CS 145 Project Report

Team23

Jeffrey Tai, 504147859
Brian Chang, 304151550
Ken Ohhashi, 004197272
Colin Fong, 304202663

Introduction

Classification methods help to provide a framework or prediction for data that appears in everyday life. A very likely source of data that requires classification comes from consumers. Companies spend millions of dollars every year in order to better predict the behavior of a consumer or what products he would be more inclined to purchase. Being familiar with the audience helps these companies cater to their buyers to increase sales and have effective advertisement.

In this project, we take a look at one of the largest companies in the world, Walmart. Given a training dataset comprised with hundreds of thousands of products purchased, we want to create an algorithm that will help to define what type of trip a consumer is making, such as a small daily dinner trip, a weekly large grocery trip, or a seasonal trip to buy clothes. With our classification, we want to be able to define incoming test data in order to categorize future visits into certain trip types.

Each member's roles were distributed. Jeffrey wrote the code for calculating information gain and gain ratio, clustering departments and its second optimization, classifying clusters based on confidence, and this report. Brian wrote the first optimization for clustering departments and assisted in coming up with some of the methods. Ken came up with the idea of classifying by most frequent department. Colin helped to parse the output classification into the formatted CSV file for submission.

Method #1: Decision Tree

Our first idea was to create a decision tree that would have branches based on the different attributes. When choosing the root node as well as any other node in the tree, it is important to select the attribute that contributes the most information, thus yielding an outcome that comes closer to classifying the entry. Building a decision tree with the divide-and-conquer algorithm is a recursive process that consists of selecting a node or attribute, branching out possible outcomes, splitting those instances into subsets, and repeating for each subsequent subset. Recursion ends for that branch if all of its subsets can be attributed to a single class.

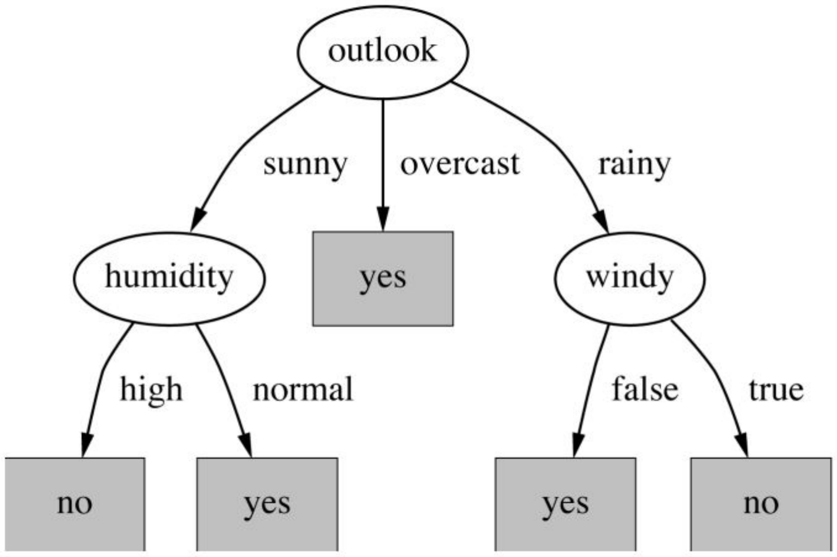

Figure 1: Decision Tree Example

There are several ways to select the node that splits the data: greatest information gain, largest gain ratio, or lowest Gini index. In our approach, we chose to calculate both information gain and gain ratio, seeing that gain ratio is more reliable than information gain.

Before we can calculate the information gain, we must understand entropy as the amount of un-orderedness in the distribution of a set. It can be calculated with the following equation, generalized to handle when there are more than 2 classes.

$$Entropy(Set) = -p_1 log_2 p_1 - p_2 log_2 p_2 - \ldots - p_n log_n p_n = -\sum_{i=1}^{n} p_i log_2 p_i$$

In this example, p^+ represents the proportion of items in the set that are in class +, while p^- denotes the proportion of items in the set of class -.

$$Information\ Gain\ (Set,\ Attribute) = Entropy(Set) - Information(Set,\ Attribute)$$
$$Information\ Gain\ (Set,\ Attribute) = Entropy(Set) - \sum_{i} \frac{|S_i|}{|S|} * E(S_i)$$

We calculate the average entropy of an attribute for each subset and compare it to the entropy of the entire set. Within our code, the "method1-informationgain.py" script computes the average entropy and entropy in order to get the information gain.

