Large-scale Item Categorization for e-Commerce

Dan Shen[*]
dianping.com
No. 492 AnHua Rd
Shanghai, China
dan.shen@dianping.com

Jean-David Ruvini
eBay Research Labs
2145 Hamilton Avenue
San Jose, CA 95032, USA
jruvini@ebay.com

Badrul Sarwar[†]
eBay Research Labs
2145 Hamilton Avenue
San Jose, CA 95032, USA
bsarwar@ebay.com

ABSTRACT

This paper studies the problem of leveraging computationally intensive classification algorithms for large scale text categorization problems. We propose a hierarchical approach which decomposes the classification problem into a coarse level task and a fine level task. A simple yet scalable classifier is applied to perform the coarse level classification while a more sophisticated model is used to separate classes at the fine level. However, instead of relying on a human-defined hierarchy to decompose the problem, we we use a graph algorithm to discover automatically groups of highly similar classes. As an illustrative example, we apply our approach to real-world industrial data from eBay, a major e-commerce site where the goal is to classify live items into a large taxonomy of categories. In such industrial setting, classification is very challenging due to the number of classes, the amount of training data, the size of the feature space and the real-world requirements on the response time. We demonstrate through extensive experimental evaluation that (1) the proposed hierarchical approach is superior to flat models, and (2) the data-driven extraction of latent groups works significantly better than the existing human-defined hierarchy.

Categories and Subject Descriptors

H.3.2 [**Information Storage**]: Record classification

General Terms

Algorithms, Performance, Experimentation

Keywords

Classification, Text

1. INTRODUCTION

Online commerce has gained a lot of popularity over the past decade. Large online consumer-to-consumer (C2C) marketplaces such as eBay.com and Amazon.com, feature a very large and long-tail inventory with millions of *items* (product offers) entered into the marketplace every day. To structure and manage items effectively and help buyers find them easily, these web sites often organize items into a taxonomy of fine-grained categories. For instance, the eBay category structure has approximately 20,000 leaf categories which cover almost all of the goods that can be legally traded in the world.

Item categorization is fundamental to many aspects of an item life cycle for e-commerce sites. A correct categorization of items is essential for tasks such as extracting relevant item-specific metadata or attributes, assigning category specific rules for listing policy enforcement, charging insertion and final value fees, determining reasonable shipping and handling fees and so on. Moreover, the quality of item categorization plays a significant role in subsequent customer facing applications such as search, product recommendation, trust & safety, product catalog building, and seller utilities. A correct item categorization system is also essential for user experience as it helps determine the relevant presentation logic in surfacing the items to users through search and browsing.

Item categorization can be formulated as a supervised classification problem where the categories are the target classes and the features are the words composing some textual description of the items. Text classification has been well studied in recent years. Classification algorithms such as Naïve Bayes [14], K-Nearest-Neighborhood (kNN) [28], Decision Tree (DT) [1], Support Vector Machines (SVM) [11] and Maximum Entropy(ME) [16] have demonstrated satisfactory empirical results in various domains of text classification. However, the computational complexity involved in some of the state-of-the-art learning algorithms is well beyond linear with respect to the numbers of training examples, features, or classes. For an e-commerce site like eBay the item categorization problem translates into a supervised classification with $20,000+$ target classes. A recent study of eBay's categorization problem describes a supervised method that is trained using several million items and approximately 3 million unique features. The system classifies several millions of newly listed items on-the-fly everyday into these $20,000+$ classes (see [21] for more details). The computational complexity may arise due to polynomial variable selection, e.g., to compute information gain in decision tree based learning, or iterative optimization algorithms, such as in maximum entropy or SVM learning. The scale of the classification problem on e-commerce web sites

[*] Work done while the author was at eBay Research Labs.
[†] Corresponding author

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

requires algorithms capable of processing huge volume of training data in reasonable time, capable of handling large number of classes and also capable of making fast real-time predictions.

In this paper, we describe a novel approach to leverage computationally demanding classification algorithms on very large datasets. We don't use any pre-existing hierarchy but rather learn a new hierarchy automatically from the data. The basic idea is to 1) group classes into disjoint sets called *latent groups* and 2) to decompose the classification task into two classification tasks, coarse level classification and fine level classification. The coarse level classifier is responsible for classifying items into one latent group while the fine level classifiers (typically one such classifier per latent group) are responsible for assigning items to the right class in a given latent group. In other words, our approach consists in using a simple classifier to make coarse-grained classification and a sophisticated classifier to make fine-grained distinctions.

Clearly, one key element of this approach is the way these latent groups are identified. Given the scale of the classification task at hand, it is very impractical to manually construct them and we propose a data-driven solution to discover them automatically. Furthermore, our approach aims at minimizing inter latent group similarity while maximizing intra latent group similarity[1]. This is very important as low inter latent group similarity makes coarse level classification relatively easy and allows the use of a simple and scalable machine learning algorithm. High intra latent group similarity makes fine level classification relatively harder but, because latent groups comprise only a subset of the classes and involve a much smaller volume of data, more sophisticated algorithms can be leveraged for this second classification task.

Comparing with the flat classification approach which treats classes separately, our decomposition of the classification problem has the following advantages:

- The constraints on the choice of learning algorithm are largely alleviated since the numbers of training samples, features, and classes are considerately reduced in the fine level classification step. Sophisticated machine learning algorithms, which cannot be directly applied for flat item classification due to their computational complexity, can now be leveraged.

