TACI: Taxonomy-Aware Catalog Integration

Panagiotis Papadimitriou, *Member, IEEE,* Panayiotis Tsaparas, *Member, IEEE,*
Ariel Fuxman, *Member, IEEE,* Lise Getoor, *Member, IEEE*

Abstract—A fundamental data integration task faced by online commercial portals and commerce search engines is the integration of products coming from multiple providers to their product catalogs. In this scenario, the commercial portal has its own taxonomy (the "master taxonomy"), while each data provider organizes its products into a different taxonomy (the "provider taxonomy"). In this paper, we consider the problem of categorizing products from the data providers into the master taxonomy, while making use of the provider taxonomy information. Our approach is based on a *taxonomy-aware* processing step that adjusts the results of a text-based classifier to ensure that products that are close together in the provider taxonomy remain close in the master taxonomy. We formulate this intuition as a structured prediction optimization problem. To the best of our knowledge, this is the first approach that leverages the *structure* of taxonomies in order to enhance catalog integration. We propose algorithms that are scalable and thus applicable to the large datasets that are typical on the Web. We evaluate our algorithms on real-world data and we show that taxonomy-aware classification provides a significant improvement over existing approaches.

Index Terms—catalog integration, classification, data mining, taxonomies.

1 Introduction

An increasing number of Web portals provide a user experience centered around online shopping. This includes e-commerce sites such as Amazon and Shopping.com, and commerce search engines such as Google Product Search and Bing Shopping. A fundamental data integration task faced by these commercial portals is the integration of data coming from multiple data providers into a single product catalog. An important step in this process is *product categorization*. All portals maintain a comprehensive "master" taxonomy for organizing their products, which is used both for browsing and searching purposes. As new products arrive from different providers they need to be assigned to the appropriate category in the master taxonomy in order to be accessible to the users. At the web scale, it is impractical to assume that data providers will manually assign all the products from their catalog to the appropriate category in the master taxonomy. Thus, we need automated techniques for categorizing products coming from the data providers into the master taxonomy.

An important observation in this scenario is that the data providers do have their own taxonomy (the "provider taxonomy"), and their products are already associated with a provider taxonomy category. The provider taxonomy may be different from the master taxonomy, but in most cases, there is still a powerful signal coming from the provider classification. Intuitively, products that are in nearby categories in the provider taxonomy, should be classified into nearby categories in the master taxonomy.

To illustrate this point, consider the example in Figure 1. The provider taxonomy is an excerpt from the taxonomy used by Amazon, and the master taxonomy is an excerpt from the taxonomy used by Bing Shopping. Now, given a product tagged with a category from Amazon's (provider) taxonomy, we want to categorize it in the Bing Shopping (master) taxonomy. Suppose we are given the product "*Boss Audio Systems CH6530*" from the category Electronics/Car Electronics/Car Audio & Video/Car Speakers/Coaxial Speakers in the Amazon taxonomy. If we use a text-based classifier to categorize this product into the Bing taxonomy, it is unclear whether this product should be classified into Electronics/Car Electronics/Car Audio/Car Speakers or Electronics/Home Audio/Speakers. If we know that most of the products in the Car Speakers/Coaxial Speakers Amazon category are categorized to the Car Speakers category in Bing, then we can conclude that most likely the new product should also be classified in Car Speakers.

However, for most products in the Electronics/Car Electronics/Car Audio & Video/Car Speakers/Coaxial Speakers from the Amazon taxonomy, the classifier is actually unable to decide if they should be classified in Car Speakers, or Audio Speakers. Therefore, we cannot use the categorization of the products in the same category as the new product to guide as for the correct

- *P. Papadimitriou is with oDesk Research, Redwood City, CA 94063, USA. Part of the work done while interning at Microsoft Research. E-mail: papadimitriou@odesk.com.*
- *P. Tsaparas is with University of Ioannina, Ioannina, Greece. Work done while working at Microsoft Research. E-mail: tsap@cs.uoi.gr.*
- *A. Fuxman is with Microsoft Research Search Labs, Mountain View, CA, 94043. E-mail: arielf@microsoft.com.*
- *L. Getoor is with University of Maryland at College Park, College Park, MD 20742. Work done while visiting Microsoft Research. E-mail: getoor@cs.umd.edu.*

Manuscript received July 21, 2011; revised January 10, 2012; accepted February 17, 2012.

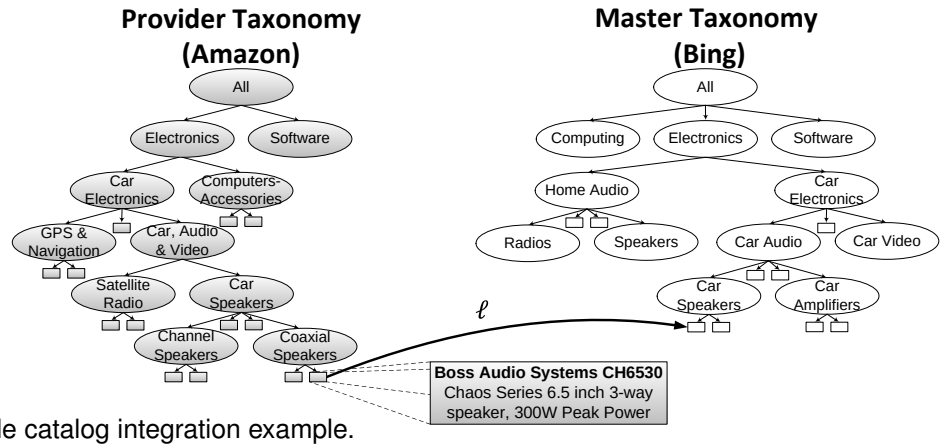


Fig. 1. A simple catalog integration example.

decision. In this case, the taxonomy information can help to determine the correct categorization. As we go up the taxonomy tree of the Amazon taxonomy, we observe that many more products in the Car Audio & Video Amazon category are classified to the Car Audio master category, as opposed to the Home Audio master category. Using this information we can conclude that most likely the products from the Electronics/Car Electronics/Car Audio & Video/Car Speakers/Coaxial Speakers category should be mapped to Electronics/Car Electronics/Car Audio/Car Speakers rather than Electronics/Home Audio/Speakers.

In this work, we propose techniques that leverage the intuition described above. We show how we can use the taxonomy information in order to "adjust" the results of a text-based classifier, and improve the product categorization accuracy. The key contribution of our work is that we make use of the *structure* of the master and provider taxonomies in order to achieve this goal, exploiting the relationships between different categories in the taxonomy, rather than relying on the category membership information of the products. In the example we described above, when the text-based classifier fails to give a clear classification of the new product, the category-membership information of the product in the provider taxonomy does not provide any additional help since there is no clear mapping between categories at the leaf level. However, there is a strong signal about the correct classification coming from mappings at higher levels of the taxonomies. Using the taxonomy structure, we can propagate this information to the lower levels of the taxonomy, and assign the correct leaf-level category to the new product.

The idea of using the "structure" of the data to enhance classification underlies many structured prediction approaches, and has been shown to work well in other areas such as computer vision [4] and NLP [2]. The general principle is that we can improve the accuracy of classification by making use of relationships between the items that are classified. However, previous approaches for catalog integration [1], [28] ignore possible relationships among the taxonomy categories, essentially treating the taxonomy as a flat collection of classes. In contrast, we exploit the full structure of the taxonomy, defining relationships between items that belong to different categories, based on the relationship of the categories in the taxonomy tree. Furthermore, most of the previous approaches suffer from scalability issues due to the large number of pairwise relationships that need to be considered. For the catalog integration problem, where we are interested in categorizing hundreds of thousands, or millions of products, such solutions are impractical. In our work, we carefully prune the relationships we consider for the classification of an item to obtain a scalable linear-time algorithm.

Contributions. In this work we make the following contributions:

1) We formulate the taxonomy-aware catalog integration problem as a structured prediction problem. To the best of our knowledge, this is the first approach that leverages the structure of the taxonomies in order to enhance catalog integration.
2) We present techniques that have linear running time with respect to the input data and are applicable to large-scale catalogs. This is in contrast to other structured classification algorithms in the literature which face challenges scaling to large datasets due to quadratic complexity.
3) We perform an extensive empirical evaluation of our algorithms on real-world data. We show that taxonomy-aware classification provides a significant improvement in accuracy over existing state-of-the-art classifiers.

The rest of the paper is organized as follows. In Section 2 we formally define the catalog integration problem. In Section 3 we present our approach, and in Section 4 we present an efficient algorithm for our formulation. Section 5 contains the experimental evaluation of our approach. In Section 6 we present the related work, and we conclude in Section 7.

2 PROBLEM DEFINITION

We now introduce some basic terminology, and formulate the *taxonomy-aware catalog integration problem*.

A *product* x is an item that can be bought at a commercial portal. Each product has a textual *representation* that consists of a *name* (a short sentence describing the product), and possibly a set of *attribute-value pairs*. For example, Figure 1 shows a product whose name is "*Boss Audio Systems CH6530*", and has also a description attribute with value "*Chaos Series 6.5-Inch 3-Way Speaker, 300W peak power*". Note that the name and the attributes of a product may vary across providers. For example, another provider may use the name "*Boss CH6530 Speakers*" for the same car speakers, and have no description, or other attributes associated with the product.

A *product taxonomy* $G = (C_g, E_g)$ is a directed acyclic graph whose nodes C_g represent the set of possible *categories* into which products are organized. Each graph edge $(c_1, c_2) \in E_g$ represents a *subsumption* (i.e., an "is-a") relationship between two categories c_1 and c_2. For example, in the master taxonomy of Figure 1, category Electronics subsumes category Car Electronics. Typically, the taxonomy structure is a rooted tree, where each category (excluding the root) has a single parent. A directed acyclic graph (DAG) is also possible, where a category may have multiple parents. We make the assumption that a taxonomy is a tree with a single root, thus imposing a clear hierarchical structure. The distance of a category from the root is the *depth* of the category in the taxonomy. For example, in the Master taxonomy of Figure 1, category Electronics has depth 1, while Car Electronics has depth 2.

