# A Maximum Likelihood Framework for Integrating Taxonomies

**Suju Rajan, Kunal Punera, and Joydeep Ghosh**
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas-78712
{rsuju, kunal, ghosh}@lans.ece.utexas.edu

**Abstract**

Many approaches have been proposed for the problem of mapping categories (classes) from a source taxonomy to classes in a master taxonomy. Most of these techniques, however, ignore the hierarchical structure of the taxonomies. In this paper, we propose a maximum likelihood based framework which exploits the hierarchical structure to obtain a more natural mapping between the source classes and the master taxonomy. Furthermore, unlike previous work, our technique also inserts source classes into appropriate places of the master hierarchy creating new categories if required. We evaluate our approach on text and hyperspectral datasets.

## Introduction

Hierarchical taxonomies can provide a wealth of information about the domain they are constructed upon. Typically, the hierarchies are arranged in a general-to-specific fashion with the root-node accounting for all the classes in the taxonomy while the leaf-nodes correspond to specific classes of the taxonomy. In general, taxonomies group similar classes closer together in the tree thereby revealing the inherent relationships between the classes or the meta-classes (sets of classes). Gene expression analysis (Segal, Koller, & Ormoneit 2001), hyperspectral analysis (Kumar, Ghosh, & Crawford 2002), web directories (Dumais & Chen 2000), and product catalogs (Agarwal & Srikant 2001) are examples of domains with well-defined hierarchical taxonomies that not only represent the relationships between the classes in a structured way, but also help improve the generalization of classifiers built over those classes.

The decentralized nature of data collection and proprietary issues often result in multiple taxonomies being defined for the same domain. For instance, the web directories of Yahoo![1] are different from that of DMOZ[2] and the product catalogs of Amazon[3] are different from that of Ebay[4]. Often it might be necessary to integrate the classes from one taxonomy (let us call it a source taxonomy) into an existing "master taxonomy". In the absence of uniform labeling of classes, this integration is non-trivial. Given a master taxonomy $\mathcal{M}$ and a source taxonomy $\mathcal{S}$, defined on the same domain, we believe any integration algorithm should be able to handle the following four scenarios:

1. If certain classes in $\mathcal{S}$ have equivalent classes in $\mathcal{M}$, then they should map directly. For instance, a class called *Cars* in $\mathcal{S}$ should map to a class called *Autos* in $\mathcal{M}$.
2. $\mathcal{S}$ may contain some classes that represent concepts that are unseen in $\mathcal{M}$. For instance, $\mathcal{S}$ may contain the latest cars to hit the market which do not have counterparts in $\mathcal{M}$.
3. Some class in $\mathcal{S}$ may be a union (superset) of some classes in $\mathcal{M}$. For instance, $\mathcal{S}$ may contain a class called *Motor-Vehicles*, while $\mathcal{M}$ contains the classes *Cars*, *Trucks*, and *Bikes*.
4. Multiple classes in $\mathcal{S}$ may belong to a single class in $\mathcal{M}$. For instance, the taxonomy $\mathcal{S}$ may be at a higher level of granularity with classes like *Cars*, *Trucks*, and *Bikes*, while $\mathcal{M}$ only contains a single *Motor-Vehicles* class.

Existing techniques invariably treat taxonomy integration as the problem of finding a 1-to-1 mapping between the classes in the two taxonomies. For each class in $\mathcal{S}$ they attempt to find the class in $\mathcal{M}$ that is 'most similar' (Agarwal & Srikant 2001; Sarawagi, Chakrabarti, & Godbole 2003; Zhang & Lee 2004; Doan *et al.* 2002). Moreover, most of these methods either completely ignore the hierarchical structure of taxonomies or do not consider mappings between classes and internal nodes in the hierarchies. The only way a more general class can be integrated into a finer grained master taxonomy is by mapping the general class to one of the internal nodes. Furthermore, all of the existing techniques only seek to obtain a mapping for the source taxonomy classes into the master taxonomy. No guidance is given on how best to update the master taxonomy with the new data. For example, when handling a situation depicted in Scenario 4, mapping certain finer-grained classes in $\mathcal{S}$ to a single class in $\mathcal{M}$ causes one to lose the distinction between those source classes.

