Scalable Semantically Driven Decision Trees for Crime Data

Shawn Johnson, George Karabatis
Department of Information Systems
University of Maryland, Baltimore County (UMBC),
1000 Hilltop Circle, Baltimore, MD 21250, USA
{yv74924, GeorgeK}@umbc.edu

Abstract. **When dealing with large volumes of data in organizations, there is always a need to associate data with its appropriate meaning, since the same data object may have different meaning to different users. This creates a problem of delivering search results that is different from a requester's intended purpose. To solve this problem, we propose a parallelizable framework capable of capturing user specified constraints that are both semantically relevant to a search/domain in question as well as contextually relevant to a user and/or organization.**

I. INTRODUCTION

When attempting to find data that is relevant to a user and/or organization based on a query, it is very common to retrieve exact matches in response to a search. While using the exact matching of strings as user search keywords can retrieve exact results, a user may be looking for data with a specific meaning based on his or her intended search preferences but the data that is provided by a system or organization may have a completely different meaning. This problem is further compounded by having to ensure that data that has recently been ingested into a system is consistent with data that currently resides on the same system. This can create an intractable problem for a user such as having to constantly poll and classify new data or accept new data that may be inappropriately classified to ensure the semantic meaning of the data is consistent. In addition, the problem of new data being added to a system on a massive scale necessitates a scalable solution. We propose an approach which allows users to personalize search terms according to the same set of concepts where search results can be universally understood by the same community of users according to personalizable search profiles. Our approach also enhances the accuracy of search results by returning semantically similar results from a specific domain. The impact of our approach dramatically enhances the ability of users to personalize a search thus retrieve more accurate results. With the introduction of our framework, we make a few key contributions:

1) We allow user specified search preferences to be expressed with robust semantics resulting in higher precision and recall that closely match the user's intended search terms.

2) We have developed a method for user specified semantics to be expressed probabilistically in the search. Search results that have a semantic similarity to a set of user specified preferences are also returned enabling us to handle uncertainty in our approach as well.

3) We have developed a way for multiple sets of user specified preferences to be expressed using the same ontology.

4) Our methodology allows for robust semantics to be expressed in a parallelizable way.

5) We have implemented a prototype and evaluated our methodology by conducting experiments using precision and recall as metrics.

Motivating Example: Many municipalities often have a need to capture semantically relevant data for an intended purpose. For example, Bob, a detective with the city of Baltimore, needs to get the current statistics of all aggravated theft incidents between 1964 and 2013, in Baltimore, Maryland, USA. Bob wants to compare this data with the same data on a national scale over the same time period. Furthermore, there are different types of data on thefts that Bob wants to compare against. To further compound the issue, different vendors provide different labels for the same type. The impact of capturing more semantically relevant data and getting more accurate results enables law enforcement officials to make better informed decisions because the information they are looking for is more precise and relevant to a specific situation for which the officials need to make a decision about.

We describe our approach in section II, and then we continue on with a discussion of our experiments

in section III, a validation of our approach in IV, and then a discussion of relevant work in section V. We then finish up our work with concluding thoughts and proposed future work in section VI.

II. Approach

To begin our learning, we make use of a single ontology that represents our categories of context, with each category of context represented via separate branches of the ontology.

Continuing our motivating example, we show a sample representation of our ontology using figure 1 to show the taxonomical tree of terms that a user can select as user preferences and how they organized. Utilizing the Operational of Context [1] we model our ontology using 5 separate categories of context.