- The decomposition approach can lead to more problem-specialized feature selection and hence more accurate classifier. Intuitively, many locally predictive features are not useful discriminators in the flat approach. For instance, a word like *case* is not very discriminative globally since it occurs within so many different classes. But at fine level classification time, it may play a crucial role to differentiate between cell phone devices and some of their accessories. As an example, we have empirically shown that the word *case* is the most indicative feature to assign the item "*Leather Cover Case for Verizon Motorola RIZR Z6tv*" to the category "*Cell Phones & PDAs → Cell Phone & PDA Accessories*" rather than the incorrect category "*Cell Phones & PDAs → Cell Phones & Smartphones*".

In the remainder of this paper, we first review related work and then we describe the 3 components of our approach, namely how we discover the latent groups, the coarse level classifier and the fine level classifier. Finally, we present experimental results on real large-scale e-commerce data.

2. RELATED WORK

Many websites organizes categories into a concept hierarchy or taxonomy. Some notable examples are the International Patent Classification (IPC) schema, the Web catalogs created by Yahoo!, the Open Directory Project (ODP), Amazon.com, eBay.com and most recently the Wikipedia subject hierarchy. A lot of results have been published around leveraging pre-existing concept hiearchies for classification. For instance, Koller and Sahami [12] applied a "gates-and experts" strategy to a small hierarchy using naive Bayes as local classifier, and demonstrated advantages over flat models on a small feature space. Weigend et al. [23] transformed a category hierarchy into a neural network, and multiplicatively combined output probabilities from each level to yield a final classification decision. On a Reuters benchmark dataset, they reported a 5% improvement in average precision by the hierarchical model over the flatten model. Dumais and Chen [6] showed that a hierarchical approach leveraging SVM as a local classifier is particularly suited for web page classification. To better exploit the semantic relationship embedded in a hierarchy, McCallum et al. [15] studied a statistical technique called "shrinkage" to smooth parameter estimates along a hierarchy. Some recent work [3, 26, 4] leveraged the hierarchical structure by developing a tree-induced loss function from a learning theoretical perspective. Readers interested in hierarchical classification may refer to Sillas and Freitas [22] for an extensive survey of this litterature.

Our approach for solving a large text classification problem consists in reformulating it into a two levels (coarse level and fine level) hierarchical classification problem. However, it ignores pre-existing hierarchical information (or any sort of pre-defined parent-child relationship between the categories) and does not belong to this trend of research. Also, while this prior work is methodologically appealing, the data and category collection under study is still relatively small and well organized compared with many real-world commercial applications. For instance, the dataset used by Dumais and Chen [6] contains only 370597 training samples, which is relatively small according to nowadays e-commerce standards.

The idea of simplifying a classification problem by modifying the output space (i.e. the target classes) is not new. Pairwise decomposition [10], one-vs-all decomposition [18] and error correcting output codes [5] are well known examples. However, these methods have a high training time because they require to learn many binary classifiers over most of the training data; have a long classification time because they require to query all these classifiers; and are thus not suitable for large scale classification problems. Kumar et al. [13], and later Bengio et al. [2], proposed algorithms to break down a classification problem into smaller problems by learning a tree structure over the set of classes. Kumar et al. used a clustering algorithm based on Fisher's discriminant to cluster training examples in disjoint groups and thus induce a partitioning of the classes. Although the classification process of this approach is orders of magnitude faster

[1] This notion of similarity will be defined later. For now one can assume it is a function of the similarity of the training samples belonging to these groups.

than previous work, the training process is computationally expensive as it requires to solve many clustering problems. Bengio et al. solution to large scale classification does not involve clustering the training samples but rather uses the confusion matrix of a surrogate one-vs-all classifier as a proxy to estimate class similarity. Two classes are assumed to be similar if there are highly confused together by the surrogate classifier. A spectral clustering algorithm where the edges of the similarity graph are weighted by the class confusion probabilities is then used to group similar classes together. Although theoretically very appealing, Bengio et al. approach leads to an accuracy comparable (and actually worse on one of their datasets) to the accuracy of the flat one-vs-all solution.

Large-scale classification problems have received increased attention in the recent years. Liu et al. [27] conducted a thorough analysis of the behavior of the SVM algorithm with respect to a large-scale classification task and found that its performance is “still far from satisfactory”. Xue et al. [8] proposed an interesting two stage strategy called ”deeped classification”. Whenever a document needs to be classified, the most similar documents in the training set are retrieved using a search approach and the classification task is reduced to the classes these documents belong to. In the second stage, a classifier is trained on these classes and used to eventually classify the document. The main drawback of this approach is that a specific classifier is trained for each document to classify, considerably slowing down the classification process.

In contrast to the approaches mentioned in this section, our solution has a fast training and classification process; allows us to use any off-the-self classification algorithm to separate classes that tend to be highly confused with each other; and leads to improved accuracy as will be shown in the next sections.

3. METHODS

In order to achieve item classification for a very large number of target classes, we propose to decompose the classification problem into smaller problems. This is achieved by grouping classes into mutually exclusive “latent groups” and performing a two stage classification process. First, a coarse level classifier assigns an item into one of the latent groups; second, a fine level classifier assigns a specific class within the latent group. This section describes in details how these three steps are carried out.