In this work, we propose a framework that not only maps, but completely integrates the classes in $\mathcal{S}$ into $\mathcal{M}$. Given



[1] http://www.yahoo.com
[2] http://dmoz.org
[3] http://www.amazon.com
[4] http://www.ebay.com

a hierarchical master taxonomy, our framework attempts to insert the classes of $\mathcal{S}$ into it, such that the resulting tree continues to maintain 'similar' classes closer in the hierarchy. The requirement of a master hierarchy is not stringent as the generation of hierarchies from flat sets of classes can be easily automated (Kumar, Ghosh, & Crawford 2002; Vural & Dy 2004). In order to integrate source classes that are at a higher abstraction level than the master taxonomy classes, classes in $\mathcal{S}$ are split prior to integration. Thus, parts of a source class may appear in multiple leaf-nodes of the resulting hierarchy. Furthermore, when multiple classes from $\mathcal{S}$ map into the same class in $\mathcal{M}$, we create a new hierarchy that maintains the distinction between these classes.

## Related Work

Past work can broadly be divided into two parts, techniques that consider the hierarchical structure of the taxonomies and those that ignore this structure. A simple approach for mapping the classes from $\mathcal{S}$ to $\mathcal{M}$, while ignoring the hierarchical structure of both $\mathcal{S}$ and $\mathcal{M}$, is to build a Naïve Bayes classifier that estimates the posterior probability of a class in $\mathcal{M}$ given a data-point in $\mathcal{S}$. Class information from $\mathcal{S}$ can then be used to form simple rules such as, if more than $x\%$ of data-points from source class $S_i$ were classified as master class $M_j$ then $S_i$ is 'most similar' to $M_j$, etc. The Enhanced Naïve Bayes (E-NB) technique (Agarwal & Srikant 2001) works on similar principles and uses the class information in $\mathcal{S}$ to shift the classification threshold of the Naïve Bayes classifier (Zhang & Lee 2004). Cross-training (Sarawagi, Chakrabarti, & Godbole 2003) and Co-Bootstrapping (Zhang & Lee 2004) approaches also attempt to classify the data-points in $\mathcal{S}$ into $\mathcal{M}$ by using the data in $\mathcal{S}$ to enhance the classifiers in $\mathcal{M}$. These methods output the mapping of classes from $\mathcal{S}$ to $\mathcal{M}$, but do not provide any clues for integrating the mapped source classes into the master taxonomy. Our approach does not ignore the hierarchical structure of the master taxonomy and outputs an updated master tree with the source classes placed at appropriate positions.

The GLUE framework (Doan *et al.* 2002) also performs a 1-to-1 mapping between the source and master taxonomy classes, but uses the hierarchical information to assign labels to the nodes using the notion of 'relaxation labeling'. The key idea is that the label assigned to a node is influenced by the labels of its neighboring nodes. While GLUE takes the hierarchy into account, source classes are mapped only to the leaf nodes of the master hierarchy. Hence, problems that involve Scenario 3 cannot be handled by GLUE. In contrast, our framework allows source classes to be mapped into any node (internal or leaf) in the master tree.

Another method proposed in (Ichise, Takeda, & Honiden 2003) explicitly requires a hierarchical structure to be defined on both $\mathcal{M}$ and $\mathcal{S}$. The algorithm outputs a list of 'alignment rules' between the nodes in the two hierarchies by using the $\kappa$-statistic to decide whether the amount of object overlap between the nodes is high enough to consider the two nodes as identical. This method requires a significant number of common data instances between the two taxonomies being merged. However, this requirement will be a severe drawback in those domains where such data is unavailable. For example, when integrating taxonomies defined on hyperspectral images, the data points are individual pixels represented as vectors of real numbers. It is impossible to identify a set of objects that co-occur in two different taxonomies in this domain. Moreover, even in discrete domains like product catalogs, it is not clear how such data would be made available. Another limitation of this approach is that it requires the criteria used to generate the splits in the two hierarchies to be similar. Hence, taxonomies with orthogonal classifications cannot be merged using this method. Our proposed framework considers a model based approach to discovering similar classes in the two taxonomies and does not require overlapping data. Further, orthogonal classifications in the two taxonomies does not affect our method as we integrate a *flat set of source classes* into an existing master hierarchy.