A *product catalog* $\mathcal{K} = (P, G, v)$ is a taxonomy G populated with a set of products P as defined by the mapping function $v : P \rightarrow C_g$ that maps each product in P to a category in C_g. Abusing the notation, we will use v to denote the output vector of function v on P, that is, $v \in C_g^{|P|}$, and element v_x of the vector v denotes the category of product x in taxonomy G. Since v is a function, we assume that each product is associated to exactly one category, although our work can also be applied to cases where this assumption does not hold.

In the *taxonomy-aware catalog integration problem*, we are given a *source catalog* $\mathcal{K}_s = (P_s, S, s)$ that corresponds to some provider's catalog defined over the *source taxonomy* $S = (C_s, E_s)$, and a *target* (or *master*) *catalog* $\mathcal{K}_t = (P_t, T, t)$ that corresponds to the catalog of the commercial portal defined over the *target (master) taxonomy* $T = (C_t, E_t)$. The goal is to learn a *cross-catalog labeling* function $\ell : P_s \rightarrow C_t$ that maps products of the source catalog to the categories of the target catalog taxonomy. Similar to before, we abuse the notation and we use $\ell \in C_t^{|P_s|}$ to denote the labeling vector, where the entry ℓ_x of the vector is the assigned category of product $x \in P_s$ in taxonomy T.

We can now define our problem abstractly as follows:

Definition 1 (Taxonomy-Aware Catalog Integration): Given a source catalog $\mathcal{K}_s$ and a target catalog $\mathcal{K}_t$, use a taxonomy-aware process f_T to learn a cross-catalog labeling function $\ell = f_T(\mathcal{K}_s, \mathcal{K}_t)$.

The key novelty in our approach is that the learning process f_T that produces labeling vector ℓ makes use of the full *taxonomy structure* of the taxonomies S and T in order to define relationships between products in the source and target catalog, and guide the classification process. Previous approaches [1] have exploited the category assignments s, and t, but not the structure of the taxonomies S and T. In the next Section we describe how to formulate the process f_T as a combinatorial optimization problem.

3 TAXONOMY-AWARE CLASSIFICATION

At a high level we can describe our approach to taxonomy-aware classification as a two-step process. First, each product is classified using a *base* classifier that is not aware of the taxonomies. We call this the *base classification step*. Then, we use the structure of the source and target taxonomies in order to adjust the output of the base classifier, and produce a final classification. We call this the *taxonomy-aware processing* step. We now discuss these two steps in detail.

3.1 The Base Classification Step

In the base classification step, we classify the products based solely on their textual representation. For this purpose, we train a text-based classifier using standard supervised machine learning techniques. In this paper, we consider Naive Bayes, and Logistic Regression [18], but any other machine learning technique could be used. We use a subset of the *target catalog* as the training set. This provides us with examples of products labeled with categories of the target taxonomy. The features of the classifier are extracted from the textual product representations. Note that at training time we have no knowledge of the providers' catalogs, and we make no use of the structure of the target taxonomy.

Let b denote the classification model we obtain after training. Given a product $x \in P_s$ from the provider catalog we apply the classifier b on the *textual representation* of the product, as this appears in the provider's catalog. In our example in Figure 1 the features are extracted from the text "*Boss Audio Systems CH6530, Chaos Series 6.5-Inch 3-Way Speaker, 300W peak power*". The output of the base classification step is a probability distribution $Pr_b[\tau|x]$ over all target categories $\tau \in C_t$ for each product $x \in P_s$. The probability $Pr_b[\tau|x]$ provides an estimate of the likelihood that product x belongs to category τ, and the confidence that the classifier has for this assignment.

We stress again that during the base classification step we do not make use of any knowledge about the target or source taxonomy, either during training, or during the application of the classifier. This refers to both the structure of the taxonomy, as well as the names of the categories. It is possible for the classifier to try to match the names of the categories between the source and target taxonomies. However, this entails the danger of over-fitting, and also as we have observed, category names often vary significantly between providers (e.g., Cameras v.s. Photography).

3.2 The Taxonomy-Aware Processing Step

The intuition behind the taxonomy-aware processing step is that the target categories assigned by the base classification step can be adjusted by taking into account the relationships of the products in the source and target taxonomies. The goal of the taxonomy-aware processing is to assign categories in the target taxonomy to the products coming from the provider's catalog, such that the assignments respect the decisions of the base classifier, while at the same time they preserve the relative relationships of the products in the source taxonomy. We will now show how the taxonomy-aware processing step can be defined as an optimization problem, and discuss the different parameters of the problem.

3.2.1 The Optimization Problem

Formally, we define taxonomy-aware processing as an optimization problem, where given a source catalog $\mathcal{K}_s$, and a target catalog $\mathcal{K}_t$, the objective is to find a labeling vector ℓ that minimizes the following *cost function*:

$$\begin{aligned}\textsc{Cost}(\mathcal{K}_s, \mathcal{K}_t, \ell) =& (1-\gamma) \sum_{x \in P_s} \textsc{ACost}(x, \ell_x) \\ &+ \gamma \sum_{x,y \in P_s} \textsc{SCost}(x, y, \ell_x, \ell_y) \quad (1)\end{aligned}$$

The taxonomy-aware process f_T is the algorithm that finds the labeling ℓ that minimizes the cost function:

$$f_T(\mathcal{K}_s, \mathcal{K}_t) = \arg\min_{\ell} \textsc{Cost}(\mathcal{K}_s, \mathcal{K}_t, \ell).$$

The first term of the cost function, ACOST, is the *assignment cost* which penalizes classifications that differ from the ones chosen by the base classifier. We define the assignment cost in Section 3.2.2. The second term, SCOST is the *separation cost* which penalizes classifications that place products that are close in the source taxonomy to distant categories in the target taxonomy (or vice-versa). We introduce the separation cost in Section 3.2.3 and we discuss it in detail in Appendix A. The parameter γ (also called *regularization parameter*) balances between the two costs. Note that an optimal classification ℓ may assign a product x to a target category ℓ_x that is different from the one chosen by the base classifier, if the neighboring products of x in the source taxonomy are assigned target categories that are close to category ℓ_x in the target taxonomy. In this way, we are getting signal both from the product representation (assignment cost) and from the source categories and taxonomy structure (separation cost).

TABLE 1
Category similarity and penalty functions used in separation cost.

Category similarity functions: $\text{sim}_G(c_1, c_2)$	
Shortest-Path similarity	$2^{-\text{hops}(c_1,c_2)}$
LCA similarity	$1 - 2^{-\text{depth}(\text{lca}(c_1,c_2))}$
Cosine similarity	$\frac{1+\text{depth}(\text{lca}(c_1,c_2))}{\sqrt{1+\text{depth}(c_1)}\sqrt{1+\text{depth}(c_2)}}$
Penalty functions: $\textsc{SCost}(s_x, s_y, \ell_x, \ell_y)$	
Absolute difference	$\lvert\text{sim}_S(s_x, s_y) - \text{sim}_T(\ell_x, \ell_y)\rvert$
Multiplicative	$\text{sim}_S(s_x, s_y)(1 - \text{sim}_T(\ell_x, \ell_y))$

3.2.2 Assignment Cost

We use the probabilities of the base classifier to define the assignment cost function $\textsc{ACost}$: $P_s \times C_t \to \mathbb{R}^+$. For a product x the cost of classifying product x to target category ℓ_x is defined as follows:

$$\textsc{ACost}(x, \ell_x) = 1 - Pr_b[\ell_x | x] \quad (2)$$

This definition penalizes the classification of product x to a category ℓ_x for which the base classifier yields small probability estimates. On the other hand, the assignment cost is small if product x is assigned to a category for which the base classifier assigns high probability.

3.2.3 Separation Cost

The *separation cost* function $\textsc{SCost} : P_s^2 \times C_t^2 \to \mathbb{R}^+$ is defined over a pair of products $x, y \in P_s$ that we want to classify, and a pair of categories $\ell_x, \ell_y \in C_t$ that are candidate target categories for x and y respectively. Intuitively, it captures the cost of *separating* products x and y to target categories ℓ_x and ℓ_y. The separation cost should be high if products x and y belong to categories s_x, s_y that are *close* in the source taxonomy, and which are assigned classes ℓ_x, ℓ_y that are *distant* in the target taxonomy.

To formally define this intuition, we need a definition of *similarity* between two categories in a taxonomy. For a given taxonomy $G = (C, E)$ a similarity function $\text{sim}_G : C \times C \to [0, 1]$ returns a value in $[0, 1]$ for any two categories $c_1, c_2 \in C$, where 1 means that categories c_1 and c_2 are identical and 0 means that they are completely dissimilar. A meaningful similarity definition should satisfy the intuition that two categories that are close in the taxonomy tree are more similar than two categories that are far apart. For example, two categories that have a common parent are more similar than two categories that have different parents and a common grandparent.

With the notion of category similarity at hand, we now define the separation cost as a function of the

similarity $\text{sim}_S(s_x, s_y)$ between categories s_x and s_y of x and y in the source taxonomy S and the similarity $\text{sim}_T(\ell_x, \ell_y)$ between ℓ_x and ℓ_y in the target taxonomy T, that is,

$$\text{SCOST}(x, y, \ell_x, \ell_y) = \delta\left(\text{sim}_S(s_x, s_y), \text{sim}_T(\ell_x, \ell_y)\right) \quad (3)$$

where δ is a *penalty function* that yields high values if the similarity $\text{sim}_S(s_x, s_y)$ of the source categories is high, while the similarity $\text{sim}_T(\ell_x, \ell_y)$ is low, or vice versa. An example of the penalty function δ is the absolute difference of the two similarity values. Intuitively, we want the relative position of assigned categories of two products in the target taxonomy to be similar to the relative position of the categories in the source taxonomy.