Visit Number	Weekday	UPC	Scan Count	Department Description	Fineline Number
4.036	0.0125	1.894	0.073	0.770	0.867

Table 1: Attribute Information Gain

Visit number is not used to identify the trip type with which it has an almost one-to-one relationship, thus yielding an extremely high information gain. The next 3 attributes with the highest gain are UPC, fineline number, and department description, respectively. It was easy to see that creating a decision tree on the UPC or fineline number was infeasible, given that there were 97,715 different UPC numbers and 5,196 different fineline numbers. A decision tree that split its attributes on either of them would have an overwhelming number of branches and subtrees. The attribute that remained was department description, which the training data only had 69 different values for. However, it was apparent that a decision tree may not be the best approach.

Method #2: Clustering Departments

Another method to classification is clustering the data such that when there is new incoming data, it would be paired with the cluster with which it has the least distance to. This distance can be measured in several ways, as the minimum distance between the closest point of each cluster, the maximum distance between furthest point of each cluster, the mean distance or the average of all points in each cluster, and an average distance which is a summation of each point in the first cluster to the each point in the second cluster.

Each collective visit in the training data would represent a cluster. The center "point" of each cluster was a 69-point coordinate, the values being the sum of the scan count for their respective department. The order of the departments represented are identical for all clusters for consistency. For example, if a cluster had the following scan counts: 3 in "PAINT AND ACCESSORIES", 2 in "DSD Grocery", and 4 in "FABRICS AND CRAFTS", and the order of departments was ("PAINT AND ACCESSORIES", "DSD GROCERY", "FABRICS AND CRAFTS", …), then the center point for this cluster would be (3, 2, 4, 0, 0, … 0).

The code presented in "method2-cluster.py" categorized the entire training data, and resulted in 95,674 clusters. The test.csv data was clustered as well, and our objective was to find the minimum distance between any 2 clusters. Euclidean distance was our standard for distance.

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

This method proved to be disastrously inefficient, requiring n^2 scan of the database. After running the script for about three minutes, it had finished clustering the testing and training data and found the minimum distance for only fifteen of the test data clusters. Therefore, the results were inconclusive and we were unable to test for accuracy.

Optimization 1:
A possible optimization that we had implemented in the same “method2-cluster.py” file was that for each test cluster, we would only look at relevant departments when calculating distance. Using the example mentioned above, if a cluster had the following scan counts: 3 in “PAINT AND ACCESSORIES”, 2 in “DSD Grocery”, and 4 in “FABRICS AND CRAFTS”, when comparing distances to the training clusters, we would only use the “PAINT AND ACCESSORIES”, “DSD Grocery”, and “FABRICS AND CRAFTS” departments (and scan counts).

Conceptually, this would better handle outliers and noisy data because only the similar departments in each cluster would be examined. Also, execution of this script was slightly more efficient due to distance being computed between a smaller amount of points, and in the same 3 minutes, we were able to identify the minimum distance between 40 of the test clusters.

However, in reality, this algorithm’s ignorance of non-relevant departments in the training cluster is incredibly erroneous. The existence of a product in another department of the training cluster is not truly an outlier; therefore, over-looking these existing products caused clusters that would otherwise seem to be radically different to be labeled as similar. This will be illustrated in the following example.

Training Data:

```
“TripType”,”VisitNumber”,”Weekday”,”Upc”,”ScanCount”,”DepartmentDescription”,”FinelineNumber”
30,7,”Friday”,60538815980,3,”SHOES”,1000
30,7,”Friday”,7410811099,2,”PAINT AND ACCESSORIES”,1017
8,9,"Friday",1070080727,1,"SHOES",115
8,9,"Friday",3107,1,"PAINT AND ACCESSORIES",103
8,9,"Friday",7603138508,1,"BOYS WEAR",654
8,9,"Friday",5460010568,1,"HOUSEHOLD CHEMICALS/SUPP",52
8,9,"Friday",2899521885,1,"FABRICS AND CRAFTS",1056
```

Test Data:

```
”VisitNumber”,”Weekday”,”Upc”,”ScanCount”,”DepartmentDescription”,”FinelineNumber”
1,"Friday",72503389714,1,"SHOES",3002
1,"Friday",72450403707,1,"PAINT AND ACCESSORIES",1018
```

Clustering by only relevant departments would cause the test data to be represented as (1, 1) because (ScanCount for Shoes, ScanCount for Paint and Accessories). The 2 clusters in the training data would be represented as (3, 2) for TripType 30 and (1, 1) for TripType 8. This is clearly inaccurate because the existence of many other departments are ignored, in which TripType 30 would most likely be correlated with the test cluster.

