

Scanned
from
best available copy

IQP/MQP SCANNING PROJECT



George C. Gordon Library
WORCESTER POLYTECHNIC INSTITUTE

00D029M

Project number: MLC-SF00 -CS

Filtering Greeting Cards @ Sparks.com

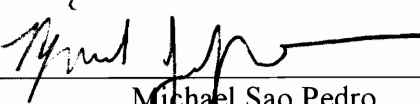
A Major Qualifying Project Report
submitted to the Faculty
of

WORCESTER POLYTECHNIC INSTITUTE

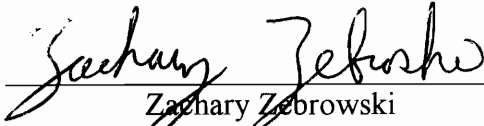
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

by

Sharad Bhojnagarwala




Michael Sao Pedro



Zachary Zebrowski

Date: March 11, 2000

Approved:



Professor Mark Claypool, Major Advisor

ABSTRACT

This project determined the possible benefits of using an online personalized filtering system for paper greeting cards at Sparks.com. A test recommender system was developed in Java that implemented content-based and collaborative filtering algorithms. Additionally, a combination of these filtering algorithms was created to see if this approach gave more accurate results. Using information extracted from their purchase history database, the system was tested against other methods for displaying cards that do not utilize personalized filtering. We found that using content-based, collaborative or top 10 approach will result in highly accurate predictions compared to the random ordering of cards currently done at Sparks.com.

Acknowledgements & Notes

Our project group would like to thank their advisor Professor Mark Claypool and the other on-site Silicon Valley project advisors Professor David Finkel and Professor James Duckworth for their support, assistance, and guidance throughout the project.

We would also like to thank Amit Mediratta, Jim Karavias, and the rest of the staff at Sparks.com for giving us the assistance and materials to perform this project.

All work on this Major Qualifying Project was performed equally by all members of the project group. Individually, Sharad Bhojnagarwala developed all the algorithms necessary to extract information from their database. Michael Sao Pedro designed and implemented the filtering algorithms and the core filtering engine. Zachary Zebrowski developed the algorithms for creating the “bag of words”, and performed the testing of the recommender system.

Table of Contents

Table of Contents	4
List of Tables	6
List of Equations	17
Executive Summary	18
1. Introduction	21
2. Related Work	25
2.0 Introduction to Related Work	25
2.1 Recommender Systems	25
2.2 Collaborative Filtering	29
2.3 Content-based filtering	32
2.4 Combination of Collaborative and Content-based filtering	33
2.5 Quality of a Recommender System	34
2.6 Filtering @ Sparks.com	36
3. Methodology and Implementation	39
3.1 Extracting the WPI Database	40
3.1.1 Details of the WPI Database	43
3.1.2 Tables in the WPI Database	46
3.2 Recommender System Overview	56
3.3 Filtering Algorithms	59
3.3.1 Bag of Words	60
3.3.2 Constructing User Profiles	62
3.3.3 Content-Based Filtering Design	64
3.3.4 Collaborative-Based Filtering Design	66
3.3.5 Combining Content-based and Collaborative Filtering Predictions	69
3.3.5.1 Combination Approach 1	70
3.3.5.2 Combination Approach 2	71
3.4 Recommender System Design	71
3.5 Testing Procedures	80
4. Results	84
4.1 Initial Tests	84
4.1.1 Number of cards bought by users	85
4.1.2 Distinct vs. Duplicate Orders	87
4.1.3 Distribution of Cards with Respect to Price	89
4.1.4 Distribution of Different Card Attributes	92
4.1.5 Distribution of Relevance Attributes For Christmas Cards	93
4.1.6 Distribution of Relevance Attributes of Birthday Cards	96
4.1.7 Bag of Words Attribute	97
4.2 Phase 1 of Testing: Content attributes	106
4.3 Phase 2 of Testing: Content-based filtering Algorithm	140
4.4 Phase 3 of Testing: Filtering Algorithm Accuracy	153
4.5 Phase 4 of Testing: Personalized Filtering vs. Non-personalized Filtering	171
5. Conclusions	179
5.1 Future Work	180
Appendix A. Company Information	185

Appendix B. SQL Queries	187
Appendix C. Filtering System Pseudocode	225
Bibliography.....	243

List of Figures

<u>Figure</u>	<u>Title</u>	<u>Section</u>	<u>Page</u>
Figure 3.1	WPI Database ER diagram	3.1.2	55
Figure 3.2	Overview of Recommender System	3.2	57
Figure 3.3	Class Modeling	3.4	72
Figure 3.4	UML design for the back end	3.4	73
Figure 3.5	UML design for the main filtering system	3.4	74
Figure 3.6	Step-by-step workings of the filtering system; a class action diagram	3.4	75
Figure 3.7	Class-responsibility-collaboration (CRC) diagrams	3.4	76
Figure 3.8	UML description for the bag of words	3.4	78
Figure 3.9	CRC diagrams for the bag of words	3.4	79
Figure 4.1.1	Number of duplicate Christmas and birthday cards card bought analysis.	4.1.1	86
Figure 4.1.2	Number of users for each bracket that bought duplicate vs. distinct cards.	4.1.2	88
Figure 4.1.3	Distribution of cards with respect to price.	4.1.3	91
Figure 4.1.4	Distribution of different card attributes.	4.1.4	93
Figure 4.1.5	Distribution of relevance attributes.	4.1.5	95
Figure 4.1.6	Distribution of relevance attributes for birthday cards.	4.1.6	97
Figure 4.1.7	Size of bag of words for distinct purchased Christmas cards vs. number of distinct orders	4.1.7	104
Figure 4.1.8	Size of bag of words for distinct purchased birthday cards vs. number of distinct orders.	4.1.7	104

Figure 4.1.9	Average number of bag of word matches vs. number of Christmas card orders.	4.1.7	105
Figure 4.1.10	Average number of bag of word matches vs. number of birthday card orders.	4.1.7	105
Figure 4.2.2	Scatter plot of average placement for bag of words for Christmas cards.	4.2	107
Figure 4.2.3	Box plot of average placement for bag of words for Christmas cards.	4.2	108
Figure 4.2.4	Box plot for Christmas cards' bag of words size distribution	4.2	109
Figure 4.2.5	Box plot for total number of brackets for Christmas cards.	4.2	109
Figure 4.2.6	Christmas cards price attribute scatter plot.	4.2	111
Figure 4.2.7	Christmas cards price attribute box plot.	4.2	112
Figure 4.2.8	Christmas cards grouped sentiments scatter plot.	4.2	113
Figure 4.2.9	Christmas cards grouped sentiments box plot	4.2	114
Figure 4.2.10	Christmas cards grouped sentiments box plot for bracket size distribution.	4.2	114
Figure 4.2.11	Christmas cards sentiments box plot for total number of brackets.	4.2	115
Figure 4.2.12	Christmas card designer ID scatter plot	4.2	116
Figure 4.2.13	Christmas card designerID box plot.	4.2	117
Figure 4.2.14	Christmas card designer ID distribution for size of bracket.	4.2	117
Figure 4.2.15	Christmas card grouped discrete characteristics scatter plot.	4.2	118
Figure 4.2.16	Christmas cards grouped discrete characteristics box plot.	4.2	119
Figure 4.2.17	Christmas cards grouped distinct characteristics	4.2	119

distribution for size of bracket.

Figure 4.2.18. Christmas cards grouped distinct characteristics total number of brackets.	4.2	120
Figure 4.2.19. Christmas cards collaborative scatter plot.	4.2	121
Figure 4.2.20. Christmas cards collaborative box plot.	4.2	122
Figure 4.2.21. Christmas cards collaborative distribution for size of bracket.	4.2	122
Figure 4.2.22. Christmas cards collaborative total number of brackets.	4.2	123
Figure 4.2.23. Birthday bag of words scatter plot.	4.2	124
Figure 4.2.24. Birthday bag of words box plot.	4.2	125
Figure 4.2.25. Birthday bag of words distribution for bracket size	4.2	125
Figure 4.2.26. Birthday bag of words total number of brackets	4.2	126
Figure 4.2.27. Birthday price attribute scatter plot.	4.2	127
Figure 4.2.28. Birthday price attribute box plot.	4.2	128
Figure 4.2.29. Birthday sentiment scatter plot.	4.2	130
Figure 4.2.30. Birthday sentiment box plot	4.2	130
Figure 4.2.31. Birthday sentiment distribution for bracket size.	4.2	131
Figure 4.2.32. Birthday sentiment total number of brackets.	4.2	131
Figure 4.2.33. Birthday card designer ID (attribute 5) scatter plot.	4.2	132
Figure 4.2.34. Designer ID box plot for birthday.	4.2	133
Figure 4.2.35. Designer ID distribution for birthday for bracket sizes	4.2	133
Figure 4.2.36. Birthday grouped distinct characteristics scatter plot.	4.2	134
Figure 4.2.37. Birthday distinct characteristics grouped box plot	4.2	135

Figure 4.2.38.	Birthday grouped distinct characteristics distribution for bracket size.	4.2	135
Figure 4.2.39	Birthday grouped distinct characteristics total number of brackets	4.2	136
Figure 4.2.41	Birthday collaborative scatter plot.	4.2	137
Figure 4.2.42	Birthday collaborative box plot.	4.2	138
Figure 4.2.43	Birthday collaborative distribution for bracket size	4.2	138
Figure 4.2.44	Collaborative total number of brackets.	4.2	139
Figure 4.3.1	Scatterplot for christmas content-based score with weights set to 1.	4.3	140
Figure 4.3.2	Box plot for Christmas content-based score with weights set to 1.	4.3	142
Figure 4.3.3	Distribution for Christmas content-based score with weights set to 1 for bracket size	4.3	142
Figure 4.3.4	Box plot for Christmas content-based weights at 1 for number of brackets.	4.3	143
Figure 4.3.5	Scatterplot for Christmas content-based score with optimized weights.	4.3	144
Figure 4.3.6	Box plot for Christmas content-based score with optimized weights.	4.3	145
Figure 4.3.7	Distribution for Christmas content-based score with optimized weights	4.3	145
Figure 4.3.8	Box plot for Christmas content-based score with optimized weights for number of brackets.	4.3	146
Figure 4.3.9	Scatter plot for birthday with content-based weights set at 1.	4.3	147
Figure 4.3.10	Box plot for birthday content-baseds with weights set with equal importnace.	4.3	148
Figure 4.3.11	Distribution for birthday with content-based weights set with equal importance.	4.3	148

Figure 4.3.12	Number of brackets box plot for birthday with content-based weights set with equal importance.	4.3	149
Figure 4.3.14	Scatterplot for birthday content-based score with optimized weights	4.3	150
Figure 4.3.15	Box plot for birthday content-based score with optimized weights.	4.3	151
Figure 4.3.16	Distribution for birthday content-based score with optimized weights for bracket size.	4.3	151
Figure 4.3.17	Box plot for birthday content-based score with optimized weights for number of brackets.	4.3	152
Figure 4.4.1	Total Christmas test algorithm 1 with weights with equal importance.	4.4	153
Figure 4.4.2	Total Christmas test algorithm 1 with weights with equal importance	4.4	154
Figure 4.4.3	Total Christmas test algorithm 1 distribution with weights set with equal importance for bracket size.	4.4	154
Figure 4.4.4	Total Christmas test algorithm 1 with weights with equal importance for total number of brackets.	4.4	155
Figure 4.4.5	Scatterplot for total Christmas test algorithm 1 with optimal weights Set.	4.4	156
Figure 4.4.6	Box plot for total Christmas test algorithm 1 with optimal weights set.	4.4	157
Figure 4.4.7	Total Christmas test algorithm 1 distribution with optimal weights set for bracket size.	4.4	157
Figure 4.4.8	Box plot for total Christmas test algorithm 1 with optimal weights set for total number of brackets.	4.4	158
Figure 4.4.9	Total birthday test algorithm 1 with weights set with equal importance.	4.4	159
Figure 4.4.10	Total birthday test algorithm 1 with weights set with equal importance	4.4	160

Figure 4.4.11	Total birthday test algorithm 1 distribution with weights set with equal importance for bracket size.	4.4	160
Figure 4.4.12	Box plot for total birthday test algorithm 1 with set with equal importance for total number of brackets.	4.4	161
Figure 4.4.13	Total Birthday Test Algorithm 1 with Weights Set at Optimized Weights	4.4	162
Figure 4.4.14	Box plot for total birthday test algorithm 1 with optimal weights set.	4.4	163
Figure 4.4.15	Total Birthday Test Algorithm 1 distribution with optimal weights set for bracket size.	4.4	163
Figure 4.4.16	Box plot total birthday test algorithm 1 with optimal weights set for total number of brackets.	4.4	164
Figure 4.4.17	Box plot for total Christmas test algorithm 2 with optimal weights set.	4.4	165
Figure 4.4.18	Box plot for total Christmas test algorithm 2 with optimal weights set.	4.4	166
Figure 4.4.19	Total Christmas test algorithm 2 distribution with optimal weights set for size of bracket	4.4	166
Figure 4.4.20	Total Christmas test algorithm 2 distribution with optimal weights set for distinct brackets	4.4	167
Figure 4.4.21	Total birthday test algorithm 2 with optimal weights set.	4.4	168
Figure 4.4.22	Box plot for total birthday test algorithm 2 with optimal weights set.	4.4	169
Figure 4.4.23	Total birthday test algorithm 2 distribution with optimal weights set for bracke size.	4.4	169
Figure 4.4.24	Total birthday test algorithm 2 distribution with optimal weights set for distinct brackets.	4.4	170

Figure 4.5.1	Box plot for content-based for Christmas cards vs. random ordering with "extreme" outliers removed.	4.5	172
Figure 4.5.2	Box plot for content-based for Christmas cards vs. "top 10" ordering without "extreme" outliers.	4.5	173
Figure 4.5.3	Box plot for collaborative for Christmas cards vs. random ordering without "extreme" outliers.	4.5	175

List of Tables

<u>Table</u>	<u>Title</u>	<u>Section</u>	<u>Page</u>
Table 3.1	Statistical analysis on raw data	3.1.1	44
Table 3.2	Example data for bag of words calculation	3.3.1	61
Table 3.3	Bag of words rating based on user's purchase of card 1	3.3.1	61
Table 3.4	Example data for profile and content calculations	3.3.2	63
Table 3.5	Example weight data for content calculations	3.3.3	66
Table 3.6	Content predictions using user's profile	3.3.3	66
Table 3.7	Example user-item matrix	3.3.4	68
Table 3.8	Example user-user matrix derived from Table 3.7	3.3.4	69
Table 3.9	Example collaborative predictions for active user	3.3.4	69
Table 4.1.1	Number of duplicate Christmas and birthday cards.	4.1.1	85
Table 4.1.2	Number of users for each bracket that bought duplicate vs. distinct cards	4.1.2	87
Table 4.1.3	Distribution of cards with respect to price.	4.1.3	90
Table 4.1.4	Distribution of different card attributes.	4.1.4	92
Table 4.1.5	Distribution of relevance attributes for Christmas cards	4.1.5	94
Table 4.1.6	Distribution of relevance attributes for birthday cards.	4.1.6	96
Table 4.1.7	Time statistics for bag of words	4.1.7	99

Table 4.1.8	Average bag size for each user based on the number of Christmas cards purchased	4.1.7	99
Table 4.1.9	Word profile size statistics for each specific number of distinct purchased Christmas cards.	4.1.7	100
Table 4.1.10	Word profile size statistics for each specific number of distinct purchased birthday cards.	4.1.7	101
Table 4.1.11	Bag of words average number of words matched for each number of distinct cards purchased.	4.1.7	102
Table 4.1.12	Simple linear regression results for size of bag of words for Christmas cards.	4.1.7	102
Table 4.1.13	Simple linear regression for bag of words average number of word matched for Christmas cards.	4.1.7	103
Table 4.1.14	Simple linear regression results for size of bag of words for birthday cards.	4.1.7	103
Table 4.1.15	Simple linear regression for bag of words average number of word matched for birthday cards.	4.1.7	103
Table 4.2.1	Statistical summary for bag of words for Christmas cards.	4.2	107
Table 4.2.2	Christmas cards price attribute summary statistics.	4.2	111
Table 4.2.3	Christmas cards grouped sentiments attributes summary statistics.	4.2	113
Table 4.2.4	Christmas card designer ID summary statistics (attribute 5).	4.2	116
Table 4.2.5	Christmas cards grouped discrete characteristics summary statistics.	4.2	118
Table 4.2.6	Christmas cards collaborative summary statistics.	4.2	121
Table 4.2.7	Birthday bag of words summary statistics.	4.2	124
Table 4.2.8	Birthday price attribute summary statistics.	4.2	127
Table 4.2.9	Birthday sentiment summary statistics.	4.2	129

Table 4.2.10	Birthday cards designer ID (attribute 5) for birthday cards	4.2	132
Table 4.2.11	Birthday grouped attributes summary statistics.	4.2	134
Table 4.2.12	Birthday collaborative summary statistics.	4.2	137
Table 4.3.1	Christmas content-based score with weights set to 1.	4.3	140
Table 4.3.2	Christmas content-based score with optimized weights.	4.3	144
Table 4.3.3	Birthday content-based score with weights set to 1.	4.3	147
Table 4.3.4	Birthday content-based score with optimized weights.	4.3	150
Table 4.4.1	Total Christmas test algorithm 1 with weights set with equal importance.	4.4	153
Table 4.4.2	Total Christmas test algorithm 1 with optimal weights set.	4.4	156
Table 4.4.3	Total Birthday test algorithm 1 with weights set with equal importance.	4.4	159
Table 4.4.4	Total birthday test algorithm 1 with optimized weights set.	4.4	162
Table 4.4.5	Total Christmas test algorithm 2 with weights set at optimized weights	4.4	165
Table 4.4.6	Total birthday test algorithm 2 with optimal weights set.	4.4	168
Table 4.5.1	Content-based for Christmas cards vs. random ordering with "extreme" outliers included.	4.5	171
Table 4.5.2	Content-based for Christmas cards vs. random ordering with "extreme" outliers not included	4.5	171
Table 4.5.3	Content-based for Christmas cards vs. "top 10" ordering with "extreme" outliers.	4.5	173

Table 4.5.4	Content-based for Christmas cards vs. "top 10" ordering without "extreme" outliers.	4.5	173
Table 4.5.5	Collaborative for Christmas cards vs. random ordering with "extreme" outliers.	4.5	174
Table 4.5.6	Collaborative for Christmas cards vs. random ordering without "extreme" outliers.	4.5	174
Table 4.5.7	Christmas final statistics.	4.5	176
Table 4.5.8	Mean comparison.	4.5	177
Table 4.5.9	Median comparison.	4.5	177
Table 4.5.10	Birthday final statistics.	4.5	178
Table 4.5.11	Mean comparison for birthday.	4.5	178
Table 4.5.12	Median comparison for birthday.	4.5	178

List of Equations

<u>Equation</u>	<u>Title</u>	<u>Section</u>	<u>Page</u>
Equation 2.1	A content-based filtering calculation	2.3	32
Equation 3.1	Scalable attribute percent closeness formula	3.3.3	65
Equation 3.2	Combination prediction equation	3.3.5.1	70

Executive Summary

Information on the Internet keeps growing at an alarming rate. From online articles, to the types of clothes that can be bought online, information generated and presented keeps expanding day by day. One of the biggest problems associated with this “information overflow” is sifting out exactly what information is wanted. In order to help users find the information they desire, many sites use online filters.

Our project sought to determine if filtering technology could be successfully implemented to suggest paper greeting cards that users might like to buy at Sparks.com, an Internet startup company located in San Francisco, California. Filtering is the technique of comparing an incoming information's stream to the profile of the user's interest and recommending documents to the user based on that profile (1 Soboroff). By performing this comparison, information relevant to the user's interests can be sifted from a data stream, instead of displaying all the data. Currently, two well-known techniques for filtering, called collaborative and content-based filtering, have been studied and implemented in different systems. Collaborative filtering uses similarities and dissimilarities between users to determine relevant information while content-based filtering involves a direct comparison between the content or attributes of a user's profile and the document. In particular, our system uses collaborative filtering by making recommendations for a user based upon the opinions of other users who have purchased similar cards as that user. Content-based filtering is implemented by comparing the content, or attributes, of each card with respect to a user's profile generated from the content of the cards they purchased.

Because Sparks.com has such a vast inventory, users may not be able to readily find the “perfect card” they want. Therefore, the implementation of a recommender system, a system that

calculates predictions based on the users tastes regarding how well they may like something, in this case a card, by implementing filtering may help. Using a recommender system, Sparks.com customers may be able to find cards and other items of their choice even faster. Sparks.com may also benefit from this from a competitive standpoint because today none of the other online paper greeting card sites use filtering algorithms or other adaptive techniques for recommendations.

Many steps were taken to determine if and how filtering could best be utilized at Sparks.com. We first determined what relevant information could be used to filter their paper greeting cards. Next, content-based and collaborative filtering algorithms were developed so that each could be tested to see which would work best for filtering cards. In addition, two algorithms that combined collaborative and content-based predictions were developed to see if this combination approach would provide more accurate results. A test filtering system was then developed in Java, which implemented these filtering calculations. Using the results computed from our recommender system, statistical analyses were performed to determine the accuracy of the filtering predictions. Adjustments to the algorithms were then made to see if this accuracy could be improved. These experiments determined which algorithms and methods, if any, would work best for the domain of greeting cards. Based on this information, we finally decided if filtering would work and what improvements would increase the filtering accuracy.

The recommender system was tested on a subset of the Sparks.com inventory, in particular Christmas cards and birthday cards. These two domains of cards were used, because Christmas cards are more likely to reflect the buyer's tastes while birthday cards are more likely to reflect the tastes of the recipient of the card. This personalized filtering system was compared to non-personalized filtering systems, in particular a ranking of cards most purchased, and a random ordering. For Christmas cards, bag-of-words weighed most heavily in making accurate

predictions followed by sentiments, price, and the distinct attributes. For Birthday cards, bag-of-words seemed to be the most important factor in making predictions. The other content attributes did not significantly improve the accuracy of predictions. Collaborative filtering worked very well with Christmas cards, but was a poor indicator for birthday cards. Combining content-based and collaborative filtering algorithms resulted in better predictions than both of the approaches alone, but did not help the birthday cards. For both domains of cards, filtering provided a better alternative than a “top-ten” list or a random ordering. Therefore, the overall results suggested that filtering could be successfully applied to the domain of cards.

This recommender system can be implemented into Sparks.com’s main system in many ways. A “*Sparks Advisor*” can suggest other cards similar to a card chosen during an online session if the filtering calculations are performed while the user is online. A recommender system front end called “*My sparks*” can give the customer the recommended cards produced by the recommender system. Finally, a feedback system which shows other people’s ratings or comments about a particular card could not only provide information to the user, but can help in making more accurate personalized recommendations.

As the number of cards grows, Sparks.com customers may have more difficulty finding the cards they might like to purchase. Using a recommender system, each user can receive personalized lists that show users cards that match their tastes. Our research discovered that Sparks.com could benefit from recommender systems by using both collaborative and content-based filtering in the main engine. By implementing a recommender system users can more accurately find the cards they want, and in the future, receive suggestions for accompanying gifts. Sparks.com may offer using each user’s card tastes as a guideline.

1. Introduction

The explosive growth of Web sites, Internet newsgroups and mailing lists has led to the problem of information overloading. These parts of the Internet are constantly growing and thus continue to acquire new information at an amazing rate. One of the biggest problems with this information explosion is that users cannot find the information they want quickly. People are therefore in need of help to filter this mass information.

Some of today's most successful online businesses make use of filtering as an improvement to the old days of hunting around a department store to find a desired item, and moreover, to present other items that may be appealing. These online businesses use powerful computerized filtering technology to do what human employees are unable to do: find out the individual customer's needs, preferences and wants, predict what that individual customer would like, and dynamically personalize their offerings based on that knowledge. Such systems, known as *recommender systems*, enhance profitability by increasing sales and strengthen customer relationships with each transaction. IVillage (<http://www.ivillage.com>), CDnow (<http://www.cdnow.com>), E!Online (<http://www.eonline.com>), Ticketmaster Online (<http://www.ticketmaster.com>), and many others use real time recommendation systems to achieve these goals.

Filtering is the technique of comparing an incoming information's stream to the profile of the user's interest and recommending documents to the user based on that profile (1 Soboroff). By performing this comparison, information relevant to the user's interests can be sifted from a data stream, instead of displaying all the data. Currently, two well-known techniques for filtering have been studied and implemented in different systems. These filtering types are collaborative

filtering and content-based filtering. Collaborative filtering uses similarities and dissimilarities between users to determine relevant information while content-based filtering involves a direct comparison between the content or attributes of a user's profile and the document.

In order to perform filtering, two pieces of information are required: the user's preferences and, items to compare with the user's preferences. Gathering information about the user's likes and dislikes for filtering can be performed in two different ways, either explicitly or implicitly. Explicit ratings are preference judgments in which the user directly says what they like or dislike. For example, users can participate in surveys in which they rate different items to acquire these preferences. On the other hand, implicit ratings are derived from data sources such as purchase records, web logs, time spent reading a document, or URL references in Usenet postings. For our project, user preferences are implicitly derived, because Sparks.com had no system in which users rated items. Their user purchase history was used to attain these implicit predictions, by assuming that users like the cards they purchased. Once this information has been gathered, a user profile, an entity that defines a user's likes, dislikes, or both, can be created. Filtering can then finally occur by showing only information that matches closely with their profile.

Our project aims to determine if filtering technology can be successfully implemented to suggest paper greeting cards that users would like to buy at Sparks.com, an Internet startup company located in San Francisco, California. With the strength of the Internet, visitors to Sparks.com can access thousands of real greeting cards from hundreds of publishers. With such a vast inventory, users can potentially find the perfect card for almost all occasions. Presently, Sparks.com allows customers different ways of selecting cards from keyword specific to browsing by subject and recipient. Customers can create an account to keep an address book,

personal reminders as well as a variety of other services that help people stay in touch. In addition to greeting cards, Sparks.com sells personal invitation cards, gift certificates, chocolates and even stamps. Recommender systems may help Sparks.com customers find more cards and other items of their choice more quickly.

Very few of the online greeting card sites sell paper greeting cards. Furthermore, to the best of our knowledge, none of them use filtering algorithms or other adaptive techniques for recommendations. Therefore, recommender systems for paper greeting cards have not been experimented with before. Cardsale.com (<http://www.cardsale.com>) and Celebrate-It.com (<http://www.celebrate-it.com>) allow users to select a category and then present all the cards belonging to the selection. Fatming.com (<http://www.fatming.com/>) presents a category search to help users further refine their selections. None of these sites maintain any kind of user profiles or history and none provide recommendations. Amazingmail.com (<http://www.amazingmail.com/>) allows users to create an account and then make postcards and buy them. However, the use of a user's account is solely to speed up credit-card transaction processing. The web site does not tailor its recommendations based upon the user's history or tastes.

A number of online greeting cards stores are being introduced as e-commerce continues to grow. The web site that provides quick service to customers will survive. Customers will stick to the web site where they can find products of their liking quickly and easily. Sparks.com can achieve these abilities using recommender systems. A recommender system learns from customers and recommends products that they would find the most valuable from available products. These systems implement some form of filtering as the backbone for learning about customers and the products they like to make predictions. Recommender systems help to transform browsing visitors into buyers, and to increase the average order size by recommending

complementing products to buyers. There are different techniques for implementing recommender systems that are based on the amount of persistence, automation and revenues desired by the implementing E-commerce site. Because there are many diverse approaches, different recommender systems were observed so that appropriate development choices for the recommender system at Sparks.com could be made.

Many steps are taken to determine if and how filtering could best be utilized at Sparks.com. We first determine what relevant information can be used to filter their paper greeting cards. Next, different filtering algorithms are developed so that each can be tested to see which work best for filtering cards. A test filtering system is then developed in Java, which implements these filtering calculations. Using the results computed from our recommender system, statistical analyses are performed to determine the accuracy of the filtering predictions. Adjustments to the algorithms are then made to see if this accuracy could be improved. These experiments determine which algorithms and methods, if any, would work best for the domain of greeting cards. Based on this information, we finally decide if filtering would work and what improvements may increase the filtering accuracy.

2. Related Work

2.0 Introduction to Related Work

This section serves as an introduction to information relevant to understanding the terminology and studies regarding content-based and collaborative-based filtering. First, recommender systems will be discussed, then each of the filtering techniques, content-based and collaborative, will be explored. Next, the notion of quality in a recommender system will be discussed. Finally, information on how and why Sparks.com can benefit from filtering will be discussed.

2.1 Recommender Systems

Recommender systems can give recommendations based on four different prediction schemes. These four schemes are outlined as follows:

- Non-personalized: A *non-personalized* recommender system suggests products to a customer based on what other customers felt about those products on average. These recommendations are independent of the customer, therefore each customer gets the same recommendations.
- Attribute-based: An *attribute-based* recommender system shows products to a customer based on qualitative or quantitative descriptions of the products. These systems require users to explicitly state which products or characteristics of products

interest them. These systems are similar to search engines in which the user types in specific “qualities” to match.

- Item-to-item correlations: Recommendations can also be made based on *item-to-item correlations*, in which suggestive selling occurs based on the customers’ expressed interest in a small set of products. Instead of users explicitly stating what they want, the system makes suggestions based upon qualitative and quantitative similarities between items for which the user has expressed interest and other items the user has not purchased.
- People-to-people correlations: Recommendations can be made using a *people-to-people correlation*, also referred to as *collaborative filtering*, in which the system recommends products based on the correlation between users who have purchased or rated products from the site. This differs from the non-personalized approach, because the system attempts to group people with similar tastes and makes similar predictions only for people within a group.

A recommender system can adopt any one or a combination of these schemes to make a prediction [Schafer].

Many different E-commerce companies currently use different types of recommender systems. The Album Advisory feature of CDNOW.com recommends 10 albums to the customer once the customer has selected a particular album. Customers can also setup their own music store based on albums and artists they like or dislike. On the customer’s request, CDNOW.com

will predict a list based on the customer's store and other people's correlation to that customer. Similarly, Amazon.com has the *Customer who Brought* feature which suggests similar items to users who have selected a particular item. This is a people-to-people correlation and requires organic navigation, navigation introduced solely to assist the recommender system, from the customer. *Eyes* and *Amazon.com Delivers* are email based recommendations that depend on attributes and the customer has to explicitly enter keywords or select options to take advantage of this feature. Similarly, a customer can request a listing of the Top N items which are based on other people's correlation to that customer's likes or dislikes. The Feedback Profile feature at eBay.com provides average ratings and text comments to build profiles of sellers, which is based on the concepts of aggregated ratings. The Style Finder Tool at Levi's gets basic information from the customers and asks them to rate a few selected articles of Levi's clothing. Once this has been built into the customer's profile, recommendations are based on correlating with other customers. In a similar vein, MovieFinder.com's MatchMaker allows customers to locate movies with a similar theme, mood, cast or genre to a given movie. This is thus an item-to-item correlation. On the other hand, the We Predict tool at this site makes recommendations based on the history of the customer and other customer's correlations making it a people to people correlation. Just like Amazon.com and others, Reel.com provides similar Movie Matches to customers who select or buy a particular movie. There is another Editor's choice feature called Movie Map at this site which provides recommendations based on keywords and freeform.

Recommendations range from a single bit (recommended or not) to formless textual comments. Recommendations may be gathered explicitly or even by using implicit interest indicators. GroupLens monitor's user's reading times and provides personalized weighting based on past agreement among recommenders. It displays these recommendations alongside articles in

existing summary views. PHOAKS mines Usenet articles for mentions of URLs and provides a sorted display of articles. SiteSeer mines personal bookmark lists and displays recommendations. GroupLens, PHOAKS and SiteSeer employ different variants on weighted voting to aggregate evaluations. Fab goes further and does filtering/selection based upon a combination of evaluations and content analysis. Sparks.com might utilize any one or a combination of these approaches in order to display which cards a user may like. Most generally, the cards with higher prediction values might be displayed before those with lower prediction values.

Sites can also be categorized by the amount of work they require their customers to do. Customer interaction can range from minimal, such as organic navigation, to requesting recommendations or selecting lists to explicitly entering keywords or freeform. Many E-commerce sites use persistent systems that require manual efforts from the user, based on the philosophy that if creating the relationship between the customer and the site requires some degree of manual effort from the customers, they will prefer to return to their site in which they have invested. On the other hand, purely manual recommender systems are highly transferable since the customer could freely go to another site with the same manual features to obtain recommendations. Similarly, fully automatic, completely ephemeral recommendations also allow for the customers to visit other similar E-commerce sites. Hence, it seems that most E-commerce sites prefer an implementation that is persistent, partially automatic, hoping to quickly gain new customers, while rewarding long standing customers with valuable recommendations based on their input [Schafer].

2.2 Collaborative Filtering

One method for filtering information is collaborative filtering. This technique makes predictions based on the similarities and differences between profiles of several users. Correlations are based upon ratings of users and then like-minded recommendations are suggested. Collaborative filtering techniques are powerful since humans rate the relevance, interest and quality of the items. Therefore, even difficult qualitative properties of items, such as personal preference based on emotions towards an item, can be determined.

There are some problems associated with collaborative filtering. In particular three problems arise: the early rater problem, the sparsity problem, and the gray sheep problem [Claypool 1999]. The early rater problem occurs when items that are completely new are introduced to the system. These new items have no user ratings on them. The sparsity problem occurs when the number of items far exceeds what a single individual can read. Finally, the gray sheep problem is evident when there are users that do not consistently agree or disagree with others. These users will rarely receive correct predictions from collaborative filtering.

In order to overcome the early rater problem, one approach uses three different filterbots to compare spelling, included messages, and lengths to rate the message before the message is recommended [Miller 1996]. A filterbot will then rate the document based on these factors. These bots are treated as users. If certain users correlate well with the bots, then the predictions about those users are more accurate. Experiments done on Usenet newsgroups using such a system concluded that the varying metrics did help for some newsgroups.

Most collaborative-filtering algorithms are neighborhood-based methods. In these methods, a subset of users are chosen based on their similarity to the active user and a weighted average of their ratings is used to make predictions for the active user. Other algorithmic

methods that exist are Bayesian networks, singular value decomposition with neural net classification, and induction rule learning [Herlocker 1999].

The process of making collaborative-based predictions can be generalized into three steps:

1. Weight all users with respect to similarity with the active user.
2. Select a subset of users to use as a set of predictors.
3. Normalize ratings and compute a prediction from a weighted combination of selected neighbor's ratings.

[Herlocker 1999].

Each of these three steps can be implemented in many different ways.

The first step in neighborhood-based prediction algorithms is to weigh all users with respect to similarity with the active user. Different similarity weighting measures such as the Pearson correlation coefficient and the Spearman rank correlation coefficient can be used. Other methods include the vector similarity “cosine” measure, the entropy based uncertainty measure and the mean-squared difference algorithm. It has been seen that these different methods work much better for domains of different sizes, constraints and properties. Because user purchase histories are used as the data for filtering, the number of cards one user bought in common with another user can be used as the similarity factor.

Various improvements can be made to improve the definition of similarity between users in this first step. The accuracy of prediction algorithms can be improved if a correlation significance weighting factor is introduced that would devalue similarity weights based on a small number of co-rated items. This method would put more trust on the weight of users that have more data points in common. Ratings on certain items can also be more helpful in

determining user correlations than others by observing specifics about card differences instead of generalities [Herlocker 1999]. For example, a more distinct correlation difference can be made between users who buy cards with Garfield the Cat on them, and users who buy cards with any cartoon character on them.

The second step is to select a subset of users so that the active user can be “grouped”. Different techniques such as thresholds and best-n-neighbors are used to select neighborhoods. There are two types of thresholds that can be used: correlation and history. In a correlation threshold, the active user will be grouped only with users who have a similarity value greater than or equal to some threshold value. In a history threshold, the active user will be grouped with users when both the active user and other users agree on at least a fixed number of documents [Gokhale 1999]. The history threshold differs from the correlation significance weighting factor associated with the first algorithm in that the threshold determines who will be grouped with the active user based upon the prediction numbers from the first step of the algorithm, whereas the latter is a scaling factor used to acquire those prediction numbers. The other technique, best-n-neighbors, groups the active user with users who have the n-highest similarity values with the active user. Because Sparks.com had no explicit ratings for how much a user liked or disliked a card, the correlation threshold cannot be used. The history threshold, however, can be used by stating that a user will be grouped with others if these users bought at least n-number of cards in common. Alternately, the best-n-neighbors method can be used by grouping a user with n-number of other users who had the highest number of cards in common with the active user.

The final step is to normalize the ratings and compute a prediction. The basic way to combine all the neighbors’ ratings into a prediction is to compute a weighted average of the

ratings, using the correlations as the weights. This basic weighted average assumes that all users rate on approximately the same distribution [Herlocker 1999].

Although collaborative filtering can greatly enhance information filtering, it cannot locate information for a specific content information need [Herlocker 1999]. In order to acquire this specific information, content-based filtering is used.

2.3 Content-based filtering

Another way to filter information is through content-based filtering. Pure content-based filtering determines if two items are similar based solely on a profile built by analyzing the content of user-rated items. This form of filtering has proven to be effective in locating textual documents relevant to a topic using different techniques like vector-space queries [Herlocker 1999]. Content-based filtering is more than just specific keyword matching. For example, if a user purchases a greeting card with a picturesque view of the ocean on the front, this may mean that the user enjoys cards with nature pictures, not solely the ocean. This form of filtering is shielded from problems faced by collaborative filtering techniques such as the early-rater problem, the sparsity problem, and the gray-sheep problem, because they use techniques that can be applied across all documents [Claypool 1999].

One way to implement content-based filtering is by matching document keywords with keywords in the user profile. Then, the percentage of keywords that match can be given by the following formula:

Equation 2.1 – A content-based filtering calculation.

$$ContentMatchPercentage = \frac{|P \cap D|}{\min(|P|, |D|)}$$

Where:

- P = set of words in the user profile
- D = set of words in the document

Articles with a high percentage can then be displayed to the user [Claypool 1999]. This same idea can be applied to the domain of greeting cards. If we treat cards as “documents”, the content from the cards, “the words”, can be acquired by observing their many attributes.

There are some downsides to content-based filtering. One downfall is that these techniques cannot determine the quality of an article. Also, content filters may become inefficient when there are a huge amount of documents within each content category. This inefficiency comes from the increasing number of comparisons made between the profile and each document in order to determine the amount of content match. Hence, a number of approaches have been made to use a combination of content-based and collaborative filtering techniques to gain the benefits of both the techniques and remove the drawbacks present in the individual approaches [Claypool 1999]. This combination approach is exactly what we evaluate for Sparks.com.

2.4 Combination of Collaborative and Content-based filtering

Collaborative filtering systems can benefit from the addition of content-based filtering systems. Such a system can benefit from the ability to make early predictions that cover all items and users by using content-based filtering, as well as from the accurate collaborative filtering predictions as the number of users and ratings increase [Wasti 1999]. There are two ways to perform the combination of collaborative and content-based filtering. The first way is to perform “content-based collaborative filtering”. In this scheme, the grouping of users is done using the content in their profiles [Soboroff 1999]. The second scheme, which will be used in our

recommender system, involves scaling the results of content-based filtering and collaborative filtering separately [Claypool 1999].

In one system called ProfBuilder, web pages are recommended using both types of filtering systems. Users are presented with two lists, one produced by a content filter and the other by a collaborative filter [Claypool 1999]. The user thus sees the predictions of both independently. In another system called P-Tango, predictions are made using a weighted average of the predictions from two separate collaborative and content-based systems on a per-user and per-item basis. These weights are determined by the strength of each prediction and can therefore change over time to reflect the change in the user's tastes [Claypool 1999].

2.5 Quality of a Recommender System

The quality of a recommender system can be defined in many ways. Most generally, recommender systems that can accurately and consistently predict what a user would buy are of high quality. This idea of quality can depend on the abilities of the algorithms to properly compute predictions, the speed of the computations, and even the space required to store what is necessary for the calculations. The quality can also depend on the meaning and presentation of the final output.

Prediction algorithms have three dimensions on which quality of a recommender system can be measured. These dimensions are: coverage, statistical accuracy, and decision-support accuracy. These notions are defined as follows:

- Coverage: This is a measure of the percentage of items for which a recommender system can provide predictions. Coverage applies more specifically to collaborative filtering, because

small groupings of neighbors and unrated items cause certain items to have no valid predictions, which affects coverage. For example, if nobody bought 10 cards, then there is no way to determine if any of the users would like or dislike those cards based on pure collaborative filtering, thus lowering the coverage for collaborative predictions.

- Statistical accuracy: Statistical accuracy measures the accuracy of a filtering system by comparing the numerical predictions against user ratings for the items that have both predictions and ratings. Either mean absolute error (MAE) or root mean squared (RMS) can be used to attain this statistical accuracy. For example, a content-based filtering algorithm would have a high statistical accuracy if a user-rated value for a card matched closely with its predicted value. This measure is not completely useful at Sparks.com, because as stated earlier, the users currently cannot explicitly rate the cards.
- Decision-support accuracy: Decision support accuracy measures how effectively predictions help a user select high-quality items from the item set. For example, a poor decision-support accuracy would be reflected when the highest rated card for a user is .65, and the user can only see cards which rate higher than .70.

[Herlocker 1999].

In addition to filtering algorithms, the quality can also be defined in terms of the system's final output. There are two basic classes of displaying filtered information to users. The first class involves the presentation of items one-at-a-time to the users along with a rating indicating potential interest in the topic. The second class involves the presentation of some ordered list of

recommended items. In the first case, the user can be presented with the actual raw score from the filtering engine. In the latter, the same scores could be used to sort the list, hiding the actual percentage match from the user. This sorted list can be presented to the user either in a linear fashion, or in some form of tree [Breese 1998].

2.6 Filtering @ Sparks.com

In order to develop a recommender system, the domain of greeting cards was carefully analyzed. Greeting cards have a number of unique quantitative and qualitative attributes. Using past purchase histories from stored user profiles, the attributes from those purchased cards can be observed. Using this information, content-based filtering can then be applied, because enough information is then available to suggest cards with the most popular attributes on a per-user, per-item basis. Furthermore, collaborative filtering can be applied by observing which users purchased which cards. Again, a user profile of past purchases can be extremely useful. Based on similarities and dissimilarities in these attributes, users can be divided into separate groups, and collaborative filtering can take place.

In order to perform content-based filtering on cards, their attributes, or quantifications that differentiate one card from another, must be clearly defined. Sparks.com has a large amount of human-entered data for the attributes of each card. The following shows a list of these attributes recorded for every card in the database. These attributes provide information to perform content-based filtering. However, each card does not necessarily have relevant information entered for every field. For example, some of the cards have an “N/A” or a blank for certain fields and therefore contain no specific information that can be used for content-based filtering. These card attributes are defined as follows:

- Properties with IDs: characteristics of cards that have a specific property defined by a specific identification number.
 - Line ID – Identification number for a product line of a particular card manufacturer.
 - Charity ID – Identification number for the charity associated with the card.
 - Inside finish ID– Identification number for the type of material finish given to the inside of the card.
 - Outside finish ID– Identification number for the type of material finish given to the outside of the card.
 - Fold ID – Identification number for the way the card is folded.
 - Medium ID – Identification number for the type of artistic medium for the card.
 - Paper ID – Identification number for the type of paper used to make the card.
 - Designer ID – Identification number for the name of the person or company that designed the card.
 - Personalize ID – Identification number to show if and how a personalized message can be written inside the card.

- Properties with text descriptions: characteristics of cards defined by groups of words or sentences.
 - Visual objects – Text descriptions of the pictures on the front of the card.
 - Front text – Text already written on the front of the card.
 - Inside text – Text already written on the inside of the card.
 - Back text – Text already written on the back of the card.
 - Adjective themes – Text descriptions of the emotions or sentiments associated with the card.
 - Visual actions – Text descriptions of the actions depicted by the pictures on the card.

- Properties with levels: characteristics of cards defined by the level or amount of that characteristic represented. Therefore, one card can have “more” or “less” of these attributes than another.
 - Fun – Low, medium, high rating for fun description of card.
 - Beautiful – Low, medium, high rating for beauty description of card.
 - Humorous – Low, medium, high rating for level of humor associated with card.
 - Sincere – Low, medium, high rating for level of sincerity associated with card.
 - General – Low, medium, high rating for the general characteristics of the card. This attribute was vaguely defined in their system.
 - Sentimental – Low, medium, high rating for the level of sincerity for the card.
 - Sophisticated/Elegant – Low, medium high rating for the level of elegance for the card.

- Religious/Spiritual – Low, medium, high rating for the religious tone of the card.
 - Formal – Low, medium, high rating for the level of formality in the card.
 - Hip/Trendy – Low, medium, high rating for the level of trendiness associated with the card.
 - Cynical – Low, medium, high rating for the level of cynicism associated with the card.
 - Romantic – Low, medium, high rating for the romantic tone of the card.
 - A little naughty – Low, medium, high rating for the level of naughtiness associated with the card.
 - New age/mystical – Low, medium, high rating for the mystical nature of the card.
 - Adults only – Low, medium, high rating for the level of adult content in the card.
 - Suggestive – Low, medium, high rating for the level of suggestive themes in the card.
- Other Properties: any other characteristics used to quantify cards
 - Price – Cost to purchase the card.
 - Box or Single – Information regarding if the sale involves one card or a box of cards.
 - Artist name – Name of the artist who made the card.

The goal of this project was to determine if filtering could successfully be implemented at Sparks.com. Content, and collaborative filtering were both implemented and studied separately to see if one method worked better than the other. Then, a combination of collaborative and content-based filtering was implemented as described earlier in section 2.5 to see if this combination produced better results than either separately. The next section of this report, the methodology, presents a detailed description of the processes of developing the filtering system and the logic and assumptions behind filtering applied to the domain of greeting cards.

3. Methodology and Implementation

This section presents an in-depth view to the methodology and implementation processes behind the entire filtering system. First, a general overview to the methodology is given as an introduction to the steps taken to design and create the filtering system. The sections following the introduction to the methodology expand upon steps taken. In these sections, design issues, design choices, and explanations are presented.

Before even thinking of filtering, we first searched for the relevant information available to be filtered. This information was then separated into two categories: information that enables content-based filtering and information that enables collaborative-based filtering. A list of the attributes found in their database, also known as the cards' "content", was presented in Section 2.7 of the report. This sifting of information was performed by first studying their overall database schematics in the form of Entity-Relationship (ER) Diagrams and then by constructing various MS-SQL queries to sift out information relevant to collaborative and content-based filtering. The queries used to acquire the relevant information can be seen in Appendix C. In addition, the marketing division of Sparks.com informed us about which attributes of cards they believed their users found most important. Next, a new database was created with all of the information relevant to filtering. The ER diagram and all explanatory information for this database and its design can be seen in section 3.1 of this report.