Depending on the choice of the penalty and similarity functions, we obtain different definitions for the separation cost. The similarity and penalty functions that we consider in this work are listed in Table 1, together with their mathematical definition. We discuss the definitions in detail in Appendix A.

Note that as defined in Equation 3, for a given choice of the penalty function δ, the separation cost is fully defined by the tuple $(s_x, s_y, \ell_x, \ell_y)$, and it does not depend on the identities of the individual products x and y. We can thus write $\text{SCOST}(s_x, s_y, \ell_x, \ell_y)$ to denote the separation cost $\text{SCOST}(x, y, \ell_x, \ell_y)$. That is, we can define the function $\text{SCOST} : C_s^2 \times C_t^2 \to \mathbb{R}^+$ over pairs of source and target categories.

The key observation is that all pairs of products x, y such that $s_x = \sigma$ and $\ell_x = \tau$, and $s_y = \bar{\sigma}$ and $\ell_y = \bar{\tau}$, for some $\sigma, \bar{\sigma} \in C_s$, and $\tau, \bar{\tau} \in C_t$ have the same separation cost $\text{SCOST}(\sigma, \bar{\sigma}, \tau, \bar{\tau})$. That is, the separation cost is the same for all pairs of products x, y that that come from source categories σ and $\bar{\sigma}$ respectively, and are assigned to target categories τ and $\bar{\tau}$ respectively. If $n(\sigma, \tau)$ denotes the number of products from σ that are assigned to τ, according to some labeling ℓ, then, there are $n(\sigma, \tau) \cdot n(\bar{\sigma}, \bar{\tau})$ such pairs of products, and each one will contribute $\text{SCOST}(\sigma, \bar{\sigma}, \tau, \bar{\tau})$ to the separation cost. Then, we can write the separation cost in Equation 1 as follows:

$$\sum_{x,y \in P_s} \text{SCOST}(x, y, \ell_x, \ell_y) \quad (4)$$
$$= \sum_{\sigma, \bar{\sigma} \in C_s} \sum_{\tau, \bar{\tau} \in C_t} \text{SCOST}(\sigma, \bar{\sigma}, \tau, \bar{\tau}) n(\sigma, \tau) n(\bar{\sigma}, \bar{\tau})$$

Note that the sum in Equation 4 is now expressed over pairs of source categories and pairs of target categories, rather than pairs of products. We leverage this observation for the efficient computation of the separation cost that we present in Section 4.

Although we do not discuss it further in this paper, we can modify Equation 3 to handle multi-class classification of products in the source taxonomy. If x and y are classified in more than one categories in the source taxonomy then s_x and s_y are now sets of categories, rather than single categories. Given a similarity measure between the set members there are different ways for defining the similarity between sets that we can employ. One possibility is to use the maximum similarity of any pair of elements between the two sets. The intuition for such a choice is that if two products x and y appear in close-by categories in a taxonomy then they are similar, even if they also appear in some distant categories.

4 THE TACI ALGORITHM

The optimization problem in Section 3.2.1 is closely related to a variety of optimization problems such as the *metric labeling* problem [12], or structured prediction problems [2]. These problems are known to be NP-hard when asking for a hard labeling of the data. For certain variants there are approximation algorithms [5], [6], [12], [23]. Our problem can be formulated as an Integer Linear Program (ILP) or a Quadratic Integer Program (QIP), however the number of variables is proportional to the number of products in the source catalog, which is prohibitively large. To the best of our knowledge, none of the solutions that have been proposed are applicable to web-scale classification problems, where the number of products to be classified can be in the order of hundreds of thousands, or millions.

In this section, we present a scalable algorithm for the taxonomy-aware classification step that scales up to large datasets. Although we present our technique with respect to our specific problem, the methodology that we propose can be applied to other structured prediction problems in order to deal with the quadratic number of pairwise relationships that need to be considered. To aid the presentation, we refer the reader to Table 2 which explains the main terminology and symbols that we will use in this section.

4.1 Search Space Pruning

We now present a heuristic for efficiently performing the taxonomy-aware calibration step. The idea is to judiciously *fix* the category for some of the products in the source catalog in order to obtain a landscape of the mappings between the two taxonomies. We then treat each of the *open* (non-fixed) products independently making use of Equation 4 to efficiently compute the separation cost.

We identify the products whose assigned class can be fixed using the output of the base classifier. The intuition is that if the base classifier is sufficiently "confident" about the top category of a source product, then it is likely to be making the correct prediction. However, if it is not confident about the category (low probability estimate), it might not be predicting the right category and therefore we should consider adjusting the assignment at the taxonomy-aware calibration step.

TABLE 2
Algorithm terminology chart.

Generic terms	
S, T	source and target taxonomies
C_s, C_t	sets of source and target taxonomy categories
σ, τ	denote a source and target taxonomy category
P_s	set of source catalog products
Algorithm Parameters	
k	number of top categories to consider as candidates
θ	probability threshold for *fixing* the category
γ	regularization parameter between assignment and separation cost
Variables used in TACI algorithm	
$Pr_b[\tau\|x]$	probability of category τ for product x output by base classifier b
F_θ	products whose category is *fixed*
O_θ	*open* products whose category is not fixed
$h(\sigma,\tau)$	separation cost for product coming from category σ and assigned to category τ
Ψ	cache for $h(\sigma,\tau)$ values
$H_{\theta,k}$	source-target category pairs for which we need to compute the separation cost h

Let $\theta \in [0,1]$ be a threshold value that defines when the category probability estimate returned by the base classifier is large enough so that the predicted category is likely to be correct. Let F_θ be the subset of products that pass the threshold, that is:

$$F_\theta = \{x \in P_s | \max_{\tau \in C_t} Pr_b[\tau|x] \geq \theta\}. \quad (5)$$

For the products in F_θ we trust the prediction of the classifier, and we fix their labeling to the most likely base classifier output. That is, for all $x \in F_\theta$

$$\ell_x = \arg\max_{\tau \in C_t} Pr_b[\tau|x]. \quad (6)$$

Let $O_\theta = P_s \setminus F_\theta$ denote the products whose classification remains open. We treat each open product $x \in O_\theta$ *independently*, and we compute a separation cost for x *only with respect to the fixed products in* F_θ. If s_x is the source category of x, and t_x is a candidate target category, then the separation cost for this source-target pair is defined as follows:

$$h(s_x, \ell_x) = \sum_{y \in F_\theta} \text{SCOST}(x, y, t_x, \ell_y).$$

Using Equation 4 we can rewrite $h(s_x, t_x)$ as:

$$h(s_x, t_x) = \sum_{\sigma \in S, \tau \in T} \text{SCOST}(s_x, \sigma, t_x, \tau)\bar{n}(s_x, t_x)\bar{n}(\sigma, \tau) \quad (7)$$

where $\bar{n}(\sigma,\tau)$ is the number of products in F_θ that belong in category σ in S and are assigned to category τ in T. Therefore, by fixing the labeling of the products in F_θ we have effectively achieved the following: first, we have created a set of "anchors" against which we compute the separation cost for a new open product; second, the $\bar{n}(\sigma,\tau)$ values, define a mapping between the source and target taxonomies. This mapping is used to compute the separation cost, and guide the categorization of new open products.

To further speed-up the taxonomy-aware classification step, we also reduce the space of candidate source-target pairs. For an open product $x \in O_\theta$, let $\text{TOP}_k(x)$ denote the top-k candidate target categories with respect to the probability estimates $Pr_b[\tau|x]$ of the base classifier. We reduce the search space by considering only these k target categories as candidates for product x. As we discuss in Section 4.3, restricting ourselves to the top-k categories does not significantly limit our maximum achievable accuracy.

We use $H_{\theta,k}$ to denote the set of candidate source-target pairs for which we need to compute the separation cost h:

$$H_{\theta,k} = \{(s_x, \tau) : x \in O_\theta, \tau \in \text{TOP}_k(x)\}.$$

From Equation 7 it follows that the separation cost for the pairs in $H_{\theta,k}$ can be computed using the mappings defined by the products in F_θ, and it is independent of the products in O_θ. Therefore, for each candidate source-target category pair (σ,τ), we need to compute the separation cost $h(\sigma,\tau)$ only once, and store the value for future use. We describe the algorithm in detail in the next section.

4.2 Taxonomy-Aware Algorithm

Algorithm 1 describes the Taxonomy Aware Catalog Integration algorithm (henceforth referred to as the TACI *algorithm*). The algorithm assumes the existence of a base classifier trained on data from the target catalog. The input to the algorithm consists of a source catalog and a target taxonomy, as well as the parameters θ, k, and γ. The output of the algorithm is a labeling ℓ for the products in the source catalog.

In the loop of Lines 2-9, the algorithm applies the base classifier to each product. Based on the base classifier output probabilities, the algorithm either classifies the product to the top category given by the base classifier (Lines 4-6), or it leaves its classification open and stores the top k categories, sorted by probability (Lines 7-9). Given the set of open products O_θ, and their top-k candidate target categories the algorithm computes the set of candidate source-category pairs $H_{\theta,k}$ (Line 10). In the loop of Lines 12-13 the algorithm computes the separation costs for all of the candidate pairs $(\sigma,\tau) \in H_{\theta,k}$, and stores them in a hash table Ψ. Note that for each source-target pair we compute the value of h only once, and we never compute a separation cost that we will not use later on. In the loop of lines 14-15 the algorithm classifies the open products in O_θ. A product $x \in O_\theta$ is assigned to the category ℓ_x among the top-k categories in $\text{TOP}_k(x)$ that minimizes the objective function.