Unfortunately, the time it would take to run this program was excessively long and we were produce to have test results.

Optimization 2:
Understanding that optimization 1 would be even more inaccurate than the previous solution of computing the distance between all 69 departments whether or not they appeared in the test data cluster, we adjusted our approach.

We want to characterize each trip type in the training data by the most frequent department and the average number of different products per visit, not including the scan count. The first step was to read the training data and find the most frequent department that appeared for each trip type. While reading the file, we identified a collective visit and found the number of products for that visit with respect to the trip type. Once the end of the file was reached, an average was computed and assigned as a key value pair to the trip type. The purpose was to help break the tie between clusters when they had the same most frequent department. The code can be found in “method2-clusteroptimization.py”.

The test data had to also be treated as a cluster, each cluster being a collective visit with the same visit number. The most frequent department was computed and if there were multiple training clusters with the same department, the count of products was used to deduce the trip type. The output of this algorithm is in “results-old.csv”, and has been uploaded to Kaggle under Team23. The score is 24.94, while the benchmark is 34.54. This shows a small amount of classification but it is mostly inaccurate.

This flaw in this algorithm is quite apparent. From the training data, each trip type is attributed to only one “most frequent department”. There are a total of 38 trip types, including type 999, and 69 departments. If a department never appears as the most frequent for a specific trip type in the training data, but is the most frequent for the specific visit in the test data, our algorithm automatically categorizes it as type 999. This is a huge flaw because not every department can be represented from the training data.

Method #3: Confidence

The last method that we attempted was to get a general range of the support for departments for each trip type. By observation, we noticed trip types generally had a limited range on the number of different products that were purchased per trip. Aside from a few exclusions, we categorized these into trip types into three groups: ones that had under 10 products, under 20 products, and all others. We also found that most trip types had one or a very limited number of departments that were constantly appearing.

Using this, we defined the number of different products as the support for a collective visit, and confidence as the proportion of the visit that contained a specified constraint of department(s). We compiled a list of trip types and their respective frequent department along with the minimum confidence that was necessary in order to classify it (method3-triptype-confidence.xlsx). There were a few trip types in which we could not deduce a specific department to observe or seemed to have completely random or miscellaneous department frequencies. As a result, we placed those at the end and used a random number generator to classify a trip that we could not do so otherwise.

The code can be found in “method3-confidence.py” which reads directly from the test csv file and upon identifying an entire cluster, it attempts to classify it. The number of products is computed and through a series of if-else statements, the confidence of an array of departments is computed. If the minimum confidence is met, then the current visit is assigned to a designated trip type. This solution scored a 24.865 on Kaggle, which is slightly better than the second method, but it is evident that there is still a considerable margin of error in classification.

This algorithm is very susceptible to having outliers slip through the classification. The first “type” of classification is the size of the visit, and if generally a trip type contains five products per visit, but one of the the visits contain fifteen products, it would not be properly classified. Also, a random number generator is used but not in the sense that a completely random number will be output; rather, just one trip type is randomly chosen from a couple that are possible or very likely.

Conclusion

From this project, we learned that classification becomes much more difficult when there are multiple classes as opposed to a simple “Yes” or “No”. It is relatively simple to find which attribute offers the greatest information gain; the difficulty lies in choosing how to use these attributes to maximize the likeliness of our prediction. Not every attribute offers the same gain, but even those that offer considerable information may not be the best to use for categorization.

Furthermore, lower-level data mining algorithms such as decision trees and classification using confidence do not provide the greatest accuracy. There are other advanced algorithms such as random forests that would increase the accuracy tremendously. However, the methods that we have used in this report cover the majority of this course’s scope from clustering to classification to decision trees.