Intelligent Data Entry Assistant for XML Using Ensemble Learning

Danico Lee
Information and Telecommunication Technology Center
University of Kansas
2335 Irving Hill Rd,
Lawrence, KS 66045, USA
lee@ittc.ku.edu

Costas Tsatsoulis
Department of Electrical Engineering and Computer Science
and
Information and Telecommunication Technology Center
University of Kansas
2335 Irving Hill Rd,
Lawrence, KS 66045, USA
tsatsoul@eecs.ku.edu

ABSTRACT

XML has emerged as the primary standard of data representation and data exchange [13]. Although many software tools exist to assist the XML implementation process, data must be manually entered into the XML documents. Current form filling technologies are mostly for simple data entry and do not provide support for the complexity and nested structures of XML grammars. This paper presents SmartXAutofill, an intelligent data entry assistant for predicting and automating inputs for XML documents based on the contents of historical document collections in the same XML domain. SmartXAutofill incorporates an ensemble classifier, which integrates multiple internal classification algorithms into a single architecture. Each internal classifier uses approximate techniques to propose a value for an empty XML field, and, through voting, the ensemble classifier determines which value to accept. As the system operates it learns which internal classification algorithms work better for a specific XML document domain and modifies its weights (confidence) in their predictive ability. As a result, the ensemble classifier adapts itself to the specific XML domain, without the need to develop special learners for the infinite number of domains that XML users have created. We evaluated our system performance using data from eleven different XML domains. The results show that the ensemble classifier adapted itself to different XML document domains, and most of the time (for 9 out of 11 domains) produced predictive accuracies as good as or better than the best individual classifier for a domain.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning – *parameter learning.*

General Terms

Algorithms

Keywords

XML, Autofill, Ensemble Learning, Machine Learning

1. INTRODUCTION

1.1 Motivation

XML is a markup language for documents containing structured information. It has emerged as the primary standard of data representation and data exchange [13]. It enables people in the same professional discipline to exchange data and information without worrying about the data format and media. Over the past few years, practitioners in a variety of domains have developed their own XML ontologies and applications to annotate their information. A number of XML software tools have also been developed, such as Xerces, Xalan, FOP, Forrest, and Crimson from Apache, and XML Spy from Altova [1]. However, the major weakness of current software tools is that they only simplify the implementation process, but information for XML documents still needs to be manually entered into the XML applications.

On the other hand, software tools that currently have the ability to assist data entry are mostly targeted to web-based forms. They are designed especially for login forms or online shopping forms, e.g. AOL, Roboform, and Google Toolbar. The data for online input form is mostly simple, e.g. login, password, address, and credit card number, and predefined by the user of the autofill application, the specific web site, or in some other location, such as the user's address book.

1.2 Approach

Our work developed an intelligent assistant, SmartXAutofill, that automates the data entry into XML forms. Individual classifiers (predictors) were trained on a historical XML document collection and were combined into an ensemble classifier. The ensemble classifier integrates all the internal classification algorithms into a single architecture. When a user starts to fill out an input form for an XML document, each internal classifier proposes a value, and, through voting, the ensemble determines which value to accept. As the system operates it learns which internal classifiers work better and modifies its weights (confidence) in their predictive ability. As a result, the ensemble adapts itself to the specific XML domain, without the need to develop special classifiers for each XML domain. Our system allows the ensemble classifier to incorporate any type and number of internal classification algorithms. In our experiments, we have used four internal classifiers: a Naïve Bayesian classifier that uses probability techniques, a K-Nearest Neighbor classifier that uses instance-based techniques, a frequency classifier which selects the most frequently occurring value for an XML field, and a recency classifier that simply selects the most recently entered value for a field[1]. The results show that the ensemble classifier adapted itself to eleven different XML document domains, and most of the time (for 9 out of 11 domains) produced predictive accuracies at least as good as or better than the best individual classifier for a domain.

2. RELATED WORK

2.1 Form Filling Tools

Software tools that have the ability to assist in data entry are mostly targeted to web-based forms. AOL populates an input form (with at least one field filled) by searching through a set of template files that resemble the current state of the input form [8]. The template files are usually stored at the user's machine. A match occurs if a template file that resembles the current state is found. Then, the form completion program uses the template file to populate the rest of the input form.

Roboform uses Passcards to store form data that is specifically associated to a particular website [11]. When the user visits a website which has a previously saved Passcard, Roboform will notify the user that there is a Passcard for the current website. If the user decides to autofill the input form, then Roboform will populate the form with the Passcard.