We now provide an analysis of the running time of the algorithm. The first loop in Lines 2-9 iterates over the set P_s of all products in the source taxonomy, and requires $O(|C_t|)$ time to process each product, where $|C_t|$ is the number of categories in the target taxonomy. Hence, the running time of the first loop is $O(|P_s||C_t|)$. The set of candidate pairs $H_{\theta,k}$ has size at

Algorithm 1 TACI Algorithm

Input: source catalog $\mathcal{K}_s$, target taxonomy T, base classifier b, and parameters θ,k, and γ.
Output: a labeling vector ℓ

1: $F_s \leftarrow \emptyset$
2: **for all** $x \in P_s$ **do**
3: $\quad \tau^* \leftarrow \arg\max_{\tau \in C_t} Pr_b[\tau|x]$
4: $\quad$ **if** $Pr_b[\tau^*|x] \geq \theta$ **then**
5: $\quad\quad \ell_x \leftarrow \tau^*$
6: $\quad\quad F_\theta \leftarrow F_\theta \cup \{x\}$
7: $\quad$ **else**
8: $\quad\quad O_\theta \leftarrow O_\theta \cup \{x\}$
9: $\quad\quad$ Compute $\text{TOP}_k(x)$
10: Compute candidate pairs $H_{\theta,k}$
11: Initialize hash table Ψ to empty
12: **for all** $(\sigma, \tau) \in H_{\theta,k}$ **do**
13: $\quad \Psi[(\sigma, \tau)] = h(\sigma, \tau)$
14: **for all** $x \in O_\theta$ **do**
15: $\quad \ell_x \leftarrow \underset{\tau \in \text{TOP}_k(x)}{\text{argmin}} \{(1-\gamma)\text{ACOST}(x, \tau) + \gamma\Psi[(s_x, \tau)]\}$

most $|C_s||C_t|$, and thus the computation of the candidate pairs at Line 10, has cost at most $O(|C_s||C_t|)$. However, we expect this to be significantly lower, since for each product we are only considering the top-k categories, and it is unlikely that all possible pairs of categories will occur in practice. In the loop of lines 12-13, the algorithm computes the separation cost $h(\sigma, \tau)$ for all candidate pairs in $H_{\theta,k}$. The cost of calculating $h(\sigma, \tau)$ is $\text{O}(|C_s||C_t|)$, so the worst case running time of the loop is $\text{O}(|C_s|^2|C_t|^2)$. Again, we expect this to be smaller in practice, since the size of the candidate set is likely to be smaller. The loop in Lines 14-15 now just sums up the precomputed costs, for the open products in O_θ for each one of the top-k candidate categories. The processing time is $\text{O}(k|P_s|)$. Thus, the total running time of Algorithm 1 is $\text{O}(|P_s||C_t|)+\text{O}(|C_s|^2|C_t|^2)+\text{O}(k|P_s|)$, which is linear with respect to the number of products $|P_s|$ in the source catalog.

4.3 Parameter Calibration

The tuning of the parameters k, γ and θ is important for the performance of our algorithm. Similarly to other works [1], [28], we assume the existence of a validation set on which we tune the parameters. The validation set consists of products that are cross-labeled in both the source and the target taxonomy. To obtain such a set, we can either automatically match source to master taxonomy products using some universal unique identifier such as the ISBN for books, or manually label some products of the source taxonomy if such an identifier does not exist. In both cases, it is assumed that the obtained validation set is too small to be used for the base classifier training that involves tens of thousands of features, while it is big enough to tune few parameters of the TACI algorithm.

The calibration of the parameters of the algorithm works as follows. First, we run the base classifier b on the products of the validation set, and for each product x we obtain a probability distribution $Pr_b[\tau|x]$. The first parameter we set is parameter k, such that the accuracy of the classifier over the top-k categories is high. The details are described below. Then we tune the parameters θ and γ. We consider a set $\{\theta_1, ..., \theta_{N_\theta}\}$ of candidate values selected as described below. For each candidate parameter θ_i we find the "optimal" parameter γ such that the accuracy of the TACI algorithm on the validation set is maximized. The algorithm for estimating γ is described below. We output the pair of parameters θ, γ that achieve the maximum accuracy. We note that all parameters are selected such as to maximize the accuracy of the TACI algorithm on the validation set. Since the products on the validation set are cross-labeled we know the true labeling and we can compute the accuracy. To avoid overfitting, when tuning the parameters θ and γ we do not update them unless we have a significant improvement in the accuracy (0.1% of the previous value in our experiments).

Tuning Parameter k: we set the parameter k, such that the accuracy of the base classifier over the top-k results (i.e., the fraction of times that the true category is contained in the top-k results) is above a certain threshold (99% in our experiments), or k reaches a predefined maximum (20 in our experiments). In this way we guarantee that the TACI algorithm can achieve accuracy up to 99%.

Tuning Parameter θ: The parameter θ determines the anchor set F_θ: the products in F_θ should have high classification accuracy, and at the same time capture the true mappings between the source and target taxonomies. The first requirement does not necessarily imply the second, and thus we cannot set the threshold θ in the same way we did for k. We thus perform a linear search for N_θ different values, and we select the one for which the calibration step gives the best accuracy. We judiciously select the candidate values for θ using the probabilities output by base classifier to guide our choice. Let $\pi_x^* = Pr_b[\tau^*|x]$ denote the probability of the most likely category of product x. We sort the probabilities π_x^* and we select N_θ values, evenly spaced, as the candidate θ values. Thus, the choice of the candidate values controls the fraction of products in the validation set that are below the threshold: this is the set of products for which we run the taxonomy-aware step. This way we can focus on the appropriate range of candidate values.

In our experiments we used $N_\theta = 40$ different values that divide the base classifier probabilities in 40 equally sized buckets. For example, if the validation set has 4000 products, then each interval $[\theta_i, \theta_{i+1})$ contains 100 products, that is, 2.5% of the total products. Furthermore, for the θ_i candidate value there are $100i$

products that have probability $\pi_x^* \leq \theta_i$ for which we run the taxonomy-aware processing step.

Tuning Parameter γ: The parameter γ is important since it controls the effect of the taxonomy signal on the classification. If the taxonomy provides a misleading signal, then we need to minimize its contribution to the optimization problem. We determine the parameter γ by making use of the assignment and separation costs over the validation set, and finding the value of γ that gives the best accuracy. For a given product x in the validation set, let $\tau_1, ..., \tau_k$ be the set of candidate target categories in $\text{TOP}_k(x)$, and let τ^* denote the correct category for x. Note that τ^* may not be one of the categories in $\text{TOP}_k(x)$. For each target category τ_i we can compute an assignment cost η_i and a separation cost ζ_i. For a given value of the parameter γ the cost of assigning x to this category is $c_i = (1-\gamma)\eta_i + \gamma\zeta_i$. The item x will receive the correct category τ^*, if $\tau^* \in \text{TOP}_k(x)$, and for all $\tau_i \in \text{TOP}_k(x)$, $c^* \leq c_i$. In this case, the correct category has the minimum cost, and it will be selected by our algorithm. We want to find a value for γ that ensures that the above requirement is satisfied for as many products in the validation set as possible.

The algorithm proceeds as follows. For each candidate target category τ_i of product x we find the range of values for γ that satisfy the inequality:

$$(1-\gamma)\eta^* + \gamma\zeta^* \leq (1-\gamma)\eta_i + \gamma_i\zeta \ ,$$

where η^* and ζ^* denote the assignment and separation cost for the correct category τ^*. This is the range of values of γ for which the correct category τ^* is selected over τ_i. If the interval $[l_x, u_x]$ is the intersection of the ranges of values for all τ_i, then for any γ in this interval the correct category $\tau*$ is selected over *all* categories in $\text{TOP}_k(x)$. If this interval is empty, or it does not intersect with $[0, 1]$ (e.g., it is negative) then we discard item x since there is no value of γ for which we can assign x the correct category. Otherwise, any value in the interval $[l_x, u_x]$ can be used as the γ value. In this case we say that the interval *satisfies* product x.

After processing all products, we obtain a set of intervals, one for each product that was not discarded. For any value $\gamma \in [0, 1]$, the number of intervals in which it is contained is the number of products it satisfies. Intersecting all the intervals we find the range of values $[\gamma_l, \gamma_u]$ that satisfies the maximum number of products. We return the mean of the interval $\gamma = (\gamma_l + \gamma_u)/2$ as the final value for the regularization parameter.

5 Experimental Evaluation

In this section, we present the experimental evaluation of our approach. The main goals of this evaluation are the following: (1) to show the benefits of our taxonomy-aware calibration step and compare the taxonomy-aware algorithm against other catalog integration approaches (Section 5.2); (2) to evaluate the different cost and similarity functions we consider (Section 5.3); and (3) to study the running time of our algorithm (Section 5.4), and the sensitivity to parameter values (Section 5.5).

5.1 Experimental Setup

Datasets. We use as master catalog the catalog of Bing Shopping, which aggregates data feeds from retailers, distributors, resellers, and other commercial portals. We consider three providers: Amazon, Etilize, and Pricegrabber. The Amazon source catalog contains $65,512$ products; the Etilize catalog has $136,876$ products; and the Pricegrabber catalog consists of $544,898$ products.

In all the experiments, we consider a target taxonomy that consists of all the categories in Bing Shopping taxonomy that are related to consumer electronics (all of the dataset products come from such categories). The resulting taxonomy has 297 categories. There are 37 internal nodes, and 260 leaf nodes. The height of the tree is 4 and the average branching factor is 8.2. As source taxonomies, we use taxonomies from the Amazon, Etilize and Pricegrabber providers, restricted to the consumer electronics domain. In the case of Etilize, the taxonomy has 523 nodes, of which 94 are internal nodes, and 429 are leaf nodes. The taxonomy tree has height 5, and an average branching factor of 5.6. The Pricegrabber taxonomy has 1001 nodes, 211 of which are internal nodes and 790 are leaf nodes. Pricegrabber taxonomy has height 3 and average branching factor is 4.8. Amazon has 1788 nodes in total, 1312 of which are leaves and 476 are internal nodes. The height of the taxonomy is 7, and the average branching factor is 3.8.