Other related work lies in the domain of 'ontology matching' in which the hierarchical structure of the ontologies are used along with a number of heuristics to merge the elements of the ontologies. Some of the more popular ontology matching methods are the Chimaera (McGuninness *et al.* 2000), FCA-MERGE (Stumme & Maedche 2001), and PROMPT (Noy & Musen 2000), all of which require a lot of human interaction.

## Exploiting Hierarchical Taxonomies for Integration

A hierarchical taxonomy can be used as a classifier in which a multi-class problem can be broken down into a set of simpler problems. If the hierarchies are well-defined, each subproblem will be simpler than the original one and would also typically require a smaller set of features to resolve it (Koller & Sahami 1997). Hierarchical classifiers have been shown to give slight improvements in classification accuracies over using a flat set of classes (Segal, Koller, & Ormoneit 2001) (Dumais & Chen 2000). Another advantage of using hierarchies is that when labeled data is scarce, shrinkage techniques (McCallum *et al.* 1998) can be used to improve the parameter estimates at a node by using the corresponding estimates of its parent nodes. Hence, it is beneficial to preserve the hierarchical information while integrating the source classes into the master tree.

Most real-world hierarchies have some inherent constraints on the relationships between its nodes. For instance, the children of a node may be required to be 'more similar' to it than its siblings. Once such constraints in the hierarchy have been identified, the problem of taxonomy integration can be solved in two stages.

1. **Mapping Stage:** Given a flat set of source classes that are to be merged into the master hierarchy, a suitable parent node has to be identified for each source class.
2. **Integration Stage:** Once the parent nodes have been identified, the source classes have to be integrated into the existing hierarchy such that the resulting hierarchy continues to satisfy its constraints.

Let $\{S_i\}_{i=1}^n$ be the set of source classes from a source taxonomy $\mathcal{S}$ and let $\{X_i\}_{i=1}^n$ be the corresponding set of data

points associated with each source class. Let $\{M_j\}_{j=1}^{m}$ be the set of master classes from the master taxonomy $\mathcal{M}$ and let $T$ be a hierarchy defined on these classes. $T$ therefore has $m$ leaf nodes each of which corresponds to a class from $\{M_j\}_{j=1}^{m}$. Given the data associated with the master taxonomy, we use it to define the set of data points ($\mathcal{D}_v$) associated with each node $v$ of $T$. If $v$ is an internal node, the set of data points $\mathcal{D}_v$ at $v$ and the pdf ($p_v$) estimated from the $\mathcal{D}_v$ can be used to place the following constraint on $v$:

$$\mathcal{L}(\mathcal{D}_v|p_{c_i}) > \mathcal{L}(\mathcal{D}_v|p_{s_j}) \; \forall c_i \in Child(v) \; and \; \forall s_j \in Sib(v) \quad (1)$$

where $Child(v)$ and $Sib(v)$ denote the child and sibling nodes of $v$ and $\mathcal{L}(\mathcal{D}_{v_1}|p_{v_2})$ is the mean log-likelihood of the data points $\mathcal{D}_{v_1}$ given the pdf $p_{v_2}$, i.e.,

$$\mathcal{L}(\mathcal{D}_{v_1}|p_{v_2}) = \frac{1}{|\mathcal{D}_{v_1}|} \sum_{\forall d \in \mathcal{D}_{v_1}} log(p_{v_2}(d)) \quad (2)$$

Constraint 1 ensures that the children of a node are 'closer' to it than its siblings. Given the above constraint, if the relationships between the source and the master classes are known **apriori** the mapping of each source class to a node in $T$ is done as detailed below. We will later present an algorithm that can handle all the scenarios even when there is no prior knowledge about the relationships.

## Mapping Stage:

**Scenarios 1 and 4:** Under Scenario 1 or 4, the taxonomy integration problem reduces to that of classification. The only difference between the two scenarios is that, while there is a 1-to-1 mapping between the master and source classes under Scenario 1, Scenario 4 implies a many-to-1 mapping from the source to the master taxonomy. In both cases, the leaf node $v^*$ in $T$ that is closest to a class $S_i$ is obtained using:

$$v^* = \underset{\forall v \in T_L}{argmax} \; \mathcal{L}(X_i|p_v) \quad (3)$$

where $T_L$ is the set of leaf nodes of the hierarchy.