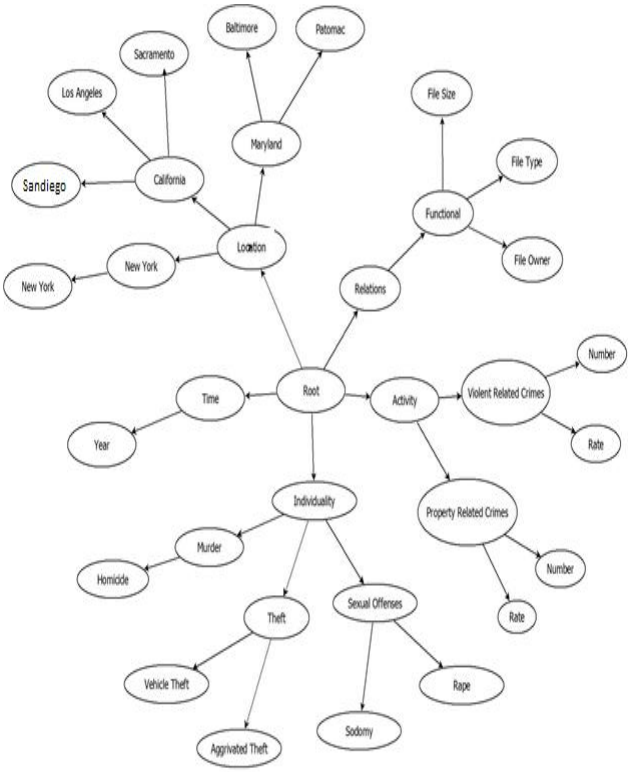

Fig. 1 Sample representation of an ontology:

Operational Definition of Context [1], we model our ontology using 5 separate categories of context. They are:

1. Individuality Context - The individuality context are attributes that describe an entity's type such as specifying the type of theft
2. Time Context - The time context is anything that describes any kind of temporal attributes related to an entity such as a year
3. Location Context - The location context describes any kind of location attribute related to an entity such as a city or state
4. Activity Context - The activity context describes a goal related to entity such as how much of something a law enforcement official may be looking for such as a total amount of violent crime
5. Relations Context - The relations context are attributes of an entity that describe an entity's relation to another entity or its parts

User preferences in our ontology are saved with literals that are added to each node a user has selected. We chose not to add countries in our ontology because we are assuming that our domain is within all 50 states within the USA. Bob the detective stores his preferences in a user profile that matches parts of the ontology (in figure 1) such as all aggravated thefts that have occurred in Baltimore, Maryland, between 1964 and 2013. A special depth first search algorithm then searches the ontology and matches saved user specified literals for each node within each branch of the ontology and creates a special in memory tree model. This in memory tree model is a sub-tree of the ontology that matches all of the user selected preferences that were found in the original ontology. Referring again to figure 1, only parts of the ontology that match aggravated theft, Baltimore, Maryland that are between 1964 and 2013 are copied into the new in memory tree model.

After the original ontology has been parsed and the in memory tree model has been implemented, we are now able to build the rest of our decision tree. Records (files) are initially classified based on user preferences using the in memory tree model provided in figure 2. For example, all records that are classified must include records that have the attributes Baltimore, Maryland, aggravated theft, or between the years 1964-2013 (meeting Bob's specified user preferences). Now that only instances remain that match the user specified constraints above, (i.e., the records are all containing values for the attributes Baltimore, Maryland, aggravated theft, or in the range 1964-2013) we must build out the remaining part of a tree based on the number of user specified splits or criteria to enable the system to return search results with increasing levels of precision by dividing the records into further subsets as specified by the user

with each additional split. For example, if the user has specified that he or she wishes to have the ontology be 5 hops deep and the ontology is only 3 hops deep, additional splits must be completed (if there are enough records left to split against in order to meet the user's specified preferences). Referring to figure 2, additional splits are made using the lowest level collection of leaves for the individuality and location branches of the ontology (additional splits were not made to the time branch of the ontology because the user chose not to do so since year was the most granular measure of time specified in the data). Because the user has selected the decision tree to be 4 hops deep, additional splits are made using the lowest collection of leaves for each branch of the ontology. In the location ontology, the lowest collection of leaves in that branch of the ontology are Baltimore, Potomac, Los Angeles, Sacramento, Albany, and New York City. Because Potomac has the lowest impurity (see Semantically Driven Gini Algorithm) computed from the lowest collection of leaves, it is selected as the first candidate split using the Semantically Driven Gini-Index Algorithm. No more splits are performed on the location branch of the ontology because it is now 4 hops deep. On the individuality branch of the ontology, a single split is made using The Semantically Driven Gini-Index Algorithm, with vehicle theft being selected because it had the lowest impurity (see Semantically Drive Gini Algorithm) of the lowest leaves. No more splits are made for the individuality branch of the ontology because it is now 4 hops deep as well. Our special in memory tree model/user driven ontology looks (logically) like Figure 2.

