Hierarchical Classification of Amazon Products

Bin Wang
Stanford University, bwang4@stanford.edu
Shaoming Feng
Stanford University, superfsm@ stanford.edu

***Abstract* - This projects proposed a hierarchical classification strategy for Amazon products. Classification strategy and the inference efficiency were the two focuses. A strategy to apply general classification algorithms to hierarchical classification problem was proposed. Parallel programing and hashing techniques were applied to improve inference efficiency.**

I. Introduction

Automatic hierarchical classification is drawing more attentions nowadays. With the explosion of information on the Internet, managing information by classifying them into hierarchical categories is becoming more and more important. Internet encyclopedias like Wikipedia, Medias like newspapers and online retailers like Amazon are the major demand sources.

This project aims to solve the hierarchical classification problem for Amazon products. In this problem, each product will be classified into a hierarchical category where both the destination category and the parental categories are clearly specified. There are two major challenges in this problem – efficiency and accuracy. Due the huge size of Amazon product catalog, there are enormous possible assignments for a product. Hence finding the most possible assignment will be a computational consuming as well as time consuming process. An efficient classification strategy must be developed so that each product can be classified fast enough to keep retailers and customers on Amazon from waiting for too long. Besides efficiency, accuracy is also an important consideration. If the products cannot be classified accurately, there will be troubles for both customers who want to search for a product and for retailers who want to maintain their catalog.

These two challenges in hierarchical classification are also the focuses of this project. After the problem formulization in section II, our methods will be presented in section III and be evaluated in section IV. Conclusions and future works are presented in section V.

II. Problem Formulization

A. Data Set

The dataset is a collection of Amazon products with highly detailed product information and their pre-assigned labels (categories).

The labels are organized hierarchically as shown in Figure 1. It is different from a simple tree structure since a node may have multiple parents. E.g. Node 1.2.1 in Figure 1 has node 1.2 and node 2 as its parents. Such structure is called Directed Acyclic Graph (DAG)[1]. As what the name implies, the relations between nodes are directed and there is no cycle in the graph.

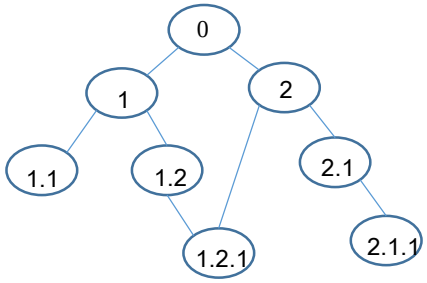

Figure 1. Category Hierarchy

There are19103 labels in total in the directed acyclic graph. Each label is a node in the graph and its sub-labels (sub-categories) are its children nodes. The graph has a single root. Most of the label nodes has a depth (distance from root node, with each edge a length of 1) of 4, where the maximum depth is 9. Also, each product may multiple labels which locate at different places in the graph.

The label information of a product is stored in the attribute "BrowseNodes". They are described as a "path from root node to its label". Since the product may have different labels, there may be multiple paths in this attribute.

Each product has 19 attributes where each attribute contains multiple properties and sub-properties with rich information for a certain aspect. In this project "Pruned Editorial Reviews" was the particular attribute that was used to build our base classifier. They are the concise product description provided by the vendor or some trusted parties.

B. Accuracy Improvement

Improving the accuracy for hierarchical classification in general is too broad a topic for a term project. Hence we narrowed down the scope to finding a good strategy for applying a given base classification algorithm.

Given a classification algorithm as our base classification algorithm, such a Naïve Bayes or logistic regression, there are many ways for them to be applied to a hierarchical classification problem. And for different problems, the best base algorithm may also vary. Hence, it is more important to find a good strategy for applying the base algorithm, which was set as the major goal for accuracy improvement in this project. Detailed strategies will be shown in next section.

C. Efficiency Improvement

There are around 20,000 categories and more than 330,000 products in the catalog. Efficiency will then be an important concern. To improve the inference efficiency, i.e. to minimize the time used for prediction, both programming techniques and algorithm improvement were explored.

D. Evaluation Metrics

Both accuracy and efficiency were evaluated in this project.

Accuracy per layer (depth) was the major criteria for different strategies. This is because this metric does not only capture how well a strategy is doing, but also reflects the advantage and disadvantage of a strategy in detail.

Inference time and training time were the major criteria for efficiency evaluation. Inference time refers to the time that an algorithm needs to make a prediction while training time stands for the time needed on training. Inference time is a more crucial requirement for applications since it affects user experience and system setup directly. A shorter training time will make development less expensive.