After determining the information to be filtered and creating a new subset of their database, the overall design of the filtering system was developed. This design is presented in Figure 3.2 in the next section of this report. Next, the actual algorithms for collaborative and content-based filtering, and for the combination of both filtering types were developed. These algorithms and descriptions of how they work can be seen in section 3.3 of this report. Next, the

entire filtering system was designed using CRC cards and pseudocode. All of these design schematics can be viewed in Section 3.4 of the report. Next, the system was implemented in Java. Java was chosen because their entire existing system was already developed in Java, and because of its ability to easily interact with MS-SQL. The actual code for the system can be viewed in Appendix B.

After the working system was developed, tests were performed to see how well the filtering system worked in terms of speed and accuracy. Collaborative and content-based filtering were first tested separately. After testing each separately, the combination approach was tested to see if the results improved. The actual testing strategies used are discussed further in Section 3.5. Finally, their entire current system was studied to determine where a filtering system would fit.

3.1 Extracting the WPI Database

A subset of the Sparks.com database, dubbed the “WPI Database” was created for the filtering system for two reasons. The first was to allow for offline calculations. This way, profile updates and other calculations could take place on a separate database rather than the database being used for active user processes. The second reason was that calculations could be done at any time period on a separate system. This way the filtering system would not slow down the main system.

The WPI database was the basis for a decision-support system, also known from an analytical perspective as a business intelligence system. The WPI database was designed to overcome some of the problems encountered if Sparks.com was to attempt performing strategic analysis using the same database used for online transaction processing (OLTP). Their OLTP

system is characterized by having large numbers of concurrent users actively adding and modifying data. At a specific point in time, the OLTP database represents the state of a particular business function at a specific point in time.

Because there is a large volume of data maintained, the OLTP system may become overwhelmed with processing tasks. As the original database grows larger with more complex data, response time can deteriorate quickly due to competition for available resources. This particularly applied to the Sparks.com OLTP system, because many users continually add new data to the database while fewer users generate reports from the database. Therefore as the volume of data increases, reports take longer to generate. As Sparks.com collects increasing volumes of data by using OLTP database systems, the need to analyze data becomes more important.

The Sparks.com OLTP system is designed specifically to manage transaction processing and minimize disk storage requirements by a series of related and normalized tables. Hence, using the Sparks.com OLTP database to analyze data presents a certain set of problems:

- Users may not understand the complex relationships among the tables, and therefore cannot generate ad hoc queries.
- Application databases may be segmented across multiple servers, making it difficult for users to find the tables in the first place.
- Security restrictions may prevent users from accessing the detailed data they need.
- Database administrators may prohibit ad hoc querying of OLTP systems, to prevent analytical users from running queries that could slow down the performance of mission-critical production databases.

By working on a subset of the database extracted from the OLTP system, the response time for reports and queries improved because a smaller amount of data was present. The aforementioned problems could also be avoided by simplifying table relationships and changing securities.

Additionally, by using a subset of the OLTP system database, we gained the following benefits:

- Data was organized to facilitate analytical queries rather than transaction processing.
- Data transformation rules could be applied to validate and consolidate data when this data moved from the Sparks.com OLTP database into the WPI database.
- Security and performance issues were resolved without requiring changes in the production systems.
- Pre-aggregation of frequently queried data was made possible by enabling a very fast response time to ad hoc queries.
- An intuitive multidimensional data model was developed that facilitated selection, navigation, and exploration of the data.
- A powerful tool for creating new views of data was constructed based upon a rich array of ad hoc calculation functions.
- Technology to manage security, client/server query management and data caching, and facilities to optimize system performance was created based upon our filtering needs.

In order to test the filtering system we captured a large subset of user and card data that remained static during the entire project. This way, filtering accuracy and consistency could be observed more easily, because the system did not update dynamically as new orders were placed. Furthermore, this prevented bias caused by the generation of new orders in the sense that these

new orders can introduce changes in predictions. Therefore, the same predictions were guaranteed for any particular user each time the system was run on the static database. Moreover, the system can be easily uploaded to the OLTP database.

3.1.1 Details of the WPI Database

The WPI Database extracted from the Sparks.com OLTP was further divided into two parts:

- **WPI XMAS Database:** This database contained customer, card and order information about Christmas cards. Christmas cards generally reflect the taste of the recipients, because people may buy Christmas cards depending on their own tastes, likes and dislikes.
- **WPI BDAY Database:** This database contained customer, card and order information about birthday cards. Birthday cards generally reflect the tastes of the receiver.

The filtering algorithms were tested using both of these databases. The analysis of filtering performance results helped to determine whether filtering worked well for both types of cards. Before extracting the WPI database from the Sparks.com OLTP, it was necessary to determine if the Sparks.com was dense enough to support filtering. Initial statistical analysis was done on the Sparks.com OLTP and the following results shown in Table 3.x were deduced.

Table 3.1. Statistical analysis on raw data.

Analysis	Sparks.com OLTP	WPI XMAS DB	WPI BDAY DB
Total number of customers in the database	119,902	4,112	7,063
Total number of cards in the database	10,193	526	1,533
Total number of *active cards in the database	7,193	526	1,533
Total number of orders in the database	87,587	9,200	12,957
Number of customers who have purchased one or more cards	34,152	3,898	4,624
Number of customers who have purchased more than one card	13,796	1,614	2,419
Number of cards that are bought by at least one customer	6,372	516	1,414
Number of cards that are bought by more than one customer	5,635	498	1,280
Average number of customers one card is sold to	9	17	9
Average number of cards one customer buys	2	2	1
Maximum number of times a card is sold	338	96	83

Maximum number of cards a user purchased	441	79	58
Male Customers	18,165	911	1,405
Female Customers	83,809	2,149	4,199
Not Available	17,928	838	1,489
Maximum number of word matches between user and card profile	Not Available	1,534	503
Minimum number of word matches between user and card profile	Not Available	0	0
Avg number of word matches between user and card profile	Not Available	21.1	34.5
Avg size of customer bag	Not Available	207.1	195.5
Avg size of card word bag	Not Available	21.9	21.8

* Active cards are the cards that are presently in stock. Only active cards are displayed on the web page

The raw data shows that there are roughly 7 to 10 times more users than cards. In order to perform filtering, users must have previously bought cards. In order to get a benchmark on the possible success of filtering, the density of the database was calculated. Only users who bought cards were used in calculating the density, because users that did not buy cards could not benefit from filtering.

The density was calculated using the total number of active cards, number of customers who bought more than one card, and the number of orders using the values from Table 3.1. The number of orders could be used, because each order represented a user-item pair, which takes

care of the fact that cards could be purchased simultaneously. The following shows the density calculations for each chunk of card types:

- Overall Sparks.com OLTP: 0.088% density
- WPI Christmas: 1.084% density
- WPI Birthday: 0.349% density

The Christmas cards section has the greatest level of density and therefore has the best chance of performing filtering successfully.

3.1.2 Tables in the WPI Database

A series of tables were created in the WPI database to hold all relevant user, card, and purchase information. This section explains each of the tables in detail along with the meaning of each of their attributes. Statistics about these attributes are also presented. Finally, the entire relationship between all of the tables is presented in an ERD diagram in Figure 3.1.

- 1.) SP_WPI_CARD
SP_WPI_XMAS_CARDS
SP_WPI_BDAY_CARDS

These tables contain information about the card SKU numbers and the 27 attributes of the cards necessary to perform content-based filtering. In addition, fields such as visual objects, visual actions, inside text, front text, back text and adjective themes of the cards are also stored in these tables. These fields that contain text are used to build a “bag of words” for each card. This bag of words is stored as a separate column in this table. Finally, the card profile of each card is built using its content attributes and stored in this table.

As described in Section 2.7 of this report, there are many different types of card attributes available to describe the paper greeting cards at Sparks.com. The following list again presents these card attributes and also the justification for their inclusion or exclusion in the database:

Properties with IDs

- **Line ID** – The product line of a company that made the card.

There are 239 possible product lines for the cards. About 2000 cards do not have any product lines listed in the database. Line ID is a potential attribute to test filtering on because the number of line-ids is small compared to the number of cards that have them.

- **Fold ID** – Type of folding the card has.

About 300 cards do not have a fold associated with them. The rest of the cards have about 10 types of fold. Hence, we used this attribute to test for filtering.

- **Medium ID** – Artistic medium of the card.

601 cards have none associated. The rest of the cards have about 21 medium ids between them. Hence, this attribute was also used for filtering.

- **Paper ID** – Type of paper used to make card.

Only about 250 cards do not have their paper ID information. About 40 paper IDs are assigned to cards. Therefore this is another good attribute for testing filtering.

- **Designer ID** – Name of person or company that designed the card.

About 9000 cards have no relevant information for designers. The rest of cards have 1 of 94 designers. Also, designer ID was suggested to be a popular attribute to do filtering from initial surveys we conducted at Sparks.com. Therefore this attribute was included to conduct filtering.

- **Personalize ID** – How personalized text will be written inside the card. 4 options: Handwritten only, computerized text only, both handwritten and personalized text and no personalization. About 7000 cards can use either handwritten or computerized. Hence, personalize ID was used for filtering.
- **Envelope color** – color of the envelope available with the card.
Cards can have one of 15 different envelope colors. Customers might be choosy about envelope colors and hence this attribute was considered for content-based filtering.

Properties with Text Descriptions (Bag of Words Attributes)

- **Adjective themes** – Emotions associated with the card.
This attribute was used to build the bag of words for the card and used in filtering. These attribute emotions were entered by the data entry group and were not taken from any particular list.
- **Visual actions** – Actions depicted by the picture on the card.
This attribute was used to build the bag of words for the card and used in filtering.

- **Visual objects** – Text descriptions of the pictures on the front of the card.

This column can be parsed in the database to get breakdowns of each type of object. For example, a list of cards with at least one picture of a ‘heart’, or a list of Christmas cards with pictures of candles and Santa can be acquired using their *visual objects* field. This column can thus be used to divide cards into specific categories to determine what kind of cards a particular user likes. One particular drawback with using the visual objects attribute is that the list of distinct words that define the pictures on all the cards may be lengthy, leading to computational overhead. This attribute was used to build the bag of words for the card and used in filtering.

- **Back text** – Any text found on the back of the card.

Out of the 11,000 cards studied, about 8000 cards do not have back text information available. The rest have information about the name of the company or the artist, company slogans, copyright information, or paper information. Back text was used to build the bag of words for the card.

- **Front text and inside text** – Any text written on the front or inside of the card.

This was the same as visual objects. Any articles and prepositions can be removed.

Therefore it will be less to sift out than visual objects. Both front text and inside text were used to build the bag of words for the cards.

Properties with Levels

- **Relevance Attributes:** 16 attributes that are used to describe the tone of a card.

These 16 attributes were listed in Section X.Y. The data entry group at Sparks.com has manually fed in the tones reflected by each and every card in the database. Each and every card has at least 2 tones. Moreover, for each tone, a card can have low/medium/heavy values. Hence, relevance attributes were considered for filtering.

Properties with continuous values

- **Envelope Width** – width of envelopes

Envelopes are available in different weights. Hence, this attribute was also used to make recommendations on cards.

- **Envelope Height** – height of envelopes.

Envelopes are available in different heights. Hence, this attribute was also used to make recommendations on cards.

- **Price** – Cost of the card.

Many users often consider the price of the card while buying it. So this attribute was used for filtering.

Other Properties (Not used in this Filtering Engine)

- **Charity ID** – Portion of the sale of card goes to charity.

About 60% of the cards do not have a charity associated with them. The rest have one of 23 charities associated with them. Again, because the number of charities is small relative

to the number that have charities, this is another good attribute to try on filtering.

However, charity ID information about a card is not visible to customers on the present Sparks.com web page. Hence, charity ID cannot be used for making recommendations until then, and is therefore not included in content profiles.

- **Inside finish ID** – Type of material finish the card is given inside.

About 150 cards do not have an inside finish associated with them. The other 10,850 have one of about 5 inside finishes associated with them. This could also be a potential attribute to do filtering on once the Sparks.com web page provides the inside finish id information about cards to its customers. Therefore this attribute was not included in content profiles.

- **Outside finish ID** – Type of material finish the card is given outside.

About 1800 cards do not have an outside finish associated with them. Hence, outside finish could be a potential attribute to do filtering on once the Sparks.com web page provides the outside finish id information about cards to its customers. Therefore this attribute was not included in content profiles.

- **Box or Single** – Information if only one card or a box of cards sold.

For all of the cards this information was not available. Therefore no filtering will be performed using this attribute.

- **Artist name** – Name of the artist who made the card.

About 7,500 cards have no artist information. The remaining cards have one of 1,500 different artist names associated with it. This means that about 3000 cards have 1500 different artist names. Hence, artist names were not used for filtering cards.

2.) SP_WPI_ORDERS
SP_WPI_XMAS_ORDERS
SP_WPI_BDAY_ORDERS

These tables store the customer IDs and the card SKUs for all the orders. Queries can be run on this table to find out statistics about the buying patterns of customers such as average order size, and minimum and maximum number of orders. This is useful, because a list of the cards purchased by each user can be used for collaborative filtering. Only the distinct orders are passed to the filtering system since the use of duplicate orders might give undue advantage to certain cards and users and thus bias the final predictions. Hence, separate tables were created to hold distinct orders.

3.) SP_WPI_CUSTOMERS
SP_WPI_XMAS_CUSTOMERS
SP_WPI_BDAY_CUSTOMERS

These tables store customer information such as unique customer ID, gender, and birthday. A unique customer profile is made for each customer based upon the card profile of the cards purchased by that customer. The customer profile is also stored in this database. Gender and birthday information are not used by the present filtering engine for filtering.

4.) SP_WPI_XMAS_CUS_BAG_PROFILE
SP_WPI_BDAY_CUS_BAG_PROFILE

A bag of words is built for each customer by taking the union of the card bag of words for the cards that were purchased by that customer. The unique customer ids and their bag of words are stored in this table.

5.) SP_WPI_ATTR

This table contains information about the content attributes that were used to form a card profile for each card. Information about the average types of each attribute (mean, mode or median), the types of attributes (discrete, 0 or 1, continuous) and the weights attached with each attribute are also stored in this table.

6.) SP_XMAS_CARD_USER_WORD_RAT
SP_BDAY_CARD_USER_WORD_RAT

For each customer and for each card, the customer bag of words is compared with the card bag of words. The weighted number of matches results in a word rating for each customer-card pair. This word rating is stored in this table. In addition, the content, collaborative, and final predictions for each card and user are also stored in this table once the filtering system runs on all the customers and cards.

7.) SP_WPI_XMAS_CARDINT_MAP
SP_WPI_BDAY_CARDINT_MAP

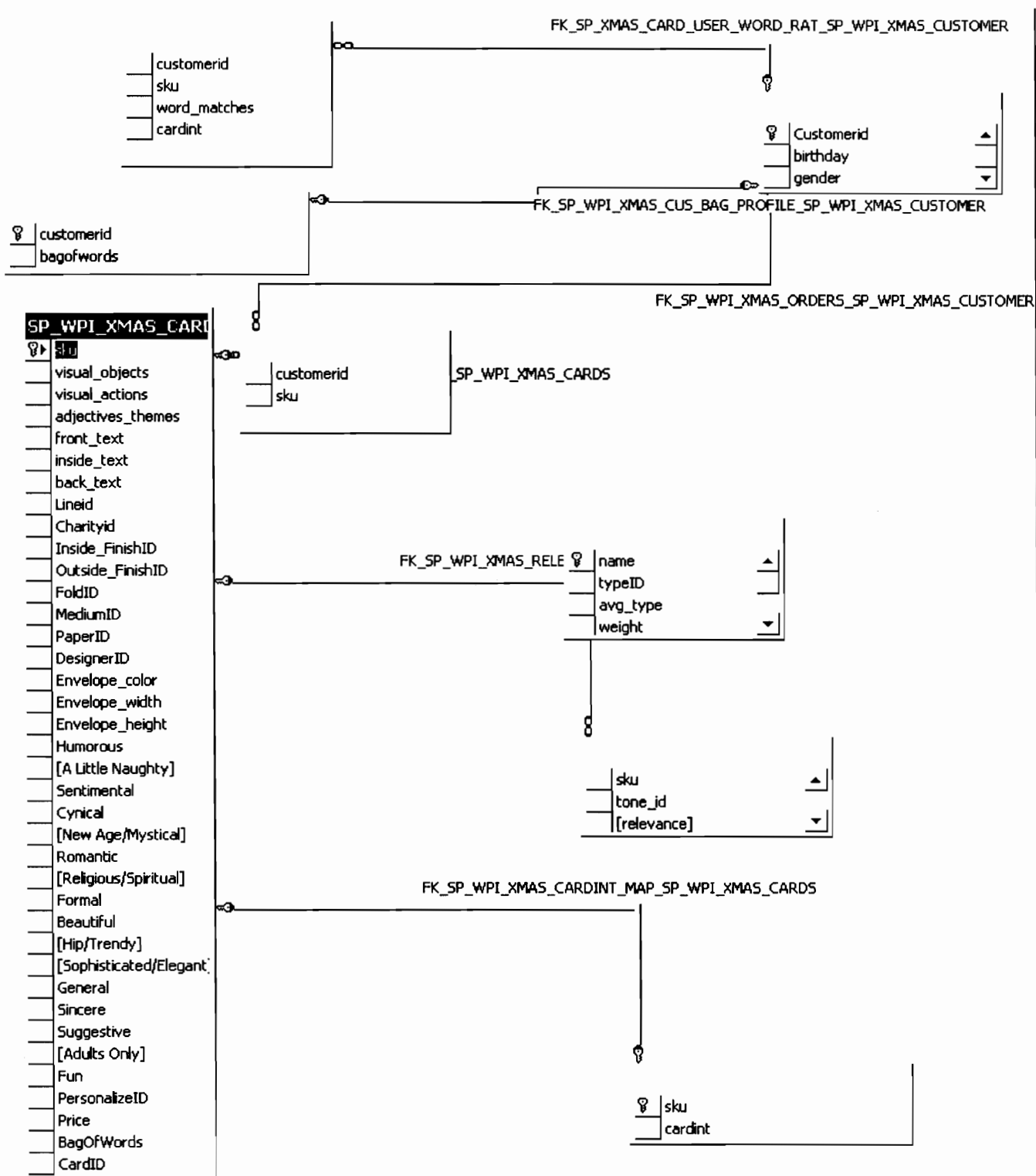
Each card is mapped to an integer from 1 to the maximum number of cards. Since original card SKUs are seven digits long, mapping them to smaller integers helps to improve speed and performance of the filtering algorithm.

- 8.) SP_WPI_XMAS_RELEVANCE
 SP_WPI_BDAY_RELEVANCE

These tables contain information about the card SKUs and their relevance attributes. This information is combined and finally inserted into the SP_WPI_XMAS_CARDS or SP_WPI_BDAY_CARDS tables respectively.

In order to describe the relationships between these tables and attributes more clearly, an ER diagram was constructed. The relationship between tables in the WPI database extracted from the Sparks.com OLTP is shown in Figure 3.1.

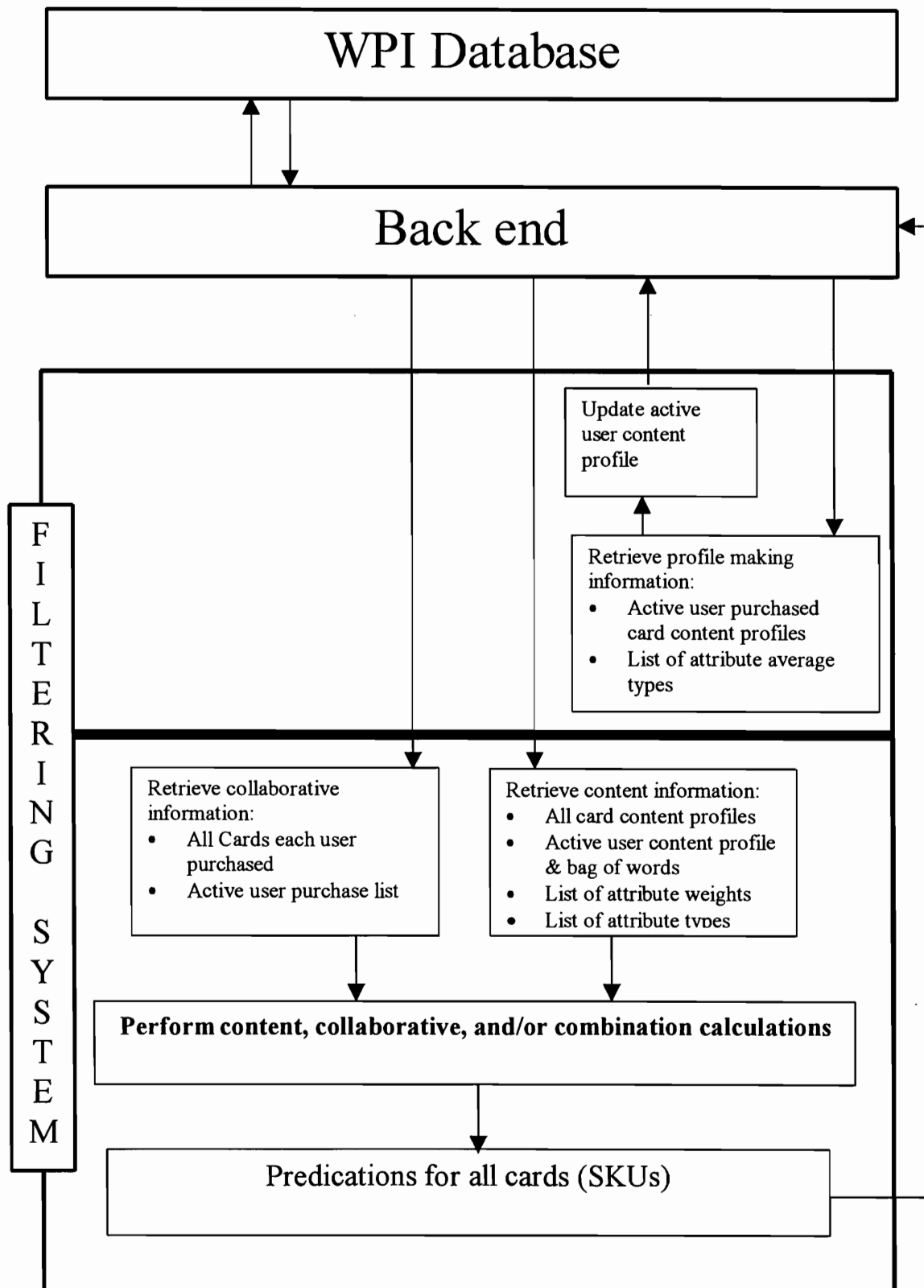
Figure 3.1. WPI Database ER diagram.



3.2 Recommender System Overview

As previously described, an overview of the functions of the recommender system was created before any collaborative or content-based algorithms were designed. Figure 3.2 summarizes the main functions and interactions within the recommender system. Additionally, an overview of the tasks and information needed in order to make predictions is outlined.

Figure 3.2. Overview of recommender system.



The recommender system is broken into two separate portions: the back end and the main filtering engine. This separation was done so that the filtering system would not be dependent on the database architecture. Therefore, if another back end were to be designed, or if the database type changed, the filtering system would still work if passed the appropriate information in the format specified. Another reason why these systems were separated was now the filtering system could either perform calculations during online or offline times. In other words, filtering could be done on the fly while a user was logged into the system, or instead as a nightly event. In general, if the updates were done online, latency would be increased, thus slowing down an active user, and in our case, decreasing test calculation times.

The back end's main purpose is to interact with the database and pass all relevant information between the filtering system and the database. In order to send data to the filtering system, the back end calls the database using a particular query, formats the data in the manner requested by the filtering system, and then passes that information to the filtering system. In order to update the database, the filtering system grabs the information from the filtering system, formats that data in a way recognizable by the database, and constructs and calls a storage query.

The filtering system is a stand-alone system whose main purpose is to create user profiles and calculations predictions for every card in the filtering system knows about for each user. Therefore, every time the system is called to make predictions, predictions will be made for every card known by the filtering system. Because the system is stand-alone, the filtering system only knows about card and user information given from some outside source, in this case the back end. The main filtering system's main functions can be broken down into five tasks:

- Calculating user profiles
- Calculating content-based predictions

- Calculating collaborative predictions
- Calculating the combination of content-based and collaborative predictions
- Formatting predictions in a consistent manner

The main filtering system works with one user at a time. Therefore, a content-based prediction for one user cannot be made at the same time as the collaborative prediction for another user.

After the main filtering engine calculates predictions, the information is returned to the back end, as shown in Figure 3.1. In a real system, a front end may be implemented which shows the users the results of the calculations. However, in order to simplify testing, the back end receives the predictions so that this information could be stored in the database, and so statistical analyses on the accuracy of the filtering system could be more easily performed.

The next two sections expand on the details of the main filtering system highlighted in Figure 3.1. First, an in-depth view and explanation of the filtering algorithms used is presented. Then, the actual code design is explained.

3.3 Filtering Algorithms

In this section, each of the algorithms involved in filtering are discussed. First, the bag of words, which was calculated outside of the main filtering system, is discussed. Next, the ideas behind profile making, content-based predictions, collaborative predictions, and combination predictions are examined. The theory behind each is discussed, and then an example calculation for each algorithm except for the combination approach due to its simplicity is provided. The actual pseudocode for these algorithms can be viewed in Appendix C.

3.3.1 Bag of Words

The bag of words is an entity used in our content-based filtering scheme that contains all of the words found in the text-based attributes of cards outlined in Section 2.7. Each card has a bag of words containing all of the text found and textual descriptions of the images, actions, and emotions associated with each card. Additionally, each user has a “net” bag of words, which holds not-necessarily distinct words for each card the user purchased.

The bag of only words for cards and users contains only relevant words from the text descriptions. In other words, useless words, prefixes, and suffixes from words were removed. Useless words, words such as articles, conjunctions, and prepositions, were removed using a stop list, because they not only describe nothing specific about the card, but may also create more matches between cards that do not reflect content similarities. Prefixes and suffixes were removed so that there would be no difference between the words “run” and “running”, which imply the same meaning. These operations were performed using a stemming utility and stop word list from the following website:

http://www.dcs.gla.ac.uk/idiom/ir_resources/linguistic_utils/.

As previously described a bag of words is created using six text-based card attributes. For each card, all the words in these fields are combined into one bag. The count of words is preserved to give more weight to words that occur more than once in a card. Therefore, a word may appear more than once in a given bag. A user bag of words is built by combining the bag of words of the cards that the user purchased. Again, the total count of each word is preserved.

In order to calculate the percentage of match between a card’s bag of words and the user’s bag of words, a comparison between the number of words in common is taken. If the same word matches twice, it is counted twice in the comparison. This process repeats for each card to

get the number of word matched. Finally, each match is divided by the highest rating in order to normalize the results.

In order to show more clearly how the bag of words makes calculations, an example using the following data shown in Table 3.2 is performed.

Table 3.2. Example data for bag of words calculation.

Card #	Inside Text	Visual Objects	Bag of Words
1	Merry Christmas and a Happy New Year	snow; reindeer; Santa; sleigh	merry; christmas; happy; new; year; snow; reindeer; santa; sleigh
2	Season's Greetings and Happy New Year	snow; tree; children	seasons; greetings; happy; new; year; snow; tree; children
3	Santa says he hopes you weren't naughty this year	Santa; presents; coal	santa; says; hopes; naughty; year; santa; presents; coal

As shown, the bag of words for each of the cards contains a list of words taken from the inside text and visual objects fields, ignoring unnecessary words. Suppose that a user purchased card 1. This user's bag of words would contain all of the words found in the bag of words for card 1. In order to determine the percent match between words, the number of words in common must be calculated and then those ratings must be normalized based on the highest number of words in common. These results are shown in Table 3.3.

Table 3.3. Bag of words rating based on user's purchase of card 1.

Card #	Counted # of words in common	Normalized rating
1	10	1.00
2	4	0.40
3	5	0.50

One interesting note to mention is that the third card has 5 words in common and not 2 (Santa and year). Because Santa would appear in the user's profile twice if the user purchased card 1, the word gets counted twice. Since Santa also appears in this card's bag of words twice, that word Santa gets a count of 4. A second important observation is that the cards purchased seem to score higher than all of the cards not purchased in the example presented. This is not necessarily the case, because as the user profile grows to include more cards, the chance to gain not only more words, but also duplicate words increases and thus as in the case with card 2, certain words may get scaled much higher than others.

3.3.2 Constructing User Profiles

In order to perform content-based filtering, attribute profiles were used in addition to the bag of words described in the previous section. As described in Section 3.1, there is large number of card attributes to be used in content-based filtering. The card content profile is made by simply concatenating all of the attributes into one field separated by some marker, and by adding a stop marker after the last. The user profile is constructed by taking an average of each attribute from the cards that the user purchased. Therefore, if the user purchased five cards, the user profile would be constructed from an average of each of those five cards' attributes. The user profile looks exactly the same as the card profile and has the same number of attributes.

Averages can be taken in three ways: a mean, a median, or a mode. One issue in creating user profiles using averages was alluded to in Section 2.7, specifically the presence of zero values. As previously mentioned a zero was interpreted as "not available" rather a particular value. This is especially important in terms of the mode, because if a card contained a zero for a particular attribute, that zero did not count in final average. For example, if out of the five cards

purchased, three of the values for a particular attribute were zero, only the other two would be used to determine the value for that attribute in the user profile. Therefore, the only way a user could receive a 0 in their profile was if all the cards a user purchased all had zeroes for the same attribute. The following list shows attribute and average type pairs:

- Properties with IDs: mode
- Properties with levels: median
- Envelope color: mode
- Envelope height and width: median
- Price: median

Mode was used in places where properties could not be defined as “more” or “less” than another, and median was used otherwise. The mean was not used, because the mean is too sensitive to outliers in the data pattern.

In order to further explain the process of this algorithm an example using the example data from Table 3.4 is presented.

Table 3.4. Example data for profile and content-based calculations.

Card #	Designer ID	Fun rating	Price (cents)	Final Profile
1	5	30	395	5 / 30 / 395 #
2	8	0	250	8 / 0 / 250 #
3	5	0	325	5 / 0 / 325 #

As shown in Table 3.4, the card profiles are constructed by concatenating each attribute and separating them using a ‘/’ mark, and ending with a ‘#’ mark. In this example, the user purchased all three cards. If the user purchased all three cards, the final user profile would be the following:

- Final user profile: 5 / 30 / 325 #

The first attribute, the Designer ID, is calculated using the mode, because this is a property that has an ID. The second attribute, which is also calculated using the mode, is *not* 0, but 30. As stated earlier, a zero meant that the information was not available and therefore would not be factored into averaging. Therefore, even though there are more zeroes than any other number, they were thrown out and thus the mode is 30. Finally, the third attribute, price, is calculated using the median.

One interesting issue arises in the mode calculations when there are more than one most common numbers. When this arises in our system, the *lowest* value is taken as the mode. For example, if the sorted data set for a particular attribute appeared as {0, 0, 15, 15, 30, 30}, the mode calculated for that attribute, ignoring the zeroes, would be 15.

3.3.3 Content-Based Filtering Design

The general idea behind content-based filtering is to recommend cards based on the content similarity with respect to the user's profile. Before any filtering can take place, the content-based information relative to cards was collected in the form of the bag of words described in Section 3.3.1 and attribute separation as described in Section 3.3.2. The idea behind content matching is straightforward: compare the user's profile with a card's profile and see how well they match. However, the way in which they match differs per attribute.

Content attribute values between user profiles and card profiles are matched in two ways: either discretely or continuously. In a discrete match, the user profile attribute must exactly match the card's attribute. Therefore these attribute values can either match 100% or 0%. In a continuous match, the level of closeness between a user profile attribute and a card's profile attribute is calculated. In other words, the level of closeness may also lie between 0% and 100%.

Each content attribute, including the bag of words which is treated as one attribute, has one of these attribute types assigned. The following presents a list showing each attribute's average type:

- | | | |
|-----------------------------|---|------------|
| • Properties with IDs | — | discrete |
| • Properties with levels | — | continuous |
| • Envelope color | — | discrete |
| • Envelope width and height | — | continuous |
| • Price | — | continuous |

In order to calculate the total content match for a card, the user's profile was compared to each card one at a time, and a weighted sum of each of the attributes, including the bag of words was taken. During this comparison, the attribute type was observed in order to determine if a discrete match should be checked or if a percent closeness should be calculated. This closeness is determined by the following formula:

Equation 3.1. Continuous attribute percent closeness formula.

$$\%closeness = 1 - \frac{|user_attribute_value - card_attribute_value|}{Max(user_attribute_value, card_attribute_value)}$$

In addition to separating between discrete and scaled attributes, the amount of weight a particular attribute received in the total content-based prediction could be modified. Unlike a document, a card had a fixed number of attributes in both the user profile and the card profile. This way attributes that were seemingly more important to users received appropriate weighting.

One consideration is taken into account when performing these calculations, specifically the presence of zeroes. Any time a zero appears in either the user profile or card profile, that attribute was disregarded; its weight value is then spread evenly amongst the other attributes. This prevents cards with a number zeroes for attribute values not to be calculated with lower values, which is important because these cards may still match very well with the users profile for all other attributes.

In order to more fully explain the calculation procedure for content-based filtering, an example using the data from Table 3.4, the user profile calculated in Section 3.3.2, and the attribute weights in Table 3.5 is presented.

Table 3.5. Example weight data for content-based calculations.

Attribute	Weight
Designer ID	0.2
Fun rating	0.3
Price (cents)	0.5

Using all of the information aforementioned, the content-based predictions were calculated. Table 3.6 shows the predictions for each card.

Table 3.6. Content-based predictions using user's profile.

Card #	Designer ID	Fun rating	Price (cents)	Final content prediction
1	100%	100%	82.28%	91.14%
2	0%	N/A	76.92%	54.94%
3	100%	N/A	100%	100%

The final content-based predictions shown in Table 3.6 were calculated by taking the sum of the products of each attribute and its weight factor, ignoring any N/A values. As shown in Table 3.6, card 3 has the highest rating even though the fun rating contains a value of N/A. Any information not available in either the card profile or user profile would not be used; instead its weight would be evenly distributed to each of the other attributes.

3.3.4 Collaborative-Based Filtering Design

The general idea behind collaborative filtering is to try and group users together and make predictions based on the opinions of the users in that group. Most collaborative filtering

algorithms use correlations between user ratings to determine these groups. Because Sparks.com had no rating system, and because cards purchased is used as the implicit indicator, a slightly different approach to collaborative-based filtering is taken.

This different approach does not specifically follow the algorithmic procedure described in Section 2.2, but rather uses a twist on mathematical clustering to achieve collaborative filtering. The algorithm *does not* attempt to cluster users into groups, but rather observes the proximity between all the other users with respect to the active user, giving stronger weight to the opinions of those who users who fall “closer” to the active user. In other words, more weight is given to the opinions of users who agree more with the active user than those users who do not. For example, if two users bought two different cards and the active user agreed on more cards with the first user, then that first card would carry a higher recommendation than the second. The implementation of this idea is broken down into three steps as follows:

1.) Find out how close each user is to the active user.

This is performed by observing the user-item matrix, a matrix containing information about which users bought which cards, and then by counting the number of cards the active user has in common with every other user. The more cards a user has in common with the active user, the closer that user is located.

2.) Rate all the cards, giving a stronger weight to the cards bought by users closer to the active user.

This weight is determined by the number of cards the active user had in common with another user. Therefore, if the active user agreed upon 6 cards with another user, then

all the cards bought by that user would be given a weight of 6. Additionally, if two or more users bought that card, the rating would then be the sum of the weights for all those users.

3.) *Normalize all of the ratings.*

This is performed by finding the max rating and dividing every other rating by that maximum rating.

This algorithm only uses distinct purchases from each order. Therefore, if a user bought a card 32 times, this card will only count one time in the calculation, unlike the bag of words where multiple identical words increase the match amount. Additionally, the algorithm does not count matches between an active user and himself, as shown in Table 3.8 by the gray sections.

In order to better explain this algorithm, an example is performed using the data from Table 3.7.

Table 3.7. Example user-item matrix.

	Card 1	Card 2	Card 3	Card 4	Card 5
Mike	Y		Y		Y
Zak		Y		Y	
Sharad			Y	Y	Y
Amy	Y	Y		Y	Y

The first step of the algorithm determines how many cards each user has with each other. Table 3.8 shows these results in a user-user matrix. The number of cards in common with another user determines the weight between users.

Table 3.8. Example user-user matrix derived from Table 3.7.

	Mike	Zak	Sharad	Amy
Mike		0	2	2
Zak	0		1	2
Sharad	2	1		2
Amy	2	2	2	

Next, all of the cards for an active user are given a rating based upon the weights between users and normalized. For this example, Sharad will be the active user. His predictions are shown in Table 3.9.

Table 3.9. Example collaborative predictions for active user.

	Rating	Final Normalized Prediction
Card 1	4	100.00%
Card 2	3	75.00%
Card 3	2	50.00%
Card 4	3	75.00%
Card 5	4	100.00%

An interesting point to note about these ratings is that even though the maximum weight between Sharad and the other users is 2, the ratings for cards one and five is 4. This occurred because more than one user purchased those cards, in particular, Mike and Amy both bought those cards. Therefore each received a rating of $2 + 2 = 4$, a sum of the weights from each person.

3.3.5 Combining Content-based and Collaborative Filtering Predictions

After the final content-based and collaborative predictions are calculated, they can be combined to form another prediction for each card. Our project uses two different approaches to

calculate this combination prediction. These approaches are similar to the approach used in the P-Tango system discussed in Section 2.4. Both algorithms implemented in our filtering system use a sum of a weighted combination of the collaborative and content-based predictions, which are calculated on a per user basis. The algorithms differ in how the weights are determined.

3.3.5.1 Combination Approach 1

In this first approach, the weights are updated each time the user purchases new cards, and are determined by the relative accuracy of each filtering system by observing the absolute error in *both* the collaborative and content-based systems. The absolute error is determined using a distance factor. The predictions for each card are sorted in numerical order and then checked for placing. The highest rated card is given a place of 1, the next highest 2, and so on. The absolute error is calculated as the sum of the distances purchased cards are located from the first place. For example, if the user purchases 2 cards and the collaborative filtering portion ranks them as 100th and 35th place respectively, the absolute error for collaborative filtering is $(100 - 1) + (35 - 1) = 133$.

Each algorithm's absolute error is then calculated and then weights are assigned. The first weight is assigned by dividing one error by the sum of both errors. The second weight is then calculated as 100% minus the first weight. The system with the *lower* absolute error received the *higher* weight value. Finally, a combination set of predictions is calculated as shown in Equation 3.2.

Equation 3.2. Combination prediction equation.

$$\text{Combination Prediction} = (\text{Cont_Weight}) * (\text{Cont_Prediction}) + (\text{Collab_Weight}) * (\text{Collab_Prediction})$$

3.3.5.2 Combination Approach 2

This combination varies slightly from the first approach discussed in the previous section. In this approach, collaborative filtering is used solely as the factor for determining the weights. To acquire the weights a “benchmark” of the collaborative system is performed by first randomly pulling out a card the active user purchased, and then observing how well the collaborative system predicts for that card. The same distance metric as described in the previous combination is then used to see how far away the card ranked from first place. The value is then divided by the total number of places in order to determine the first weight. The final steps to determine the second weight, assign both weights to each type of filtering, and combine the results are exactly the same as in the previous algorithm.

3.4 Recommender System Design

This section further expands upon the general design presented in Section 3.1. First, each of the classes is viewed as a whole as shown in Figure 3.3. Then, the UML diagrams for the back end and filtering system are presented in Figures 3.4 and 3.5. Next, the steps for going through the entire system are presented in detail in Figure 3.6. The CRC diagrams for the back end and filtering system are presented in Figure 3.7. Finally, Figures 3.8 and 3.9 show the UML diagram and CRC classes for the bag of words, respectively.

One important fact to note is the presence of the “Main Main” class shown in Figure 3.3. This class acts as the interface between the back end and the main filtering engine. This area has three functions. The first function is to pass messages back and forth from the filtering system to the database. The second is to regulate which messages get sent at what time. The final function is to perform statistical analyses on the filtering system as described in Section 3.5.

Figure 3.3 Class Modeling.

This diagram shows the relationship and arrangement of the different classes in the filtering system. Collab, content and DB form the back end of the system and are connected to the WPI Database. Filtering Main, Collaborative System, Content System and Profile Maker System form the prediction calculation component of the system. MainMain connects the back end and the calculation component.

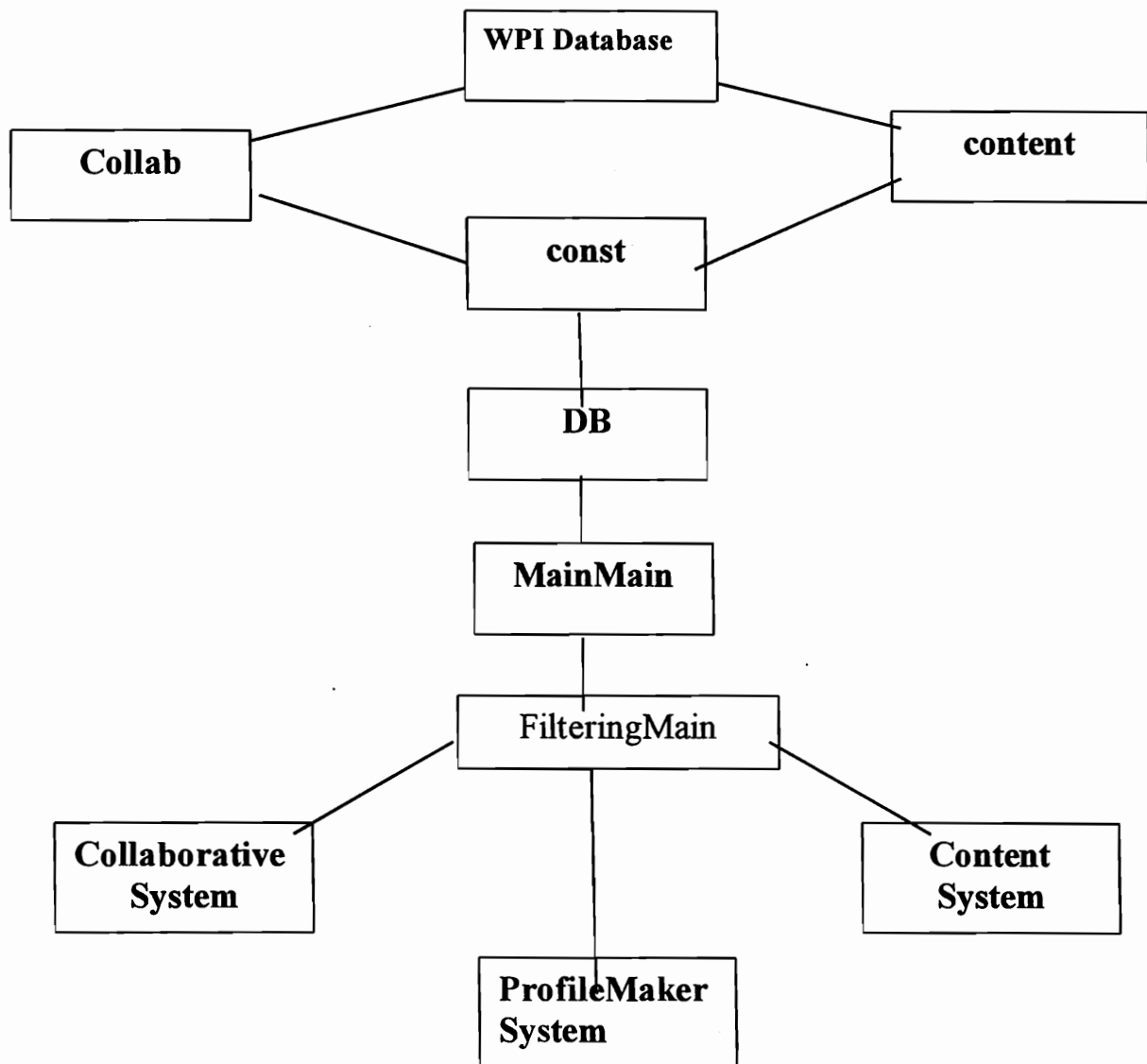


Figure 3.4. UML design for the back end.

This diagram gives a detailed listing of the functions in each class, the input and output parameters of the functions, and the relationships between the different classes. The const class is connected to the WPI database; and DB class is connected to the class MainMain.

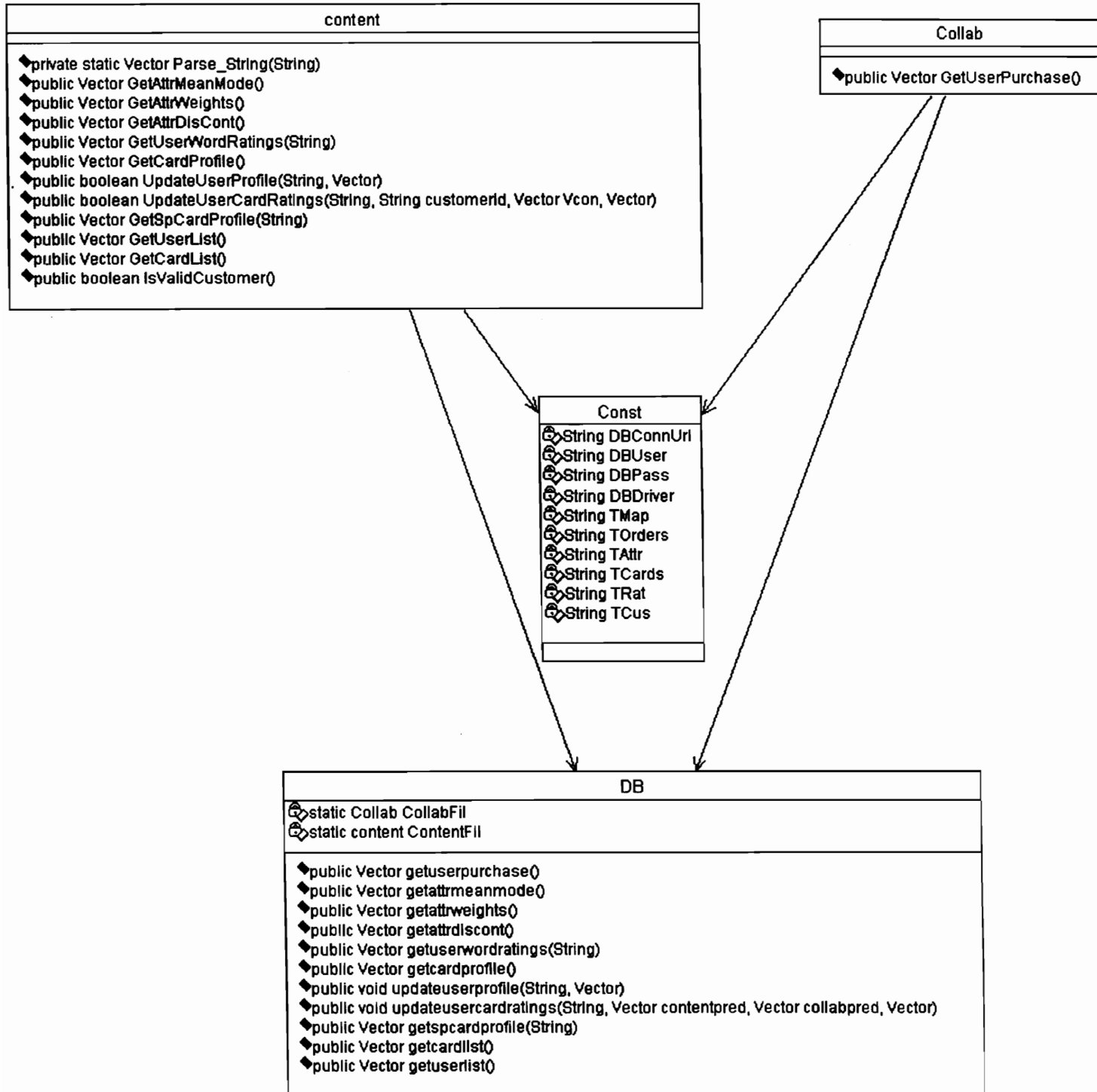


Figure 3.5. UML design for the main filtering system.