**Scenario 2:** Let $S_i$ be a source class that is 'different' from all the master classes. Using (3) will still identify a $v^*$ but inserting $S_i$ at $v^*$ might violate (1). The likelihood of $v^*$ given its siblings is then used as a threshold to check if $S_i$ can be inserted as a child of node $v^*$. $S_i$ can be inserted at $v^*$ if:

$$\mathcal{L}(\mathcal{D}_{v^*}|p_{X_i}) >= \underset{\forall v \in Sib(v^*)}{max} \mathcal{L}(\mathcal{D}_{v^*}|p_v) \quad (4)$$

$p_{X_i}$ represents the pdf estimated from the data associated with the source class $S_i$. When there are multiple $v^*$s, the best one is identified by combining (3) and (4) as follows:

$$v^* = \underset{\forall v \in T}{argmax} \; \mathcal{L}(X_i|p_v)$$

*subject to the constraint*

$$\mathcal{L}(\mathcal{D}_{v^*}|p_{X_i}) >= \underset{\forall v \in Sib(v^*)}{max} \mathcal{L}(\mathcal{D}_{v^*}|p_v) \quad (5)$$

While an exhaustive search can be performed over all the nodes in $T$ to find the $v^*$ that satisfies the above constraint, a greedy approach is to first identify a suitable $v^*$ using (3). If $v^*$ does not satisfy (4), the parent node of $v^*$ is used as the new $v^*$ and constraint (4) is checked again. This process is repeated until either a $v^*$ that satisfies the constraint is found or the root-node is reached. Typically, $v^*$ will be set to the root-node if $S_i$ represents a class that is 'very different' from all the classes in $\mathcal{M}$.

**Scenario 3:** Finally, let $S_i$ be a source class that is a superset of some of the master classes. In this case, subsets of $X_i$ have to be inserted at different nodes of the master tree. A hierarchical classifier, defined on the master classes, is first used to create subsets of $X_i$ as follows: apply the classifier to $X_i$, if more than a user-defined number (say $\theta$) of $X_i$ share the same class label, flag that subset as a new source class say $S_i^j$. A suitable node $v^*$ to insert $S_i^j$ is then identified by using (5). While setting $\theta$ to a very low value might flag arbitrary subsets of $X_i$ as new classes, setting it to a very high value will not allow $X_i$ to be split at all. In our experiments, we used the size of the smallest master class to set $\theta$ (20% of the smallest master class).

Note that extending the algorithm to find the mapping for source classes under Scenario 3 has generalized the algorithm such that it can handle all four scenarios even when there is **no prior** knowledge about the relationships between the source and the master classes. Further, since our approach merges subsets of the source classes into the master tree, even those source taxonomies with vastly different categorization policies can be easily integrated into the existing hierarchy. However, we still need to ensure that the subsets created by the classifier are meaningful and are not the consequence of some artifact of the classifier. The following pseudocode provides details of our proposed framework:

**Algorithm:**

1. For each source class $S_i$, apply the classifier to the corresponding $X_i$. Let $X_i^j$ represents the subset of data-points of source class $i$ that were classified into master class $M_j$. Let $list\_master_i$ be an empty list corresponding to $S_i$.
2. For each $j$ s.t., $|X_i^j| > \theta$. Compute the mean log-likelihood $\mathcal{L}(X_i^j|p_j)$ where $p_j$ is the pdf associated with the leaf-node that represents class $M_j$ in $T$.
3. Starting with the subset $X_i^j$ which has the largest mean log-likelihood, use (5) to identify a suitable $v^*$ (not necessarily a leaf node) to insert $X_i^j$ into.
4. If $list\_master_i = \emptyset$, add $v^*$ to $list\_master_i$ and process the next subset of $X_i$.
5. If $list\_master_i \neq \emptyset$ and $v^* \in list\_master_i$, this case occurs when $X_i^j$ ends up at the same node as some previously processed subset of $X_i$. Such a scenario implies that subsets of $X_i$ have been created due to some artifact of the classifier. Hence, the current subset $X_i^j$ of $X_i$ is combined with the previously seen subset of $X_i$ that had also arrived at $v^*$. A suitable $v^*$ is then identified for this combined data.