3.1 Latent groups discovery

The key element of our approach is obviously how the latent groups of classes are built. Ideally, these groups should be such that similar classes are grouped together and classes that are not similar belong to different groups, so that a simple classifier can be used to separate groups while a sophisticated algorithm may be applied to separate classes within each group.

One way to measure similarity between classes is through the similarity of the training samples that belong to these classes. As mentioned in section 2, this is the approach followed by [13] who used a clustering algorithm to cluster training examples in disjoint groups and thus induce a partitioning of the classes. However, clustering is a hard and computationally expensive process.

As in [2], our approach consists of leveraging the confusion matrix of a classification algorithm to approximate the similarity of classes. More precisely, following the intuition that categories that are hard to separate by a classification algorithm may actually be similar, we approximate the similarity of two classes by the probability of the classifier to incorrectly predict one of the categories when the correct label is the other category. In practice, class confusion probabilities are estimated empirically by (1) training a flat classifier on all the classes and (2) applying the classifier on a development data set. This classifier can be any classifier but, as we will explain below, we recommend using the exact same algorithm for coarse level classification. More formally, the confusion probability between two categories $\mathrm{Conf}(c_1, c_2)$ is defined as follows:

$$\mathrm{Conf}(c_1, c_2) = \frac{\sum_{t: f_s(t) \neq f_m(t), f_s(t) \in \{c_1, c_2\}, f_m(t) \in \{c_1, c_2\}} (1)}{\sum_{t: f_s(t) \in \{c_1, c_2\}, f_m(t) \in \{c_1, c_2\}} (1)},$$

where, $f_s(t)$ is the class of item t and $f_m(t)$ is the class predicted by the model. It is clear that the distance measure is specialized for the task and reflects the difficulty of making a distinction between two classes.

Once the confusion probabilities have been estimated, we use a graph algorithm to generate problem-specific latent groups. We represent the relationship among classes using an undirected graph ($G = (V, E)$), where the set of vertices V is the set of all the classes and E is the set of all edges. Two vertices are connected by an edge if the confusion probability $\mathrm{Conf}(c_1, c_2)$ is greater than a given threshold α (to be discussed later).

If the actual confusion probability of every pair of classes was known, one could formulate the latent group discovery problem as finding all maximum cliques, a clique being defined as a set of vertices any two of which are adjacent. However, and this is a key observation, because the confusion probabilities are estimated using a finite development set, all possible class confusions may not be observed and our graph may lack some edges. As a consequence, we formalize the latent group discovery problem as finding dense subgraphs [9], that is, sets of vertices highly connected with each other, which turns out to be a much easier problem than the *NP-complete* clique problem.

We approximate the dense subgraph enumeration problem using an efficient two steps algorithm we devised specifically for this purpose. The first step consists of randomly imposing a direction to all the edges of our undirected confusion graph. Second, once the graph is directed, we formulate the subgraph enumeration problem as a strongly-connected components enumeration problem. A directed graph or subgraph is called strongly connected if there is a path from each vertex in the graph to every other vertex. We use a well known and efficient algorithm from Tarjan [24] to list all the strongly connected components.

Randomly imposing a direction to edge may seem arbitrary but it practically leads to a good approximation of dense subgraphs. This is due to the fact that the denser a subgraph, the higher the probability that a path between any two nodes exists. However, the sensitivity of the performances of our approach to the algorithm used to identidy the latent groups is a very interesting question and on-going research work.

Algoritm 1 below summarizes the latent group discovery process.

Algorithm 1 Algorithm for latent group discovery

```
Input: i) Set of categories $\mathcal{C} = \{c_1, c_2, ..., c_n\}$
       ii) a threshold value $\alpha$
Output: Set of dense subgraphs $\mathcal{G} = \{g_1, g_2, ..., g_m\}$ that forms a latent group
1: Train a Classifier $\mathcal{H}$ on all flat classes.
2: Compute pair-wise confusion probabilities Conf$(c_i, c_j)$ between classes.
3: Build the confusion graph, $G = (V, E)$
4: Apply the input threshold $\alpha$ to remove loosely connected edges.
5: Run a dense subgraphs approximation algorithm to find a set of dense subgraphs $\{g_1, g_2, ..., g_m\}$ each dense subgraph forms a latent group $\mathcal{G}$.
```

The parameter α controls the size of the latent groups. Higher values of α induce smaller latent groups and thus reduce the computational complexity at the fine classifier level. But classes within these groups are likely to be very similar and harder to separate. The right value for this parameter is problem specific and can be found using a development set.

3.2 Coarse level classification

Once classes have been clustered into latent groups, the coarse level classifier can be trained. The coarse level classifier is responsible for classifying items into one latent group. For this classifier, each latent group is a class (in the machine learning sense) and the training set is the union of the samples of all the classes belonging to that group.

Any scalable classifier can be used at the coarse level and the Naïve Bayes algorithm or the k-Nearest-Neighborhood algorithm are good candidates. However, we recommend using the exact same classifier that was used to estimate the confusion probabilities during the latent group discovery process (see Section 3.1) so that no re-training is needed.

3.3 Fine level classification

The fine level classifiers are dedicated to separating classes within latent groups and are responsible for the final classification, that is assigning a class to an item (as opposed to the coarse level classifier which assigns a latent group). There is one fine level classifier per latent group. Each category in a latent group is a target class for this multi-class classifier.