Any documents that match any of the user preferences or Semantically Driven Gini-Index Algorithm are then tagged using a custom document summarization algorithm. The tagged files are then copied to The Hadoop Distributed File System (HDFS). Separate mapper calls are then kicked off using search conditions that meet the user specified preferences. A single reducer job is kicked off consolidating a set of files that meet a final set of preferences as specified by the user in the in-memory tree model. A final set of key value pairs of files using the key as the name of the matching file and a value of 1 that matches all of the user specified preferences are emitted signifying that the algorithm is complete. For example, only the names of files that meet all the user preferences Baltimore, aggravated theft, and between the time range of 1964-2013 are emitted.

In Memory Tree Building Algorithm: To utilize only paths of the ontology that the user has selected as his or her search preferences, a smaller graph of the in-memory tree model is created copying each of the paths that a user has selected. First, the user specifies the number of hops deep (or total depth) he or she wishes for the size of the decision tree to be. Second, the user picks from a list of selected labels that have already been mapped via our ontology via a tree like drop down list or via a search box. (The mappings are a graphical representation of the ontology, where the user can select any one of the nodes of the ontology as a user preference). After our search is complete we populate a list of labels from our ontology and copy the names of the nodes (and their parents) that the user has selected. The same set of user preferences will be copied from all five categories of the ontology below (with each branch of the ontology being stored as a separate but parallel part of our in memory tree model). The in memory tree model persists in memory as a service. Files are matched against the user preferences that have been stored in the in-memory tree model.

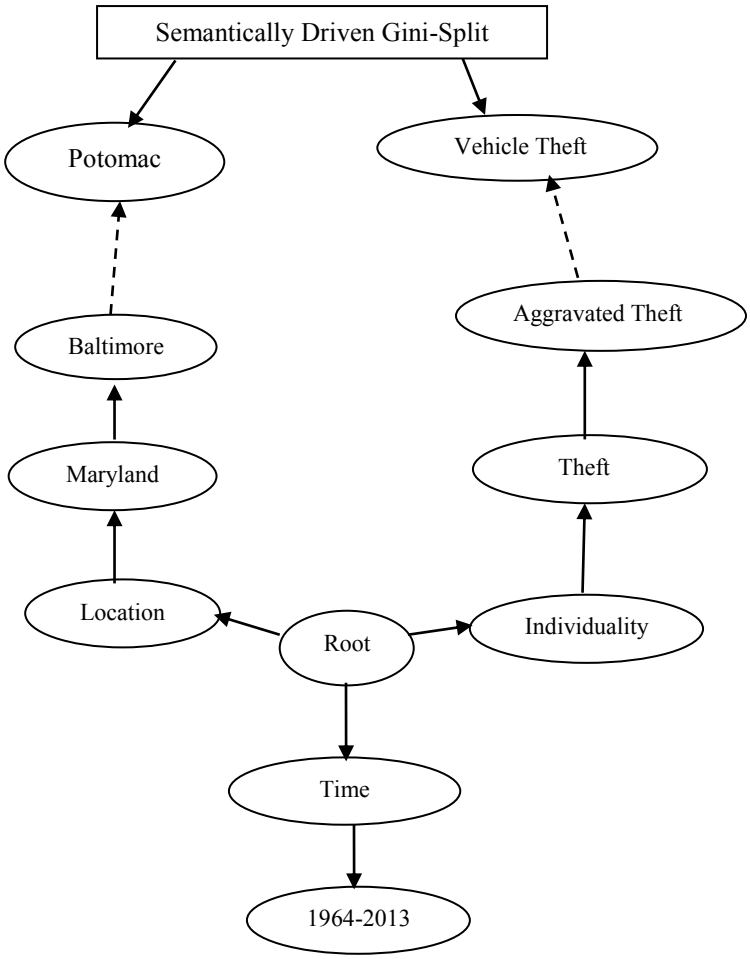

Fig. 2. A logical representation of an in memory tree model (including splits from the Semantically Driven Gini-Index Algorithm):