This diagram shows the detailed listing of the functions with their input and output parameters for all the classes involved with the prediction calculation portion of the system. The MainMain class connects to the backend of the system via the DB class.

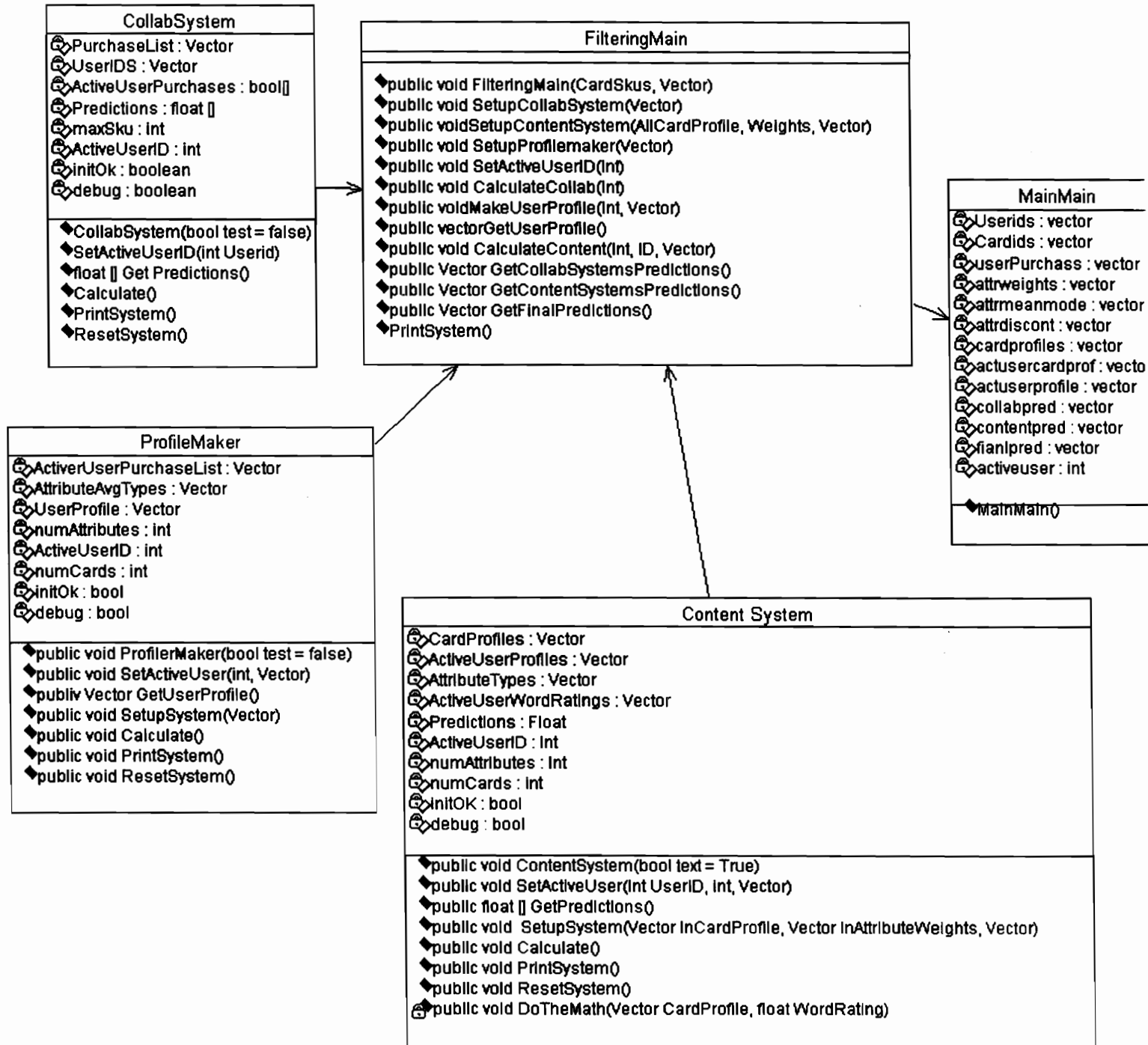


Figure 3.6. Step-by-step workings of the filtering system; a class action diagram.

This diagram explains the ordered steps taken by the filtering engine – from taking a userID as an input to outputting the prediction results. This also helps to describe the interactions and relationships between the different classes involved.

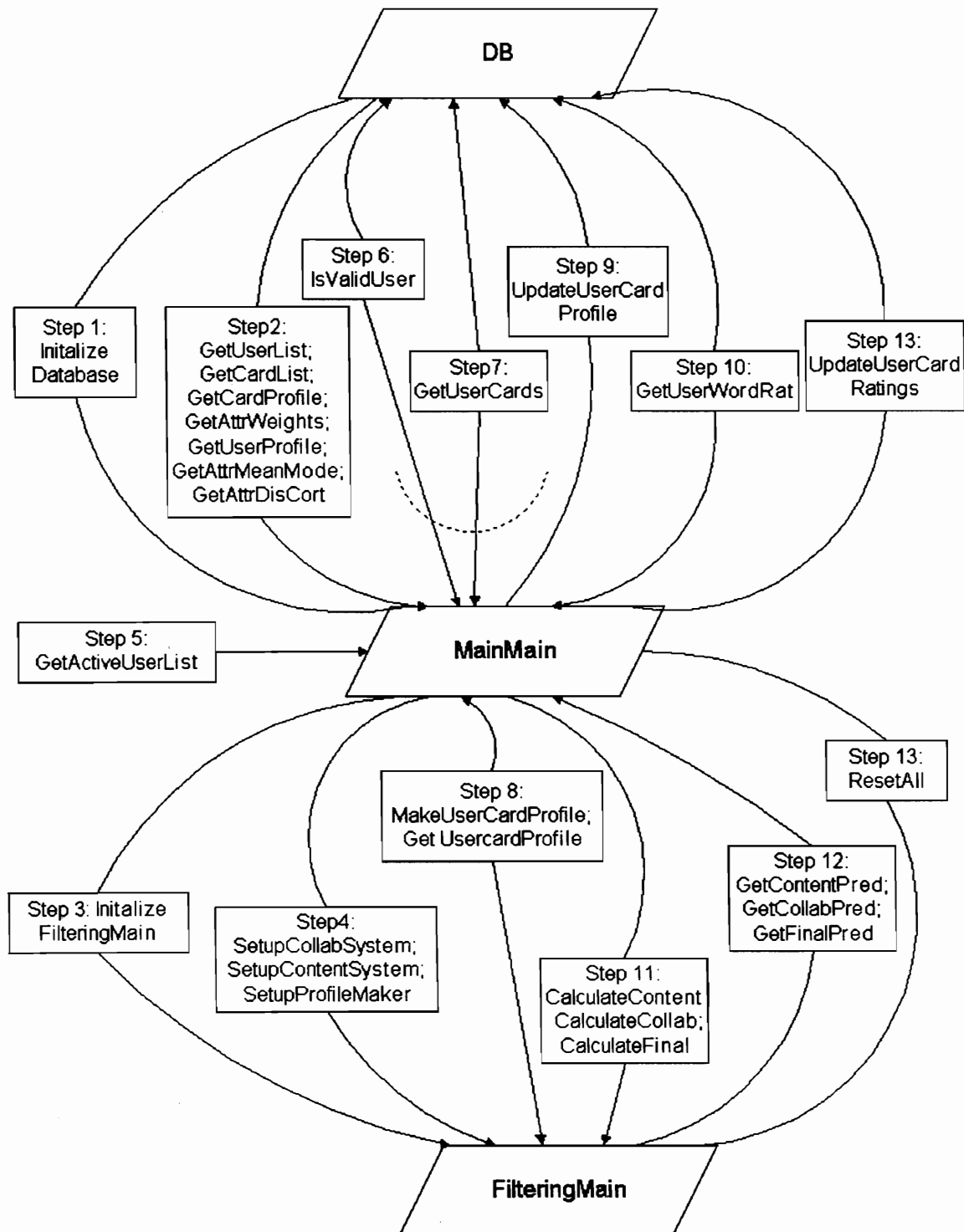
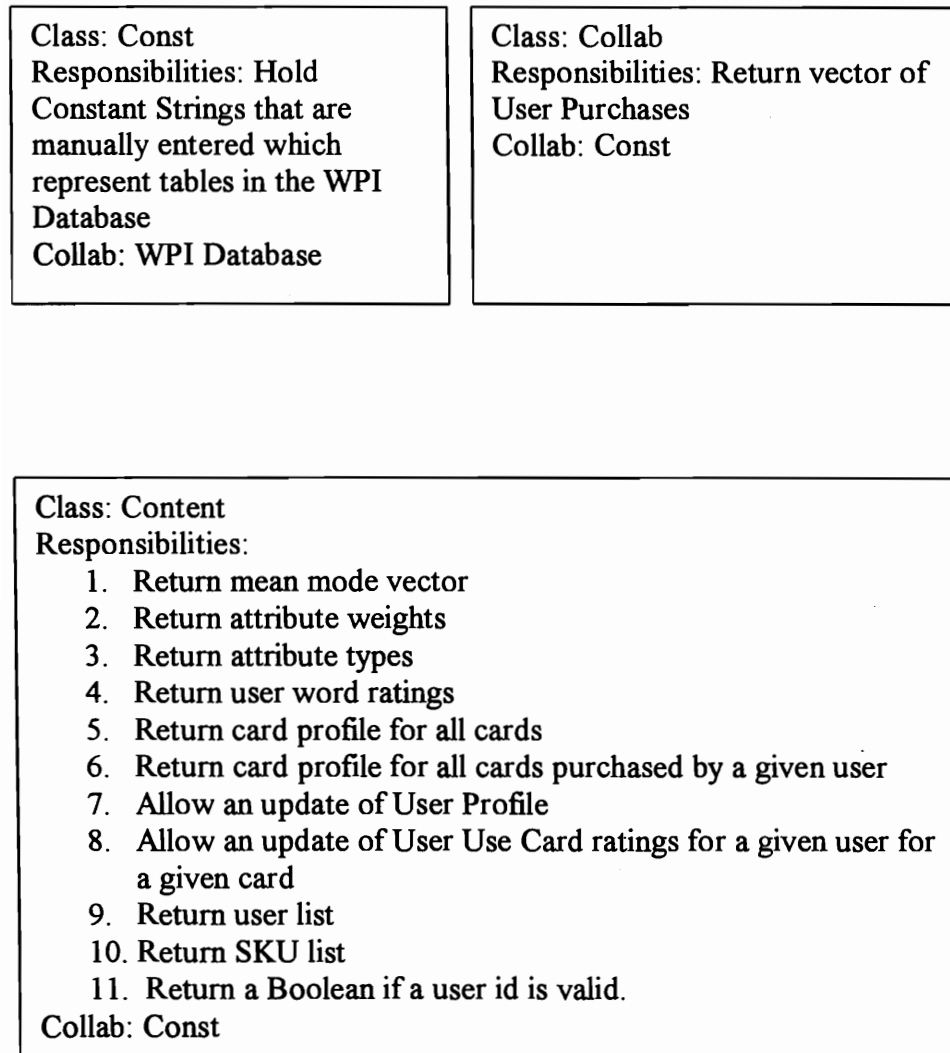


Figure 3.7. Class-responsibility-collaboration (CRC) diagrams.

These CRC diagrams are used to show the dependencies between the classes of the entire recommender system.



Class: DB

Responsibilities:

1. Return mean mode vector
2. Return attribute weights
3. Return attribute types
4. Return user word ratings
5. Return card profile for all cards
6. Return card profile for all cards purchased by a given user
7. Allow an update of User Profile
8. Allow an update of User Use Card ratings for a given user for a given card
9. Return user list
10. Return SKU list
11. Return a Boolean if a user id is valid.
12. Return vector of User Purchases

Collab: Content Collab

Class: CollaborativeSystem

Responsibilities:

1. Calculate collaborative predictions.
2. Acquire active user purchase list.

Class: ContentSystem

Responsibilities:

1. Calculate content predictions

Collab: none

Class: ProfileMaker

Responsibilities:

1. Calculate user profiles.

Collab: none

Class: FilteringMain

Responsibilities:

1. Pass and regulate messages to different calculating systems.
2. Calculate combination predictions
3. Keep track of all users and SKUs
4. Provide access to predictions in a uniform format

Figure 3.8. UML description for the bag of words.

This figure shows the interactions between the different classes used to implement the bag of words.

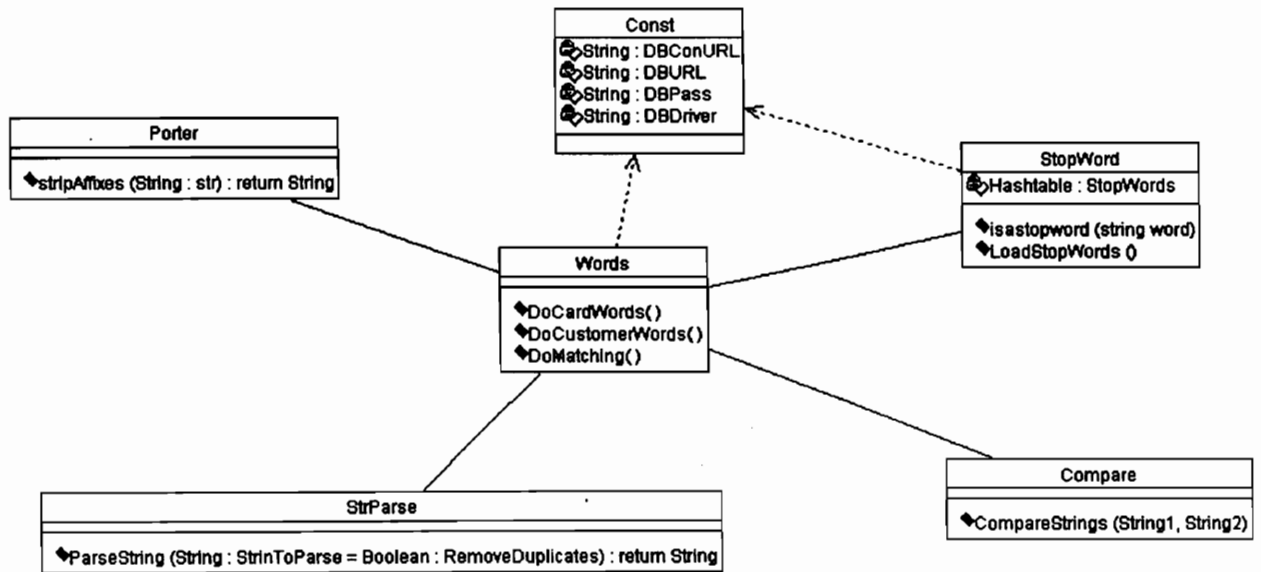


Figure 3.9. CRC diagrams for the bag of words.

The CRC diagrams show the dependencies for the classes used to create the bag of words.

Class: StopWords

Responsibilities:

1. To load the stop words from the database and store in a hash table
2. To return a Boolean which value is set to true if the word is a stop word

Collab: Words, Const, WPI Database

Class: StrParse

Responsibilities: To take a string and return the values are true or false

Collab: Words

Class: Porter

Responsibilities: Reduce a word to a stem.

Collab: Words

Class: Compare

Responsibilities: Takes two strings and returns how many words are the same

Collab: Words

Class: Words

Responsibilities:

1. To get a bag of words for individual cards
2. To get a bag of words for each user
3. To compare all cards bag of words for all users bag of words and store the result

Collab: Const, StopWords, Porter, StrParse, Compare, WPI Database

3.5 Testing Procedures

Initially, all the 27 content card attributes were equally weighed. The filtering engine was run and the results were analyzed to ensure the accuracy of the filtering algorithms. Testing was done separately on Christmas and birthday cards and their results were analyzed to see which combination works best for each of the two domains. The testing procedure was divided into 4 different phases. The goal of the first phase of testing was to find out which content attributes were more relevant for getting better content-based predictions for cards. In the second phase, the attributes were weighed according to the results obtained from the first phase. The filtering engine was then run with the updated weights and the content-based and collaborative predictions were compared to the case where the attributes were weighed equally. The attribute weights were further tuned based upon the accuracy of the results obtained. In the third phase of testing, the combined filtering algorithms were run on the sample set of users. The prediction accuracy results obtained were compared to the results obtained from the content-based and collaborative filtering approach alone. This was done to determine if the combination approach gave better results than the individual filtering algorithms. In the fourth phase of testing, the accuracy of the overall filtering system was tested by comparing it with a non-personalized system. For this, the content-based, collaborative and combined filtering predictions of cards were compared with results obtained from a random placement of cards.

Phase 1 – Testing the importance of attributes

The 27 content attributes were divided into 4 groups to decide the importance of attributes in making predictions. The groups were: price, bag of words, sentiments and a group of the other distinct attributes. These groups were created to facilitate analysis and study of the

importance of content attributes in making card predictions. A separate algorithm developed the bag of words rating for each card and user, and so this rating was treated as a separate group. The 16 sentiment attributes were considered as another group. Each card has an average of 2 sentiments attached to it. Hence, studying each of the 16 sentiments separately would not give substantial results, since most of the cards would have a rating of 0 (N/A) for that sentiment. The marketing and business development groups at Sparks.com suggested price to be a potential attribute driving user purchases. Hence, price was placed in a separate group. The remaining 9 distinct attributes were placed in a different group since they were discrete attributes and none of them were suggested to be particularly influencing card purchases. Another test was run on the discrete attribute Line ID to prove the point that testing these discrete attributes separately would not give any substantial results. Finally, tests were run with equal weights given to all the attributes to analyze the accuracy of collaborative filtering predictions.

For each group, the weight of the group was assigned as 1 and the remaining attributes were given a weight of 0. Ten random customers, or the maximum number of customers that were available, were extracted from the database from each of the groups of customers who had bought exactly n distinct cards for 2 to 100. This was done to ensure that the number of cards the customers bought did not bias the results. Next, one of the cards that the customers bought was removed from the database randomly. The filtering engine ran for the list of customers that had their cards removed. For each user, the results were returned in a decreasing order of card ratings. Thus, the placement/ranking for each card in this list was obtained. For each customer, the placement of the card that was removed was extracted and placed in a separate table. Statistics were generated by extracting these placement values, as well as the total number of distinct rankings, also called brackets, and the mean, median and mode of the size of all rankings

for the user. These statistics helped to decide which attributes worked best for content-based filtering predictions. Eventually, the user's orders were inserted back into the database.

Phase 2 – Testing predictions with weighted attributes

At the end of Phase 1, the importance of content attributes in making accurate predictions was determined. In Phase 2 of testing, the attribute weights were changed to reflect the results of Phase 1. For Christmas cards, the bag of words received the heaviest weight of 0.7, sentiments were given a weight of 0.1, price was set to be 0.1 and the group of the discrete attributes was given a weight of 0.1. Similarly, for birthday cards, bag of words, sentiments, price and the group of discrete attributes received weights of 0.85, 0.05, 0.05 and 0.05 respectively. Once the attribute weights were set, the Phase 1 tests were run again for 110 users and statistics were obtained for the average placement/ranking of cards. These statistics were compared to the statistics obtained when the attributes were weighed equally to determine if differently weighed attributes resulted in better predictions.

Phase 3 – Combined-filtering algorithms

In this phase, the combined-filtering algorithms were run on the sample set of users. Statistics were obtained on the average placement/rankings of the cards. These statistics were compared to the results obtained by running the content-based and the collaborative filtering methods alone. This was done to determine whether the combined-filtering approach gave more accurate predictions.

Phase 4 – Filtering system vs. non-personalized approach

In this phase, first the results of the content-based and collaborative approaches alone and combination of content-based and collaborative filtering approaches were compared to two non-personalized approaches:

- 1) The first approach was the random ordering of cards approach. The placements/rankings of cards for the sample set of Christmas and birthday card customers were generated randomly. Statistics were obtained from these rankings and compared with the statistics obtained from phase 2. This was done to determine if the filtering approach gave more accurate predictions than random ordering.
- 2) In the second approach, all the cards were sorted in the order of the decreasing number of times that they were purchased. Then the placements/ranking were generated for the sample set of Christmas and birthday card customers. The statistics generated were compared to the results from Phase 2.

4. Results

This section shows all of the results obtained from running the recommender system described throughout Section 3 of this report. The tests to gain the results were performed according to the procedures outlined in Section 3.5. Using this information, the potential success of filtering on the domain of greeting cards and filtering algorithms was determined.

Section 4.1 contains the initial tests that were run on the database to understand the buying patterns of users, card sales and the content information available about the cards. Once the content-based, collaborative and combination algorithms were implemented in the filtering engine, testing was done to determine which filtering algorithms worked best for the different types of cards. Tests were done to determine the weights to be given to the different attributes in content-based filtering in Section 4.2. In Section 4.3, the results obtained by using weighted attributes for content-based filtering have been explained. Section 4.4 presents exhaustive statistical testing and analysis that was carried out to compare the accuracy of different algorithms with each other. Finally, Section 4.5 compares our personalized filtering approaches to non-personalized approaches, in particular the random ordering of cards, and ordering by card popularity based on number of times the card was purchased, also referred to as a "top ten" list. The testing results have been described in this section with the aid of tables, scatter plots, box plots and histograms.

4.1 Initial Tests

This section describes the initial queries and tests that were run on the WPI database to understand the density of the database, and to roughly determine if filtering could be successful. Specifically, the areas studied were the numbers of distinct cards purchased by users and the

number of repeat orders. To ensure the validity of using content card attributes, the distribution of cards with respect to price, relevance attributes and distinct attributes was also studied. Tests were also run to understand the bag of words results generated for cards and users.

4.1.1 Number of cards bought by users

Filtering techniques are not very effective for users who only purchase one card. Ideally, as the user buys more cards, the chances of making better recommendations for them increase. Therefore, this relationship between users and purchases was studied by gathering the following data shown in Table 4.1.1 from the orders database. The histogram shown in Figure 4.1.1 graphically depicts this data. The duplicate column includes duplicate orders, that is, if a customer bought the same card more than once, those orders are included in the duplicate column. The distinct column only includes such orders once.

Table 4.1.1. Number of duplicate Christmas and birthday cards.

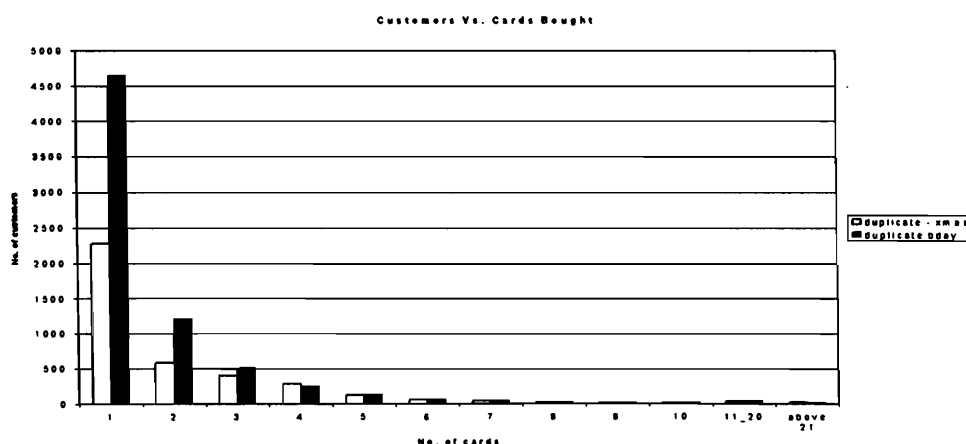
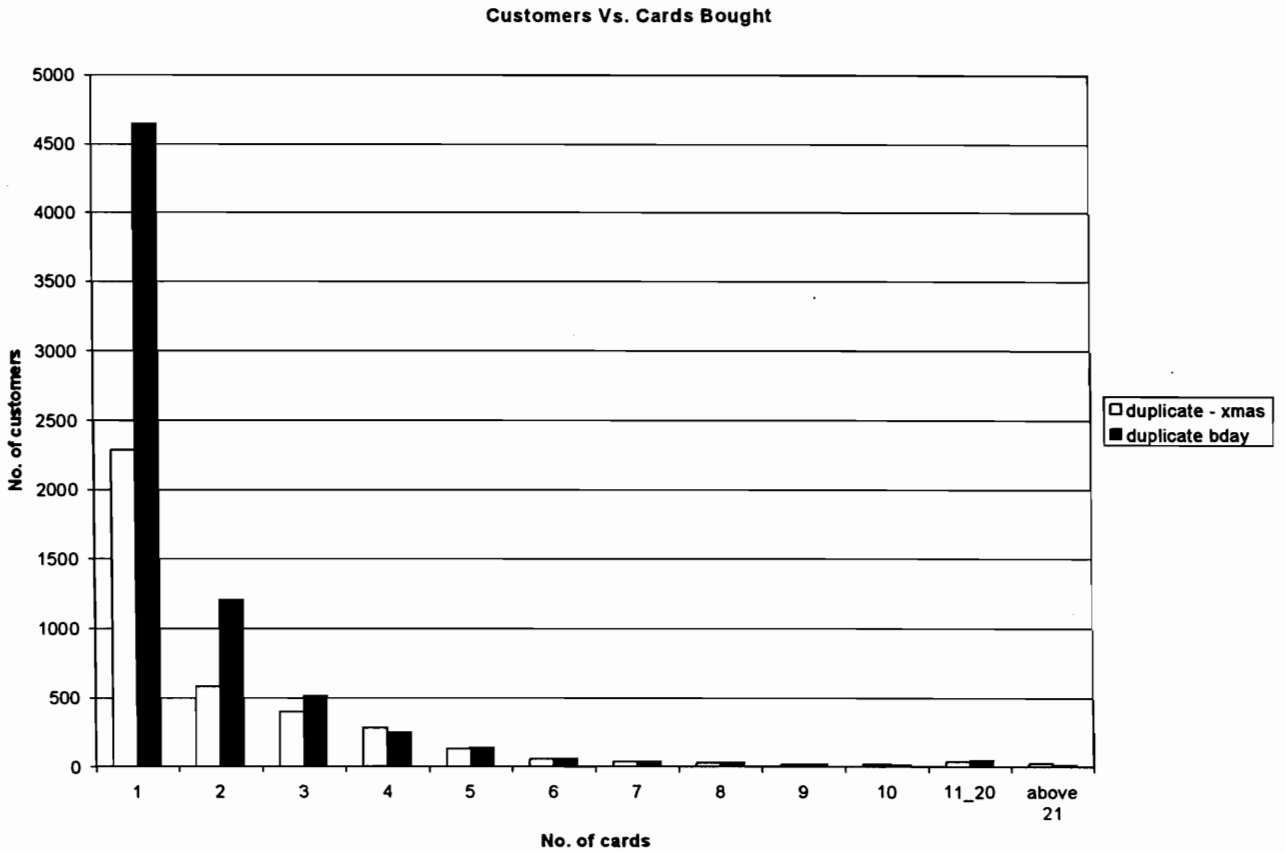


Figure 4.1.1. Number of duplicate Christmas and birthday cards card bought analysis.



From the data, (for Christmas duplicate cards allowed) $3898 - 2284 = 1614$ users have purchased more than 1 Christmas card. Because our testing method would not work for users who purchased less than 1 card due to the fact that removing one card would leave their purchase profiles empty, only these 1614 users along with the 526 Christmas cards will be used to form the data set for the filtering engine. The same idea applied for birthday cards as well.

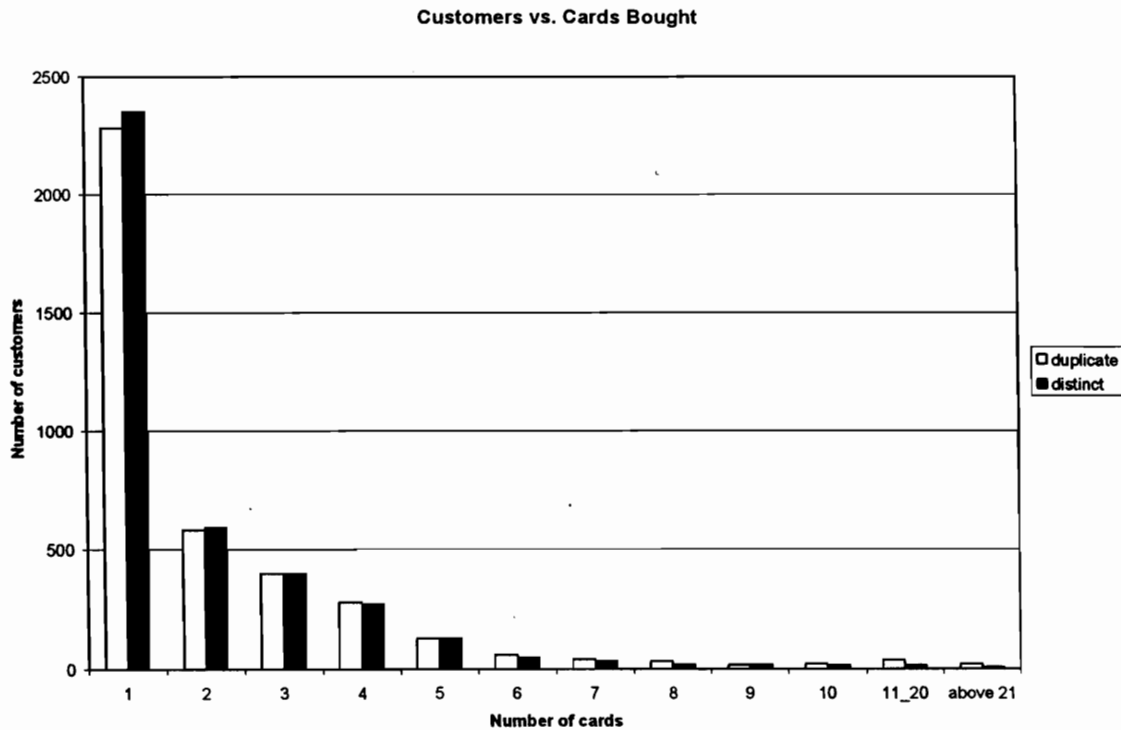
4.1.2 Distinct vs. Duplicate Orders

Users often purchase the same card more than once. This can influence the results of filtering algorithms by giving higher predictions to certain cards. Figure 4.1.2 gives an idea of the number of duplicate orders in the database based on the data presented in Table 4.1.2.

Table 4.1.2. Number of users for each bracket that bought duplicate vs. distinct cards.

num cards	duplicate	distinct
1	2284	2354
2	581	591
3	399	401
4	280	271
5	129	129
6	58	48
7	40	31
8	31	17
9	16	18
10	21	15
11_20	37	15
above 21	22	8

Figure 4.1.2. Number of users for each bracket that bought duplicate vs. distinct cards.



Moreover, allowing duplicate customer orders to affect customer's profile and calculations can be a decisive factor on the outcome of the filtering results. However, as seen in Figure 4.1.2, if only distinct orders are taken, there is a minor change in the distribution. This implies that customers purchase the same card more than once very few times. Although this happened few times, we decided to create an orders table consisting of only distinct orders and used that table for providing data to the filtering system so that in particular collaborative filtering would not be biased by duplicate purchases.

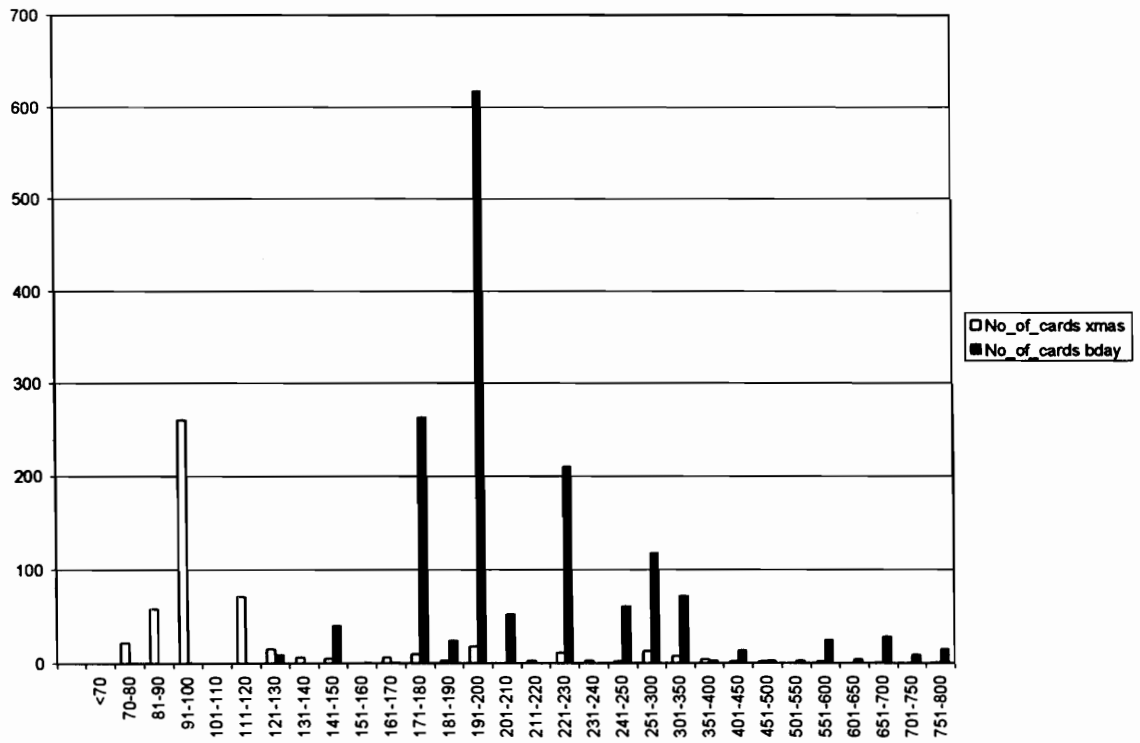
4.1.3 Distribution of Cards with Respect to Price

Price was suggested to be a potential card attribute influencing cards bought by customers. Hence, it was important to classify the distribution of cards with respect to price. Table 4.1.3 and Figure 4.1.3 show the data and histogram for price information respectively.

Table 4.1.3. Distribution of cards with respect to price.

Price_Range (cents)	No_of_cards xmas	No_of_cards bday
<70	0	0
70-80	22	0
81-90	58	0
91-100	260	1
101-110	0	0
111-120	71	0
121-130	15	9
131-140	6	0
141-150	5	40
151-160	0	1
161-170	6	1
171-180	10	263
181-190	3	24
191-200	18	617
201-210	0	52
211-220	3	0
221-230	11	210
231-240	3	0
241-250	2	61
251-300	13	118
301-350	8	72
351-400	4	3
401-450	2	14
451-500	2	3
501-550	0	3
551-600	2	25
601-650	0	4
651-700	1	28
701-750	0	9
751-800	1	15
Total	526	1533

Figure 4.1.3. Distribution of cards with respect to price.



- Note: Shipping of cards is free. There is however a 75 cents charge for personalization with the cards. This personalization charge is the same for all cards.

As shown in Figure 4.1.3, the price of Christmas cards ranges from 70 cents to 8 dollars. More than half of the cards are under 1 dollar, while the rest spread from \$1 - \$8. Birthday cards also show a similar spread. Because of these spreads, price appeared to be a good attribute to use for filtering.

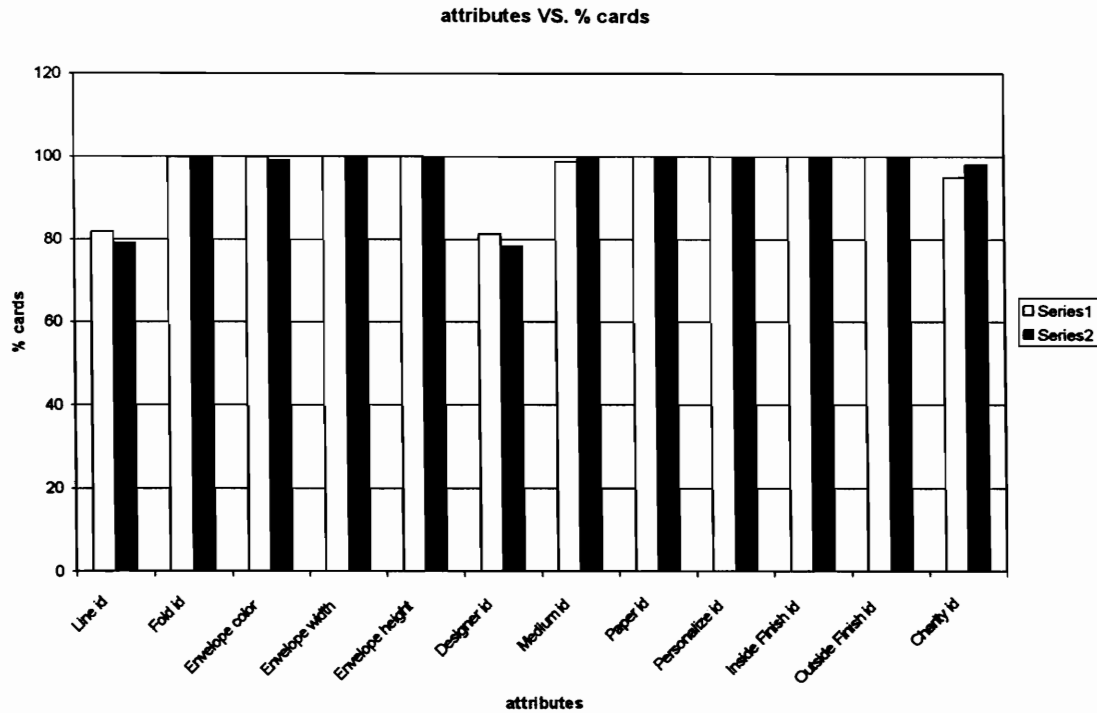
4.1.4 Distribution of Different Card Attributes

The attributes listed in the table below helped to build a unique card profile for each card. Table 4.1.4 and Figure 4.1.4 show statistics about these different card attributes. Although Charity ID, inside finish ID, and outside finish ID were not used as highlighted in Section 3.1, they may be useful once this information is shown to the users while buying the cards.

Table 4.1.4. Distribution of different card attributes.

Attributes	% cards	% cards
	xmas	bday
Line ID	81.75	79.06
Fold ID	99.81	100.00
Envelope color	99.81	99.15
Envelope width	100.00	99.54
Envelope height	100.00	99.54
Designer ID	81.18	78.21
Medium ID	98.86	99.54
Paper ID	100.00	100.00
Personalize ID	100.00	100.00
Inside Finish ID	100.00	99.93
Outside Finish ID	100.00	99.74
Charity ID	94.87	98.11

Figure 4.1.4. Distribution of different card attributes.



Note: As stated in Section 3.1, inside finish ID, outside finish ID and charity ID were not used.

As seen in Figure 4.1.4, more than 80% of the cards have a value for each of the attributes mentioned above. This implies that very few cards have a value of N/A (0) for these attributes. Hence, if a mode is taken to compute the average user profile from the cards for these attributes, the chances of 0's in the user's profile will be less, and thus these attributes were used in filtering.

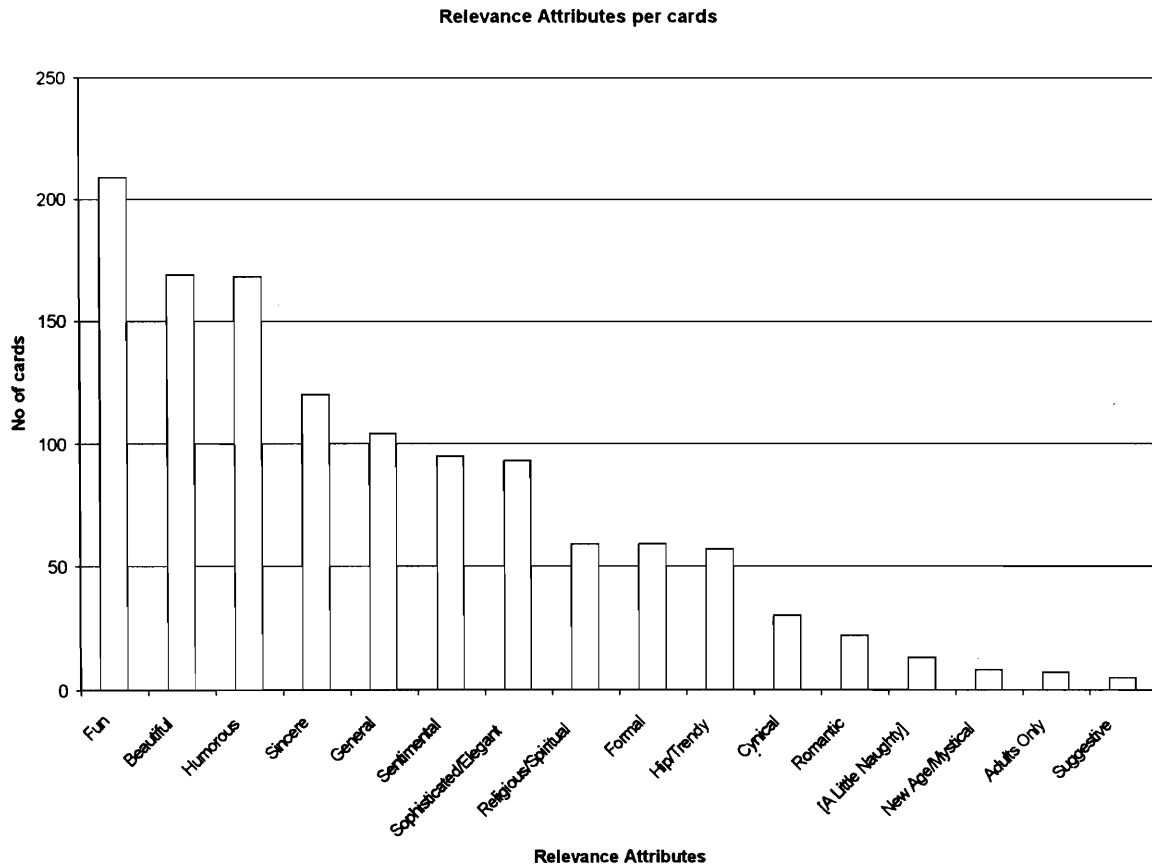
4.1.5 Distribution of Relevance Attributes For Christmas Cards

The relevance attributes represent the tones of the cards, which take on values of low, medium or high. If the number of cards having the different tones is very less, then these attributes could not be used to compute content-based filtering results. Table 4.1.5 and Figure 4.1.5 show the results of the data collection for the relevance attributes.

Table 4.1.5. Distribution of relevance attributes for Christmas cards.

Relevance_attribute	no_of_cards
	xmas
Fun	209
Beautiful	169
Humorous	168
Sincere	120
General	104
Sentimental	95
Sophisticated/Elegant	93
Religious/Spiritual	59
Formal	59
Hip/Trendy	57
Cynical	30
Romantic	22
[A Little Naughty]	13
New Age/Mystical	8
Adults Only	7
Suggestive	5
Total	1218
Total_cards	526

Figure 4.1.5. Distribution of relevance attributes.



Note: Relevance values of 0(n/a) have been dropped.

As shown in Figure 4.1.5, unlike other attributes, cards did not have many ratings for the relevance attributes. Less than half of the cards had relevance ratings. Hence, taking mode directly resulted in an excess number of 0's in the user's profile. To prevent an excess number of zeroes to be in the user profile, as highlighted in Section 3.3.3, only non-zero values were used to compute the average in the user's profile.

Although the number of cards having relevance ratings was small, the conglomeration of the sentiments into one group may have potential for filtering.

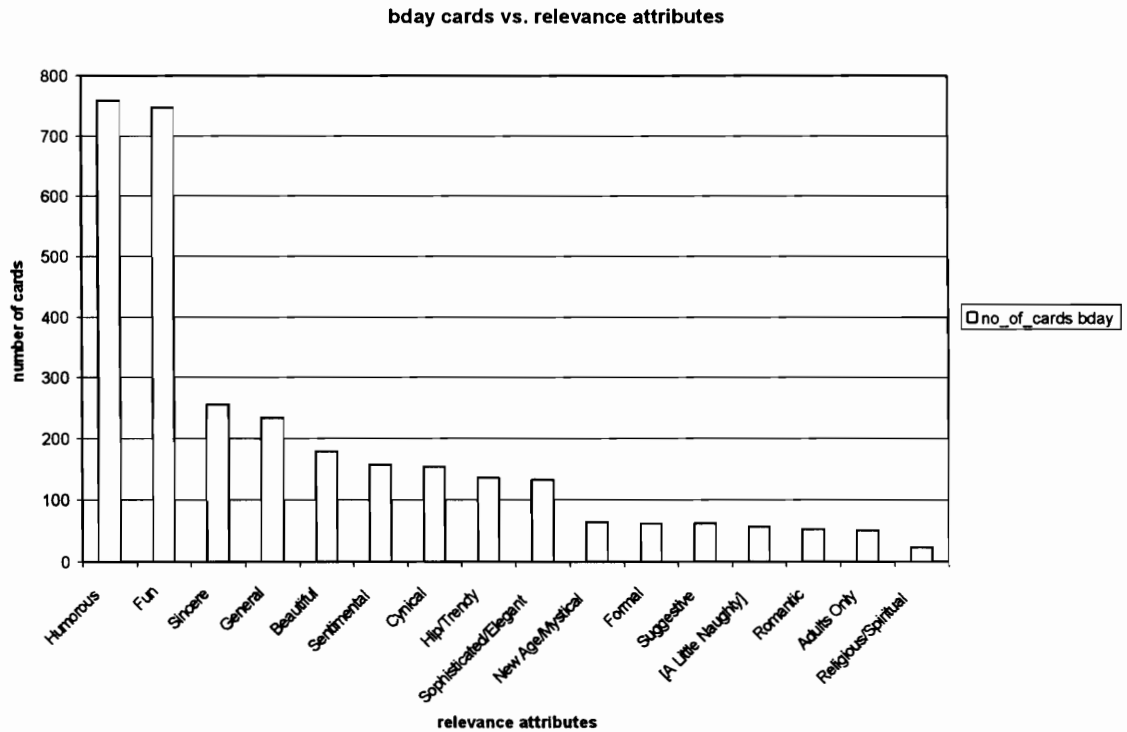
4.1.6 Distribution of Relevance Attributes of Birthday Cards

The same tests described in Section 4.1.5 were also performed on birthday cards for the same reasons. Table 4.1.6 and Figure 4.1.6 shows the data and histogram for the relevance attributes of birthday cards.