The dimensionality reduction obtained by training each fine level classifier on a latent group, whose size can be controlled empirically by using the α parameter described in Section 3.1, allows using a sophisticated and computationally demanding algorithm for fine level classification.

The next sections shows, through experiments on real data, that decomposing the classification tasks into coarse level and fine level subtasks, leads to significantly higher classification accuracy.

4. EXPERIMENTS

This section presents experimental results on a real dataset. The goal of this empirical study is to address the following research questions:

- How does the state-of-the-art text classification technologies perform on the large-scale item categorization on e-commerce site?
- Does the item classification task benefit from the hierarchical category structure typically available at e-commerce sites? In other words, we want to compare the performance of flat classification with hierarchical classification models.
- Does the proposed coarse level - fine level method bring any performance gains over the flat classification or more conventional hierarchical classification models?

In the following, we describe the dataset used for these experiments, the algorithms we compared and finally discuss the results.

4.1 Dataset

We collected a sample data from the eBay.com e-commerce site for this experiment.

eBay organizes items (product offers) into a six-level category structure similar to a topic hierarchy, where there are 39 top-level nodes called *meta categories*, and more than 20,000 bottom-level nodes called *leaf categories*. The hierarchy is designed and maintained by human experts. Table 1 shows the leaf category distribution on the hierarchy. It can be observed that the leaf categories are highly skewed over the nodes at higher levels. For the nodes at the first three levels (*Level 1*, *Level 2* and *Level 3*), there can be hundreds or thousands of leaf categories under a node. Our task is to categorize items into the most relevant leaf categories, that is to solve a 20,000+ class classification problem.

An item is typically listed in one category, although it is possible that the item might be suitable for multiple categories by nature. Furthermore, eBay has a very large and complex hierarchical taxonomy in which the topic nodes almost cover all of the goods in the world. The category nodes are not guaranteed to be mutually exclusive with each other, although the hierarchy was designed by human experts and has been extensively refined over the past years. Some of ambiguous leaf categories are even across meta categories, such as:

1. Clothing, Shoes & Accessories → Men's Shoes
2. Sporting Goods → Outdoor Sports → Camping & Hiking → Clothing, Shoes & Accessories → Shoes → Men

Although it seems at first sight that the inventory listed in these categories should be very different, many type of shoes can be listed in both categories and sellers use both categories interchangeably.

The distribution of items over leaf categories exhibits high skewness and heavy-tail nature, as shown in Figure 1. 86.9% of the categories (the head) only containing less than 1,000 items per category, while 1% of the categories (the tail) with more than 10K items per category account for 51.7% of the items. As a consequence, a large portion of the categories

Level	Top nodes	Leaf Nodes within a top node		
		Avg	Max	Min
Level 1 (meta)	39	547.1	4526	3
Level 2	487	43.8	793	1
Level 3	4346	4.9	495	1
Level 4	13904	1.5	80	1
Level 5	19942	1.1	35	1
Level 6 (leaf)	21337	1	1	1

Table 1: Distribution of categories in eBay topic hierarchy. Level 1 represents the "meta" categories and level 6 represents the "leaf" categories

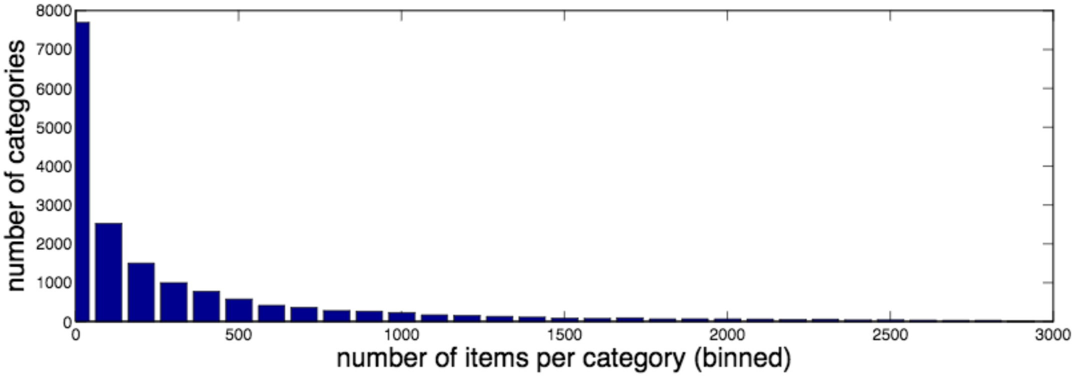

Figure 1: Distribution of items over eBay leaf categories. eBay categories exhibit high skewness with the vast majority of categories holding little inventory.

still suffers from data sparseness in spite of a huge inventory on the whole. The lack of training data for the corresponding classes may adversely affect the performance of machine learning algorithms which require adequate labeled documents for each class to learn reliable statistics. Moreover, many algorithms tend to behave poorly on unbalanced training sets as they tend to favor strongly predominant classes.

It is dauntingly expensive to label items manually for such a large number of categories. To alleviate this problem we tap into the expertise of millions of savvy eBay sellers and use the high volume of existing item inventory labeled by sellers as the proxy of ground-truth labels, under the assumption that items successfully sold have been placed into appropriate categories by sellers. Although the assumption does not always hold, it avoids the otherwise very expensive human labeling, and serves as a good approximation. However, the reduction in cost of human labeling comes at the expense of a quality drop of the training data. We observed, using editorial validation, that approximately 15% of the time the label provided by the seller is incorrect. This error rate in label approximation can give a loose upper bound of actual prediction error, based on observed experimental results.