Google Toolbar enables users to automatically complete forms on the web [5]. The users first need to enter their own personal data in the AutoFill Options of the Toolbar, and the Toolbar stores the data on user's own computer. When the user goes to a website with an input form, Google Toolbar will highlight the input fields that are recognized by its AutoFill feature. User can choose to have Google complete the form with the previously stored information. Similar behavior is exhibited by browsers such as Internet Explorer and Safari, which also read information from the user's address book.

The data supported by previous form filling tools is usually simple, e.g. login, password, address, and credit card number and has a linear structure. Also, the user must have already entered and stored the data in some form for it to be accessible. However, XML grammars contain nested composition and complex tree-like structures and the information in them is usually not already stored, but needs to be entered[2]. Not only are previous data entry tools unable to support XML grammars, but also incapable to learn and improve their performances.

It is also important to note that our classification/prediction algorithms are *approximate*, and make suggestions based on an approximate match (probabilistic, syntactic or otherwise) between the values in a historical collection of XML documents and the values in a document that is partially filled and for which they predict the empty node values. This is significantly different from other autofill systems which require a perfect match between the incomplete document and the values and documents already stored.

2.2 Ensemble Learning

Ensemble learning is a machine learning technique that selects a collection, or *ensemble*, of hypotheses from the hypothesis space and combines their predictions [12]. An ensemble consists of a set of independently trained classifiers whose predictions are combined when classifying new instances. The one of the most widely used ensemble methods is called *boosting*. Boosting works with a weighted training set. Each example in the training set has an associated weight $w_j >= 0$. The higher the weight of an example, the higher is the importance attached to it during training. Boosting, produces a series of classifiers, where the training set used for each member of the series is chosen based on the performance of the earlier classifier(s) in the series. All examples start with $w_j = 1$ and boosting increases the weights (importance) of misclassified examples and decreases the weights of the classified examples. Therefore, examples that are incorrectly predicted by previous classifiers in the series are chosen more often than examples that were correctly predicted. As a result, boosting produces new classifiers for its ensemble that are better able to correctly predict examples for which the current ensemble performance is poor [7].

The idea of our ensemble classifier is based on boosting. However, instead of boosting the importance of the examples in the training set, it boosts the internal classifiers based on their past performances through weighting the individual classifiers. Our ensemble learns which internal classifiers work better for a particular XML domain and adapts itself without developing special classifiers for infinite number of XML domains. In addition, previous work in ensemble classifiers and boosting combined the same type of classifier, learned by the same methodology, but trained on different examples. Our work combines different types of classifiers into an integrated classification framework.

3. PREDICTING XML ENTRY DATA USING ENSEMBLE

Experiments we conducted with a variety of approximate classification algorithms used to predict values for XML documents showed that the accuracy of a single classification algorithm differs greatly across nodes, documents, and document

[1] Other classifiers were also tested, including one that used rules derived by C4.5 [10], and KNN with different sizes for K, but they all had either very bad individual predictive accuracy or had individual accuracies similar to other classifiers in the ensemble.

[2] An exception to this are databases which need to be translated into XML. In this case the data already exists, and the population of an XML document involves custom-made (usually) translation programs.

collections due to the structural variability of XML and the specific nature of values used to represent information from the domain in the XML documents (see, for example, Table 1 in this paper). It is impossible to predict which classification algorithm will work best for what type of document. To solve this problem, we developed an ensemble classifier. An *ensemble classifier* is a collection of a number of classification algorithms where each of them provides predictions for the value of an XML node. The ensemble learns which individual algorithms provide better predictive accuracy for different XML domains and for different nodes in the XML documents in these domains. The result is a classifier that adapts itself to the specific XML collection, and performs better than any individual predictive algorithm.

