifying Structural Mapping between XML Fragments

Aseem Bajaj[1], Thorsten Joachims[2]
Cornell University, Ithaca, NY
{ab378[1], tj[2]} @cs.cornell.edu

Abstract

With the popularity of XML for representing & exchanging data, the requirement for agreement between parties on common Schema or DTD has become significant. Getting various parties to agree on a common standard is often time consuming and complex problem.

This work suggests an approach to identify mapping between XML documents with different schemas.

Introduction

Identifying mapping between two sets of XML documents would find usage in various areas. Here is a scenario. A user that hosts her web log on blogger.com wants to shift to City Desk (another web logging system). There has been little standardization of web log data representation. Both the systems support XML Import & Export. The matcher can help create a transformer by identifying corresponding tags in the two schemas.

```
<item>
       <title>My name is...</title>
       <description>
       Jack Bauer and ths is going to be the longest day of my life - well,
       these were exactly my thoughts during our trip to Vegas - it took us more
       than 24 hours to get there. Nice! But overall it went were smoothly without
       any serious problems. I just tried the network connection here and everything
       seems to work, even VPN - Great! So, only 10 minutes to go before the
       conference starts. More later...perhaps :)
       </description>
       <guid>http://radio.weblogs.com/0107211/2003/11/17.html#a183</guid>
       <pubDate>Mon, 17 Nov 2003 16:21:20 GMT</pubDate>
  </item>
```

```
<entry>
       <time>October 25, 2003</time>
       <URL>http://www.joelonsoftware.com/items/2003/10/25.html</URL>
       <heading>Tidbits</heading>
       <posting>
       My incoming spam is running at over 200 junk emails a day,
       but SpamBayes is catching them all, with virtually no false positives.
       Bayesian filtering, invented by Paul Graham and available in
       many open source implementations, is the best answer yet.
       </posting>
</entry>
```

entry	=	item
time	=	pubDate
URL	=	guid
heading	=	title
posting	=	description

Above example is a simple case of XML Fragment mismatch. A generalized problem can be stated as below.

Problem Description

Given a collection of XML documents, all adhering to a common structure, for a new test document, find mapping for each of the tags in the test document to the tags in document collection.

The test XML document could be very different from the training documents.

- Tags names may not match
- Attribute names may not match
- The parent-child relationships of tags may not be maintained.

The problem of structural mapping between XML documents is closely related to some other problems

- Merging two XML documents.
- Finding the difference between two XML documents
- Information Retrieval on XML Documents.
- Identifying & calculating a measure of structural similarity between XML documents

Related Work

Recently there has work on matching structural similarity between XML Documents.

`Measuring Structural Similarity among XML Documents and DTDs [5]` is used for clustering XML Documents based on similarity. For a given XML document, the approach finds a matching DTD from a group. The approach defines and/or uses four key data structures

1. DOM representation of XML document
2. DOM like tree representation of DTD
3. Instance DTD, a tree representation of XML obtained from the DTD without the optional tags
4. Repeatable Sub trees, set of sub trees that are repeatable

The approach uses the following measures for similarity

1. Existence of common elements in the two trees, it also keeps a track of elements existing exclusively in one of the two trees
2. Element repetition
3. Level at which an element occurs
4. Element weights to measure unmatched elements

The paper defines an evaluation function and matching algorithm to calculate the “similarity measure” between the XML document and DTD.

But the approach considers tag equivalence instead of tag similarity to calculate the document similarity, though later, it uses attributes and sub elements to check tag similarity. This approach cannot be used in this context as we are trying to find an approach that works even in case the tag & attributes names don’t match.

`Similarity Search in XML Data using Cost-Based Query Transformations [6]` proposes approXQL, a query language that returns ranked results from XML documents. Traditional query languages on XML like XPATH & XQL are like structured query languages that return results if any found that match the queries. The query language looks like XPATH. The basic approach creates a query tree. Then it applies following transformations on it to get a bigger tree

1. Renaming of node to search for elements that don’t have the same name as given in the query
2. Insertion of child elements to search for elements those have other child elements apart from the ones specified in the query. It allows insertion only between the children that have been specified in the condition.

3. Deletion of child elements, to search for elements that don't have the child elements specified in the condition.

The algorithm then matches elements in the XML documents that match this wider (more relaxed) query condition/tree. For each match, the conditions that are used are used to evaluate the rank for the search.