We implement our depth first search algorithm using the following representation: C for Category Contextual Model Data, O for Ontology, T_x for each branch of the ontology (each primary branch of the ontology connected to the root), where p_{abcd} represents a current node, c_{abcd} represents a child node, a indicates the current level (depth) of the ontology, b where a holds a 1 or 0 (1 if for each node selected by user or 0 if not selected by the user holds a pointer to the child node, and c is a list of all other user specified preferences, and d is a list of pointers to all child nodes of the current (parent) nodes.
In memory tree algorithm pseudocode:

Input: Ontology O
Output: in-memory tree model

LOOP: Repeat the following steps (for a + 1 of the current node until a = the lowest leaf in the tree), for T_1 - T_5, until all of O or the entire ontology is finished

1) For each category of context from Root we represent our Ontology as follows: O = $\{T_1, T_2, T_3, T_4, T_5\}$. For each branch of the tree T_x=C_{abcd} and C is the starting node for each tree (i.e., the starting node after root is Location, Time, Individuality, Activity, and Relations for each of the 5 categories of context).

2) For a of p_{abcd} of T_x add c_{abcd} to c of $p_{x\,bcd}$ where p_{abcd} is a current node, C_{abcd} is the child node of p_{abcd}, a is the current depth of T_x, b holds a 1 or 0 (1 if a user has chosen a user preference or 0 if the node has not been selected as a user preference), c holds a list of all other user specified preferences and d is a list of pointers from p_{abcd} to children nodes C_{abcd}

3) If no children exist for d, d = NULL.

END LOOP;

Semantically Driven Gini-Index Algorithm: Once the in-memory tree model building algorithm is complete, our custom Semantically Driven Gini-Index Algorithm is kicked off. First, we define our Semantically Driven Gini-Index Algorithm which calculates the impurity of each node as follows:

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2 \qquad (1)$$

Input: Lowest collection of leaves for each branch of the in-memory model

Output: Additional children nodes for each branch of the in-memory model

Where t is the node, j is the class, and p is probability of class j given a node t. The algorithm works as follows:

1) Initial splits are made based on classes specified in the in memory tree model. For example, aggravated theft is a candidate for a decision tree split since aggravated theft is a class in the tree (originally specified within the ontology).

2) Additional splits at the next level of the tree are implemented using the same collection of leaves (i.e., the lowest collection of leaves for a given branch of the ontology stored in the memory tree model) with the next split being the node with next lowest impurity using the calculation defined in (1).

3) Additional splits are induced using the same lowest collection of leaves (for each branch of the tree) until the max number of splits has been reached either by the following:

- A program driven default - This condition happens when a program driven default number of splits is reached for a given branch of the ontology. For example, if the program default is set to 5 splits, then the decision tree will not split beyond 5 hops deep for that given branch. This is true regardless of whether the split was based on a modeling part of the ontology or a part of The Semantically Driven Gini-Index Algorithm used to calculate a split.

- A user specified limit is reached - This condition happens when the user specifies a max number of splits he or she sets for a given branch of a decision tree. For example, if the max number of splits that is specified is 7 for a particular branch of the decision tree then the decision tree will stop inducing additional splits in that branch of the decision tree beyond that number regardless of whether or not the nodes being split come from the ontology

or additional splits are determined by The Semantically Driven Gini-Index Algorithm.