When a seller lists an item on eBay, he or she is asked to write a title to briefly describe the item. Once the title is entered, the site suggests proper categories for the seller to choose from. So the item categorization is only based on the title words. Unlike typical free text documents, eBay limits the length of item title to a maximum of 50 characters (about 10 words on average), which in many cases is inadequate to describe an item precisely. For example, the following item title does not contain enough information to decide the most appropriate category.

Title: 9.5 White Gold Ring with .25 Carat Diamond Stones Band

- Jewelry & Watches → Men's Jewelry → Rings → Diamond
- Jewelry & Watches → Fine Jewelry → Fine Rings → Diamond
- Jewelry & Watches → Fashion Jewelry → Rings

Furthermore, the quality of titles varies a lot on the eBay site. Some inexperienced or malicious sellers give inaccurate or fraudulent titles, which makes our data set even noisier.

One important criterion for any large-scale production system is scalability. As an item categorization system for a popular e-commerce site, the model needs to be trained offline with hundreds of millions items, and then make online prediction in real-time for an incoming item. There are four million to twenty millions items listed daily on eBay. For the experiment presented here, we randomly collected a sample of 83 million items sold over a three month period of time. We ensured that even the smallest classes have some examples. For testing we used several millions of item sold a few days after the training period.

Although we cannot share the exact dataset we used due to legal reasons, very similar data is publicly available on eBay web site and through eBay Developers API [2].

[2] See https://www.x.com/developers/ebay/ for details.

Level	Top nodes	Leaf Nodes within a top node		
		Avg	Max	Min
Balanced Splitting	4659	4.6	50	1

Table 2: Balanced splitting for the *hier-ebay-struc* baseline model.

4.2 Algorithms

4.2.1 Preprocessing

In data preprocessing step, item titles are first tokenized, and then punctuation is removed. A small stop-word list is used to omit the most common words. Furthermore, we observe that (1) numbers are indicative to distinguish between *single sale* and *whole sale* categories, and (2) prepositions, such as *with* and *for*, are useful to distinguish between *product* and *accessory* categories. Therefore, contrary to other text classification tasks, we keep numbers and prepositions in titles.

Our training set has more than 3 millions unique words after the preprocessing. We use unigrams of title words as features.

4.2.2 Coarse level classifier

For the coarse level classifier we use the K-Nearest-Neighbor (kNN) algorithm but applied feature selection to reduce the feature space.

We employ information gain (IG) as selection criteria which showed promising results in other text classification tasks [29]. We select the top 10% features with the highest IG. The optimal number of features was cross-validated in a foregoing study under a similar setting [21, 20]. There are 322,792 features in the coarse level.

We choose K-Nearest-Neighbor (kNN) algorithm as the coarse level classifier because of its great scalability, linear computational complexity, robustness to data sparseness and skewed category distribution [28], and interpretable category predictions which is especially important for a practical system.

KNN is based on a lazy learning method in which no explicit training process is required and all computation is deferred until classification. It classifies a test instance based on its closest training instances in the feature space. Given a test item, the kNN classification performs two steps: (1) find the k nearest neighbors among the items in training set, and then (2) perform a majority voting amongst the k neighbors to identify the category candidates. We use cosine values of two vectors to measure the similarity between items. The number of neighbors k is dynamically decided as follows: for an item to classify, the neighbors are fetched greedily first in descending order of similarity until the similarity score of a neighbor is below a threshold. The threshold is set as half of the score of the first neighbor. Next, the item is assigned to the class with the majority votes of its k nearest neighbors. The nearer neighbors are supposed to contribute more in the voting than the more distant ones. We have investigated various voting schema in our experiments, and finally used the reciprocal of the rank position to weight the neighbors. The voting function is a generalization of linear interpolation as follows:

$$score(t, c_i) = \sum_{t_j \in \mathrm{KNN}(t)} \frac{1}{R(t_j)} y(t_j, c_i)$$

where, KNN(t) indicates the set the k nearest neighbors of the item to classify t; $R(t_j) = \{1, 2, ..., k\}$ is the rank position of the neighbor t_j; $y(t_j, c_i) \in \{0, 1\}$ is the classification for the training item t_j with respect to the class c_i ($y = 1$ for $t_j \in c_i$, otherwise, $y = 0$). This paper does not aim to report rigorous exploration of the optimal settings for the parameter k, neighbor weighting or voting function in the kNN model. The setting introduced above, however, performed the best in our preliminary experiments of kNN-based flat classification.

The kNN classifier is often considered a slow classifier as intensive computation is needed at classification time to retrieve the neighbors. However, modern search engines make it possible to fetch and sort neighbors in milliseconds and have rendered kNN perfectly suitable for large scale classification problems. For instance, our production implementation of kNN using eBay search engine is able to classify an item in less than 100 ms in spite of a large number of classes and training samples.

4.2.3 Fine level classifier

For the fine level classifier we experimented with the Support Vector Machine (SVM) [25] algorithm. We do not apply feature selection for fine level classification as the feature space in latent groups is much smaller than the overall feature space.

Extensive empirical comparisons on text classification [19, 28] have shown that SVMs achieve excellent state-of-the-art performance.