`XML Retrieval Models based on Structural Proximity for Irregular Structured Document Sets [8]` use structural information to improve Information Retrieval. The approach takes care of the fact that XML data is generally unregulated. The paper suggests a query model called the Set of Regular Paths, SRP, to get the ranked results. The query may be used for structure-content or content only information. The approach then uses the edit distance, proximity & term weights to calculate the similarity between XML documents.

Though information retrieval is another potential application of our work, and that is what the above two papers discuss, these works cannot be useful in suggesting a mapping between XML document structures.

`Evaluating Structural Similarity in XML Documents [7]` discusses an approach to partition a group of XML documents into sub-groups based on similarity. The approach uses dynamic programming to calculate edit distances between trees. The edit distance is used as a measure of similarity. The approach identifies the following types operations that would result in tree transformation

1. Re-label Node
2. Insert Node
3. Delete Node
4. Insert Tree
5. Delete Tree

Though this approach can find good measure of similarity (and that's another application that can be built over our work), this approach cannot be adapted well to identify mapping between XML Documents in the first place.

Overview

Text Classification and Information Retrieval problems have pretty successfully used machine learning by taking words in document as features and learning from them.

Unlike traditional text classification model, that learns and classifies documents, in our approach machine learning is used to classify each of the tags in the test document. The class learning is based on the each of the tag instances in the training documents. Here is the outline of the process

- Each of the tags in each of the training documents are extracted
- The content, it's properties and structural information of the tag is used as features & the tag name as the class name for supervised training of the SVM
- Each of the tags in the test document is extracted, and in a similar fashion as the last step, the features are used to create vector representation for each of the tags.
- The SVM classifier is then used to suggest classification for each of the tags.
- A post-processing step that applies some rules (based on some observed properties of the XML) to suggest the final mappings.

The diagram below shows the vector space representation for each of the tags.

Tag	Content Features				Content Info Features	Structural Info Features	
Item					Nullcontent:1	Repeats:1	Children5:1
Creator	Matt:1	Welch:1			Tinycontent:1	IsLeaf:1	
Title	Saudis:1	Fear:1	Sand:1	Shortage:1	Tinycontent:1	IsLeaf:1	

Learning from Content

Hypothesis

Using each of the leaf tags in the XML collection as a training example, with tag name as the training class, a text classifier can classify each of the leaf tags in the test XML fragment with the correct tag name from the training schema.

Expected Results

Given sufficient training, identifying mapping for each leaf tag

- Would give high accuracy on the training set
- Would work well for
  - Text Content tags, with the same accuracy as in text classification problems
  - Date & URL tags, because of high learning of some similar tokens

Experiment Process

The experiment consists of the following steps

1. Parse each of the tags in each of the training documents, and extract out leaf tags
2. Consider each of the above extracted texts as individual documents with tag names as classification & create a Lemur Web Document input format.
3. Create a Vector Space Representation of each of the documents
4. Train the SVM with training set defined in above set
5. For the test XML fragment, extract out tags and content to form a vector space representation similar to step 3
6. Use the learnt model to classify each of the test fragments tags and find suggested tag names from the training set.

Learning from Content Properties

Hypothesis

Using various properties of content in various tags in training collection can improve classification accuracy.

Expected Results

For leaf tags, following properties can have stated results.

- Content Length: The classifier differentiates between tags based on size, like Title and Description.

Experiment Process

To the above experiment process, add the following steps.

- While extracting out tags and content from XML documents, add the following to the content before writing it in Lemur Document format
  - Depending the size, add information for sizes 0, 1, 2-4, 5-32, 33-256, 257 and above

Learning Structural Information

Hypothesis

Using structural information properties as features in vector representation of the tags can provide matches with fair accuracy.

Expected Results

The classification for tags especially non-leaf tags in the test fragment would give better accuracy with the use of structural information as features.

Experiment Process

Use the experiment processes of first hypothesis and instead of tag content, use the following properties similar to the second experiment above.

1. Find the number of child tags and
    a. Add a feature stating Number of children exactly n times
    b. Add a feature stating Number of children – 1 & Number of children + 1 k times.
    c. Let n=3 & k=1. Note that this can be distributed over a distribution curve as well, but for simplicity we keep it this way.
2. Apply Step 1 for number of attributes using set n=1 & k=0
3. Apply Step 1 for number of siblings using set n=3 & k=1
4. Apply Step 1 for number of siblings of parent node using set n=3 & k=1
5. Apply Step 1 for depth level of the tag in the document using set n=1 & k=0. Note that the root tag would be 0
6. Add feature to identify if a tag is leaf or non-leaf
7. Add feature to identify if a tag repeats.
8. Add feature to identify if a tag's parent repeats.
9. Add feature to identify if any of the tag's children repeat.