An important observation regarding the datasets is the skewed category distribution over products in all of the taxonomies. For example, the master taxonomy with all of the dataset products has five categories with more than 50K products each, while it has approximately 70 categories with less than 500 products. We can make similar observations for the source taxonomies. The skewed category distribution over classified objects has also been observed in other large web taxonomies [16]. This observation justifies the normalization step that we use in Section A.3.

Ground truth. The ground truth is the true labeling of the products in the provider catalog with categories from the master taxonomy. We obtained the ground truth as follows: the Etilize products were manually labeled by human experts; the Pricegrabber products were assigned a category from the master taxonomy by applying rules developed by domain experts; the Amazon products were associated to products in the master catalog via universal identifiers (e.g., ASIN and UPC numbers), and then they were assigned

to the master category of the matching product in the catalog. We split the ground truth data into two subsets: the *validation set* that consists of 10% of the labeled data, and the *test set* that consists of the remaining 90% of the labeled data. We use the validation set to tune the parameters of the algorithms, and the test set in order to evaluate their performance.

Algorithms. We compare our approach against two types of algorithms: *representation-based* algorithms, which classify products using only the textual representation of the product, and *category-aware* algorithms that make use of the source category of the products, but not the full taxonomy structure (i.e., assuming a flat taxonomy structure).

For the representation-based approaches, we consider two base classifiers: Naive Bayes (NB) and Logistic Regression (LR). These are standard classification algorithms [18]. The features that we use are unigrams and bigrams extracted from the textual representation of the product, that is, the product name and the attribute values. At training time the features are extracted from the representation of the products in the master catalog. At test time (or during the parameter tuning phase) we extract the features from the textual representation of the product in the provider catalog. The provider catalogs always have a name associated for each product. For the Etilize catalog we also have attribute values (such as brand name). Since we use the master catalog products to train the base classifier, we do not use textual features from the category names to avoid over-fitting (the category name directly indicates the product category).

For the category-aware approach we consider the algorithm of Agrawal and Srikant [1] (henceforth denoted as the AS algorithm), one of the few category-aware classification algorithms that can handle large-scale datasets. Their algorithm builds upon a Naive Bayes classifier, and biases the classification of a product x towards the target categories in which other products from x's source category have been classified. More specifically, the algorithm first runs Naive Bayes on the test data to obtain the base probabilities $Pr_b[\tau|x]$ for the product classifications. Then, assuming that a product x is classified to category $\tau = \text{argmax}_{\tau'} Pr_b[\tau'|x]$ it computes the conditional probabilities $Pr[\tau|\sigma]$ that a product that comes from σ will be classified to τ using the formula:

$$Pr[\tau|\sigma] = \frac{n(\tau) \times n(\sigma,\tau)^w}{\sum_{\tau' \in C_t} n(\tau') \times n(\sigma,\tau')^w}, \quad (8)$$

where $n(\tau)$ is the number of products in τ, $n(\sigma,\tau)$ is the number of products in σ predicted to be in τ, and w is a tuning parameter. Finally, product x is classified to category $\tau^* = \text{argmax}_\tau Pr[\tau|\sigma] Pr_b[\tau|x]$. The parameter w for the Agrawal and Srikant algorithm is tuned over the validation set.

For our taxonomy-aware (TACI) approach, we experimented with both Logistic Regression and Naive Bayes as base classifiers. We use TACI-LR to denote TACI over Logistic Regression, and TACI-NB for TACI over Naive Bayes. The parameters of our algorithm are tuned over the validation set. We use the same validation set for both TACI and the AS algorithm.

TABLE 3
Accuracy for all classifiers over all datasets

	NB	LR	AS	TACI-NB	TACI-LR
Amazon	77.3%	70.5%	80.5%	**81.1%**	75.9%
Etilize	75.3%	80.1%	86.0%	81.7%	**91.8%**
Pricegrabber	41.6%	49.3%	55.0%	71.2%	**74.4%**

Evaluation Metric: We use *classification accuracy* with respect to the ground truth set as our evaluation metric. The accuracy is the number of source products for which the correct target category is predicted, divided by the total number of source products.

5.2 Classification Accuracy Evaluation

In this section we compare the classification accuracy of the different approaches for catalog integration. The results for all algorithms over all datasets are in Table 3. For the TACI algorithms, we report the best results over all possible choices of similarity metric and separation cost. We study the similarity and separation cost parameters in detail in Section 5.3.

The first observation from Table 3 is that the taxonomy-aware step provides significant benefits over the representation-based base classifiers. As we can see in Table 3, TACI-NB and TACI-LR consistently outperform NB and LR, respectively. For example, on the Etilize dataset TACI-LR improves the accuracy of LR by an additional 10.7%, reducing the classification error to less than half of that of LR. In the Pricegrabber dataset, the TACI-LR algorithm gives an additional 25.1% accuracy over the LR algorithm, slashing the classification error to half. The gains as less pronounced in the Amazon dataset, although TACI-NB improves over the accuracy of NB by 3.8% .

Second, TACI outperforms the AS algorithm that we use as a category-aware baseline. The category-aware AS algorithm does achieve accuracy gains over the NB baseline, but, as we can see in Table 3, TACI has overall higher accuracy. The improvement is especially pronounced for the Etilize and Pricegrabber datasets: in the former TACI-LR improves the accuracy by an additional 5.8%, while in the latter the increase is by an additional 19.4%, reducing the classification error to almost half of that of AS. The improvement is less pronounced on the Amazon dataset.

In order to get a better understanding of the above results, we manually inspected the errors in the Amazon dataset for NB, AS, and TACI-NB (notice that both AS and TACI-NB build on top of NB). The anecdotal findings are very revealing of how the TACI algorithm

TABLE 4
Classification accuracy for different similarity metrics and separation costs.

		Absolute Difference	Multiplicative Cost
PG	Shortest-Path	**63.1%**	62.7%
	LCA	**74.4%**	62.1%
	Cosine	**69.7%**	63.7%
Etilize	Shortest-Path	**91.3%**	90.9%
	LCA	**91.7%**	91.2%
	Cosine	**91.3%**	90.6%
Amazon	Shortest-Path	77.2%	**81.1%**
	LCA	**80.6%**	80.2%
	Cosine	**79.6%**	79.4%

manages to make use of the taxonomy structure. Consider the following example, which is actually the motivation example we used in the introduction. There are 272 products in the test set tagged with the Amazon category Electronics/Car Electronics/Car Audio & Video/Car Speakers/Coaxial Speakers which, according to the ground truth, map to the master category Electronics/Car Electronics/Car Audio/Car Speaker. However, NB makes 138 errors, 118 of which correspond to products mapped to the category Electronics/Home Audio/Speakers. This can be explained from the fact that the NB classifier has only access to the product representation (i.e., the textual description), and the representation of a car speaker can be easily confused with the representation of a home audio speaker. The AS algorithm makes fewer mistakes (68 in total), but TACI-NB is considerably better with just 21 errors.

The reason that the TACI-NB performs so well is that there is a strong signal in the taxonomy that the algorithm is able to exploit. While at the leaf level there is uncertainty about the right category, as we move upwards in the taxonomy tree it becomes apparent that the car audio speakers should be mapped to a subcategory of Car Electronics and not Home Audio. For example, if we aggregate the predictions of the NB classifier over all products in the source category Electronics/Car Electronics/Car Audio & Video, 64% of them map to the target category Electronics/Car Electronics compared to 43% of the products in Electronics/Car Electronics/Car Audio & Video/Car Speakers/Coaxial Speakers that map to Electronics/Car Electronics/Car Audio/Car Speaker. It is precisely this higher-level mappings between taxonomies that the taxonomy-aware approach is able to take advantage of, while the category-aware approach is unable to make use of this signal.

5.3 Similarity and Separation Cost Analysis

In this section we study the different similarity metric and separation cost options. Table 4 shows the accuracy performance of the TACI algorithm for the three similarity measures we consider, and the two possible separation cost definitions, for all three datasets. The taxonomy-aware step is applied on top of the best-performing baseline classifier. This is logistic regression for Etilize and Pricegrabber, and the Naive Bayes for Amazon. The accuracy numbers in bold are the best for a specific similarity metric (best in the row), and the boxed numbers are the best for the specific dataset.

First, note that our technique demonstrates a consistent trend for all of the different choices of separation cost and similarity definitions. In the Etilize dataset, which appears to be amenable to the taxonomy-aware approach, all of the algorithms exhibit more or less the same accuracy. In the Amazon dataset, where the taxonomy information appears to be minimally helpful for the classification task, all of the algorithms exhibit similar minimal gains (if any). In the PriceGrabber dataset, despite the variability in the results, all of the algorithms exhibit gains of at least an additional 10% with respect to the baseline classifier accuracy.

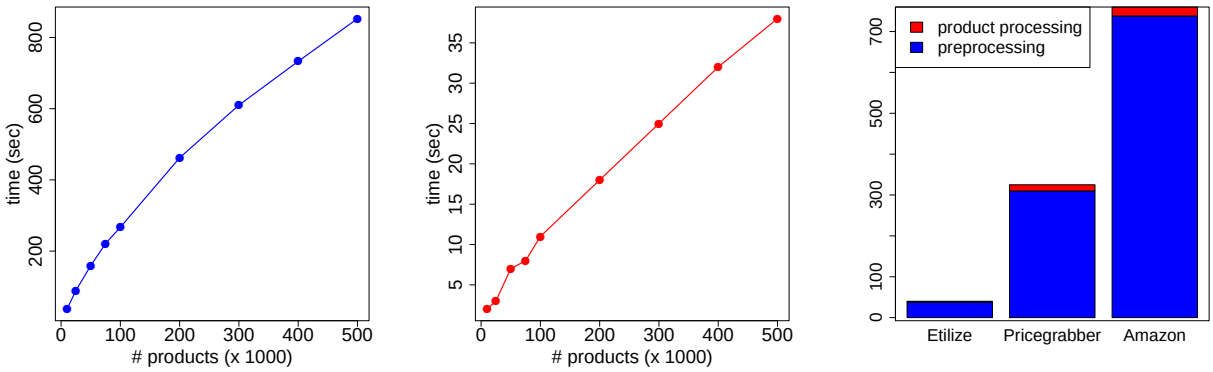
Regarding the separation cost, Absolute Difference outperforms the Multiplicative Cost for most combinations of datasets and similarity metrics. A possible explanation for this is the symmetric property of Absolute Difference Cost. Because of this property, TACI helps eliminate base classifier mistakes that classify products from dissimilar categories in the source taxonomy to close-by categories in the target taxonomy.