6. If $list\_master_i \neq \emptyset$ and $v^*$ is not in $list\_master_i$. In this case, the classifier implies that the subset $X_i^j$ is distinct from any of the previously seen subsets of $X_i$. To test the validity of the split $X_i^j$, we define a new hierarchy $T_{new}$ to be the same as the master hierarchy $T$, except that we also insert the previously seen subsets of $X_i$ as children of their corresponding parent nodes in $T$. We then attempt to find the suitable node (say $v_{new}$) in $T_{new}$ for $X_i^j$. If $v_{new}$ maps into one of the newly introduced child nodes in $T_{new}$ we consider the split $X_i^j$ to be an artifact of the classifier and deal with it as in Step 5. Else if $v_{new} = v^*$, we consider $X_i^j$ to correspond to a new source class and we add $v^*$ to $list\_master_i$.

Thus, each $list\_master_i$ represents a 'mapping' of the corresponding $S_i$ to a set of nodes in $T$. Note that, by appending the source class labels of each $S_i$ to the master tree nodes in the corresponding $list\_master_i$, our framework can obtain additional 'meta-descriptors' for the data points at those nodes.

### Integration Stage:

Once suitable parent nodes have been identified for all the source classes, they need to be integrated into the master hierarchy while making sure that the resulting tree structure continues to satisfy the constraints. Let us consider a node $v^*$ of the hierarchy and the set of source classes that map into it.

**$v^*$ is an internal node:** If $v^*$ is an internal node having a single source class $S_i$ associated with it, then $S_i$ is made a child of $v^*$. If a set of source-classes are associated with an internal $v^*$, the corresponding source classes are first arranged into a new hierarchy, the root node of which is then made the child of $v^*$.

**$v^*$ is a leaf node:** If $v^*$ is a leaf node which has a single source class $S_i$ associated with it, there might either be a 1-to-1 correspondence between $S_i$ and the master class at $v^*$ or $S_i$ might actually be a distinct sub-class of the master class at $v^*$. To distinguish between the above two cases, we propose partitioning $\{D_{v^*} \cup X_i\}$ into two clusters. Cluster validation techniques (Jain & Dubes 1988) are then used to decide if the resulting clusters are to be merged prior to being made the children of $v^*$. A similar procedure is adopted if there are multiple source classes associated with a leaf node $v^*$, except that a hierarchy of the resulting clusters is generated and inserted at $v^*$.

## Experimental Evaluation

In this section we describe the methodology employed to objectively evaluate the mapping stage of our algorithm. We define test cases that cover the different possible scenarios. After applying the algorithm on the test cases, we report the number of test cases it passed for each scenario.

### Test case generation

The first set of test cases was generated by duplicating a leaf node $v$ corresponding to a master class $M_j$ as a source class $S_i$. In order to pass the test, the algorithm must correctly map $S_i$ to the node $v$ in the master hierarchy $T$. This means that the algorithm besides not splitting the source class data $X_i$, should also identify $v$ as the node in the hierarchy that satisfies the constraint (5). Obviously, the total number of test cases that can be generated from the leaf nodes of the master hierarchy is equal to the number of master classes. The performance of our algorithm is evaluated by generating all possible test cases and reporting the fraction of test cases our algorithm passed. Note that testing our algorithm with these test cases is equivalent to testing it under Scenario 1.

The second set of test cases was generated by removing each node $v$ in $T$ and treating it as a new source class $S_i$. When $v$ is an internal node, the entire subtree below it is also moved along with it. In other words, all data points in the subtree rooted at $v$ are placed into $S_i$. The test case is passed if the algorithm identifies that $S_i$ is not present in $\mathcal{M}$ and maps it to an *internal node* of $T$. The number of possible test cases is equal to the number of non-root nodes in the taxonomy. Generating these source classes is equivalent to simulating Scenario 2. Our evaluation measure however underestimates the performance of our algorithm, since sometimes a removed node might have been very close to its ex-sibling. On being re-integrated into $T$ such a node would find its appropriate place with its ex-sibling (at a leaf node of $T$), but we would count that as a mistake.