Post Processing Rules

Hypothesis

Suggested mappings for all of the instances of a given tag in XML documents should be the same. Use of voting & confidence values after classification can suggest correct uniform mapping.

Expected Results

A few of the classification made by the classifier would be weak.

1. This could be either because of lack of positive confidence in the results
2. Or because of more than one classification being made for the various instances of the same type of tag.

The post-processor would use the classification results & it's own rules to come up with a common, stronger & better suggestion all the test instances of the same tag. This would improve the accuracy results.

Experiment Process

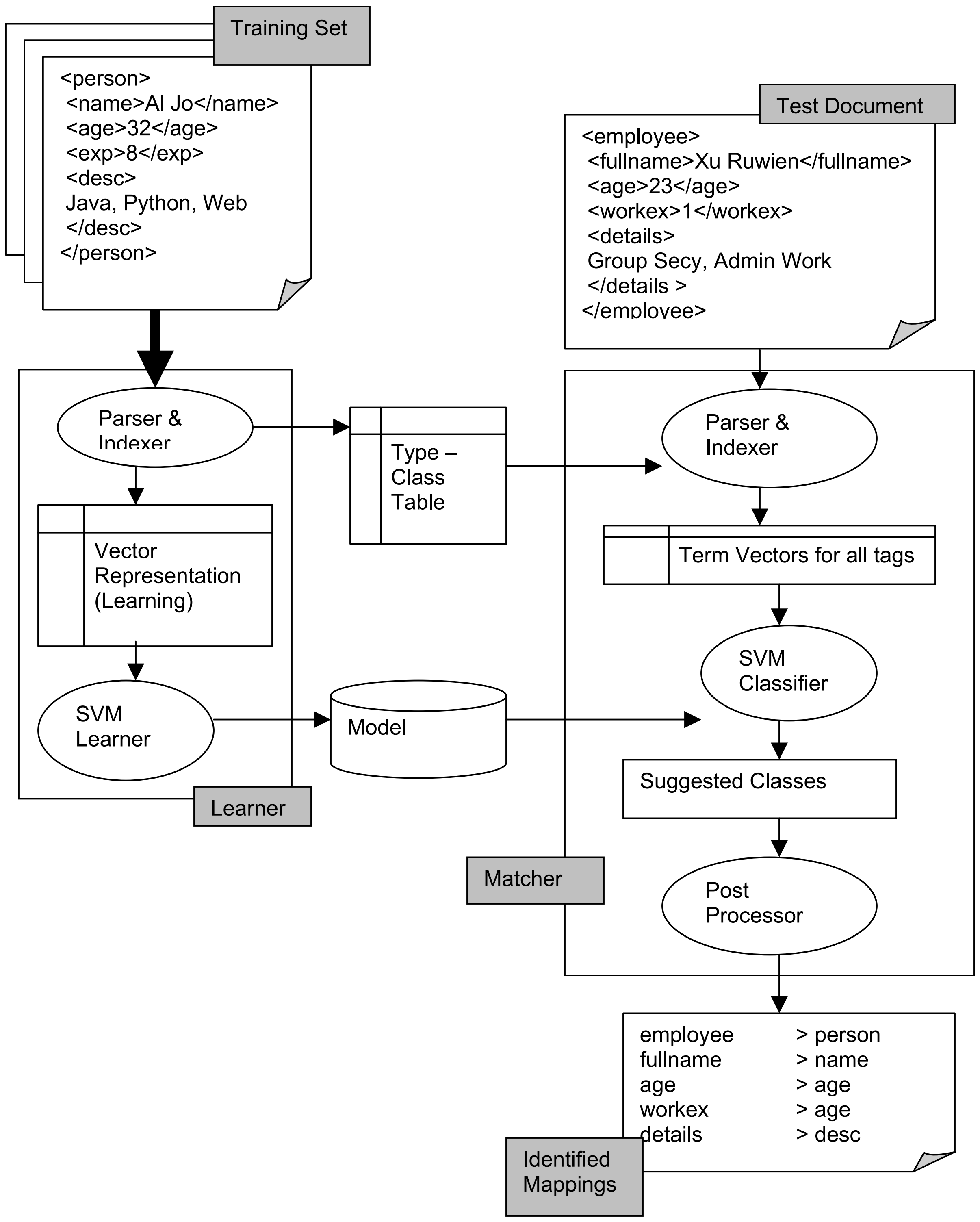
Use the above experiments to suggest tags for all the tags in the test document. Then apply the following post processing rules.