SVMs construct a binary classifier that predicts whether an instance $\vec{x}$ is positive or negative, where the instance is represented as a feature vector. In the case of linearly separable instances, the decision $f(x) = sgn(\vec{w} \cdot \vec{x} + b)$ is made based on a separating hyperplane $\vec{w} \cdot \vec{x} + b = 0$ ($\vec{w} \in \mathcal{R}^n, b \in \mathcal{R}$). The goal of SVMs is to find the optimal hyperplane that separates the positive and negative training instances with a maximal margin, by solving a dual quadratic programming problem. Given a set $\mathcal{T}$ of training data and corresponding label pairs $(x_i, y_i), i = 1, \ldots, m$, where $x_i \in R^n$ and $y_i \in \{1, -1\}$, SVMs essentially solve the following optimization problem:

$$min_{w,b,\xi} \ \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Here x_i are the training vectors and $C > 0$ is the budget parameter, a penalty for the error term. In case of linearly non-separable data SVMs project the instances from the original space $\mathcal{R}^n$ to a higher dimensional space $\mathcal{R}^N$ based on a kernel function $K(\vec{x_1}, \vec{x_2}) = \Phi(\vec{x_1}) \cdot \Phi(\vec{x_2})$. This is commonly referred to as the *kernel-trick*, which makes it possible to find a linearly separable hyperplane in the projected high dimensional space.. A kernel function has the form $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ and projects the input data x_i into a higher dimensional feature space $\phi(x_i)$. Although there are a number of kernels, in practice most often a linear

Classification Models	Top 1	Top 2	Top 5
flat-NB	64.4	77.1	87.3
flat-kNN	71.8	81.5	88.5
hier-ebay-struc	72.2	81.6	88.0
KNN-SVM	**75.4**	**83.8**	**89.9**

Table 3: Precision of various classification methods when top 1, 2 and 5 categories are returned. The classification methods are flat multinomial NB, flat k-nearest neighbor, hierarchical classifier leveraging eBay category structure and our proposed algorithm *k*NN-SVM.

kernel or radial basis function (RBF) kernels are used. They have the following form:

- linear kernel : $K(x_i, x_j) = x_i^T x_j$.
- RBF kernel : $K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2), \gamma > 0$.

For our fine level classifier, we employed a highly efficient implementation of SVM with linear kernel, LibLinear [7]. This package also supports multi-class classification using one-vs-all strategy.

4.3 Baselines

We evaluated our proposed algorithm (**kNN-SVM**) against three different baselines, namely a flat kNN classifier (*flat-kNN*), a flat multinomial Naive Bayes (*flat-NB*) and a hierarchical classifier leveraging eBay category structure (*hier-ebay-struc*). We describe the baseline methods below.

- *Flat-kNN* is an implementation of the kNN classifier. As described in section 4.2.2, we use information gain (IG) as feature selection method, employ a production search engine for initial candidate selection, and finally use cosine similarity and dynamic neighbor selection for nearest neighbor voting.

- The multinomial NB classifier (*flat-NB*) is a good choice especially for large-scale online classification mainly because of its great scalability. Many previous work [14] reported that it performs quite well empirically. Smoothing plays an important role in NB training to alleviate the problems caused by data sparseness and unbalanced distribution. Since Shen et al. [21] reported quite promising results of the Dirichlet Priors (DP) smoothing in a similar task, we adopt DP smoothing ($\mu = 1000$) in the *flat-NB* model. One notorious drawback of NB classifier though, is that it may produce poorly calibrated probability estimation.

- The third baseline, *hier-ebay-struc*, performs hierarchical classification using the existing six-level topic structure at eBay. The large number of eBay categories makes it hard to train an iteration-based second-level classifier efficiently and accurately, such as DT, SVM or ME. On the other hand, lower levels (*Level 4* and *Level 5*) contain too many nodes to train the first level classifier effectively, e.g., *Level 4* has 13,904 nodes and *Level 5* has 19,942 nodes. To optimize the performance of the hierarchical classification, we propose a balanced splitting method to extract a small number of high-level nodes under which leaf categories distribute more or less evenly. The balanced splitting works as follows. We search on the entire hierarchical tree as in depth-first search (DFS). Once a non-leaf node (internal node) contains less than 50 leaf categories, we extract the internal node as a high-level node, and remove the path from root to the extracted node from further consideration. After exhausting the entire hierarchical tree, the extracted high-level nodes are mutually exclusive and spanning all leaf categories. The row "Balanced splitting" in Table 2 shows the number of high-level nodes (4659 nodes) and the distribution of the leaf categories over these internal nodes. This approach is easy to implement and heavily relies on the existing hierarchy. *hier-ebay-struc* uses this balanced splitting hierarchy. We first apply kNN model, the same as the *flat-kNN* model, to classify items into top-level nodes, and then apply a SVM model to further classify the items into one of the leaf categories. *Flat-kNN* serves as a proxy for one-vs-all classification. Unfortunately we have not been able to scale SVM to our classification problem.

4.4 Results

The item classification system is evaluated using the usual precision metric defined as the fraction of items of the test for which the classifier recovered the correct class label. However, because some applications of item classification on e-commerce require to present several alternatives to the users, we are also interested in measuring accuracy when 2 and 5 classes are returned. This requires the fine level classifiers to return a ranked list of classes instead of a single prediction. For the various classifiers we tested, the ranking is obtained using the confidence score returned by the classifier. In the case of SVM, Platt's scaling [17] was used to obtain such confidence score. For the purpose of measuring accuracy when more than one class is returned, the classifier is considered correct when the gold standard (i.e the category picked by the seller when the item was listed on the site) is one of the returned classes. In the following "Top 1", "Top 2" and "Top 5" denotes the accuracy when 1, 2 and 5 classes are returned respectively.