III. METHODOLOGY

A. Efficiency Improvement

i. Parallel Computing

The training and prediction of every nodes were applied in a breadth first search (BFS) order. It means for nodes at the same layer (depth), the data selection, training and prediction could be done in parallel. Due to the existence of global interrupt lock (GIL) in python, only a single core can actually be utilized in python's multi-thread module. Thus multi-process module must be applied to speed up the performance. However, writing speed to global data then became a problem for different processes, so the balance between computing and writing speed needs to be carefully handled. Till now, 8x speedup was achieved on a 4 core CPU with hyper-threading.

ii. Classifier and Data Hashing

After training a classifier, it will be saved to disk for later use. Since a label may appear multiple times during prediction, hashing the classifiers will boost up the speed.

In the original program, all data were saved as a list which was processed one by one in order to filter out those that belong to a certain label so they can be trained. However, during the prediction process, if the grandchild of a node needs to be predicted, there is no need to search for the data again since we already have all the needed information when predicting that node's child. Thus a data hash table is created to hold the data for later use in next layer in the BFS order.

B. Accuracy Improvement

i. Base Classifier

Multinomial Naïve Bayes classifier was chosen as the base classifier in this project. "Pruned editorial review" of each product was used as the input feature. The reviews were first encoded into ASCII characters and then stemmed. After stemming, some stopwords were removed to reduce the feature dimension. At last, term frequency–inverse document frequency (tf-idf) was applied to weight each word properly.

ii. Classification Strategy

Flat Strategy

Flat strategy is brutal style strategy used as a base line. In this strategy, the hierarchical structure is

ignored. In other words, all nodes in the graph are considered equally. For example, "book" and "art book" doesn't have any relationship anymore. So all the 19103 labels will be trained together in this strategy.

Top-Down Strategy:

In a top-down strategy, a path to a label from the root will be predicted for a given. Considering the directed graph of the labels, several algorithms can be applied on generating a label path.

- *Greedy Search:*

Greedy search is a special case of K-beam search which will be covered in the section below.

In greedy search, when predicting from any label node, the child with most probability will be chosen.

- *K-Beam Search:*

K-beam search is more complicated and has lots of variations. The general idea of K-beam search is that instead of choosing the child with most probability directly, it will choose K children with highest probability, then at the very end (leaves of the graph) the most possible one will be picked.

The variation of this algorithm locates where when counting the probability, multiple ways of calculation can be used: node probability, path probability and discounted probability. Figure 2 shows an example for future explanation.

Given a product, assuming it has 70% probability to be a "Book" and 30% probability to be "Kitchen" for classifier "Root". And for classifier "Book", it has a 40% chance to be "Art book", and 60% chance to be "Fiction book", similarly for the classifier "Kitchen".

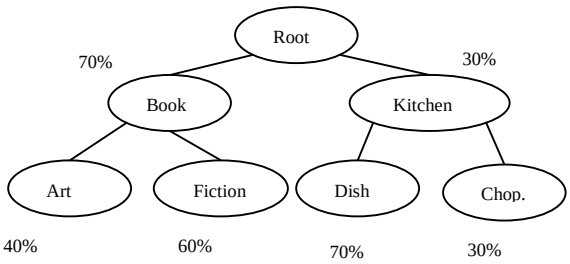

Figure 2 Classification Example

- *Node Probability*:

Node probability doesn't have any historical information when counting probabilities. In the above example with K=2, under classifier root, both "book" and "kitchen" will be kept in the result because they are the 2 node with highest possibilities. However, when predicting the next level of labels, the result would become "Dish" (70% probability) and "Fiction" (60% probability), which are higher than "Art" (40% probability) and "Chopsticks" (30% probability). In other words, information about their parent will be ignored.

- *Path Probability*:

For path probability, the multiplication of the probabilities of every node on the path would be used to choose the highest K probabilities. In K=2 case, "book" and "kitchen" would be kept for the first step without any change comparing to node probability. But for the next level of prediction, "Art" and "Fiction" will be kept as their probabilities are 0.7 * 0.4 = 0.28 and 0.7 * 0.6 = 0.42, which are higher than the probabilities of "Dish" (0.3 * 0.7 = 0.21) and "Chopsticks" (0.3 * 0.3 = 0.09).

- *Discounted Probability*:

Discounted probability is a variation of path probability with weight on the probability of different levels. In path probability in the example, "Art" is calculated as "0.7 * 0.4 = 0.28". Here a coefficient is added, so the probability becomes "0.7α * 0.4β", where the value of α and β needs to be adjusted at run time.

Combined Strategy

Since different strategies may achieve the best performance at different layer, it is better to combine strategies to maximize the overall performance.

First of all, different variation and parameters of K-beam search will be tested, to find the parameters (α, β and K) with highest accuracy for each layer. Then for a given product, if all the variations give a label in same layer, the result from the methods that has the highest test accuracy in that layer will be chosen. If different variations predict a label in different depth, only those results that are at the best performance of each method are considered. Among all the considered results, the one with highest testing accuracy with deeper layer will be chosen.