1. Identify the list of unique tag names in the Test XML Document.
2. For each tag type chosen above, check if the all the suggested tag names are same. If not, change all of them to the same value as chosen below.
    a. If there is at least one positive match (confidence > 0): Among the tag names with positive matches, choose the one that is suggested the most.

b. If there is no tag with a positive value of confidence: Among the tags names with negative values, choose the class that had the highest confidence level.

Design

Flow Diagram



Learner

Learns the structure of the training document in the following steps

1. Uses Xerces [4] & Xalan [4] libraries to parse the XML
2. Adds the content properties and structural information as words to the content. In step 5, these words get converted to features in Vector Space Representation. Following properties can be added to the content based on flags in `stamp.properties` file.
    a. Content Length (`null, single, tiny, small, large, huge`)
    b. Number of Children
    c. Number of Attributes
    d. Number of Siblings
    e. Number of Parent's Siblings
    f. If a tag is leaf
    g. If a child tag is repeating
    h. If the tag is repeating
    i. Depth of the tag

    Note: Among the above properties, "Parent's Siblings" and "Depth" are not being used for the tests because the XML document in the collection have the same structure and these properties would give undue advantage to the tests.
3. Converts the tag contents to input format for Lemur [3] Indexer
4. Lemur [3] project's indexer is used to parse the content
5. A bunch of pre-processing scripts convert this index to Vector Space Representation
6. The SVM light [1] learner is invoked through jSVM [2] API and it creates a model.

Matcher

Identifies the mapping between provided XML document and the document structure learnt by the Learner.

1. Follows the steps 1 to 5 of the learner for the test document.
2. The SVM light [1] classifier is invoked through jSVM [2] API and it suggests mapping for the tags in the test document.
3. A post processing script uses the above suggestions to come up with a final mapping.

Results & Evaluation

Test Collection

The test collection consists of main page content published by 6 web log sites. All the 6 XML documents follow a common RSS 1.0 schema.

Leave one out test

The test procedure is explained here.

- Each of XML documents above is picked one by one.
- All except the chosen document are chosen for training using the 'learner'.
- For each of the tags in the chosen test document, the matcher comes up with a suggested tag.
- The accuracy for each tag is calculated by checking if the suggested tag is same as the actual tag.
- The overall accuracy for the document is calculated by taking into account the number of instances of each of the tags.
- The above steps are performed for each of the documents in the collection.
- The average accuracy on the collection is then calculated by averaging each of the document accuracy figures.

Accuracy Results

Shown below are the accuracy results for overall accuracy of each of the documents tested. The 'Total' column prints the number of tags tested in each document, while the '+' & '-' identifies the number of correct & incorrect classifications.

```
  Class   Total   +       -       Acc

Overall  131     100     31      0.7634
Overall  131     115     16      0.8779
Overall  131     100     31      0.7634
Overall  131     115     16      0.8779
Overall  132     115     17      0.8712
Overall  131     115     16      0.8779

Average Accuracy                   0.8386
```

For each test document there are just one or two errors in mapping (as is visible in the mappings shown in the next table). An average accuracy of 83% is achieved.

Suggested Mappings

The Matcher suggests the tags for each of the tags in the test document. Here are the suggested mappings for the last run in the above test.

- The 'Actual Tag' column indicates the actual tag names with number of instances in the parenthesis.
- The 'SVM Suggestions' column indicates the suggestions by the SVM. A '+' indicates a confidence level greater than 0 while a '-' indicates a confidence level less than 0. The figures in parenthesis indicate the number.
- The 'Final Suggestion' tag indicates the final suggestion made by the matcher after processing SVM results.

```
         Actual Tag                                     SVM Suggestions    Final Suggestion

generatorAgent( 1)                                   +generatorAgent(1)      generatorAgent
       creator(16)                                +creator(1)+title(15)               title
   description(16)                            +description(15)+title(1)         description
      language( 1)                                         +language(1)            language
         title(16)         +title(11)+subject(3)-title(1)-subject(1)               title
          item(15)                                            +item(15)                item
       subject(15)                                         +subject(15)             subject
            li(15)                                              +li(15)                  li
       channel( 1)                                          +channel(1)             channel
         items( 1)                                            +items(1)               items
           RDF( 1)                                              +RDF(1)                 RDF
          date(16)                                            +date(16)                date
           Seq( 1)                                              +Seq(1)                 Seq
          link(16)                                            +link(16)                link
```

The post processor helps in making correct judgments about the mappings. In the above example out of the 3 cases where multiple suggestions are made by the SVM, the post processor makes right judgment in two cases, while in the remaining case it makes a wrong selection (but even in that case, there is no way it could have made right selection because the classifier results were heavily in favor of the answer the post processor selected)

One on One Test

To analyze the above results and to test out our hypotheses, we conducted the one on one test. The test was conducted as such

- Two documents from the collection were picked at random
- The learner was used to learn on first document
- The matcher then found mapping between the second document and the first one.