Table 4.1.6. Distribution of relevance attributes for birthday cards.

Relevance_attribute	no_of_cards
	bday
Humorous	759
Fun	747
Sincere	257
General	234
Beautiful	180
Sentimental	157
Cynical	154
Hip/Trendy	136
Sophisticated/Elegant	133
New Age/Mystical	65
Formal	62
Suggestive	62
[A Little Naughty]	57
Romantic	53
Adults Only	51
Religious/Spiritual	23

Figure 4.1.6. Distribution of relevance attributes for birthday cards.



Again, as shown in Figure 4.1.6, not many cards have relevance attribute ratings. But these again may hold good filtering potential when grouped.

4.1.7 Bag of Words Attribute

The statistics for the bag of words for Christmas and birthday cards was generated. Table 4.1.7 shows the overall performance of the system, and estimates the amount of time that it takes to generate the bag of word statistics based on their current database.

Table 4.1.8 and Table 4.1.9 show the average bag size for a given user with n orders for Christmas cards. We have included Table 4.1.8 and Table 4.1.9 because the numbers are

significantly higher in Table 4.1.8 versus Table 4.1.9. The large difference led us to infer that that Table 4.1.8 generates a bias towards certain cards. We therefore made all remaining calculations based on distinct cards bought. Table 4.1.10 contains the same information, but just for birthday cards.

Table 4.1.11, the numbers of cards bought versus average size of word bag, shows that as the number of orders increases, there is a large variance in the number of words matched for Christmas cards. This actually is not an error. We investigated, and found that the number of users who bought the maximum number of cards, the majority of them bought cards that had front text and back text missing. Our number of cards bought, therefore, did not factor in the people who bought X distinct cards.

They show distinct, but similar results. The similar results that the two sets have in common are: as the customer orders more and more cards, the bag size and the user profile both increase.

Graphs, as well as simple linear regression was performed to verify that there was positive slopes for Christmas and birthday cards with respect to bag of word size, and matches.

Table 4.1.7. Time statistics for bag of words.

Part of Algorithm	Order	Time for Xmas	Estimated Whole	Per Second
Insert Cards	Ca	00:00:19	00:03:00	27.7
Insert Customers	Cu	00:39:20	06:18:00	1.7
DoMatch	Ca*Cu	03:47:09	36:40:00	0.3
Total	Ca*Cu	04:26:48	43:00:00	

Ca = number of attributes, Cu = number of users

Table 4.1.8. Average bag size for each user based on the number of Christmas cards purchased.

Number of Cards	Christmas Average Bag Size
1	77
2	96
3	116
4	143
5	163
6	180
7	194
8	215
9	223
10	236
11	242
12	251
13	260
14	280
15	298
16	309
17	319
18	338
19	352

Note: Duplicate cards were allowed.

Table 4.1.9. Word profile size statistics for each specific number of distinct purchased Christmas cards.

Cards Purchased	Average size	Min size	Max size
1	22.0	6	62
2	44.8	14	87
3	64.3	26	118
4	87.9	26	118
5	114.4	68	186
6	132.3	68	186
7	161.2	114	228
8	176.1	143	246
9	192.9	142	251
10	226.5	191	271
11	257.3	221	297
12	259.3	222	288
13	291.0	271	315
14	N/A	N/A	N/A
15	371.3	342	396
16	N/A	N/A	N/A
17	367.0	353	381
18	N/A	N/A	N/A
19	N/A	N/A	N/A

Note: Columns denoted as N/A have no customers who had bought that number of distinct cards.

Table 4.1.10. Word profile size statistics for each specific number of distinct purchased birthday cards.

Cards Purchased	Average size	Min size	Max size
1	8.3	27	6
2	16.7	37	12
3	25.1	42	19
4	33.8	53	28
5	43.0	61	37
6	52.8	72	44
7	61.6	81	52
8	72.9	87	60
9	84.7	96	72
10	94.5	105	82
11	101.9	113	83
12	109.0	119	99
13	126.0	134	121
14	142.0	142	142
15	138.6	144	135
16	145.0	1445	145
17	156.0	156	156
18	165.5	167	164
19	175.7	180	173

Table 4.1.11. Bag of words average number of words matched for each number of distinct cards purchased.

Number of Cards	Christmas Average Matches	Birthday Average Matches
1	3	0
2	7	0
3	11	1
4	15	1
5	19	1
6	21	2
7	24	2
8	23	3
9	32	3
10	33	3
11	30	4
12	22	4
13	36	4
14	17	3
15	19	6
16	29	7
17	43	7
18	15	0
19	3	6

Table 4.1.12. Simple linear regression results for size of bag of words for Christmas cards.

Correlation coefficient: 0.73

Estimate of sigma: 1.60

Parameter	Estimate	Std.Err	DF	Tstat	Pval
Intercept	0.0175	0.7621	17	0.0230	0.491
Orders	0.2982	0.0668	17	4.4617	0.0002

Table 4.1.13. Simple linear regression for bag of words average number of word matched for Christmas cards.

Correlation coefficient: 0.35

Estimate of sigma: 10.60

Parameter	Estimate	Std.Err	DF	Tstat	Pval
Intercept	14.3508	5.0599	17	2.8361	0.0057
Orders	0.6807	0.4437	17	1.5338	0.0717

Table 4.1.14. Simple linear regression results for size of bag of words for birthday cards.

Correlation coefficient: 0.99

Estimate of sigma: 10.36

Parameter	Estimate	Std.Err	DF	Tstat	Pval
Intercept	82.6776	4.9455	17	16.7176	0
Orders	14.3761	0.4337	17	33.1436	0

Table 4.1.15. Simple linear regression for bag of words average number of word matched for birthday cards.

Correlation coefficient: 1.00 (See fitted line plot in Graphics Panel.)

Estimate of sigma: 3.70

Parameter	Estimate	Std.Err	DF	Tstat	Pval
Intercept	-2.4180	1.7651	17	-1.3699	0.0943
Orders	94.6887	0.1548	17	61.1646	0

Figure 4.1.7. Size of bag of words for distinct purchased Christmas cards vs. number of distinct orders.

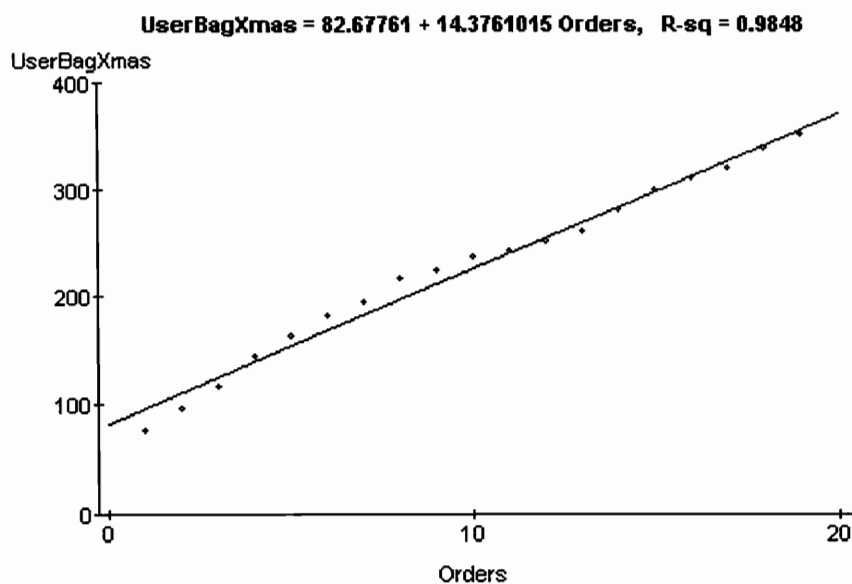


Figure 4.1.8. Size of bag of words for distinct purchased birthday cards vs. number of distinct orders.

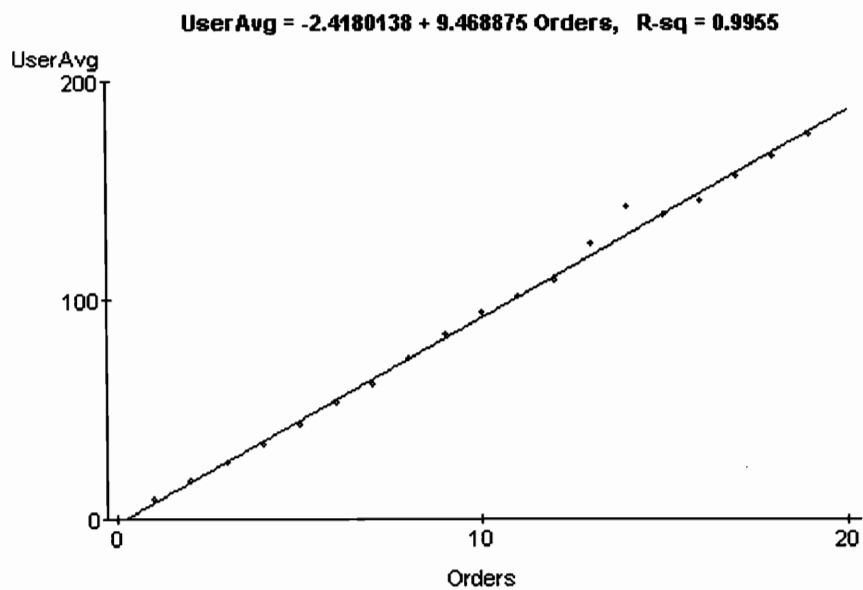
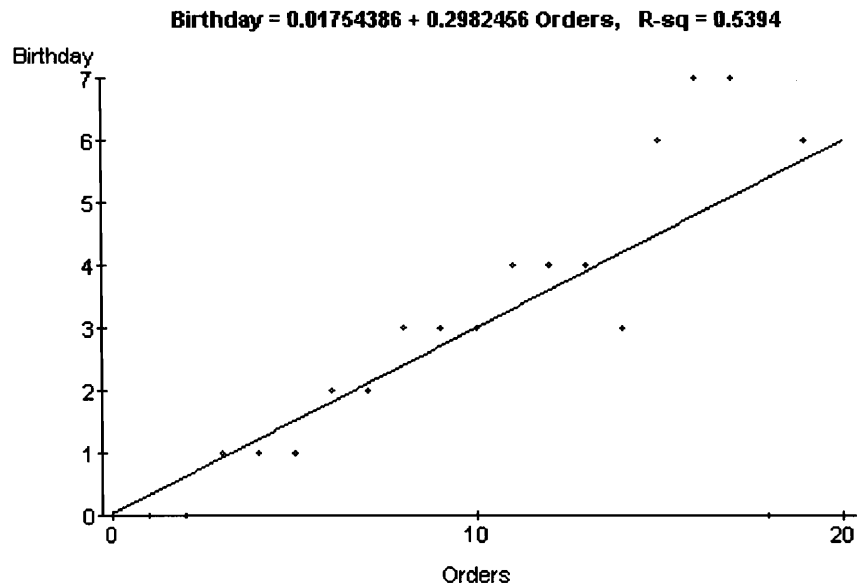


Figure 4.1.9. Average number of bag of word matches vs. number of Christmas card orders.



Figure 4.1.10. Average number of bag of word matches vs. number of birthday card orders.



4.2 Phase 1 of Testing: Content attributes

The graphs include a scatter plot and one to three box plots. The scatter plot is the distinct count of orders versus the content-based score (or collaborative total score) when considered separately. One box plot is the average placement of where the card would be shown if the attribute was only considered. (The average placement is computed by first finding out what bracket the card is in (the bracket has the same content-based or collaborative score), and then taking the first and last card that is in that bracket from highest to least.) For every group except price, we included a second box plot that contains the Mean, Median, and Mode of the size of the bracket. For every group except for the discrete attributes and price, we included another box plot, which is the total number of brackets, also defined as the number of discrete ratings, for that attribute.

We also include “general” statistics about the attribute. All statistics were computed by using the program “WebStat 2.0” which is available at the following URL:

<http://www.stat.sc.edu/webstat/version2.0/>

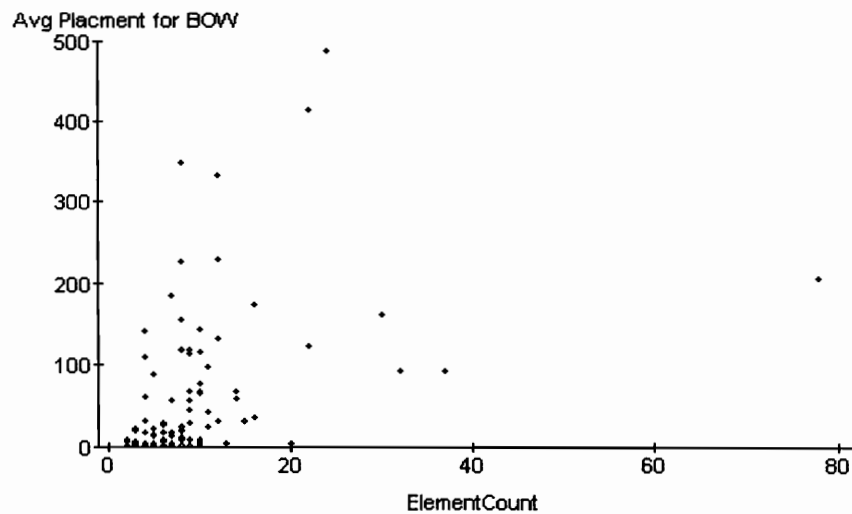
We decided not to include in the report some of the charts on discrete characteristics of the card because of similar results. The attributes not included were: PersonalizeID, Envelope_height, Envelope_width, Envelope_color, PaperID, MediumID, FoldID and LineID. Design ID was therefore used as the tool to describe the others.

After the results, which are collected in the tables and figures below, we performed the real test.

Table 4.2.1. Statistical summary for bag of words for Christmas cards.

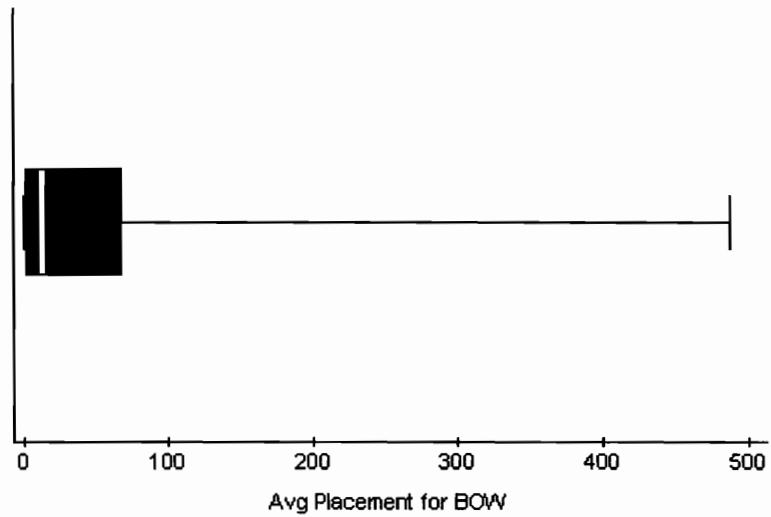
Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
BOW Avg Placement	111	52.783	7606.407	87.214	13	486.5	1	487.5	2	69
Brackets	111	70.051	1726.815	41.555	64	272	22	296	42	86
SizeAvg	111	9.770	24.206	4.919	8.218	22.132	1.777	23.909	6.116	12.523
Size Mode	111	1.990	2.681	1.637	1	8	1	9	1	2
Size Median	111	31.387	247.657	15.737	27	82	5	87	20	40

Figure 4.2.2. Scatter plot of average placement for bag of words for Christmas cards.



As seen in Figure 4.2.2, there is no direct correlation between the number of cards bought and the accuracy of the bag of word predictions.

Figure 4.2.3. Box plot of average placement for bag of words for Christmas cards.



As can be seen from Figure 4.2.3, the average placement rankings is relatively high.

Figure 4.2.4. Box plot for Christmas cards' bag of words size distribution.

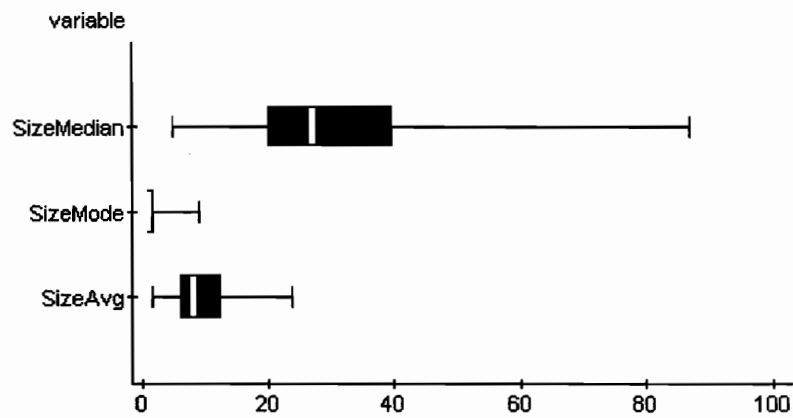
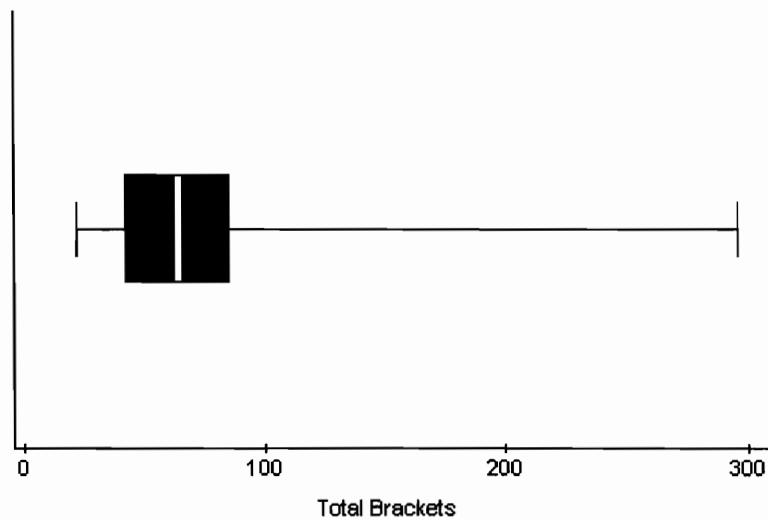


Figure 4.2.5. Box plot for total number of brackets for Christmas cards.



As can be seen from Figures 4.2.4, and 4.2.5, the number of brackets is pretty high, which implies that the predictions are well spread for all cards.

It seems that the bag of words is a very good indicator for Christmas cards because the number of brackets is relatively large, the size of a bracket is small, and the average placement is high.

Table 4.2.2. Christmas cards price attribute summary statistics.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
AvgPrice Avg Placement	111	233.873	16137.065	127.031	223.5	471	11.5	482.5	96	354

The total number of brackets was 34, size average was 15.4, the size mode was 1 and the size median was 7 for *all* customers.

Figure 4.2.6. Christmas cards price attribute scatter plot.

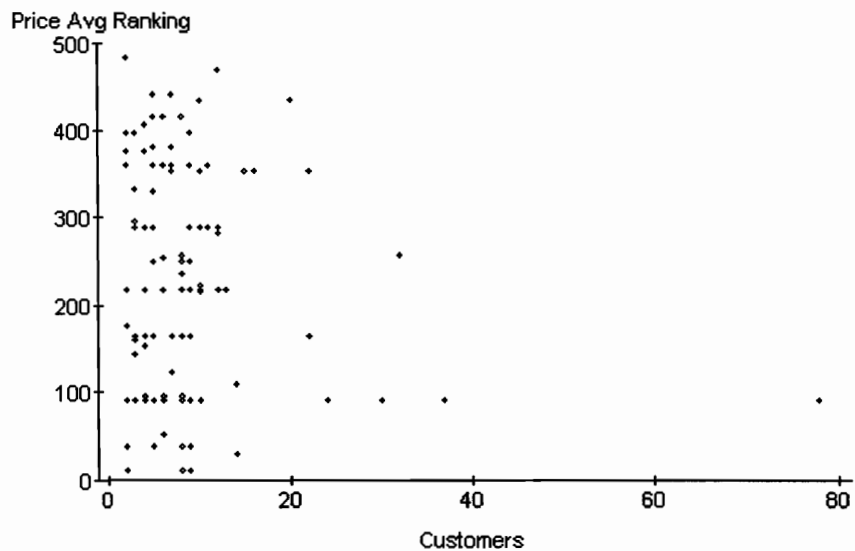
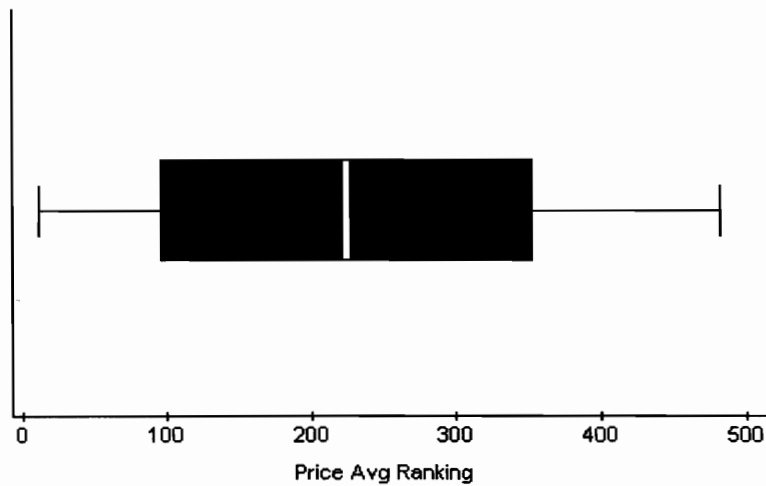


Figure 4.2.6 shows that there is no direct relationship between the number of cards bought and the average placement rankings for price.

Figure 4.2.7. Christmas cards price attribute box plot.

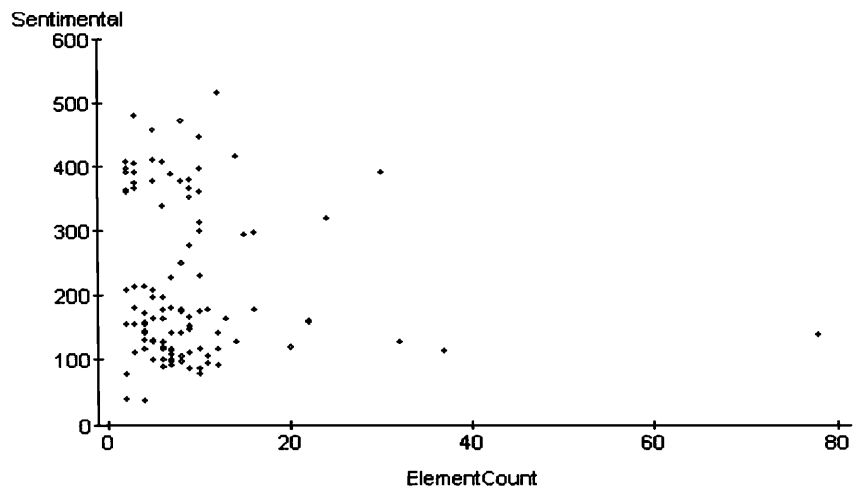


According to Figure 4.2.7, it seems that price is a good indicator for Christmas cards because there were a large number of brackets, and the size of the bracket is relatively small, and the average placement, while a little high, seems promising.

Table 4.2.3. Christmas cards grouped sentiments attributes summary statistics.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Senti Avg Placment	111	215.977	14893.052	122.037	165.5	497	38	517	118.5	352
Brackets	111	17.027	82.662	9.091	14	36	5	41	10	22
SizeAvg	111	41.477	568.267	23.838	37.571	92.370	12.829	105.2	23.909	52.6
SizeMode	111	112.846	11732.585	108.317	61	351	1	352	20	210
SizeMedian	111	133.301	9292.592	96.398	94	339	13	352	49	210

Figure 4.2.8. Christmas cards grouped sentiments scatter plot.



As seen in Figure 4.2.8, there is no direct relationship between the number of cards that customers buy and their sentiments placement rating.

Figure 4.2.9. Christmas cards grouped sentiments box plot.

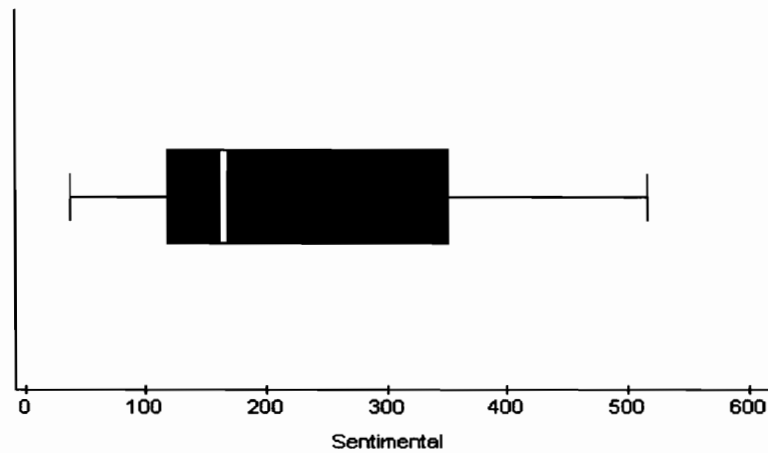
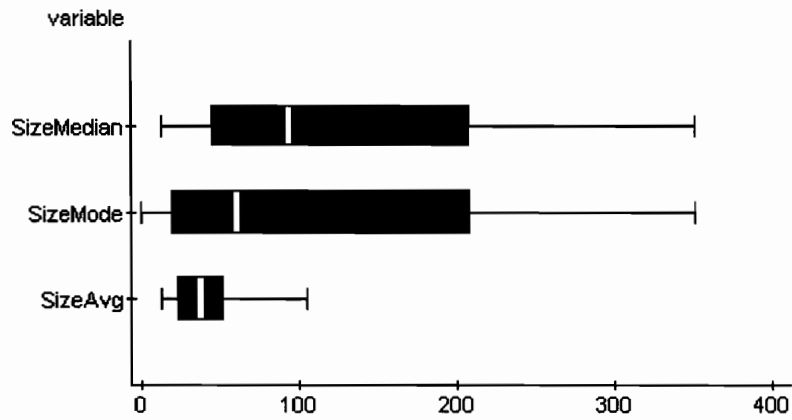
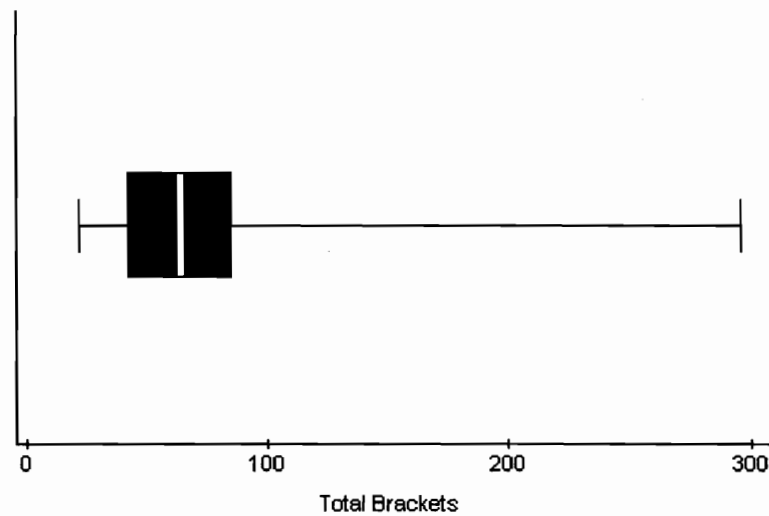


Figure 4.2.10. Christmas cards grouped sentiments box plot for bracket size distribution.



As seen in Figures 4.2.9 and 4.2.10, the average size of brackets is small and the number of brackets is relatively high.

Figure 4.2.11. Christmas cards sentiments box plot for total number of brackets.

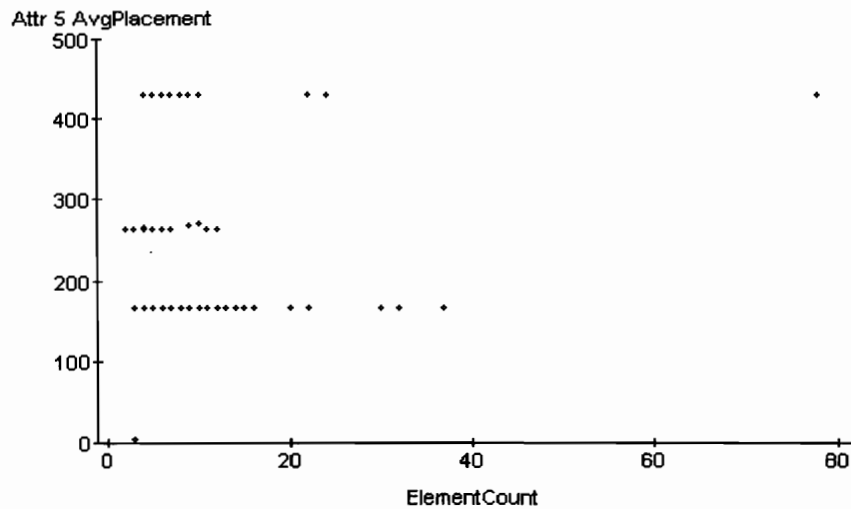


Accoding to Figure 4.2.11, it seems that sentiments is a good indicator for Christmas cards because there were a large number of brackets, and the size of the bracket is relatively small, and the average placement, while a little high, seems promising, more so than price.

Table 4.2.4. Christmas card designer ID summary statistics (attribute 5).

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Designer ID Avg Placement	111	234	9524.196	97.591	166	425.5	3.5	429	166	263.5
Brackets	111	1.747	0190	0.436	2	1	1	2	1	2
SizeAvg	111	329.342	13165.336	114.740	263	263	263	526	263	526
SizeMode	111	290.072	22520.723	150.069	195	331	195	526	195	526
SizeMedian	111	338.468	14192.143	119.130	263	263	263	526	263	526

Figure 4.2.12. Christmas card designer ID scatter plot.



As seen in Figure 4.2.12, there is no evident relationship between the number of cards purchased and the accuracy of placement predictions.

Figure 4.2.13. Christmas card designerID box plot.

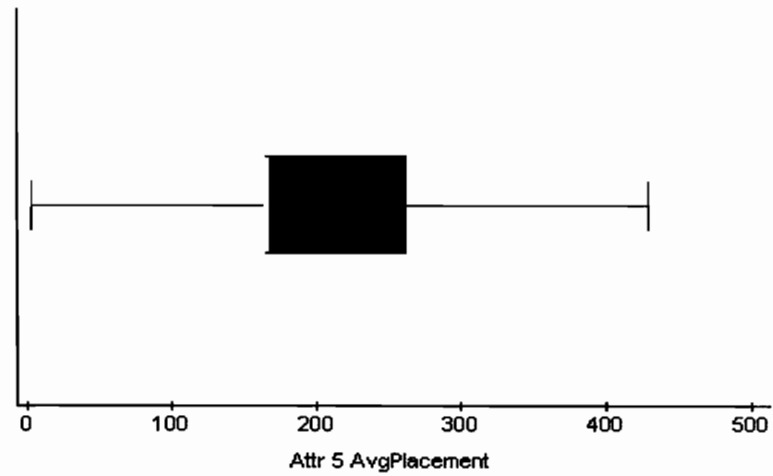
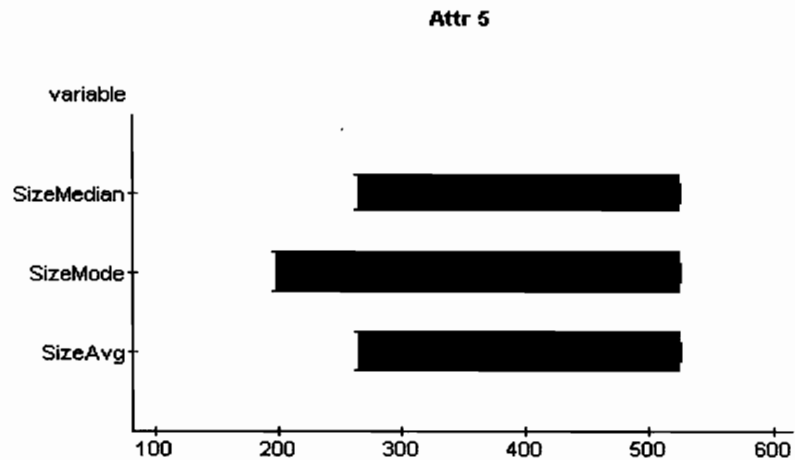


Figure 4.2.14. Christmas card designer ID distribution for size of bracket.

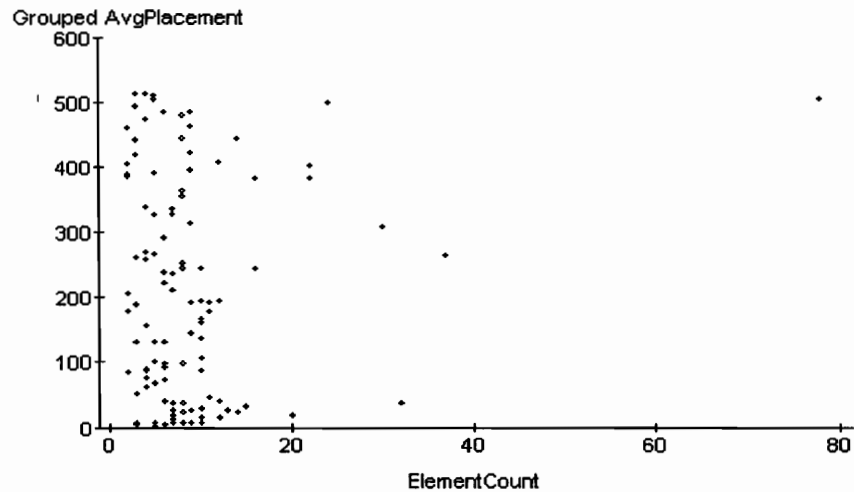


According to Figures 4.2.13 and 4.2.14, it seems that Designer ID (and other individual attributes) is not a good indicator for Christmas cards by itself. Rather, there were only two brackets, and the size of the bracket is large, and the average placement was high.

Table 4.2.5. Christmas cards grouped discrete characteristics summary statistics.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Grouped Avg Placement	111	211.707	27910.602	167.064	191.5	511.5	2.5	514	47	383
Brackets	111	178.342	114.863	33.835	193	122	87	209	177	199
SizeAvg	111	3.128	0.946	0.972	2.725	3.529	2.516	6.045	2.643	2.971
SizeMode	111	1.090	0.137	0370	1	2	1	3	1	1
SizeMedian	111	29.603	209.677	14.480	28	125	14	139	20	35

Figure 4.2.15. Christmas card grouped discrete characteristics scatter plot.



As shown in Figure 4.2.15, there seems to be no evident relationship between the number of cards purchased and the grouped discrete card characteristics, as seen from the graph above.

Figure 4.2.16. Christmas cards grouped discrete characteristics box plot.

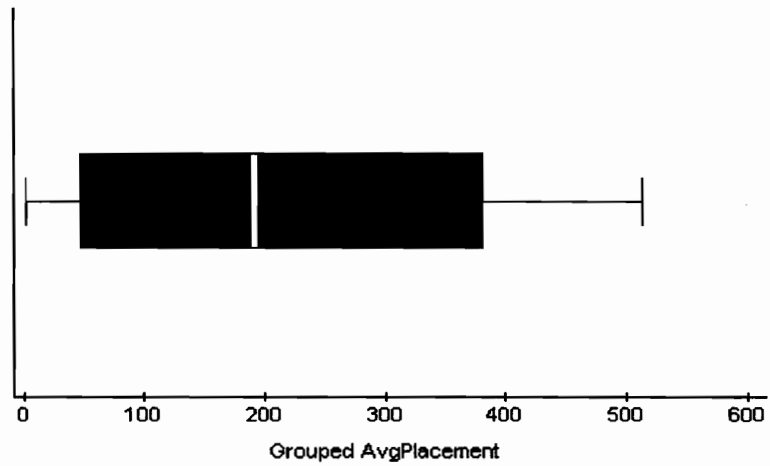


Figure 4.2.17. Christmas cards grouped distinct characteristics distribution for size of bracket.

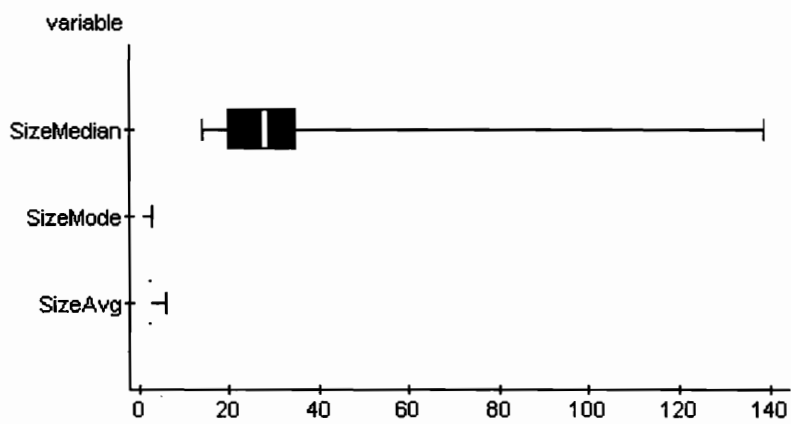
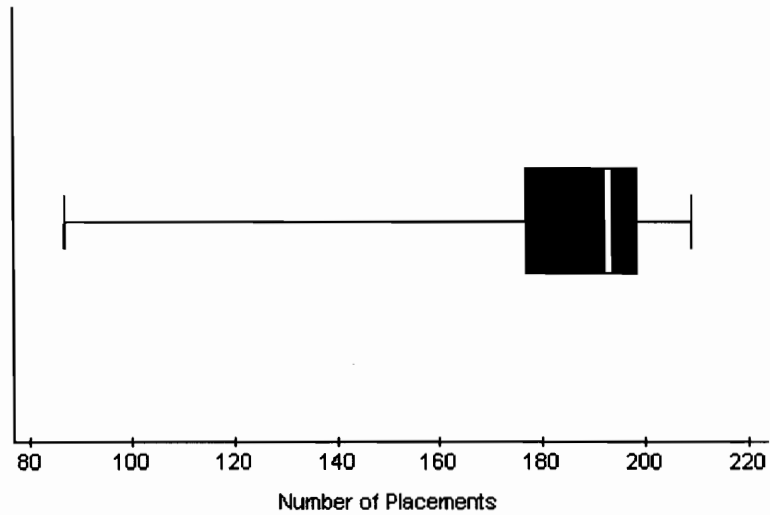


Figure 4.2.18. Christmas cards grouped distinct characteristics total number of brackets.

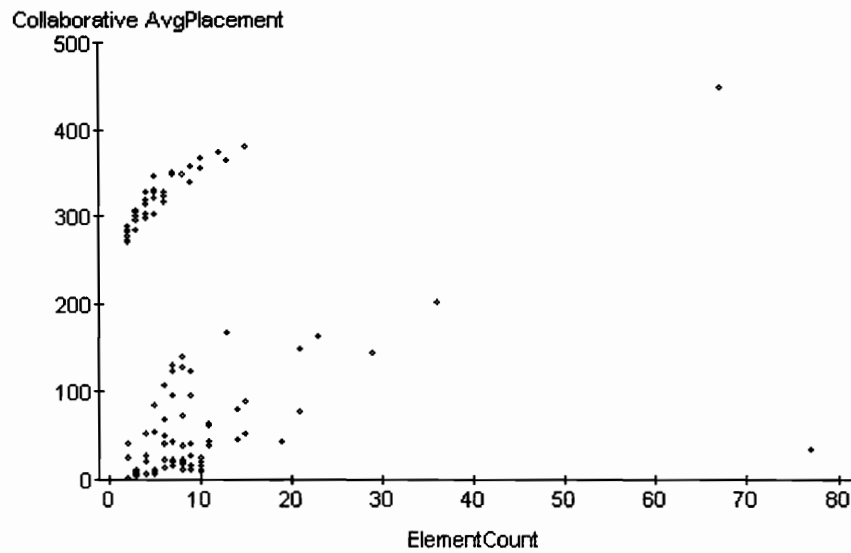


According to Figures 4.2.16, 17, and 18, it seems that the grouped attributes are a good indicator for Christmas cards because there are generally a large number of brackets, and the size of the bracket is relatively small, and the average placement, while a little high, seems promising.

Table 4.2.6. Christmas cards collaborative summary statistics.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Collaborative Avg Placement	108	151.782	19636.51	139.152	86.25	447	2	449	22.25	302.75
Brackets	108	13.342	59.386	7.706	12.5	57	3	60	90	15.5
SizeAvg	108	51.7903	937.445	30.617	42.147	166.566	5.766	175.333	33.97	58.444
SizeMode	108	373.092	6544.982	80.901	373.5	401	112	513	330	432
SizeMedian	108	373.092	6544.282	80.901	373.5	401	112	513	530	432

Figure 4.2.19. Christmas cards collaborative scatter plot.



As shown in Figure 4.2.19, further analysis needs to be done to determine a definite relationship between the number of cards purchased and the collaborative predictions.

Figure 4.2.20. Christmas cards collaborative box plot.

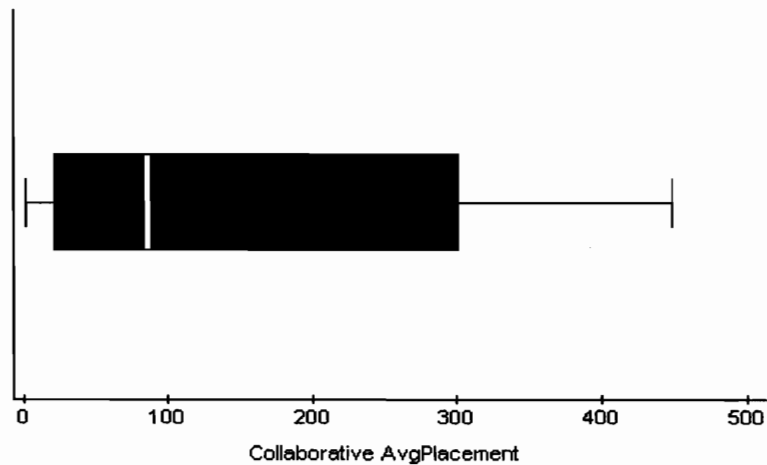
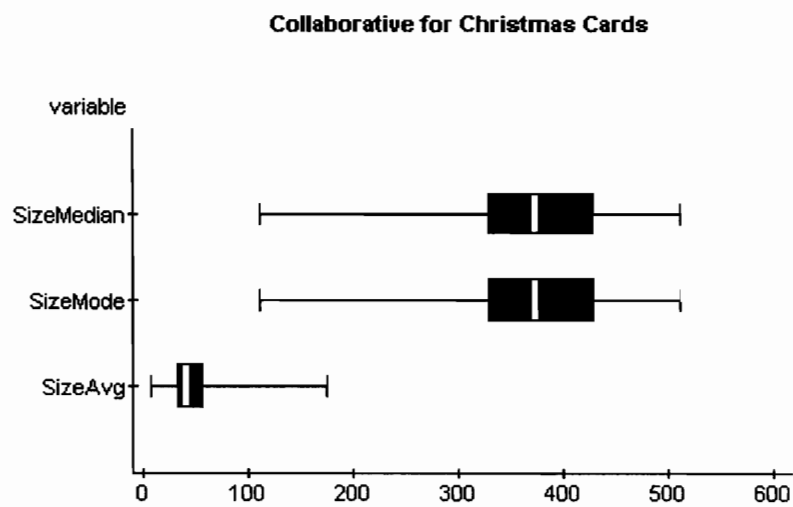
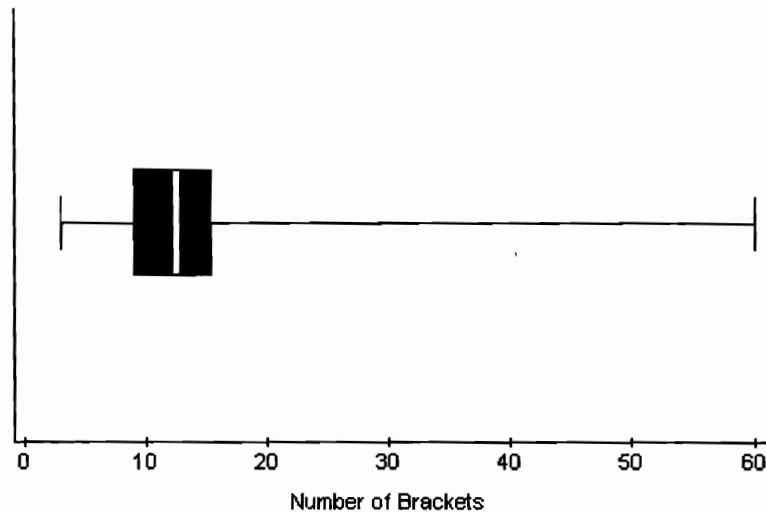


Figure 4.2.21. Christmas cards collaborative distribution for size of bracket.



As shown in Figure 4.2.20 and 4.2.21, the size of brackets is small and the number of brackets is pretty high.

Figure 4.2.22. Christmas cards collaborative total number of brackets.

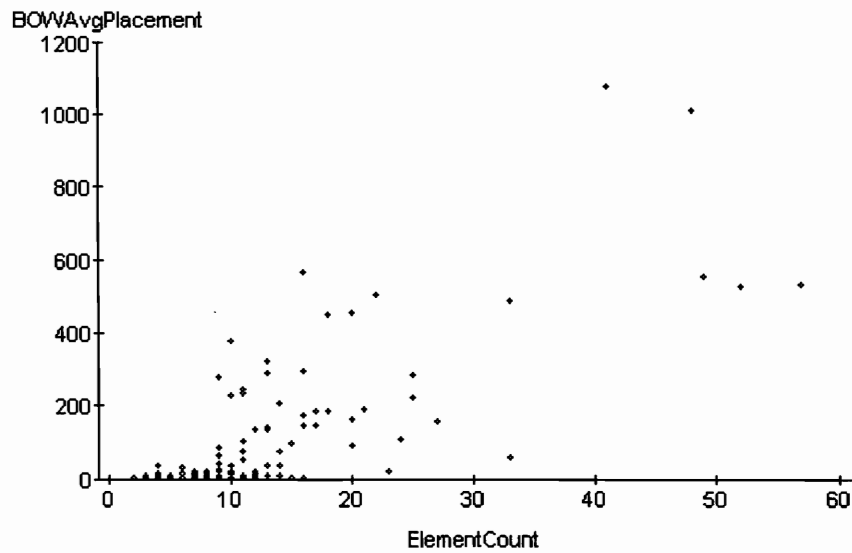


According to Figure 4.2.22, it seems that the collaborative predictions are a very good indicator for Christmas cards because the number of brackets is large, and the size of the bracket is relatively small, and the average placement is excellent for the majority of the users. The “upper cluster” which is shown in the scatter diagram when a user has a few orders needs to be analyzed further.