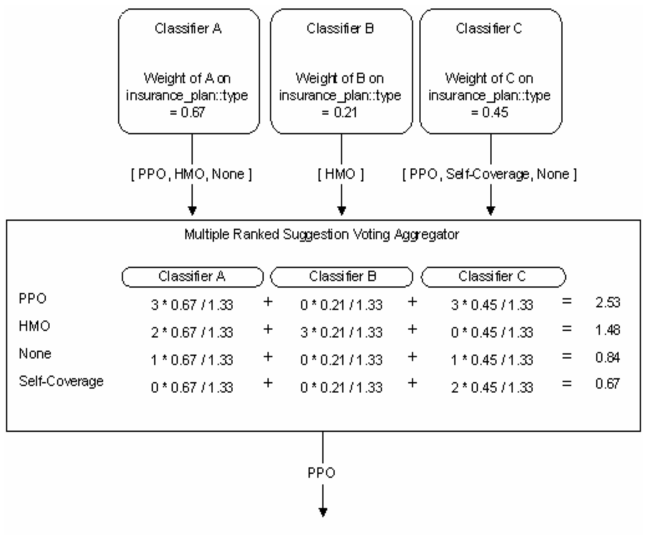
Our ensemble algorithm for predicting XML node values incorporates individual classification algorithms and learns which is most accurate for each node in the collection. Suggestions to the ensemble are made based on an approximate match between data entered in the current input form, compared to data stored in a historical document collection. If many suggestions are made, they are ranked by the internal classifiers based on probable correctness. Each unique node value from the set of suggestions provided by all internal classifiers is voted on and the overall most likely one is selected as the final suggested value. Voting is based on the past performance (accuracy) of the suggested values for the given node in this document collection for each internal classifier. After a suggestion has been made, the weights for each classifier are updated to record the accuracy of the suggestion.

3.1 Suggestion Aggregator - Voting Among Internal Classifiers

Since the ensemble receives predictions from many internal classifiers, voting is used for aggregating these inputs into a single suggestion. The voting system forms a consensus decision on which value is suggested by most of the classifiers. All classifiers return the same number of maximum suggestions. Each classifier's suggestions are ranked; if a classifier returns N suggestions the top one receives a value of N, the second one N-1, and so on, with the last suggestion receiving a 1. In addition to their rank, suggestions are modified by the weight of their classifier. Initially all classifiers have the same weight, but this is modified based on which classifier works better for different nodes in the XML domain (more on this later). An example of the voting mechanism is shown on Figure 1. In this example, three classifiers provide a maximum of three suggestions each. Classifier A makes three suggestions; the top one receives a rank value of 3, the second one of 2, and the third one of 1; the rank values are multiplied by the weight of the classifier (0.67) and then normalized by the sum of the weights of all the classifiers. The same occurs for the suggestions by the other classifiers. The suggestion with the highest support is the one selected by the ensemble and presented to the user.

3.2 Ensemble Learning – Weighting Each Classifier by Past Performance

The weighting system in our ensemble learns from past performance of the internal classifiers. As classification algorithms exhibit different predictive accuracy for different nodes, depending on the number of instances of that node in the collection and the type and statistical distribution of values for the node across the collection, the overall accuracy of the suggestion system is improved by learning how each internal classifier performs on each node and weighting their votes accordingly.


Classifier A
Weight of A on insurance_plan::type = 0.67
Classifier B
Weight of B on insurance_plan::type = 0.21
Classifier C
Weight of C on insurance_plan::type = 0.45
[PPO, HMO, None]
[HMO]
[PPO, Self-Coverage, None]
Multiple Ranked Suggestion Voting Aggregator
PPO

	Classifier A		Classifier B		Classifier C		
PPO	3 * 0.67 / 1.33	+	0 * 0.21 / 1.33	+	3 * 0.45 / 1.33	=	2.53
HMO	2 * 0.67 / 1.33	+	3 * 0.21 / 1.33	+	0 * 0.45 / 1.33	=	1.48
None	1 * 0.67 / 1.33	+	0 * 0.21 / 1.33	+	1 * 0.45 / 1.33	=	0.84
Self-Coverage	0 * 0.67 / 1.33	+	0 * 0.21 / 1.33	+	2 * 0.45 / 1.33	=	0.67

Figure 1. Ensemble classifier: individual, internal classifiers make predictions which are ranked, with the rank multiplied by the classifier weight (confidence). The suggestion with the highest total support is presented to the user.