Regarding the similarity metrics, the LCA similarity seems to perform the best overall. It has the highest accuracy for Etilize and Pricegrabber, and it is close to the best performance for Amazon. The LCA metric captures nicely the intuition that the similarity of two nodes in the taxonomy is determined by the point at which they merge (or split). It also fits well with the TACI approach that tries to make use of mappings between ancestral nodes of two categories in order to bias the classification. This intuition is not as well captured by the Shortest-Path metric that does not take into account the position of two nodes in the taxonomy. The Cosine similarity is similar to LCA, but it penalizes differently two nodes depending on how deep they are in the taxonomy. This is reasonable, but given that the parent relation in a taxonomy implies also subsumption (every SLR digital camera is also a digital camera), it is not always intuitive.

5.4 Running Time

In this section we study the running time of our algorithm with respect to the number of products and the number of categories in the source taxonomy. We ran our experiments on a Windows Server machine with two 4-core Zeon E5504 processors at 2.0GHz. We parallelized the algorithm for the preprocessing step to make use of the full power of the 8 cores.

In our first experiment, we study the running time of the taxonomy-aware calibration step as the num-



(a) Preprocessing time (Pricegrabber) (b) Processing time (Pricegrabber) (c) Running time (all datasets)

Fig. 2. Scalability experiments: Preprocessing and processing time for different samples of the Pricegrabber dataset. Comparison of total execution time for all three datasets.

ber of source catalog product increases using the Pricegrabber dataset. In particular we subsampled 9 different datasets from the Pricegrabber dataset of sizes 10K, 25K, 50K, 75K, 100K, 200K, 300K, and 500K. Each sample is a subset of the immediately larger sample We use the parameters γ, k and θ that we optimized for the full Pricegrabber dataset.

Figures 2(a) and 2(b) show the preprocessing time (the loop in lines 12-13 of Algorithm 1), and the product processing time (the loop in lines 13-14) against the number of products in the dataset. The first observation by looking at the y-axis scale for the two plots is that the running time is dominated by that of the preprocessing step, which is an order of magnitude larger than that of the processing step. This is expected, since in the preprocessing step we perform essentially all of the computation necessary for classifying the open products in O_θ. Thus the running time of the processing step is small, and scales in a linear way.

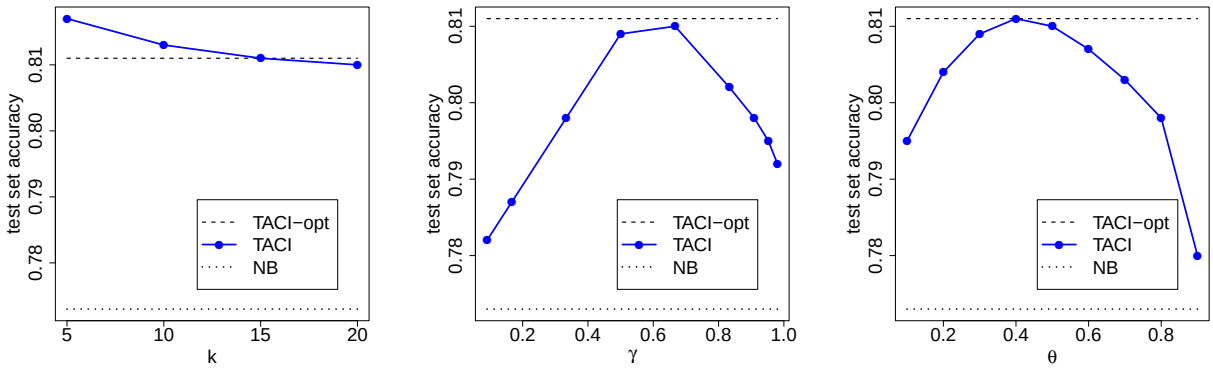
The running time of the preprocessing step depends on the number of candidate source-target category pairs that we need to consider. As more products are introduced, the number of candidate pairs increases. For very large datasets the number of candidate pairs should saturate, and it should remain constant. We do observe a concave trend in our plot (the increase in the running time becomes smaller as the data size grows), but our dataset is not large enough for it to be clearly demonstrated. However, it is clear that the preprocessing time grows in a sublinear way, preserving our requirement for linear time processing.

In the second experiment we study the running time of the TACI algorithm as the number of source catalog categories increases. We applied the TACI algorithm for all three datasets using the same number (50K) of products, so that we understand the effect of the spread of the products over different categories. The 50K Etilize products come from 201 categories, the 50K Pricegrabber products come from 653 different categories, and the 50K Amazon products come from 1,584 different categories. Figure 2(c) shows how the preprocessing and processing running time varies for the set of 50K products from the tree different datasets. The running time is again dominated by the preprocessing time. Notice that in practice the running time increases almost linearly with the number of categories. This increase is significantly lower than the quadratic increase computed by the worst case analysis.

5.5 Parameter Sensitivity Analysis

The tuning of the parameters is important to the correct operation of the TACI algorithm. We rely on a validation set for this purpose. The validation set is a small subset of the source catalog data for which we have the true labeling in the target taxonomy. Such a labeling is created through manual effort, or automated techniques such as matching unique identifiers (e.g., UPC codes). Since the size of the validation set is typically small, this requires a manageable amount of effort. Still, there are cases where it may be hard to obtain such a validation set, or where the validation set is limited only to subsets of the catalog. In such cases we can only obtain rough estimates of the best possible parameters.

In this section we study the sensitivity of the TACI algorithm to the setting of the parameter values. For each of the three parameters k, γ, and θ, we consider several possible values and we will show that there is a large range of parameter values for which we obtain results comparable to those obtained when the parameter values are set by the calibration step. For the parameter k that determines the number of candidate categories that we consider for an item, we consider four different values $K = \{5, 10, 15, 20\}$. For the parameter γ that determines the tradeoff between the assignment and separation cost, we consider seven different possible values such that the ratio $\frac{\gamma}{1-\gamma}$ takes values in the set $\Gamma = \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$. For the parameter θ that determines the threshold on the confidence of the base classifier, we consider ten different values. In order to deal with the variability of the output of the base classifier, and to be consistent



(a) Parameter k (b) Parameter γ (c) Parameter θ

Fig. 3. Sensitivity experiments on the Amazon dataset.

with the way we choose the values in the calibration step, we select the values for θ such we can control the fraction of the products for which we need to apply the taxonomy aware processing step. Since we have ten values for θ, this means that for the i-th value of θ, $10i\%$ of the products have probability less than the threshold, and thus need to be processed.

In our experiment, for each dataset we consider the version of the TACI algorithm that obtains the best accuracy in Section 5.2. This is the TACI-LR for Etilize and Pricegrabber, and the TACI-NB for Amazon. When testing the sensitivity of one of the parameters, we fix the other two to the values we used in the experiment in Section 5.2. We run the experiment over the full labeled truth set, and we measure the accuracy of the specific parameter setting. In Figure 3 we present the results for the Amazon dataset. The results of the other two datasets showed similar trends and they are omitted due to space constraints. For each of the three parameters, we plot the accuracy we obtain for the values of the parameter we consider. In the plots the accuracy for the different parameter values is denoted by TACI. For comparison's sake we also plot the accuracy of the base classifier, and that of the TACI algorithm with the parameters we obtained from the calibration step (denoted as "TACI-opt" in the plots).

Figure 3(a) shows that the algorithm does not seem affected from the choice of the parameter k; we obtain more or less the same accuracy for all values of k. Surprisingly, smaller k values seem to do better, and $k = 5$ outperforms the value $k = 15$ that we used in our experiments. However, the difference in accuracy is small.

As expected, the performance is more sensitive to the setting of the tradeoff parameter γ that balances the weight between the assignment and the separation costs. Figure 3(b) shows that for values of γ in the range between 0.5 and 0.75 the TACI accuracy is close to the optimal. However, values of γ that are out of this range result in significantly worse accuracy, although it is still higher than the baseline NB accuracy.

The observations are similar for the setting of parameter θ. In Figure 3(c) we plot the accuracy versus the fraction of products that are below the threshold value of θ. When this fraction is between 0.2 and 0.5, we obtain accuracy that is comparable that of the calibrated TACI algorithm. For other values of θ the accuracy drops significantly, but we still obtain some improvement over the base classification.

In conclusion, the TACI algorithm is obviously affected from the setting of the parameters. However, when varying the parameters the accuracy plots are mostly smooth, and there is a range of values for all parameters for which the algorithm achieves performance comparable to that of the calibrated parameter values. Thus, TACI is not overly sensitive to the exact setting of the parameters, and it can perform comparably well for a wide range of parameter values.

6 Related Work

Catalog Integration. To the best of our knowledge, no other scalable catalog integration approach exploits the structure of the product taxonomies. Previous work makes use of source category information, but treats the source and target taxonomies as flat. In the experimental section, we compared our method with the one of Agrawal and Srikant [1]. Their method scales to large datasets (like ours), but it showed lower classification accuracy than our method in the experiments. Sarawagi et al. introduce *cross-training* [28], which is an approach to semi-supervised learning in the presence of multiple label sets. Unlike our approach, they assume the existence of some training data labeled using both the source and master taxonomy. Their experimental results show that the accuracy of cross-training in the catalog integration problem is comparable to the method of Agrawal and Srikant. Finally, Zhang et al. have also developed approaches to catalog integration by using boosting [33] and transductive learning [32], [34]. Although these approaches achieve better classification accuracy than AS, similar to the cross-training approach, they require training data that are labeled in both the source and the target taxonomies. So such methods are not applicable to our problem setting.