Table 4.2.7. Birthday bag of words summary statistics.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
BOW Avg Placement	148	85.010	30058.46	173.373	7.5	1076.5	1	1077.5	2.5	80.75
Brackets	148	20.047	130.916	11.441	16	57	7	64	13	23
SizeAvg	148	345.783	1637.077	40.460	95.812	195.046	23.953	219	66.65	177.92
Size Mode	148	345.783	44229.965	210.309	294	1123	30	1153	205	444.5
Size Median	148	394.885	39734.43	199.334	352.5	1077	76	1153	262	505.5

Figure 4.2.23. Birthday bag of words scatter plot.



As shown in Figure 4.2.23, it seems that as the number of cards bought increases, the placement predictions become less accurate.

Figure 4.2.24. Birthday bag of words box plot.

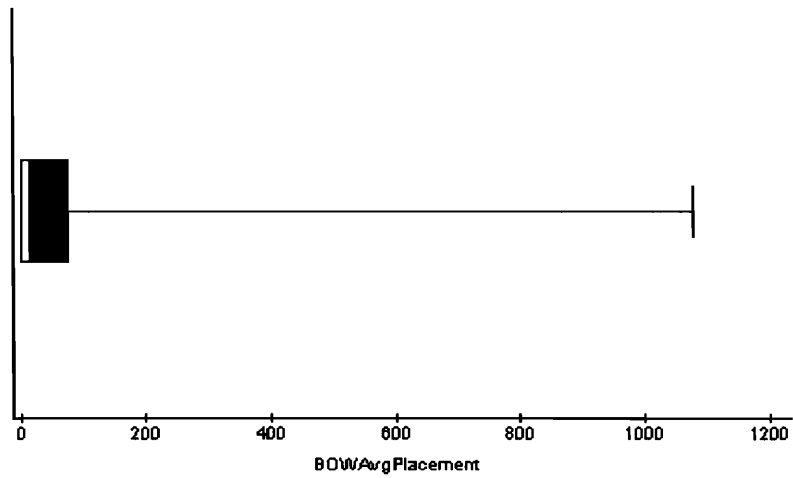


Figure 4.2.25. Birthday bag of words distribution for bracket size.

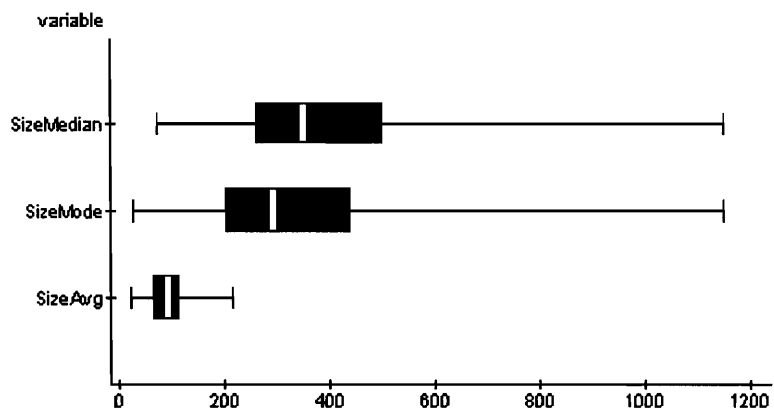
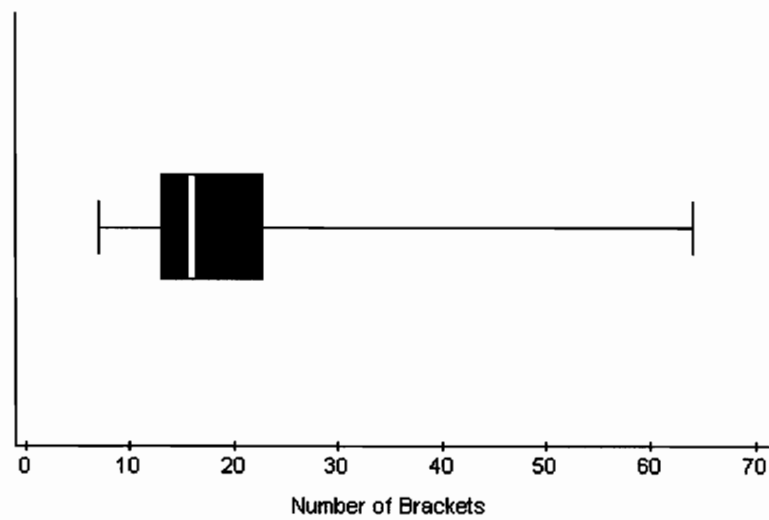


Figure 4.2.26. Birthday bag of words total number of brackets.



Based on Figures 4.2.24, 25, and 26, we think that bag of words is a very good indicator for birthday cards because the number of brackets is relatively large, the size of a bracket is small, and the average placement is high.

Table 4.2.8. Birthday price attribute summary statistics.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Price Avg Placement	148	563.811	187173.34	432.635	493	1522	2.5	1524.5	187.5	947

The total number of brackets was 46, size average was 33.32088, the size mode was 2 and the size median was 20 for *all* customers.

Figure 4.2.27. Birthday price attribute scatter plot.

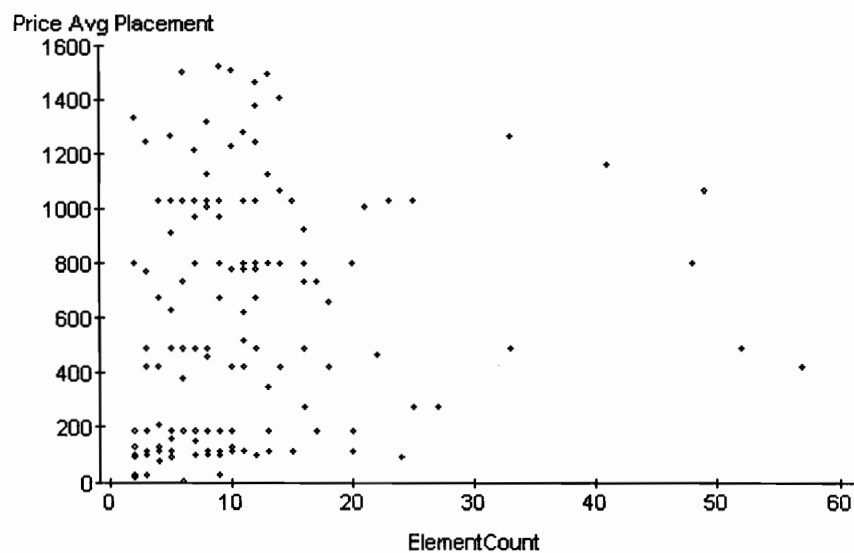
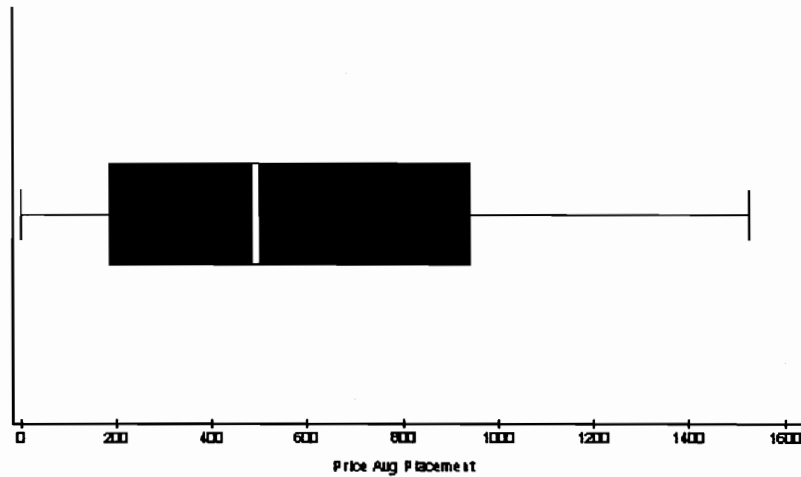


Figure 4.2.28. Birthday price attribute box plot.



According to Figures 4.2.27 and 4.2.28, it seems that price is poor indicator for birthday cards because the number of brackets is relatively large, the size of a bracket is small, and the average placement is high.

The following three tests (4.1.9, 4.1.10 and 4.1.11) use a slightly different data set for birthdays when compared to the rest of the database. The original WPI database was accidentally deleted and could not be recreated. Sentiments and DesignerID test contained the same information for some reason when we originally created the graphs, and went unnoticed until after WPI database was deleted. We therefore reran the tests on this new database. Since DesignerID was compared to Grouped Attributes, we also reran grouped attributes to ensure we had the correct information.

Table 4.2.9. Birthday sentiment summary statistics.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	119	630.226	101708	318.918	450	1122	143	1265	390	932
Brackets	119	20.571	124.823	11.172	19	55	4	59	12	26
SizeAvg	119	88.001	3677.345	60.641	68.736	304.36	22.13	326.5	50.23	108.83
SizeMode	119	226.159	41352	203.353	164	1131	13	1144	89	322
SizeMedian	119	224.756	37063	192.518	208	1101	43	1144	110	332

Figure 4.2.29. Birthday sentiment scatter plot.

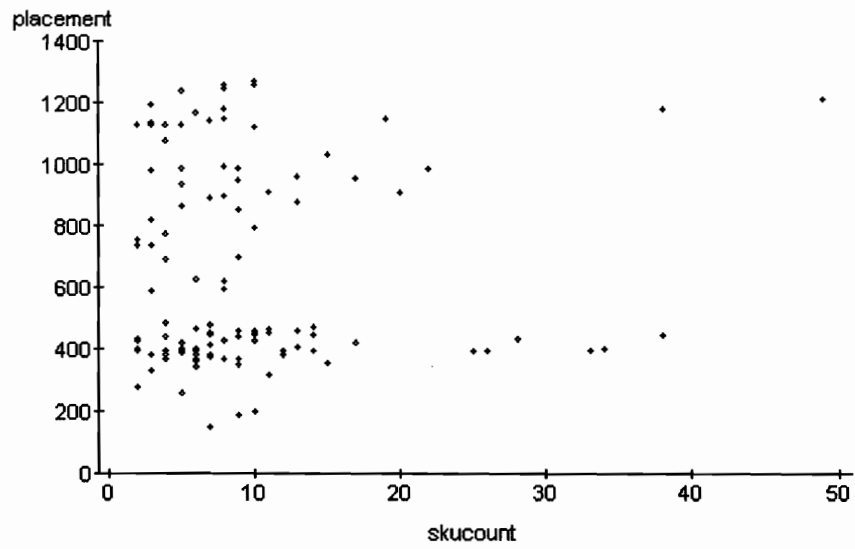


Figure 4.2.30. Birthday sentiment box plot.

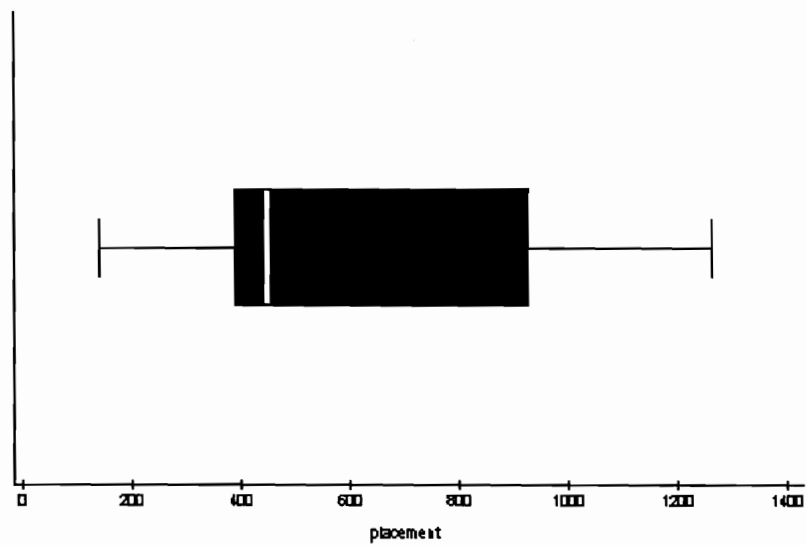


Figure 4.2.31. Birthday sentiment distribution for bracket size.

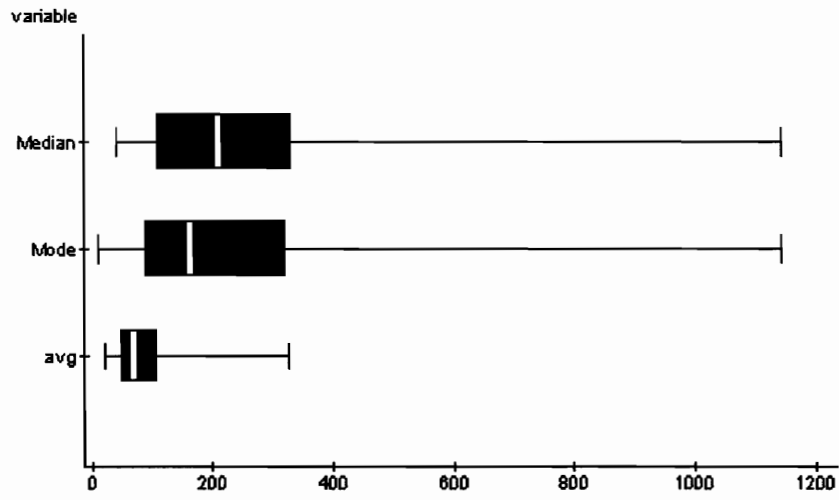
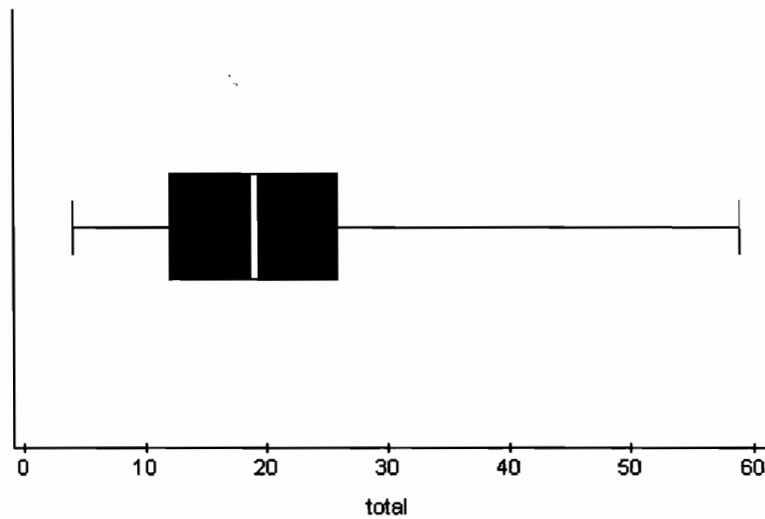


Figure 4.2.32. Birthday sentiment total number of brackets.



As shown in Figures 4.2.29 through 4.3.32, while the total number of brackets is large, and the size of the bracket is relatively small, the grouped attributes perform rather poorly, with an average placement of 630 out of 1,360.

Table 4.2.10. Birthday cards designer ID (attribute 5) for birthday cards.

Variable	N	Mean	Variance	Std.Dev.	Median	Range	Min	Max	Q1	Q3
DesginerID Avg Placement	180	545.197	78663.79	280.470	364	1004	13	1017	364	685.5
Brackets	180	2	0	0	2	0	2	2	2	2
Size Avg	180	653	0	0	653	0	653	653	653	653
Size Mode	180	673.705	55982.53	236.606	579	719	579	1298	579	579
Size Median	180	737.427	44503.99	210.959	653	645	653	1298	653	653

Figure 4.2.33. Birthday card designer ID (attribute 5) scatter plot.

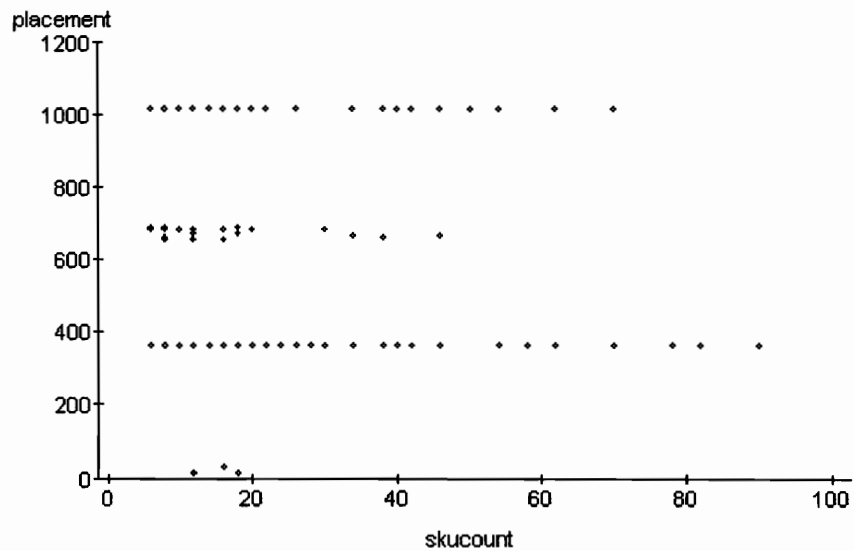


Figure 4.2.34. Designer ID box plot for birthday.

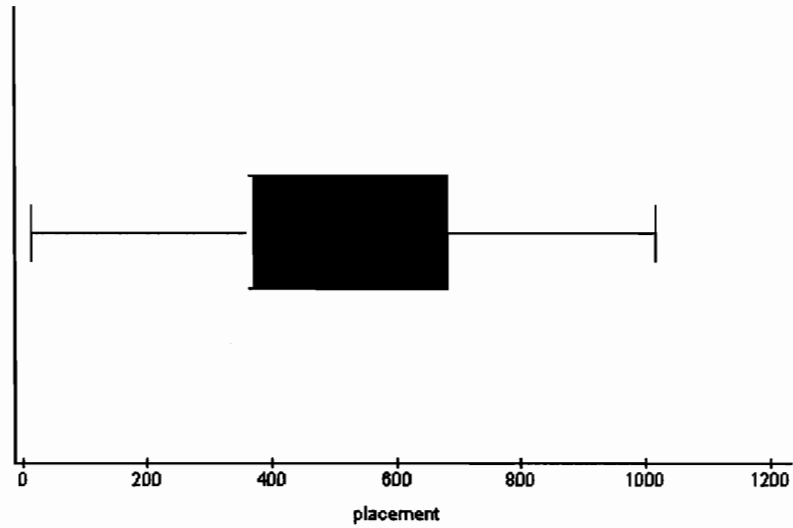
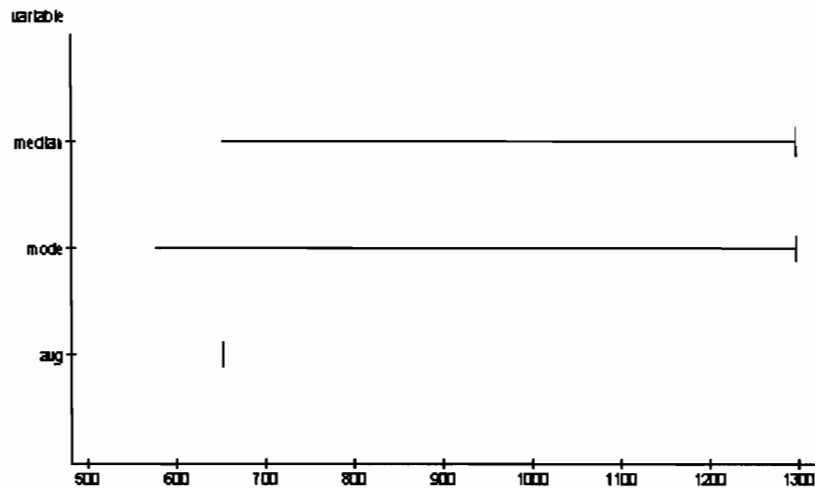


Figure 4.2.35. Designer ID distribution for birthday for bracket sizes.



While the average size is high, Designer ID and other distinct attributes by themselves are inconclusive indicators because the average placement is high. (Out of 1,306 cards, the average placement was 545.)

Table 4.2.11. Birthday grouped attributes summary statistics.

Variable	N	Mean	Variance	Std.Dev	Median	Range	Min	Max	Q1	Q3
Grouped AvgPlacement	148	532.783	188602.53	434.283	411.75	1520.5	1.5	1522	156.25	881.25
Brackets	148	466.628	1262.670	35.534	478	161	341	502	460	488
SizeAvg	148	3.308	0.092	0.304	3.207	1.441	3.053	4.495	3.143	3.332
SizeMode	148	1.020	0.19	0.141	1	1	1	2	1	1
SizeMedian	148	83.68	698.371	26.426	84.5	187	30	217	65	92.5

Figure 4.2.36. Birthday grouped distinct characteristics scatter plot.

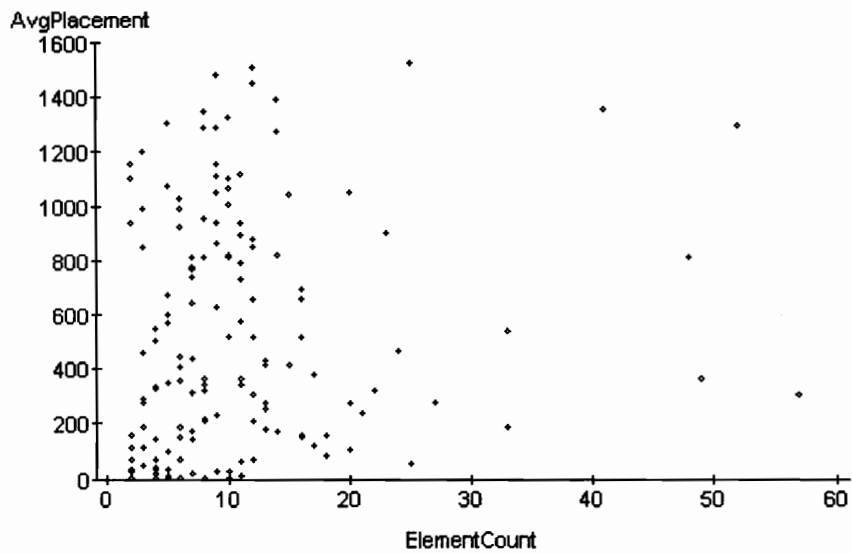


Figure 4.2.37. Birthday distinct characteristics grouped box plot.

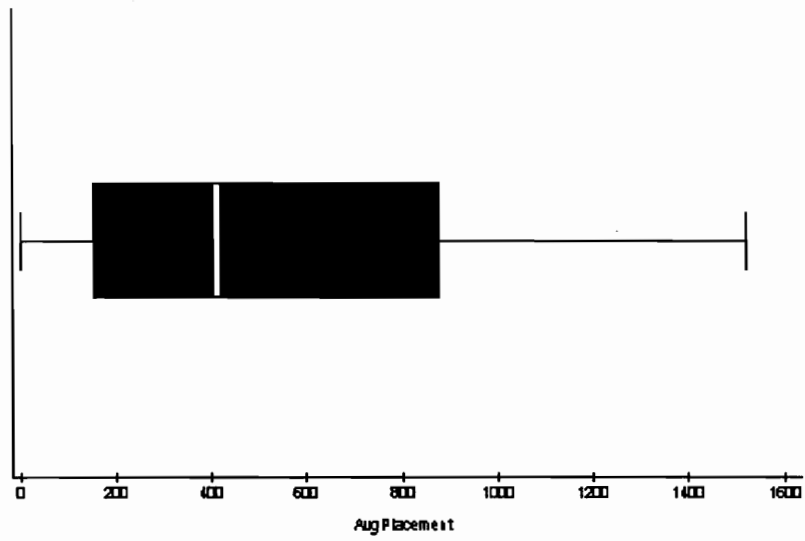


Figure 4.2.38. Birthday grouped distinct characteristics distribution for bracket size.

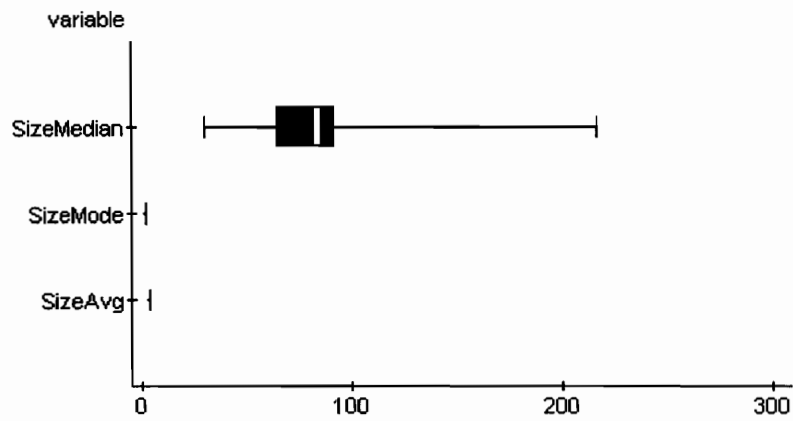
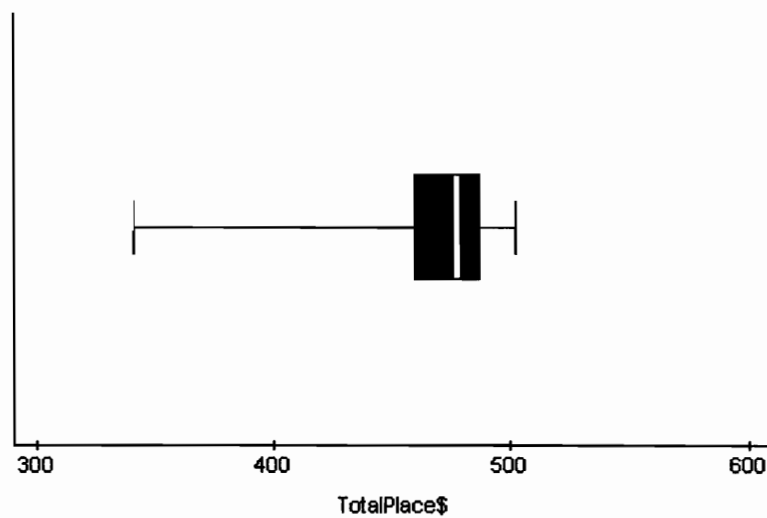


Figure 4.2.39. Birthday grouped distinct characteristics total number of brackets.

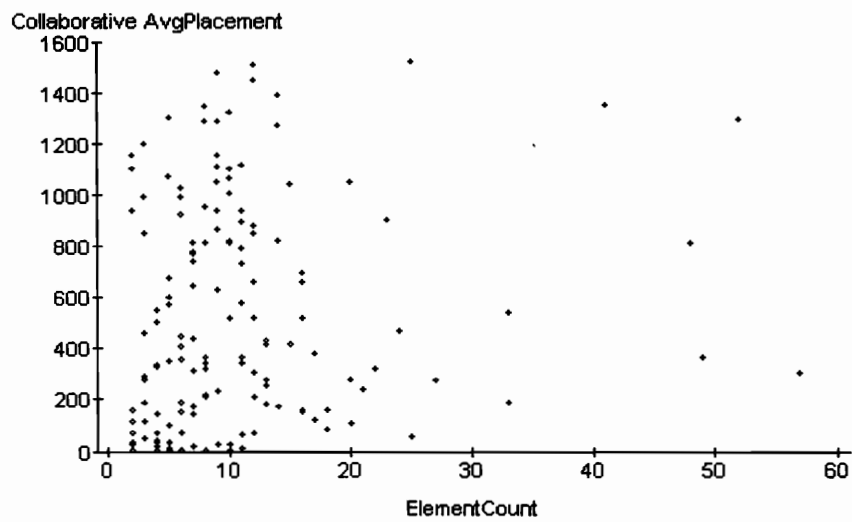


As shown in Figures 4.2.36 through 4.2.39, we think that the grouped characteristics are a fair indicator for Birthday cards. While the number and size of brackets are acceptable, the average placement is relatively poor.

Table 4.2.12. Birthday collaborative summary statistics.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	148	532.783	188602.53	434.283	411.75	1520.5	1.5	1522	156.25	881.25
Brackets	148	466.628	1262.670	35.534	478	161	341	502	460	488
SizeAvg	148	3.3082	0.092	0.304	3.207	1.441	3.053	4.495	3.143	3.332
SizeMode	148	1.202	0.019	0.141	1	1	1	2	1	1
SizeMedian	148	83.685	698.371	26.426	84.5	187	30	217	65	92.5

Figure 4.2.41. Birthday collaborative scatter plot.



As seen in Figure 4.2.41, there is no direct relationship between the number of cards purchased and collaborative placement predictions.

Figure 4.2.42. Birthday collaborative box plot.

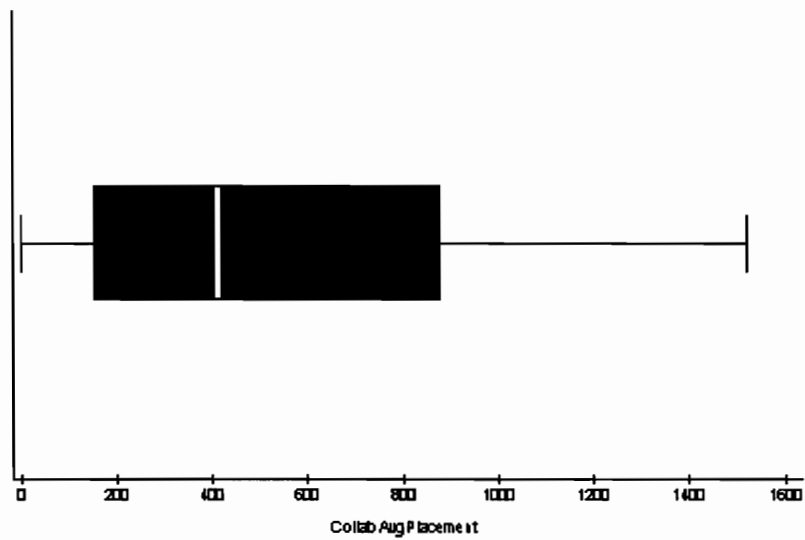


Figure 4.2.43. Birthday collaborative distribution for bracket size.

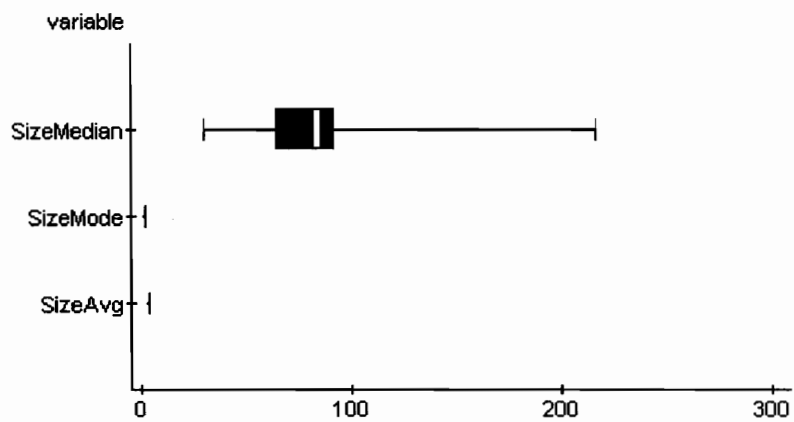
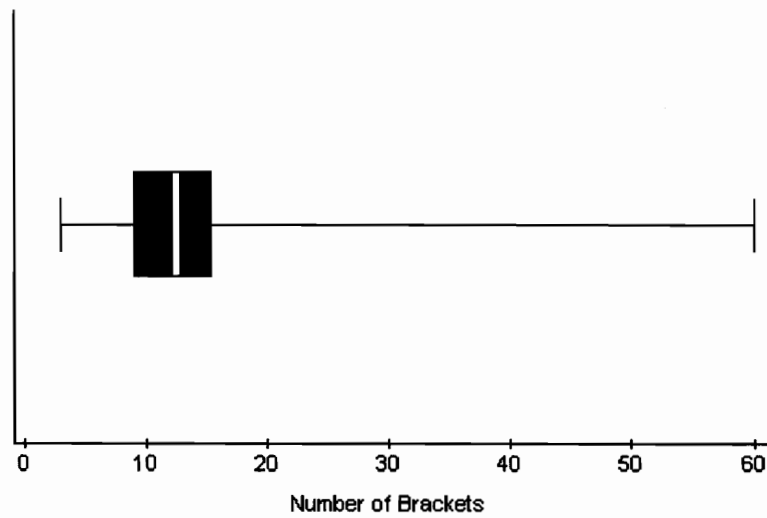


Figure 4.2.44. Collaborative total number of brackets.



As shown in Figures 4.2.42, 43, and 44, it seems that the collaborative predictions are a very poor indicator for birthday cards because while the number of brackets is relatively large, and the size of the bracket is small, the main disadvantage is that the average placement is extremely poor for the majority of the users.

4.3 Phase 2 of Testing: Content-based filtering Algorithm

From the results seen in Section 4.2, the heaviest weight for Christmas cards will be given to the bag of words, followed by sentiments, price and then the grouped distinct characteristics. The total weight for bag of words was set to .7, sentiments set to .1, price set to .1 and grouped distinct characteristics set to .1.

For these tests, the most important statistic is the *average placement*. This is the statistic that tells on average where the card removed will be located in a sorted list. Therefore, the smaller the average placement, the more accurate the content-based system is.

Table 4.3.1. Christmas content-based score with weights set to 1.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	113	176.769	26527.496	162.872	124	522	1	523	28	285
Brackets	113	501.415	1328.138	36.443	515	171	355	526	504	520
SizeAvg	113	1.055	0.009	0.095	1.021	0.481	1	1.481	1.011	1.043
SizeMode	113	1	0.017	0.133	1	2	0	2	1	1
SizeMedian	113	2.831	2.266	1.505	2	9	1	10	2	3

Figure 4.3.1. Scatterplot for christmas content-based score with weights set to 1.

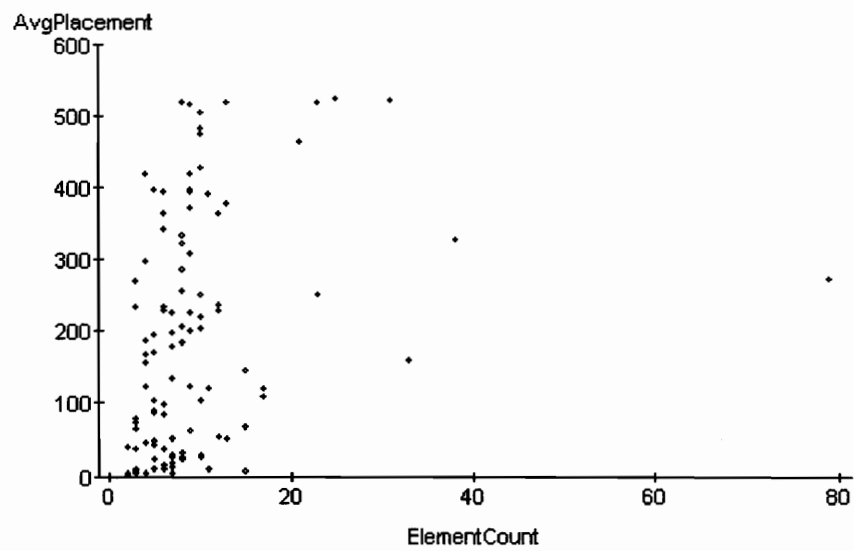


Figure 4.3.2. Box plot for Christmas content-based score with weights set to 1.

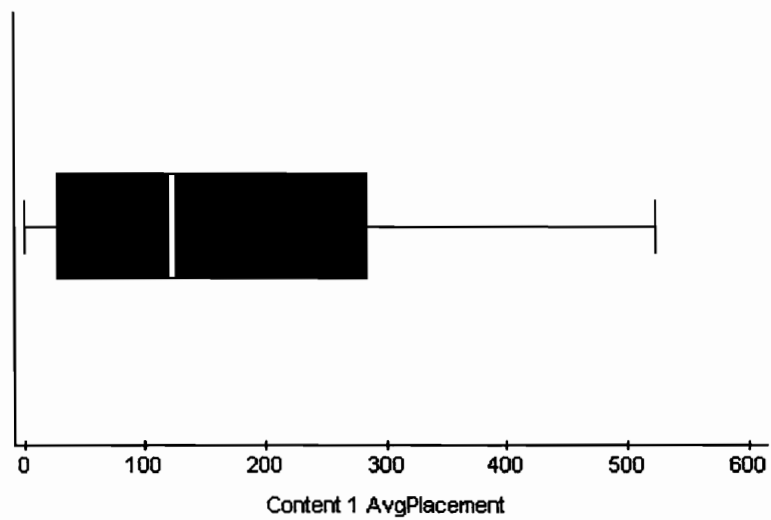


Figure 4.3.3. Distribution for Christmas content-based score with weights set to 1 for bracket size.

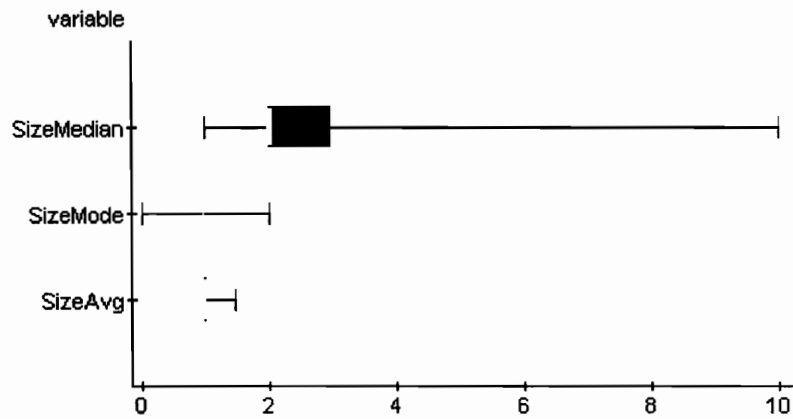
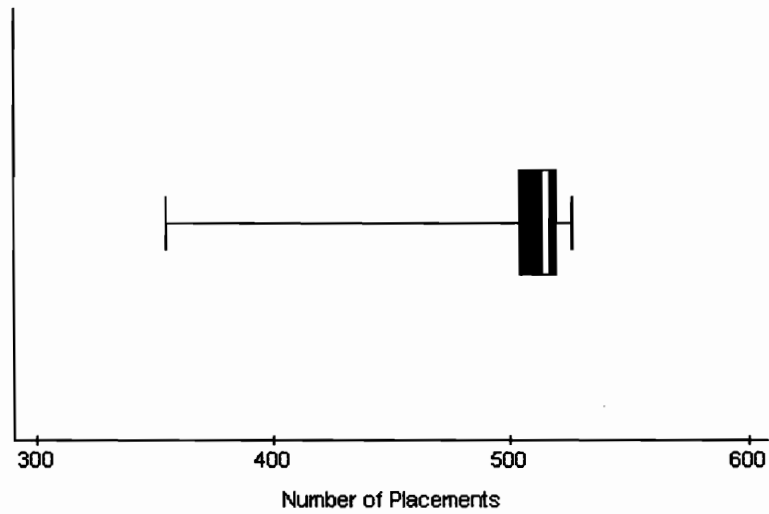


Figure 4.3.4. Box plot for Christmas content-based weights at 1 for number of brackets.



As shown in Figures 4.3.1 through 4.3.4, the average placement is high, around 170.

Additionally, the number of brackets was also large, meaning that there was good distinction between predictions.

Table 4.3.2. Christmas content-based score with optimized weights.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	113	56.690	8005.149	89.471	22	475	1	476	5	64
Brackets	113	501.725	1337.075	36.566	515	171	355	526	503	521
SizeAvg	113	1.055	0.009	0.095	1.021	0.481	1	1.481	1.00	1.04
SizeMode	113	1.061	0.183	0.428	1	4	0	4	1	1
SizeMedian	113	3.230	6.321	2.514	3	19	1	20	2	3

Figure 4.3.5. Scatterplot for Christmas content-based score with optimized weights.

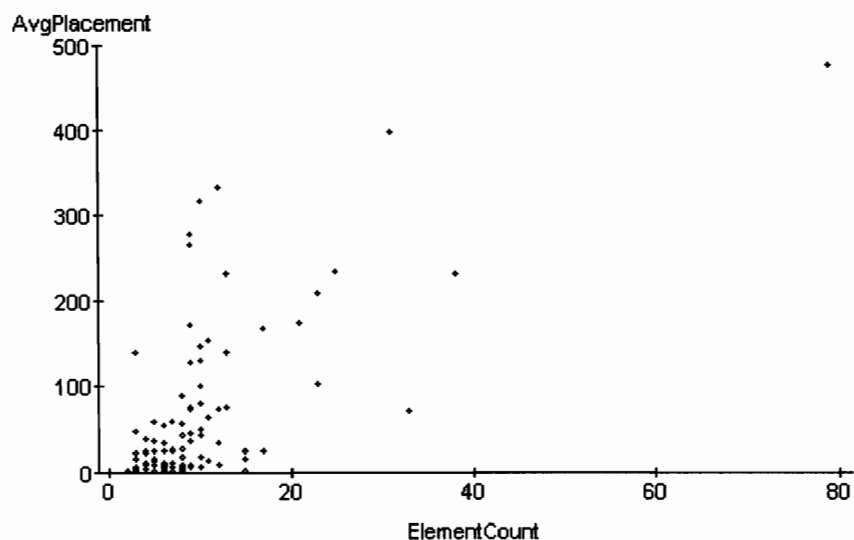


Figure 4.3.6. Box plot for Christmas content-based score with optimized weights.

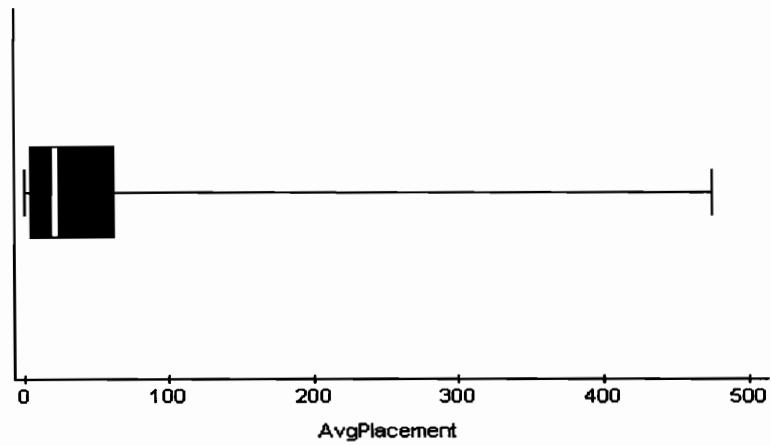


Figure 4.3.7. Distribution for Christmas content-based score with optimized weights.

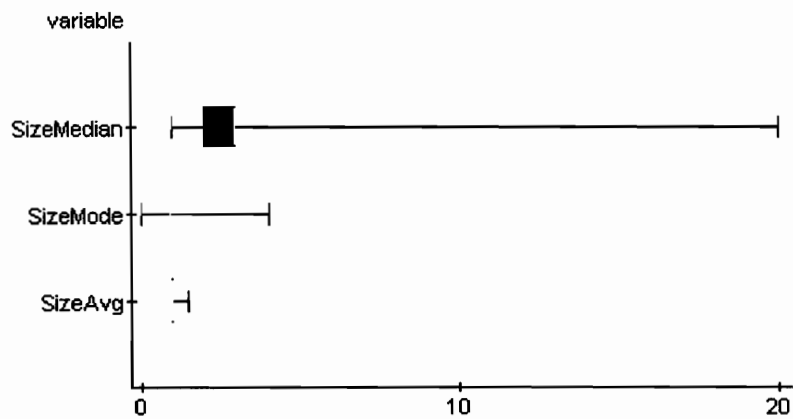
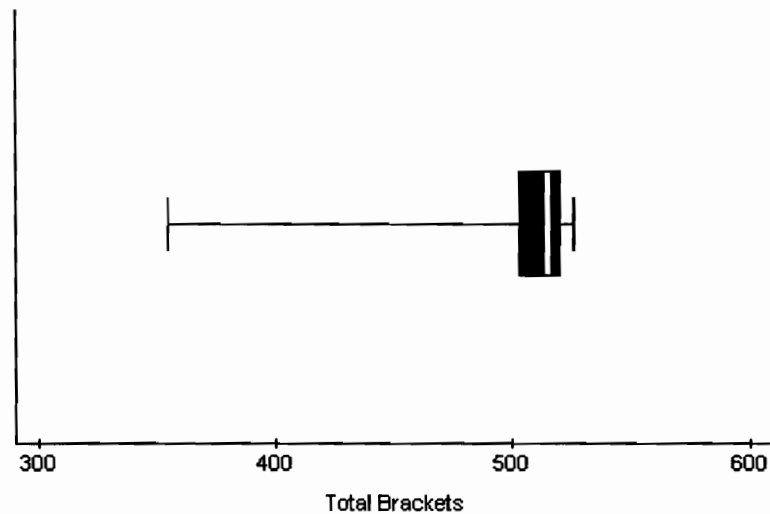


Figure 4.3.8. Box plot for Christmas content-based score with optimized weights for number of brackets.



As shown in Figures 4.3.5 through 4.3.8, the hypothesis about the assigned weights to the different attributes has been confirmed because the average placement has gone from an average of 176 to 56!

From the results obtained for birthday cards, heaviest weight will be given to the bag of words. Price, sentiments, or grouped characteristics do not weigh heavily in birthday cards. Hence, the total weight for bag of words will be set equal to .85, sentiments set to .05, price set to .05 and grouped distinct characteristics set to .05.

Table 4.3.3. Birthday content-based score with weights set to 1.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	148	345.101	143941.95	379.39682	234.5	1521	1	1513	36	530.5
Brackets	148	1305.189	11028.223	105.8689	1335	481	977	1478	1251.5	530.5
SizeAvg	148	1.183	0.011	0.105	1.148	0.500	1.037	1.453	1.114	1.224
SizeMode	148	1.020	0.019	0.141	1	1	1	2	1	1
SizeMedian	148	10.645	5.650	2.377	11	20	3	23	10	12

Figure 4.3.9. Scatter plot for birthday with content-based weights set at 1.