Our measure of predictive performance of a classification on a node is the error rate of its suggestions for that XML node. Error rate is defined as the number of incorrect suggestions over the total number of suggestions (ignoring cases where the algorithm can make no suggestion)[3]. In other words, the weight for classifier *i* on node *n* for a specific XML domain is:

$$wt_{i,n} = 1 - errorrate$$

or

$$wt_{i,n} = 1 - \left(\frac{s_{i,n..incorrect}}{s_{i,n..total}} \right)$$

For each node, we record the total number of suggestions for that node by each internal classifier and the number of errors in suggestions for that node by each classifier. Since classifiers that make fewer suggestions may be favored (fewer suggestions imply fewer errors), the weight of each predictor is normalized by dividing it by the sum of all weights for all internal classifiers on the node.

$$\text{Normalized } wt_{i,n} = \frac{wt_{i,n}}{\sum_{i=1}^{c} wt_{i,n}}$$

[3] A suggestion is considered to be correct if it was accepted by the user and it was one of the top two suggested by the classifier.

where $wt_{i,n}$ is the weight assigned to the i^{th} classifier when suggesting a value for node *n* and *c* is the total number of internal classifiers.

The weight for each internal classifier on each node is updated after each suggestion is made. If the suggestion is correct, then the $s_{i,n\ total}$ will be incremented. If the suggestion is incorrect, both $s_{i,n\ total}$ and $s_{i,n\ incorrect}$ will be incremented.

Initially the values suggested by each internal classifier are weighted equally. If the user accepts the value, the internal classifiers that suggested it get rewarded by increasing their weight. If the user rejects the value, the internal classifiers get punished by decreasing their weight. As a result, our system adapts to the specific domain of XML documents and increases its prediction performance.

4. EXPERIMENTS AND RESULTS

In our experiment we used four simple prediction algorithms inside an ensemble which learned predictor weights for each node for each XML domain as described above. Each classifier used approximate methods and the values entered in an XML form to predict the empty valued nodes in the same form. As more nodes were filled in, the predictions changed.

The value v_i of a node n_i could be predicted using traditional probability theory as the conditional probability of $n_i=v_i$ given the values entered in all other nodes: $P(n_i=v_i \mid n_1=v_1, n_2=v_2, \ldots, n_{i-1}=v_{i-1})$. Since this is computationally infeasible in a real-time application such as an autofill system, the first classifier we used was based on Naïve Bayes theory. Naïve Bayes analyzes the relationship between each independent and dependent variable to derive a conditional probability for each relationship. When a new case is analyzed, a prediction is made by combining the effects of the independent variables on the dependent variable (the outcome that is predicted). The Naïve Bayesian Classifier computes the probability of a value for an empty node, based on the conditional probabilities of the predicted value given the actual values of the nodes that have been filled in. In other words, given the values of *i-1* fields: $v_1, v_2, \ldots, v_{i-1}$, and given a possible v_i for the value of the empty node n_i, the Naïve Bayesian Classifier computes the probability that v_i is predictably correct as follows:

$$P(n_i=v_i)=\frac{\prod_{k=1}^{i-1}P(n_i=v_i \mid n_k=v_k)}{\prod_{k=1}^{i-1}P(n_i=v_i \mid n_k=v_k)+\prod_{k=1}^{i-1}\left(1-P(n_i=v_i \mid n_k=v_k)\right)}$$

The second classifier was a simple K-Nearest-Neighbor, where K=1 (our previous experiments showed no significant difference in predictive accuracy when using K=3, 5, or 7). The third classifier was based on frequency and suggested the most frequently used value for an XML node. The final predictor simply suggested the most recently entered value for a node.

Although it would be possible to include other commonly used classifiers such as artificial neural networks or support vector machines, we chose not to in this experiment because such classifiers require retraining as the document collection changes. This would place a computational burden on the system and slow it down considerably; when response time is a critical requirement, this would make the system inefficient to the user. It might be possible to have such classifiers retrained off-line (for example, when the system is idle), but this was not tried in our current experiments. Our current classifiers use the new values and new documents added to the collection without requiring a complete retraining.

The goals of the experiments were the following:

a. establish whether the ensemble could be trained to select the best classification algorithm for each node in each XML document domain

b. establish whether the performance of the ensemble for each XML domain was at least as good as the performance of the best individual classifier for that domain

c. determine accuracy and speed of classification

4.1 Data Selection

A variety of XML domain document data were gathered to span the size and complexity dimensions. All data used for the experiment was collected on the Internet, except that data from one domain was collected directly from the Center for Army Lessons Learned. XML samples have been collected from the following domains: thesaurus by Australian Public Affairs Information Service (APAIS) [2], Bio Medical journal papers (BioMed) [3], Center for Army Lessons Learned (CALL), a comprehensive protein database (iProClass) [6], a protein database by Protein Information Resource (PSD) [14], astronomical data from NASA (NASA) [14], a non-redundant reference protein database (NREF) [9], a protein knowledgebase for a protein sequence database maintained by the Swiss Institute for Bioinformatics and European Bioinformatics Institute (SPROT) [14], a protein sequence databank (UniProf) [4], course listing of the University of Wisconsin at Milwaukee (UWM) [14] and course listing of Washington State University (WSU) [14]. Their size ranged from around 50 to 5000 documents, and they had between 20 and 420 nodes per document.