Nandi and Bernstein [20] propose an approach for matching taxonomies based on query term distributions. The approach is quite different to ours. First, it performs the mapping at the taxonomy level, mapping categories from the source to the target, while we perform the mapping at the instance level by categorizing individual product instances to the target taxonomy. Secondly, the approach is not based on classification but rather on exploiting distributions of terms associated to the categories.

Metric Labeling and Structured Prediction. Our formulation of the catalog integration problem as an optimization problem is inspired by the metric labeling problem that was introduced in [12]. In the metric labeling problem, the goal is to find the optimal labeling of some objects so that they minimize an assignment and a separation cost. The problem is NP-hard [12] and the different existing approximate solutions formulate it as an LP [12] or a QP [23]. The complexity of these methods makes them inapplicable to large-scale datasets with more than a few hundreds of products. The objective of our optimization problem is also similar to the objective that arises in computer vision problems [4], [5], [13]. The most popular application is image restoration, where the goal is to restore the intensity of every pixel in an image using the values of the observed intensities. The algorithms developed in this area focus on separation costs defined in the Euclidean space, i.e., the similarity of two items decreases linearly with their euclidean distance. Although such algorithms are scalable, they cannot be adapted to the separation cost definitions that are suitable for taxonomies.

Structured prediction—the study of machine learning algorithms whose goal is to predict complex objects with internal statistical dependencies—is an active area of machine learning research [3]. It has direct applications to natural language processing, in which most prediction problems *are* structured in nature: sequences of syntactic or semantic labels for words in a sentence (part of speech tagging or entity recognition), syntactic trees (parsing), foreign language sentences (machine translation), graph matchings (word- or sentence-alignment) or logical forms (semantic parsing). In these areas, research has primarily focused either on the computational problem of test-time inference, which is typically a discrete optimization problem [9], [14], [26], [27], [29], or of model training, which typically involves optimizing over an exponentially large set of potential outputs [7], [15], [25], [30]. Like these natural language processing problems, our work involves recognizing statistical dependencies in structured data; the primary difference is that in our setting, the structures are always known ahead of time and our goal is to leverage the information present in these structures. The most similar line of work in structured prediction to our setting is that of bipartite graph alignment [10], [11], [19], which, in our application domain, would amount to trying to learn a generic taxonomy alignment model for arbitrary new portals making use of additional information in the taxonomies.

Ontology Alignment and Schema Matching. There is a large body of work in ontology alignment. Representative examples include Glue [8], a system that uses machine learning to learn how to map between ontologies; and Iliads [31], a system which makes use of machine learning and logical inference techniques to output alignments. In general, the focus in ontology alignment is to map nodes of a source taxonomy to nodes of a target taxonomy. In contrast, in our work we are not interested in solving the (much harder) alignment problem between taxonomies, but rather given an *instance* (i.e., a product) the goal is to categorize it in the target taxonomy using aids from the taxonomy structure. The end-goal is always the categorization of the product. This distinction is very important in many practical scenarios. For example, suppose that the source taxonomy contains the leaf category Cameras, and that it has no nodes that distinguish between different types of cameras. Suppose that the target taxonomy contains the categories Cameras, Cameras/SLR Digital Cameras, and Cameras/Compact Cameras. An ontology alignment technique would only be able to tell that the category Cameras in the source corresponds to the category Cameras in the target. However, given an actual product whose source category is Cameras, it is necessary to employ instance-level techniques like the ones presented in this paper, to decide whether the product corresponds to Cameras/SLR Digital Cameras or Cameras/Compact Cameras.

Similar considerations apply to schema matching techniques [22]. Such techniques find correspondences between elements of different schemas, such as tables and attributes. While the correspondences may be obtained by exploiting (aggregated) instance information, the output of schema matching techniques is always given at the schema level. In our context, that represents associating categories of the master taxonomy to categories of the provider taxonomy. In contrast, as we argued above, our techniques deal with the different problem of determining for each input *data instance* element the appropriate category in the taxonomy. While schema matching techniques typically exploit schema structure (e.g., a graph with edges based on foreign keys relations in similarity flooding [17]), the leveraged structure is significantly different from the one that we exploit in this paper, namely hierarchical relationships in a taxonomy.

7 Conclusions

In this paper, we presented an efficient and scalable approach to catalog integration that is based on the use of source category and taxonomy structure information. We also showed that this approach leads to

substantial gains in accuracy with respect to existing classifiers.

While we focused on shopping scenarios, our techniques are relevant to many other important domains. In particular, they are applicable to classification in any domain where there is a concept of a master taxonomy and there are information providers which use their own taxonomy to label the items that they provide. This includes important verticals such as Local, Travel, Entertainment, etc. One example in Entertainment is the integration of media for streaming purposes. For instance, the Xbox Dashboard now provides the ability to access movies and TV shows from multiple providers, such as Netflix, Hulu, different TV networks, etc. These providers use their own taxonomy to label movies and shows, and thus the need to properly organize them under a master taxonomy. As another example, in the Local domain, different providers may label restaurants in a different way. For example, one provider may tag a restaurant as "Ethnic/Greek" while another may tag it as "Mediterranean".

While our techniques were used for classification, they can also be used for other problems. For example, their output could be be used as a feature for item matching, when we want to match elements classified under the master taxonomy (e.g., the products in the master catalog) to incoming offers from the providers.

For future work, we would like to explore semi-supervised learning techniques to incrementally retrain the base classifier with elements chosen during the taxonomy-aware calibration step. Finally, it would be interesting to use our techniques in an active learning setting, in order to identify candidate products for labeling.

REFERENCES

[1] R. Agrawal and R. Srikant. On integrating catalogs. In *WWW*, pages 603–612, New York, NY, USA, 2001. ACM.

[2] G. Bakir, T. Hofmann, B. Schlkopf, A. Smola, B. Taskar, and S. Vishwanathan. *Predicting Structured Data*. MIT Press, 2007.

[3] G. Bakir, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, and S. Vishwanathan, editors. *Predicting Structured Data*. MIT Press, 2007.

[4] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.

[6] C. Chekuri, S. Khanna, J. S. Naor, and L. Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *SODA*, pages 109–118, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[7] H. Daumé III, J. Langford, and D. Marcu. Search-based structured prediction. *Machine Learning Journal*, 2009.

[8] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal*, 12(4):303–319, 2003.

[9] T. Finley and T. Joakims. Training structural SVMs when exact inference is intractable. In *International Conference on Machine Learning (ICML)*, 2008.

[10] A. Fraser and D. Marcu. Getting the structure right for word alignment: Leaf. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

[11] J. Graca, K. Ganchev, and B. Taskar. Learning tractable word alignment models with complex constraints. *The Computational Linguistics Journal (CL)*, 36(3), 2010.

[12] J. Kleinberg and Éva Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and markov random fields. *J. ACM*, 49(5):616–639, 2002.

[13] V. Kolmogorov and R. Zabih. What energy functions can be minimizedvia graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

[14] A. Kulesza and F. Pereira. Structured learning with approximate inference. In *NIPS*, 2007.

[15] P. Liang, H. Daumé III, and D. Klein. Structure compilation: Trading structure for features. In *International Conference on Machine Learning (ICML)*, 2008.

[16] T.-Y. Liu, Y. Yang, H. Wan, H.-J. Zeng, Z. Chen, and W.-Y. Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explor. Newsl.*, 7(1):36–43, 2005.

[17] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.

[18] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[19] R. C. Moore, W. tau Yih, and A. Bode. Improved discriminative bilingual word alignment. In *ACL*, 2007.

[20] A. Nandi and P. A. Bernstein. Hamster: Using search clicklogs for schema and taxonomy matching. *PVLDB*, 2(1):181–192, 2009.

[21] C. Pesquita, D. Faria, A. O. Falco, P. Lord, and F. M. Couto. Semantic similarity in biomedical ontologies. *PLoS Comput Biol*, 5(7):e1000443, 07 2009.

[22] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4), 2001.

[23] P. Ravikumar and J. Lafferty. Quadratic programming relaxations for metric labeling and markov random field map estimation. In *ICML*, pages 737–744, New York, NY, USA, 2006. ACM.

[24] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[25] S. Ross, G. J. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AIStats*, 2011.

[26] A. M. Rush and M. Collins. Exact decoding of syntactic translation models through lagrangian relaxation. In *ACL*, 2011.

[27] A. M. Rush, D. Sontag, M. Collins, and T. Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *EMNLP*, 2010.

[28] S. Sarawagi, S. Chakrabarti, and S. Godbole. Cross-training: learning probabilistic mappings between topics. In *KDD*, pages 177–186, New York, NY, USA, 2003. ACM.

[29] V. Stoyanov, A. Ropson, and J. Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *AISTATS*, 2011.

[30] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453–1484, Sep 2005.

[31] O. Udrea, L. Getoor, and R. J. Miller. Leveraging data and structure in ontology integration. In *SIGMOD*, pages 449–460, New York, NY, USA, 2007. ACM.

[32] D. Zhang and W. S. Lee. Web taxonomy integration through co-bootstrapping. In *SIGIR*, pages 410–417, New York, NY, USA, 2004. ACM.

[33] D. Zhang and W. S. Lee. Web taxonomy integration using support vector machines. In *WWW*, pages 472–481, New York, NY, USA, 2004. ACM.

[34] D. Zhang, X. Wang, and Y. Dong. Web taxonomy integration using spectral graph transducer. In *ER*, pages 300–312, 2004.

APPENDIX A
THE SEPARATION COST

To define the separation cost we need to define the similarity and penalty functions. In this section we discuss different possible definitions for both functions and outline their properties. We also discuss a normalization step that we apply to the separation cost function in our experiments, in order to alleviate the effects of categories with large number of products.