The next set of test cases was generated by duplicating a non-leaf node $v$ of $T$, as a source class $S_i$ in $\mathcal{S}$. This ensures that the class $S_i$ is a union of some master classes $\{M_j\}_{j=1}^{l}$, that is $S_i$ falls under Scenario 3. The test is considered successful if our approach places the appropriate parts of $S_i$ into the correct leaf nodes in $T$. The number of test cases that can be generated this way is equal to the number of non-leaf, non-root nodes.

The final set of test cases was created by taking each non-leaf node $v$ in $T$ and duplicating all the classes under the subtree rooted at $v$ as classes in $\mathcal{S}$. Then the entire subtree is collapsed into a leaf node $v$ in $T$. In all these test cases, the classes in $\mathcal{S}$ are subsets of classes in $\mathcal{M}$ which is equivalent to testing our algorithm under Scenario 4. We consider each source class $S_i$ thus generated as a separate test-case and the test is deemed passed if each source class is integrated by our algorithm into the subtree rooted at $v$.

### Datasets

We evaluate our approach on a text dataset and two hyperspectral datasets. Once the master hierarchies are obtained for the datasets, mixture models are used for likelihood estimation at the internal nodes of the tree, with each component of the mixture model representing a child of that node. To cater to the possibility of a master class being a superset of some source classes, mixture models were also used at the leaf nodes. Thus, the data points corresponding to each master class at the leaf node was clustered (5 clusters) and each cluster was used as a component of the mixture model at that leaf node. Given a set of mixture components $\{v_k\}_{k=1}^{5}$ that represents a node $v$ in the tree, the likelihood of $v$ given a set

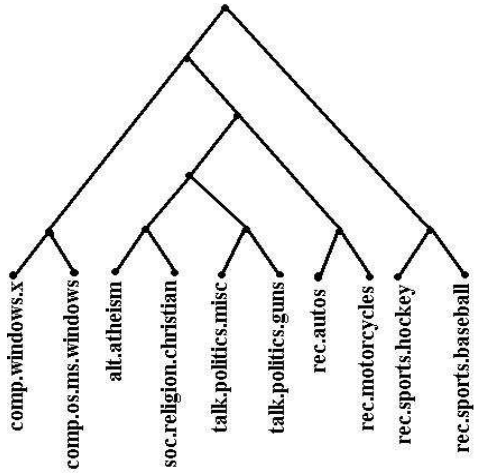


Figure 1: The master hierarchy created for the 10-newsgroups dataset.

of data-points $X_i$ is now modeled as

$$\mathcal{L}(\mathcal{D}_v|p_{X_i}) = \max_{\forall k} \mathcal{L}(\mathcal{D}_{v_k}|p_{X_i}) \quad (6)$$

where $D_{v_k}$ is the set of data points associated with the $k^{th}$ mixture component of node $v$.

**Text Data** The text data was derived from the popular 20-Newsgroup dataset (Lang 1995). This dataset consists of 1000 documents in each of 20 different newsgroups. However, some of these newsgroups are on very similar topics and have many cross-postings. For example, groups *talk.religion.misc* and *soc.religion.christian* have highly similar vocabularies and are very difficult to separate. Since we did not want to evaluate our algorithm on this artificial distinction between classes, we pruned the list of newsgroups to obtain 10 groups on different topics. The master taxonomy and source taxonomy each had 300 documents from each of the 10 classes.

The k-means version of the Divide-by-2 (DB2) technique of (Vural & Dy 2004) was used to generate the master hierarchy for the text dataset. Prior to calculating the means, the dimensionality of the data points was reduced by using the Fisher Index criterion (Chakrabarti *et al.* 1998). The reduced set of features was normalized using IDF and Spherical K-Means (Dhillon, Fan, & Guan 2001) was then used to perform the clustering. The list of classes used and the master taxonomy generated for this dataset is shown in Figure 1. A hierarchical multi-classifier system, consisting of SVMs (Joachims 1998) with linear kernels at each node of the hierarchy was used as the classifier.