Here are the results for the best case found in the above random selection.

```
    Class  Total   +       -       Acc
  Overall  131     115     16      0.8779
```

Even though the training was done with just one document, the accuracy here is comparable to the case of training with 5 documents. This is because each of the documents has many instances of a tag & the SVM gets a chance to train on many examples.

One on One Test with Content Information

The above tests were repeated without using Structural Information. The table below represents results using content and it's properties as features.

```
    Class  Total   +       -       Acc
  Overall  131     95      36      0.7252
```

Simply using content and its properties is enough to give an accuracy of 72%. Though this is not as good as the previous case, it proves the hypothesis that content information can be used to identify structural mapping between XML fragments.

Note: Tests were also conducted by using purely content as the features. The accuracy for the document was poor, around 1%. By using just one document as training hardly gave the SVM a chance to learn, hence the results. Since these are obvious results, they have not being included here.

One on One Test with Structural Information Only

The one on one test was repeated without using Content & it's properties & only using structural information. Here is the table of results.

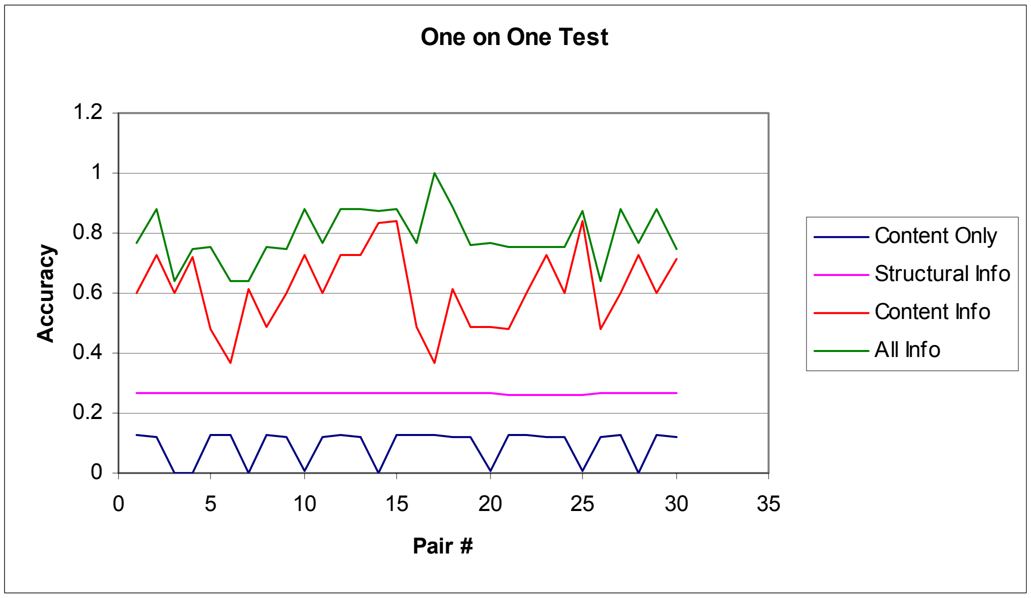
```
    Class  Total   +       -       Acc
  Overall  131     35      96      0.2672
```

The '`items`' & '`RDF`' tags in the documents are non-leaf tags. Note that they have been mapped correctly here, while they were mapped incorrectly in the last test, while simply using content information. This is because they don't have any content and the SVM doesn't have a chance to learn any feature about them except the fact that they have null content.