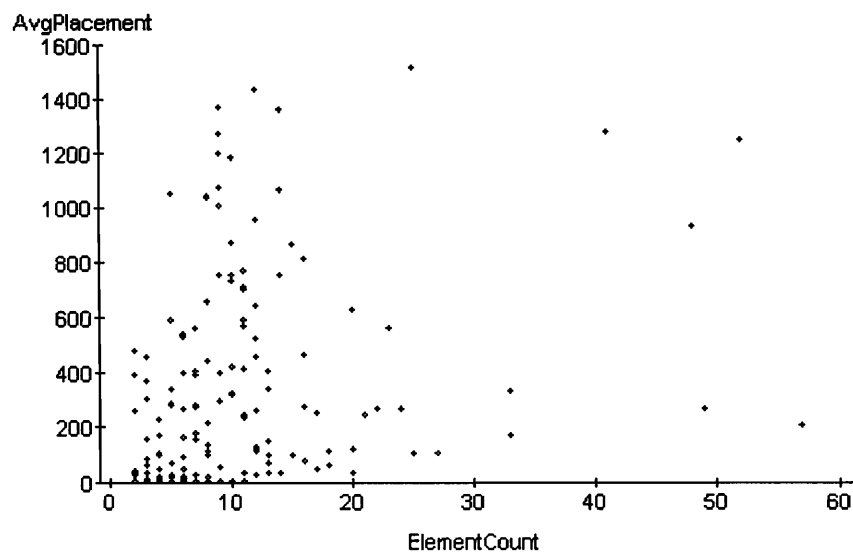


Figure 4.3.10. Box plot for birthday content-baseds with weights set with equal importance.

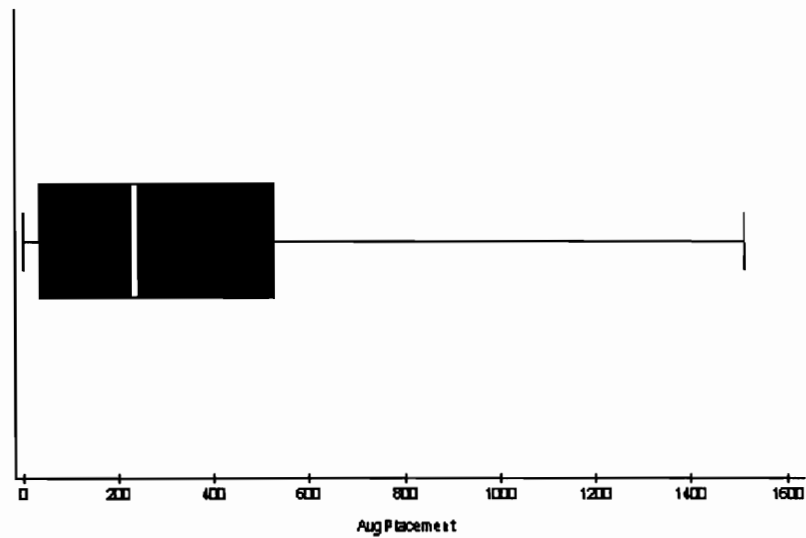


Figure 4.3.11. Distribution for birthday with content-based weights set with equal importance.

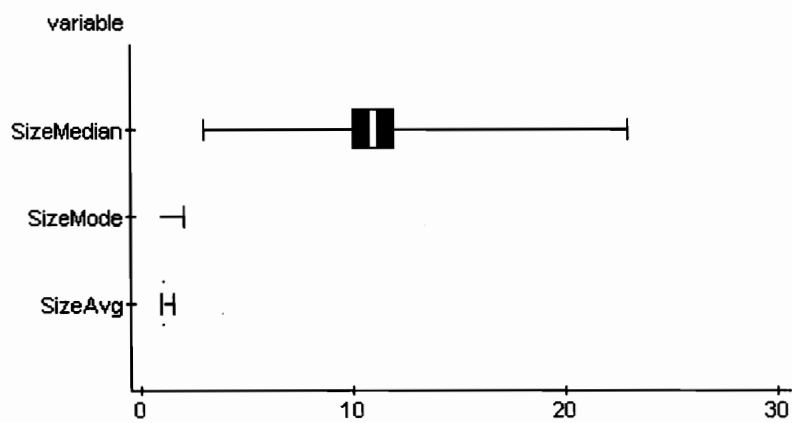
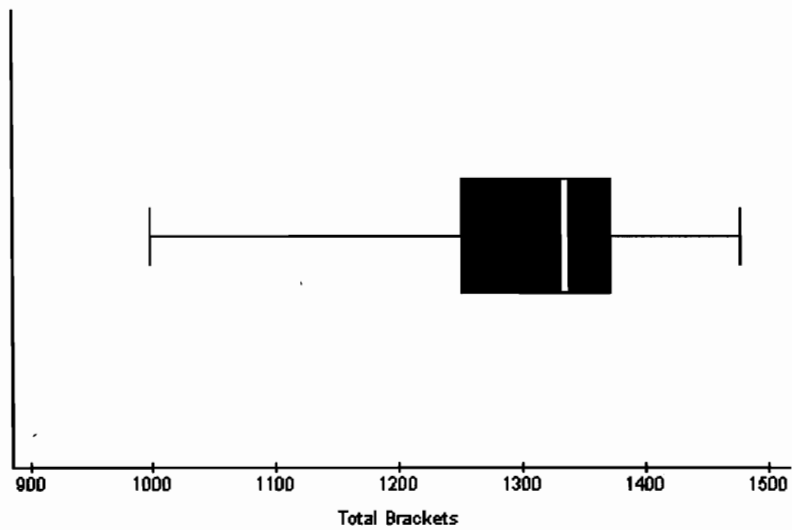


Figure 4.3.12. Number of brackets box plot for birthday with content-based weights set with equal importance.



As shown in Figures 4.3.9 through 4.3.12, the average placement is high, about 200.

Table 4.3.4. Birthday content-based score with optimized weights.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	148	81.222	28749.117	169.555	8	1106	1	1107	3	75
Brackets	148	1316.736	10881.638	104.315	1345	489	997	1489	1267	1388
SizeAvg	148	1.172	0.010	0.102	1.139	0.505	1.031	1.537	1.104	1.209
SizeMode	148	1.202	0.298	0.546	1	3	1	4	1	1
SizeMedian	148	10.310	12.841	3.583	11	23	4	27	8	27

Figure 4.3.14. Scatterplot for birthday content-based score with optimized weights.

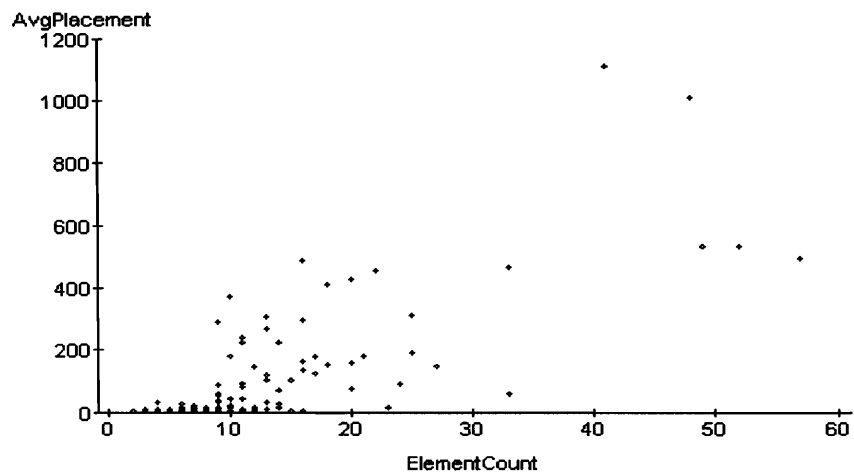


Figure 4.3.15. Box plot for birthday content-based score with optimized weights.

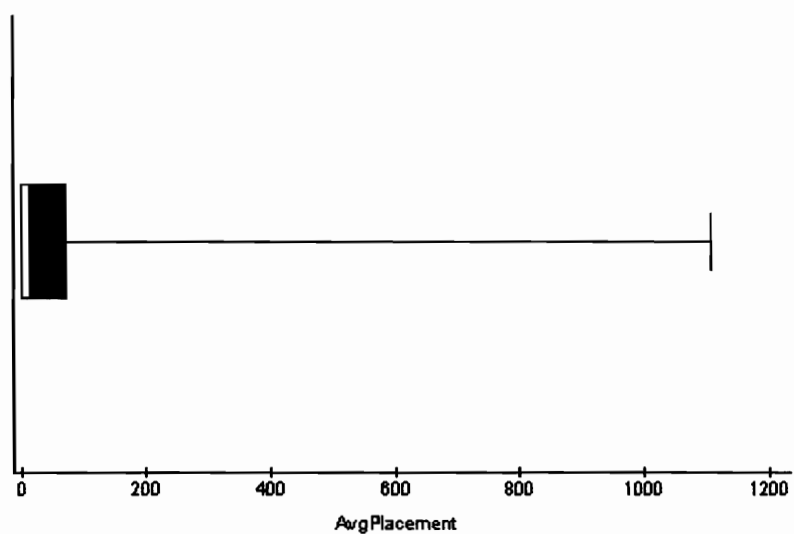


Figure 4.3.16. Distribution for birthday content-based score with optimized weights for bracket size.

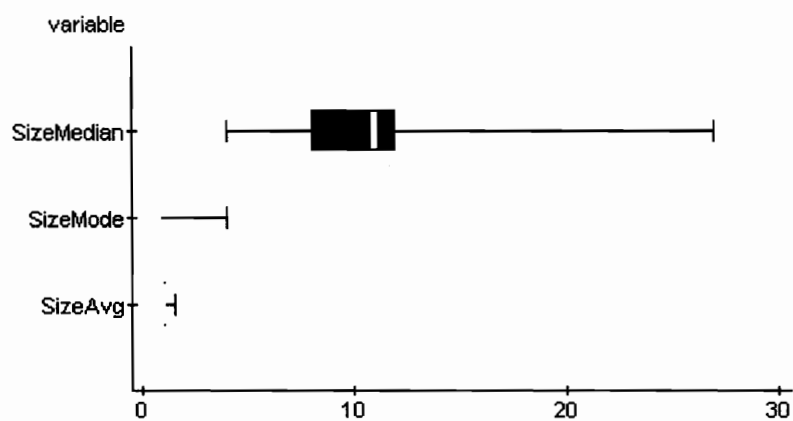
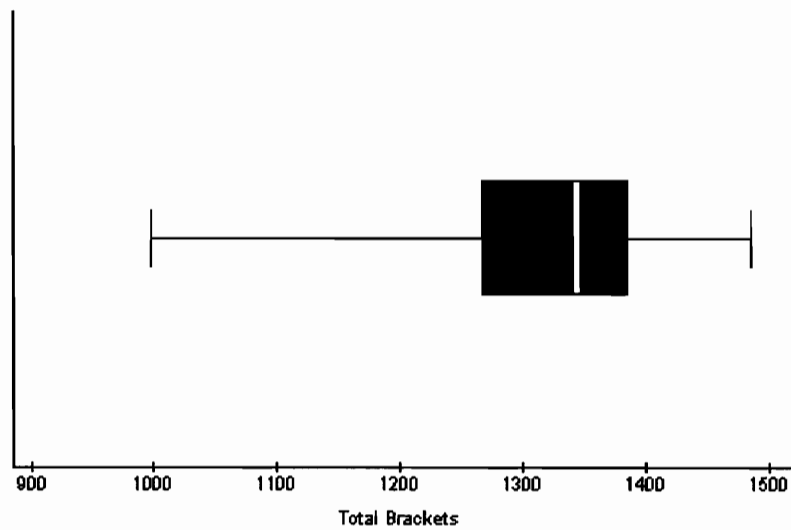


Figure 4.3.17. Box plot for birthday content-based score with optimized weights for number of brackets.



As shown in Figures 4.3.14 through 4.3.17, the hypothesis has been confirmed because the average placement has gone from an average of 345 to 81.

4.4 Phase 3 of Testing: Filtering Algorithm Accuracy

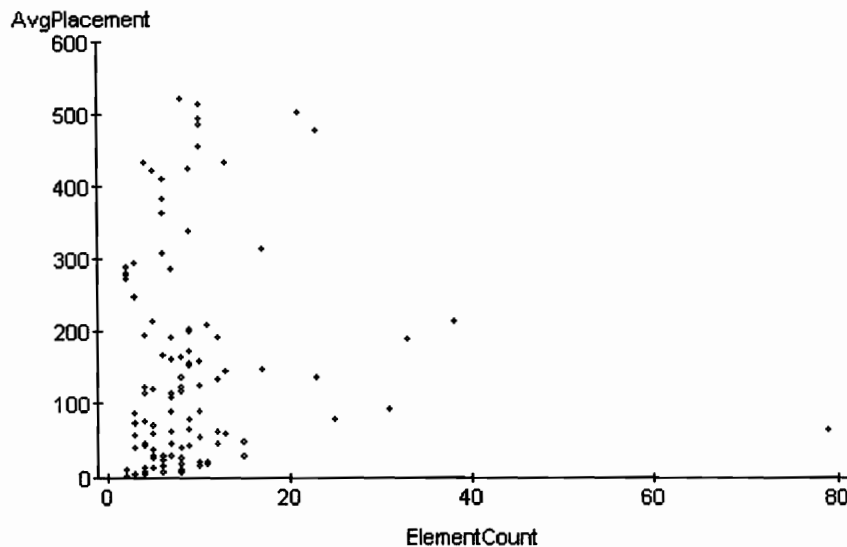
The accuracy of different filtering algorithms has been tested and compared in this section.

Again, as described in Section 4.3, the average placement is the most important statistic to observe.

Table 4.4.1. Total Christmas test algorithm 1 with weights set with equal importance.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	113	147.982	21040.566	145.053	94	520	2	522	30	214
Brackets	113	470.548	21387.09	146.243	518	523	3	526	508	523
SizeAvg	113	11.525	1238.972	35.199	1.015	174.333	1	175.333	1.005	1.035
SizeMode	113	45.008	20232.598	142.241	1	511	0	511	1	1
SizeMedian	113	46.398	20109.867	141.809	2	510	1	511	2	3

Figure 4.4.1. Total Christmas test algorithm 1 with weights with equal importance.



As seen in Figure 4.4.1, the number of cards purchased does not seem to have much influence on the average placement predictions.

Figure 4.4.2. Total Christmas test algorithm 1 with weights with equal importance.

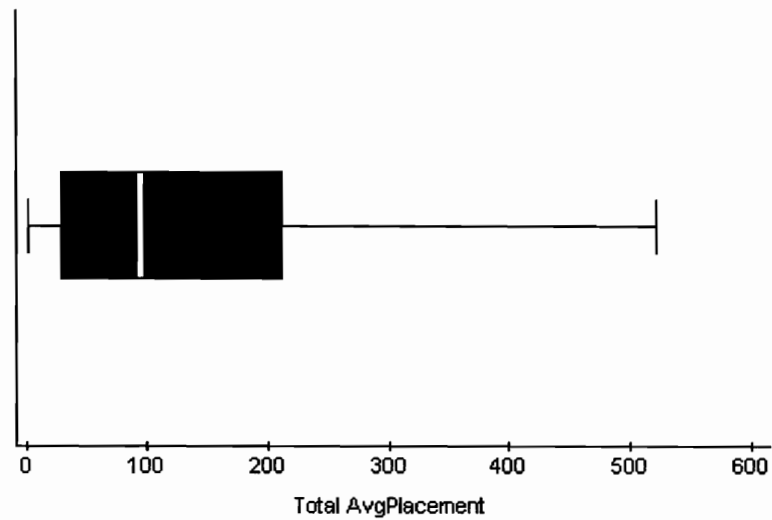


Figure 4.4.3. Total Christmas test algorithm 1 distribution with weights set with equal importance for bracket size.

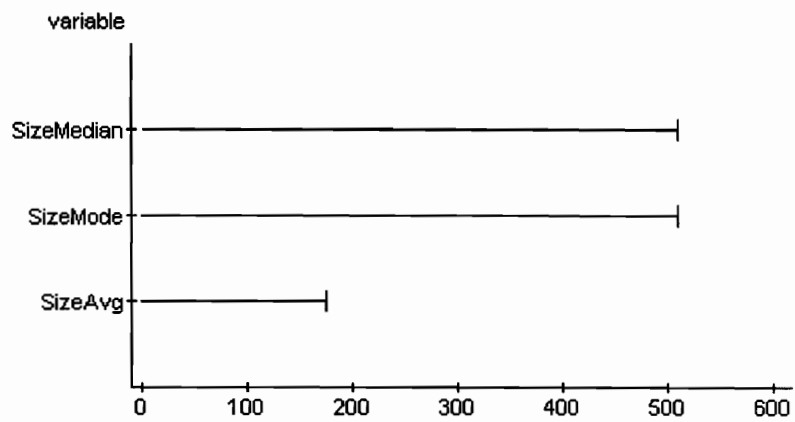
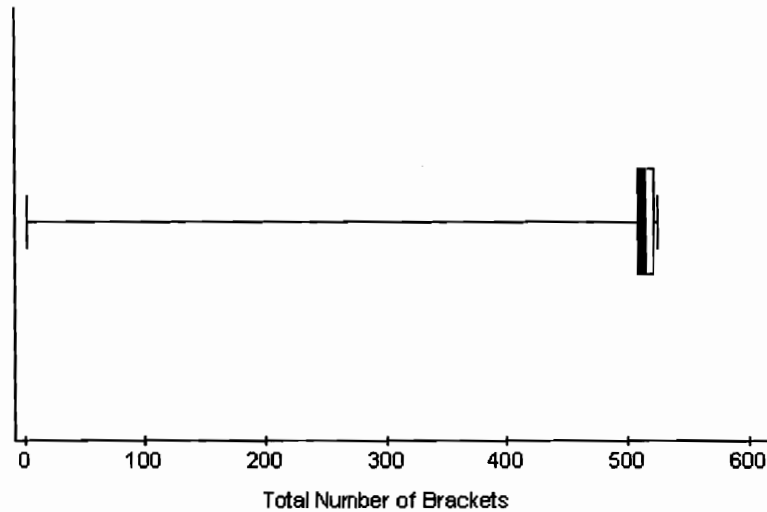


Figure 4.4.4. Total Christmas test algorithm 1 with weights with equal importance for total number of brackets.



Figures 4.4.2 through 4.4.4 show that when all attributes are set to 1, the system does relatively well for the first algorithm.

Table 4.4.2. Total Christmas test algorithm 1 with optimal weights set.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	113	86.761	8503.197	92.212	50	366	2	368	14	124
Brackets	113	470.743	21400.531	146.289	519	523	3	526	509	523
SizeAvg	113	11.525	1238.981	142.230	1.013	174.333	1	175.333	1.005	1.033
SizeMode	113	45.0442	20229.525	142.230	1	511	0	511	1	1
SizeMedian	113	46.433	20106.943	141.789	2	510	1	511	2	3

Figure 4.4.5. Scatterplot for total Christmas test algorithm 1 with optimal weights Set.

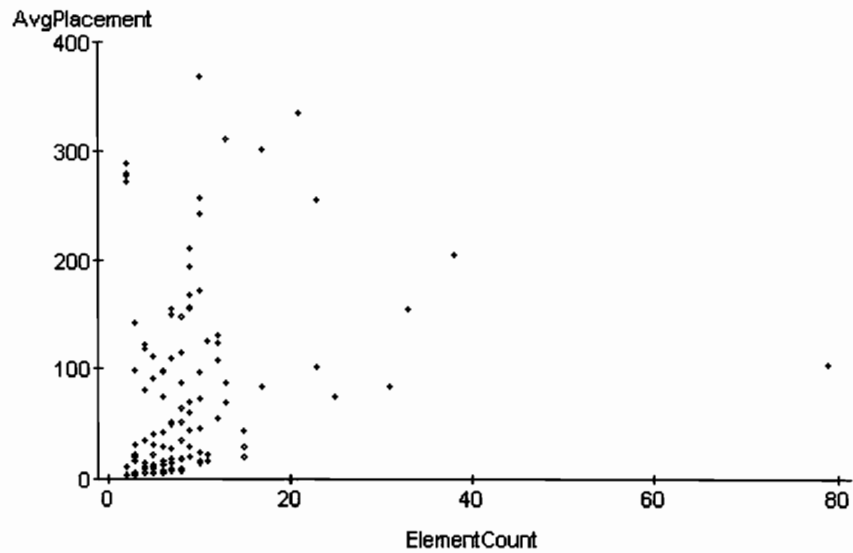


Figure 4.4.6. Box plot for total Christmas test algorithm 1 with optimal weights set.

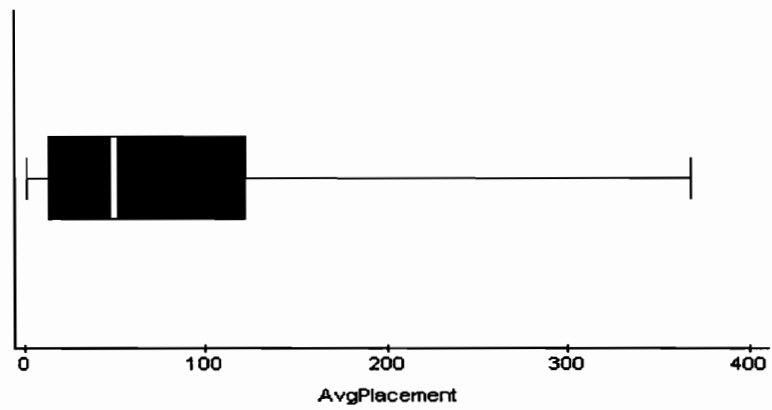


Figure 4.4.7. Total Christmas test algorithm 1 distribution with optimal weights set for bracket size.

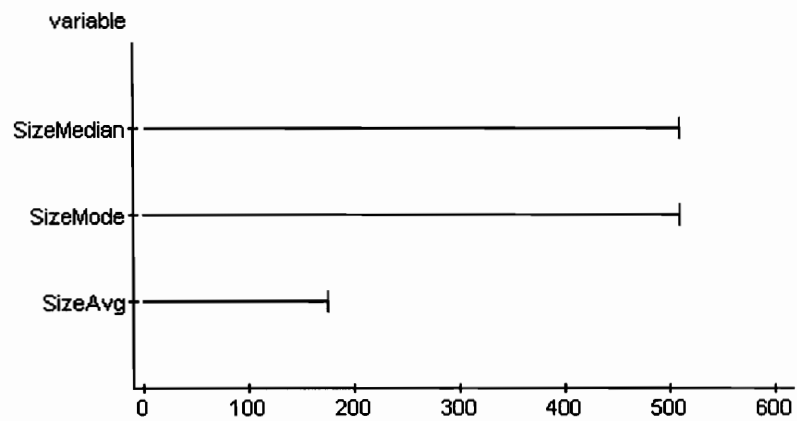
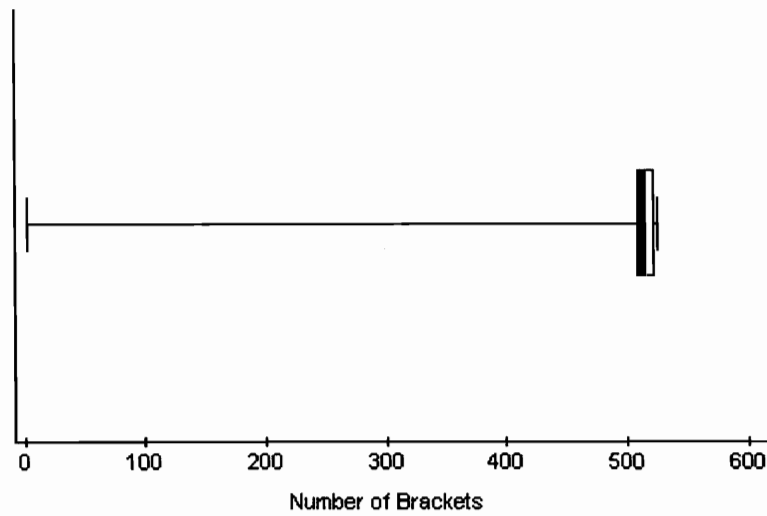


Figure 4.4.8. Box plot for total Christmas test algorithm 1 with optimal weights set for total number of brackets.



According to Figure 4.4.5 through 4.4.8, for birthdays the original system with adjusted weights performed better with an average placement of 86, as compared to 147.

Table 4.4.3. Total Birthday test algorithm 1 with weights set with equal importance.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	148	411.810	156963.42	396.186	311.5	1515	3	1518	67.5	634
Brackets	148	1312.844	38560.445	196.368	1364	1502	3	1505	12758	1413
SizeAvg	148	6.671	2372.357	48.706	1.123	509.981	1.018	511	1.089	1.202
SizeMode	148	21.455	30687.064	175.177	1	1529	1	1530	1	1
SizeMedian	148	31.270	30289.068	174.037	11	1526	4	1530	10	12

Figure 4.4.9. Total birthday test algorithm 1 with weights set with equal importance.

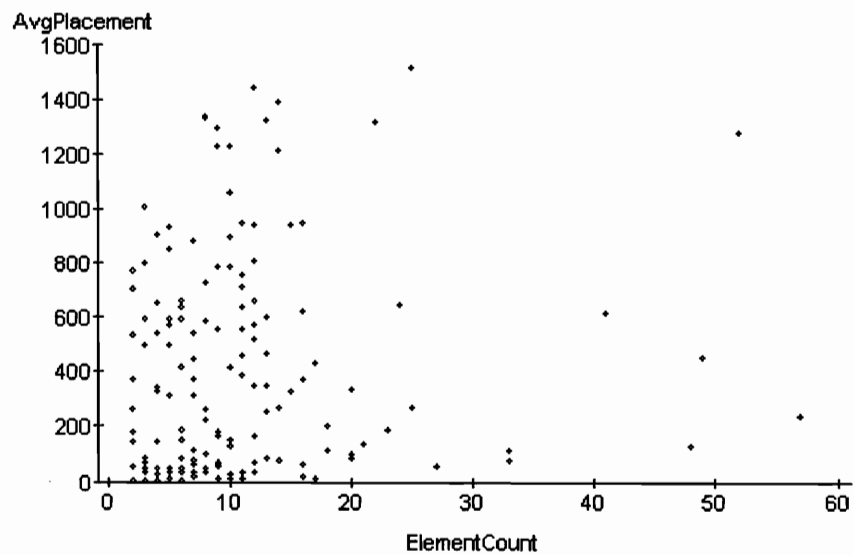


Figure 4.4.10. Total birthday test algorithm 1 with weights set with equal importance.

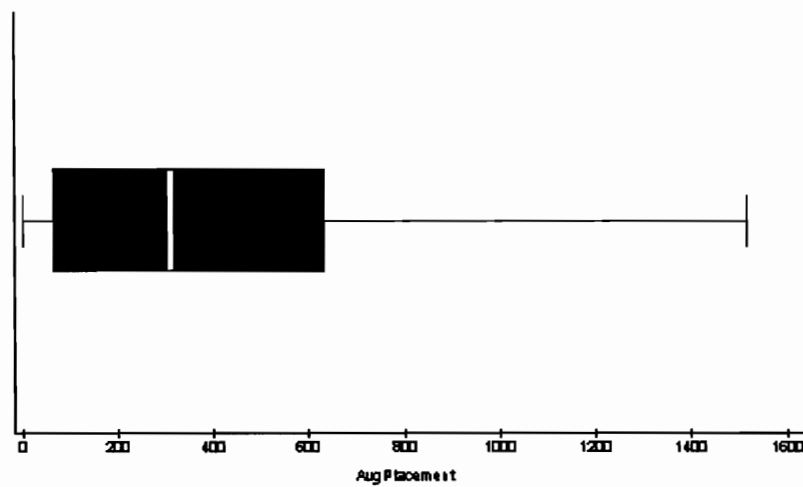


Figure 4.4.11. Total birthday test algorithm 1 distribution with weights set with equal importance for bracket size.

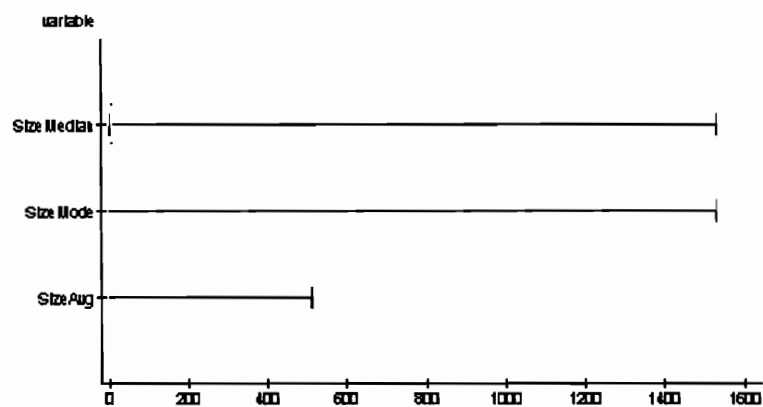
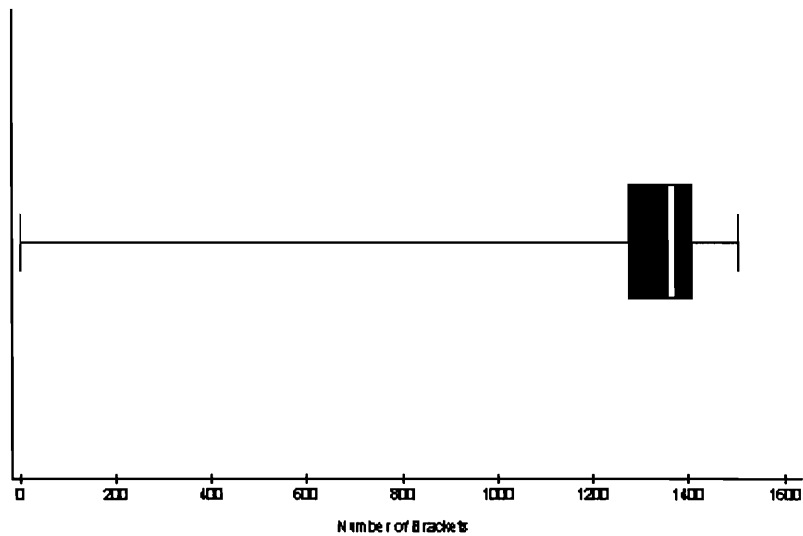


Figure 4.4.12. Box plot for total birthday test algorithm 1 with set with equal importance for total number of brackets.



According to Figures 4.4.9 through 4.4.12, with weights set at 1, it can be seen that for birthday cards the first algorithm performs poorly with average placement around 411.

Table 4.4.4. Total birthday test algorithm 1 with optimized weights set.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	148	84.415	24266.2	155.776	15	767.5	1	768.5	6	70.5
Brackets	148	1318.648	39037.71	197.579	1371	1505	3	1508	1283	1418.5
SizeAvg	148	6.666	2372.411	48.707	1.118	509.983	1.065	540	1.	7
SizeMode	148	21.567	30682.22	175.163	1	1529	1	1530	1	1
SizeMedian	148	30.844	30318.2	174.121	11	1528	2	1530	9	12

Figure 4.4.13. Total Birthday Test Algorithm 1 with Weights Set at Optimized Weights

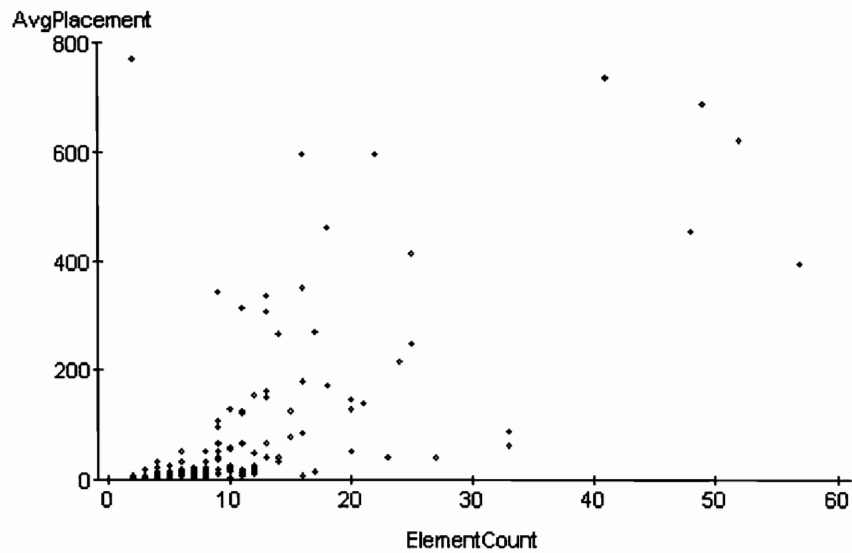


Figure 4.4.14. Box plot for total birthday test algorithm 1 with optimal weights set.

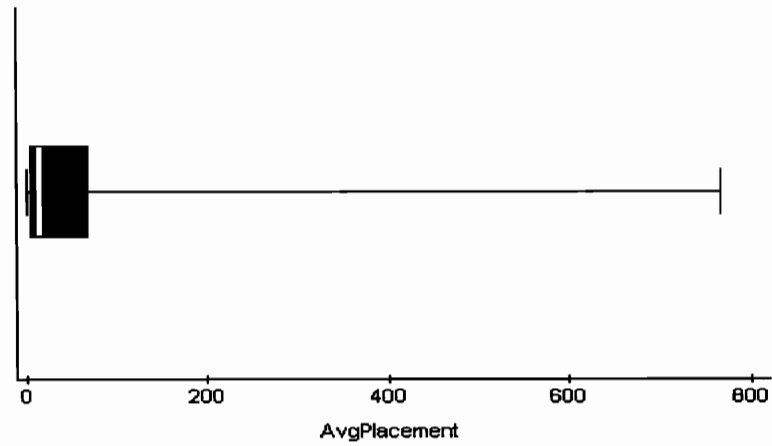


Figure 4.4.15. Total Birthday Test Algorithm 1 distribution with optimal weights set for bracket size.

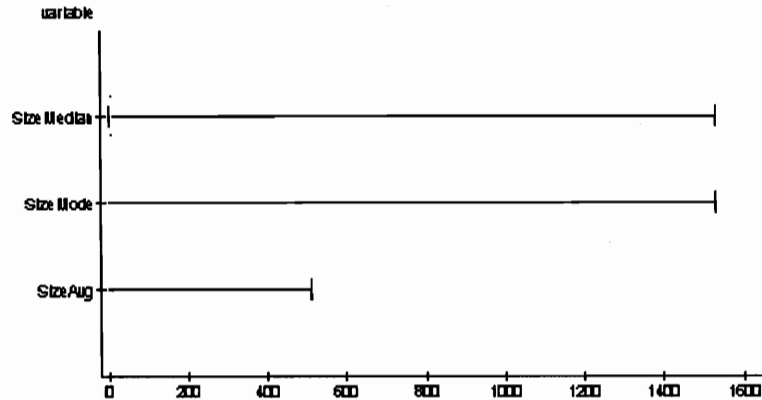
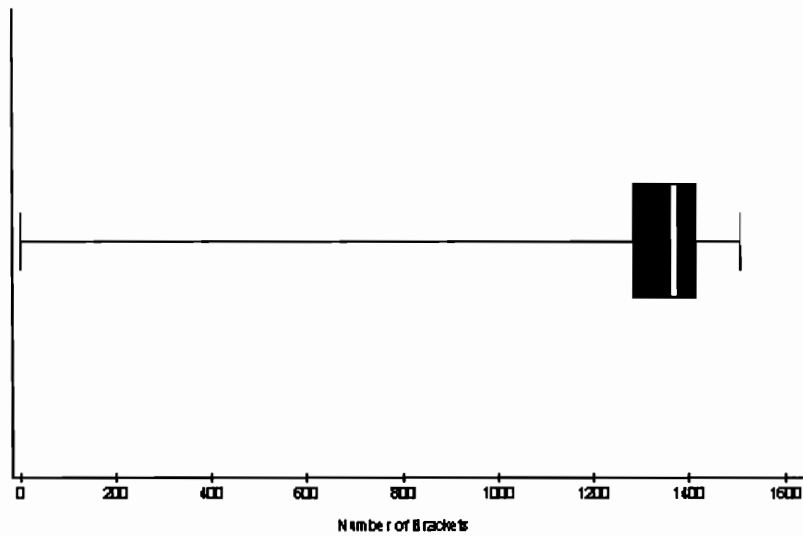


Figure 4.4.16. Box plot total birthday test algorithm 1 with optimal weights set for total number of brackets.



According to Figures 4.4.13 through 4.4.16, for birthdays the original system with adjusted weights performed better with an average placement of 84, as compared to 411.

Table 4.4.5. Total Christmas test algorithm 2 with weights set at optimized weights.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	113	68.084	5999.200	77.454	36	375	1	376	14	91
Brackets	113	506.982	1381.964	37.174	520	171	355	526	512	524
SizeAvg	113	1.044	0.009	0.09	1.011	0.481	1	1.481	1.003	1.027
SizeMode	113	0.991	0.312	0.558	1	4	0	4	1	1
SizeMedian	113	2.964	6.588	0.558	2	19	1	20	2	3

Figure 4.4.17. Box plot for total Christmas test algorithm 2 with optimal weights set.

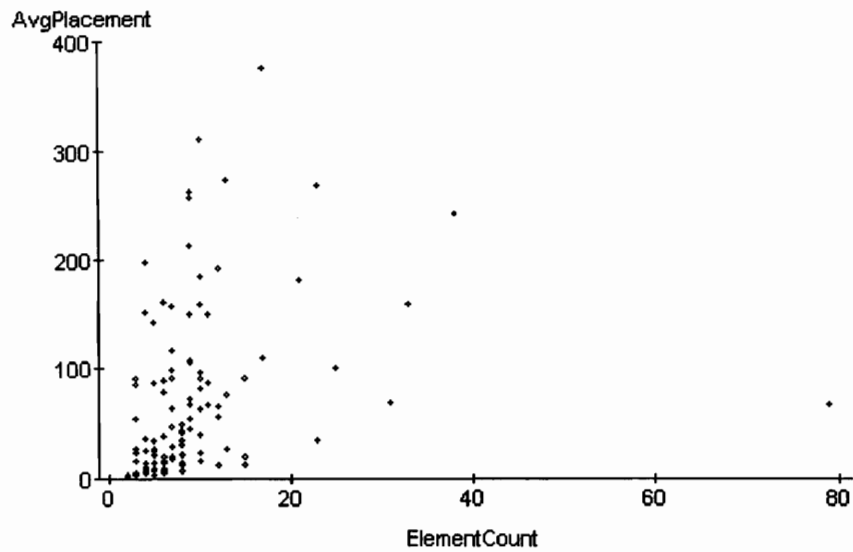


Figure 4.4.18. Box plot for total Christmas test algorithm 2 with optimal weights set.

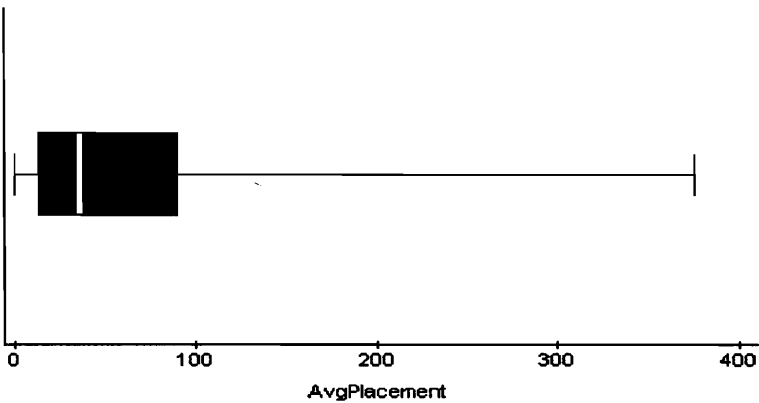


Figure 4.4.19. Total Christmas test algorithm 2 distribution with optimal weights set for size of bracket.

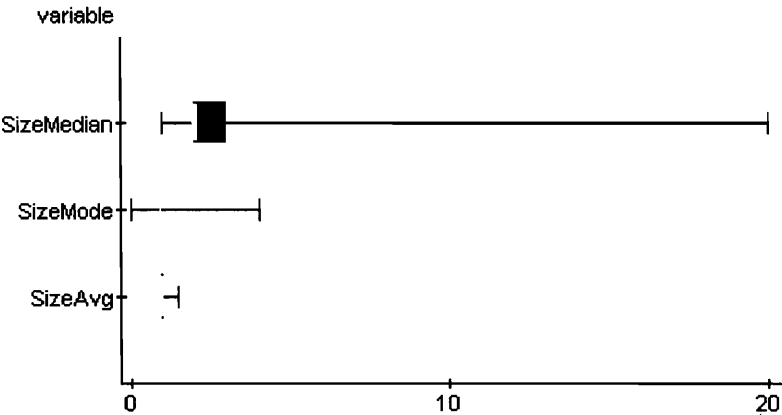
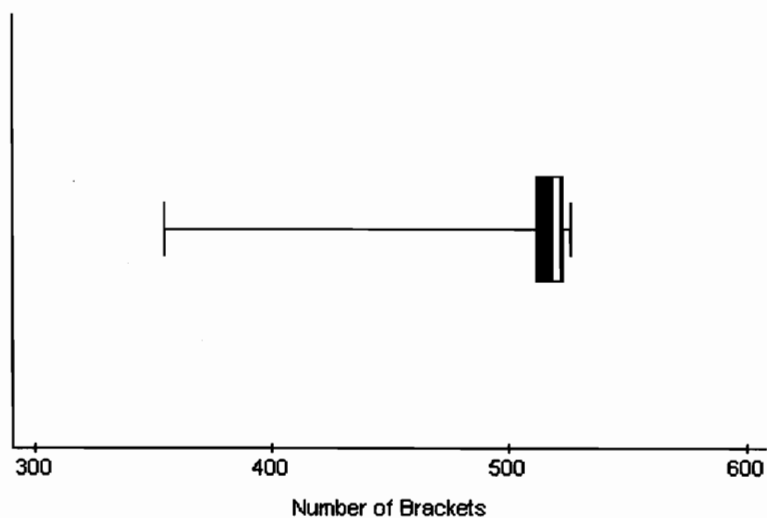


Figure 4.4.20. Total Christmas test algorithm 2 distribution with optimal weights set for distinct brackets.



According to Figures 4.4.17 through 4.4.20, for Christmas, the second algorithm performed better with an average placement of 68, as compared to 86. Therefore, we shall use the second algorithm for our comparisons.

Table 4.4.6. Total birthday test algorithm 2 with optimal weights set.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Placement	148	10.5908	29203.898	170.891	43	1089	1	1090	8	112
Brackets	148	1348.033	19320.236	138.997	1396	787	721	1508	1307.5	1441
SizeAvg	148	1.152	0.023	0.152	1.098	1.109	1.016	2.126	1.063	1.172
SizeMode	148	1.128	0.167	0.408	1	3	1	4	1	1
SizeMedian	148	10.918	22.523	4.745	11	39	2	41	9.5	12

Figure 4.4.21. Total birthday test algorithm 2 with optimal weights set.

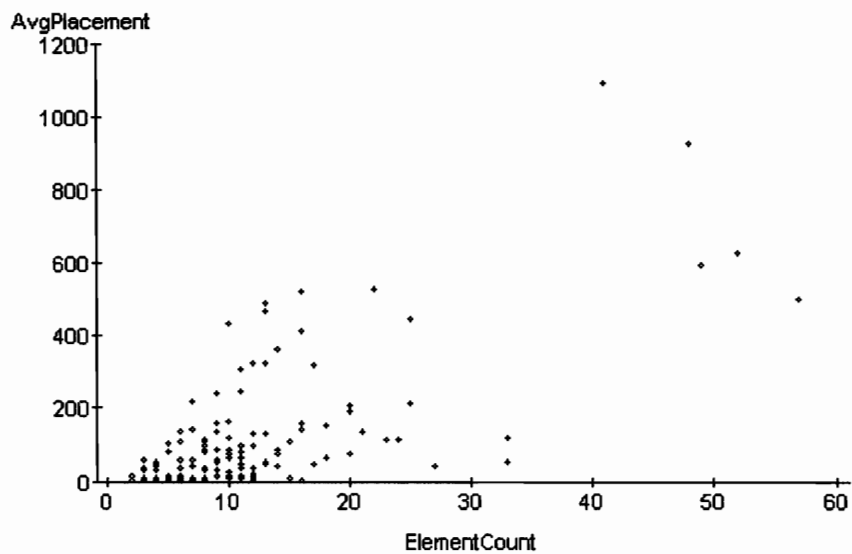


Figure 4.4.22. Box plot for total birthday test algorithm 2 with optimal weights set.

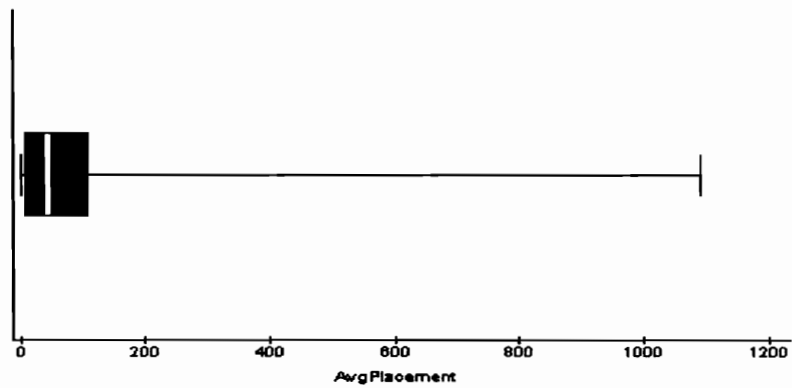


Figure 4.4.23. Total birthday test algorithm 2 distribution with optimal weights set for bracke size.

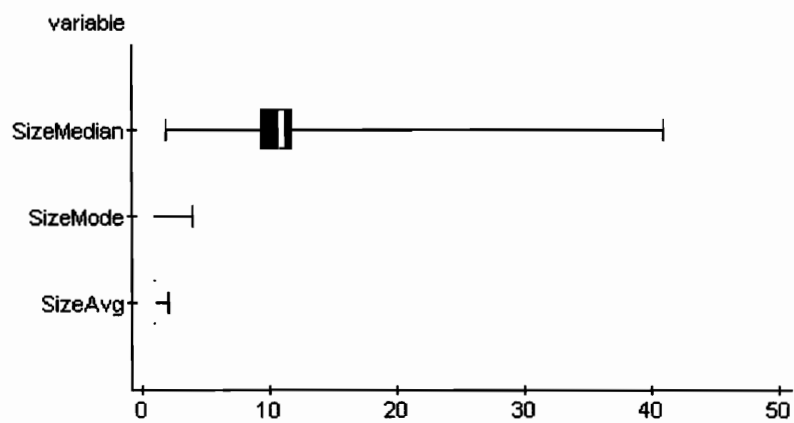
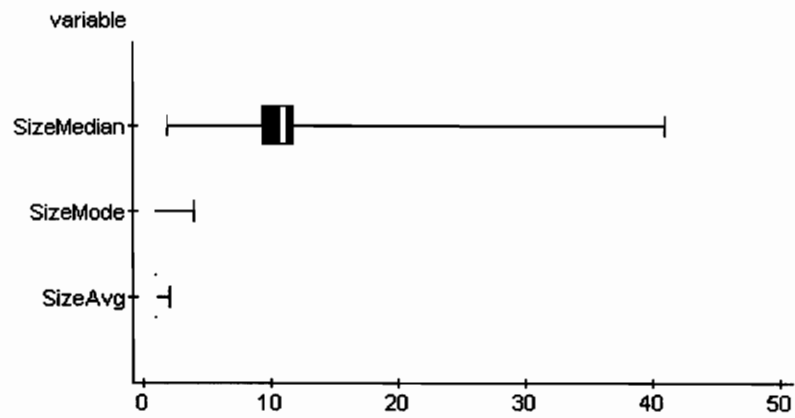


Figure 4.4.24. Total birthday test algorithm 2 distribution with optimal weights set for distinct brackets.