- The max number of possible splits has been reached - This condition occurs when all possible splits from within an ontology as well as all of the lowest level of leaves have been utilized in a split resulting in a max number of splits that can be used to build a given decision tree. For example, if the ontology is 4 hops deep and the lowest level of leaves totals 4 leaves as well, this makes the max number of splits possible for the decision tree to be 8 split.

III. EXPERIMENTS

To validate our approach, we took roughly 100,000 files from the UCR Data Repository and processed them against 60 user specified preferences stored within the in memory tree model. The semantically mapped features were saved as tags in a modified version of each file, making matching for each set of user preferences a matter of matching the tags that have been specified by the user saved in the in memory tree model. Finally, the resulting MapReduce Jobs generated a file name with a value of 1 for each set of semantically mapped preferences that were matched. We used the following sample Scenario 1: Bob is looking for all crimes that occurred within the state of Maryland, between 1998 and 2002, he is searching for crime totals for larceny theft in which the files are saved as .xls files. Scenarios 2 through 10 are variants of scenario 1, where semantically mapped preferences were matched against the same files from the UCR Data Repository Values for scenarios 2-10 were chosen at random.

We have run three sets of experiments:

1) No Context and No Ontologies - Experiments with user preferences as exact search terms on MapReduce Jobs.

2) Ontologies but No Context - Experiments with files that were tagged using ontologies in RDF/OWL Files saved in the in memory model, but without any user preferences saved within them (copying the entire ontology for each category of context). The tagged files were then processed in a MapReduce Job producing the results.

3) Ontologies and Context - These experiments were executed using the files that were tagged using both the ontologies modeled in RDF/OWL Files, but also with the saved user preferences that were parsed from the RDF/OWL Files as well. The tagged files were then processed using MapReduce producing the results.

IV. VALIDATION

We use two well-known metrics to validate our approach: Precision and Recall. Recall is defined as the fraction of the records retrieved that are relevant to the query. In other words, recall reveals the percentage of the retrieved and relevant records that are relevant (whether in the answer set or outside the answer set). Precision is defined as the fraction of the retrieved records that are relevant to the search. In other words, precision reveals the percentage of retrieved and relevant records in the answer set. Recall and precision are calculated as follows:

$$recall = \frac{|\{relevant\ records\} \cap \{retrieved\ records\}|}{|\{relevant\ records\}|} \quad (2)$$

$$precision = \frac{|\{relevant\ records\} \cap \{retrieved\ records\}|}{|\{retrieved\ records\}|} \quad (3)$$