4.2 Experimental Approach

Before an experiment was run, we seeded a historical collection of XML documents with 10 documents; this is required since our algorithms use historical information to make predictions. The classification algorithms made no suggestion for a particular field if there were no historical data for it or if every previous value for the field was unique, for example, nodes that contain abstracts of reports or papers, collections where a single document is produced daily and where the date node will be unique. Clearly if no information exists it is better not to make a suggestion than to make a wrong suggestion.

XML documents usually have hundreds of elements. In order to exhaustively test the performance of predicting values on a single document, the experiment would have to request suggestions for every possible field in every possible state of an XML input form. For instance, if an input form has one hundred elements, the number of suggestions required to exhaustively test the possible predictions on every state of form completion is one hundred factorial. Since exhaustive testing is not possible, the experiment performed repeated trials. In each iteration, documents were randomly selected and all elements were suggested in a single random order. In other words, a node was selected randomly, the classifier made a prediction, the suggested value was compared to the actual value, the error rate was modified as required, and then the correct value was filled in. Next, another random node was selected and the process repeated.

We ran tests after training on 10, 20 or 40 documents, and tested on 80 documents. As the documents are completed and added to the historical database, the performance of the algorithms changes, but not in a significant manner.

Table 1. Accuracy for Naïve Bayes, KNN, Frequency and Recency Classifiers. The best results for each trial are shown in bold face.

XML Domain	*No of Training Doc*	*Accuracy (%)*			
		Naïve Bayes	*KNN*	*Freq*	*Recency*
APAIS	10	51.10	50.03	***51.48***	48.87
	20	50.79	50.02	***51.00***	48.91
	40	50.86	49.86	***51.00***	48.87
BioMed	10	11.80	10.45	***13.84***	12.95
	20	11.54	10.22	***14.15***	12.59
	40	12.49	10.37	***15.17***	12.35
CALLS	10	97.06	98.53	98.53	***100***
	20	97.06	98.53	98.53	***100***
	40	97.06	97.06	97.06	***100***
iProClass	10	29.21	***37.59***	37.23	34.61
	20	28.08	***37.38***	36.92	34.14
	40	30.02	36.78	***36.84***	34.39
NASA	10	15.25	18.43	22.17	***30.85***
	20	15.04	19.00	21.42	***30.47***
	40	14.61	19.13	21.09	***30.39***
NREF	10	69.54	80.65	***81.36***	79.06
	20	71.23	81.62	***82.35***	80.05
	40	70.59	81.86	***82.35***	79.90
PSD	10	17.98	23.23	22.61	***23.85***
	20	16.87	22.34	22.35	***23.62***
	40	16.01	20.87	22.01	***23.57***
SPROT	10	8.39	10.98	12.72	***15.20***
	20	8.90	10.56	12.72	***15.16***
	40	8.89	9.84	12.41	***15.05***
UniProf	10	***37.05***	34.51	35.44	16.74
	20	***37.00***	33.84	35.69	16.77
	40	***36.35***	33.84	35.64	16.67
UWM	10	26.69	25.01	28.21	***48.05***
	20	26.45	24.82	28.26	***48.06***
	40	27.10	24.70	29.24	***48.05***
WSU	10	31.46	28.29	33.45	***43.53***
	20	31.70	28.66	33.86	***43.58***
	40	31.81	28.72	33.94	***43.65***

Table 1 summarizes the results of the individual internal prediction algorithms (to the ensemble) with an initial seed of 10 documents, and then trained by 10, 20 or 40 other XML documents. The accuracy value is computed as the total number of correct predictions made on all nodes for all documents for a total of 80 test documents, over the total number of predictions made[4].

In bold face we are showing the best results for each trial. As can be seen, different predictive algorithms perform better for different domains, although we did not study whether differences were statistically significant. In some cases the differences seem insignificant (e.g. CALLS and APAIS collections), while in other cases they are quite large (e.g. WSU and UWM collections).