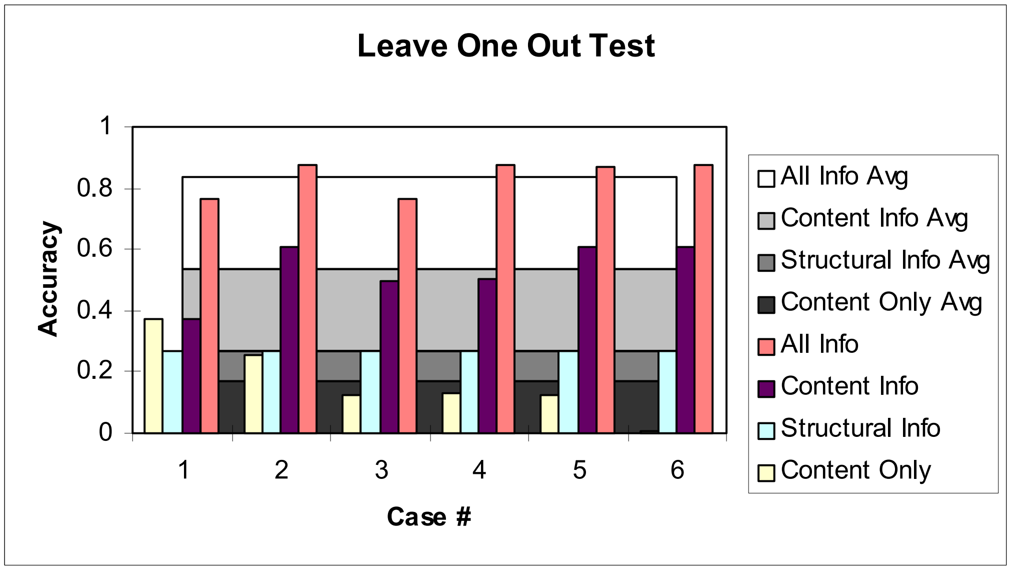
The results of the above table, shows that structural information in itself is not enough to find mapping between the structures. But the structural information supplements the content information to boost the performance significantly.

Comparison: Content & Structural Information

The above four one-on-one test (using various features) were repeated for all 30 pairs in the document collection. The graph below shows the results.



Similarly the leave one out test was repeated by making the same variations in choice of features as above.



The accuracy results for tests using pure content as features improve in the latter case because the SVM get a chance to learn more from the data. But the accuracy results using pure structural information remains exactly the same. This is because the structure of the document is same and there is nothing new to learn for the SVM after it has looked at one document.

In both the cases it's clear that even though content & structural information are useful in identifying mapping between XML documents, combining them gives better accuracy numbers.

Conclusion

The work presented the idea that it is possible to use various properties of XML documents to identify mapping between them. Specifically, we use properties of tag content & information about structure of the XML documents as features in Vector Space Model to find the mapping between tags of two XML documents. Though each of these two pieces of information can find mappings to some extent, using them in conjugation gives the best results.

Scope of Future Work

Here is a list of possible suggestions for further development going ahead from here

- Extending the post-processing into a converging algorithm, that iterates over classifications suggested by SVM and previous classifications suggested by itself to come up with better classifications.
- Incorporating user feedback as an input for the above converging algorithm.
- The work can be made into a useful standalone library by coding XML Parsing, Indexing, Learning, Classification & Post-Processing in one language eliminating all the intermediate data objects from the user
- The work can be used for defining and evaluating XML Structural Similarity. It can be used for at least two powerful uses.
    - Content Publish-Subscribe Engine: A publish subscribe model would depend on known XML Schema(s) to place subscription for clients. Unfortunately a lot syndicated data coming from various content sources (typically news sources) would not have a standardized schema. A structural similarity matcher would allow the user to place subscription in form of a ‘desired XML Fragment’ rather than a well-structured query.
    - XML Structure Aware Search: Search Engines can use the XML structure of the documents to improve search results by looking at ‘certain types’ of tags that attach weight to occurrences of terms. The matcher help the search engines identify such tags because the tag names and position would not be fixed for any document.

References

[1] Thorsten Joachims, SVM light, a Support Vector Machine Implementation
`http://svmlight.joachims.org/`

[2] Heloise Hwawen Hse, jSVM, a Java Wrapper for SVM light
`http://www-cad.eecs.berkeley.edu/~hwawen/research/projects/jsvm/doc/manual/`

[3] Lemur Project
`http://www.cs.cmu.edu/~lemur/`

[4] Jakarta Project
`http://jakarta.apache.org`

[5] E. Bertino, G. Guerrini, M. Mesiti, I. Rivara, C. Tavella: Measuring Structural Similarity between XML Documents and DTDs

[6] Torsten Schlieder: Similarity Search in XML Data using Cost-Based Query Transformations

[7] Andrew Nierman, H. V. Jagadish: Evaluating Structural Similarity in XML Documents

[8] Shinjae Yoo: XML Retrieval Models based on Structural Proximity for Irregular Structured Document Sets