According to Figures 4.4.21 through 4.4.24, for birthday cards, our results show that algorithm 1 actually performs better than algorithm 2. For algorithm 1, the average placement was 84 as compared to 105.

4.5 Phase 4 of Testing: Personalized Filtering vs. Non-personalized Filtering

In this section, filtering accuracy results are compared with non-personalized approaches, in particular random cards approach and the most popular cards ("top ten") approach. At first *extreme outliers* were included, but the percent differences that were less than 0. After realizing that the box plot would not be nice with the outliers, they were removed to show how well the improvements are for the ones that predictions were improved. Moreover, doing this has the added benefit of knowing the poorness of the result.

As stated in Section 4.3, the average placement is the most important statistic to note, because it shows the average placement of the removed card. Again, the lower this value is, the more accurate the system is.

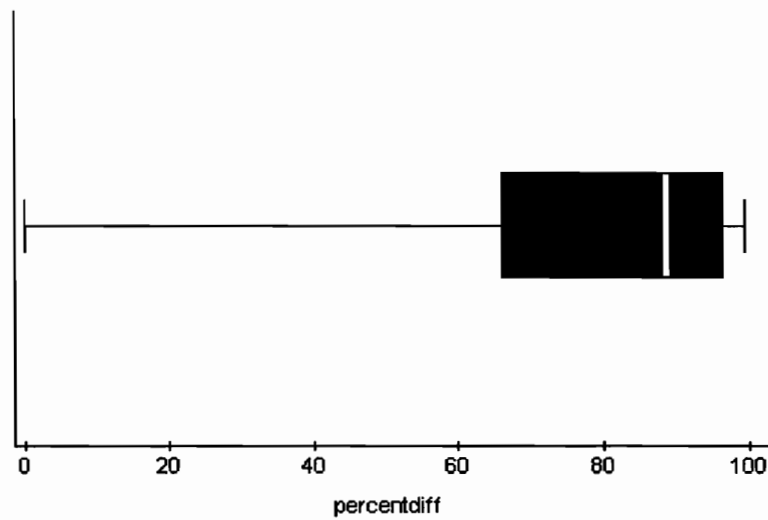
Table 4.5.1. Content-based for Christmas cards vs. random ordering with "extreme" outliers included.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Percent Difference	113	-88.849	539056.3	734.204	92.307	5949.808	-5850	99.808	67.821	98.523

Table 4.5.2. Content-based for Christmas cards vs. random ordering with "extreme" outliers not included

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Percent Difference	97	86.136	343.453	18.532	93.975	69.619	30.188	99.808	80	98.888

Figure 4.5.1. Box plot for content-based for Christmas cards vs. random ordering with "extreme" outliers removed.



As shown in Figure 4.5.1, on average, minus the extreme outliers (which only accounted for 16 users) the content-based system improved the ranking by 86%.

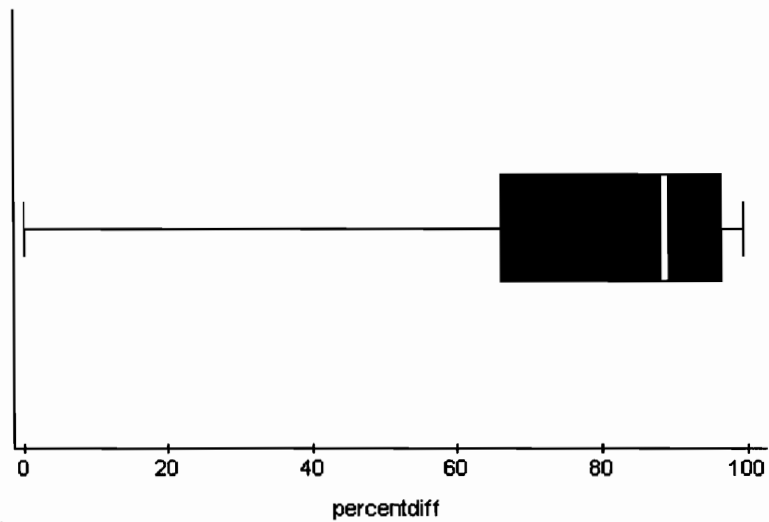
Table 4.5.3. Content-based for Christmas cards vs. "top 10" ordering with "extreme" outliers.

Variable	N	Mean	Var-iance	Std. Dev.	Med-ian	Range	Min	Max	Q1	Q3
Percent Difference	113	-469.779	6438527	2537.425	80	19899.676	-19800	99.675	14.285	95.327

Table 4.5.4. Content-based for Christmas cards vs. "top 10" ordering without "extreme" outliers.

Variable	N	Mean	Varianc e	Std. Dev.	Media n	Range	Min	Max	Q1	Q3
Percent Difference	88	77.010	733.366	27.080	88.584	99.675	0	99.675	60.045	96.666

Figure 4.5.2. Box plot for content-based for Christmas cards vs. "top 10" ordering without "extreme" outliers.



As shown in Figure 4.5.2, on average, minus the extreme outliers (which only accounted for 25 users) the content-based system improved the ranking by 77%.

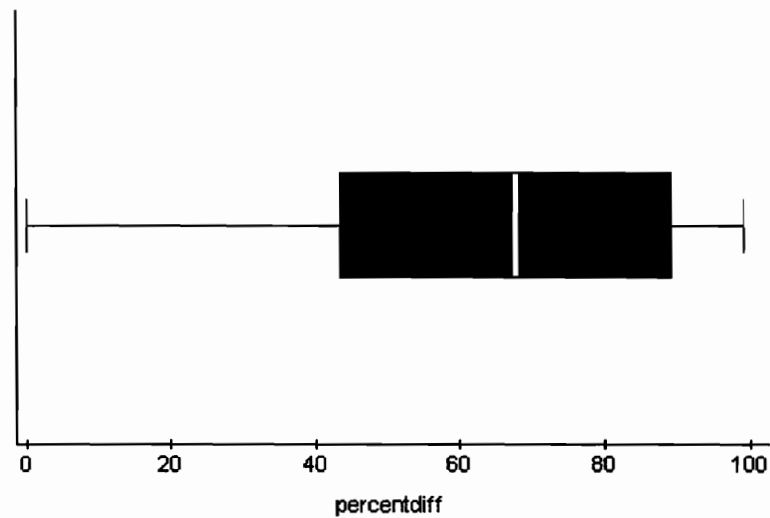
Table 4.5.5. Collaborative for Christmas cards vs. random ordering with “extreme” outliers.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Place-ment	108	-948.880	1.262	3553.183	-26.533	224499.32	-22350	99.319	-291.111	66.643

Table 4.5.6. Collaborative for Christmas cards vs. random ordering without “extreme” outliers.

Variable	N	Mean	Variance	Std. Dev.	Median	Range	Min	Max	Q1	Q3
Avg Place-ment	51	63.801	925.55	30.422	67.783	99.168	0.151	99.319	43.333	89.466

Figure 4.5.3. Box plot for collaborative for Christmas cards vs. random ordering without “extreme” outliers.



As shown in Figure 4.5.3, on average, minus the extreme outliers (which accounted for 53 users) the content-based system improved the ranking by 77%.

Tables 4.5.7 through 4.5.12 show the final comparisons between personalized and non-personalized filtering. In these tables, the strength of each algorithm for each domain of cards can be seen. These statistics provide the most useful information for comparing the algorithms and approaches on a general basis.

Tables 4.5.7 and 4.5.10 show the average placement statistics for all of the filtering algorithms for Christmas cards and birthday cards respectively. The others show the percent of improvement of one algorithm over another in terms of average placement and median placement. These improvements were calculated as a percent difference, where a positive percentage means that one algorithm was closer to the first position than another, while a negative percentage means that one algorithm is further than another.

Table 4.5.7. Christmas final statistics.

Variable	N	Mean	Variance	Std.Dev	Median	Range	Min	Max	Q1	Q3
Random	128	297.125	25952.914	161.099	314.5	571	4	575	158.5	416
TopTen	128	100.472	100.472	13732.167	117.184	46.5	1.5	461	15	165
Content	128	82.730	16429.945	128.179	21.5	549	1	550	4	103.5
Collab	128	156.621	21915.574	148.039	99.75	508.5	2	510.5	24	286.5
Combined	128	99.851	12774.694	113.0251	49.5	407	1	408	14.5	136.5

Page missing in
original

IQP/MQP SCANNING PROJECT



George C. Gordon Library
WORCESTER POLYTECHNIC INSTITUTE

Table 4.5.10. Birthday final statistics.

Variable	N	Mean	Variance	Std.Dev	Median	Range	Min	Max	Q1	Q3
Random	130	619.907	14272.97	377.793	555	1281	15	1296	315	922
TopTen	130	154.519	35923.56	189.535	86.5	836	1.5	837.5	17.5	198.5
Content	130	132.292	37035.23	192.445	55.5	1105	3	1108	17	162
Collab	130	458.115	111571.36	334.022	706.5	925	6	931	100	749
Combined	130	176.115	24629.88	156.939	152.5	941	4	945	47	248

Table 4.5.11. Mean comparison for birthday.

	Random	TopTen	Content	Collab	Combined
Random		-301.18	-368.59	-35.32	-251.99
TopTen	75.07		-16.80	66.27	12.26
Content	78.66	14.38		71.12	24.88
Collab	26.10	-196.48	-246.29		-160.12
Combined	71.59	-13.98	-33.13	61.56	

Table 4.5.12. Median comparison for birthday.

	Random	TopTen	Content	Collab	Combined
Random		-541.62	-900.00	21.44	-263.93
TopTen	84.41		-55.86	87.76	43.28
Content	90.00	35.84		92.14	63.61
Collab	-27.30	-716.76	-1172.97		-363.28
Combined	72.52	-76.30	-174.77	78.41	

For birthday cards, our algorithm performs well, 72.52% better than random, but 76% worse than Top Ten placements. Bag of word attribute appears to be a strong attribute and should be considered alone for birthday cards. It performs better than random, top ten, collaborative, and is 63% better than our total prediction.

5. Conclusions

The goal of this project was to determine if Sparks.com would benefit from an online filtering system. Initially, the database of card attributes and order information at Sparks.com was analyzed, and the relevant information needed for filtering was sifted out into another database, called the WPI database. This database was further divided into a Christmas cards and a birthday cards database to study the results of filtering for these two types of cards. A filtering engine was written in Java implementing content-based filtering, collaborative filtering, and a combination of the two filtering approaches. The filtering results were compared with actual purchases made to determine the accuracy between the types of cards, and between the filtering algorithms.

As shown in Tables 4.5.7 through 4.5.12, comprehensive testing was performed and the following results were deduced:

- For Christmas cards, bag-of-words weighed most heavily in making accurate predictions followed by sentiments, price, and the distinct attributes.
- For birthday cards, bag-of-words was the most important factor in making predictions. The other content attributes did not significantly improve the accuracy of predictions.
- Collaborative filtering worked very well with Christmas cards, but was a poor indicator for attaining accurate results for the recipient-driven domain of birthday cards.
- Collaborative algorithm predictions are worse than content-based predictions for cards.
- For both card domains, combining content-based and collaborative filtering algorithms using a per-user, per-card approach resulted in slightly worse predictions than content-based alone, but better than collaborative.

- The predictions of filtering were very accurate for Christmas cards, sender-driven cards, compared to non-personalized approaches. Even for birthday cards, recipient- driven cards, the results of personalized filtering were better than random ordering of cards or ordering by card sales.
- A per-user per-card approach towards filtering can be implemented in a real-time system by dividing filtering into offline and online calculations.

Thus, all the approaches, content-based, collaborative and the top 10 approach show a huge improvement over the random approach used by Sparks.com. The overall results therefore suggested that personalized filtering can be applied to the domain of cards, in particular for birthday and Christmas cards. Because each domain can be classified using two types of opinion “genres” for cards: sender-driven, and recipient-driven, filtering can most likely be applied to all collections of cards which fall under the same opinion “genres” such as Mother’s Day cards, Father’s Day cards, apology cards, and get-well-soon cards.

5.1 Future Work

The filtering engine can be made more accurate and realistic by incorporating different factors that might influence customer purchases. Some of these factors are:

- Some customers purchase different types of cards for different occasions. Some customers might like Christmas cards with a serious and religious sentiment, but may prefer birthday cards with a naughty and funny sentiment. Introducing card-type specific weights in the algorithms can enhance filtering.

- Customers might also buy different sentiment cards depending on the recipient. For example, some customers might like to buy funny and sentimental cards for friends, romantic cards for close ones, beautiful cards for relatives, or even religious cards for their parents. By storing recipient profiles along with the customer profiles, algorithm weights can be updated dynamically to support recipient-specific cards.
- Often the time of year also influences the cards that are purchased by customers. Further analysis needs to be done to gauge how customer patterns differ during different times of the year. The filtering algorithms should take into account the time of the year and assign weights accordingly.
- Customer tastes can often change with age and time. Customers may like funny and romantic cards when they are young, but as time passes, they may begin to develop a liking for more serious cards. The algorithms can have an age factor or a history threshold that affects the predictions of cards.
- Customer tastes might be influenced by their demographic information. Steps can be taken to collect such information about customers and use that for filtering.
- Many customers login to Sparks.com as anonymous customers; that is, they do not maintain a profile. Since these customers do not have any purchase history, they must be treated

specially, and the filtering algorithms must be designed accordingly to possibly include them as well.

- Sparks.com sells chocolates, gift certificates and flowers in addition to greeting cards. Analysis must be done to understand how closely customer purchases are driven by similar tastes across these domains of items. Cross-suggestion between these items can help to increase sales and increase customer satisfaction and convenience.
- Further time analysis and tests can be done to decide an optimum way to conduct filtering in terms of offline and online calculations.
- There were 27 card attributes that were used for content-based filtering. These attributes were divided into 4 groups to study their weights and importance in making accurate predictions. Further comprehensive testing can be done on each of them separately by testing with different weights to decide on an optimum weight scale for best results.
- Two approaches of combining content-based and collaborative filtering were tested based on accuracy of their individual predictions. Other factors such as number of cards purchased and coverage results can be used to supplement the combination approach. Also, instead of basing the final predictions solely on the accuracy of the collaborative system as done in

combination algorithm 2, the final predictions could be based on the accuracies of *both* the collaborative system and the content system.

- Further study for the selection of the optimal programming language, operating system, and computer system can be done to improve the speed of the system.

The bag-of-words approach used for filtering can be further enhanced in the following ways:

- Further analysis can be done by dividing the words into groups and having a separate bag of words for each group.
- Individual word analysis can be done by removing the common words in groups of cards to reduce the size of the database and to increase the speed of creating the bag of words.
- Each word can be mapped to an integer to improve the speed of the database or other algorithms to reduce processing time.
- Words can also be divided into common “categories” to further increase the speed of the database.
- Possible improvements to the bag of words approach can be analyzed by not including back text or front text in.

In the current filtering engine, different algorithms can be run on the database generating their own predictions. These predictions are then combined together with confidence measures to produce final predictions. This structure of the filtering engine makes it easy to incorporate the enhancements mentioned above into the system.

Appendix A. Company Information

Sparks.com, an Internet startup, aims to bring a traditional greeting card store to your desktop. The company, incorporated in May 1998, was powered by angel-investor funds. Investments from Benchmark Capital and Venture Strategy Group also helped in providing funds for the initial startup. As is typical of the Internet start-up, employees use saw horses and doors as desks, wear a million types of hats, and cheerfully work on breakneck "Internet time." What may be slightly different is that employees sit amongst the greeting card racks. Today, their 15,000 sq. ft. warehouse in San Francisco's India Basin houses an always-growing team, currently consisting of about 35 employees and 12-20 contractors.

Felicia Lindau is the CEO of this Internet startup company. Felicia, the 31-year-old daughter of a self-made entrepreneur in Texas, always planned to be an entrepreneur. After spending 11 years in technology marketing, with the last 4 spent launching new brands like MSN, Excite and Amazon.com, she developed the idea for an online greeting card store. The inspiration developed after she realized she did not have the time to go card shopping for her mother's birthday. This frustration led to inspiration, and inspiration led to a detailed business plan. Armed with nothing but experience, devotion and that business plan, Felicia left her position as an Account Supervisor (on Amazon.com) at Foote, Cone and Belding. In March of 1998, she officially began to raise funding and begin creating a team to build Sparks.com.

The second major player at Sparks.com is the CTO, Jason Monberg. Jason, a Palo Alto native, had previously worked with Felicia as a founding member of the Interactive Group at Anderson Lembke advertising. Together they introduced Microsoft to the world of online advertising. Soon after, Jason jumped Anderson Lembke for a post as a lead engineer and systems integrator at CKS Group. There he worked on large e-commerce sites such as IBM and

Levi's. Jason's experience matched perfectly with Felicia's, and thus she knew Jason would be of great help. In March 1998, Jason officially became a founding partner (http://www.sparks.com/help_faqs/about_overview.html).

With the strength of the Internet, visitors to Sparks.com can access thousands of paper greeting cards from hundreds of publishers. With such a vast inventory, users can potentially find the perfect card for almost all occasions. Presently, Sparks.com allows customers different ways of selecting cards from keyword specific to browsing by subject. Customers can create an account to keep an address book, personal reminders as well as a variety of other services that help people stay in touch. In addition to greeting cards, Sparks.com sells personal invitation cards, gift certificates and even stamps.

Appendix B. SQL Queries

Building the WPI Database, Testing and Statistics

1)

Run from: sp_card

Returns: the list of visual_objects attribute of sp_card table

```
SELECT visual_objects
FROM [dbo].[SP_CARD]
WHERE [dbo].[SP_CARD].visual_objects LIKE '%Far Side%'
```

Result:

visual_objects

Far Side; Gary Larson; bears; forest; people; cartoon
sofa; furniture store; dogs; cartoon;Far Side
woman; cartoon; The Far Side; joke; birthday
The Far Side;comic strip;flying saucer;UFO;alien;
The Far Side;Gary Larson;comic strip;dogs;
The Far Side; dogs; men; picket fence
The Far Side; dogs; snowcat; snow; decorations; wreath; tree
The Far Side; cows; cartoon; family;
.....

2)

Run from: sp_card

Returns: the list of back_text attributes which are neither Blank nor N/A

```
SELECT back_text
FROM [dbo].[SP_CARD]
WHERE [dbo].[SP_CARD].back_text NOT LIKE " AND
      [dbo].[SP_CARD].back_text NOT LIKE '%N/A%'
```

Result:

back_text

-----Oscar's

Garage 'You still want us to change the oil'
'To help you monitor your progress, we make a plaster cast of every member's posterior once a month.'
You didn't slobber, did you?)
'Snack' by Anne Davis
'Mom And Dad' Animal Art By Anne Davis

.....(2687 lines)

3)

Run From: sp_card

Returns: the count of cards that have the word christmas and santa in their visual_objects field and that are not charity cards.

```
SELECT COUNT(*)
FROM [dbo].[SP_CARD]
WHERE [dbo].[SP_CARD].visual_objects LIKE '%christmas%' AND
      [dbo].[SP_CARD].visual_objects LIKE '%Santa%' AND
      [dbo].[SP_CARD].charity_id = 1
```

Result:

30

4)

Run From: sp_card

Returns: the count of distinct artist names (this does not count blank as 1 artist)

```
SELECT COUNT(DISTINCT (artist_name))
FROM [dbo].[SP_CARD]
WHERE artist_name NOT LIKE '' AND artist_name NOT LIKE "
```

Result:

1490

5)

Run From: sp_customer

Returns: the attributes of a customer whose address information is not null.

```
SELECT *
FROM customer01.dbo.sp_customer
WHERE address_id IS NOT NULL
```

Result:

None

(This means that the addresses of none of the customers have been stored in the database)

6)

Run From: sp_order

Returns: The total amount of money spent by the customer at Sparks.com to buy cards

```
SELECT SUM(order_total)
FROM customer01.[dbo].[SP_ORDER]
```

WHERE customer_id = 1934

Result:

2.8500

7)

Run From: sp_order

Returns: The total number of orders placed by a customer in which he/she received a discount

```
SELECT COUNT(*)
FROM customer01.[dbo].[SP_ORDER]
WHERE customer_id = 1934
```

Result:

32

```
SELECT COUNT(*)
FROM customer01.[dbo].[SP_ORDER]
WHERE customer_id = 1934 AND discount > 0
```

Result:

14

(So almost half the orders had discounts available)

8)

Run From: sp_customer

Returns: For the customer with id = 1890, returns the list of cards that he/she bought.

```
SELECT customer01.[dbo].[SP_ORDER].customer_id,
       customer01.[dbo].[SP_ORDER_LINE_ITEM].sku
FROM customer01.[dbo].[SP_ORDER_LINE_ITEM], customer01.[dbo].[SP_ORDER]
WHERE customer_id = 1890 AND
       customer01.[dbo].[SP_ORDER_LINE_ITEM].order_id =
       customer01.[dbo].[SP_ORDER].order_id
```

Result:

customer_id sku

customer_id	sku
1890	0002682
1890	0002110
1890	0002110

9)

Run From: sp_customer

Returns: For each customer having ID > 60,000: returns the customer id and the sku (card id) of the cards that they bought.

```
SELECT customer01.[dbo].[SP_ORDER].customer_id,  
       customer01.[dbo].[SP_ORDER_LINE_ITEM].sku  
FROM customer01.[dbo].[SP_ORDER_LINE_ITEM], customer01.[dbo].[SP_ORDER]  
WHERE customer01.[dbo].[SP_ORDER_LINE_ITEM].order_id =  
customer01.[dbo].[SP_ORDER].order_id AND customer_id > 60000  
ORDER BY customer_id
```

Result:

customer_id sku

customer_id	sku
60002	8000003
60002	0009579
60002	0008539
60004	0006765
60007	7010829
60008	0005057
60008	0003813
60008	0001188
60008	0005058
60008	0003353
60008	0004028
60010	0000268
60010	0007994
60013	0005998
60013	0005963
60013	0003814

10)

Run From: sp_customer

Returns: The count of the number of cards that customer with ID = 1933 bought.

```
SELECT COUNT(customer01.[dbo].[SP_ORDER_LINE_ITEM].sku)  
FROM customer01.[dbo].[SP_ORDER_LINE_ITEM], customer01.[dbo].[SP_ORDER]  
WHERE customer01.[dbo].[SP_ORDER_LINE_ITEM].order_id =  
customer01.[dbo].[SP_ORDER].order_id AND customer_id = 1933
```

Result:

64

11)

Run From: sp_customer

Returns: A list of customer Ids and the count of cards that they have bought so far at Sparks.com

```
SELECT customer_id, COUNT([dbo].[SP_ORDER_LINE_ITEM].sku)
      AS num_of_cards
FROM [dbo].[SP_ORDER_LINE_ITEM], [dbo].[SP_ORDER]
WHERE [dbo].[SP_ORDER_LINE_ITEM].order_id = [dbo].[SP_ORDER].order_id
GROUP BY customer_id
ORDER BY customer_id
```

Result:

customer_id num_of_cards

```
-----
1790      1
1791     18
1793     19
1811      7
1812      4
1813     70
1819     31
(and so on for all the customers...)
```

12)

Run From: sp_customer

Returns: for a given customer id, the card id, and the visual objects field for the cards that customer 1934 bought which had a heart in their visual objects field.

```
SELECT customer_id, sp_card.sku, sp_card.visual_objects
FROM customer01.dbo.sp_order, customer01.dbo.sp_order_line_item, sp_card
WHERE customer01.dbo.sp_order_line_item.order_id = customer01.dbo.sp_order.order_id
AND
      customer01.dbo.sp_order_line_item.sku = sp_card.sku AND customer_id = 1934 AND
      sp_card.visual_objects LIKE '%heart%'
```

Result:

customer_id sku visual_objects

```
-----
-----1934 0006124 shamrock; clover; hearts
1934 0002484 heart; string; wood frame; red;
1934 0009392 bow;heart flowers;Mother's Day;
1934 0009501 heart; rose; love; Mother's Day
1934 0009434 heart
1934 0009435 hearts
```


(6 row(s) affected)

13)

Run From: sp_customer

Returns: For each customer, the count of the total number of cards that were bought which have hearts in them.

```
SELECT customer_id, COUNT(sp_card.sku)
FROM customer01.dbo.sp_order, customer01.dbo.sp_order_line_item, sp_card
WHERE customer01.dbo.sp_order_line_item.order_id = customer01.dbo.sp_order.order_id
  AND customer01.dbo.sp_order_line_item.sku = sp_card.sku AND sp_card.visual_objects LIKE
'%heart%'
GROUP BY customer_id
ORDER BY customer_id
```

Result:

customer_id

1791	1
1811	2
1813	3
1819	4
1869	6
1887	1
1889	12
1894	24
1897	2

(and so on...)

14)

Run From: sp_card

Returns: a list of customers, and for each customers, a list of the cards that they bought.

```
SELECT customer01.[dbo].[SP_ORDER].customer_id,
       customer01.dbo.SP_ORDER_LINE_ITEM.sku
FROM customer01.dbo.SP_ORDER_LINE_ITEM,
       customer01.dbo.[SP_ORDER]
WHERE customer01.[dbo].[SP_ORDER_LINE_ITEM].order_id =
       customer01.[dbo].[SP_ORDER].order_id
ORDER BY customer01.dbo.SP_ORDER.customer_id
```

Result:

customer_id sku

```

-----
1790    0003628
1791    0007024
1791    0000023
1791    0000293
1791    0000299
1791    0000419

```

(and so on...)

15)

Run From: sp_customer

Returns: the customer id and the number of heart cards that they bought, for only those cases, where they bought more than 10 heart cards

```

SELECT customer_id, COUNT(sp_card.sku)
      AS no_of_heart_cards
FROM customer01.dbo.sp_order,
      customer01.dbo.sp_order_line_item, sp_card
WHERE customer01.dbo.sp_order_line_item.order_id = customer01.dbo.sp_order.order_id
      AND
      customer01.dbo.sp_order_line_item.sku = sp_card.sku AND
      sp_card.visual_objects LIKE '%heart%'
GROUP BY customer_id
HAVING COUNT(sp_card.sku) > 10
ORDER BY customer_id

```

Result:

customer_id no_of_heart_cards

```

-----
1889     12
1894     24
2275     36
2452     11
2792     12
14965    20
29050    21
65447    26
112096   11
133938   15
163314   16

```

(11 row(s) affected)

16)

Run From: sp_customer

Returns: the customer id and the total number of cards that they bought, for only those cases where the total number of cards is greater than 10.

```
SELECT customer_id, COUNT(sp_card.sku)
      AS no_of_total_cards
FROM customer01.dbo.sp_order,
      customer01.dbo.sp_order_line_item, sp_card
WHERE customer01.dbo.sp_order_line_item.order_id = customer01.dbo.sp_order.order_id
      AND
      customer01.dbo.sp_order_line_item.sku = sp_card.sku
GROUP BY customer_id
HAVING COUNT(sp_card.sku) > 10
ORDER BY customer_id
```

Result:

customer_id no_of_total_cards

```
-----
1791      18
1793      18
1813      62
1819      30
1869      51
1887      27
1889     164
(and so on...)
```

17)

Run From: sp_customer

Returns: Creates a view that has two columns: the customer id – and the count of heart cards that they bought – only if they bought more than 10 heart cards.

```
CREATE VIEW HEART_LOVERS2 AS (SELECT customer_id, count(sp_card.sku) AS
no_of_cards
```

```
      FROM customer01.dbo.sp_order,
            customer01.dbo.sp_order_line_item,
            sp_card
WHERE customer01.dbo.sp_order_line_item.order_id
      = customer01.dbo.sp_order.order_id
      AND
      customer01.dbo.sp_order_line_item.sku
      = sp_card.sku AND
      sp_card.visual_objects LIKE
      '%heart%'
GROUP BY customer_id
```

```
HAVING COUNT(sp_card.sku)
> 10)
```

Result:

View created successfully

18)

Run From: sp_customer

Returns: This query creates a view that consists of two columns: the first column is a list of the customer ids from the heart_lovers2 view. The second column consists of the total number of cards that were bought by that customer.

```
CREATE VIEW who_lovers AS (SELECT HEART_LOVERS2.customer_id,
                                COUNT(sp_card.sku)
                                AS no_of_total_cards
FROM HEART_LOVERS2,
     customer01.dbo.sp_order,
     customer01.dbo.sp_order_line_item,
     sp_card
WHERE customer01.dbo.sp_order_line_item.order_id
      = customer01.dbo.sp_order.order_id
      AND
      customer01.dbo.sp_order_line_item.sku
      = sp_card.sku AND
      HEART_LOVERS2.customer_id = customer01.dbo.sp_order.customer_id
GROUP BY HEART_LOVERS2.customer_id)
```

Result:

View created successfully.

19)

Run From: sp_customer

Returns: This query returns 3 columns: the customer id, the number of heart cards that they bought (if greater than 10), and a list of the total number of cards that they got.

```
SELECT heart_lovers2.customer_id, heart_lovers2.no_of_cards,
       who_lovers.total_no_of_cards
FROM heart_lovers2, who_lovers
WHERE heart_lovers2.customer_id = who_lovers.customer_id
```

Result:

customer_id	no_of_cards	no_of_total_cards
14965	20	283
1894	24	440

65447	26	370
2275	36	257
2452	11	83
29050	21	248
1889	12	164
133938	15	44
112096	11	18
163314	16	45
2792	12	70

(11 row(s) affected)

20)

Run From: sp_customer

Returns: The above query can be slightly modified to find out the percentage of heart cards that they got out of the total number of cards.

```
SELECT heart_lovers.customer_id, heart_lovers.no_of_cards,
       who_lovers.no_of_total_cards,
       (heart_lovers.no_of_cards * 100 / who_lovers.no_of_total_cards)
       AS percentage
FROM heart_lovers, who_lovers
WHERE heart_lovers.customer_id = who_lovers.customer_id
```

Result:

customer_id	no_of_cards	no_of_total_cards	percentage
-----	-----	-----	-----
14965	20	283	7
1894	24	440	5
65447	26	370	7
2275	36	257	14
2452	11	83	13
29050	21	248	8
1889	12	164	7
133938	15	44	34
112096	11	18	61
163314	16	45	35
2792	12	70	17

(11 row(s) affected)

21)

Run From: sp_customer

Returns: creates a view that returns for each card, the customer-ids of the customer who bought that card.

```
CREATE VIEW collab_view AS SELECT customer01.dbo.SP_ORDER_LINE_ITEM.sku
    AS num
    FROM customer01.dbo.SP_ORDER_LINE_ITEM,
    customer01.dbo.[SP_ORDER]
    WHERE customer01.[dbo].[SP_ORDER_LINE_ITEM].order_id
    = customer01.[dbo].[SP_ORDER].order_id
    GROUP BY customer01.dbo.SP_ORDER_LINE_ITEM.sku
```

Result:

22)

Run From: sp_customer

Returns: creates a view that returns for each card, the total number of customers who have purchased that card.

```
CREATE VIEW count_cards AS SELECT customer01.dbo.SP_ORDER_LINE_ITEM.sku
    AS num,
    COUNT(customer01.[dbo].[SP_ORDER].customer_id)
    AS cnt
    FROM customer01.dbo.SP_ORDER_LINE_ITEM,
    customer01.dbo.[SP_ORDER]
    WHERE customer01.[dbo].[SP_ORDER_LINE_ITEM].order_id
    = customer01.[dbo].[SP_ORDER].order_id
    GROUP BY customer01.dbo.SP_ORDER_LINE_ITEM.sku

SELECT *
FROM COUNT_CARDS
```

Result:

num	cnt
0003449	10
0000456	31
0006913	1
0008466	7
0010456	50
0002096	17
6000007	4
(and so on...)	

23)

Run From: sp_customer

Returns: the cardid which has been bought by the largest number of customers.

```
SELECT *
FROM count_cards
WHERE cnt =
      (SELECT MAX(CNT)
       FROM count_cards)
```

Result:

num	cnt
8000029	338

(1 row(s) affected)

Inserts and Updates

24)

Run From: sp_wpi_card

Returns: This query inserts the sku and foldID attributes from sp_card table into sp_wpi_card table.

```
INSERT INTO sp_wpi_card
      (sku, foldID)
SELECT sku, fold_id
FROM catalog01.dbo.sp_card
```

25)

Run From: sp_wpi_card

Returns: This query selects all the listed attributes data from sp_card table and inserts them into the sp_wpi_card database.

```
INSERT INTO sp_wpi_card
      (sku, lineID, charityID, inside_finishID, outside_finishID, foldID,
       mediumID, paperID, personalizeID)
SELECT a.sku, line_ID, charity_ID, inside_finish_ID,
       outside_finish_ID, fold_ID, medium_ID, paper_ID,
       personalize_ID
FROM catalog01.dbo.sp_card a,
     catalog01.dbo.sp_catalog b
WHERE ((b.prod_type_id = 1) AND (a.sku = b.sku)) AND
```

```
(b.catalog_status_name = 'ACTV') AND
(b.qa_status_name = 'DE Complete')
```

/// prodtype = card, status is active, and data entry complete.

26)

Run From: sp_wpi_card

Returns: Once all the other card attributes have been added, to add the price from sp_catalog database

```
UPDATE sp_wpi_card
SET sp_wpi_card.price = catalog01.dbo.sp_catalog.price
FROM catalog01.dbo.sp_catalog
WHERE sp_wpi_card.sku = catalog01.dbo.sp_catalog.sku
```

27)

Run From: sp_card

Returns: This is an approach to find the total number of cards, which seem to have a happy theme. (By adding OR statements to the WHERE condition, more synonyms of the word happy can be added to find out the exact number of cards which are happy, joyful, smily etc.

```
SELECT COUNT(*)
FROM [dbo].[SP_CARD]
WHERE adjectives_themes LIKE '%Happy%' OR
      visual_actions LIKE '%Happy%' OR
      visual_objects LIKE '%Happy%' OR
      inside_text LIKE '%Happy%' OR
      back_text LIKE '%Happy%' OR
      front_text LIKE '%Happy%'
```

An initial approximate was found for the different types of cards, using queries described above:

Sentiments	3-attributes	6-attributes	synonyms
Depressed/Dejected/Low/Sorrow/	10	11	41
Sad/Unhappy/Melancholy/Cynical	58	81	84

Happy/Joy/Glad/Delighted	438	3317	3755
Silly/Goofy/Comical/Satire	149	159	165
Joke/Jest/Crack	86	93	131
Funny/Amusing/Entertaining/ Humorous	536	550	1428
Hilarious/Lively/Excited/Ecstatic	3	3	7

28)

Run From: sp_wpi_card

Returns: combines the 13 card attributes and adds them to a cardid attribute for each card in the database.

```

UPDATE SP_WPI_xmas_cards
SET CARDID = CAST(LineID AS Varchar(5))
+ '/' + CAST(FoldID AS Varchar(5))
+ '/' + CAST(MediumID AS Varchar(5))
+ '/' + CAST(PaperID AS Varchar(5))
+ '/' + CAST(DesignerID AS Varchar(5))
+ '/' + CAST(Envelope_color AS Varchar(5))
+ '/' + CAST(Envelope_width AS Varchar(5))
+ '/' + CAST(Envelope_height AS Varchar(5))
+ '/' + CAST(Humorous AS Varchar(5))
+ '/' + CAST([A Little Naughty] AS Varchar(5))
+ '/' + CAST(Sentimental AS Varchar(5))
+ '/' + CAST(Cynical AS Varchar(5))
+ '/' + CAST([New Age/Mystical] AS Varchar(5))
+ '/' + CAST(Romantic AS Varchar(5))
+ '/' + CAST([Religious/Spiritual] AS Varchar(5))
+ '/' + CAST(Formal AS Varchar(5))
+ '/' + CAST(Beautiful AS Varchar(5))
+ '/' + CAST([Hip/Trendy] AS Varchar(5))
+ '/' + CAST([Sophisticated/Elegant] AS Varchar(5))
+ '/' + CAST(General AS Varchar(5))
+ '/' + CAST(Sincere AS Varchar(5))
+ '/' + CAST(Suggestive AS Varchar(5))
+ '/' + CAST([Adults Only] AS Varchar(5))
+ '/' + CAST(Fun AS Varchar(5))

```

```
+ '/' + CAST(PersonalizeID AS Varchar(5))
+ '/' + CAST(Price AS Varchar(5)) + '#'
```

Result:

29)

Run From: sp_wpi_card

Returns: enters the sentiment rating for each card in the database under the attributes sentiment_1, sentiment_2 and sentiment_3

/*** Sentiment attribute ***/

Number	Word
0	Emotion of card could not be deduced
1	Depressed, dejected, low, sorrow, grief, anguish, pain, remorse, distress, misery, woe, suffering, mourne, weep, bewail, repent, despondent, disconsolate
2	Sad, unhappy, melancholic, gloom, dismal, downhearted, dispirited, morose
3	Serious, grave, solemn, somber, stern, grim, severe, sober,
4	Cynical, unfriendly, hostile, alienated, vicious, malicious, venomous, oppose
7	Laugh, smile, chuckle, jest, crack, sparkle, joke, silly, goofy, comic, satire, funny, humor, hilar, amusing, entertaining, lively, excited
8	Happy, joy, glad, cheerful, bliss, contend
9	ecstatic, vivacious, animated, rapt, exhilarated

Number	Word
1	Hate, disgust, revul, detest, odium, abhor, dislike, averse, annoy
9	Love, heart, sappy, romance, longing, kiss, sweet, missing, affection, fond, passion, like, hug, loving, sharing

Query used for the initial approach of dividing cards into different sentiment groups on the basis of the words in the card. This approach was abandoned since Sparks.com already stores the tones

of the cards and their relevance. However, this can be a possible approach for databases that do not store card tone/relevance information.

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%depressed%')
or (b.visual_objects like '%depressed%') or (b.adjectives_themes like '%depressed%')
or (b.back_text like '%depressed%') or (b.inside_text like '%depressed%') or
(b.front_text like '%depressed%'))))
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%dejected%')
or (b.visual_objects like '%dejected%') or (b.adjectives_themes like '%dejected%')
or (b.back_text like '%dejected%') or (b.inside_text like '%dejected%') or
(b.front_text like '%dejected%'))))
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%low%')
or (b.visual_objects like '%low%') or (b.adjectives_themes like '%low%')
or (b.back_text like '%low%') or (b.inside_text like '%low%') or
(b.front_text like '%low%'))))
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%sorrow%')
or (b.visual_objects like '%sorrow%') or (b.adjectives_themes like '%sorrow%')
or (b.back_text like '%sorrow%') or (b.inside_text like '%sorrow%') or
(b.front_text like '%sorrow%'))))
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%anguish%')
or (b.visual_objects like '%anguish%') or (b.adjectives_themes like '%anguish%')
or (b.back_text like '%anguish%') or (b.inside_text like '%anguish%') or
(b.front_text like '%anguish%'))))
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%depressed%')
or (b.visual_objects like '%depressed%') or (b.adjectives_themes like '%depressed%')
or (b.back_text like '%depressed%') or (b.inside_text like '%depressed%') or
(b.front_text like '%depressed%'))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%grief%')
or (b.visual_objects like '%grief%') or (b.adjectives_themes like '%grief%')
or (b.back_text like '%grief%') or (b.inside_text like '%grief%') or
(b.front_text like '%grief%'))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%repent%')
or (b.visual_objects like '%repent%') or (b.adjectives_themes like '%repent%')
or (b.back_text like '%repent%') or (b.inside_text like '%repent%') or
(b.front_text like '%repent%'))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%dismal%')
or (b.visual_objects like '%dismal%') or (b.adjectives_themes like '%dismal%')
or (b.back_text like '%dismal%') or (b.inside_text like '%dismal%') or
(b.front_text like '%dismal%'))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%pain%')
or (b.visual_objects like '%pain%') or (b.adjectives_themes like '%pain%')
or (b.back_text like '%pain%') or (b.inside_text like '%pain%') or
(b.front_text like '%pain%'))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%remorse%')
or (b.visual_objects like '%remorse%') or (b.adjectives_themes like '%remorse%')
or (b.back_text like '%remorse%') or (b.inside_text like '%remorse%') or
(b.front_text like '%remorse%'))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%distress%')
or (b.visual_objects like '%distress%') or (b.adjectives_themes like '%distress%')
or (b.back_text like '%distress%') or (b.inside_text like '%distress%') or
(b.front_text like '%distress%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%misery%')
or (b.visual_objects like '%misery%') or (b.adjectives_themes like '%misery%')
or (b.back_text like '%misery%') or (b.inside_text like '%misery%') or
(b.front_text like '%misery%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%woe%')
or (b.visual_objects like '%woe%') or (b.adjectives_themes like '%woe%')
or (b.back_text like '%woe%') or (b.inside_text like '%woe%') or
(b.front_text like '%woe%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%suffering%')
or (b.visual_objects like '%suffering%') or (b.adjectives_themes like '%suffering%')
or (b.back_text like '%suffering%') or (b.inside_text like '%suffering%') or
(b.front_text like '%suffering%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%bewail%')
or (b.visual_objects like '%bewail%') or (b.adjectives_themes like '%bewail%')
or (b.back_text like '%bewail%') or (b.inside_text like '%bewail%') or
(b.front_text like '%bewail%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%despondent%')
or (b.visual_objects like '%despondent%') or (b.adjectives_themes like '%despondent%')
or (b.back_text like '%despondent%') or (b.inside_text like '%despondent%') or

```

```
(b.front_text like '%despondent%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 2 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%sad%')
```

```
or (b.visual_objects like '%sad%') or (b.adjectives_themes like '%sad%')
```

```
or (b.back_text like '%sad%') or (b.inside_text like '%sad%') or
```

```
(b.front_text like '%sad%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 2 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%unhappy%')
```

```
or (b.visual_objects like '%unhappy%') or (b.adjectives_themes like '%unhappy%')
```

```
or (b.back_text like '%unhappy%') or (b.inside_text like '%unhappy%') or
```

```
(b.front_text like '%unhappy%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 2 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%gloom%')
```

```
or (b.visual_objects like '%gloom%') or (b.adjectives_themes like '%gloom%')
```

```
or (b.back_text like '%gloom%') or (b.inside_text like '%gloom%') or
```

```
(b.front_text like '%gloom%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 2 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%melanchol%')
```

```
or (b.visual_objects like '%melanchol%') or (b.adjectives_themes like '%melanchol%')
```

```
or (b.back_text like '%melanchol%') or (b.inside_text like '%melanchol%') or
```

```
(b.front_text like '%melanchol%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 2 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%downhearted%')
```

```
or (b.visual_objects like '%downhearted%') or (b.adjectives_themes like '%downhearted%')
```

```
or (b.back_text like '%downhearted%') or (b.inside_text like '%downhearted%') or
```

```
(b.front_text like '%downhearted%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 2 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%dispirited%')
```

```

or (b.visual_objects like '%dispirited%') or (b.adjectives_themes like '%dispirited%')
or (b.back_text like '%dispirited%') or (b.inside_text like '%dispirited%') or
(b.front_text like '%dispirited%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 3 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%serious%')
or (b.visual_objects like '%serious%') or (b.adjectives_themes like '%serious%')
or (b.back_text like '%serious%') or (b.inside_text like '%serious%') or
(b.front_text like '%serious%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 3 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%sober%')
or (b.visual_objects like '%sober%') or (b.adjectives_themes like '%sober%')
or (b.back_text like '%sober%') or (b.inside_text like '%sober%') or
(b.front_text like '%sober%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 3 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%somber%')
or (b.visual_objects like '%somber%') or (b.adjectives_themes like '%somber%')
or (b.back_text like '%somber%') or (b.inside_text like '%somber%') or
(b.front_text like '%somber%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 3 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%grim%')
or (b.visual_objects like '%grim%') or (b.adjectives_themes like '%grim%')
or (b.back_text like '%grim%') or (b.inside_text like '%grim%') or
(b.front_text like '%grim%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 3 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b

```

```

WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%grave%')
or (b.visual_objects like '%grave%') or (b.adjectives_themes like '%grave%')
or (b.back_text like '%grave%') or (b.inside_text like '%grave%') or
(b.front_text like '%grave%'))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 3 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%stern%')
or (b.visual_objects like '%stern%') or (b.adjectives_themes like '%stern%')
or (b.back_text like '%stern%') or (b.inside_text like '%stern%') or
(b.front_text like '%stern%'))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 3 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%severe%')
or (b.visual_objects like '%severe%') or (b.adjectives_themes like '%severe%')
or (b.back_text like '%severe%') or (b.inside_text like '%severe%') or
(b.front_text like '%severe%'))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 4 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%cynical%')
or (b.visual_objects like '%cynical%') or (b.adjectives_themes like '%cynical%')
or (b.back_text like '%cynical%') or (b.inside_text like '%cynical%') or
(b.front_text like '%cynical%'))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 4 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%unfriendly%')
or (b.visual_objects like '%unfriendly%') or (b.adjectives_themes like '%unfriendly%')
or (b.back_text like '%unfriendly%') or (b.inside_text like '%unfriendly%') or
(b.front_text like '%unfriendly%'))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 4 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%hostile%')

```



```
or (b.visual_objects like '%hostile%') or (b.adjectives_themes like '%hostile%')
or (b.back_text like '%hostile%') or (b.inside_text like '%hostile%') or
(b.front_text like '%hostile%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 4 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%alienated%')
```

```
or (b.visual_objects like '%alienated%') or (b.adjectives_themes like '%alienated%')
```

```
or (b.back_text like '%alienated%') or (b.inside_text like '%alienated%') or
```

```
(b.front_text like '%alienated%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 4 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%vicious%')
```

```
or (b.visual_objects like '%vicious%') or (b.adjectives_themes like '%vicious%')
```

```
or (b.back_text like '%vicious%') or (b.inside_text like '%vicious%') or
```

```
(b.front_text like '%vicious%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 4 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%venomous%')
```

```
or (b.visual_objects like '%venomous%') or (b.adjectives_themes like '%venomous%')
```

```
or (b.back_text like '%venomous%') or (b.inside_text like '%venomous%') or
```

```
(b.front_text like '%venomous%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 4 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%malicious%')
```

```
or (b.visual_objects like '%malicious%') or (b.adjectives_themes like '%malicious%')
```

```
or (b.back_text like '%malicious%') or (b.inside_text like '%malicious%') or
```

```
(b.front_text like '%malicious%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 4 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%oppose%')
```

```
or (b.visual_objects like '%oppose%') or (b.adjectives_themes like '%oppose%')
```