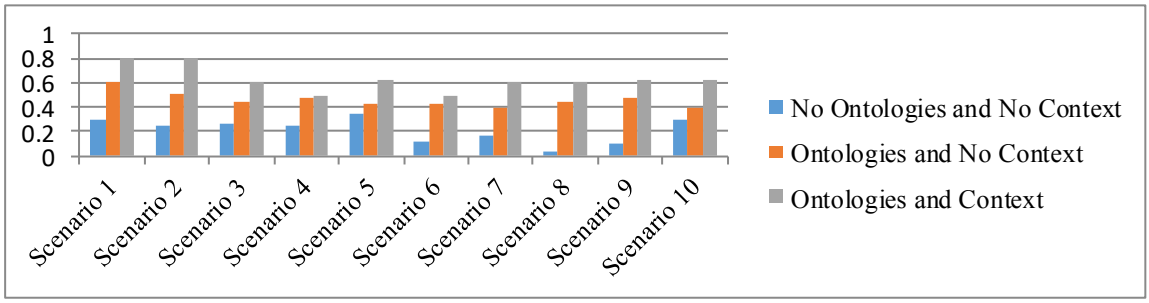

Fig. 3. Experiment results using recall

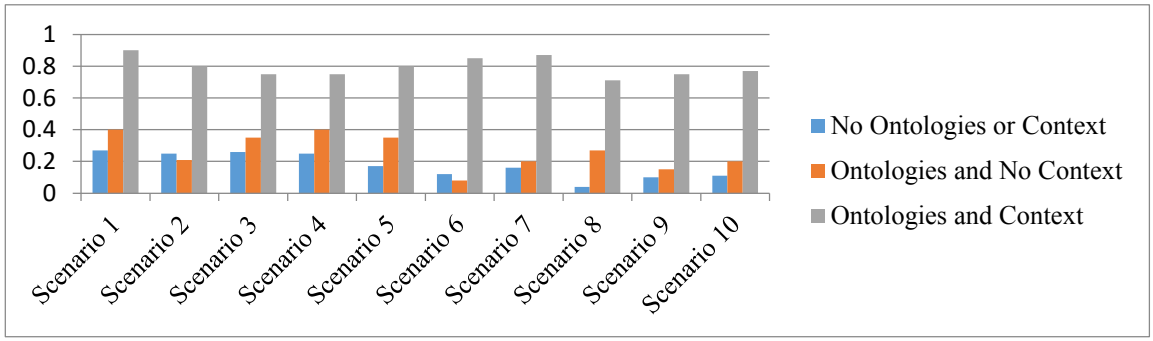

Fig. 4. Experiment results using precision

In the above scenarios or sets of test conditions that simulated using sets of user specified preferences (like in Scenario 1 discussed above), the recall was computed for no ontologies and no context by dividing matching exacting search terms from a user's search against the content of the files that are being parsed. The recall was extremely low for no ontologies and no context because the exact key words used in MapReduce Jobs matched a very small percentage of the files that were relevant to a user's search. The mean average of all 10 scenarios was 28%. The recall for ontologies and no context was computed by taking the matching terms in the in memory tree model that were parsed from the ontology and matching them against properties and content in the files being parsed. The searches for ontologies and no context yielded a higher recall because all the files that were tagged using the in memory tree model matched 1 or more of the classes specified in the ontology with a total of 47% recall, resulting in almost a 20% increase from no ontologies and no context. The recall for ontologies and context was computed by taking the matching preferences stored in the in memory tree model and matching them against properties and content of the files being parsed. Searches involving ontologies and context had a very high recall because all results retrieved matched both the semantically driven preferences that were saved in the in memory tree model as well as with the semantically specified constraints from the ontology resulting in 63% recall, a 16% increase from ontologies and no context and a roughly 35% increase improvement in recall vs. no ontologies and no context.

Precision was computed for no ontologies and no context by dividing matching exacting search terms from a user's search against the content of the files that are being parsed. The precision for no ontologies and no context was extremely low because the overwhelming majority of the search results that were returned did not match the intended user search preferences because exact key words were used for each search resulting in a 17% precision. The precision for ontologies and no context was computed by taking any of the matching terms in the in memory tree that were parsed from ontology and stored in the in memory tree model without any user specified preferences and matching them against the properties and content of the files that are being parsed. Ontologies and no context resulted in a low precision as well because the entire ontology was stored in each in memory tree model resulting in tagged files that only partially or did not match a user's intended search terms at 26% precision. The precision for ontologies and context was computed by taking any of the matching preferences stored in the in memory tree model that parsed from the ontology and selected by the user from the ontology and matching them against properties and content and of the files being parsed. Ontologies and context resulted in a very high precision versus no ontologies and no context and ontologies and no context, because documents that returned key value pairs in our MapReduce Jobs matched the semantically mapped user preferences as well the semantic constraints specified in the ontology resulting in an 80% precision; a 54% increase in ontologies and no context and a 63% increase in accuracy vs. no ontologies and no context.

In summary we learned that enabling a user to pick semantically enriched preferences from an ontology of terms that reflect an existing domain from a corpus can lead to a much higher precision and recall than using exact search terms or just using semantically enriched search terms parsed from an ontology. By building an in memory tree model we are able to both represent the knowledge representative of a domain and we also enable personalization by a user of that knowledge as well. In order to enable robust personalization, semantics must first be reflected in a model before a user can pick them. This is depicted in our results by showing that the precision and recall is higher than with just choosing exact search terms and the precision and recall further improves when allowing a user to pick attributes he or she wishes to use in a search with terms picked from the ontology.