For the likelihood estimation, multinomial mixture models were used to represent each of the internal nodes in the hierarchy. The mixture components at the leaf nodes were obtained by clustering the data at that node using Spherical K-Means after dimensionality reduction and normalization.

**Hyperspectral Data** The hyperspectral datasets were obtained from two sites: NASA's Kennedy Space Center (KSC), Florida (Kumar, Ghosh, & Crawford 2002) and the Okavango Delta, Botswana (Ham *et al.* 2005). The KSC and Botswana datasets consist of 10 and 14 land-cover classes respectively. For both these datasets, the data was split into the master and source taxonomies using a 60-40% split.

The master hierarchy for the hyperspectral datasets was generated using the Binary Hierarchical Classifier (Kumar, Ghosh, & Crawford 2002) algorithm which creates a hierarchy of the master classes while simultaneously generating a binary hierarchical classifier which can then be used to classify the data from the source taxonomy.

Likelihood estimation for the hyperspectral datasets was performed using mixtures of Gaussians at the internal nodes of the tree with each Gaussian representing a child of that node. The mixture model for the leaf-nodes was obtained by clustering the data using K-Means after reducing the dimensionality of the feature space by the Best-Bases algorithm (Kumar, Ghosh, & Crawford 2001).

| Scenario | 10-Newsgroups | |
|---|---|---|
| | # cases | % Correct |
| 1 | 10 | 98 $\pm$4.4 |
| 2 | 18 | 65.6 $\pm$11.4 |
| 3 | 8 | 100 $\pm$0.0 |
| 4 | 25 | 76.8 $\pm$5.2 |

Table 1: Text data (Avg % with Std.Dev).

| Scenario | KSC | | Botswana | |
|---|---|---|---|---|
| | # cases | % Correct | # cases | % Correct |
| 1 | 10 | 98 $\pm$4.4 | 14 | 100 $\pm$0.0 |
| 2 | 18 | 68.9 $\pm$6.1 | 26 | 68.4 $\pm$8.3 |
| 3 | 8 | 100 $\pm$0.0 | 12 | 100 $\pm$0.0 |
| 4 | 25 | 69.7 $\pm$8.3 | 41 | 90.2 $\pm$2.4 |

Table 2: Hyperspectral datasets (Avg % with Std.Dev).

## Discussion

Tables 1 and 2 show the performance of the mapping stage of our algorithm under the four different scenarios as detailed in the section on test case generation. The reported percentages and the corresponding standard deviations were obtained by averaging over five runs. The high percentage of correct mappings for Scenarios 1 and 3 shows that our algorithm performs as expected under these scenarios. In our experimental evaluations, we found that 'mistakes' under Scenario 2 occurred only when the test case was generated from a leaf node of the master hierarchy. As explained earlier, our evaluation measure underestimates the performance of our algorithm by counting as a mistake the mapping of a new source class to a leaf node instead of an internal one. For instance, consider the hierarchy shown in Fig.1, and let us assume that the class *comp.windows.x* is removed from the master hierarchy and is being sent in as a new source class. In this case, it is not unnatural for the new source class *comp.windows.x* to map into the leaf node corresponding to *comp.os.ms.windows*, so in reality our algorithm has better mapping accuracies than what is indicated by the re-

sults. Most of the errors in Scenario 4, occurred when the larger sub-trees were collapsed into a single node to generate the test case. Collapsing sub-trees nearer to the root of the hierarchy creates artificial master classes that represents several diverse 'concepts'. The successful mapping of the corresponding source classes to this leaf node depends on its mixture model. However, as we do not control the type of clusters generated at the leaf node, the resulting mixture model might be unable to capture all the concepts at that node resulting in wrong mappings. Using a larger number of clusters to model the leaf nodes is a possible solution to this problem.

## Conclusion and Future Work

In this paper, we presented a maximum likelihood based approach to integrate a source taxonomy into an existing master taxonomy. Utilizing the hierarchical structure defined on the master taxonomy enabled us to obtain more natural mappings between the two taxonomies as opposed to most of the existing approaches which do not fully exploit the master hierarchy. Our proposed framework not only maps, but also integrates the source classes into the master hierarchy by defining new classes in the master taxonomy. The efficacy of our approach was demonstrated empirically by evaluating it on some text and hyperspectral hierarchies. A natural extension to our current framework is enabling it to utilize additional information in source hierarchies. We also plan to test our approach on larger datasets with predefined hierarchies such as web directories and product catalogs.