```
or (b.back_text like '%oppose%') or (b.inside_text like '%oppose%') or
```

```
(b.front_text like '%oppose%'))))
```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%laugh%')
or (b.visual_objects like '%laugh%') or (b.adjectives_themes like '%laugh%')
or (b.back_text like '%laugh%') or (b.inside_text like '%laugh%') or
(b.front_text like '%laugh%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%delighted%')
or (b.visual_objects like '%smile%') or (b.adjectives_themes like '%smile%')
or (b.back_text like '%smile%') or (b.inside_text like '%smile%') or
(b.front_text like '%smile%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%chuckle%')
or (b.visual_objects like '%chuckle%') or (b.adjectives_themes like '%chuckle%')
or (b.back_text like '%chuckle%') or (b.inside_text like '%chuckle%') or
(b.front_text like '%chuckle%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%jest%')
or (b.visual_objects like '%jest%') or (b.adjectives_themes like '%jest%')
or (b.back_text like '%jest%') or (b.inside_text like '%jest%') or
(b.front_text like '%jest%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%crack%')
or (b.visual_objects like '%crack%') or (b.adjectives_themes like '%crack%')
or (b.back_text like '%crack%') or (b.inside_text like '%crack%') or
(b.front_text like '%crack%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%sparkle%')
or (b.visual_objects like '%sparkle%') or (b.adjectives_themes like '%sparkle%')
or (b.back_text like '%sparkle%') or (b.inside_text like '%sparkle%') or

```

```
(b.front_text like '%sparkle%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%joke%')
```

```
or (b.visual_objects like '%joke%') or (b.adjectives_themes like '%joke%')
```

```
or (b.back_text like '%joke%') or (b.inside_text like '%joke%') or
```

```
(b.front_text like '%joke%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%silly%')
```

```
or (b.visual_objects like '%silly%') or (b.adjectives_themes like '%silly%')
```

```
or (b.back_text like '%silly%') or (b.inside_text like '%silly%') or
```

```
(b.front_text like '%silly%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%goofy%')
```

```
or (b.visual_objects like '%goofy%') or (b.adjectives_themes like '%goofy%')
```

```
or (b.back_text like '%goofy%') or (b.inside_text like '%goofy%') or
```

```
(b.front_text like '%goofy%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%comic%')
```

```
or (b.visual_objects like '%comic%') or (b.adjectives_themes like '%comic%')
```

```
or (b.back_text like '%comic%') or (b.inside_text like '%comic%') or
```

```
(b.front_text like '%comic%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%satire%')
```

```
or (b.visual_objects like '%satire%') or (b.adjectives_themes like '%satire%')
```

```
or (b.back_text like '%satire%') or (b.inside_text like '%satire%') or
```

```
(b.front_text like '%satire%'))))
```

```
GO
```

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%funny%')
or (b.visual_objects like '%funny%') or (b.adjectives_themes like '%funny%')
or (b.back_text like '%funny%') or (b.inside_text like '%funny%') or
(b.front_text like '%funny%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%humor%')
or (b.visual_objects like '%humor%') or (b.adjectives_themes like '%humor%')
or (b.back_text like '%humor%') or (b.inside_text like '%humor%') or
(b.front_text like '%humor%'))))
```

```
or (b.visual_objects like '%humor%') or (b.adjectives_themes like '%humor%')
or (b.back_text like '%humor%') or (b.inside_text like '%humor%') or
(b.front_text like '%humor%'))))
```

```
or (b.back_text like '%humor%') or (b.inside_text like '%humor%') or
(b.front_text like '%humor%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%hilar%')
or (b.visual_objects like '%hilar%') or (b.adjectives_themes like '%hilar%')
or (b.back_text like '%hilar%') or (b.inside_text like '%hilar%') or
(b.front_text like '%hilar%'))))
```

```
or (b.visual_objects like '%hilar%') or (b.adjectives_themes like '%hilar%')
or (b.back_text like '%hilar%') or (b.inside_text like '%hilar%') or
(b.front_text like '%hilar%'))))
```

```
or (b.back_text like '%hilar%') or (b.inside_text like '%hilar%') or
(b.front_text like '%hilar%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%amus%')
or (b.visual_objects like '%amus%') or (b.adjectives_themes like '%amus%')
or (b.back_text like '%amus%') or (b.inside_text like '%amus%') or
(b.front_text like '%amus%'))))
```

```
or (b.visual_objects like '%amus%') or (b.adjectives_themes like '%amus%')
or (b.back_text like '%amus%') or (b.inside_text like '%amus%') or
(b.front_text like '%amus%'))))
```

```
or (b.back_text like '%amus%') or (b.inside_text like '%amus%') or
(b.front_text like '%amus%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%fun%')
or (b.visual_objects like '%fun%') or (b.adjectives_themes like '%fun%')
or (b.back_text like '%fun%') or (b.inside_text like '%fun%') or
(b.front_text like '%fun%'))))
```

```
or (b.visual_objects like '%fun%') or (b.adjectives_themes like '%fun%')
or (b.back_text like '%fun%') or (b.inside_text like '%fun%') or
(b.front_text like '%fun%'))))
```

```
or (b.back_text like '%fun%') or (b.inside_text like '%fun%') or
(b.front_text like '%fun%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```
(SELECT a.sku
```

```
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
```

```
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%entertain%')
or (b.visual_objects like '%entertain%') or (b.adjectives_themes like '%entertain%')
or (b.back_text like '%entertain%') or (b.inside_text like '%entertain%') or
(b.front_text like '%entertain%'))))
```

```
or (b.visual_objects like '%entertain%') or (b.adjectives_themes like '%entertain%')
or (b.back_text like '%entertain%') or (b.inside_text like '%entertain%') or
(b.front_text like '%entertain%'))))
```

```
or (b.back_text like '%entertain%') or (b.inside_text like '%entertain%') or
(b.front_text like '%entertain%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
```

```

(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%lively%')
or (b.visual_objects like '%lively%') or (b.adjectives_themes like '%lively%')
or (b.back_text like '%lively%') or (b.inside_text like '%lively%') or
(b.front_text like '%lively%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 7 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%excite%')
or (b.visual_objects like '%excite%') or (b.adjectives_themes like '%excite%')
or (b.back_text like '%excite%') or (b.inside_text like '%excite%') or
(b.front_text like '%excite%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%happy%')
or (b.visual_objects like '%happy%') or (b.adjectives_themes like '%happy%')
or (b.back_text like '%happy%') or (b.inside_text like '%happy%') or
(b.front_text like '%happy%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%joy%')
or (b.visual_objects like '%joy%') or (b.adjectives_themes like '%joy%')
or (b.back_text like '%joy%') or (b.inside_text like '%joy%') or
(b.front_text like '%joy%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%glad%')
or (b.visual_objects like '%glad%') or (b.adjectives_themes like '%glad%')
or (b.back_text like '%glad%') or (b.inside_text like '%glad%') or
(b.front_text like '%glad%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%happy%')

```

```

or (b.visual_objects like '%happy%') or (b.adjectives_themes like '%happy%')
or (b.back_text like '%happy%') or (b.inside_text like '%happy%') or
(b.front_text like '%happy%'))))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN

```

```

(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b

```

```

WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%joy%')

```

```

or (b.visual_objects like '%joy%') or (b.adjectives_themes like '%joy%')

```

```

or (b.back_text like '%joy%') or (b.inside_text like '%joy%') or

```

```

(b.front_text like '%joy%'))))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN

```

```

(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b

```

```

WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%cheer%')

```

```

or (b.visual_objects like '%cheer%') or (b.adjectives_themes like '%cheer%')

```

```

or (b.back_text like '%cheer%') or (b.inside_text like '%cheer%') or

```

```

(b.front_text like '%cheer%'))))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN

```

```

(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b

```

```

WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%happy%')

```

```

or (b.visual_objects like '%happy%') or (b.adjectives_themes like '%happy%')

```

```

or (b.back_text like '%happy%') or (b.inside_text like '%happy%') or

```

```

(b.front_text like '%happy%'))))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN

```

```

(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b

```

```

WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%joy%')

```

```

or (b.visual_objects like '%joy%') or (b.adjectives_themes like '%joy%')

```

```

or (b.back_text like '%joy%') or (b.inside_text like '%joy%') or

```

```

(b.front_text like '%joy%'))))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN

```

```

(SELECT a.sku

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b

```

```

WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%contend%')

```

```

or (b.visual_objects like '%contend%') or (b.adjectives_themes like '%contend%')

```

```

or (b.back_text like '%contend%') or (b.inside_text like '%contend%') or

```

```

(b.front_text like '%contend%'))))

```

GO

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN

```

```

(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%happy%')
or (b.visual_objects like '%happy%') or (b.adjectives_themes like '%happy%')
or (b.back_text like '%happy%') or (b.inside_text like '%happy%') or
(b.front_text like '%happy%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%joy%')
or (b.visual_objects like '%joy%') or (b.adjectives_themes like '%joy%')
or (b.back_text like '%joy%') or (b.inside_text like '%joy%') or
(b.front_text like '%joy%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 8 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%bliss%')
or (b.visual_objects like '%bliss%') or (b.adjectives_themes like '%bliss%')
or (b.back_text like '%bliss%') or (b.inside_text like '%bliss%') or
(b.front_text like '%bliss%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%rapt%')
or (b.visual_objects like '%rapt%') or (b.adjectives_themes like '%rapt%')
or (b.back_text like '%rapt%') or (b.inside_text like '%rapt%') or
(b.front_text like '%rapt%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%ecsta%')
or (b.visual_objects like '%ecsta%') or (b.adjectives_themes like '%ecsta%')
or (b.back_text like '%ecsta%') or (b.inside_text like '%ecsta%') or
(b.front_text like '%ecsta%'))))
GO
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%animated%')
or (b.visual_objects like '%animated%') or (b.adjectives_themes like '%animated%')

```

```
or (b.back_text like '%animated%') or (b.inside_text like '%animated%') or
(b.front_text like '%animated%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%vivacious%')
or (b.visual_objects like '%vivacious%') or (b.adjectives_themes like '%vivacious%')
or (b.back_text like '%vivacious%') or (b.inside_text like '%vivacious%') or
(b.front_text like '%vivacious%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_1 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%exhila%')
or (b.visual_objects like '%exhila%') or (b.adjectives_themes like '%exhila%')
or (b.back_text like '%exhila%') or (b.inside_text like '%exhila%') or
(b.front_text like '%exhila%'))))
```

GO

.....

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%hate%')
or (b.visual_objects like '%hate%') or (b.adjectives_themes like '%hate%')
or (b.back_text like '%hate%') or (b.inside_text like '%hate%') or
(b.front_text like '%hate%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%detest%')
or (b.visual_objects like '%detest%') or (b.adjectives_themes like '%detest%')
or (b.back_text like '%detest%') or (b.inside_text like '%detest%') or
(b.front_text like '%detest%'))))
```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%odium%')
or (b.visual_objects like '%odium%') or (b.adjectives_themes like '%odium%'))
```



```

or (b.back_text like '%odium%') or (b.inside_text like '%odium%') or
(b.front_text like '%odium%'))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%abhor%')
or (b.visual_objects like '%abhor%') or (b.adjectives_themes like '%abhor%')
or (b.back_text like '%abhor%') or (b.inside_text like '%abhor%') or
(b.front_text like '%abhor%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%dislike%')
or (b.visual_objects like '%dislike%') or (b.adjectives_themes like '%dislike%')
or (b.back_text like '%dislike%') or (b.inside_text like '%dislike%') or
(b.front_text like '%dislike%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%averse%')
or (b.visual_objects like '%averse%') or (b.adjectives_themes like '%averse%')
or (b.back_text like '%averse%') or (b.inside_text like '%averse%') or
(b.front_text like '%averse%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%revul%')
or (b.visual_objects like '%revul%') or (b.adjectives_themes like '%revul%')
or (b.back_text like '%revul%') or (b.inside_text like '%revul%') or
(b.front_text like '%revul%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 1 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%disgust%')

```

```

or (b.visual_objects like '%disgust%') or (b.adjectives_themes like '%disgust%')
or (b.back_text like '%disgust%') or (b.inside_text like '%disgust%') or
(b.front_text like '%disgust%'))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%sappy%')
or (b.visual_objects like '%sappy%') or (b.adjectives_themes like '%sappy%')
or (b.back_text like '%sappy%') or (b.inside_text like '%sappy%') or
(b.front_text like '%sappy%'))))

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%sweet%')
or (b.visual_objects like '%sweet%') or (b.adjectives_themes like '%sweet%')
or (b.back_text like '%sweet%') or (b.inside_text like '%sweet%') or
(b.front_text like '%sweet%'))), catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%love%')
or (b.visual_objects like '%love%') or (b.adjectives_themes like '%love%')
or (b.back_text like '%love%') or (b.inside_text like '%love%') or
(b.front_text like '%love%'))))

```

```

FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%romance%')
or (b.visual_objects like '%romance%') or (b.adjectives_themes like '%romance%')
or (b.back_text like '%romance%') or (b.inside_text like '%romance%') or
(b.front_text like '%romance%'))), catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%missing%')
or (b.visual_objects like '%missing%') or (b.adjectives_themes like '%missing%')
or (b.back_text like '%missing%') or (b.inside_text like '%missing%') or
(b.front_text like '%missing%'))), catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%longing%')
or (b.visual_objects like '%longing%') or (b.adjectives_themes like '%longing%')
or (b.back_text like '%longing%') or (b.inside_text like '%longing%') or
(b.front_text like '%longing%'))), catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%hug%')
or (b.visual_objects like '%hug%') or (b.adjectives_themes like '%hug%')
or (b.back_text like '%hug%') or (b.inside_text like '%hug%') or
(b.front_text like '%hug%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%hug%')
or (b.visual_objects like '%hug%') or (b.adjectives_themes like '%hug%')
or (b.back_text like '%hug%') or (b.inside_text like '%hug%') or
(b.front_text like '%hug%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%loving%')
or (b.visual_objects like '%loving%') or (b.adjectives_themes like '%loving%')
or (b.back_text like '%loving%') or (b.inside_text like '%loving%') or
(b.front_text like '%loving%'))))
GO

```

Page missing in
original

IQP/MQP SCANNING PROJECT



George C. Gordon Library
WORCESTER POLYTECHNIC INSTITUTE

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%cute%')
or (b.visual_objects like '%cute%') or (b.adjectives_themes like '%cute%')
or (b.back_text like '%cute%') or (b.inside_text like '%cute%') or
(b.front_text like '%cute%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%shar%')
or (b.visual_objects like '%shar%') or (b.adjectives_themes like '%shar%')
or (b.back_text like '%shar%') or (b.inside_text like '%shar%') or
(b.front_text like '%shar%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%affection%')
or (b.visual_objects like '%affection%') or (b.adjectives_themes like '%affection%')
or (b.back_text like '%affection%') or (b.inside_text like '%affection%') or
(b.front_text like '%affection%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%fond%')
or (b.visual_objects like '%fond%') or (b.adjectives_themes like '%fond%')
or (b.back_text like '%fond%') or (b.inside_text like '%fond%') or
(b.front_text like '%fond%'))))
GO

```

```

UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%passion%')
or (b.visual_objects like '%passion%') or (b.adjectives_themes like '%passion%')
or (b.back_text like '%passion%') or (b.inside_text like '%passion%') or
(b.front_text like '%passion%'))))

```

GO

```
UPDATE wpi.dbo.sp_wpi_card SET Sentiment_2 = 9 where SKU IN
(SELECT a.sku
FROM wpi.dbo.sp_wpi_card a, catalog01.dbo.sp_card b
WHERE (a.sku = b.sku) AND ((b.visual_actions LIKE '%like%')
or (b.visual_objects like '%like%') or (b.adjectives_themes like '%like%')
or (b.back_text like '%like%') or (b.inside_text like '%like%') or
(b.front_text like '%like%'))))
GO
```

26)

Run From: sp_wpi_customer

Returns: extracts the attributes from the original customer database and inserts them into the sp_wpi_customer database

```
/** Customer Table */
INSERT INTO SP_WPI_CUSTOMER
    (CustomerID,birthday,gender)
SELECT Customer_ID,Birthday,Gender
FROM CUSTOMER01.dbo.SP_CUSTOMER
GO
```

30)

Run From: sp_wpi_order

Returns: extracts the information from the original database and inserts it into the sp_wpi_order database

```
INSERT INTO SP_WPI_ORDER
(customerid, sku)
SELECT customer01.dbo.SP_ORDER.customer_id AS customerid,
       customer01.[dbo].SP_ORDER_LINE_ITEM.sku AS sku
FROM customer01.dbo.SP_ORDER_LINE_ITEM,
       customer01.dbo.[SP_ORDER]
WHERE customer01.[dbo].[SP_ORDER_LINE_ITEM].order_id =
       customer01.[dbo].[SP_ORDER].order_id
ORDER BY customer01.dbo.SP_ORDER.customer_id
```

31)

Run From: sp_wpi_order

Returns: returns a 4-column table, customerid, sku, another customer id, same sku

```
CREATE VIEW collab_grouping AS SELECT a.CustomerID AS C1,
                                     b.CustomerID AS C2,
                                     a.sku
FROM [dbo].[SP_WPI_ORDER] a,
     [dbo].[SP_WPI_ORDER] b
```

32)

Run From: sp_wpi_order

Returns: For a given customer id, this finds out on how many cards do the other customers agree with this customer id.

```
SELECT CustomerID
FROM sp_wpi_ORDER
WHERE sku IN
      (SELECT sku
       FROM SP_WPI_ORDER A
       WHERE A.CustomerID = 1934)
```

33)

Run From: sp_wpi_order

Returns: This query says that for each number, how many customers bought that number of cards.

```
CREATE VIEW avg_cards2 AS SELECT COUNT(*) AS num
```

```
FROM [dbo].[SP_WPI_ORDER]
GROUP BY customerID
```

```
SELECT DISTINCT (a.num), COUNT(a.num)
FROM avg_cards a
WHERE a.num IN
    (SELECT DISTINCT (b.num)
    FROM avg_cards b)
GROUP BY a.num
ORDER BY a.num
```

34)

Run From: sp_wpi_card

Returns: All the attributes of the cards, which were not classified, by the sentiment attribute

```
SELECT *
FROM catalog01.dbo.sp_card
WHERE sku IN
    (SELECT sku
    FROM [dbo].[SP_WPI_CARD]
    WHERE sentiment_1 = 0 AND sentiment_2 = 0)
```

35)

Runs from: sp_wpi_card

Returns: the count of cards that are not classified by sentiments.

```
SELECT COUNT(*)
FROM sp_wpi_card
WHERE sentiment_1 = 0 AND sentiment_2 = 0
```

36)

Runs from: sp_wpi_card

Returns: Sets the back_text columns containing no information to be XXXBlankXXX. This helps to maintain consistency while making the bag of words for each card.

```
UPDATE sp_wpi_card
SET sp_wpi_card.back_text = 'XXXBlankXXX'
WHERE sp_wpi_card.back_text LIKE " OR
    sp_wpi_card.back_text LIKE '%N/A%' OR
    sp_wpi_card.back_text LIKE 'No Text' OR
```

sp_wpi_card.back_text LIKE 'Blank'

37)

Runs from: sp_tone_card_map

Returns: The tone_id, sku and relevance of the cards that do not have their relevance as 0 (or not available).

```
SELECT tone_id, sku, relevance
FROM [dbo].[SP_TONE_CARD_MAP]
WHERE relevance != 0 AND sku IN
    (SELECT sku
     FROM wpi.dbo.sp_wpi_card)
```

38)

Runs from: sp_wpi_relevance

Returns: Inserts all the cards having relevance not equal to 0 into the sp_wpi_relevance table. These relevance attributes of the cards can be used for content-based filtering.

```
INSERT INTO wpi.dbo.sp_wpi_relevance
(sku, tone_id, relevance) (SELECT sku, tone_id,
    relevance
    FROM [dbo].[SP_TONE_CARD_MAP]
    WHERE relevance != 0 AND
    sku IN
    (SELECT sku
     FROM wpi.dbo.sp_wpi_card))
```

39)

Runs from: sp_wpi_xmas_cards

Returns: inserts the relevance information for all cards for the tone *fun*.

```
UPDATE SP_WPI_XMAS_CARDS
SET SP_WPI_XMAS_CARDS.Fun = sp_wpi_xmas_relevance.relevance
FROM sp_wpi_xmas_relevance
WHERE SP_WPI_XMAS_CARDS.sku = sp_wpi_xmas_relevance.sku
```


40)

Runs from: sp_wpi_bday_cards

Returns: inserts all the birthday cards into the database along with some of their attributes

```
INSERT INTO wpi.dbo.sp_wpi_bday_cards
(sku, lineID, charityID, inside_finishID, outside_finishID, foldID,
mediumID, paperID, personalizeID)
SELECT a.sku, line_ID, charity_ID, inside_finish_ID,
outside_finish_ID, fold_ID, medium_ID, paper_ID,
personalize_ID
FROM catalog01.dbo.sp_card a, catalog01.dbo.sp_catalog b,
SP_EVENT_CARD_MAP c
WHERE ((b.prod_type_id = 1) AND (a.sku = b.sku)) AND
(b.catalog_status_name = 'ACTV') AND
(b.qa_status_name = 'DE Complete') AND c.event_id = 2 AND
a.sku = c.sku AND relevance != 0
```

41)

Runs from: sp_wpi_bday_cards

Returns: Using queries such as below, the other attributes can be filled up in the sp_wpi_bday_cards table.

```
UPDATE sp_wpi_bday_cards
SET sp_wpi_bday_cards.back_text = catalog01.dbo.sp_card.back_text
FROM catalog01.dbo.sp_card
WHERE sp_wpi_bday_cards.sku = catalog01.dbo.sp_card.sku
```

42)

Runs from: sp_wpi_bday_cards

Returns: The tones of the cards for fun are inserted in the sp_wpi_bday_cards table.

```
UPDATE wpi.dbo.sp_wpi_bday_cards
SET wpi.dbo.sp_wpi_bday_cards.Fun = b.relevance
FROM SP_TONE_CARD_MAP b, sp_tone c
WHERE b.tone_id = c.tone_id AND
wpi.dbo.sp_wpi_bday_cards.sku = b.sku AND
c.tone_name LIKE '%fun%'
```

Appendix C.

Filtering System Pseudocode

FilteringMain

Variables List

- Class ContentSys – Instantiation of the content system
- Class CollabSys – Instantiation of the collaborative system
- Class ProfileSys – Instantiation of the profile maker system
- Vector UserIDs – List of all the user Ids
- Vector CardSKUs – List of all mapped Card SKUs
- Vector UserProfile – User's calculated content profile
- float[] collabResults – Collaborative prediction values in SKU order
- float[] contentResults – Content prediction values in SKU order
- float[] finalResults – Combination values in SKU order
- float contentWeight – Content weight value
- float collabWeight – Collaborative weight value
- int contentPlaces[] – Rankings for content predictions
- int collabPlaces[] – Rankings for collaborative predictions
- int netCollabError – Amount of placement error in collaborative system
- int ActiveUserID – ID of user logged in
- int maxSKU – Max value for a SKU
- int numUsers – Number of users

Public Functions

FilteringMain(Vector UserIDs, Vector CardSKUs)

// Initializes the entire filtering system

- if CardSKUs != NULL
 - maxSKU = CardSKUs.maxValueInDataStructure ()
 - else BREAK W/ ERROR
 - if UserIDs != NULL
 - numUsers = UserIDs.numberOfElements()
 - else BREAK W/ ERROR
 - new CollabSys = CollabSystem()
 - new ContentSys = ContentSystem()
 - new ProfileSys = ProfileSystem()
 - collabResults = new float[maxSKU + 1]
 - contentResults = new float[maxSKU + 1]
 - finalResults = new float[maxSKU + 1]
 - set ActiveUser = -1
- // No active user in system
- yet

SetActiveUserID(int user)

// Set the Active User ID

- if ActiveUserID == -1
 - set new ActiveUserID
- else BREAK W/ ERROR

int GetActiveUserID()

// Return the ActiveUserID

SetupContentSystem(Vector of Vector of ints for CardIDs [Card Profiles], Vector of floats for AttributeWeights, Vector of ints for AttributeTypes)

// Initializes the ContentSystem by passing all appropriate variables. Overwrites old setup data.

- ContentSys.SetupSystem(CardProfiles, AttributeWeights, AttributeTypes)

SetupCollabSystem(Vector of Vector of ints containing purchased CardSKUs)

// Initializes the CollaborativeSystem by passing all appropriate variables. Overwrites old setup data.

- CollabSys.SetupSystem(Vector of Vector of ints containing purchased CardSKUS, Vector UserIDs, int maxSKU, int numUsers)

SetupProfileMaker(Vector of ints for AttributeAVGTypes)

// Initializes the ProfileMaker by passing all appropriate variables. Overwrites old setup data.

- ProfileSys.SetupSystem(Vector AttributeAVGTypes)

CalculateContent(int UserID, Vector of ints for UserProfile, Vector of floats for WordRating)

// Ask ContentSystem to calculate content predictions for all cards & store predictions

- if UserID != ActiveUserID
 - BREAK w/ ERROR
- ContentSys.SetActiveUser(UserID, UserProfile, WordRating)
- ContentSys.Calculate()
- contentResults = ContentSys.GetPredictions()

CalculateCollab(int UserID)

// Ask CollaborativeSystem to calculate collaborative predictions for all cards & store predictions

- if UserID != ActiveUserID
 - BREAK w/ ERROR
- CollabSys.SetActiveUserInfo(int UserID)
- Benchmark(UserID)
- CollabSys.Calculate()
- collabResults = CollabSys.GetPredictions()

CalculateFinal(int UserID)

// Calculate the final prediction based on weighted content & collaborative predictions

- Vector UserPurchaseList // Hold user's purchases
- float weight // Temp weight storage
- int location // SKU location in array
- int contentError = 0 // Net content error based on past purchases
- int collabError = 0 // Net collab error based on past purchases
- int loopvar // Loop variable
- if(UserID != ActiveUserID)
 - BREAK w/ ERROR
- MakeRank(collabResults, collabPlaces, SKUs)
- MakeRank(contentResults, contentPlaces, SKUs)
- UserPurchaseList = CollabSys.GetUserPurchaseList() // Get purchase list
- while(UserPurchaseList still has elements) // Calculate absolute error from
 - 1st position
 - location = UserPurchaseList.currentElementValue()
 - contentError += contentPlaces[location] - 1
 - collabError += collabPlaces[location] - 1
- weight = collabError / (collabError + contentError)
- contentWeight = weight
- collabWeight = 1 - contentWeight
- for(loopvar = 1; loopvar <= maxSKU; loopvar++) // Make combination
 - // K1*cont + K2*collab
 - finalResults[loopvar] = ((contentWeight*contentResults[loopvar]) + (collabWeight*collabResults[loopvar]))

CalculateFinal2(int UserID)

// Second approach to calculating the final prediction based on weighted content & collaborative predictions

- int loopvar // Loop variable
- if netCollabError == -1 // Not enough cards to make valid collab prediction
 - contentWeight = 1
 - collabWeight = 0
- else
 - contentWeight = netCollabError / maxSKU // Base content off of collab
 - collabWeight = 1 - contentWeight
- for(loopvar = 1; loopvar <= maxSKU; loopvar++) // Make combination: $K1 \cdot \text{cont} + K2 \cdot \text{collab}$
 - finalResults[loopvar] = ((contentWeight*contentResults[loopvar]) + (collabWeight*collabResults[loopvar]))

MakeUserProfile(int UserID, Vector of Vector of ints for UserID's purchased cards)

// Create a new User Profile

- if UserID != ActiveUserID
 - BREAK w/ ERROR
- ProfileSys.SetActiveUser(int UserID, Vector of Vector of ints for UserID's purchased cards)
- ProfileSys.Calculate()
- UserProfile = ProfileSys.GetUserProfile()

ResetContentSystem()

- remove all data from contentResults[]
- remove all data from contentPlaces[]
- ContentSys.ResetSystem()

ResetCollabSystem()

- remove all data from collabResults[]
- remove all data from collabPlaces[]
- CollabSys.ResetSystem()

ResetProfileSystem()

- remove all data from vector UserProfile
- ProfileSys.ResetSystem()

ResetFinalPredictions()

- remove all data from finalResults[]

ResetAll()

// Re-initialize system so another user's predictions can be calculated

- ResetContentSystem()
- ResetCollabSystem()
- ResetProfileSystem()
- ResetFinalPredictions()
- ActiverUserID = -1

float[] GetContentPredictions()

float[] GetCollabPredictions()

float[] GetFinalPredictions()

Vector GetUserProfile()

Private Functions

MakeRank(float[] values, int[] rank, int[] SKUs)

// Create ordered list of card predictions

- int loopvar // Loop variable
- if((values.length != rank.length) || (rank.length != SKUs.length))
 - BREAK w/ ERROR
- sort(values, SKUs, 1, maxSKU) // Sort in DESCENDING order
- for(loopvar = 1; loopvar <= maxSKU; loopvar++) // Lowest value is first, which is worst
 - rank[SKUs[loopvar]] = loopvar; // prediction

Benchmark(int UserID)

// Benchmark the collaborative system by removing a card and seeing where it places

- float[] benchmarkData // Hold temp results
- int[] SKUs // Integer array of SKUs
- int cardRemoved // SKU of card removed
- int location // used to copy SKUs
- benchmarkData = new float[maxSKU + 1]
- SKUs = new int[maxSKU + 1]

- while(still more CardSKUs) // Copy SKUs from vector to array
 - location = CardSKUs.currentElementValue()
 - SKUs[location] = location
- cardRemoved = CollabSys.BenchmarkCalculation(UserID)
- if(cardRemoved == -1) // No cards left
 - netCollabError = -1
- else
 - benchmarkData = CollabSys.GetPredictions()
 - MakeRank(benchmarkData, collabPlaces, SKUs) // Rank predictions
 - netCollabError = collabPlaces[cardRemoved] - 1 // How well was card
 - // predicted (distance from 1st position)?

Collaborative System

Variables List

- Vector of Vector of ints containing SKUS of all items users purchased in order of UserIDs vector
 - Vector PurchaseList
- Vector of ints containing SKUS of items the active user purchased
 - Vector UserIDPurchaseList
- Vector UserIDs // All UserIDs
- bool[] activeUserPurchases // List active user purchases
- float[] predictions // Holds collab predictions for active user
- int maxSKU // Highest mapped SKU
- int numUsers // # users
- int ActiveUserID // Active user ID
- bool initOK // Setup done correctly?
- bool debug // Debugging variable

Public Functions

CollabSystem(bool test = FALSE)

// Constructor

- debug = test
- ActiveUserID = -1
- initOK = FALSE

SetActiveUserID(int UserID)

// Set active user ID and get their purchase information

- int location = 0 // Location in vector of User's purchase
- if initOK == FALSE
 - BREAK w/ ERROR
- if ActiveUserID != -1
 - BREAK w/ ERROR
- while(UserID != UserIDs.currentElementValue() && no more UserIDs left to check)
 - location++
 - UserIDs.nextElement()
- If ActiveUserID not found
 - BREAK w/ ERROR
- UserIDPurchaseList = Vector located at PurchaseList(location)
- While(still more elements in UserIDPurchaseList)
 - activeUserPurchases[UserIDPurchaseList.currentElementValue] = TRUE
 - UserIDPurchaseList.nextElement()

- ActiveUserID = UserID

Vector GetUserPurchaseList()

// Return active user's purchase list

- if initOK == FALSE
 - BREAK w/ ERROR
- if ActiveUserID == -1
 - BREAK w/ ERROR
- Return UserIDPurchaseList

float[] GetPredictions()

// Return collaborative prediction for all SKUs

- if initOK == FALSE
 - BREAK w/ ERROR
- if predictions == NULL
 - BREAK w/ ERROR
- else return predictions

SetupSystem(Vector of Vector of ints containing purchased CardSKUs, Vector InUserIDs, int inMaxSKU, int inNumUsers)

// Setup the CollaborativeSystem

- int loopvar // loop counter
- if CardSKUs = NULL
 - BREAK w/ ERROR
- if InUserIDs = NULL
 - BREAK w/ ERROR
- if InMaxSKU < 1
 - BREAK w/ ERROR
- if InNumUsers < 0
 - BREAK w/ ERROR
- PurchaseList = CardSKUs
- UserIDs = inUserIDs
- maxSKU = inMaxSKU
- numUsers = inNumUsers
- predictions = new float[maxSKU + 1]
 - check if memory allocated
- purchases = new bool[maxSKU + 1]
 - check if memory allocated
- ResetSystem()
- initOK = TRUE

Calculate()

// Calculate collaborative prediction for all SKUs for ActiveUser

- Vector userPurchases[maxSKU] // hold which users bought which card
- int user_user[numUsers] // hold # cards in common between active and other users
- int j,k // loop variables
- int local_id = 0 // Local UserID to compare with ActiveUser
- int curr_user_id; // Ensures that ActiveUser not compared with themselves
- float max_prediction = 0 // Highest prediction value for active user
- Vector CompareUserPurchases // Hold user to compare list of SKUs purchased
- if initOK == FALSE
 - BREAK w/ ERROR
- if ActiveUserID == -1
 - BREAK w/ ERROR
- for(j=0; j < numUsers; j++)
 - user_user[j] = 0
- while(still elements in PurchaseList) // Correlate people
 - curr_user_id = UserIDs.nextelement();
 - if curr_user_id != active_user_id
 - CompareUserPurchases = PurchaseList.currentElementValue()
 - while(still elements in CompareUserPurchases)
 - if(activeUserPurchases[CompareUserPurchases.ElementValue()] == TRUE)
 - user_user[local_id]++
 - userPurchases[CompareUserPurchases.ElementValue()].addElement(local_id)
 - compare_user_id++
- for(j=1; j <= maxSKU; j++) // Construct predictions
 - if(userPurchases[j] == NULL) // Cards already purchased OR cards not bought get value of 0
 - predictions[j] = 0
 - else
 - while(still elements in userPurchases[j])
 - predictions[j] += user_user[userPurchases[j].currentElementValue()]
 - if(predictions[j] > max_prediction)
 - max_prediction = predictions[j];
- for(j=0; j < num_skus; j++) // Normalize Ratings
 - predictions[j] /= max_prediction;

int BenchmarkCalculation(int UserID)

// Run a calculation to determine the accuracy of the Collaborative System by removing a card

// Returns the card removed to perform the test, but puts that card back in.

- RandomIntegerGenerator Randomizer // Random number generator
- int size // # cards user purchased
- int location // Location in vector of card to pull out
- int cardRemoved // SKU to pull out
- if initOK == false
 - BREAK w/ ERROR
- if ActiveUserID == -1
 - BREAK w/ ERROR
- Randomizer = new RandomIntegerGenerator()
- size = UserIDPurchaseList.size()
- location = Math.abs(Randomizer.nextInt()) % size
- if size <= 1 // Will be no cards, so collaborative can't be trusted
 - return -1
- cardRemoved = UserIDPurchaseList.elementAt(location)
- if activeUserPurchases[cardRemoved] == false // Something went wrong; card pulled
 - BREAK w/ ERROR // was not purchased
- activeUserPurchases[cardRemoved] = false // "pull out" card
- Calculate()
- activeUserPurchases[cardRemoved] = true // put it back in
- return cardRemoved

PrintSystem()

// Display all of the information in the CollaborativeSystem

- Vector ComparePurchases // All users purchases to display
- int j // Loop variable
- if initOK == FALSE
 - BREAK w/ ERROR
- Print("**** USER PURCHASE LIST ****\n")
- for(j=0; j < numUsers; j++)
 - print("Local user: %d \t", j)
 - print("Real userID: %d \t", UserIDs.currentElementValue())
 - ComparePurchases = PurchaseList.currentElementValue()
 - while(still elements in ComparePurchases)
 - print(" %d; ", ComparePurchases.currentElementValue())
 - print("\t")
 - if ActiveUserID == UserIDs.currentElementValue()
 - print("ACTIVE USER ")
 - print("\n")
 - UserIDs.nextElement()
 - PurchaseList.nextElement()
- Print("**** PREDICTIONS FOR ACTIVE USER ****\n")
- for(j=1; j <= maxSKU; j++)
 - print("SKU: %d\t", j)
 - print("Prediction: %f\n", predictions[j])

ResetSystem()

// Clean out system for next active user

- delete ActiveUserPurchases[]
- delete predictions[]
- for(loopvar = 1; loopvar <= maxSKU; loopvar++)
 - predictions[loopvar] = -1
 - activeUserPurchases[loopvar] = FALSE
- ActiveUserID = -1

Content System

Variables List

- Vector of Vector of ints containing Card Profiles
 - CardProfiles
- Vector of ints for ActiveUserProfile // Active user content profile
- Vector of floats for AttributeWeights // Attribute weights
- Vector of ints for AttributeTypes // Scalable or discrete attributes
- Vector ActiveUserWordRating // User card word rating
- float[] predictions // Final content prediction
- int ActiveUserID // Active user ID
- int numAttributes // # of attributes for a profile
- int numCards // # of cards
- bool initOK
- bool debug

Public Functions

ContentSystem(bool test = TRUE)

// Constructor

- debug = test
- ActiveUserID = -1
- initOK = false

SetActiveUser(int UserID, Vector InActiveUserProfile, Vector InWordRating)

// Sets the active user and their parameters.

- if initOK == FALSE
 - BREAK w/ ERROR
- if ActiveUserID != -1
 - BREAK w/ ERROR
- if InActiveUserProfile == NULL
 - BREAK w/ ERROR
- if InWordRating == NULL
 - BREAK w/ ERROR
- if InWordRating.numElements() != numCards
 - BREAK w/ ERROR
- if InActiveUserProfile.numElements() + 1 != numAttributes // Word rating sensitive
 - BREAK w/ ERROR
- ActiveUserProfile = InActiveUserProfile

- WordRating = InWordRating
- ActiveUserID = UserID

float[] GetPredictions()

// Returns the content-based predictions for all SKUs

- if initOK == FALSE
 - BREAK w/ ERROR
- if predictions == NULL
 - BREAK w/ ERROR
- Else return predictions

SetupSystem(Vector of Vectors of InCardProfiles, Vector InAttributeWeights, Vector InAttributeTypes)

// Initialize the system for use. Overwrites old data

- if InCardProfiles == NULL
 - BREAK w/ ERROR
- if AttributeWeights == NULL
 - BREAK w/ ERROR
- if AttributeTypes == NULL
 - BREAK w/ ERROR
- if(AttributeWeights.numElements() != AttributeTypes.numElements())
 - BREAK w/ ERROR
- if AttributeWeights.sumOfElements() != 1
 - BREAK w/ ERROR
- While(still more AttributeTypes)
 - If AttributeTypes.currentElementValue() == 0
 - BREAK w/ ERROR
- numCards = CardProfiles.numElements()
- numAttributes = AttributeWeights.numElements()
- CardProfiles = InCardProfiles
- while(still more CardProfiles)
 - if CardAttributes.currentElementValue().size() != numAttributes - 1 // take
// bag of words into account
 - BREAK w/ ERROR
- initOK = TRUE

Calculate()

// Calculate the content-based predictions

- float currentWordRating
- float result
- int loopvar = 0
- if initOK == FALSE

- BREAK w/ ERROR
- while(still more vectors in CardProfiles)
 - CurrentCardProfile = CardProfiles.currentElementValue()
 - currentWordRating = ActiveUserWordRating.currentElementValue()
 - result = DoTheMath(CurrentCardProfile, currentWordRating)
 - predictions[loopvar] = result
 - ActiveUserWordRating.nextElement()
 - loopvar++

PrintSystem()

// Print out everything in the content system

- Vector CardAttributes // Hold the attributes for a card
- int loopvar = 0 // Loop Variable
- if initOK == FALSE
 - BREAK w/ ERROR
- Print("*** CARDS & THEIR ATTRIBUTES \n***")
- While(still elements in CardProfiles)
 - CardAttributes = CardProfiles.currentElementValue()
 - While(still elements in CardAttributes)
 - Print("%d \", CardAttributes.currentElementValue())
 - Print("#\n")
- Print("\n*** ATTRIBUTE WEIGHTS ***")
- While(still elements in AttributeWeights)
 - Print("Attrib %d \t Value: %f\n", loopvar++, AttributeWeights.currentElementValue())
- loopvar = 0
- Print("\n*** ATTRIBUTE TYPES ***\n")
- While(still elements in AttributeTypes)
 - Print("Attrib %d: \t Type: %d", loopvar++, AttributeTypes.currentElementValue())
- Print("\n*** USER PROFILE ***")
- While(still elements in ActiveUserProfile)
 - Print("%d \", ActiveUserProfile.currentElementValue())
- Print("#\n")
- loopvar = 0
- Print("\n*** WORD RATINGS ***\n")
- While(still elements in WordRatings)
 - Print("SKU %d: \t Rating: %d", loopvar++, WordRating.currentElementValue())
- Print("\n**** PREDICTIONS FOR ACTIVE USER ****\n")
- for(loopvar=1; j <= numCards; loopvar++)
 - print("SKU: %d\t", loopvar)
 - print("Prediction: %f\n \t", prediction[loopvar])

ResetSystem()

// Clean out system for next active user

- delete ActiveUserProfile
- delete WordRating
- delete predictions[]
- ActiveUserID = -1

Private Functions

float DoTheMath(Vector CardProfile, float WordRating)

// Big bad boy of calculations

- float total = 0 // Rating total
- float percentage = 1 // % useable attributes
- int userValue // A user profile value
- float weight // Weight for attribute
- float sumCheck // Hold temp calculation
- int type // Attribute type
- int cardValue // A card profile value
- While(still more elements in AttributeWeights)
 - sumCheck = 0
 - weight = AttributeWeight.currentElementValue()
 - type = AttributeAVGType.currentElementValue()
 - if type == -1 // it's a word rating
 - sumCheck = WordRating
 - else
 - userValue = ActiveUserProfile.currentElementValue()
 - cardValue = CardProfile.currentElementValue()
 - if cardValue == 0 || userValue == 0 // N/A or 'none'
 - percentage = percentage - attribute.weight
 - else
 - if type < 0 // it's a discrete value
 - if cardValue == userValue // match
 - sumCheck = 1
 - else // it's a scaled value
 - sumCheck = 1 - absvalue((cardValue - userValue) / Max(cardValue, userValue))
 - ActiveUserProfile.nextElement()
 - CardProfile.nextElement()
 - total = total + (weight * sum_check)
 - AttributeAVGType.nextElement()
- if percentage == 0
 - return 0
- else return (total / percentage)

Profile Maker System

Variables List

- Vector of Vector of ints containing Cards Active User purchased
 - Vector ActiveUserPurchaseList
- Vector AttributeAVGTypes // How to take avg for attribute
- Vector UserProfile // User Profile to be constructed
- int numAttributes // # attributes
- int ActiveUserID // Active user ID
- int numCards // # active user purchases
- bool initOK // Check if Setup done correctly
- bool debug // Debugging variable

Public Functions

ProfileMaker(bool test = FALSE)

// Constructor

- debug = test
- ActiveUserID = -1
- initOK = FALSE

SetActiveUser(int UserID, Vector of Vectors of ints containing cards user purchased)

// Set Active User ID and copy over private variables

- Vector CardAttributes // Holds attributes for a card
- if initOK == FALSE
 - BREAK w/ ERROR
- if ActiveUserID != -1
 - BREAK w/ ERROR
- if CardsPurchahsed == NULL
 - BREAK w/ ERROR
- While(still elements in CardsPurchased) // Check that all have same # of attributes
 - CardAttributes = CardsPurchased.currentElementValue()
 - if CardAttributes.numElements() != numAttributes
 - BREAK w/ ERROR
- ActiveUserPurchaseList = CardsPurchased
- numCards = CardsPurchased.numElements()
- ActiveUserID = UserID

Vector GetUserProfile()

// Returns the constructed user profile

- if initOK == FALSE
 - BREAK w/ ERROR
- if UserProfile == NULL
 - BREAK w/ ERROR
- Return UserProfile

SetupSystem(Vector InAttributeAVGTypes)

// Prepare the system to create a user profile

- if InAttributeAVGTypes == NULL
 - BREAK w/ ERROR
- numAttributes = InAttributeAVGTypes.numElements()
- AttributeAVGTypes = InAttributeAVGTypes
- initOK = TRUE

Calculate()

// Calculate the active user's profile

- int total // Hold user profile value for an attribute
- int values[] // Attribute values
- int currentAttribute = 0 // CurrentAttribute to process
- int k // Loop variable
- UserProfile = new Vector(numAttributes)
- values = new int[numCards]
- if initOK == FALSE
 - BREAK w/ ERROR
- if ActiveUserID == -1
 - BREAK w/ ERROR
- while(still attributeAVGTypes left)
 - k = 0
 - while(still elements in ActiveUserPurchaseList)
 - values[k] = ActiveUserPurchaseList.currentElement().valueOf(currentAttribute)
 - k++
 - if(attribute.type == MEDIAN)
 - total = Median(values)
 - else if(attribute.type == MODE)
 - total = Mode(values)
 - else if(attribute.type == MEAN)
 - total = Mean(values)
 - else BREAK w/ ERROR

- UserProfile.addElement(total)
- currentAttribute++

PrintSystem()

// Display System on the screen

- Vector CardAttributes // Hold the attributes for a card
- int loopvar = 0 // Loop Variable
- if initOK == FALSE
 - BREAK w/ ERROR
- Print("*** CARDS USER PURCHASED & THEIR ATTRIBUTES \n***")
- While(still elements in ActiveUserPurchaseList)
 - CardAttributes = ActiveUserPurchaseList.currentElementValue()
 - While(still elements in CardAttributes)
 - Print("%d \ ", CardAttributes.currentElementValue())
 - Print("#\n")
- Print("\n*** ATTRIBUTE AVERAGE TYPES ***")
- While(still elements in AttributeAVGTypes)
 - Print("Attrib %d \t Type: %s\n", loopvar++, AttributeAVGTypes.currentElementValue())
- Print("\n*** FINAL USER PROFILE ***")
- While(still elements in UserProfile)
 - Print("%d \ ", UserProfile.currentElementValue())
- Print("#\n")

ResetSystem()

// Clean out Profile Maker for next user

- delete ActiveUserPurchaseList
- delete UserProfile
- ActiveUserID = -1

Private Functions

int Mean(int[] values)

int Mode(int[] values)

int Median(int[] values)

Bibliography

- Breese, John S., et. al., Empirical Analysis of Predictive Algorithms for Collaborative Filtering, Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, July, 1998.
- Claypool, Mark, et. al., Combining Content-Based and Collaborative Filters in an Online Newspaper. Computer Science Department, Worcester Polytechnic Institute, Worcester, MA, August 1999.
- Gokhale, Anuja, Claypool, Mark, Thresholds for More Accurate Collaborative Filtering. Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing. Aug. 1999.
- Herlocker, J., Konstan, J., Borchers, A., Riedl, J.. An Algorithmic Framework for Performing Collaborative Filtering. Proceedings of the 1999 Conference on Research and Development in Information Retrieval. Aug. 1999.
- Miller, Bradley N., et. al., Experiences with Grouplens: Making Usenet Useful Again. Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, June, 1996.
- Schafer, J. Ben, et. al., Recommender Systems in E-Commerce. Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN.
- Soboroff, I., et. al., Combining Content and Collaboration in Text Filtering. Workshop on Machine Learning for Information Filtering. 1999.
- http://www.sparks.com/help_faqs/about_overview.html. About Sparks.com, Sparks.com website. Attingo, Inc. 1999.