For testing the ensemble classifier, the document collections were separated into three sets: seed (10 documents as before), training collection and testing collection. *Training collection* was used to train the ensemble by modifying its weights. The ensemble randomly picked out a document from the training collection, received suggested values from the internal classifiers and modified the classifier weights based on the accuracy of the suggestion. The documents from the training collection, once used for training, were added to the seed and were used in future predictions (but not in testing). As before, 10, 20 or 40 documents were used for training the ensemble.

After training, the weights were frozen and the ensemble was tested on 80 documents in the same way as the individual classifiers. These documents formed the *testing collection.* Accuracy was calculated as before.

Table 2 shows the results of the ensemble classifier as compared to the results of the best individual classifier for each domain and number of training documents. In bold face we are showing which classifier worked better. Note that the ensemble classifier worked as well as or better than the best individual predictor in 9 out of 11 domains.

In addition to accuracy we studied whether the weights for each classifier for each node for an XML domain would change inside the ensemble during training. This was observed for all experiments. For example, the weights of some nodes from the iProClass XML domain are shown in Table 3 (iProClass documents had a maximum of 47 nodes).

Table 2. Comparison of the accuracy for the best individual, internal classifier for each XML test domain versus the ensemble classifier. The best results are shown in bold face.

XML Domain	*No of Training Doc*	*Best Classifier*		*Ensemble Accuracy (%)*
		Name	*Accuracy (%)*	
APAIS	10	Freq	51.48	***52.13***
	20	Freq	51.00	***51.19***
	40	Freq	51.00	***51.90***

[4] Since some collections had few documents, documents might be used more than once in testing iterations, but since the suggestions are made in random order and prediction is based on the random state of the form, they were considered separate trials.

BioMed	10	Freq	13.84	***15.95***
	20	Freq	14.15	***15.95***
	40	Freq	15.17	***16.8***
CALLS	10	Recency	100	***100***
	20	Recency	100	***100***
	40	Recency	100	***100***
iProClass	10	KNN	37.59	***38.69***
	20	KNN	37.38	***37.89***
	40	Freq	36.84	***37.46***
NASA	10	Recency	***30.85***	28.07
	20	Recency	***30.47***	26.07
	40	Recency	***30.39***	27.36
NREF	10	Freq	81.36	***81.50***
	20	Freq	82.35	***82.55***
	40	Freq	82.35	***82.55***
PSD	10	Recency	23.85	***27.10***
	20	Recency	23.62	***26.36***
	40	Recency	23.57	***25.26***
SPROT	10	Recency	15.20	***17.70***
	20	Recency	15.16	***17.40***
	40	Recency	15.05	***17.54***
UniProf	10	NB	37.05	***37.58***
	20	NB	37.00	***37.60***
	40	NB	36.35	***37.45***
UWM	10	Recency	***48.05***	44.57
	20	Recency	***48.06***	44.51
	40	Recency	***48.05***	45.09
WSU	10	Recency	43.53	***45.82***
	20	Recency	43.58	***45.84***
	40	Recency	***43.65***	45.09

As can be observed from Table 3, different classifiers are preferred for different nodes (Naïve Bayes for Node 13 and Node 28, KNN for Node 22, Frequency for Node 14, and Recency for Node 6 and Node 7), and the weights have been modified from their initial value of 1, through the training process.

Table 3. Weights of selected XML nodes from iProClass domain.

NodeId	*Naïve Bayes*	*KNN*	*Freq*	*Recency*
6	0.4146	0.5805	0.4829	***0.7724***
7	0.6878	0.7528	0.5772	***0.8423***
13	***0.1404***	0.1066	0.1184	0.0880
14	0.0100	0.0100	***0.0186***	0.0100
22	0.0449	***0.1124***	0.0375	0.0637
28	***0.2543***	0.1890	0.2474	0.0825

Finally, we were interested in the time performance of the ensemble as any autofill system needs to be faster than the user and must work in real-time. Since the performance of the ensemble ranged between 1 and approximately 60 milliseconds, we believe that the real-time requirement is satisfied.

4.3 Discussion

In our experiments we expected that the ensemble classifier would work at least as well as the best performing internal classification algorithm for a domain, and also that for different domains it would match the performance of different internal classifiers. This was the case in 9 out of the 11 domains where we tested our algorithms. We currently have no explanation as to why the ensemble did worse than the best algorithm in the UWM and NASA domains (although in both cases it did significantly better than the second-best classifier); it is possible that more training of the ensemble would have improved its performance in these two cases.