Let's take the following situation where 3-beam search performs best at depth 3, 2-beam search performs best at depth 4 and greedy search performs best on reset of the depths as an example. For a given product, if all methods predict the label to be in

depth 3, the result of 3-beam search would be chosen since it performs best at depth 3. If 3-beam search predicts the result to be a label at depth 2, 2-beam search predicts a label in depth 4 and greedy search predicts the label to be in depth 8. Only the result of 2-beam search and greedy search would be considered since only greedy search and 2-beam search gave predicts at the layer where their testing accuracy is better than other variations. And for this case, the result of greedy search would be chosen since it performs better at the deeper layer.

This methodology ensures that no degrading will happen in the balance of accuracy at different layers.

IV. Result and Evaluation

A. Feature Engineering

Table 1 shows the feature selection process for the base classifier in this project. Though improvement of base classifier is not our major concern. Some effort was spent to make sure it has a moderate performance.

Table 1 Feature selection for base classifier

Accuracy in %	33706 Data		101118 Data		303354 Data	
	Train	Test	Train	Test	Train	Test
Reviews in UTF-8 coding	45.6	43.2	51.1	49.4	57.7	56.9
Reviews in ASCII coding	45.4	42.9	50.9	49.1	57.6	56.6
Reviews with stemming	44.9	42.7	50.3	48.7	56.9	56
Reviews with stemming and stop word	48.9	46.1	53.6	51.8	60	58.9

B. Flat Strategy

Table 2 Flat Classification Performance

Accuracy in %	33706 Data		101118 Data		303354 Data	
	Train	Test	Train	Test	Train	Test
Mono-layer Prediction	2.98	2.8	3.01	2.25	3.01	2.8

As we can see from the result of flat strategy, the accuracy of both training and testing data are extremely low, which means flattening the labels and train them in a traditional way doesn't fit the needs of hierarchical classification problem at all.

C. Top-Down strategy

Table 3 shows the prediction accuracy for different top-down strategy with various parameter settings. After experimenting with different discounting parameters, no discount performs best in this case.

Table 3 Top-Down Strategy Performances

	Greedy	K(2)-beam	K(3)-beam	K(4)-beam	K(5)-beam	Combined Strategy
L1	55.32	53.92	51.09	49.9	49.1	55.32
L2	37.55	42.43	41.56	40.7	39.8	41.25
L3	23.38	25.31	24.22	22.7	21.5	23.31
L4	16.27	17.39	16.64	15.3	14.5	17.39
L5	12.37	11.63	10.8	9.71	9.05	12.37
L6	11.93	10.11	9.13	8.12	7.53	11.93
L7	11.79	9.74	8.68	7.65	7.01	11.79
L8	11.79	9.73	8.67	7.64	7	11.79
L9	11.79	9.73	8.67	7.64	7	11.79

As shown in the table, the greedy search (K(1)-beam) provides a fairly good accuracy comparing to K-beam search. However, through Layer 2 to Layer 4, K(2)-beam performs better than greedy search.

Hence the greedy search will be major parameter set. When combined strategy was applied, it achieve the best overall accuracy.

D. Efficiency Improvement

Table 4 Efficiency Improvement

	Single Product Prediction Time	Total Prediction Time	Training Time
Original	20~200 ms	3370.6 s	~1 day
Improved	1.12~3.12 ms	178.0 s	~2 hours

Here is a comparison between initial version of our K(3)-beam prototype and the final optimized version with hashes and parallel computing.

Total prediction time was improved by 18X and training time was improved by 12X, which is reasonable with 8 processes and hashing. Also, due to the limitation of multi-threading module in python interpreter as explained in above section, it is believed that a great leap in prediction time will be achieved with C or other language where multi-cores and good memory sharing efficiency are available.

V. Conclusion and Future Work

This project explored the strategies of applying base classifier on a hierarchical classification problem as well as methods to boost both inference and training speed. Combining greedy search and K-beam search with different parameters after experiments is the best strategy that was found. Parallel programing and hashing techniques were proposed as the solutions to improve efficiency.

But it is interesting that the accuracy of K-beam search does not increase with the increase of K. This implies the fact that the discounted path probability cannot truly reflect how good a path is. Hence looking for a better way of "scoring" different search strategies for each path will be the most important work for the future.

Acknowledgement: We want to say thank you to Prof. Ashutosh Saxena and Aditya Jami for providing the project materials and the weekly advice. Also we want to say thank you to Prof. Andrew Ng and CS229 TA's for their help during the quarter.

Reference

[1] Sun, Aixin, and Ee-Peng Lim. "Hierarchical text classification and evaluation."*Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, 2001.

[2] Babbar, Rohit, et al. "On Flat versus Hierarchical Classification in Large-Scale Taxonomies." *Advances in Neural Information Processing Systems*. 2013.