Table 3 shows the overall accuracies of the flatten classification models (*flat-NB*, *flat-kNN*), the hierarchical classification models (*hier-ebay-struc* and our proposed approach (*KNN-SVM*). At all three recall levels ("Top 1", "Top 2" and "Top 5"), our approach performs the best in terms of accuracy. For "Top 1" in particular, it outperforms *flat-kNN* and *hier-ebay-struc* by about 3.6 percentage points, 3.2 percentage points, and the edge over *flat-NB* is 11 percentage points.

As for the conventional text classification models, *flat-kNN* significantly outperforms *flat-NB* by 7.4 percentage points for "Top 1". This gives the insight that the instance-based learning method, i.e. kNN, might be more suitable

1.	Cell Phones & PDAs → Cell Phone & PDA Accessories Electronics → iPod & MP3 Accessories → Cases
2.	Video Games → Wholesale Lots → Accessories Video Games → Accessories
3.	Jewelry & Watches → Loose Beads → Crystal Crafts → Beads & Jewelry Making → Beads, Pearls & Charms → Beads
4.	Books → Textbooks, Education Books → Nonfiction
5.	Sports Mem, Cards → Fan Shop → Cards → Football Sports Mem, Cards → Fan Shop → Autographs-Original → Football-NFL → Trading Cards
6.	Business & Industrial → Restaurant & Catering → Commercial Kitchen Equipment → Food Preparation Equipment → Food Processors Home & Garden → Inside the Home → Kitchen, Dining & Bar → Small Kitchen Appliances → Food Processors

Table 4: Examples of the category pairs which are frequently misclassified by the flatten classification model *flat-kNN*

Model	First level classification		Second level classification			
	Number of Nodes	Top 1	Number of leafs per node			Top 1
			Avg	Max	Min	
hier-ebay-struc	4659	79.8	4.6	50	1	72.2
KNN-SVM	6494	86.1	3.3	45	1	73.0

Table 5: Number of categories and accuracy for two level hierarchical classification models.

for classifying heterogeneous and short texts, i.e. the items on eBay. One conclusion we can draw here is that data sparseness penalizes *flat-NB* severely while it does not affect *flat-kNN* in the same fashion.

Another interesting finding is that the hierarchical classification model *hier-ebay-struc* does not show significant improvement over the flatten classification model *flat-kNN*. A later manual inspection of the items misclassified by *flat-kNN* suggests one explanation for this—the error cases from *flat-kNN* are beyond adjacent leaf categories on the category structure. This violates one prerequisite of hierarchical classification that it requires satisfactory classification accuracy at the top level and tackles the difficulty of classifying very confusing categories at lower levels. Table 4 shows examples of 6 category pairs which are most frequently misclassified by the flatten classification model *flat-kNN*. It is can be observed that some of the error cases are even across meta categories. Thus in the hierarchical model *hier-ebay-struc*, this type of errors will be made on the top level, which cannot be recovered by SVM models at the lower levels due to algorithmic design.

We now attempt to answer the third question concerning our proposed two level classification with latent group approach. The experiments show that our approach is significantly better than the conventional hierarchical model *hier-ebay-struc*. In Table 5 we show the detailed performance comparison between two hierarchical classification models. For *KNN-SVM*, classes are grouped into 6494 latent groups and each latent group contains 3.3 classes on average. For the top level classification, *KNN-SVM* performs better than *hier-ebay-struc* by 6.3 percentage points, although the number of categories is more than that in *hier-ebay-struc*. It indicates the latent group generation method we proposed in this paper captures class similarities properly, groups the most similar classes effectively, and ultimately ensures satisfactory performance on the top level classification. Table 6 further illustrates some examples of leaf categories within a latent group. It is obvious that the leaf categories in this latent group are actually hard to distinguish from each other despite some of them are quite far away in the eBay hierarchy structure.

Finally, the main motivation of this research is to leverage computationally demanding classification algorithms on very large datasets. The mere fact that we are able to leverage SVM (as a fine level classifier) and achieve higher classification accuracy on a problem where a flat SVM does not scale is a very significant step in that direction. But speed of execution is also a very important criterion in term of scalability and we now show that the goal is achieved on that front as well. The training time of *KNN-SVM* which includes training the kNN coarse level classifier (i.e. indexing the training sample using the search engine), estimating class confusion probabilities, running the dense subgraph enumeration algorithm and training the fine level classifier is about 3 hours for the setting of this experiment using a 10 thread implementation. The classification time of *KNN-SVM* is 520ms per item (for our research prototype, the production implementation taking only 100ms per item). This was measured on a 16 core x86 processor operating at 2.3 Ghz with 128Gb of memory. Table 7 shows a comparison with the multinomial NB classifier and the K-Nearest-Neighbor algorithm. Although *KNN-SVM* is slower than the flat classifiers, it is remarkably fast given the problem involves more than 20,000 classes and 83 million training samples.