An interesting observation is that the ensemble learns to prefer different algorithms for different nodes in the same domain, leading to adaptation and learning at the lowest possible level. This would explain why it performed better than the best individual predictor in 9 domains: the ensemble learns the best algorithm per node instead of per domain, while individual, internal algorithms make predictions based on the whole document in a domain.

An unexpected result was that the best individual predictive algorithms were most often the simplest ones: recency and frequency, while more complex algorithms were seldom the better ones, and often had much lower accuracy.

5. CONCLUSIONS

In this paper we presented an ensemble classifier that assists the data entry into XML documents by predicting the values of XML nodes. The ensemble integrates individual classification algorithms and learns which ones predict the values of XML nodes more accurately. Our approach is different from current autofill technologies, since our internal classifiers use approximate predictive techniques (e.g naïve conditional probability or partial syntactic match). Current autofill technologies, on the other hand, require a perfect match between the data in the document being completed and some document in memory. Our ensemble also differs from traditional ensemble classifiers, since it integrates disparate classification and prediction algorithms, instead of algorithms induced by the same methodology through boosting.

Our experiments showed that the ensemble adapted itself to a variety of XML domains, learned the best predictive algorithm for a node in an XML domain, and in 9 out of 11 test domains performed as well as or better than the best individual predictive algorithm. The accuracy achieved ranged greatly from domain to domain, with lowest predictive accuracy of 15.95% and best accuracy of 100%. The average accuracy of the ensemble was 44.3% and the median accuracy 37.9%. The system performed in real-time, with worst performance of approximately 60 milliseconds, satisfying the requirement that the autofill be faster than the user.

6. FUTURE WORK

Now that we have verified that the ensemble approach works, we intend to study which classifiers to include in the ensemble to improve its lower end performance. Other possible extensions include tweaking the rewards formula for the ensemble, and exploring more complex classifiers such as Neural Networks or Support Vector Machines that may require off-line retraining. We will also study the use of ensemble-based prediction in different applications including web forms and databases. Finally, it will be interesting to study why simple prediction techniques seem to perform much better than complex ones, and whether this is a feature constrained to specific XML domains, or a more general feature of XML documents.

7. ACKNOWLEDGEMENTS

This work was supported in part by the Kansas Technology Enterprise Corporation.

8. REFERENCES

1. Altova, “XML Spy”, URL < http://www.xmlspy.com/>, Sept 17, 2004.
2. “APAIS Thesaurus,” URL <http://www.nla.gov.au/apais/thesaurus/about.html>, Oct 1, 2003.
3. “BioMed Center – The Open Source Publisher,” URL <https://www.biomedcentral.com/home/>, Sept 24, 2003.
4. “European Bioinformatics Institute,” URL <http://www.ebi.ac.uk/trembl/index.html>, June 22, 2004.
5. “Google Toolbar,” URL <http://toolbar.google.com/>, Sept 8, 2004.
6. “iProClass – An Integrated Protein Classification Database”, URL< http://pir.georgetown.edu/iproclass/>, June 18, 2004.
7. Maclin, R. and Opitz, D. “An Empirical Evaluation of Bagging and Boosting,” *Fourteenth National Conference on Artificial Intelligence (AAAI-97),* AAAI Press, 546-551.
8. Maxwell, D., von Reis, W. and Scott, G., “Method and Apparatus for Populating a Form with Data,” *US Patent 6,589,290,* July 8, 2003.
9. “PIR Information Resource,” URL <http://pir.georgetown.edu/pirwww/>, June 15, 2004.
10. Quinlan, J. R. Induction of Decision Trees. *Machine Learning, 1,* 81-106. 1986.
11. “RoboForm: Free Password Manager, Form Filler, Password Generator: Fill Forms, Save Passwords, AutoFill, AutoLogin, AutoSave,” URL <http://www.roboform.com/>, Sept 8, 2004.
12. Russell, S. and Norvig, P. Artificial Intelligence A Modern Approach, Second Edition. Prentice Hall, 664-668.
13. Su, H., Kramer, D., Chen L., Claypool, K. and Rundensteiner, E. A. “XEM: Managing the evolution of XML Documents”. Eleventh International Workshop on Research Issues in Data Engineering, IEEE Computer Society Press, 103-110.
14. “University of Washington XML Repository,” URL <http://www.cs.washington.edu/research/xmldatasets/www/repository.html>, June 15, 2004.