V. RELATED WORK

Zhang et al. [2] formulated an approach that calculated the information gain for finding the best split for a node between two or more separate taxonomies. This approach does not incorporate any kind of semantic inference or contextually driven attributes for building a decision tree, neither does it address any issues with trying to make a decision tree parallelizable. Gajderowicz et al. formulated an approach for enriching manually created ontologies using decision trees [3]. They also developed a system for using decision trees for ontology matching [4]. Johnson et al., formulated an approach for enriching ontologies off of custom built decision trees [5]. Bouza et al. described an approach by using an ontology to build user profiles to make various recommendations on user behavior [6]. Our approach not only allows a user to specify preferences, but also ensures that they are semantically similar to any search results; it is parallelizable too. Fanizzi et al. developed a novel framework for learning custom description logic learning languages using decision trees. While this approach is novel for learning description logic concept definitions, it does not incorporate user preferences [7].

Nenkova et al. developed a technique for summarizing documents based on frequency [8]. Arun and Gunavathi developed a technique for summarizing documents using context sensitive weights for indexing [9]. This work did not utilize the contextual properties of parts of a document nor were user preferences utilized when creating the summaries. Witte et al. developed a fuzzy graph technique for multi-document summarization [10]. Barzilay et al. developed a technique for summarizing documents using the contextual attributes found in text across a series of documents [11]. Yang et al. developed a summarizing framework using the social contextual information, but they did not utilize user specified preferences beyond social ones such as a time or location that a document was created [12].

VI. CONCLUSIONS

We described and validated an approach to specify semantically driven user preferences in a parallelizable way. We also encountered a few limitations. First, we found that extensive exploration and sampling of data files was needed to be able to properly model preferences in our ontology to confirm a consistent structure of a file format when creating our in memory tree model. Second, we found that the user preferences that were specified in the in memory model needed to closely mirror the user preferences that were specified in the ontology or this would lead to searches returning incorrect results or errors resulting in our program because the structure of RDF/OWL Model was incorrect. For our future work we plan on testing much larger datasets. Also planned are further attempts to model more expressive attributes for user specified preferences. Finally, we plan on utilizing Apache Spark [13] and new versions of Hadoop to allow for both a more novel design and implementations of our approach.

References

[1] A. Zimmermann, A. Lorenz and R. Oppermann, "An Operational Definition of Context," 2007.

[2] J Zhang, A. Silvescu and V. Honava, "Ontology-Driven Induction of sion Trees at Multiple Levels of Abstraction," Berlin Heidelberg, 2002.

[3] B. Gajderowicz, M. Soutchanski and A. Sadeghian, "Trees, Ontology Enhancement through Inductive Decision Trees," in *Uncertainty Reasoning for the Semantic Web II*, Springer Berlin Heidelberg, 2013.

[4] B. Gajderowicz, "Using Decision Trees for Inductively Driven Semantic Integration and Decision Matching," Ryerson University, Toronto, 2011.

[5] I. Johnson, J. Abécassis, B. Charnomordic, S. Destercke and R. Thomopoulos, "Making Ontology-Based Knowledge and Decision Trees Interact: An Approach to Enrich Knowledge and Increase Expert Confidence in Data-Driven Models," in *Knowledge Science, Engineering and Management*, Springer Berlin Heidelberg, 2010.

[6] A. Bouza, G. Reif, A. Bernstein and H. Gall, "SemTree: Ontology-Based Decision Tree Algorithm for Recommender Systems," Karlshruhe, 2008.

[7] N. Fanizzi, C. d'Amato and F. Esposito, "Induction of Concepts in Web Ontologies through Terminological Decision Trees," Barcelona, 2010.

[8] A. Nenkova, L. Vanderwende and K. R. McKeown, "A Compositional Context Sensitive Multi-document Summarizer: Exploring the Factors That Influence Summarization," Seattle, Washington, 2006.