**Acknowledgments:** This work was supported by NSF Grants IIS-0312471 and IIS-0307792. We would also like to thank Srujana Merugu for her helpful suggestions.

## References

Agarwal, R., and Srikant, R. 2001. On integrating catalogs. In *Proc. WWW8*, 603–612.

Chakrabarti, S.; Dom, B.; Agrawal, R.; and Raghavan, P. 1998. Scalable feature selection, classication and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal* 7(3):163–178.

Dhillon, I. S.; Fan, J.; and Guan, Y. 2001. Efficient clustering of very large document collections. In R. Grossman, C. Kamath, V. K., and Namburu, R., eds., *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers. 357–381.

Doan, A.; Madhavan, J.; Domingos, P.; and Halevy, A. 2002. Learning to map between ontologies on the semantic web. In *Proc. WWW10*, 662–673.

Dumais, S. T., and Chen, H. 2000. Hierarchical classification of web content. In *Proc. 23rd Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 256–263.

Ham, J.; Chen, Y.; Crawford, M. M.; and Ghosh, J. 2005. Investigation of the random forest framework for classification of hyperspectral data. *IEEE Trans. Geoscience and Remote Sensing* 43(3):492–501.

Ichise, R.; Takeda, H.; and Honiden, S. 2003. Integrating multiple internet directories by instance-based learning. In *IJCAI*, 22–30.

Jain, A. K., and Dubes, R. C. 1988. *Algorithms for Clustering Data*. New Jersey: Prentice Hall.

Joachims, T. 1998. Text categorization with support vector machines: Learning with many relevant features. In *Machine Learning: ECML-98, 10th European Conf. on Machine Learning*, 137–142.

Koller, D., and Sahami, M. 1997. Hierarchically classifying documents using very few words. In *Proc. 14th Intl. Conf. on Machine Learning (ICML)*, 170–178.

Kumar, S.; Ghosh, J.; and Crawford, M. M. 2001. Best-bases feature extraction algorithms for classification of hyperspectral data. *IEEE Trans. Geoscience and Remote Sensing* 39(7):1368–1379.

Kumar, S.; Ghosh, J.; and Crawford, M. M. 2002. Hierarchical fusion of multiple classifiers for hyperspectral data analysis. *Pattern Analysis and Applications, spl. Issue on Fusion of Multiple Classifiers* 5(2):210–220.

Lang, K. 1995. Newsweeder: Learning to filter netnews. In *Intl. Conf. on Machine Learning*, 331–339.

McCallum, A.; Rosenfeld, R.; Mitchell, T.; and Ng, A. 1998. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. 15th Intl. Conf. on Machine Learning (ICML)*, 359–367.

McGuninness, D.; Fikes, R.; Rice, J.; and Wilder, S. 2000. The Chimaera ontology environment. In *Proceedings of AAAI 2000*, 1123–1124. AAAI Press / The MIT Press.

Noy, N. F., and Musen, M. A. 2000. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of AAAI 2000*, 450–455. AAAI Press / The MIT Press.

Sarawagi, S.; Chakrabarti, S.; and Godbole, S. 2003. Cross-training: Learning probabilistic mappings between topics. In *Proc. of the 9th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, 177–186. Washington, DC: ACM Press.

Segal, E.; Koller, D.; and Ormoneit, D. 2001. Probabilistic abstraction hierarchies. In *In Proc. 15th Intl. Conf. on NIPS, 2001*.

Stumme, G., and Maedche, A. 2001. FCA-MERGE: Bottom-up merging of ontologies. In *IJCAI*, 225–234.

Vural, V., and Dy, J. G. 2004. A hierarchical method for multi-class support vector machines. In *Proc. 21st Intl. Conf. on Machine learning (ICML)*.

Zhang, D., and Lee, W. S. 2004. Web taxonomy integration through co-bootstrapping. In *Proc. 27th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 410–417.