A.1 Category Similarity

Category similarity in a taxonomy is a well-studied topic, and different metrics have been proposed. Catia et al. [21] provide a good summary of such metrics. In this paper we present some representative metrics and we compare them in the experiments section. Recall that the value of $\mathrm{sim}_G(c_1, c_2)$ is the similarity of categories c_1 and c_2 in the taxonomy G. We consider the following similarity measures.

Shortest-Path similarity: This similarity measure is defined using the *shortest path* between two categories in the taxonomy tree. More specifically, the similarity between c_1 and c_2 is defined as follows:

$$\mathrm{sim}_G(c_1, c_2) = 2^{-\mathrm{hops}(c_1, c_2)} \quad (9)$$

where $\mathrm{hops}(c_1, c_2)$ is the length of the shortest path between c_1 and c_2 in taxonomy G. We also apply an exponential decay function in order to further penalize categories that are far apart. For example, if $c_1 = c_2$ then $\mathrm{sim}_G(c_1, c_2) = 1$, if c_1 is the parent of c_2 then $\mathrm{sim}_G(c_1, c_2) = 1/2$, and if c_1 and c_2 have the same parent then $\mathrm{sim}_G(c_1, c_2) = 1/4$.

The shortest-path similarity has the property that the similarity is maximized when $c_1 = c_2$, which is a desirable property for a similarity function. However, since we use the similarity between categories to define similarity between products, this has the effect that two products under a high-level category such as Electronics are as similar as two products under a deep, specific category such as Electronics/Car Electronics/Car Audio/Car Speakers. Similarly, for two categories c_1 and c_2 that are children of the same node in the taxonomy, their similarity will always be $\mathrm{sim}_G(c_1, c_2) = 1/4$, regardless of whether they are children of the root node, or of a node deep in the taxonomy. This stems from the fact that the similarity definition does not take into account the position of the categories within the taxonomy tree, and the semantics of the subsumption relationship defined by the edges in a taxonomy.

LCA similarity: This similarity measure is defined using the depth of the lowest common ancestor $\mathrm{lca}(c_1, c_2)$ of categories c_1 and c_2. More specifically, the similarity between c_1 and c_2 is defined as follows:

$$\mathrm{sim}_G(c_1, c_2) = 1 - 2^{-\mathrm{depth}(\mathrm{lca}(c_1, c_2))} \quad (10)$$

The intuition for the use of the LCA comes from the work of Resnik [24]. Categories *deeper* in the taxonomy, i.e., further away from the root node, are more narrow and specific than categories higher in the taxonomy, i.e., closer to the root. The LCA category of c_1 and c_2 is the most specific category that subsumes both categories. The deeper this category is, the higher the similarity between c_1 and c_2.

For example, in the master taxonomy tree in Figure 1 categories Electronics and Software have similarity $1/2$, while categories Electronics/Car Electronics/Car Audio/Car Speakers and Electronics/Car Electronics/Car Audio/Car Amplifiers have similarity $7/8$. Note that in this definition if $c_1 = c_2 = c$ then $\mathrm{sim}_G(c_1, c_2) = 1 - 2^{-\mathrm{depth}(c)}$. The similarity increases the deeper the category c, and it is not maximized. This makes sense in our scenario where we want products under a high-level category to be less similar than products under a deeper more specific category.

Cosine similarity: For this similarity measure, we represent a category c using the categories in the path from the root to category c. Let $\{x_0, x_1, ..., x_d\}$ denote the set of categories in path from the root $x_0 =$Root, to the category $x_d = c$, where $d = \mathrm{depth}(c)$ and $x_i \in C$ are categories in the taxonomy tree. We represent a category c as a binary vector v_c over the category space C, that takes value 1 for each category $x \in C$ in the path from c to the root. Formally,

$$v_c[x] = \begin{cases} 1 & \text{if category } x \in \{x_0, x_1, ..., x_d\}; \\ 0 & \text{otherwise.} \end{cases}$$

For example, the master taxonomy tree in Figure 1 has 12 category nodes in total (including the root), and thus each category c is represented as a 12-dimensional binary vector. The vector for category Electronics takes value 1 for the entries of categories Root and Electronics and zero for all other entries. The vector for category Electronics/Car Electronics/Car Audio takes values 1 for the entries of categories Root, Electronics, Electronics/Car Electronics and Electronics/Car Electronics/Car Audio.

Given the vector representation we can now define the similarity of two categories c_1 and c_2 as the cosine similarity of their corresponding vectors v_1 and v_2. The cosine of the two vectors is defined as

$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{|v_1||v_2|}$$

Where $v_1 \cdot v_2$ is the dot product of the two vectors and $|v|$ is the length of the vector v. Note that the number of 1's in a category vector v_c is equal to the number of categories in the path of category c to the root, which is equal to $1 + \mathrm{depth}(c)$. Therefore, $|v_c| = \sqrt{1 + \mathrm{depth}(c)}$. The dot product of two binary vectors is equal to the number of entries for which both vectors have value 1. Since the vectors represent paths in the tree, this is the number of categories in

the path from the least common ancestor to the root. That is, $v_1 \cdot v_2 = 1 + \text{depth}(\text{lca}(c_1, c_2))$.

Therefore, we can compute the cosine similarity between categories c_1 and c_2 as follows:

$$\text{sim}_G(c_1, c_2) = \frac{1 + \text{depth}(\text{lca}(c_1, c_2))}{\sqrt{1 + \text{depth}(c_1)}\sqrt{1 + \text{depth}(c_2)}} \quad (11)$$

Using this similarity measure, two categories c_1 and c_2 are similar if their lowest common ancestor is deep in the taxonomy, and categories c_1 and c_2 are close to their least common ancestor, that is their shortest path is small. For example, in the master taxonomy tree in Figure 1 categories Electronics/Car Electronics and Electronics/Home Audio are more similar than Electronics/Car Electronics/Car Audio and Electronics/Home Audio/Speakers since the former pair is closer to their least common ancestor Electronics. The measure has also the property that similarity of a category to itself is 1. Furthermore, categories that are deeper in the taxonomy are more similar than categories that are higher in taxonomy.

Given the set representation of a category c, we can also use other set similarity measures such as the Dice and Jaccard coefficient. We experimented with these similarity measures, and we obtained results very similar with the cosine similarity, so we do not report them in this paper.

We note that our problem formulation is not tied to any specific similarity measure; any similarity measure can be utilized. We compare experimentally the measures we defined in this section in Section 5.3 where we also discuss how their properties affect their performance.

A.2 The Penalty Function

To complete the definition of the separation cost, we need to define the penalty function δ that we will use. In this paper we consider the following two definitions for the penalty function, and, as a result, two different definitions for the the separation cost.

Absolute Difference Cost: The penalty function is the absolute difference between the similarities in the two taxonomies:

$$\text{SCOST}(s_x, s_y, \ell_x, \ell_y) = |\text{sim}_S(s_x, s_y) - \text{sim}_T(\ell_x, \ell_y)| \quad (12)$$

The absolute difference cost is symmetric: it penalizes both the placement of similar products in dissimilar categories, and the placement of dissimilar products in similar categories.

We also experimented with the squared difference between the similarities as a candidate penalty function. We observed similar results, and we do not report them in this paper.

Multiplicative Cost: The penalty function is the dissimilarity $(1 - \text{sim}_T(\ell_x, \ell_y))$ of the categories ℓ_x and ℓ_y in the target taxonomy T, weighted by the similarity $\text{sim}_S(s_x, s_y)$ of s_x and s_y in the source taxonomy S:

$$\text{SCOST}(s_x, s_y, \ell_x, \ell_y) = \text{sim}_S(s_x, s_y)(1 - \text{sim}_T(\ell_x, \ell_y)) \quad (13)$$

This definition is inspired by the metric labeling problem [12] and yields an asymmetric cost. That is, the cost penalizes similar products (high $\text{sim}_S(s_x, s_y)$) that are classified to dissimilar target categories (low $\text{sim}_T(\ell_x, \ell_y)$), but it does not penalize heavily dissimilar products that are placed in similar categories. Such a property is useful if the target taxonomy contains combined categories that correspond to different categories in the source taxonomy, e.g., target has category Computing & Electronics while the source taxonomy has two separate categories Computing and Electronics.

We discuss the differences between the two penalty functions along with their performance comparison in Section 5.3 of the experiments.

A.3 Normalization

In our experiments we normalize the separation cost $\text{SCOST}(s_x, s_y, \ell_x, \ell_y)$ by a *normalization parameter* $\rho(s_x, s_y)$. The reason for the normalization is that in practice we observed that the distribution of products into the source taxonomy may have a significant impact on the results. Categories that have a very large number of products can pull in products from distant categories, simply because of their size. This is common in large web taxonomies, where the distribution of the number of objects per category is often very skewed (e.g., power-law or log-normal [16]). In these cases, separating the products from small-size source categories from the products of a large source category incurs a high penalty, and thus our optimization technique will tend to merge products from small categories together with those of the large category in the target taxonomy. The normalization parameter serves the purpose of eliminating this "gravitational pull" of categories with very large number of products.

We calculate the normalization factor ρ as follows. Let x and y be two products, and s_x and s_y be their respective source categories. We compute $\rho(s_x, s_y)$ as the total number of products whose source category is at the same similarity radius from s_x as s_y:

$$\rho(s_x, s_y) = \textstyle\sum_{\sigma:\text{sim}_G(s_x,\sigma)=\text{sim}_G(s_x,s_y)} n(\sigma) \quad (14)$$

where $n(\sigma)$ is the total number of products in source category σ. Note that the normalization factor it is asymmetric, that is, it is a normalization with respect to x instead of being symmetric with respect to the pair x, y. It is easy to make it symmetric by taking the sum or average. However, we define it asymmetric on purpose, since in Section 4 we compute the separation of an element x with respect to a set of already classified elements. This normalization factor was shown to perform well in our experiments, prohibiting large categories from attracting too many products.