[9] J. M.E and C. Gunavathi, "Document Summarization and Classification sing Concept and Context Similarity Analysis," 2014.

[10] R. Witter, R. Krestel and S. Bergler, "Context-based Multi-Document Summarization Using Fuzzy Coreference Cluster Graphs".

[11] R. Barzilay, K. R. McKeown and M. Elhadad, "Information Fusion in the Context of Multi-Document Summarization".

[12] Z. Yang, C. Keke, J. Tang, Z. Li, S. Zhong, L. Jaunzi and Y. Zi, "Social Context Summarization," Beijing, China, 2011.

[13] Apache Spark, "Apache Spark," [Online]. Available: http://spark.apache.org. [Accessed 10 06 2016].

[14] T. Boujari, "Instance-Based Ontology Alignment Using Decision Trees," Institutionen för Datavetenskap, 2012.

[15] H. F. Witschel, "Using Decision Trees and Text Mining Techniques for Extending Taxonomies," in *In Proceedings of Learning and Extending Lexical Ontologies by Using Machine Learning Methods*, 2005.

[16] W. J. Wei Dai, "A MapReduce Implementation of C4.5 Decision Tree Algorithm," *International Journal of Database Theory and Application,* pp. 49-60, 2014.

[17] R. Mirambicka, A. R. Sulthana and G. Vadivu, Decision Tree Applied to Learning Relations Between Ontologies.

[18] D. Jeon and W. Kim, "Development of Semantic Decision Tree," in *Data Mining and Intelligent Information Technology Applications (ICMiA), 2011 3rd International Conference on*, 2011.

[19] M. Patil, S. Khomane, V. Saykar and K. Moholkar, "Web People Search Using Ontology Based Decision Trees," *International Journal of Data Mining & Knowledge Management Process,* 2012.

[20] B. Panda, J. S. Herbach, S. Basu and R. J. Bayardo, "PLANET: Massively Parallel Learning of Tree Ensembles," Google, 2009.

[21] J. Han, Y. Liu and X. Sun, "A Scalable Random Forest Algorithm Based on MapReduce," in *Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on*, Bejing, China, 2013.

[22] W. Yin, V. Simmhan and V. K. Prasanna, "Scalable Regression Tree Learning on Hadoop Using OpenPlanet," in *MapReduce '12 Proceedings of third international workshop on MapReduce and its Applications*, New York, New York, 2012.

[23] S. d. Río, V. López, J. M. Benítez and F. Herrera, "On The Use of MapReduce For Imbalanced Big Data Using Random Forest," *Information Sciences,* pp. 112-137, 2014.

[24] S. Tyree, K. Q. Weinberger and K. Agrawal, "Parallel Boosted Regression Trees for Web Search Ranking," in *WWW 2011 – Session: Ranking*, NY, NY, 2011.

[25] X. Zhanga, C. Liua, S. Nepalb, C. Yanga, W. Dou and J. Chen, "A Hybrid Approach For Scalable Sub-Tree Anonymization Over Big Data Using MapReduce On Cloud," *Journal of Computer and System Sciences,* pp. 1080-1020, 2014.

[26] J. Ye, J.-H. Chow, J. Chen and Z. Zheng, "Stochastic Gradient Boosted Distributed Decision Trees," in *CIKM '09 Proceedings of the 18th ACM Conference on Information and Knowledge Management*, New York, New York, 2009.

[27] A. Ghoting, P. Kambadur, E. Pednault and R. Kannan, "NIMBLE: a Toolkit For The Implementation of Parallel Data Mining and Machine," in *KDD '11 Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, New York, 2011.

[28] G.-Q. Wu, H.-G. Li, X.-G. Hu, Y.-J. Bi, J. Zhang and W. Xindong, "MReC4.5: C4.5 Ensemble Classification with MapReduce," in *2009 Fourth ChinaGrid Annual Conference*, Yantai, Shandong, 2009.