1. Clothing, Shoes & Accessories → Men's Accessories → Backpacks, Bags & Briefcases → Duffle, Gym Bags
2. Clothing, Shoes & Accessories → Unisex Clothing, Shoes & Accs → Unisex Accessories → Bags & Backpacks → Backpacks & Bookbags
3. Clothing, Shoes & Accessories → Unisex Clothing, Shoes & Accs → Unisex Accessories → Bags & Backpacks → Duffle, Gym Bags
4. Clothing, Shoes & Accessories → Women's Handbags & Bags → Duffle & Gym Bags
5. Computers & Networking → Computer Accessories → Laptop Cases & Bags → Backpacks
6. Travel → Luggage

Table 6: Example of leaf categories in a latent group. Latent groups are discovered automatically using a graph algorithm. Our algorithm discovers latent group that spans three different meta categories namely, "Clothing, Shoes & Accessories", "Computers and Networking", and "Travel".

Classification model	Training (h)	Classification (ms)
flat-NB	0.66	20
flat-kNN	1	360
KNN-SVM	3	520

Table 7: Training speed (in hours) and classification speed (in millisecond) of two flat models and our two levels approach.

5. CONCLUSION

Item categorization, which can be formulated as a supervised text classification problem, is a fundamental technical challenge for e-commerce sites such as eBay. But the number of classes, the volume of the training data and the size of the feature space make the use of computationally intensive algorithms impractical for large scale deployment. In this paper, we proposed a novel and practical approach that allows leveraging off-the-shelf text classification algorithm by automatically grouping similar classes into latent groups using a dense subgraph enumeration algorithm and decomposing the problem into a coarse level classification task and a fine level classification task. The coarse level classifier is responsible for assigning a latent group while the fine level classifiers are responsible for assigning a class within a latent group. Experiments on large scale real data from eBay show that our approach significantly outperforms other known approaches while maintaining a very practical speed of execution.

For future work we are particularly interested in exploring the sensitivity of this two-levels approach in terms of classification accuracy with respect to the coherence of the latent groups i.e., the method used for enumerating the dense subgraphs.

6. ACKNOWLEDGMENTS

Authors are grateful to Rajyashree Mukherjee, Suresh Raman, Niraj Kothari, Chelly Yan and Wallace Mann for fruitful discussions about this research and for engineering assistance in deploying our algorithm into the production classification system. Thanks to Chih-Jen Lin for useful comments. Also thanks to the anonymous reviewers for providing us with important suggestions.

7. REFERENCES

[1] C. Apté, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Trans. Inf. Syst.*, 12:233–251, 1994.

[2] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[3] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *Proc. of the 13th ACM International Conference on Information and Knowledge Management(CIKM)*, pages 78–87, 2004.

[4] O. Dekel, J. Keshet, and Y. Singer. Large margin hierarchical classification. In *Proc. of the 21st International Conference on Machine Learning(ICML)*, pages 27–34, 2004.

[5] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Artificial Intelligence Research*, 2:263–286, 1995.

[6] S. T. Dumais and H. Chen. Hierarchical classification of web content. In *Proc. of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–263, 2000.

[7] R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, pages 1871–1874, 2008.

[8] Q. Y. G.-R. Xue, D. Xing and Y. Yu. Deep classification in large-scale text hierarchies. In *Proc. of the 31st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 619–626, 2008.

[9] A. V. Goldberg. Finding a maximum density

subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984.

[10] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *Annals of Statistics*, 26:451–471, 1998.

[11] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. of the 10th European Conference on Machine Learning(ECML)*, pages 137–142, 1998.

[12] D. Koller and M. Sahami. Hierarchically classifying docuemnts using very few words. In *Proc. of the 14th International Conference on Machine Learning(ICML)*, pages 171–178, 1997.

[13] S. Kumar, J. Ghosh, and M. M. Crawford. Hierarchical fusion of multiple classifiers for hyperspectral data analysis. *Pattern Analysis and Applications*, 5:210–220, 2002.

[14] A. Macallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Proc. of the AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[15] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. of the 15th International Conference on Machine Learning(ICML)*, pages 359–367, 1998.

[16] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *Proc. of IJCAI-99 Workshop on Machine Learning for Information Filtering*, pages 61–67, 1999.

[17] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[18] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Machine Learning Research*, 5:101–141, 2004.

[19] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34:1–47, 2002.

[20] D. Shen, J. Ruvini, M. Somaiya, and N. Sundaresan. Item categorization in the e-commerce domain. In *Proc. of The 20th ACM International Conference on Information and Knowledge Management (CIKM)*, 2011.

[21] D. Shen, J. D. Ruvini, R. Mukherjee, and N. Sundaresan. A study of smoothing algorithms for item categorization on e-commerce sites. In *Proc. of The Ninth International Conference on Machine Learning and Applications(ICMLA)*, 2010.

[22] C. N. Silla and A. A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22:31–72, 2011.

[23] A. S.Weigend, E. D.Wiener, and J. O. Pedersen. Exploiting hierarchy in text categorization. *Information Retrieval*, pages 193–216, 1999.

[24] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[25] V. N. Vapnik. *Statistical Learning Theory*, 1998.

[26] K. Weinberger and O. Chapelle. Large margin taxonomy embedding with an application to document categorization. *Advances in Neural Information Processing Systems*, pages 1737–1744, 2008.

[27] T. yan Liu, Y. Yang, H. Wan, H. jun Zeng, Z. Chen, and W. ying Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explorations*, 7:2005, 2005.

[28] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proc. of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.

[29] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. of the 14th International Conference on Machine Learning(ICML)*, pages 412–420, 1997.